Владимирский государственный университет

Д. А. ЯКУБОВИЧ Е. С. ЕРОПОВА

ПРАКТИКУМ ПО ПРИКЛАДНОЙ ИНФОРМАТИКЕ: ВЕБ-РАЗРАБОТКА НА БАЗЕ HTML5, CSS3 И JAVASCRIPT

Владимир 2023

Министерство науки и высшего образования Российской Федерации
Федеральное государственное бюджетное образовательное учреждение
высшего образования
«Владимирский государственный университет
имени Александра Григорьевича и Николая Григорьевича Столетовых»

Д. А. ЯКУБОВИЧ Е. С. ЕРОПОВА

ПРАКТИКУМ ПО ПРИКЛАДНОЙ ИНФОРМАТИКЕ: ВЕБ-РАЗРАБОТКА НА БАЗЕ HTML5, CSS3 И JAVASCRIPT

Электронное издание

Владимир 2023

ISBN 978-5-9984-1866-2

УДК 004.738.5
ББК 32.973.4

Рецензенты:
Кандидат педагогических наук, доцент
заслуженный учитель РФ, почётный работник общего образования РФ,
почётный работник воспитания и просвещения
директор МАОУ г. Владимира «Промышленно-коммерческий лицей»
В. Е. Емельянов

Кандидат физико-математических наук, доцент
зам. директора по учебно-методической работе Педагогического института
Владимирского государственного университета
имени Александра Григорьевича и Николая Григорьевича Столетовых
В. А. Игонин

Якубович, Д. А. Практикум по прикладной информатике: веб-разработка на базе HTML5, CSS3 и JavaScript [Электронный ресурс] / Д. А. Якубович, Е. С. Еропова ; Владим. гос. ун-т им. А. Г. и Н. Г. Столетовых. – Владимир : Изд-во ВлГУ, 2023. – 951 с. – ISBN 978-5-9984-1866-2. – Электрон. дан. (49,3 Мб). – 1 электрон. опт. диск (CD-ROM). – Систем. требования: Intel от 1,3 ГГц ; Windows XP/7/8/10 ; Adobe Reader ; дисковод CD-ROM. – Загл. с титул. экрана.

Содержит 33 темы по обучению студентов основам разработки веб-сайтов на базе технологий HTML5, CSS3 и языка программирования JavaScript. Материал систематизирован и проиллюстрирован многочисленными примерами.

Предназначен для проведения лекционных и практических занятий со студентами педагогических вузов по дисциплинам «Прикладная информатика», «Информационные технологии в образовании», «Практикум по решению задач на ЭВМ». Может быть использован для организации самостоятельной работы студентов, чтения курсов повышения квалификации педагогических кадров и самообучения.

Рекомендовано для формирования профессиональных компетенций в соответствии с ФГОС ВО.

Ил. 740. Библиогр.: 22 назв.

ISBN 978-5-9984-1866-2

ОГЛАВЛЕНИЕ

ВВЕДЕНИЕ

В настоящее время прикладная информатика затрагивает обширный спектр информационных технологий и включена в учебный план подготовки студентов высших учебных заведений в качестве отдельной дисциплины. Среди изучаемых вопросов много внимания уделяется работе с веб-технологиями.

Высокий интерес к веб-технологиям связан с их богатыми возможностями и потенциалом, а также активным развитием. Современный веб позволяет эффективно решать различные прикладные задачи, не уступая при этом возможностям настольного программного обеспечения.

Особое место в веб занимают технологии HTML, CSS и язык программирования JavaScript, которые лежат в основе большинства веб-ресурсов сети Интернет. Однако зачастую подготовка студентов в рамках дисциплины ограничивается лишь поверхностным изложением основ этих технологий. Немаловажной проблемой является использование в обучении учебно-методического материала, который описывает устаревшие спецификации HTML, CSS и JavaScript, в то время как за последние 10 лет указанные технологии претерпели существенные изменения, поэтому многие студенты уже на начальном этапе изучения теряют интерес к вопросам верстки веб-страниц и веб-программированию.

Практикум подробно раскрывает особенности верстки веб-страниц на базе HTML5 и CSS3, а также основы программирования на языке JavaScript. Особое внимание в курсе уделено детальному разбору практических примеров.

Глава 1 детально описывает наиболее важные особенности архитектуры сети Интернет и технологии веб-разработки. Среди рассматриваемых технологий выделяется Frontend-разработка, в которой фундаментом являются HTML, CSS и JavaScript. Дается описание программного обеспечения, которое необходимо веб-разработчику. Материалы главы можно использовать для чтения курса лекций.

Глава 2 посвящена технологии гипертекстовой разметки HTML. Читатель изучит роль и возможности языка разметки HTML, познакомится с профессиональным текстовым редактором Visual Studio Code, научится использовать его возможности в верстке веб-страниц. Отдельное внимание уделено нововведениям спецификации HTML5.

Глава 3 описывает возможности и принципы работы технологии каскадных таблиц стилей CSS3. В главе раскрываются особенности разметки, стилизации и дизайна веб-страниц на основе комбинирования HTML и CSS, представлены фундаментальные принципы блочной верстки и организации адаптивного веб-дизайна.

Глава 4 излагает основы программирования на языке JavaScript, который позволяет динамически управлять структурой веб-страниц и поведением элементов пользовательского интерфейса. Высокая гибкость синтаксиса и активное развитие современного JavaScript в связке с HTML и CSS позволяют использовать язык не только в веб-разработке, но и в программировании обширного спектра практических задач.

Практикум нацелен на формирование фундаментальных компетенций в области веб-разработки. В каждой главе содержится необходимый теоретический материал, многочисленные и детально прокомментированные примеры решенных задач, иллюстрации. Для формирования практического опыта работы с представленными технологиями теоретический материал сопровождается практическими заданиями.

Практикум предназначен для подготовки студентов педагогических специальностей в рамках дисциплин «WEB-технологии», «Прикладная информатика», «ПРЗ на ЭВМ»; также материалы издания могут быть рекомендованы при подготовке курсовых, выпускных квалификационных работ и организации факультативных курсов по информатике для школьников.

ГЛАВА 1.
ТЕХНОЛОГИИ ВЕБ-РАЗРАБОТКИ

1.1. Элементы архитектуры и особенности функционирования сети Интернет

1.1.1. Понятие сети Интернет и веб

Сеть Интернет

Понятие

Определение

Сеть Интернет *– это информационно-коммуникационная сеть и всемирная система объединённых компьютерных сетей, обеспечивающая хранение и передачу информации.*

Интернет является фундаментальной технологией, позволяющей компьютерам, телефонам, серверам другим устройствам с сетевым подключением осуществлять передачу данных и взаимодействие.

Уровни сети Интернет

Работоспособность сети Интернет обеспечивается на двух уровнях: физическом и логическом.

Физический уровень определяется техническими средствами (аппаратная часть) и программными оболочками, которые отвечают за функционирование сети. В него включаются:

- кабели и средства спутниковой передачи данных;
- сервера и дата центры;
- особенности топологии сети Интернет;
- локальные и домашние сети, маршрутизаторы;

- программное обеспечение сети Интернет;
- сетевые службы (электронная почта, DNS, FTP и т.д.)

На *логическом* уровне обеспечивается взаимодействие пользователя с данными в сети или с другими пользователями. Среди логических средств выделим:

- интернет-сервисы (технология веб, электронная почта, системы телеконференций и онлайн трансляции, чаты);
- off-line сервисы (Email, Usenet, Mailing Listings) и on-line сервисы (FTP, DNS, Telnet);
- интернет ресурсы (веб-сайты, интернет-магазины, интернет-порталы, протоколы передачи данных, разные методы адресации, инструменты разработки и администрирования веб-сайтов);
- веб-технологии.

Особенности сети Интернет

- Обеспечивает передачу информации между компьютерами в единой сети.
- Реализует адресацию каждого компьютера внутри сети.
- Объединяет различные сети без иерархии, где все компьютеры равноправны.
- Обладает высокой надежностью (в Интернете нет единого центра управления).
- Интернет не принадлежит какой-либо коммерческой организации или правительству.
- Интернет – это сеть из множества сетей, которые, в свою очередь, состоят из более мелких подсетей.

Таким образом Интернет является распределенной системой.

Элементы сети

- *Провайдеры* – это организации, предоставляющие услуги доступа к сети Интернет.
- *Хребет Интернета* (*Backbon*) – представляет собой глобальную сеть высокоскоростных соединений между интернет-узлами крупнейших мировых телекоммуникационных компаний.

- *Узлы Интернета* – компьютеры, подключенные к сети. В качестве узлов также можно рассматривать и веб-сайты.
- *Адрес* – уникальное имя компьютера, используемое для его идентификации в сети.
- *Протокол* – это единые правила взаимодействия между разнотипными рабочими станциями, работающими под управлением разных операционных систем или оболочек.

Всемирная паутина

Понятие

Определение

Всемирная паутина (WWW, WEB, веб) *– это распределённая система, предоставляющая доступ к связанным между собой документам, расположенным на различных компьютерах, подключённых к сети Интернет.*

Термин «веб» буквально переводится как «паутина». Технология *World Wide Web* (*WWW*) обеспечивает интерактивную работу с ссылками, посредством которых пользователи могут перемещаться по разным страницам и веб-сайтам сети Интернет.

Родоначальником концепции всемирной паутины стал Тим Бернес-Ли, который предложил основные идеи ее структуры в 1989 году. Наряду с веб он разработал технологию гипертекстовой разметки страниц HTML, которая стала базовой для большинства современных веб-страниц. В 1991 году появляется первая программа-браузер, что в дальнейшем приводит к массовому распространению Всемирной паутины.

Структура и принципы работы компонент веб

В обобщённом виде веб содержит следующие компоненты:

- Веб-ресурсы в форме гипертекста – *веб-страницы*. Совокупность связанных веб-страниц, объединенных по смыслу или в иерархии каталогов называют *веб-сайтом*.
- Доступ к веб-ресурсам осуществляется посредством работы протоколов HTTP или HTTPS.
- Каждый ресурс имеет свой уникальный URL или URI адрес.

- Пользователь просматривает и взаимодействует с веб-ресурсами с помощью браузеров (часто также называют *клиентом*). Запрос пользователя (например, в форме ключевых слов для поисковика или данных из формы) отправляется на сервер, обрабатывается и высылается в виде документа или операции.

Разница между сетью Интернет и Всемирной паутиной

Зачастую пользователи отождествляют понятие Всемирной паутины WWW и сети Интернет, однако это некорректно.

Всемирная паутина включает в себя технологии, необходимые для работы с веб-документами: разметка и стилизация электронных страниц, скрипты, веб-сервера, протоколы HTTP и HTTPS, браузеры и веб-приложения.

Сеть Интернет организует взаимодействие между многочисленными компьютерными сетями. Она включает в себя каналы связи и аппаратуру, необходимую в том числе и для работы Всемирной паутины. Без доступа в сеть Интернет работа с WWW теряет смысл (однако электронные документы могут храниться на ПК пользователя и просматриваться с помощью браузера).

1.1.2. Архитектура и функционирование сети Интернет

Принципы работы сети Интернет

Выделяют три способа доступа к сети Интернет.

1. Прямое подключение (выделенное соединение)

Осуществляется через шлюзы – это подключение к магистральным каналам посредством выделенного компьютера (например, телефонной линии). Обычно используется большими корпорациями, которые обеспечивают служащим доступ в Интернет. Для такого подключения характерен большой объем трафика.

2. Соединение через чужой шлюз

Предоставляется частной организацией или образовательным учреждением. В настоящее время организации отказываются от этого подхода, поскольку может быть получен несанкционированный доступ к локальной сети организации.

3. Услуги сервис-провайдера

Организуют сеансовый доступ в сеть для компаний и частных лиц. Связь обеспечивается только на время подключения к Интернету. Качество соединения, объем трафика зависит от поставщика. Наиболее распространенный способ для большинства рядовых пользователей сети.

Взаимодействие компонентов сети

Взаимодействие в сети Интернет кратко можно описать следующим образом (рис. 1.1):

1. Провайдер обеспечивает доступ клиента к сети.
2. Передача данных осуществляется по сети кабелей и оптоволоконных соединений. Управление потоками данных осуществляют маршрутизаторы, выбирающие оптимальные маршруты информационных потоков.
3. Глобальная (а также региональная и обычно корпоративная) сеть включает подсеть связи, к которой подключаются локальные сети, отдельные компоненты и терминалы.
4. Инфраструктура сети делится на разные уровни:
    - магистральный уровень;
    - уровень сетей и точек доступа;
    - уровень региональных и других сетей;
    - интернет-провайдеры;
    - пользователи.

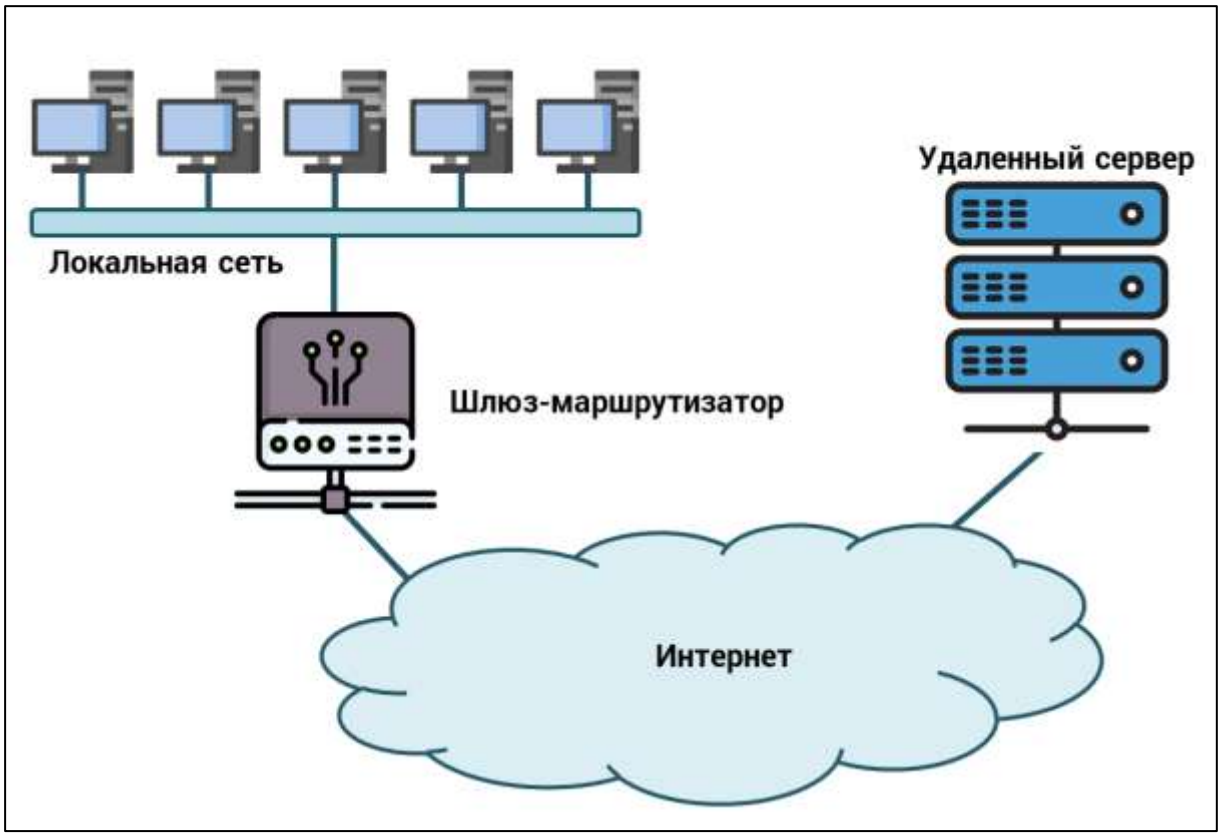


Рис. 1.1. Схема связи локальных сетей с удаленным сервером

Клиент-серверная модель

Понятие и принцип работы

Определение

Архитектура «Клиент-Сервер» *– модель организации взаимодействия в сети, предполагающая разделение процессов предоставления услуг и отправки запросов на разных компьютерах в сети, каждый из которых выполняют свои задачи независимо от других.*

Архитектура «клиент-сервер» может быть реализована не только в сетях с доступом в Интернет, но и локальных изолированных компьютерных сетях.

Общий принцип работы модели:

1. компьютеры-клиенты посылают запросы на сервер (хост-систему);
2. сервер обрабатывает запрос и отправляет ответ клиенту;
3. клиент формирует ответ пользователю в дружественном программном интерфейсе.

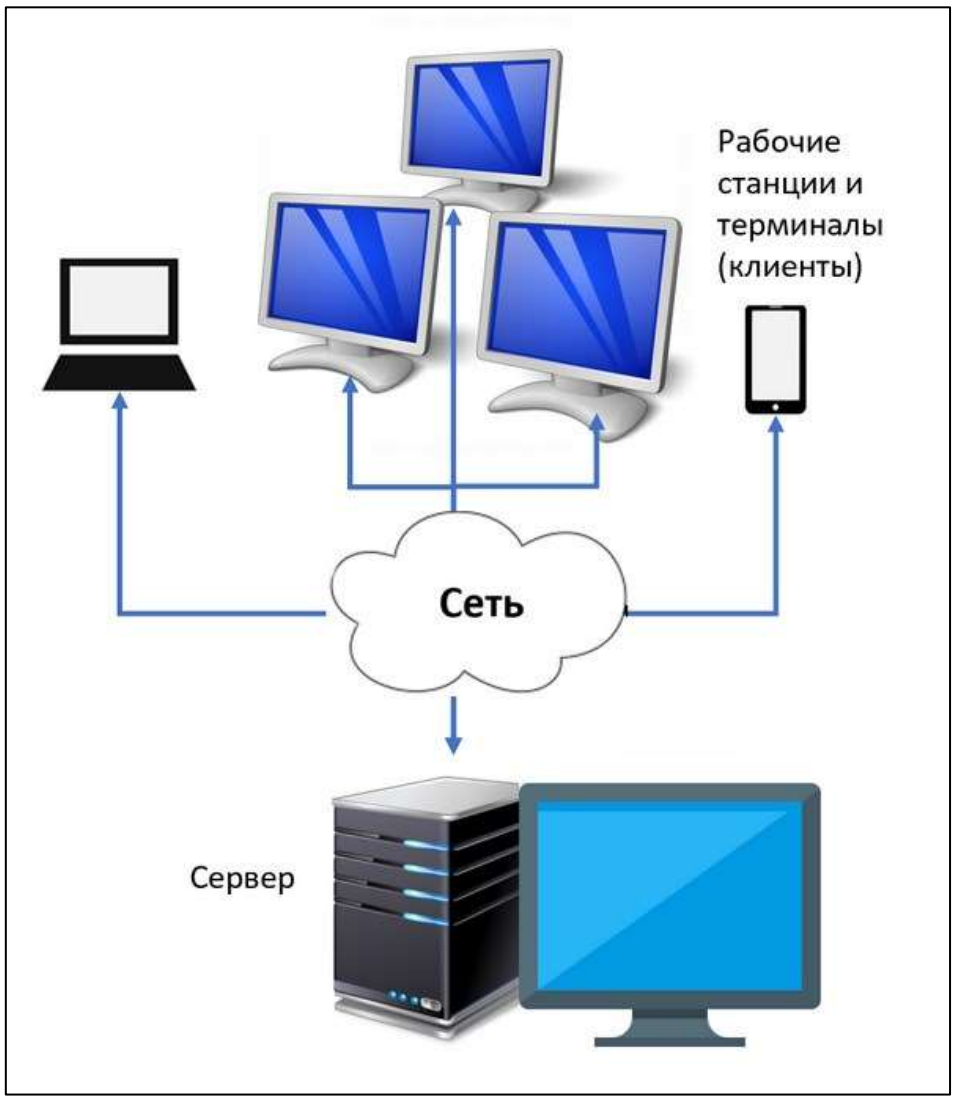


Рис. 1.2. Модель клиент-сервер

Архитектуру «клиент-сервер» принято разделять на разные уровни взаимодействия в зависимости от количества звеньев системы.

Одноуровневая архитектура (1-Tier)

В одноуровневой архитектуре прикладные программы рассредоточены по рабочим станциям (отдельным ПК), которые обращаются к общему серверу базы данных или файловому серверу. Задача сервера состоит лишь в предоставлении данных, а их обработка производится ресурсами рабочей станции.

Одноуровневая архитектура достаточно проста и надежна в реализации. С другой стороны, она может иметь сложности в управлении, поскольку каждая рабочая станция потенциально способна генерировать данные в разных форматах. Возникает проблема их синхронизации на отдельных машинах.

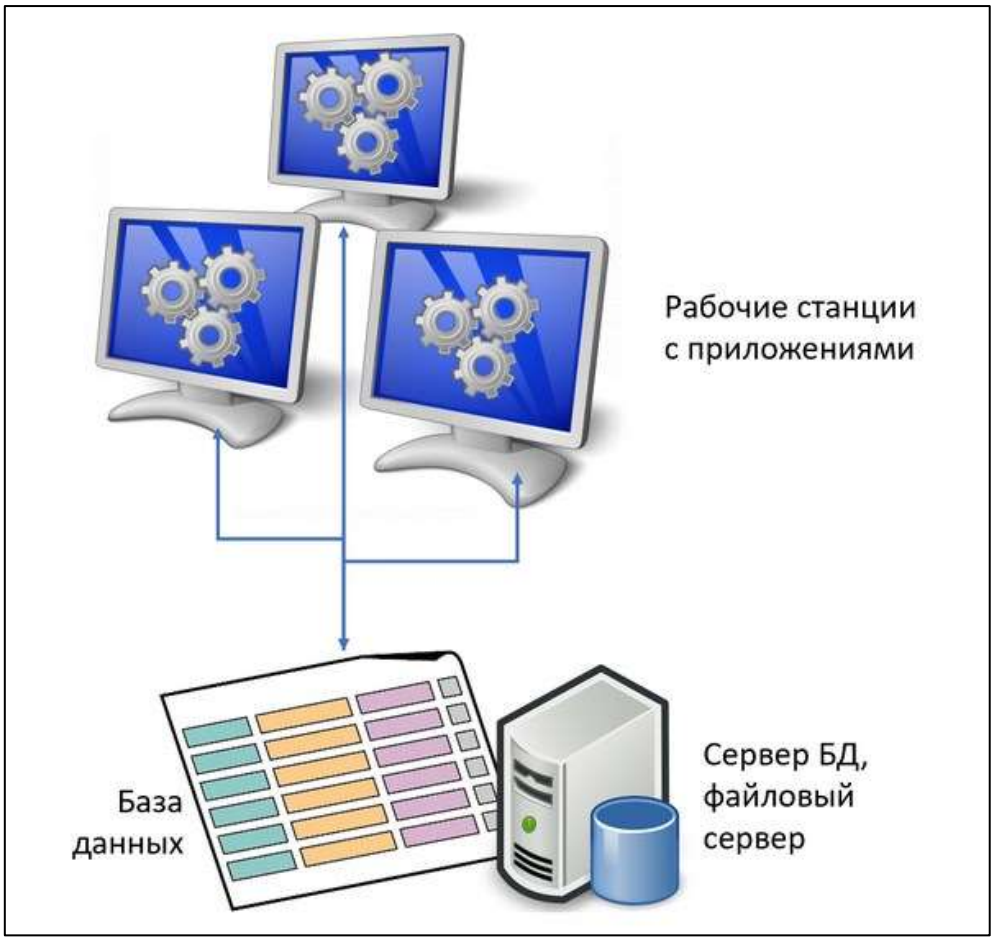


Рис. 1.3. Одноуровневая архитектура

Двухуровневая архитектура (2-Tier)

В двухуровневой архитектуре прикладные программы сосредоточены уже на сервере приложений (например, системы 1С, CRM). Рабочие станции содержат программы-клиенты, которые предоставляют пользователям интерфейс для работы.

Для этой архитектуры выделяю два варианта реализации.

1. *Fat client – thin server* («толстый клиент – тонкий сервер»). Данные хранятся на сервере, а обработка производится на машине клиента.
2. *Thin client – fat server* («тонкий клиент – толстый сервер»). Данные хранятся и обрабатываются на сервере по запросу клиента (пример – облачные технологии).

Двухуровневая архитектура проста в конфигурировании, удобна для пользователя и хорошо масштабируется. Однако она обладает и рядом важных недостатков:

- производительность и стабильность работы зависит от числа пользователей;
- данные хранятся на одном сервере, что потенциально может являться источником проблем в безопасности;
- зависимость от базы данных одного производителя.

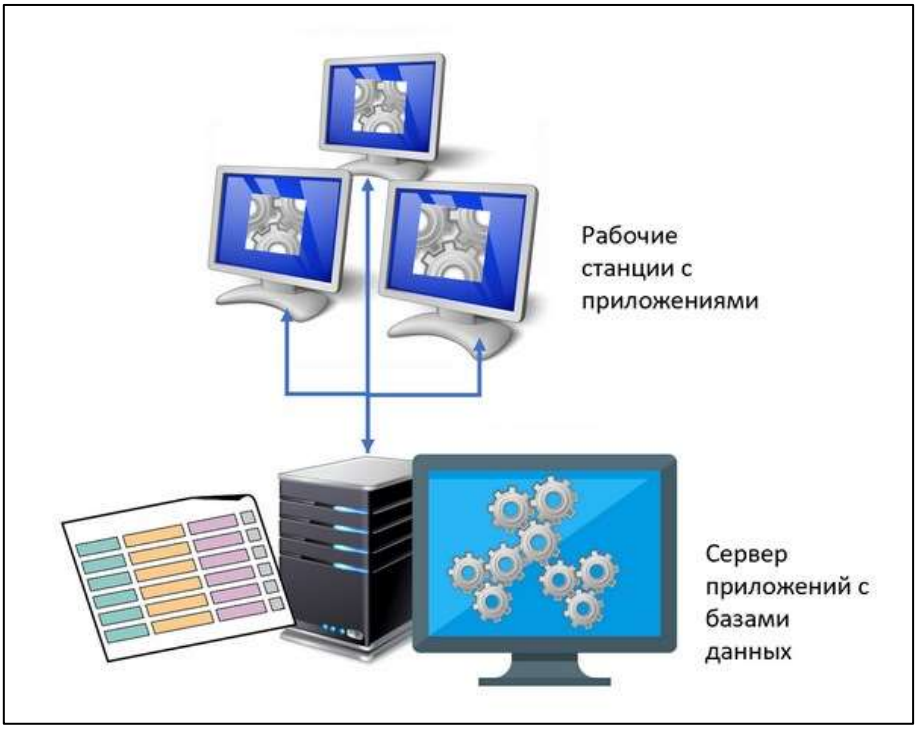


Рис. 1.4. Двухуровневая архитектура

Трехуровневая архитектура (3-Tier)

В модели трехуровневой архитектуры сервер разбивается на отдельные уровни: сервер базы данных и сервер приложений. Клиенты обращаются к базе данных не напрямую, а через промежуточное ПО на сервере приложений. Сервер приложений обрабатывает запросы и отправляет результаты клиенту.

Подобное разделение сервера существенно повышает безопасность и защищённость баз данных от несанкционированного доступа.

С другой стороны такая архитектура более сложна, поскольку требуется промежуточная коммуникация между клиентом и сервером. Не менее важным является и необходимость дополнительных затрат на реализацию и обслуживание подобной архитектуры.

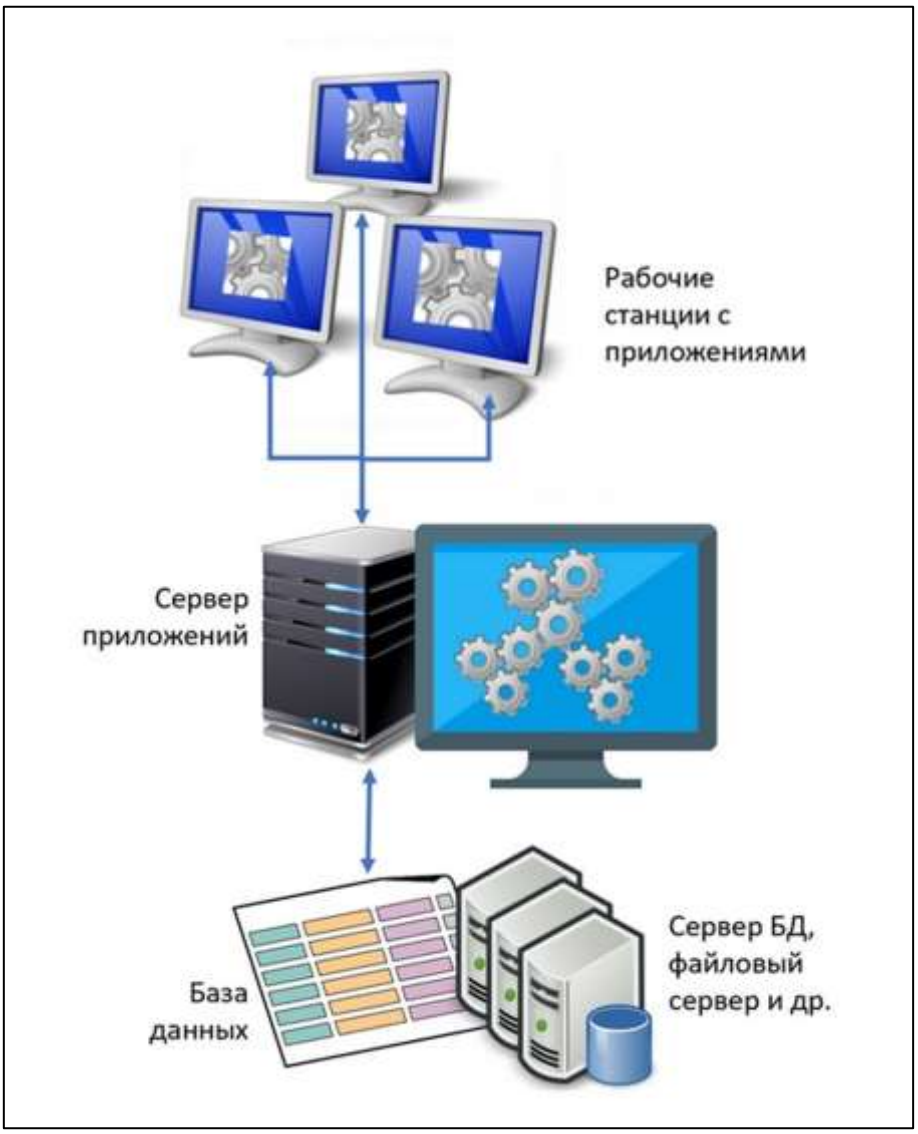


Рис. 1.5. Трехуровневая архитектура (3-Tier)

Многоуровневая архитектура (N-Tier)

Многоуровневая архитектура представляет собой развитие трехуровневой архитектуры, в которой используется синхронизированная работа нескольких серверов приложений.

Подобные системы обладают высокой гибкостью предоставления услуг пользователям. Они позволяют комбинировать работу различных серверов приложений на разных уровнях, организуя обработку больших объемов разнородных данных.

Многоуровневый подход эффективен при разработке корпоративных облачных систем. Грамотное разбиение сложных систем на несколько уровней упрощает обновление устаревших компонент.

Очевидным недостатком многоуровневой архитектуры сети является высокая сложность реализации и обслуживания, многокомпонентность, дороговизна оборудования.

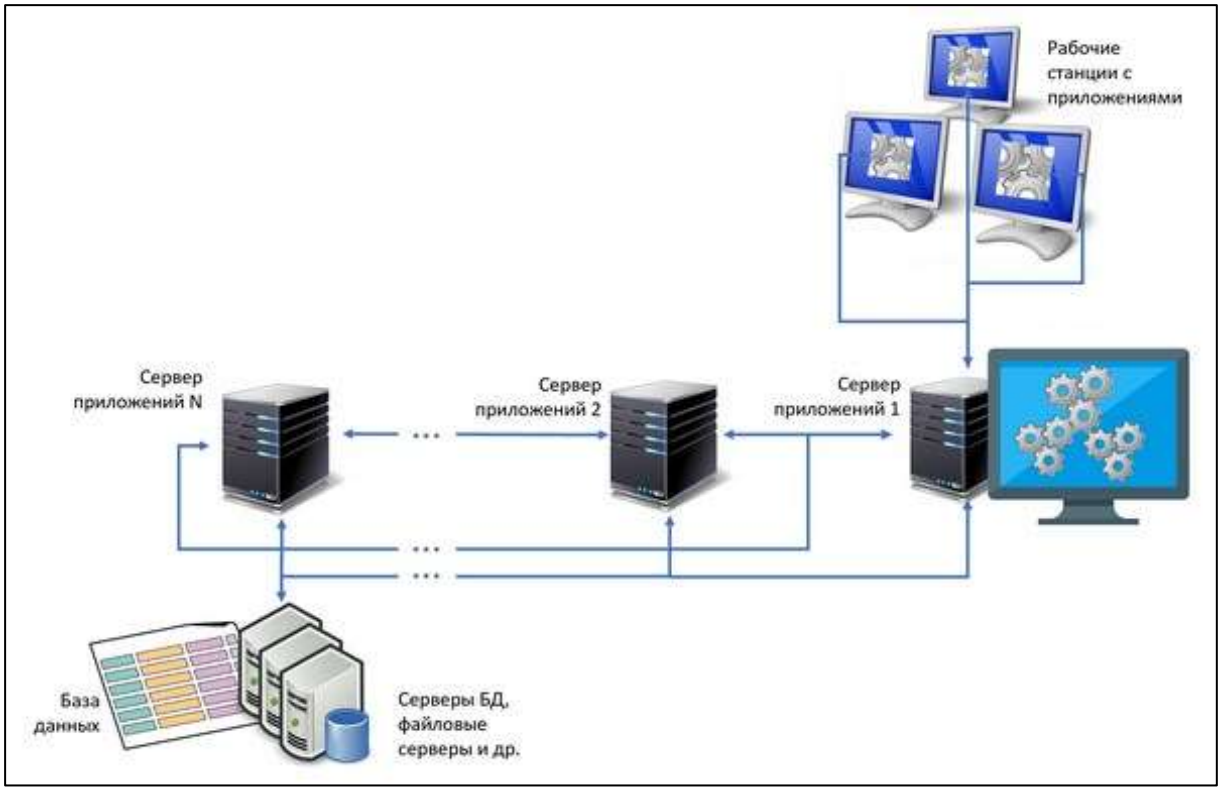


Рис. 1.6. Многоуровневая архитектура

Характеристики архитектуры «клиент-сервер»

1. *Асимметричность протоколов.* Поддерживается схема связи «один ко многим», а источником запросов обычно является клиент.
2. *Инкапсуляция услуг.* Клиент не видит процедуру обработки запроса, а только ответ на него с некоторой задержкой по времени.
3. *Целостность.* Управление программами и общими данными централизовано на сервере, что снижает стоимость обслуживания и защищает целостность данных. Данные клиентов остаются персонифицированными и независимыми.
4. *Местная прозрачность.* Программное обеспечение «клиент-сервер» обычно скрыто от клиентов и перенаправляет запрос на услуги через сеть.
5. *Обмен на основе сообщений.* Клиенты и сервер нежёстко связанные. Их задача – обмен запросами и ответами.
6. *Модульный дизайн, допускающий расширение*. Позволяет системе устойчиво работать при изменении числа клиентов, масштабировании или выходе из строя отдельных узлов.
7. *Независимость от платформы.* Хорошо оптимизированное приложение «клиент-сервер» не зависит от платформ и операционной системы.
8. *Масштабируемость.* Системы «клиент-сервер» можно масштабировать горизонтально (по количеству серверов и клиентов) и вертикально (по производительности и сложностям услуг).
9. *Разделение функционала.* Сервер предоставляет услуги обработки данных, а клиент является потребителем.
10. *Общее использование ресурсов.* Сервер способен предоставлять услуги нескольким клиентам одновременно и разграничивать их доступ к совместно используемым ресурсам.

Вопросы для самопроверки

1. Что представляет собой сеть Интернет и какими особенностями она обладает?
2. Охарактеризуйте роль основных элементов сети Интернет.
3. Что включается в понятие Всемирной паутины и чем она отличается от сети Интернет?
4. Перечислите основные способы подключения к сети Интернет, их достоинства и недостатки.
5. Опишите особенности построения архитектуры клиент-серверной модели и разные ее уровни.

Практикум

1. Проанализируйте возможности веб-ресурсов на примере редакторов презентаций. Подберите несколько примеров сайтов со свободным доступом. В качестве примера можно использовать возможности почтовых сервисов Yandex, Google, Mail.ru.
2. Сравните функционал веб-редакторов с возможностями настольных офисных приложений и редакторов: насколько он отличается?

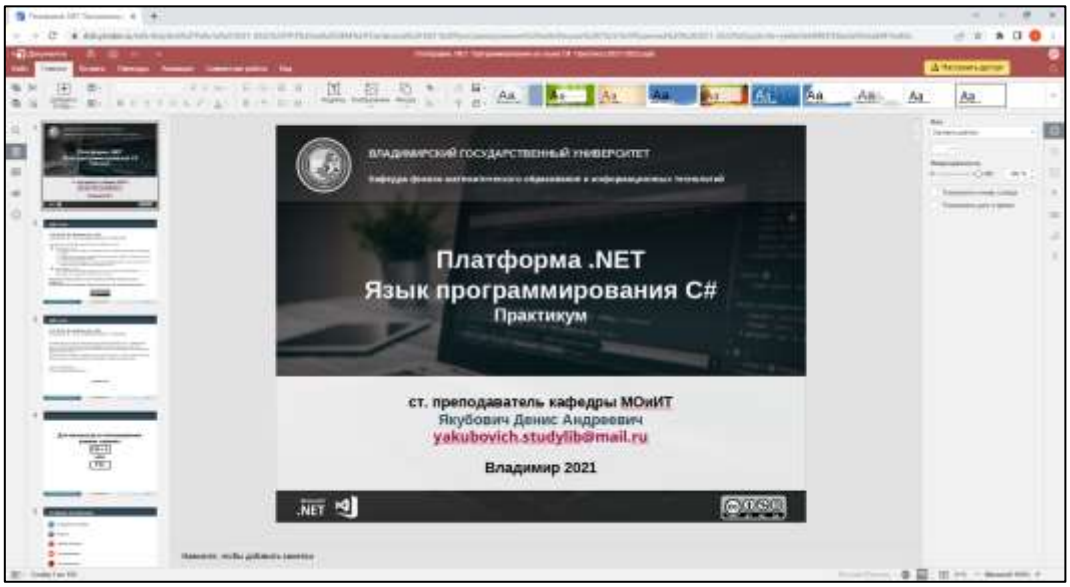


Рис. 1.7. Редактирование презентаций в онлайн-редакторе Yandex

1.2. Особенности адресации в сети Интернет

1.2.1. Протоколы

Определение

Сетевые протоколы *— это набор правил к ПО и требований к аппаратному обеспечению, определяющих принципы взаимодействия устройств в сети.*

Необходимость в протоколах исторически связана с распространением многочисленных производителей оборудования и различных принципов их работы. Протоколы позволяют универсализировать работу в сети, предписывая компаниям придерживаться единых правил совместимости производимого аппаратного и программного обеспечения.

В зависимости от назначения сетевые протоколы делятся на следующие категории:

- *Транспортные протоколы* (TCP/IP, UDP): контролируют передачу данных между межсетевыми протоколами и приложениями на уровне операционной системы.
- *Протоколы маршрутизации* (IP, ICMP, RIP, др.): используются маршрутизаторами (роутерами) для определения маршрутов передачи данных в составной вычислительной сети (интрасеть).
- *Протоколы поддержки сетевого адреса* (DNS, ARP, др.): отвечают за динамическое преобразование адресов.
- *Протоколы прикладных сервисов* (FTP, TELNET, HTTP, NNTP, др.): обеспечивают взаимодействие сети и пользователя.
- *Шлюзовые протоколы* (EGP, GGP, IGP): обеспечивают передачу данных в сети Интернет между сетями как автономными системами.
- Другие протоколы (SMTP, NFS).

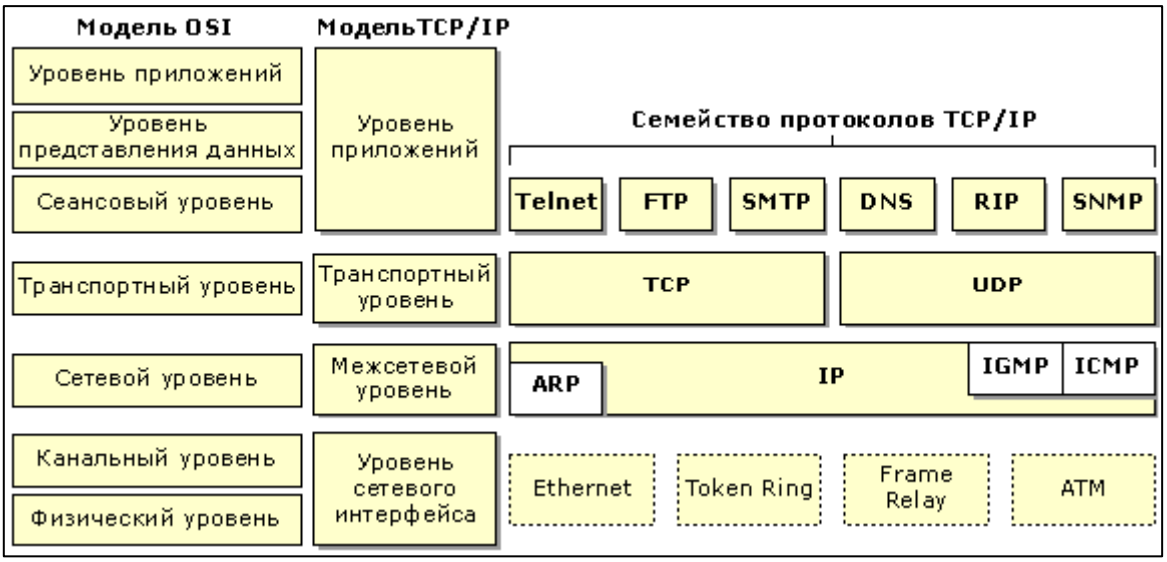


Рис. 1.8. Пример стека протоколов TCP/IP

1.2.2. Модель OSI

Понятие, назначение и принцип работы

Определение

Сетевая модель OSI *(The* ***O****pen* ***S****ystems* ***I****nterconnection model) – сетевая модель стека сетевых протоколов OSI/ISO, которые обеспечивают взаимодействие сетевых устройств.*

Модель OSI обобщает и стандартизирует сетевое взаимодействие независимо от их внутреннего устройства. Она состоит из набора протоколов, отвечающих за обмен данными внутри локальных сетей и сети Интернет.

Модель OSI задается семью слоями (*уровни* взаимодействия), каждый из которых отвечает за выполнение четко определенной функции сетевого взаимодействия. Уровни имеют свой набор протоколов. На нижнем уровне обработка данных ведется в битах и далее структура данных усложняется по мере перехода к верхнему уровню. Нижний слой оперирует понятиями «тип кабеля», «тип коннектора» и т.п., а верхний – протокол HTTP, или средства API.

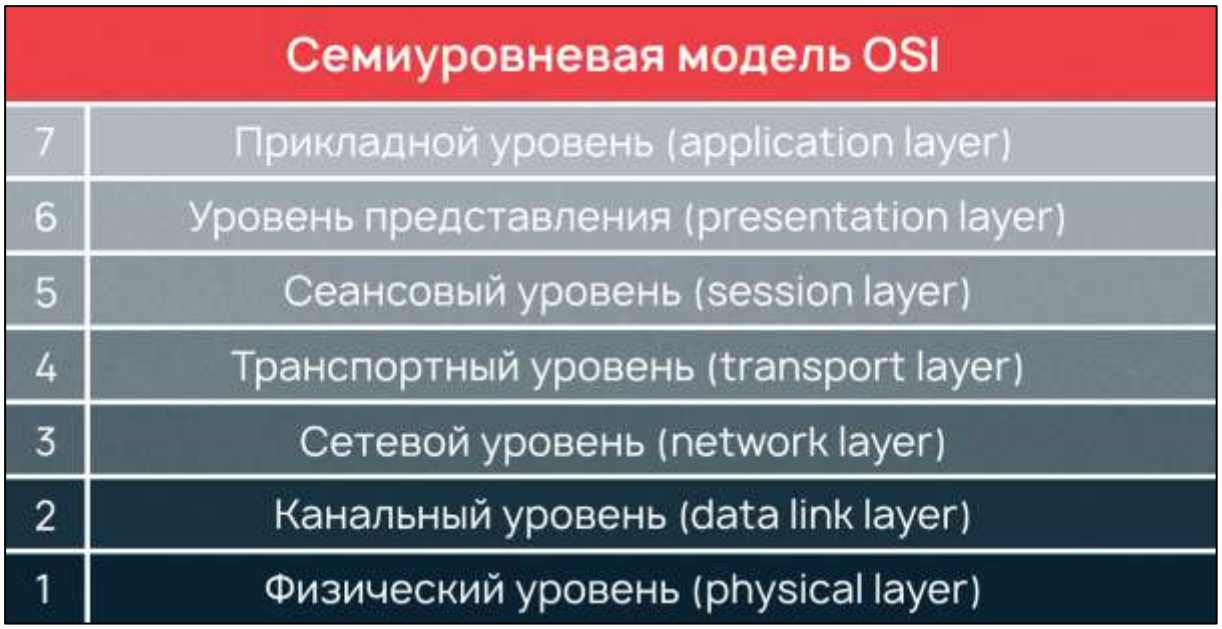


Рис. 1.9. Уровни протокола OSI

Процесс передачи данных согласно OSI содержит три обязательных компонента:

1. устройство-отправитель;
2. устройство-получатель;
3. пересылаемая информация (данные).

Процесс преобразования информации с нижнего на верхний уровень называется *декапсуляцией*, а в обратном направлении – *инкапсуляцией*. На каждом из семи уровней информация представляется в виде блоков данных протокола PDU (Protocol Data Unit), т.е. в понятной для данного уровня форме.

Все уровни модели OSI можно условно разграничить на два:

1. Уровни группы *Media Layers* (1, 2, 3) отвечают за передачу информации (по кабелю, беспроводной сети). При передаче информации задействуются коммутаторы, маршрутизаторы и т.п.
2. Уровни группы *Host Layers* (4, 5, 6, 7) используются непосредственно на устройствах.

Рассмотрим подробнее задачи каждого уровня OSI.

Уровни OSI

1. Физический уровень OSI

Самый нижний уровень OSI, где данные имеют физическую оболочку и передаются в виде тока или радиоволны по проводам или в форме сигналов. Для оцифровизации информации она кодируется в двоичную форму.

В качестве аппаратной части на этом уровне выступают концентраторы и репитеры. Их задача объединить несколько устройств в общий сегмент сети и передавать сигналы между устройствами без обработки.

Наиболее известными протоколами физического уровня являются Ethernet, Bluetooth, Wi-Fi и ИК-порт.

Рис. 1.10. Сетевой коммутатор

2. Канальный уровень OSI

Канальный уровень проверяет целостность полученных данных и исправляет ошибки, а также кодирует сигналы в биты.

Полученные и преобразованные в двоичную форму данные разбиваются на *фреймы* (*кадры*), состоящие из служебной информации (например, адрес отправителя, адреса получателя, передаваемые данные).

Канальный уровень содержит два подуровня:

1. *MAC* (*Media Access Control*) – уровень контроля доступа к среде. Присваивает физические уникальные MAC-адреса устройствам сети для взаимодействия с физическим уровнем (нижним).

2. *LLC* (*Logical Link Control*) – уровень управления логическим каналом. Необходим для взаимодействия с сетевым уровнем (верхним).

На канальном уровне часто используется протокол ARP, который сопоставляет 64-битные MAC-адреса с 32-битными IP-адресами. MAC-адрес задается производителем аппаратуры (материнской платы ПК) и фиксирован. Чтобы узнать его, например, в операционной системе Windows, можно воспользоваться командной строкой и ввести команду

```
ipconfig /all
```

или

```
getmac /v
```

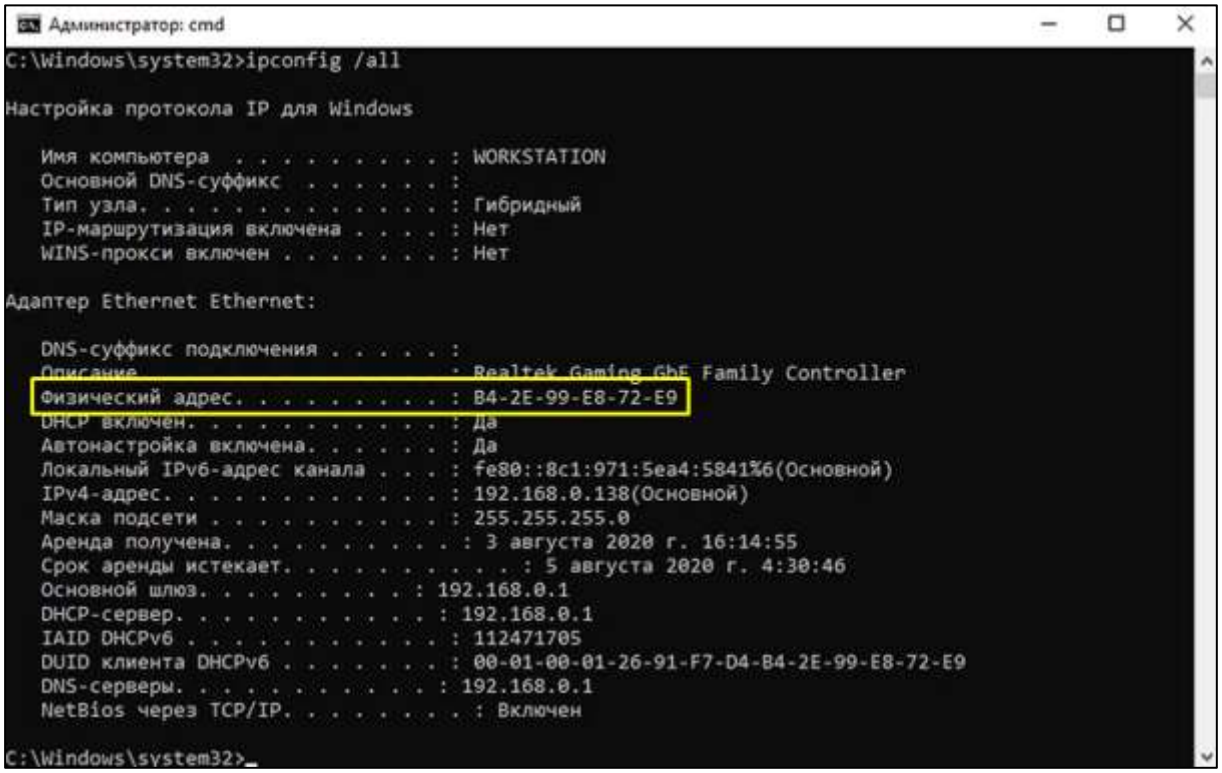


Рис. 1.11. Сетевой 64-битный MAC-адрес компьютера

3. Сетевой уровень OSI

На этом уровне осуществляется маршрутизация данных внутри сети (роутеры). Маршрутизаторы работают с MAC-адресом и строят маршрут от одного устройства к другому с учетом всех потенциальных неполадок в сети. В качестве аппаратуры здесь используются роутеры.

Данные на сетевом уровне описываются в виде *пакетов* и в некотором смысле похожи на фреймы канального уровня. Однако вместо MAC-адресов здесь используются уже IP-адреса, преобразованием которых занимается протокол ARP (см. выше).

Рис. 1.12. Маршрутизатор

4. Транспортный уровень OSI

Транспортный уровень является посредником между Host Layers и Media Layers. Его главная задача – транспортировка пакетов. На этом уровне используются протоколы:

- TCP/IP (гарантирует корректную доставку данных, например, передача логина и пароля);
- UDP (не проверяет целостность данных и допускает некоторые потери точности, подходит для передачи мультимедийных данных).

При передаче данных по протоколу TCP пакеты делятся на *сегменты*, что позволяет учитывать пропускную способность устройств и снижать риск ошибок.

В протоколе UPD данные делятся на *датаграммы* – особая форма пакета данных, обладающая автономностью. Они не зависят от сети и могут передаваться по разным маршрутам и в произвольном порядке.

Уровни 1-4 – зона ответственности сетевых инженеров, 5-7 – разработчиков.

Рис. 1.13. Уровни Host layers и Media layers

5. Сеансовый уровень OSI

Сеансовый уровень оперирует уже данными, имеющими определенный формат (расширение), поддерживает сеанс или сессию связи, управляет взаимодействием между приложениями, синхронизирует задачи и отвечает за обмен информацией.

Зачастую этот уровень применяются в средах приложений, требующих удаленного вызова процедур. Например, программы для видеотрансляций.

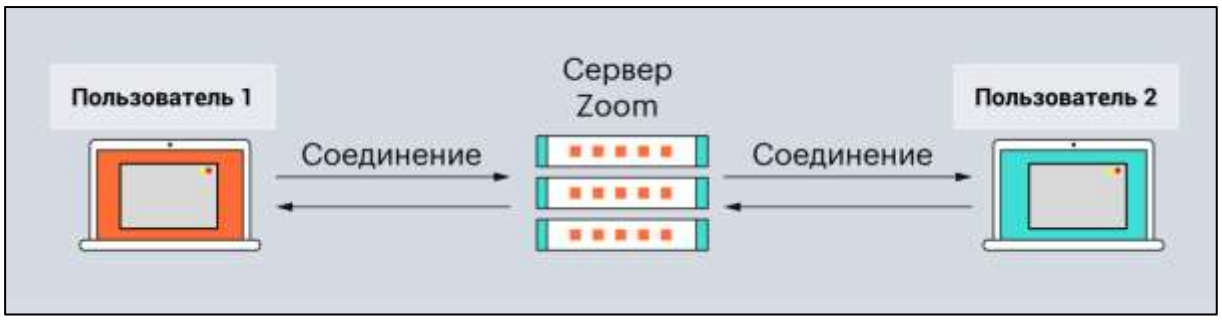


Рис. 1.14. Сеансовый уровень на примере онлайн трансляции в программе Zoom

6. Уровень представления OSI

Отвечает за преобразование форматов данных в привычный для человека и машине вид (например, перекодирование, сжатие, смена кодировки текста). Также на этом уровне данные при необходимости могут шифроваться.

7. Прикладной уровень OSI

Прикладной уровень представляет графический интерфейс, где пользователь взаимодействует с OSI по минимуму или неявно. До-

ступ к сетевым службам организуется, например, с помощью возможностей браузера.

Наиболее востребованные сетевые интерфейсы организованы работой протоколов HTTP, HTTPS, FTP и SMTP.

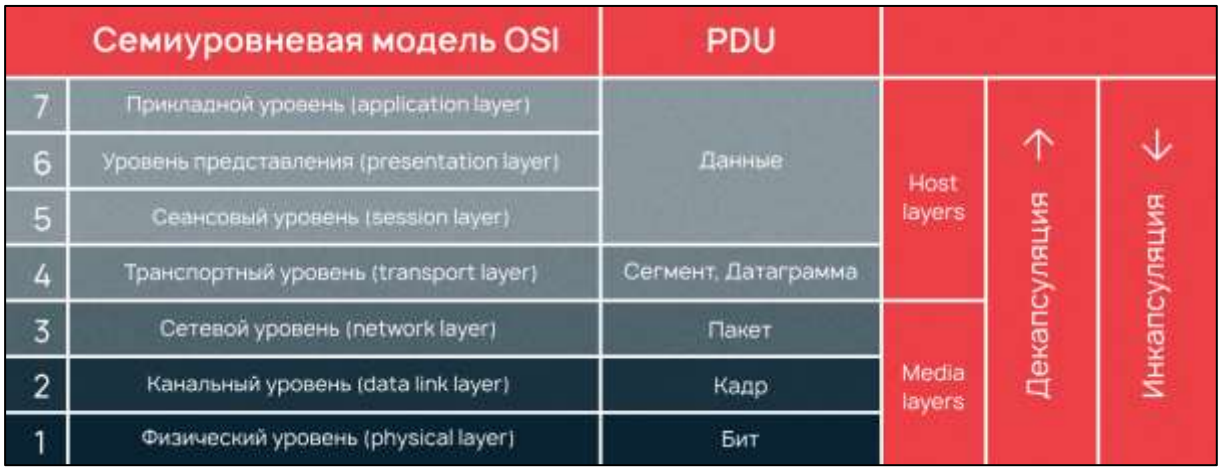


Рис. 1.15. Форма PDU представления данных на каждом уровне

1.2.3. MAC-адрес

Определение

MAC-адрес *(физический адрес) – это уникальный аппаратный номер сетевого оборудования (компьютера, сервера, роутера, порта коммутатора и т.п.) или сетевого порта.*

MAC-адрес присваивается любому устройству, оснащенному сетевой картой: стационарные ПК, ноутбуки, принтеры, USB-носители, маршрутизаторы, планшеты и смартфоны.

Некоторые устройства обладают двумя сетевыми картами и получают по два MAC-адреса. Например, ноутбуки имеют отдельные сетевые карты для работы технологий Ethernet и Wi-Fi.

MAC-адрес представляет собой уникальную комбинацию цифр и букв в шестнадцатеричной форме и длиной 48 символов (6 октетов). Первые 3 октета определяют организацию-изготовителя.

Рис. 1.16. Структура MAC-адреса и коды некоторых производителей сетевого оборудования

MAC-адрес обеспечивает второй (канальный) уровень модели OSI и отвечает за управление доступом к среде.

Как было отмечено ранее, узнать MAC-адрес своего ПК можно разными утилитами, в частности – использованием возможностей командной строки.

1.2.4. Протокол TCP/IP

Назначение протокола

Определение

TCP/IP *– набор протоколов, определяющий технологию межсетевого взаимодействия сети Интернет.*

TCP/IP необходим для обеспечения связи подключенных к сети устройств, которые часто называют *хостами*. Важным достоинством этих протоколов является возможность обеспечить аппаратную независимость в сетях с разным технологиями передачи данных и оборудованием.

Так как в протоколах Internet определяется только блок передачи и способ его отправки, TCP/IP не зависит от особенностей сетевого аппаратного обеспечения, позволяя организовать обмен информацией между сетями с различной технологией передачи данных.

TCP/IP позволяет:

- передавать файлы в другие системы;
- осуществлять вход в удаленную систему;

- выполнять команды удаленно;
- печатать файлы в удаленной системе;
- управлять электронными сообщениями;
- организовывать диалог с удаленными пользователями;
- управлять сетью.

Протокол TCP/IP основан на модели OSI и имеет 4 уровня: канальный (интерфейсный), межсетевой, транспортный и прикладной.

Структура TCP/IP

1. *Протокол IP* (Internet Protocol, межсетевой протокол) реализует распространение информации в IP-сети и выполняется на третьем (сетевом) уровне модели ISO/OSI. Основная задача – маршрутизация пакетов. Он не отвечает за надежность доставки информации, за ее целостность и за сохранение порядка потока пакетов.
2. *Высокоуровневый протокол TCP* (Transmission Control Protocol, протокол управления передачей). Это протокол с установлением логического соединения между отправителем и получателем. Обеспечивает связь с гарантированной доставкой информации.

Порты и сокеты

Концепция работы TCP/IP связана с понятием *сокета* – это специальная конструкция из IP-адреса и номера порта.

Рис. 1.17. Сокет и его структура

Для передачи данных прикладные процессы обращаются к транспортному уровню и передают данные в некоторой закодированной форме. Транспортному уровню не важна структура данных сообщения, ему достаточно знать IP адресанта и номер порта, через который осуществляется передача данных.

Порты в диапазоне 0-1023 зарезервированы под операционные системы, остальные номера в диапазоне 1024-49151 являются условно свободными и могут использоваться сторонними приложениями.

Номер IP является уникальным для каждого хоста, а номер сокета обычно зафиксирован для определенного типа приложений. Например, за получение электронной почты отвечает 110 порт, передача данных по протоколу FTP осуществляется по 21-му порту, открытие сайтов – по 80-му порту и т.д.

Общая схема адресации сокетов на примере работы с веб-сайтами выглядит следующим образом (см. рис. 1.18):

1. Клиент обращается к веб-ресурсу для передачи данных, используя его DNS-имя. Однако запрос идет не напрямую, поскольку на данном этапе IP-адрес ресурса еще неизвестен. В начале обращение идет к DNS-серверу, который находит связанный с DNS-именем IP-адрес необходимого веб-ресурса.
2. DNS-сервер возвращает IP-адрес клиенту.
3. Далее клиент уже по известному IP-адресу обращается к веб-ресурсу, используя фиксированный для данной операции порт.
4. Наконец, веб-ресурс обрабатывает запрос клиента и отправляет ему ответ.

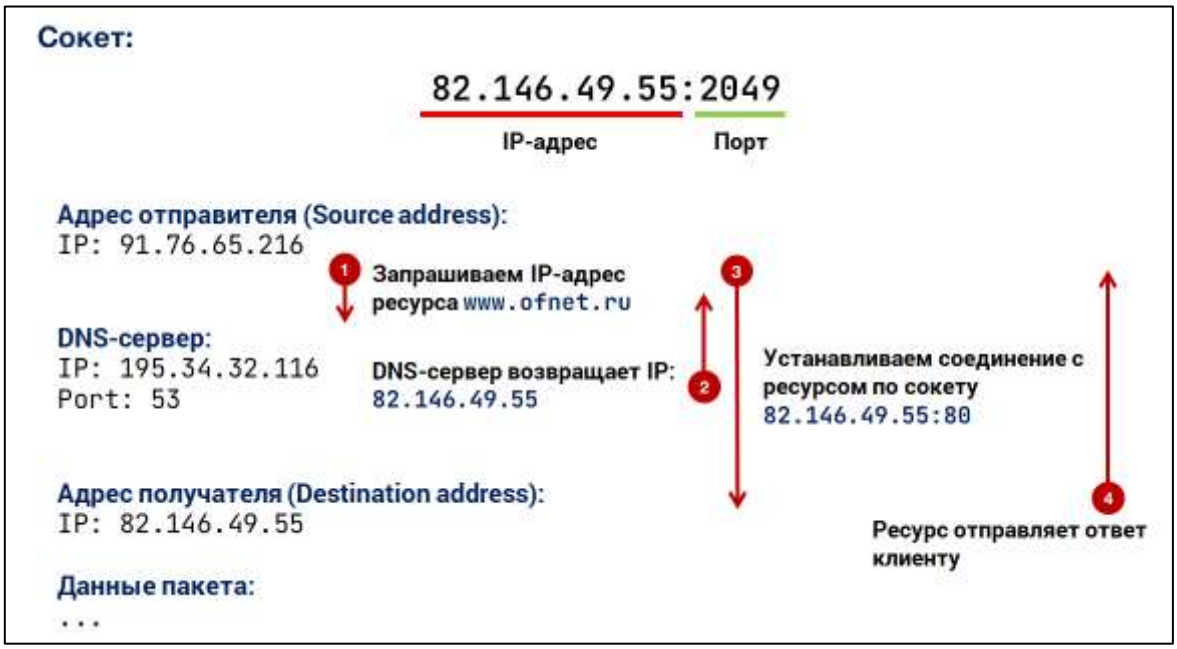


Рис. 1.18. Упрощенная схема адресации сокетов

Посмотреть список установленных соединений можно с помощью команды netstat:

```
netstat -a
netstat -an
```

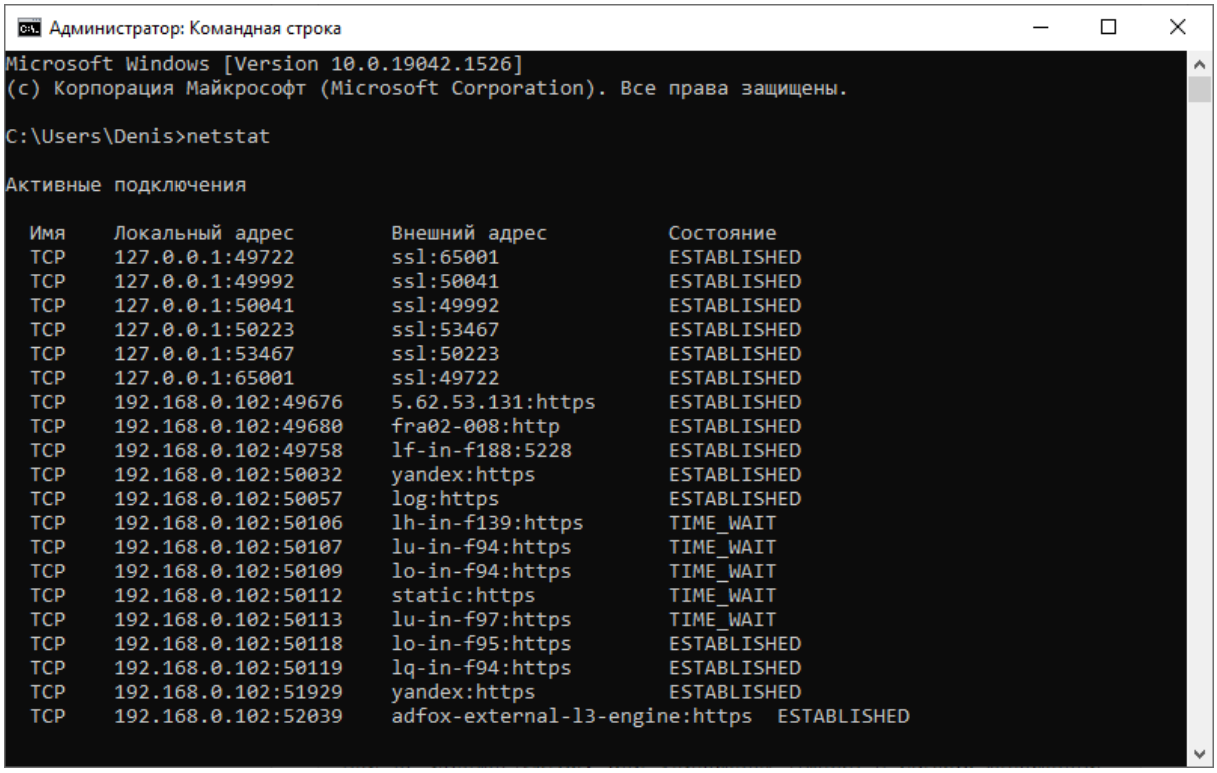


Рис. 1.19. Список установленных соединений с указанием портов

1.2.5. IP-адрес

Понятие и структура

Определение

IP-адрес *– это уникальный идентификатор (адрес) устройства (обычно компьютера), подключённого к локальной сети или сети Интернет.*

IP-адрес задается 32-битным двоичным числом, которое обычно записывается группами по 4 октета, каждый из которых может принимать значение от 0 до 255. Октеты отделяют точками.

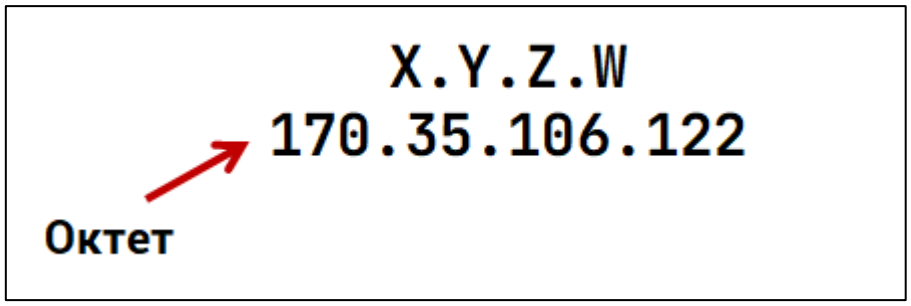


Рис. 1.20. Пример IP-адреса в спецификации IPv4

Подобный стандарт адреса называют IPv4.

В настоящее время IP-адреса являются основным типом адресации на сетевом уровне. Правила их работы подчинены одноименному протоколу IP, описывающему передачу пакетов данных между сетями. IP-адрес назначается администратором при установке конфигурации компьютеров и маршрутизаторов.

IP-адрес состоит из двух частей:

1. *Сеть* – это часть IP, которая не меняется во всей сети и все адреса устройств начинаются именно с номера сети.
2. *Узел* – это изменяющаяся часть IP. Каждое устройство имеет свой уникальный адрес в сети.

Согласно протоколу IP номер узла назначается независимо от локального адреса узла. Поскольку маршрутизатор входит сразу в несколько сетей, то каждый его порт получает собственный IP-адрес. Узел также может иметь доступ в несколько IP-сетей и каждому соединению назначается свой IP-адрес. Иными словами, IP-адрес характеризует не отдельный компьютер или маршрутизатор, а одно сетевое соединение.

Классовая адресация

Исторически IP создавался как классовый протокол, предполагающий возможность делить адресное пространство на классы, каждый из которых будет ориентирован на потребности конкретной сети клиента.

Концепция классификации в IP предполагает ограничение предельного размера системы на основании соотношения количества сетей и узлов. Например, один класс может выделять 127 адресуемых сетей и более 16 миллионов адресуемых клиентов, а другой – около 2 миллионов сетей по 254 клиентов в каждой.

Протокол IP обычно делят на три класса, которые в целом позволяют покрыть потребности как больших, так и малых организаций.

Рис. 1.21. Общепринятое деление IP на классы

Однако на самом деле выделяют 5 классов: классы D и E используются в специальных сетевых задачах, например – для многоадресной передачи.

Рис. 1.22. Полное деление на классы IP

Рассмотрим следующий пример. Пусть адрес сети определен как 172.16.0.0.

Тогда:

- 172.16.0.1 – адрес первого устройства в сети;
- 172.16.255.254 – адрес последнего устройства в сети;
- 172.16.255.255 – широковещательный IP адрес;
- 172.17.0.0 – адрес следующей сети;
- всего 65534 устройства в сети.

Чтобы определить, какая часть IP-адреса относится к номеру сети, а какая – узла, используется битовая *маска подсети*: число, позволяющее однозначно выделить из IP номер подсети и узла.

Например, имеется узел 12.34.56.78. Если задана маска 255.255.255.0 (/24), то сеть имеет адрес: 12.34.56.0 (первые 24 бита или 3 октета – маска). Тогда IP узлов этой подсети будет выделено в диапазоне 12.34.56.1 - 12.34.56.254.

Бесклассовая адресация

Основная проблема концепции классового протокола IP заключается в предельном количестве адресов. В силу стремительного роста количества подсетей и клиентов сети Интернет острее ставится вопрос о расширении адресного пространства.

Начиная со второй половины 90-х годов классовая маршрутизация стала вытесняться бесклассовой маршрутизацией, в которой количество адресов в сети определяется только маской подсети. Главная цель – экономно использовать ограниченный ресурс IP-адресов.

Например, задан диапазон 10.96.0.0/11. Маска сети: 255.224.0.0.

Тогда диапазон адресов: 10.96.0.0 - 10.127.255.255.

Одной из популярных бесклассовых современных моделей является протокол CIDR.

IPv4 и IPv6

Описанные выше правила протокола IP относятся к наиболее распространённому на данный момент стандарту IPv4, который использует 32-битную адресацию, потенциально ограничивающую допустимое пространство до 4.3 млрд. адресов.

В 1999 году был внедрен протокол IPv6, который предполагает уже 128-битную адресацию (т.е. 2^{128} адресов). Целью расширения адресного пространства являлось устранение ограничений протокола IPv4 и решение связанных с ним технических проблем совместимости устройств в сети.

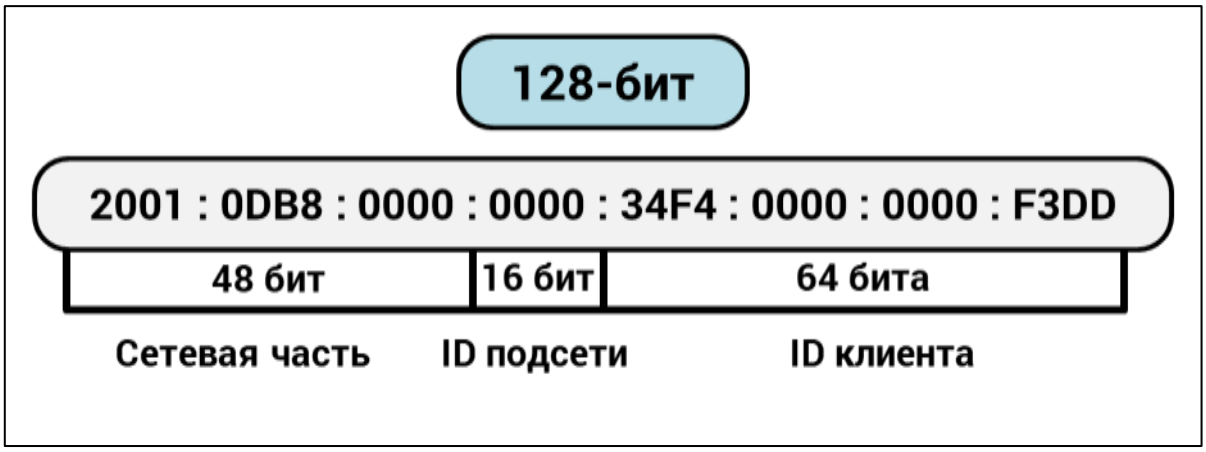


Рис. 1.23. Пример IP-адреса в протоколе версии IPv6

Среди основных достоинств перехода с IPv4 на IPv6 является:

- оптимизация и упрощение маршрутизации;
- улучшение совместимости с мобильными сетями;
- возможность масштабирования сетей;
- исключены конфликты частных IP-адресов;
- повышение уровня безопасности.

Однако даже по прошествии более чем 20 лет IPv6-адресация занимает лишь около 2% мирового трафика в отличие от IPv4. Это связано, в частности, с необходимостью перехода компаний производителей сетевых плат на новый стандарт адресации. С другой стороны частично ограничения IPv4 позволила снять бесклассовая адресация CIDR.

Внешний и внутренний IP адрес

Устройству в сети может быть одновременно назначено два сетевых IP-адреса – *внутренний* и *внешний*.

Каждое устройство внутри домашней или локальной сети провайдера получает уникальный внутренний IP-адрес. Он не может повторяться у других устройств этой сети, однако может совпадать с адресами другой локальной сети. Например, устройства в соседних квартирах могут иметь одинаковые внутренние IP-адреса, поскольку жильцы дома пользуются услугами разных провайдеров.

Выход в глобальную сеть требует использования внешних IP-адресов, которые должны быть только уникальными. В частности, провайдер назначает внешний IP-адрес роутеру (маршрутизатору) абонента. Роутер организует домашнюю сеть, где устройства получа-

ют локальные IP-адреса (обычно в диапазоне 192.168.0.0 – 192.168.255.255).

Получить информацию о внешнем IP-адресе можно в настройках роутера, специальных утилитах или веб-сайтах.

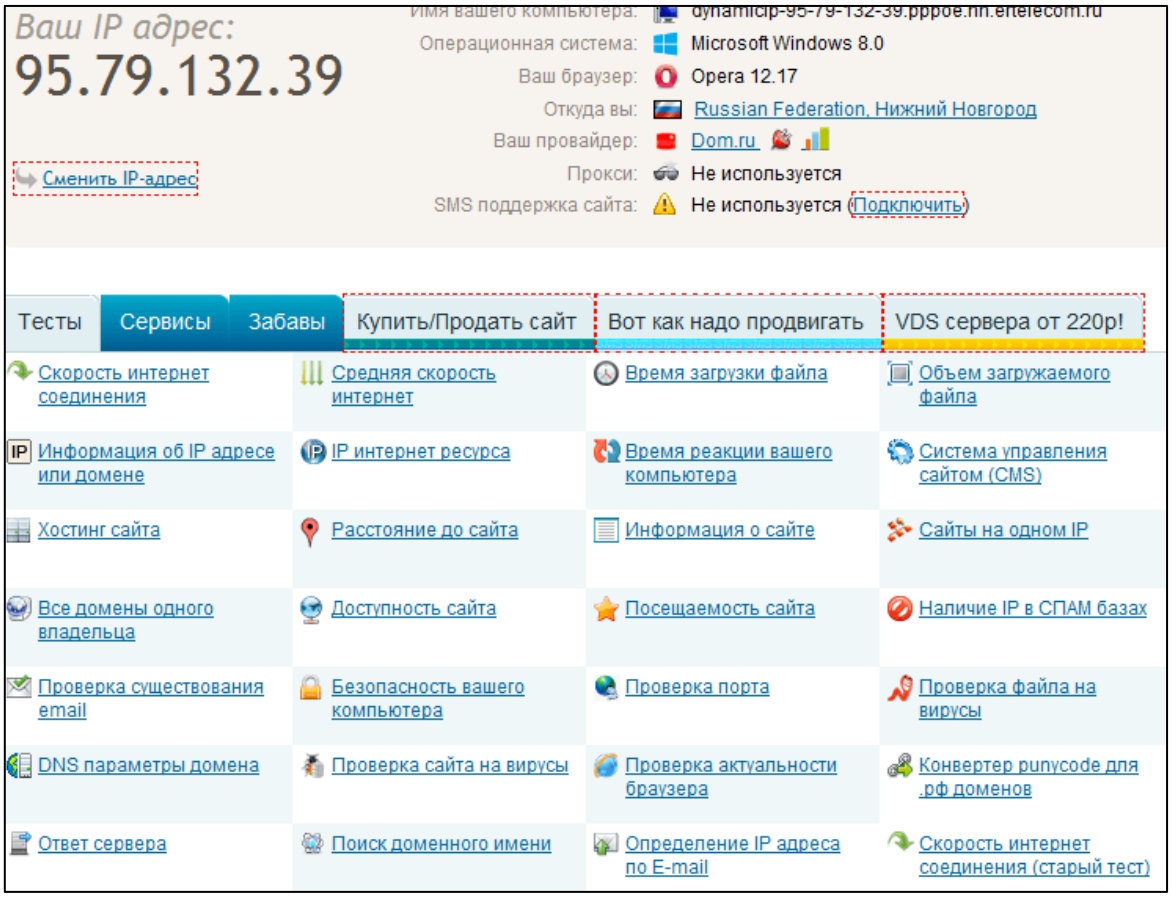


Рис. 1.24. Информация об Интернет-провайдере и IP-адресе на сайте 2ip.ru

Статические и динамические IP-адреса

Внешние IP-адреса, назначаемые провайдером сети Интернет, могут быть статическими и динамическими.

Статический IP-адрес остается неизменным в течение длительного времени. Обычно статический IP назначается веб-сайтам, что гарантирует пользователям доступ к серверу.

Динамический IP-адрес назначается пользователю провайдером при каждом подключении к сети. Работа с динамическими IP-адресами характерна для больших сетей, где количество доступных адресов ограничено.

Динамические IP-адреса считаются более безопасными для массового пользователя, поскольку усложняют несанкционированный

доступ к информации на ПК. Они также часто полезны в организации SMM-рассылок, обходе блокировок и «банов», работе с файлообменниками.

Статические IP-адреса используются в системах авторизации интернет-банков, которые предоставляют услуги привязки учетной записи к IP. На основе статической IP-адресации организуется удаленное подключение к ПК (например, программами TeamViewer или Remote Administrator), настраиваются локальные сервера.

Также стоит учитывать, что обычно пользователи получают динамический IP-адрес. Интернет-провайдеры могут взимать дополнительную плату за статический IP-адрес, либо не предоставлять услуг по его использованию.

1.2.6. DNS-сервер

Доменные имена

Доменные имена как текстовый адрес

Определение

DNS *(Domain Name System, доменное имя хоста) – уникальное символьное имя адреса ресурса в удобной для человека форме, скрывающее IP адрес.*

Технология DNS является одной из важнейших в обеспечении адресации в сети Интернет. Основная ее задача состоит в соотнесении IP-адресов устройств в сети с текстовым доменным именем ресурса. Это позволяет упростить пользователям обращение к веб-сайтам: вместо числового IP-адреса (который может быть к тому же и динамическим) пользователь обращается к ресурсу по его текстовому имени.

DNS-адреса имеют древовидную структуру, называемую *пространством имен домена*. Каждый узел имеет уникальное имя. Верхним уровнем (*корнем*) домена является символ точки. От корневого элемента ответвляются поддоменые зоны или узлы (компьютеры ил другие устройства в сети).

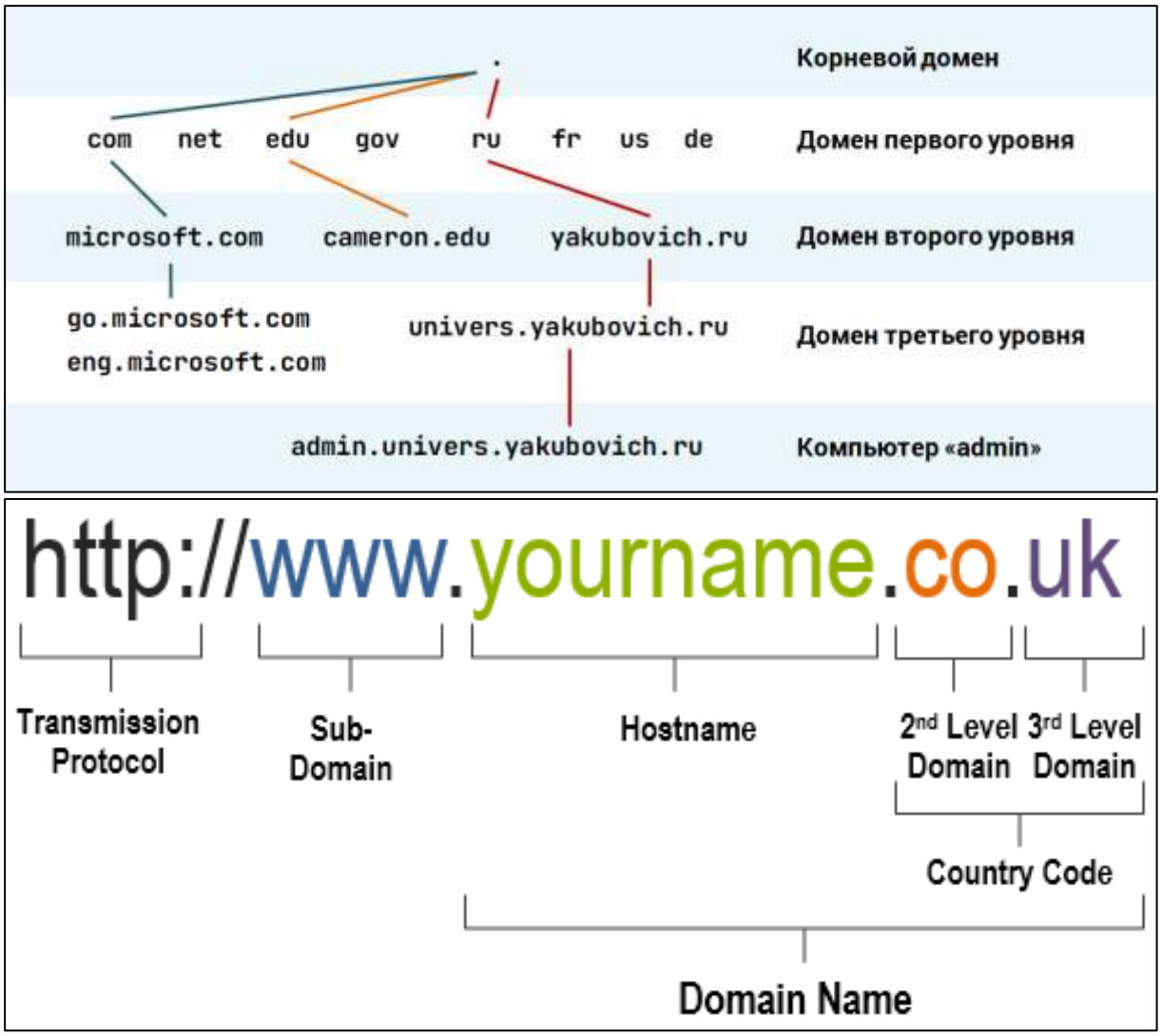


Рис. 1.25. Примеры древовидной структуры доменных имен и их уровней

Плоская и иерархическая адресации

Существует два подхода к формированию пространства DNS-адресов.

Плоская DNS-адресация является последовательностью символов, которые не отражают структуры и общим правил формирования доменного имени. Подобный подход не может использоваться в больших системах в силу и возможной неоднозначности при дублировании имен разных узлов.

Иерархическая DNS-адресация позволяет разбивать имя домена на несколько частей, например:

- домен первого уровня .ru;
- домена второго уровня example.ru,
- домена третьего уровня api.example.ru и т. д.

Именно иерархический подход используется в сети Интернет как основной.

Принцип работы DNS-сервера

Назначение

Определение

DNS-сервер *– это специальный компьютер, хранящий (кэширующий) IP-адреса веб-сайтов.*

DNS-сервер обслуживает доменные зоны, которые ему делегированы и хранит данные о ресурсных записях этих зон. Если сервер получает запрос о другом домене, то он обращается к другим DNS-серверам.

DNS-сервера не хранят таблицы соответствий IP-адресов и доменов, а временно кешируют данные для часто используемых доменов. Когда в адресной строке браузера пользователь вводит домен сайта, сервер автоматически извлекает его IP-адрес и отправляет по нему запрос. Этот процесс также называется «*разрешение адреса домена*».

В настоящее время во всем мире используется 13 корневых DNS-серверов, которые хранят данные об обслуживании зон верхнего уровня.

Схема работы

1. Предположим, пользователь обращается в адресной строке браузера к веб-ресурсу с доменным именем api.example.ru. Браузер отправляет запрос DNS-серверу сети. Если доменное имя находится в зоне ответственности сервера, он отвечает сам, иначе пересылает запрос одному из высокоуровневых доменных серверов или корневому.
2. Далее запрос начинает поэтапное перемещение по разным уровням. В начале корневой сервер пересылает запрос серверу первого уровня (зона .ru). Сервер первого уровня обращается ко второму (example.ru) и так далее, пока не будет найден сервер, который хранит точный адрес ресурса (либо ответит, что такое имя не существует).

3. После того, как адрес ресурса найден, начинается обратное движение.
4. Также в ряде случаев требуется процедура обратного сопоставления – извлечение DNS по известному IP (требуется, например, в работе сервера электронной почты). Для этого используется специальный домен in-addr.arpa.

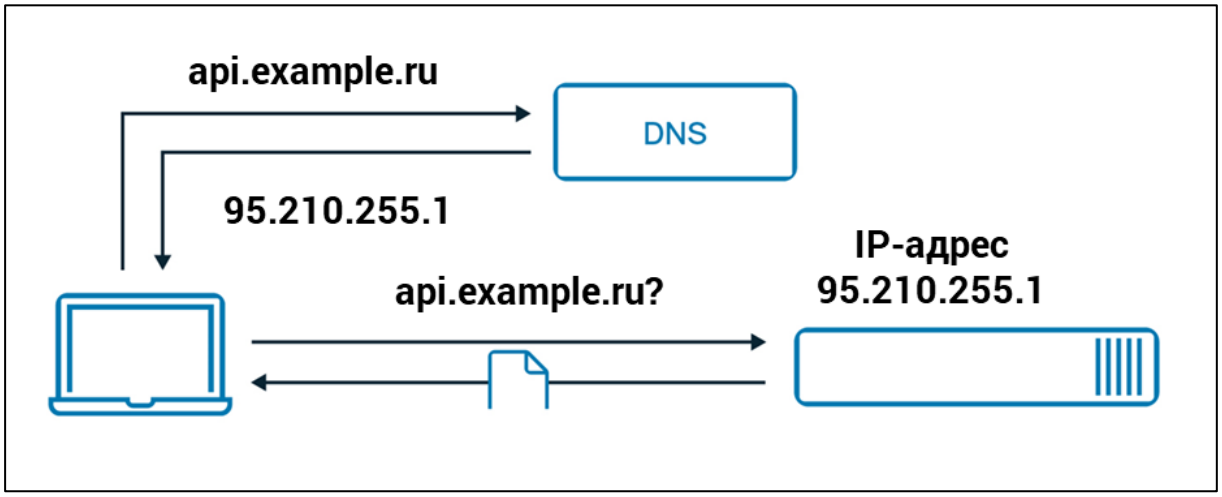


Рис. 1.26. Схема сопоставления DNS-имени и IP-адреса с помощью DNS-сервера

Регистрация и выделение доменов

За каждую доменную зону первого уровня отвечает организация, определяющая правила выделения доменов и обеспечивающая обслуживание этой зоны. Например, доменные зоны ru, su, рф администрирует Координационный центр национального домена сети Интернет: https://cctld.ru.

Подобные организации также устанавливают правила работы и задают технические требования к регистраторам доменов. Наиболее крупным регистратор в РФ является Ru-Center (nic.ru).

При этом важно учитывать, что регистрация домена не является приобретением доменного имени, а фактически – это его аренда на определённый срок (минимально – 1 год). Владельцем всех доменных имен является организация ICANN – международная корпорация по управлению доменными именами и IP-адресами.

После регистрации домена пользователь становится администратором домена и может использовать его для размещения сайта, настройки почтового сервера или других задач.

Вопросы для самопроверки

1. Для каких задач используются сетевые протоколы?
2. В чем состоит особенность протоколов модели OSI?
3. Перечислите и опишите уровни модели OSI.
4. Какую роль играет MAC-адрес устройства?
5. Что такое сокет и как его определить средствами операционной системы).
6. В чем отличие классовой и бесклассовой IP-адресации.
7. В каких случаях рациональнее использовать статический IP-адрес?
8. Опишите схему работы DNS-сервера.

Практикум

Задание 1

1. Изучите некоторые возможности утилиты командной строки netstat. Узнать дополнительные приемы работы с ней можно узнать, например, по ссылкам:
   a. 11 приемов использования netstat;
   b. NETSTAT - статистика активных подключений TCP.
2. Откройте командную строку от имени администратора.
3. Отобразите справку по команде.
4. Запросите статистику Ethernet и статистику по всем протоколам (используйте необязательные опции команды).
5. Получите информацию только по протоколам TCP и UDP.
6. Запросите показ списка активных подключений TCP и коды процессов с обновлением каждые 10 секунд.

Задание 2

1. Найдите информацию по основным возможностям утилиты командной строки ipconfig.
2. Среди полученных данных найдите IP-адрес и MAC-адрес компьютера, кукую маску использует подсеть.
3. Получите аналогичную информацию, используя визуальный интерфейс вашей операционной системы (рис. 1.27). Например, для Windows 10 это можно осуществить несколькими способами (см. ссылку).

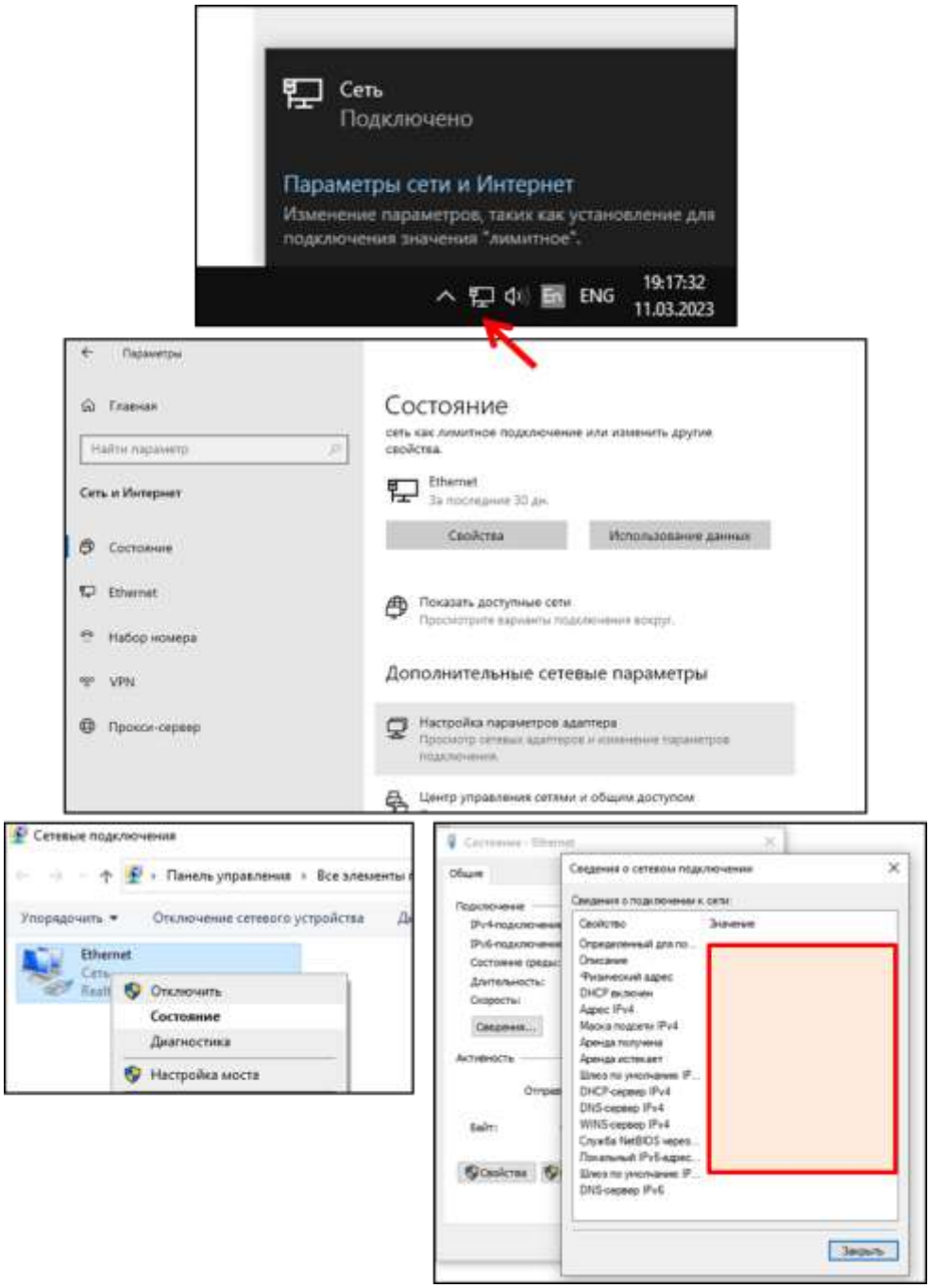


Рис. 1.27. Получение сведений о сетевом подключении

Задание 3

1. Воспользуйтесь одним из онлайн-сервисов для получения информации о IP-адресе вашего ПК, например:
   a. https://2ip.ru/;
   b. https://www.reg.ru/web-tools/myip.
2. Какую информацию о вашем устройстве и провайдере удалось установить?

3. Выключите и включите (если возможно) ваш роутер. Поменялся ли IP-адрес? Является ли он динамическим?

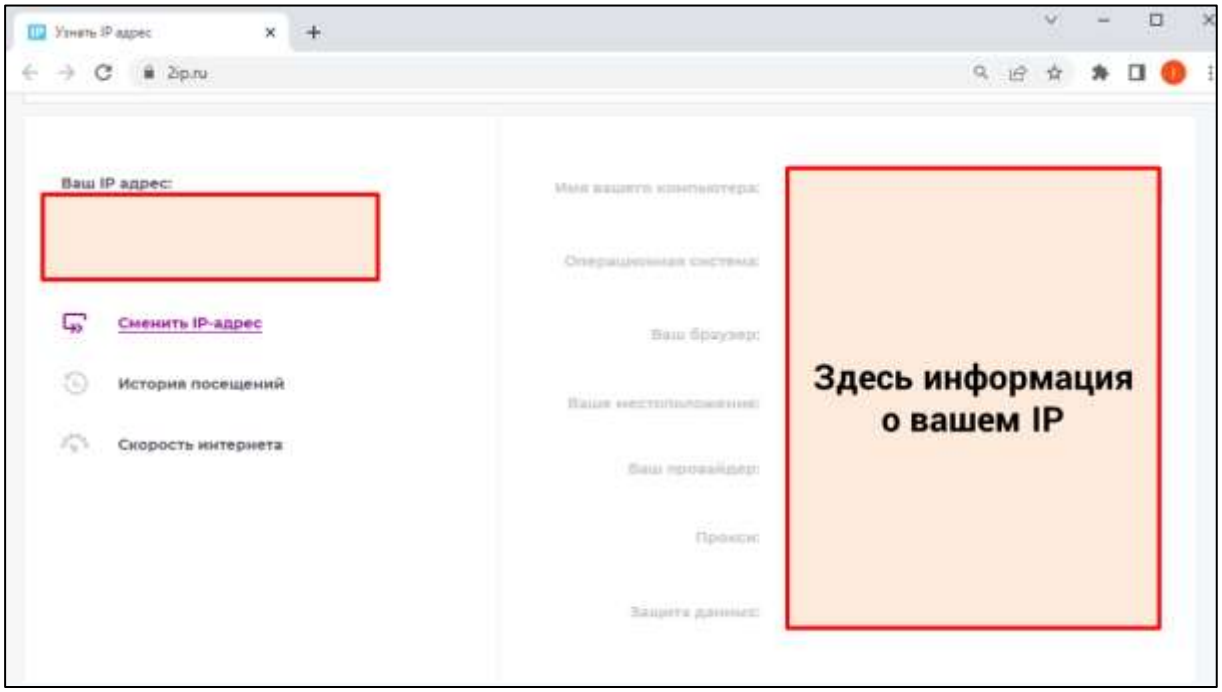


Рис. 1.28. Информация о IP устройства в сети на примере веб-сервиса 2ip.ru

Задание 4

1. Откройте настройки вашего роутера. Для этого в адресной строке браузера введите его IP-адрес. Обычно это 192.168.0.1, однако в общем случае он может зависеть от производителя. Подробнее информацию о подключении к роутеру можно узнать по ссылке: https://192-168-0-1.ru/.
2. Введите логин и пароль для авторизации (изначально должны быть указаны внизу / сзади вашего роутера).
3. Изучите информацию о настройках роутера и сетевом соединении (рис. 1.29-рис. 1.30). Если для входа был установлен пароль по умолчанию или очень простой для взлома, рекомендуется поменять его на более надежный (рис. 1.31).

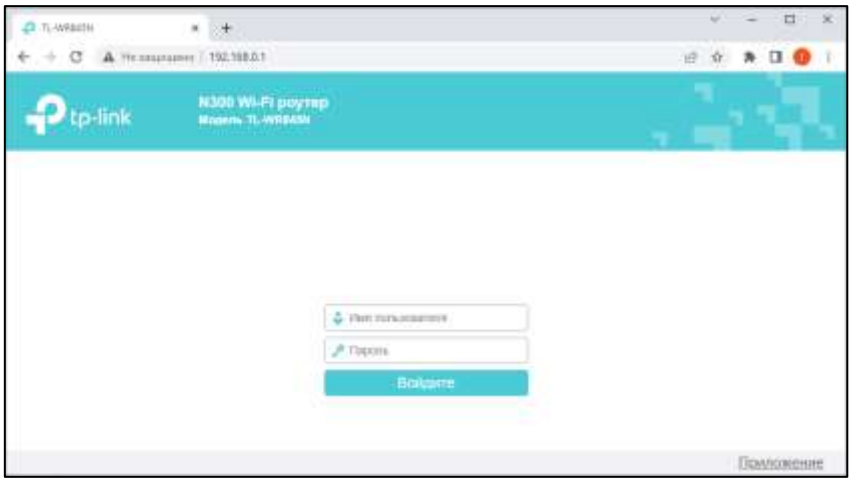


Рис. 1.29. Авторизация в окне настройки маршрутизатора

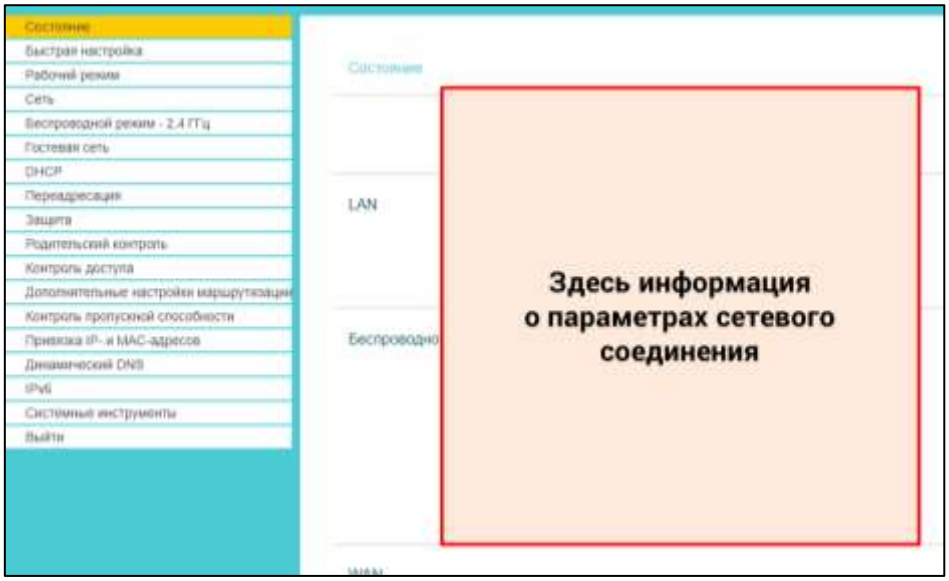


Рис. 1.30. Просмотр информации о сетевых настройках

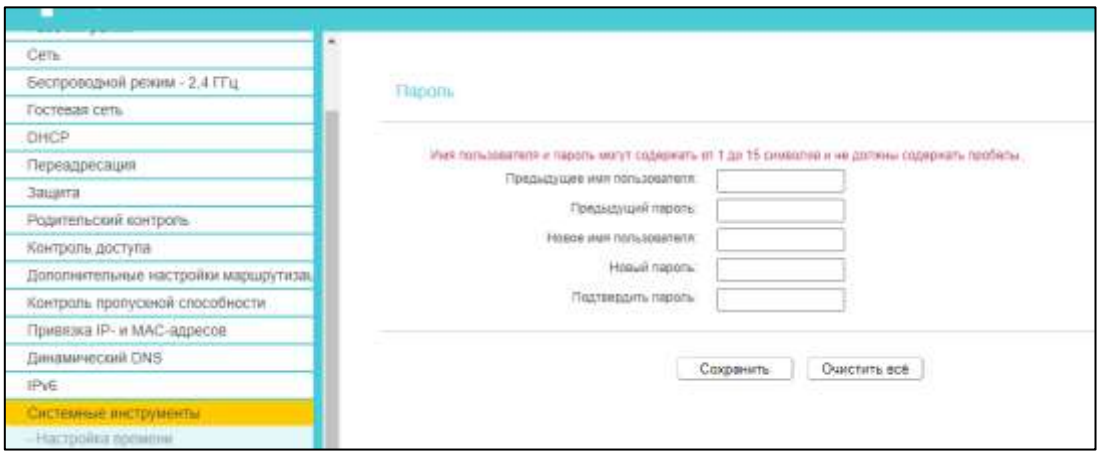


Рис. 1.31. Обновление пароля для доступа к настройкам роутера

1.3. Службы сети Интернет

1.3.1. Службы сети Интернет

Понятие службы

Определение

Служба *(или* ***сервис****) сети Интернет – это две или более программы, взаимодействующие между собой согласно определенным протоколам и предоставляющие услуги пользователям.*

Обычно взаимодействие происходит между двумя узлами, один из которых выполняет роль *клиента*, а другой – *сервера*. К наиболее известным сетевым службам относят электронную почту, технологию WWW, телеконференции, списки рассылки, FTP, чаты, IRC.

В зависимости от времени отклика службы их услуги можно разделить на две категории:

1. *Отложенные* (*off-line*), которые допускают задержку времени между запросом и получением информации. Примером являются почтовые сервисы.
2. *Прямые* (*on-line*), для которых важен немедленный ответ в режиме реального времени. Этот режим работы характерен для чатов, видеотрансляций и т.д.

Основные службы и их назначение

Краткий перечень

На протяжении истории развития Интернет-технологий было создано множество сервисов, часть из которых уже не используется либо теряет свою популярность.

Среди наиболее актуальных отметим:

- World Wide Web – Всемирная паутина, служба для поиска и просмотра гипертекстовых электронных документов, поддерживающая обработку графики, звука и мультимедиа;
- E-mail – служба электронной почты;
- FTP – служба передачи файлов;

- Usenet, News – сервисы телеконференций и группы новостей;
- ICQ – служба обмена текстовыми сообщениями в реальном времени;
- Telnet – служба для удаленного управления компьютерами в режиме терминала.

WWW

WWW (*World Wide Web*) – это служба, обеспечивающая прямой доступ к электронным документам сети Интернет и позволяющая взаимодействовать с веб-сайтами в интерактивном режиме. В настоящее время это одна из наиболее распространенных и продолжающих свое активное развитие сетевых технологий.

Основу WWW составляет технология *гипертекста* – особых фрагментов текста или графических элементов в разметке веб-документа, которые позволяют осуществлять переход внутри документа или на другие веб-страницы сети Интернет.

Изначально работа службы WWW базировалась на трех стандартах (далее каждая технология рассматривается подробнее):

1. HTML – язык гипертекстовой разметки документов, представляющий собой набор специальных команд для разметки документа;
2. URL – универсальный способ идентификации ресурсов в сети;
3. HTTP – основной протокол для обмена гипертекстовой информацией.

В последствии была добавлена технология CGI (Common Gateway Interface), определяющая универсальный интерфейс шлюзов. Она потребовалась для взаимодействия HTTP-сервера с другими программами (например, СУБД). Достоинством CGI является поддержка мультимедийных ресурсов (аудио, видео, текст, анимация).

Для обмена данными служба WWW использует протокол HTTP. Документы в формате HTML анализируется браузерами и отображаются в надлежащей визуальной форме.

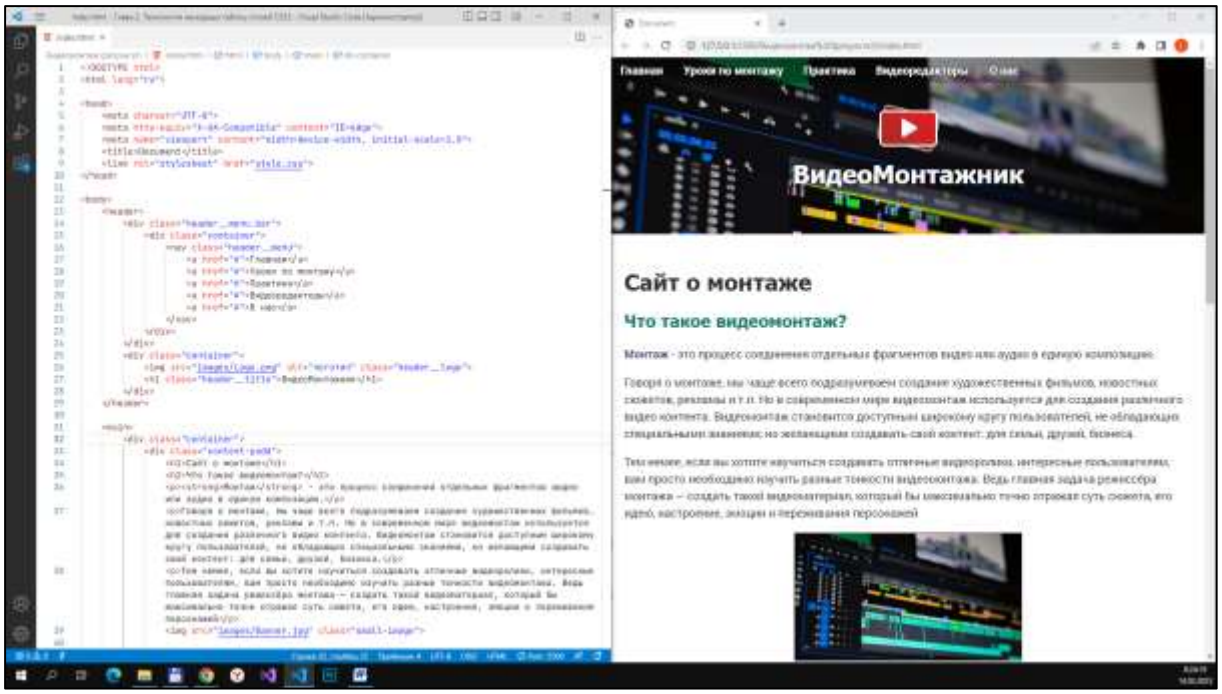


Рис. 1.32. Разметка HTML-документа (текстовый файл) и его представление в браузере

E-mail

Технологически пересылка электронных сообщений была реализована еще задолго до появления сети Интернет и веб-сервисов. В 1965 году учёные Массачусетского технологического института осуществляют отправку первого аналога современного электронного письма в рамках системы ARPANET. Позже инженер компании BBN Рэй Томлинсон разработал программу для пересылки электронных сообщений по сети. Он также ввел в обозначение почтового адресата символ *@*.

Электронная почта (*E-mail*) – сервис, предназначенный для передачи текстовых сообщений в сети Интернет, а также между другими сетями электронной почты. Современные почтовые сервисы поддерживают массовую рассылку, работу с мультимедиа и могут быть синхронизированы с разным ПО и веб-сервисами.

Передача сообщения предполагает разбиение потока данных на пакеты, каждый из которых содержит адрес получателя и отправителя. После доставки пакета получателю анализируется его адрес и в случае совпадения пакет принимается, иначе отправляется дальше. Все полученные пакеты накапливаются и соединяются в единое сообщение. Копии пакетов дополнительно сохраняются на узлах-

отправителях, пока узел-получатель не подтвердит успешную доставку. Этим обеспечивается надёжность передачи сообщений.

Каждый клиент почтового сервиса получает уникальный почтовый адрес следующего вида:

```
<имя пользователя>@<имя почтового сервера>
```

Например, в почтовом ящике

```
denis.stud_worker@mail.ru
```

именем пользователя (логином) является `denis.stud_worker`, а `mail.ru` указывает на почтовый сервис.

Одной из первых и долгое время популярных программ для работы с электронными сообщениями стала Microsoft Mail, запущенная в 1988 году. Сегодня пользователи активно использую почтовые клиенты Mailbird, eM Client, Claws Mail, TouchMail, Thunderbird, The Bat!.

Для работы почтовых сервисов необходим ряд протоколов, в частности SMTP, POP3, IMAP.

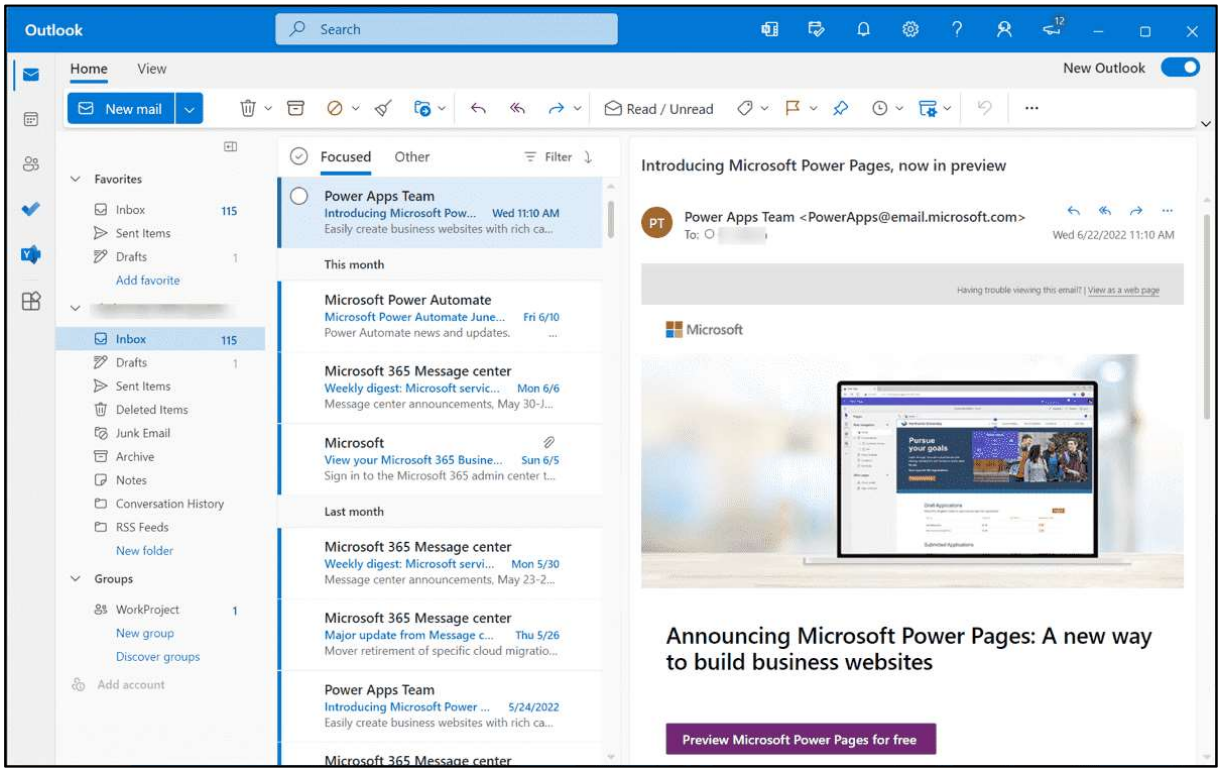


Рис. 1.33. Почтовый клиент MS Outlook

В 1996 году впервые была организована работа с Email через веб-интерфейс. Разработанный Джеком Смитом первый почтовый

сервер HotMail позволил пользователям отправлять и получать сообщения, используя браузер.

В настоящее время такой подход активно вытесняет работу с веб-клиентами, поскольку не требует обязательной установки дополнительного ПО на устройство пользователя. Работать с электронными письмами можно непосредственно с веб-сайта.

Отдельными проблемами пересылки электронных сообщений стала рассылка нежелательного контента («спам») и потенциальная опасность получения зараженных компьютерными вирусами файлов (даже несмотря на развитие механизмов фильтрации писем).

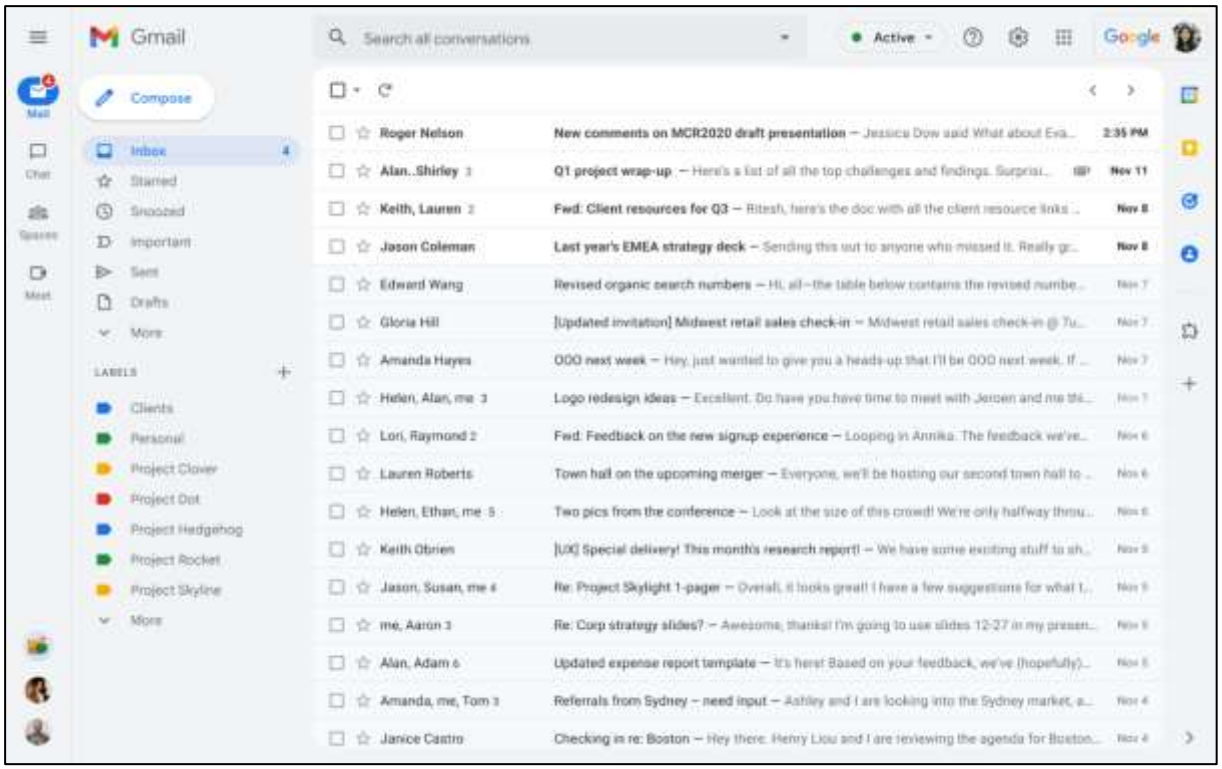


Рис. 1.34. Сервис электронной почты Gmail от Google

FTP

Служба передачи файлов FTP используется для пересылки больших документов и архивных файлов. FTP появился в 1971 году, задолго до HTTP и TCP/IP, и до сих пор остается одним из старейших используемых прикладных протоколов.

FTP использует одновременно два соединения между сервером и клиентом. Первое соединение отвечает за передачу данных, а второе – за управление потоками. Служба FTP передает копии файлов из одного узла сети Интернет на другой в соответствии с одноименным

протоколом FTP (File Transfer Protocol, протокол передачи файлов). Компьютеры в FTP-соединениями называют *FTP-серверами*.

В настоящее время используются такие клиенты как WinSCP, CuteFTP, FileZilla, Total Commander, FTPManager. Многие из таких программ также являются файловыми менеджерами.

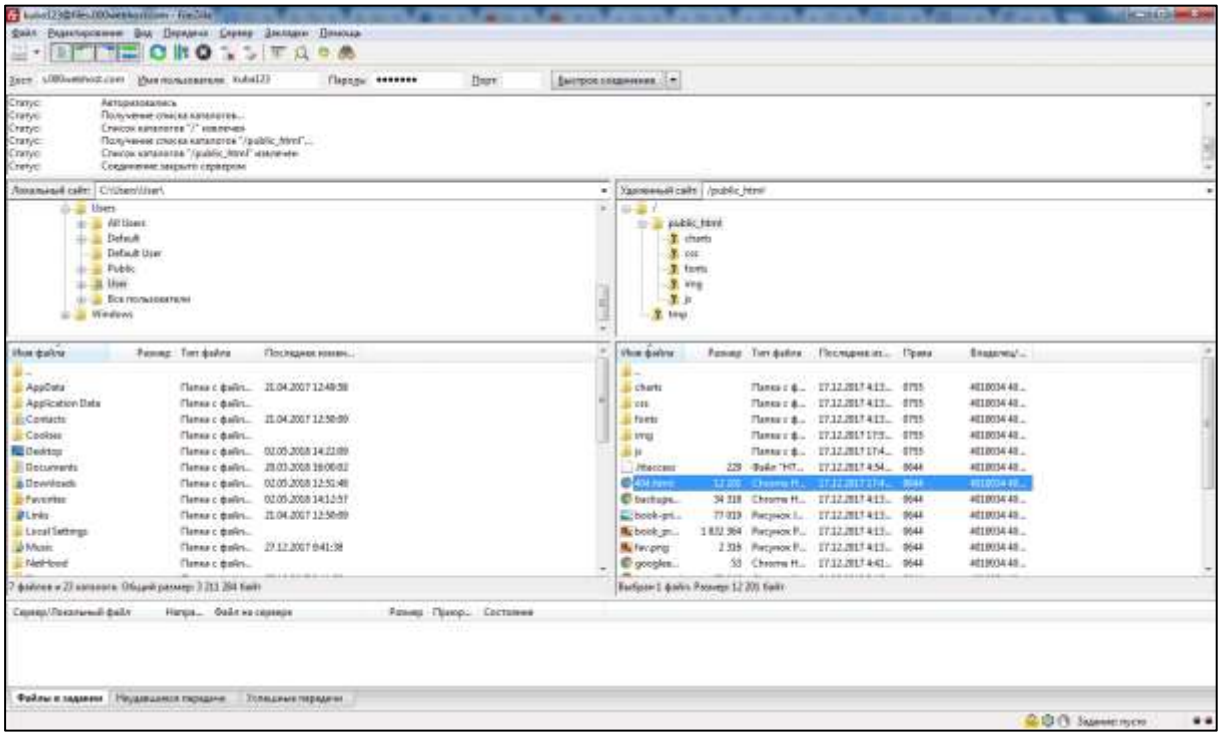


Рис. 1.35. FTP-клиент FileZilla

Usenet

Usenet (от англ. user network) представляет собой компьютерную сеть для общения и публикации файлов. Usenet объединяет новостные группы, в которых пользователи могут публиковать сообщения. В силу популярности предлагаемых форматов общения сервис оказал большое влияние на развитие веб-технологий.

Usenet поддерживает формат телеконференций и использует сетевой протокол передачи новостей NNTP. Для распространения новостей Usenet обрабатывает запросы пользователей и распространяет данные на другие сервера, подписанные на искомую тему.

В телеконференциях можно выделить несколько доступных групп:

- news – вопросы о системе телеконференций;
- comp – компьютерные технологии и ПО;

- rec – развлечения, досуг, хобби, искусство;
- sci – научно-исследовательская деятельность и ее прикладные вопросы;
- soc – общественные вопросы;
- talk – дебаты;
- misc – другие темы.

Внутри каждой категории предусмотрена иерархия. Например, категория rec.music.beatles посвящена обсуждению творчества Битлз и включена в подкатегорию музыки.

ICQ

ICQ (от англ. «I seek you», «я ищу тебя») – система мгновенного обмена сообщениями, поддерживающая голосовую и видеосвязь. ICQ бесплатен и может работать на многих современных операционных системах.

Мессенджер ICQ был создан в середине 1996 года. Для работы пользователю было необходимо установить приложение (ICQ-клиент) и зарегистрироваться в системе. ICQ получил высокую популярность благодаря скромным требованиям к Интернет-трафику, гибким возможностям обмена информацией и мобильности.

Однако с появлением крупных порталов и социальных сетей популярность сервиса стремительно упала. Многие веб-ресурсы и приложения для онлайн-трансляции поддерживают все функции, которые были доступны в ICQ.

Рис. 1.36. Старый интерфейс ICQ

TELNET

TELNET является службой удаленного доступа и управления системой. Сервис работает на основе одноименного протокола и состоит из программы-клиента и программы-сервера.

Задачи программы-клиента:

- устанавливает соединение с сервером;
- получает данные от абонента, преобразует их к стандартному формату и высылает серверу;
- принимает от сервера результаты запроса в стандартном формате и преобразует их в удобный для клиента формат.

Задачи программы-сервера:

- ожидает запрос в стандартизированной форме;
- обслуживает полученный запрос;
- высылает результаты программе-клиенту.

TELNET разрабатывался для организации удалённого доступа к интерфейсу командной строки. Со временем его также стали использовать в других текстовых интерфейсах.

Важную. роль TELNET играет в обеспечении удалённой работы с сетевым оборудованием: он предоставляет полный набор команд для управления системами в текстовой форме, однако требует от специалиста опыта работы с командной строкой.

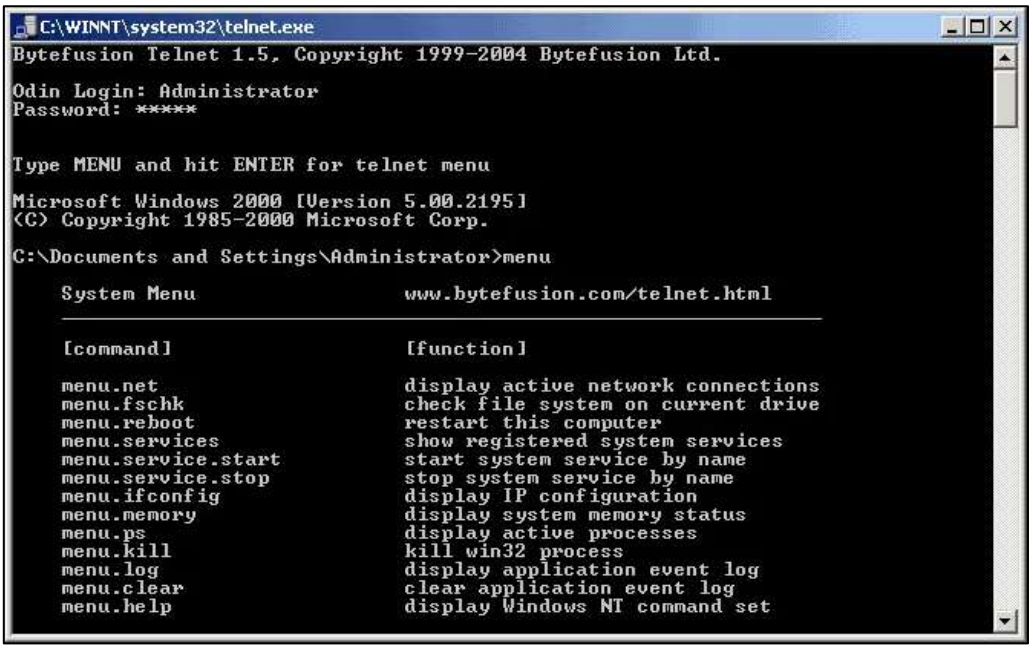


Рис. 1.37. Работа с TELNET

1.3.2. Технология WWW

Технология гипертекста

Предыстория

Определение

Гипертекст *– способ организации текста, позволяющий осуществлять нелинейные переходы между разными фрагментами, связанными ссылками.*

Первые концепции будущей гипертекстовой технологии были предложены еще в 1945 году Ванневаром Бушом, который обратил внимание на необходимость создания механизмов, упрощающих поиск информации. Ученый предложил модель механической гипертекстовой системы Memex, которая бы позволила автоматизировать выборку информации по запросу пользователя и делать это за малый промежуток времени.

Термин «гипертекст» был сформулирован Т. Нельсоном в 1965 году. В концепции Нельсона такой текст предполагал возможность интерактивного перехода между различными частями электронных документов, что отличается от привычной работы с печатными источниками (например, энциклопедиями).

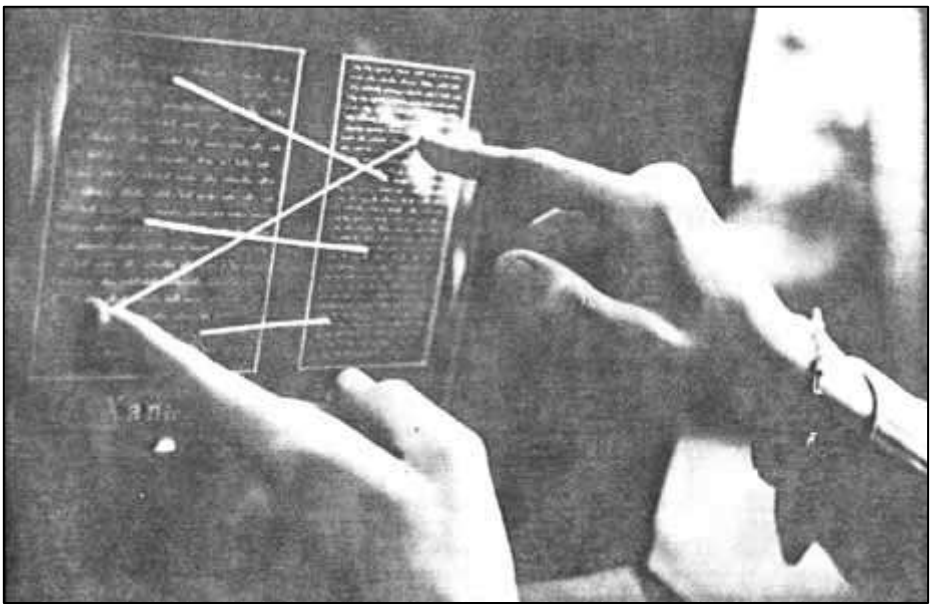


Рис. 1.38. Визуальное представление гипертекстовой связи

Структуру гипертекста можно представить в форме базы данных, узлами которой являются линейные фрагменты текста. Между этими фрагментами может быть установлена ссылочная связь, на которую не накладываются какие-либо ограничения типов данных, соотношения и структуры (рис. 1.38).

Существенный вклад в развитие будущих гипертекстовых технологий внес Дуглас Энгельбарт. 9 декабря 1968 года Энгельбарт продемонстрировал систему oN-Line System (NLS), которая поддерживала многие функции, характерные современным компьютерным технологиям: манипулятор (мышь), работа с графикой, видеоконференцсвязь, оконный интерфейс программ, контроль версий, а также гипертекст. Его выступление было столь значимым в истории, что его назвали «Мать всех демов».

Рис. 1.39. oN-Line System в действии

В 1977 году в Массачусетском технологическом институте была создана карта фильмов Аспена – это интерактивное видео, позволившее зрителям проводить виртуальную экскурсию по городу Аспен.

Зарождение HTML

Родоначальником современного гипертекста считается Тим Бернерс-Ли, который в 1980 году создает первую электронную энциклопедию INQUIRE (аналог Wikipedia). Система состояла из базы данных, в которой использовались двунаправленные гиперссылки на связанные текстовые блоки.

Рис. 1.40. Тим Бернес-Ли и системе

Позже, в период с 1986-1991 годов Тим работает над созданием нового языка гипертекстовой разметки HTML, который должен был прийти на смену существовавшего SGML. HTML разрабатывался в качестве языка обмена научной и технической документацией. При этом упор делался на простоту и удобство понимания его как человеком, так и обработку компьютером. В разметке документа использовались специальные дескрипторы, называемые тегами: они определяли структуру и роль элементов в разметке.

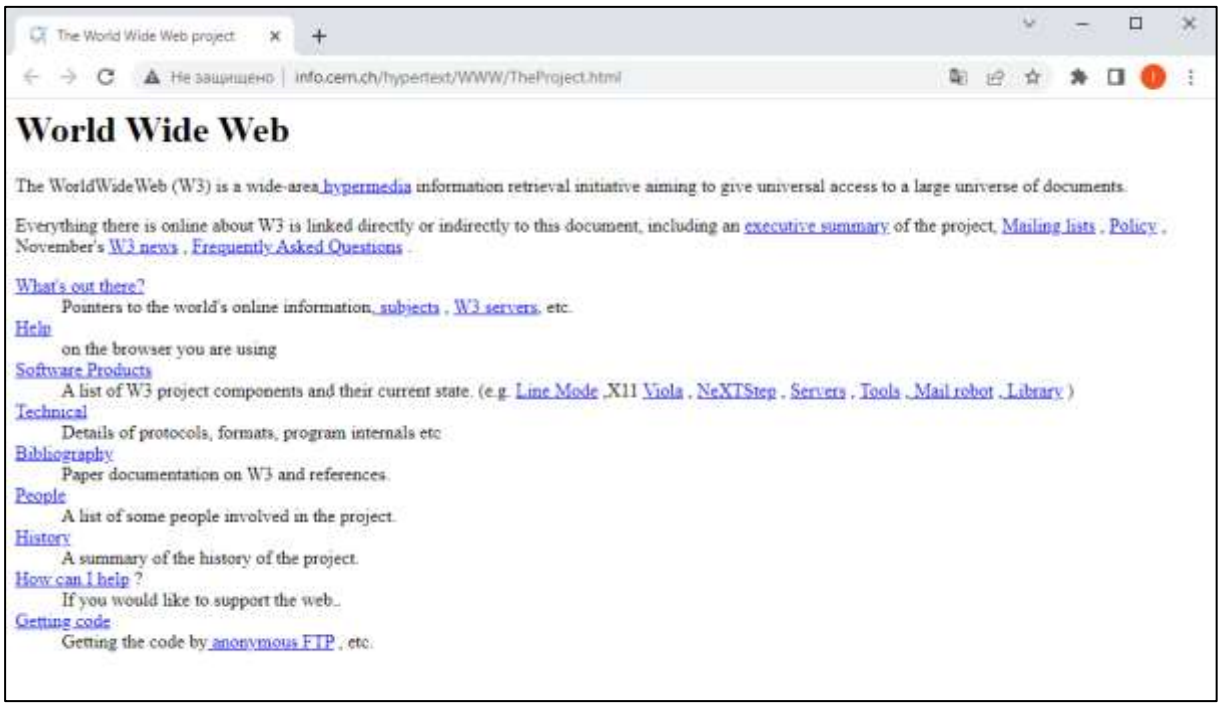


Рис. 1.41. Первая веб-страница в сети, размеченная на HTML 1.0 (1989год)

HTML не только задавал структуру документа, но и определял правила его визуального оформления. Со временем кроме возможности интерактивных переходов была добавлена поддержка мультимедийных возможностей.

В настоящее время спецификация HTML5 позиционируется как язык логической и структурной разметки документа, задающий только минимальные правила форматирования документов в браузере. За визуальное оформление отвечают другие технологии, в частности – CSS, которые работают в связке с HTML.

Рис. 1.42. Современный веб-сайт, размеченный с помощью HTML: поддерживает работу с мультимедиа

URL. URI. URN

URI

Определение

URI *(Uniform Resource Identifier, единообразный идентификатор ресурса) – короткая последовательность символов, идентифицирующая абстрактный или физический ресурс.*

URI одновременно идентифицирует веб-ресурс как по адресу, так и по названию.

Формально URI имеет следующий вид (в квадратных скобках обозначена необязательная часть):

```
схема:[//полномочия]путь[?запрос][#фрагмент]
```

- *Схема* – показывает способ обращения к ресурсу; чаще всего это сетевой протокол (http, ftp, ldap).
- *Путь* – данные, необходимые для идентификации ресурса (например, адрес сайта).
- *Запрос* – дополнительные данные ресурса (например, для поиска и фильтрации).
- *Фрагмент* – компонент для идентификации вторичного ресурса (например, место на странице).

Например, URI

```
http://example.com:8042/over/there?name=fruits#apples
```

обращается к ресурсу «example.com» через порт 8042, где ищет подкаталог «/over/there» и веб-страницу с параметром названия «name» со значением «fruits» и переходит к позиции разметки области «apples».

URI может идентифицировать и другие типы ресурсов, например:

- `mailto:denis@example.com` – ссылка на почтовый ящик;
- `telnet://192.0.2.16:80/` – обращение к TELNET;
- `urn:isbn:0-476-27557-4` – ссылка на ISBN.

URL

Определение

***URL** (Uniform Resource Locator) – это URI, который дополнительно предоставляет и информацию о местонахождении этого ресурса (локализацию).*

URL обычно используется для поиска ресурсов на сервере: он содержит точный адрес веб-ресурса.

Идентификатор URL имеет структуру, аналогичную URI, и содержит следующие компоненты:

- Протокол доступа к ресурсу (http, https, ftp).
- Локализация сервера по DNS или IP-адресу.
- Номер порта на сервере.
- Точное местоположение в иерархии каталогов сервера.
- Необязательный идентификатор фрагмента.

Например, URL

```
ftp://da.myfiles.net/programming/javascript-book-2023.pdf
```

указывает на файл `javascript-book-2023.pdf`, обращение к которому ведется посредством протокола FTP. Файл расположен на ресурсе `da.myfiles.net`, внутри каталога `programming`.

Другие примеры URL:

- `https://google.com` – адрес веб-сайта поисковика;
- `https://yandex.ru/search/?text=ethernet&lr=192` – запрос по ключевому слову «ethernet» в поисковике.

URN

Определение

***URN** (Uniform Resource Name) – идентификатор ресурса в виде его названия.*

URN обозначает уникальное название ресурса, вне зависимости его расположения или существования.

Достоинствами URN является

- независимость от адреса ресурса;
- возможность его перемещения из одного места в другое;
- доступ к ресурсу по имени посредством обращения к разным протоколам.

Например, задан URN следующего вида:

```
urn:isbn:4990223511
```

Доступ к ресурсу может быть организован иными способами (протоколами):

```
urn:issn:1082-9873
urn:doi:10.1000/1
urn:oasis:names:specification:docbook:dtd:xml:4.1.2
```

В настоящее время URN считается экспериментальным и менее распространен, в отличие от URL, поскольку такая адресация требует дополнительного развития сетевой инфраструктуры. URN в будущем должны заменить URL, что позволит избавиться от проблемы идентификации и поиска ресурсов в сети в случае их перемещения.

Разница между URL и URI

Из приведенного выше описания следует, что URL и URN являются частной реализацией URI. Следующая таблица раскрывает разницу между URI и URL:

Таблица 1.1. Сопоставление URI и URL

URI	URL
URI – это идентификатор ресурса.	URL – локатор ресурса.
Идентификация ресурса и отличие от других ресурсов с помощью имени или местоположения ресурса.	Поиск местоположения или адреса ресурса в Интернете.
Можно использовать для идентификации и отличия HTML, XML и других файлов друг от друга.	Можно использовать только для идентификации и поиска веб-страниц.
Может быть протоколом, обозначением, спецификацией или др.	Обычно протокол HTTP, HTTPS, FTP и т.п.

Протокол HTTP

Понятие и назначение HTTP

Определение

HTTP *(HyperText Transfer Protocol, протокол передачи гипертекста) – это протокол прикладного уровня (OSI, 7-й уровень), предназначенный для передачи данных. Изначально предназначался для передачи гипертекстовых документов, размеченных на HTML.*

Протокол HTTP работает в клиент-серверной модели передачи данных. Общая схема работы протокола следующая:

1. устанавливается TCP-соединение (обычно через 80-й порт);
2. клиентское приложение (например, браузер) отправляет запрос клиента;
3. сервер обрабатывает запрос и отправляет клиенту ответ;
4. осуществляется разрыв TCP-соединения.

В настоящее время протокол HTTP является основным, благодаря которому осуществляется работа Всемирной паутины.

HTTP часто задействуется в роли «транспортного» протокола передачи информации другим протоколам прикладного уровня, например, SOAP, XML-RPC и WebDAV.

Многие программные продукты используют HTTP для передачи данных и могут иметь разные форматы, например, XML или JSON.

Структура протокола HTTP

HTTP-запрос состоит из трех компонент (рис. 1.43):

1. *Заголовок сообщения*: в начале указана строка состояния, устанавливающая тип сообщения, и поля заголовка.
2. Пустая строка.
3. Тело сообщения — непосредственно данные сообщения.

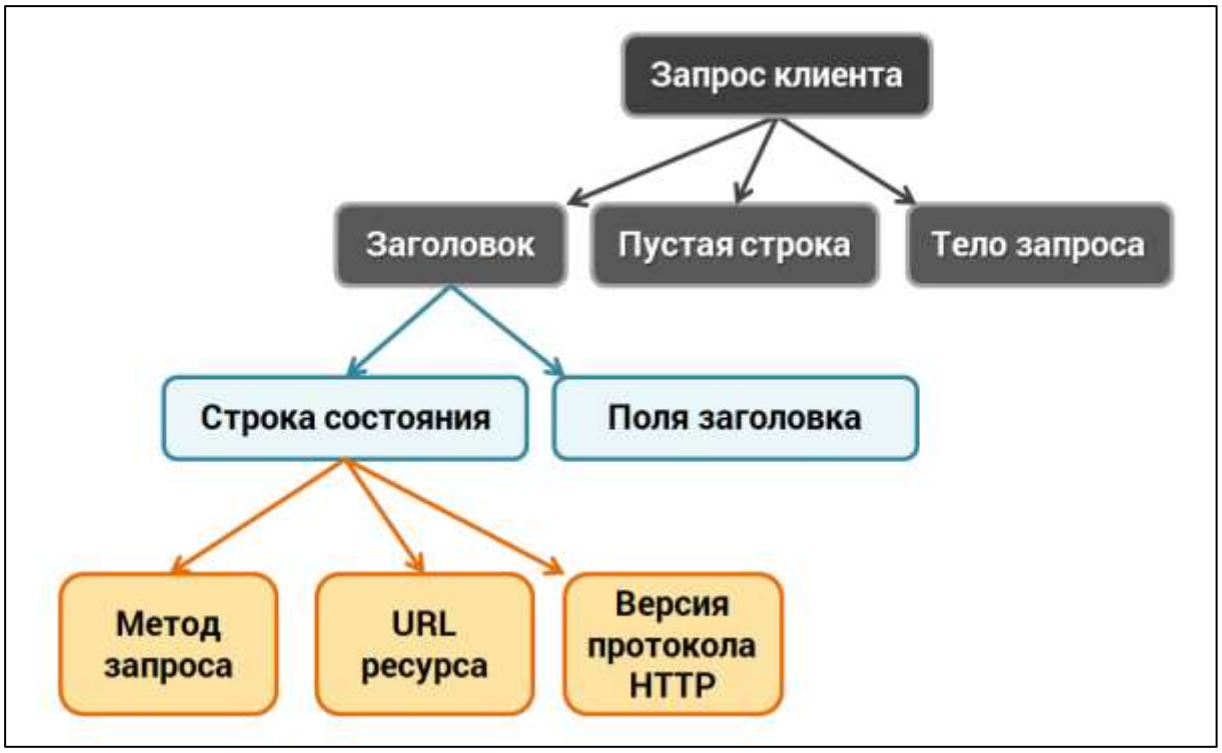


Рис. 1.43. Структура HTTP-запроса

Ключевым элементом структуры запроса является начальная строка. Она указывает *метод*, который осуществляет действия со страницей. Методу могут быть указаны дополнительные параметры.

Стартовая строка запроса имеет вид:

```
Метод URL HTTP/Версия
```

- *Метод* задает название метода для обработки запроса.
- *URL* определяет местоположение веб-ресурса.
- *Версия* указывает версию протокола HTTP в ответе сервера.

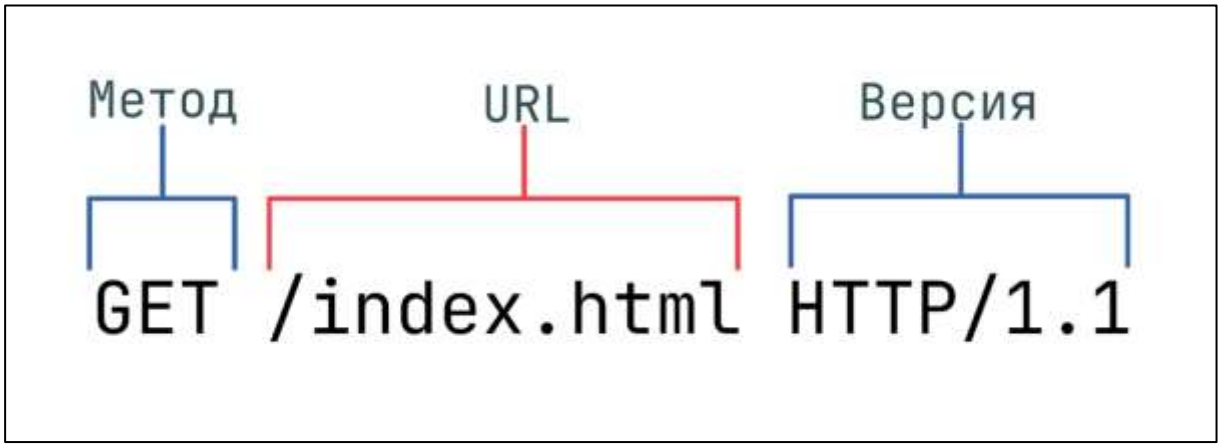


Рис. 1.44. Пример стартовой строки запроса и ее компоненты

Методы запроса клиента

Рассмотрим часто востребованные методы HTTP.

Метод *GET* – запрашивает данные из указанного ресурса (сервера). Дополнительные данные передаются в строке запроса.

Синтаксис метода:

```
GET /path/resource?param1=value1&param2=value2 HTTP/1.1
```

где `param1` и `param2` – параметры метода, `value1` и `value2` – их значения (параметров может быть больше).

Например, запрос

```
GET / HTTP/1.1
Host: test.yakubovich.ru
```

обращается к главной странице ресурса `test.yakubovich.ru`.

Метод *SET* предназначен для отправки данных на ресурс (сервер). Дополнительные данные передаются в строке запроса.

Обычно SET отправляет данные из *формы* – фрагмент разметки HTML-документа, содержащий управляющие элементы (используется тег-контейнер `<form>`).

Общий синтаксис метода:

```
POST /path HTTP/1.1
Host: domain
query
```

Например, следующий запрос

```
POST /test/test_form.php HTTP/1.1
Host: test.yakubovich.ru
status=admin&rank=2
```

запрашивает генерацию веб-страницы `test_form.php` из каталога `test`, который расположен на ресурсе `test.yakubovich.ru`. Дополнительно запрос передает статус клиента и его ранг.

Метод *OPTIONS* позволяет определить возможности веб-сервера или параметров соединения для конкретного ресурса.

В качестве ответа возвращается список поддерживаемых методов. Дополнительно в заголовке ответа может размещаться информация о поддерживаемых расширениях.

Например, запрос

```
OPTIONS * HTTP/1.1
```

обратится к веб-серверу, а не к конкретному ресурсу.

Метод DELETE удаляет текущий ресурс.

Важная информация HTTP-запроса хранится в полях заголовка. *Поля заголовка* следуют за строкой состояния и уточняют запрос, передавая серверу дополнительные данные.

Формат каждого поля имеет вид «ключ-значение»:

```
Имя_поля: значение
```

```
GET /index.html HTTP/1.0

Connection: Keep-Alive
User-Agent: Mozilla/4.04 [en]   (Win95;  I)
Host: oak.oakland.edu
Accept: image/gif, image/x-xbitmap, image/jpeg, image/pjpeg, image/png, */*
Accept-Language: en
Accept-Charset: iso-8859-l,*,utf-8
```

Рис. 1.45. Пример GET-запроса с дополнительными данными

В следующей таблице описана роль некоторых полей заголовка.

Таблица 1.2. Некоторые поля HTTP-запроса

Поле	**Значение**
Host	Доменное имя либо IP-адрес узла, к которому обращается клиент.
Referer	URL документа, который ссылается на ресурс, указанный в строке состояния.
From	E-mail пользователя, работающего с клиентом.
Accept	MIME-типы данных, обрабатываемых клиентом (т.е. формат файлов).
Accept-Language	Двухсимвольные идентификаторов поддерживаемых языков.
Accept-Charset	Поддерживаемые наборы символов.
Content-Type	MIME-тип данных в теле запроса (для случая, если запрос не состоит только из заголовка).

Content-Length	Число символов в теле запроса.
Range	Указывается в случае, когда клиент запрашивает только часть документа.
Connection	Необходимо для управления TCP-соединением. • При значении `Close` после обработки запроса сервер должен закрывает соединение. • При значении `Keep-Alive` не закрывает TCP-соединение, и его можно использовать для других запросов.

Ответ сервера

Структура HTTP-ответа (сообщения сервера) аналогична запросу: он состоит из строки состояния, поля заголовка, пустой строки и тела ответа.

Ответ сервера клиенту начинается со строки состояния:

```
Версия_протокола   Код_ответа   Пояснительное_сообщение
```

- *Версия_протокола* задается в том же формате, что и в запросе клиента.
- *Код_ответа* – это трехзначное десятичное число, представляющее в закодированном виде результат обслуживания запроса сервером.
- *Пояснительное_сообщение* дублирует код ответа в символьном виде. Не обрабатывается клиентом и предназначена для системного администратора или оператора.

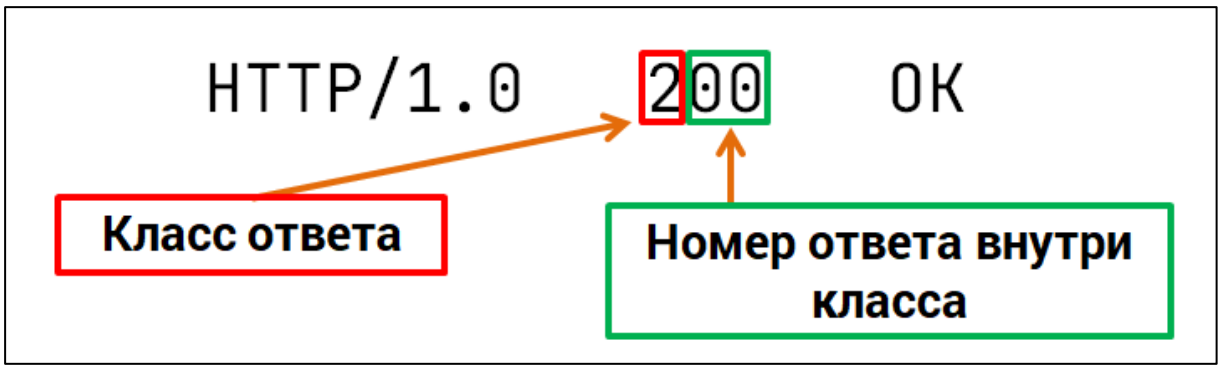


Рис. 1.46. Строка состояния ответа

С помощью кода ответа HTTP-сервер информирует клиента о статусе запроса. Выделяют 5 классов состояний.

Таблица 1.3. Классы состояний ответа

Класс	**Описание**
1xx	Специальные сообщения, часто называемые информационными. Не влияют на обработку запроса. Используется редко.
2xx	Показывают успешную обработку запроса клиента.
3xx	Показывают необходимость перенаправления запроса клиента.
4xx	Ошибка в запросе со стороны клиента (обычно синтаксическая).
5xx	Ошибка на стороне сервера.

Обозначим наиболее распространенные коды ответов.

Таблица 1.4. Примеры кодов ответов сервера

Код	**Расшифровка**	**Интерпретация**
100	Continue	Сервер принял часть запроса и ожидает его продолжения.
200	OK	Успешная обработка запроса и передача данных клиенту.
201	Created	Создание нового ресурса.
202	Accepted	Обработка запроса сервером не завершена: нет гарантии, что запрос был обработан без ошибок.
206	Partial Content	Возвращена часть ресурса (при этом запрос содержал поле заголовка Range).
301	Multiple Choice	Запрос указывает несколько ресурсов. Тело ответа может содержать указания для дальнейшей идентификации ресурса.
302	Moved Temporaril	Запрашиваемый веб-ресурс отсутствует на сервере или временно изменил адрес.
400	Bad Request	В запросе обнаружена синтаксическая ошибка.

403	Forbidden	Ресурс недоступен для искомого пользователя.
404	Not Found	Указанный клиентом ресурс отсутствует на сервере.
405	Method Not Allowed	Сервер не поддерживает метод обработки.
500	Internal Server Error	Какой-либо компонент сервера работает некорректно.
501	Not Implemented	Сервер не может выполнить запрос клиента (недостаточно функционала).
503	Service Unavailable	Служба временно недоступна.
505	HTTP Version not Supported	Сервер не поддерживает указанную версию HTTP.

Обычно тело ответа формируется HTML-разметкой. Кроме того, в составе ответа могут также передаваться медиа-файлы: изображения, аудио, видеофрагмент, любой другой формат данных, поддерживаемых клиентом.

Правила обработки полученного ресурса зависят от содержимого поля заголовка Content-type (см. выше).

Рассмотрим пример ответа. Поля заголовка содержат необходимую информацию, необходимую клиенту для корректной обработки ответа (рис. 1.47).

```
HTTP/1.1 200 OK
Server: Microsoft-IIS/5.1
X-Powered-By: ASP.NET
Date: Mon, 20 OCT 2008 11:25:56 GMT
Content-Type: text/html
Accept-Ranges: bytes
Last-Modified: Sat, 18 Oct 2008 15:05:44 GMT
ETag: “b66a667f948c92:8a5”
Content-Length: 426
```

Рис. 1.47. Пример ответа сервера на запрос клиента

В теле ответа может быть сформирован HTML-код разметки веб-страницы, например:

```
<html>
<body>
    <form action='http://localhost/Scripts/test.pl'>
        <p>Operand1: <input type='text' name='A'></p>
        <p>Operand2: <input type='text' name='B'></p>
        <p>Operation:<br>
            <select name='op'>
                <option value='+'>+</option>
                <option value='-'>-</option>
                <option value='*'>*</option>
                <option value='/'>/</option>
            <select>
        </p>
        <input type='submit' value='Вычислить'>
    </from>
</body>
</html>
```

Браузер отобразит этот документ согласно базовым правилам стилизации компонент разметки:

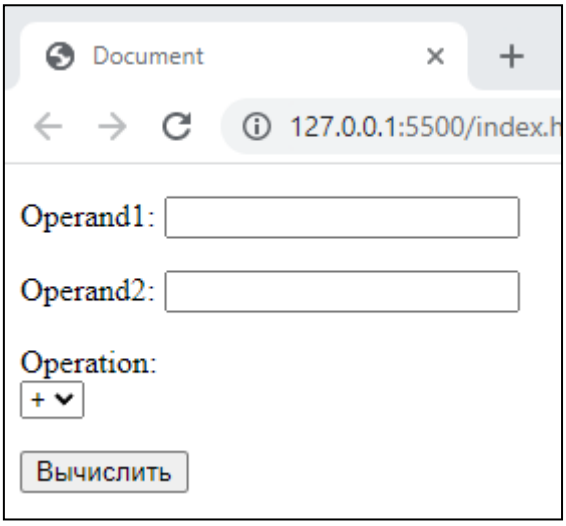


Рис. 1.48. Веб-страница, сформированная сервером в ответе клиенту

Протокол HTTPs

Протокол HTTP поддерживает передачу различных типов файлов. Однако он обладает существенным недостатком – данные HTTP-запроса передаются в открытом виде. Это связано с тем, что протокол HTTP не предусматривает шифрования данных.

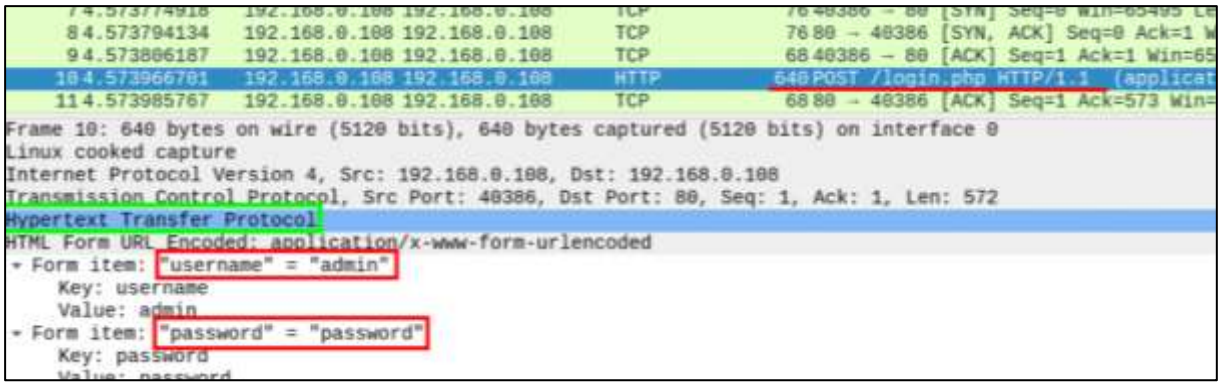


Рис. 1.49. Перехват HTTP-запроса: видны конфиденциальные данные

Для работы веб-сервисов, которые требуют защиты передаваемых данных в сети (почтовые сервисы, интернет-магазины, платежные системы, веб-сайты и т.д.) используется новый протокол – HTTPs.

Определение

HTTPs *(HyperText Transfer Protocol, secure) – расширение протокола HTTP, поддерживающее шифрование. Данные, передаваемые по протоколу HTTP, «упаковываются» в криптографический протокол SSL или TLS, чем обеспечивается защита этих данных.*

HTTPs шифрует данные перед отправкой их на транспортный уровень. По умолчанию он использует порт 443.

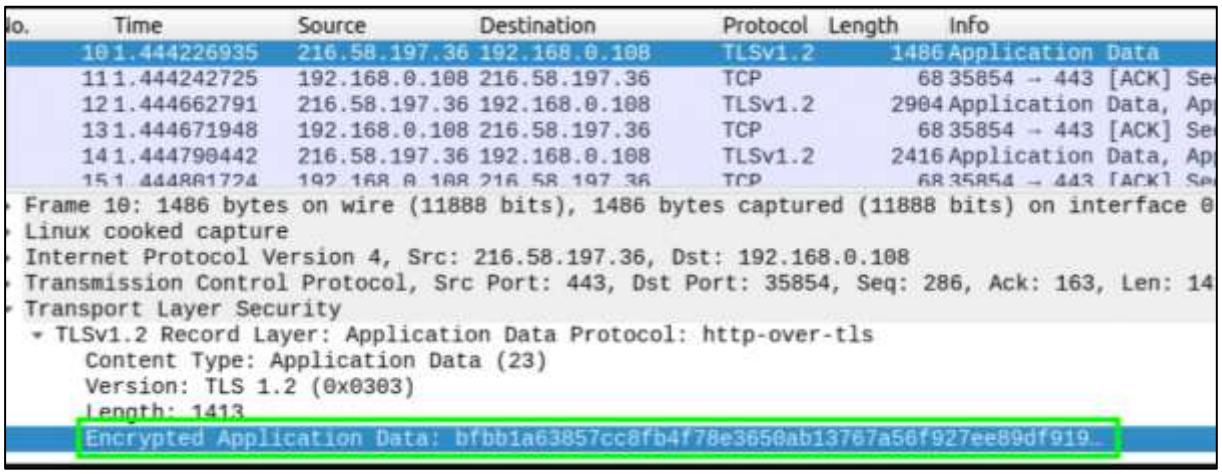


Рис. 1.50. Протокол HTTPs зашифровал данные

Перехват данных, передаваемых протоколом HTTPs уже не отображает конфиденциальной информации: она передается в едином зашифрованном потоке (рис. 1.50).

SSL-сертификаты

Определение

SSL-сертификат *(Secure Sockets Layer) – это цифровой сертификат, подтверждающий подлинность веб-сайта и позволяющий работать с зашифрованным соединением.*

SSL является протоколом защиты данных, который шифрует соединение между веб-сервером и веб-браузером. Он позволяет перенаправлять запросы с протокола HTTP на HTTPs.

Разработка протокола SSL велась в 90-х гг и была доступна в нескольких версиях, поскольку возникал ряд проблем безопасности. В 1999 году была предложена обновленная версия протокола – TLS (Transport Layer Security), используемая и в настоящее время. В силу исторических обстоятельств администраторы зачастую продолжают использовать аббревиатуру SSL для TSL-протоколов.

В настоящее время компании, организации, а также физические лица, размещающие веб-сайты и веб-сервисы в сети Интернет используют SSL-сертификаты, чтобы обеспечить конфиденциальность своих пользователей и защитить онлайн-транзакции от несанкционированного доступа.

Процесс работы SSL-сертификата состоит из следующих шагов:

1. Клиент (браузер) или сервер запрашивает веб-ресурс, защищенный SSL-соединением. Запрашивается его идентификация.
2. Веб-сервер отправляет клиенту копию SSL-сертификата.
3. Клиент проверяет достоверность SSL-сертификата. В случае подтверждения веб-серверу отправляется утвердительный ответ.
4. В случае успеха веб-сервер отправляет подтверждение с цифровой подписью и открывает сеанс зашифрованной передачи данных.

Узнать, использует ли веб-ресурс защищенное соединение можно по адресной строке: защищенный сайт помечается символом замка. Дополнительную информацию о сертификате можно найти в меню (рис. 1.51).

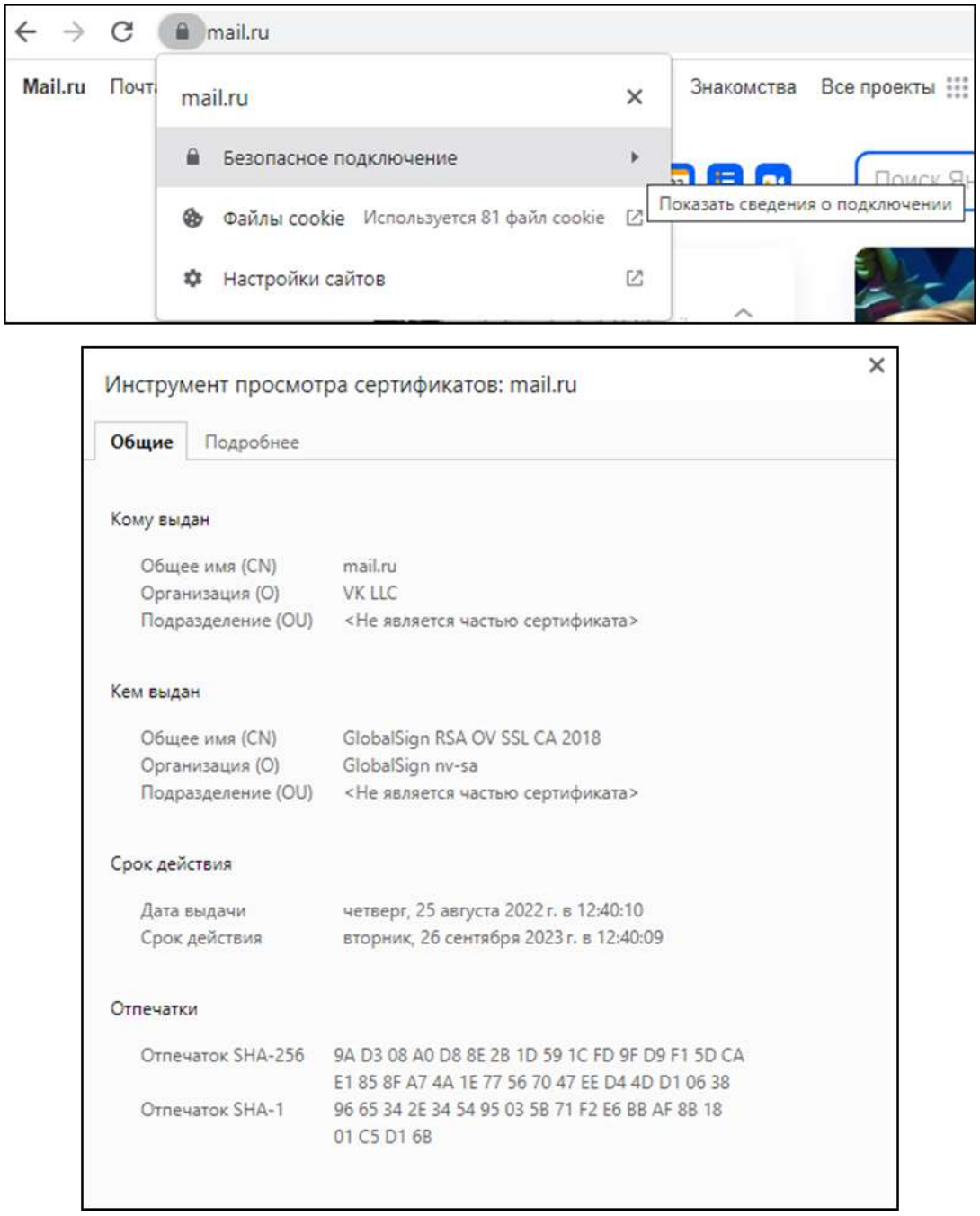


Рис. 1.51. Информация о SSL-сертификате на примере веб-ресурса Mail.ru

Вопросы для самопроверки

1. Для чего используются службы сети Интернет? Перечислите и опишите некоторые из них.
2. Опишите сущность технологии гипертекста. Почему она получили большое развитие и популярность?
3. Что представляют собой идентификаторы URI, URL, URN и в чем их отличие?
4. Почему URN идентификация считается более универсальной по сравнению с URL? В чем заключаются проблемы перехода на URN?
5. Какую роль играют протоколы HTTP и HTTPs в передаче данных?
6. Опишите структуру HTTP-запроса клиента.
7. Опишите структуру HTTP-ответа сервера.

Практикум

Задание 1

1. Зайдите в любой почтовый онлайн-сервис, где зарегистрирован ваш почтовый ящик. Далее изложение на примере Mail.ru.
2. Перейдите в раздел *Настройки*.
3. Осуществите привязку резервной почты.
4. На свое усмотрение дополнительно привяжите ящик к телефонному номеру (рис. 1.52). Двухфакторная аутентификация при входе с неизвестного устройства дополнительно будет отправляться SMS с кодом подтверждения (рис. 1.53).
5. В разделе *Безопасность* обновите пароль до наивысшего уровня надежности («зеленого» уровня). В пароль предусмотреть заглавные и малые буквы, цифры, спецсимволы (рис. 1.54). Осуществите выход из аккаунта и повторную авторизацию.

6. Просмотрите *Историю действий* за последний месяц. Убедитесь, что в логах отмечены только ваши устройства, с которых осуществлялся вход в сервис (рис. 1.55).
7. В разделе *Общие* установите желаемое имя отправителя и подпись.

Задание 2

1. Осуществите настройку почтового клиента MS Outlook для своей версии Office на примере синхронизации с почтовым ящиком на сайте Mail.ru или Yandex.ru.
2. Для начала изучите материал по следующим ссылкам:
   a. Настройка почты в Outlook: help.reg.ru.
   b. Настройка почтовых программ для Mail.ru: help.mail.ru.

Задание 3

1. Проанализируйте защищенность сайтов, которыми вы чаще всего пользуетесь (почтовые сервисы, социальные сети, сайт вуза, развлекательные порталы).
2. Какие из них используют SSL-сертификаты? Проверьте сроки действия сертификатов.

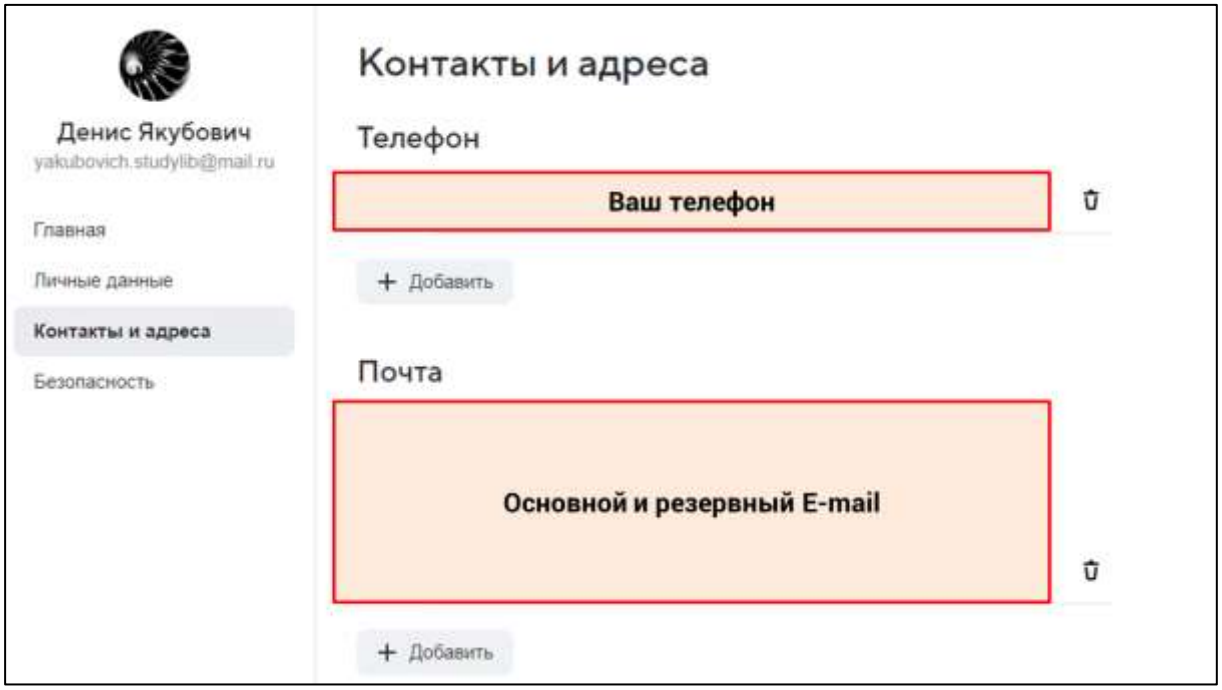


Рис. 1.52. Привязка телефона и резервной почты

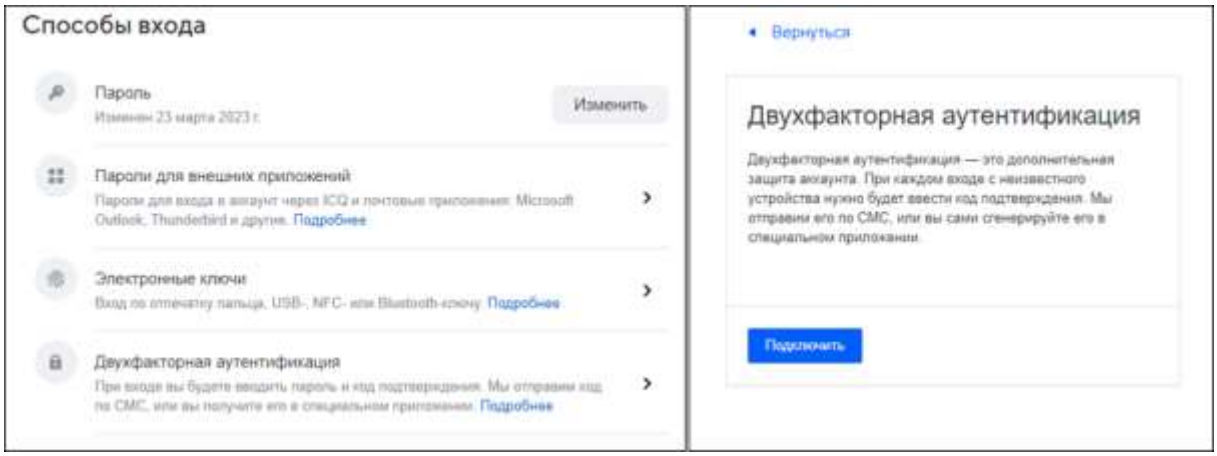


Рис. 1.53. Подключение двухфакторной аутентификации

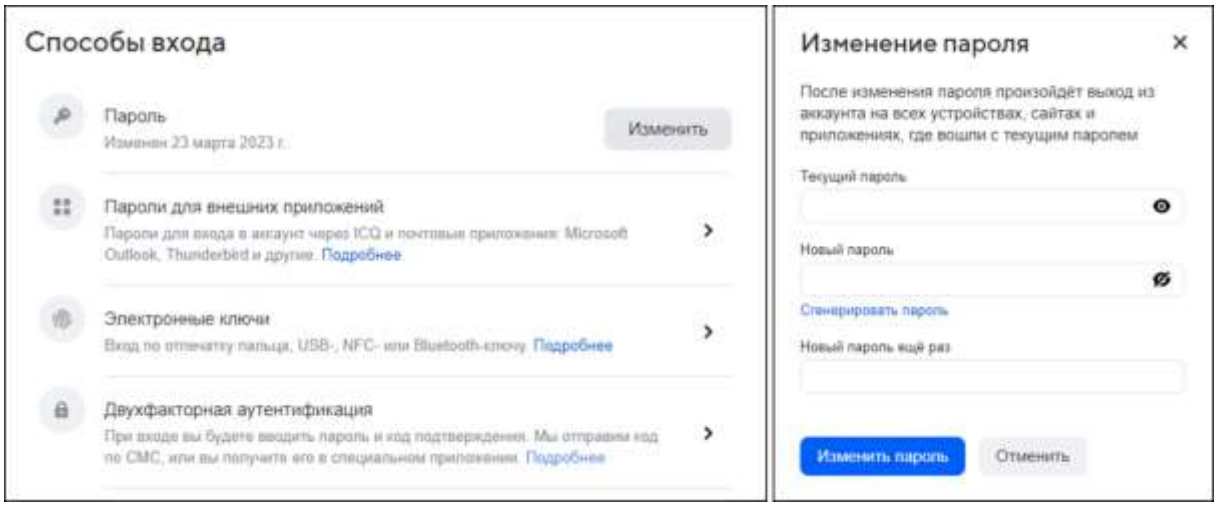


Рис. 1.54. Смена пароля на более надежный

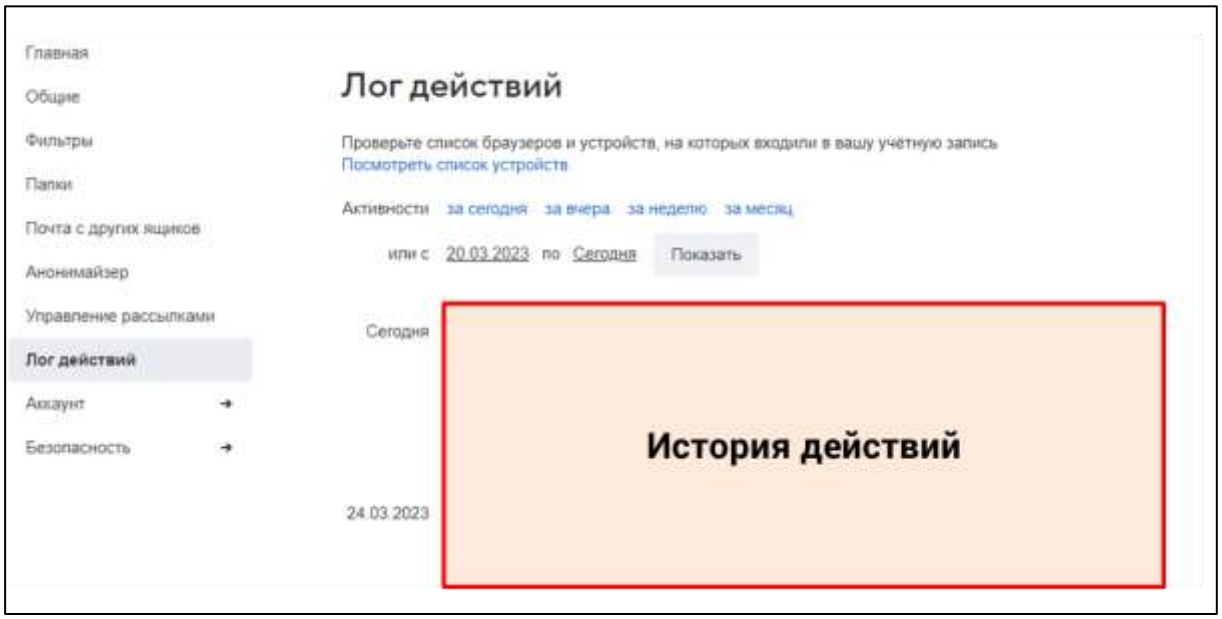


Рис. 1.55. История входов в почтовый сервис

1.4. Технологии разработки программного обеспечения

1.4.1. Веб-разработка и ее этапы

Понятие и область применения

Определение

Веб-разработка *– это процесс создания веб-сайта или веб-приложения.*

Веб-разработка является одним из наиболее привлекательных и активно развивающихся направлений современной IT-индустрии. Веб-разработка нацелена на создание веб-приложений и веб-сайтов, в процессе реализации которых разрабатывается дизайн и осуществляется верстка веб-страниц, программируется обработка данных на стороне клиента и сервера, конфигурируется сетевое оборудование.

Зарождение многих технологий веб-разработки началось в 90-е годы. Появившаяся в то время технология HTML задала большой потенциал развития и универсализации веб-документов, однако она была существенно ограничена в возможностях. До появления популярных языков веб-программирования основной код писался на разных языках, в частности – C, C++, Perl, которые были мало приспособлены под сетевые технологии. Это сказывалось и на дизайн интерфейса сайтов: он был мало интерактивным и невзрачным.

Однако разработчики видели в сети Интернет будущее и активно развивали технологии, которые позволяли упрощать процесс разработки и интегрировать его в массовую культуру. Большой вклад в развитие внес язык JavaScript, технологии CSS и Flash. Стали появляться программы-браузеры с привычным для современного пользователя дизайном. Позже к обработке серверных данных подключили язык PHP, что позволило организовать более сложную архитектуру веб-сервисов и добавить в них динамику.

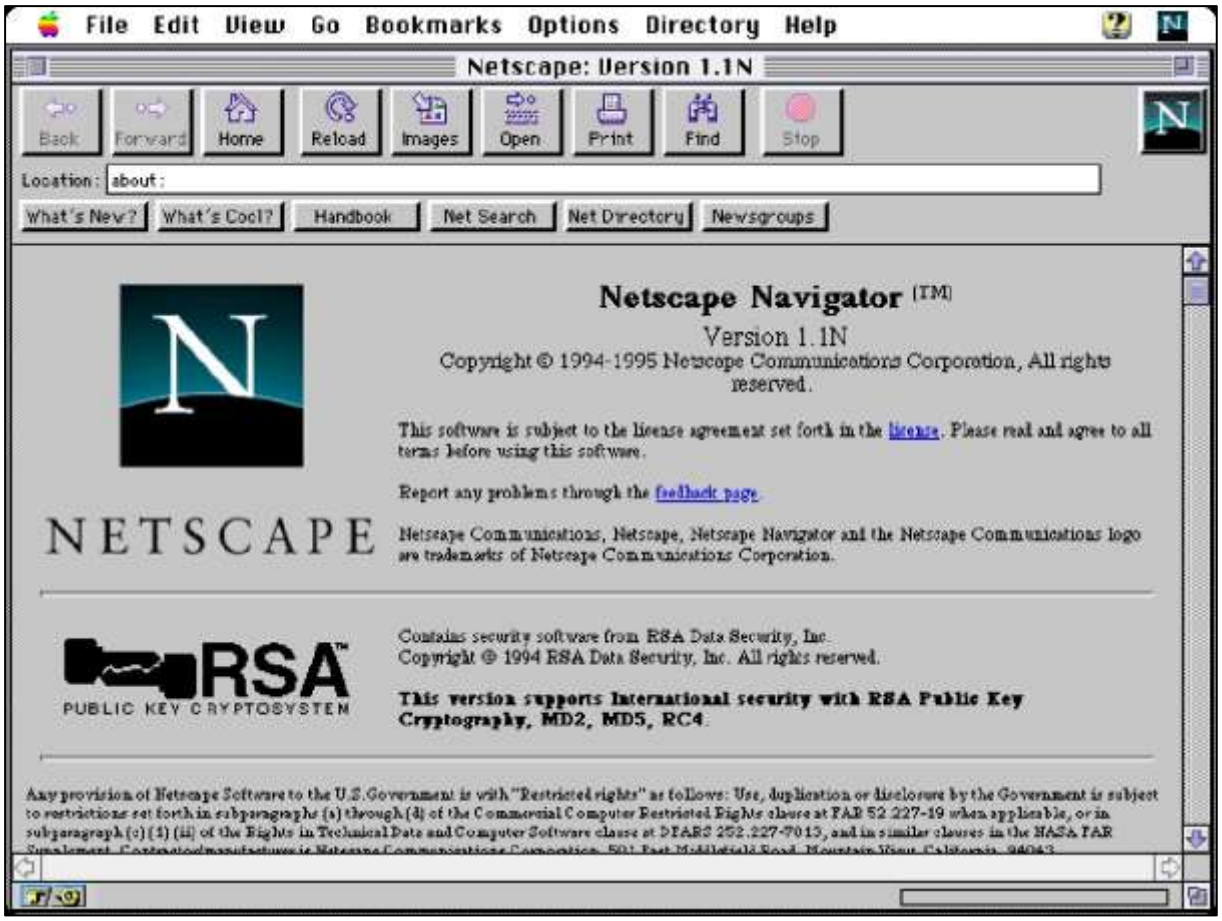


Рис. 1.56. Одна из первых версий браузера Mosaic

Сайтостроение быстро стало востребованным в бизнесе. Крупные компании начали продвигать свои товары и услуги через интернет-площадки. Не меньшую популярность веб-разработка стала получать и среди обычных пользователей, которые создавали личные веб-странички для самых разнообразных целей.

Основные этапы веб-разработки

Разработка веб-сайтов и веб-приложений может требовать обширный комплекс мероприятий.

В общем виде веб-разработка предполагает следующие основные этапы:

1. планирование дизайна веб-проекта;
2. вёрстка веб-страниц (интерфейса);
3. программирование функционала веб-проекта на стороне клиента и сервера;
4. конфигурирование веб-сервера.

Рассмотрим более развернуто жизненный цикл веб-разработки на примере создания веб-сайтов.

Этап 1. Сбор и анализ данных

Одной из важнейших и первоначальных задач веб-разработки является предварительное исследование целевой аудитории и сбор информации об аналогичных проектах. Успешность решения этой задачи определяет эффективность реализации последующих этапов разработки.

Разработчики должны иметь четкое представление по следующим вопросам:

- какую цель и задачи имеет искомый проект;
- для какой целевой аудитории ориентирован веб-ресурс;
- каким функционалом должен обладать веб-ресурс и чем выгодно отличаться от ближайших конкурентов.

Детальный анализ собранных данных позволяет определить наилучшую стратегию создания и дальнейшего развития проекта. Веб-ресурсы для продаж, поиска и анализа информации, продвижения товаров и услуг, обучения требуют разный функционал, поэтому затраты на их разработку должны быть оптимизированы.

Этап 2. Техническое задание

Для формализации планируемых работ создается *техническое задание* и ведется *проектная документация*. На основе технического задания составляется смета, а само техническое задание прикладывается к договору.

Техническое задание включает:

- описание структуры веб-сайта (в виде схем);
- эскизы или пожелания к дизайну интерфейса;
- полный перечень функциональных возможностей;
- описание особенностей дизайна;
- технологических особенностей и возможности ресурса;
- порядок и инструменты для дальнейшего администрирования ресурса.

Техническое задание может быть предложено как клиентом, так и студией (на основе пожеланий и правок клиента). На данном этапе формулируется общая концепция проекта, планируется его функционал, особенности дизайна и оформления веб-страниц.

Чтобы контролировать временные рамки разработки проекта, зачастую определяют график работ на основе диаграмм Гантта.

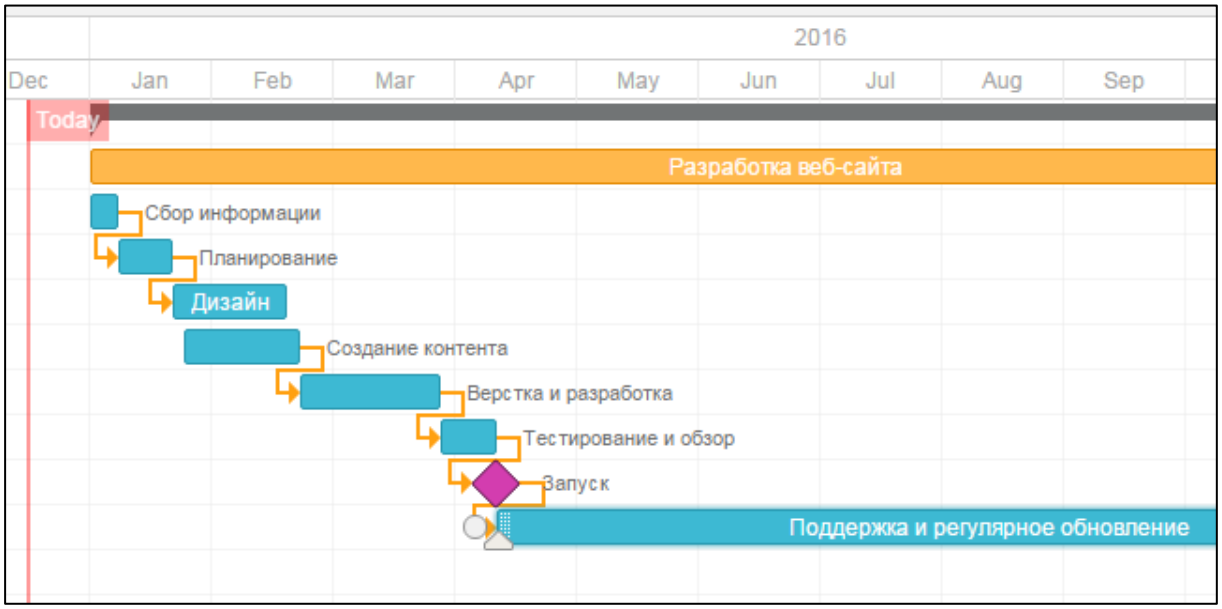


Рис. 1.57. Диаграмма Гантта для запланированного графика веб-разработки

Этап 2. Проектирование и прототипирование

На этом этапе заказчик и разработчики получают представление, каким в целом будет проект.

Определить точную структуру и взаимосвязь с другими элементами будущего веб-ресурса помогает карта сайта. Карта сайта позволяет заранее оптимизировать навигацию по ресурсу, сделать ее удобной и простой в использовании. При этом карта сайта не связана с дизайном.

На базе сформированного технического задания и карты сайта проектировщик рисует схематичные прототипы веб-страниц. *Макет* будущего веб-сайта позволяет клиенту получить более развернутую информацию о планируемом интерфейсе. Он не содержит элементов дизайна, а только описывает элементы и их компоновку на веб-страницах.

Для создания макетов можно использовать различные веб-сервисы, например, Moqups.

Также на этапе проектирования разработчики определят, какой стек технологий потребуется в разработке.

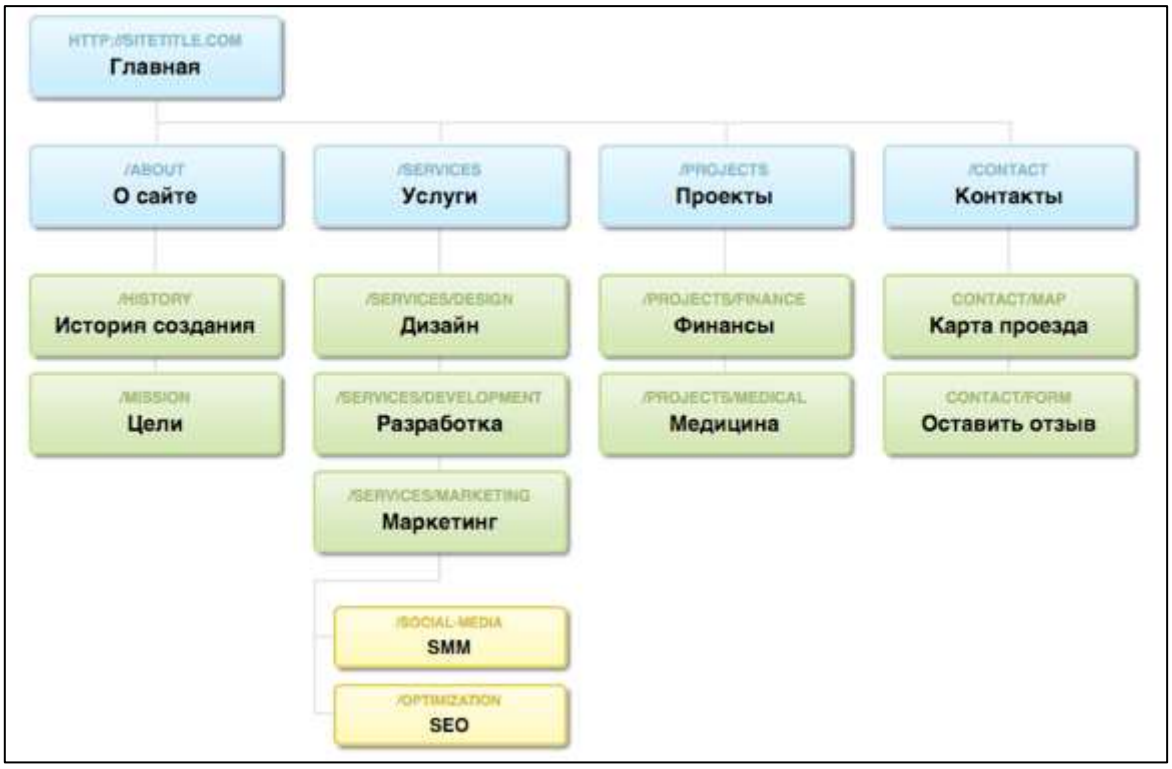


Рис. 1.58. Пример реализации карты сайта в виде схемы

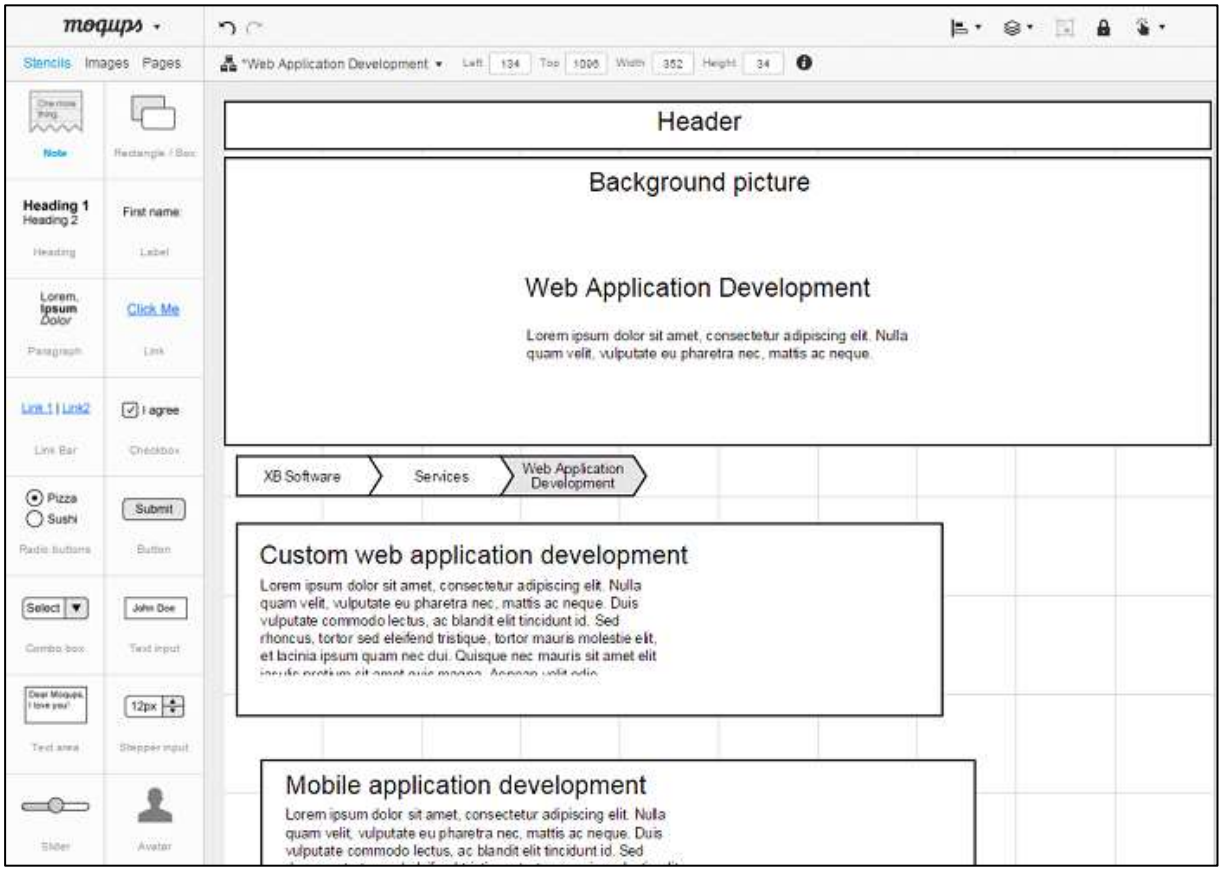


Рис. 1.59. Схематичный макет сайта, выполненный в сервисе Moqups

Этап 3. Разработка дизайна

Согласно утвержденных клиентом прототипу и макету дизайнер разрабатывает внешний вид проекта. Обычно в начале прорабатывается дизайн-концепция главной страницы сайта, которая отражает общую стилистику в соответствии с фирменным стилем компании, в частности – логотипом.

Шаблон страницы, в отличие от макета, визуализирует структуру страницы, ее содержимое и функционал. В шаблоне прорабатывается цветовая гамма, логотипы и изображения, что дает точное представление о визуальном оформлении веб-сайта.

Шаблоны веб-сайта можно разрабатывать в графических редакторах, например, Adobe Photoshop, Illustrator, Canva, Figma.

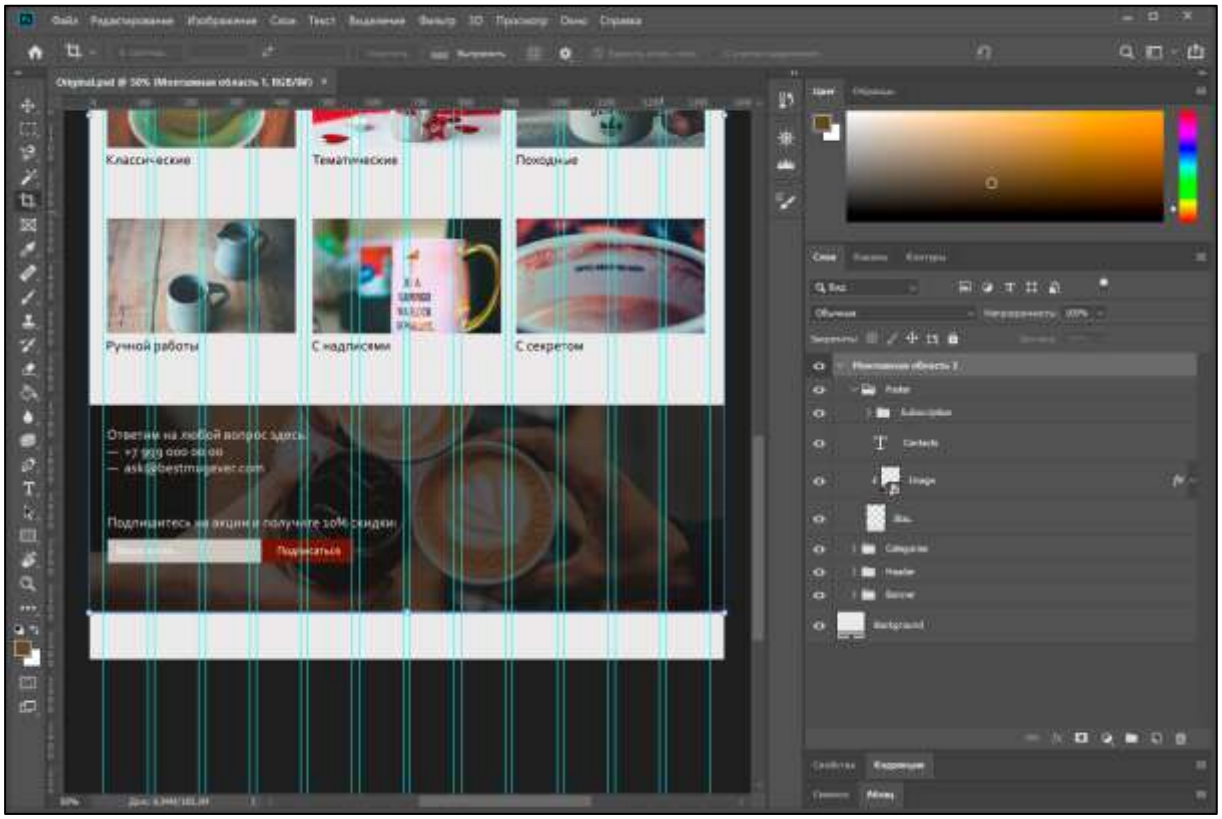


Рис. 1.60. Использование PhotoShop в разработке шаблона сайта

После утверждения шаблона заказчиком разработчики приступают к этапу верстки веб страниц.

Этап 4. Верстка

Этап верстки предполагает перевод визуального дизайна сайта в форму электронных веб-страниц. Основными технологиями здесь яв-

ляются язык разметки HTML и каскадные таблицы стилей CSS. В настоящее время также активно используются различные фреймворки, которые позволяют упрощать верстку больших проектов.

Верстка является технологическим процессом и привержена ряду правил:

- структурированность и SEO-оптимизация кода;
- валидность (корректность) разметки;
- кроссбраузерность (стабильное отображения в разных браузерах);
- адаптивность дизайна (корректная работа на разных устройствах, в т.ч. мобильных).

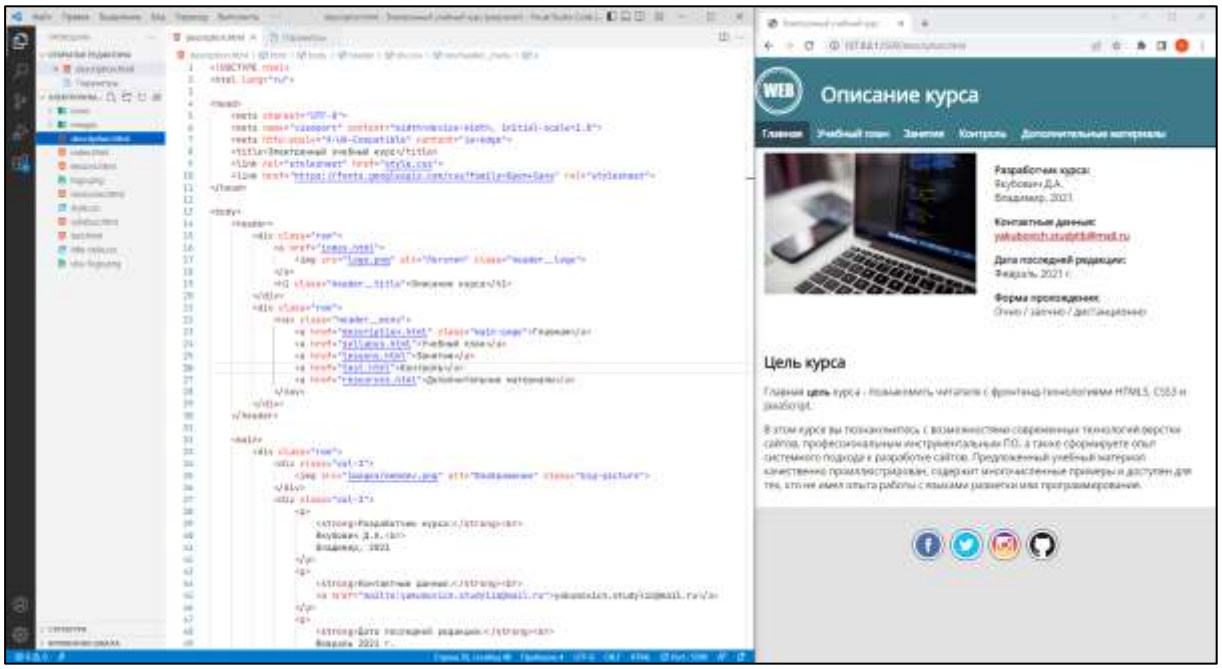


Рис. 1.61. Верстка веб-страниц

Этап 5. Программирование функционала и интеграция с CMS

Размеченные на страницах управляющие элементы должны быть запрограммированы, чтобы обеспечить обработку данных и интерактивный режим работы с сайтом. Здесь могут использоваться возможности как специальных скриптовых, так и «классических» языков программирования.

На стороне клиента решаются вопросы обеспечения динамичности поведения веб-страниц и подготовке данных на отправку северу (например, введенных через формы). На стороне сервера программи-

рование предполагает работу с базами данных, отправкой и обработкой запросов.

К этому этапу также можно отнести и интеграцию внешними системами, например – CMS. Системы управления сайтом предназначены для администрирования веб-ресурса и управления его контентом. Разработчику важно настроить систему под реализованный дизайн сайта и предоставить заказчику удобный интерфейс управления содержимым, исключив необходимость вмешательства клиента в верстку и программный функционал.

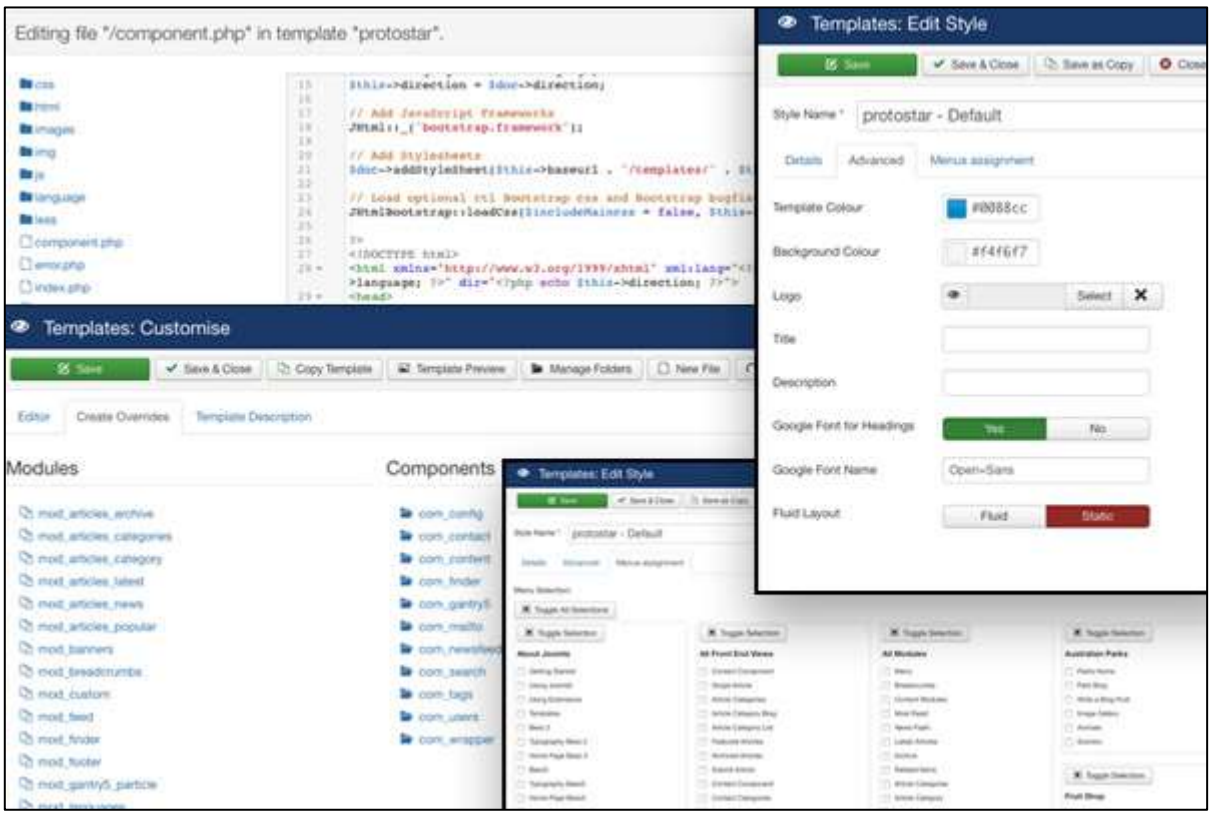


Рис. 1.62. Управление содержимым в CMS Joomla

Этап 6. Наполнение сайта содержимым

Важным этапом в создании сайта является подготовка мультимедийного контента. Уже на этапе разработки дизайна необходимо проработать элементы оформления: логотипы, иконки, анимацию блоков и т.д.

По желанию заказчика наполнение содержимым может осуществляться студией, либо самим клиентом посредством настроенной под его потребности CMS.

Для успешного продвижения сайта могут потребоваться услуги профессионального копирайтера, способного грамотно выстроить текст с учетом различных аспектов восприятия информации.

Этап 7. SEO-оптимизация

SEO-оптимизация (Search Engine Optimization, технология оптимизации поиска) направлена на оптимизацию текстовых элементов веб-страницы (заголовков, ключевых слов, описания). Главная задача SEO – повысить релевантность веб-сайта при выдаче в крупнейших поисковых системах.

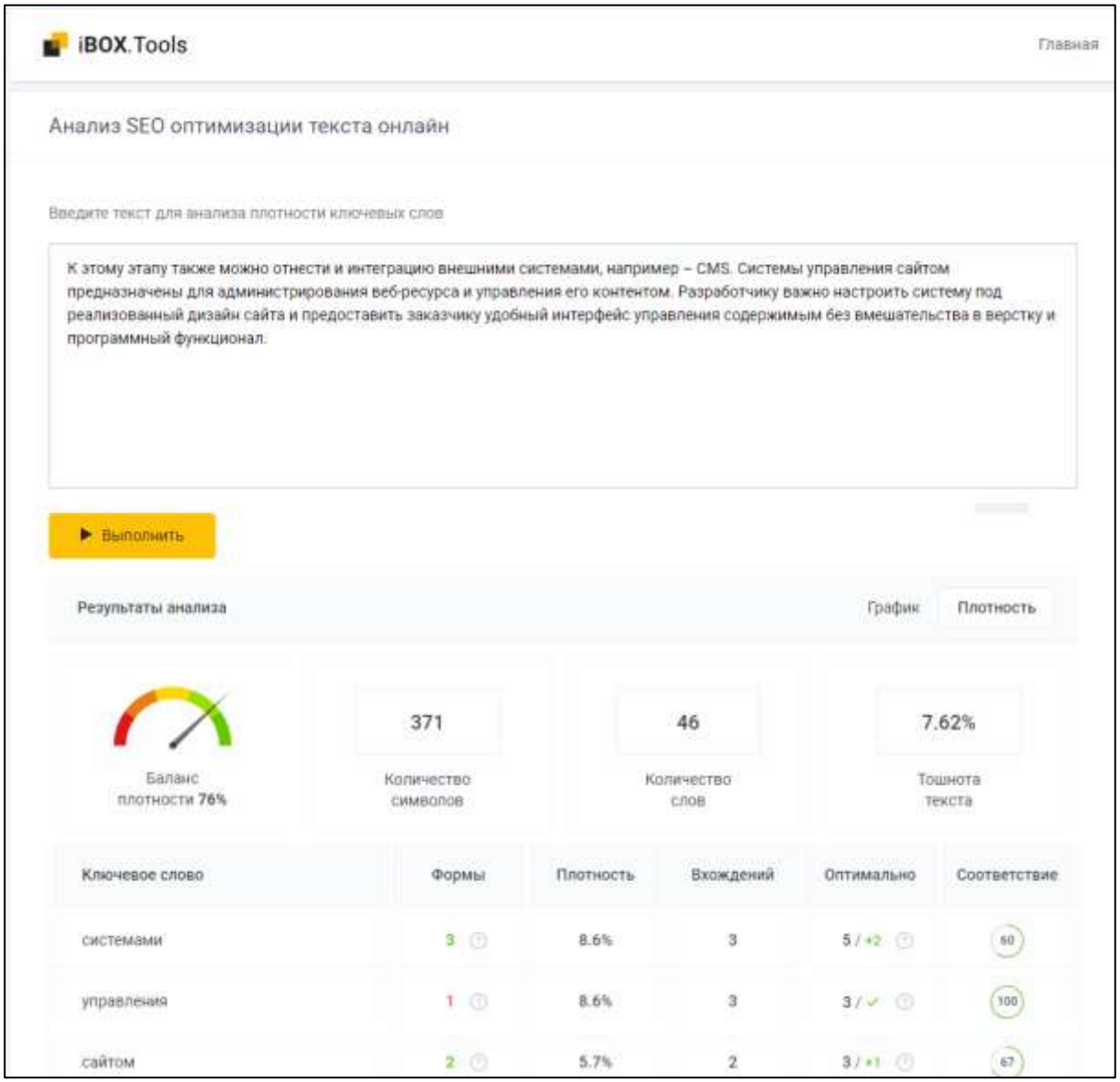


Рис. 1.63. Анализ SEO-оптимальности текста на примере iBOX.Tools

Основными составляющими SEO являются:

- валидность разметки;
- копирайтинг текстов;
- управление тегами;
- учет трендов и популярных запросов пользователей.

С другой стороны, такие компании, как Yandex и Google не разглашают собственные алгоритмы ранжирования сайтов, поэтому зачастую результаты SEO не всегда приводят к должному результату.

Этап 8. Тестирование и запуск

Перед релизом осуществляется тестирование работоспособности сайта в целом и его отдельных компонент. Тесты оценивают безотказность основных функций, время загрузки страниц, потребляемый трафик, удобство поиска и т.д. В работе используются разные методологии тестирования.

После успешного поведения тестов веб-сайт загружается на сервер выбранной хостинг компании и привязывается к временно арендованному домену. Дополнительно проводится техническая и SEO-оптимизация, подключаются компоненты Яндекс.Метрика и Google Analytics, осуществляется нацеливание проекта. Заказчик получает полный доступ к управлению веб-сайтом и его документации.

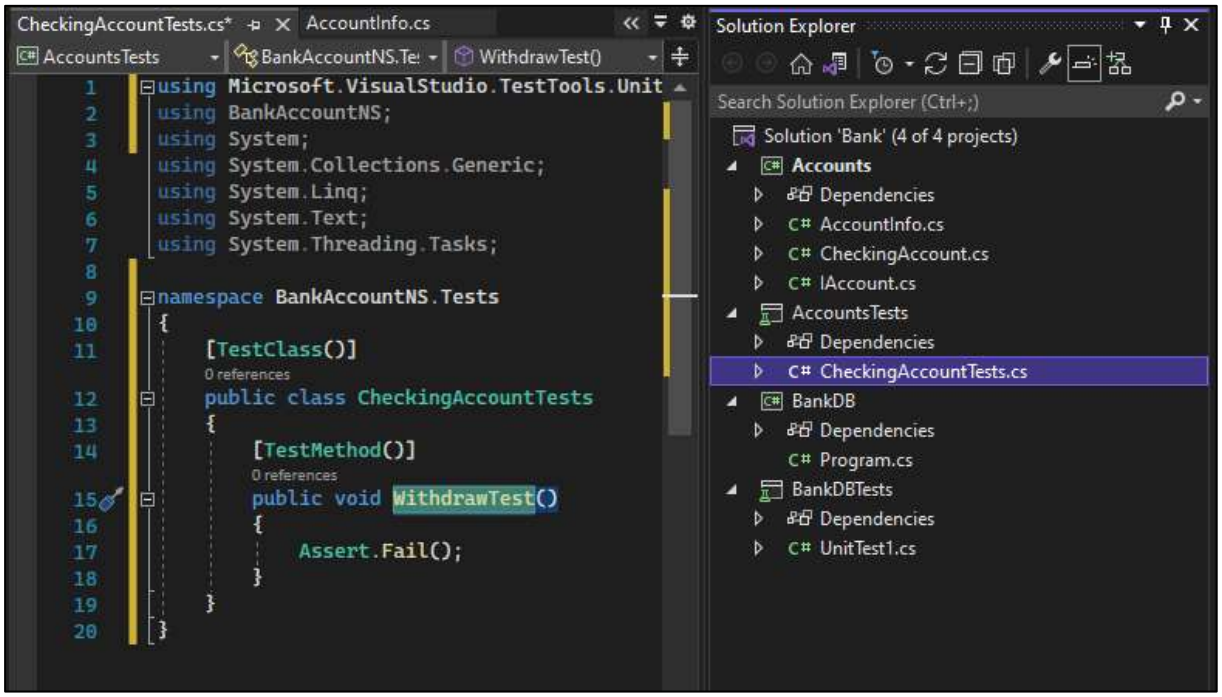


Рис. 1.64. Модульное тестирование на примере среды Visual Studio

Этап 9. Поддержка и сопровождение

Любой успешный сайт со временем будет терять свою актуальность, если его владельцы перестанут обновлять информацию. Для поддержания конкурентоспособности важно проводить технические работы и вносить дополнительный функционал.

Модернизация сайта может включать масштабирование сайта, расширение доступных услуг, интеграция сторонних проектов. Отдельное внимание уделяется рекламному продвижению, развитию репутации бренда в сети.

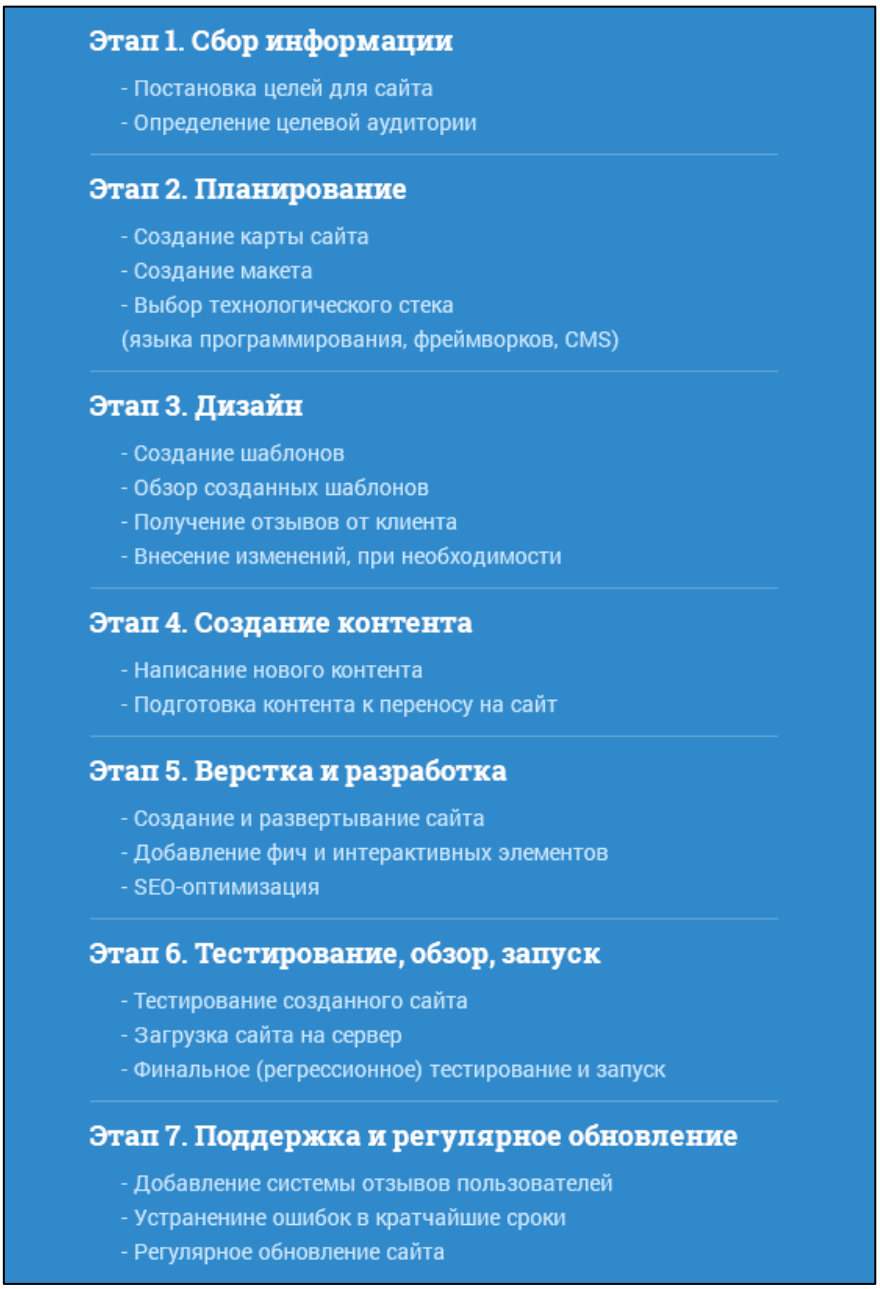


Рис. 1.65. Чек-лист разработки веб-сайта

1.4.2. Разновидности веб-разработки

Направления современной веб-разработки

В современной веб-разработке выделяют три основных направления: это frontend, backend и fullstack-разработка. Каждое направление требует от веб-разработчика владение определенным стеком технологий и средств разработки.

С другой стороны границы между frontend, backend и fullstack-технологиями весьма формальные. Многие языки программирования и фреймворки могут использоваться реализации разных задач, поэтому веб-разработчик должен иметь представление о смежных технологиях и быть способным к их интеграции в проекте.

Frontend

Назначение

Frontend нацелен на разработку внешнего вида веб-сайта. Frontend имеет достаточно низкий порог вхождения и лоялен к начинающим веб-разработчикам.

Однако спектр деятельности frontend-разработчика весьма широк: его задача не только разработать удобный и многофункциональный веб-интерфейс сайта или приложения, но и реализовать его программное функционирование на стороне клиента (браузера) и обеспечить интеграцию ресурса с разными типами устройств и платформ. В отличие от веб-дизайнера frontend-разработчик отвечает за техническое обеспечение работы сайта.

Обязанности разработчика

Frontend разработка предполагает решение следующих задач:

- Верстка каркаса сайта с помощью языка разметки HTML и его оформление с помощью технологии CSS согласно разработанному веб-дизайнером макету.
- Программирование логики интерфейса пользователя на языке JavaScript и одним из выбранных фреймворков (например, React, Vue.js, Angular или др.). JavaScript и связанные с ним фреймворки позволяют реализовать анимацию и интерактивную работу с элементами сайта.

- Обеспечение адаптивности дизайна веб-приложения или веб-сайта.
- Подготовка проекта для дальнейшего связывания с backend-технологиями, т.е. обработки данных на стороне сервера.

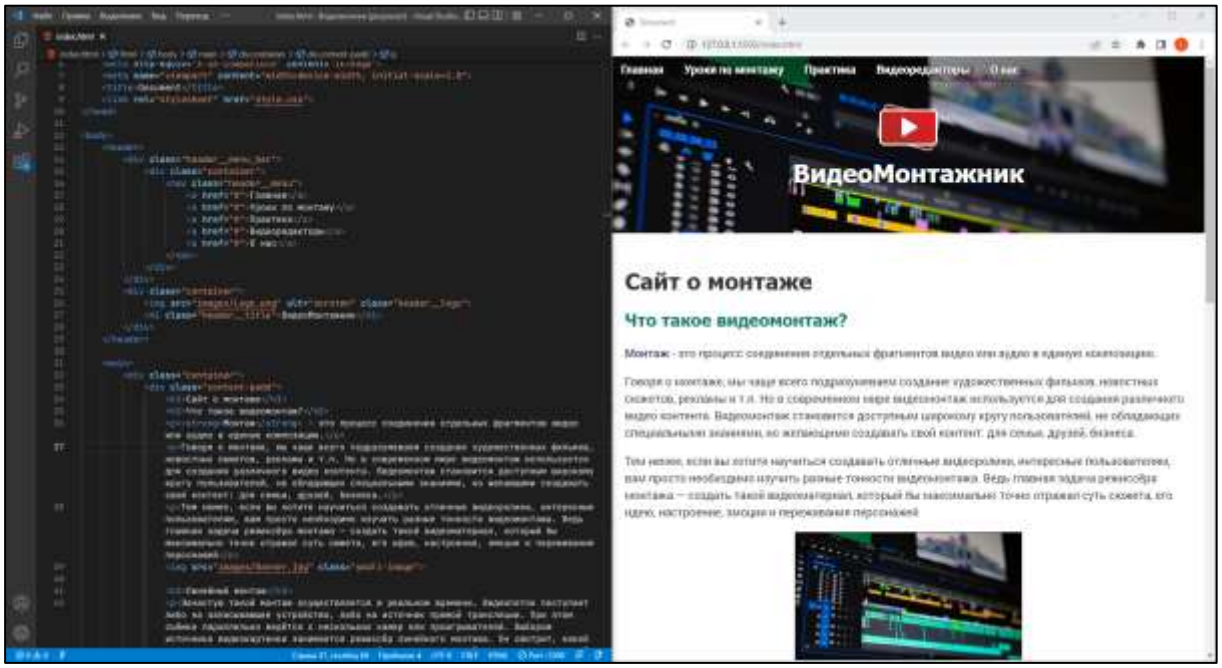


Рис. 1.66. HTML-разметка как часть frontend-разработки

Backend

Назначение

Backend-разработка отвечает за построение внутренней логики работы приложений и сайтов, а также обеспечение обработки данных на стороне сервера. Среди часто реализуемых задач выделяют аутентификацию, разграничение прав доступа, обработка действий пользователя и данных, введенных в формы, интеграция с другими приложениями и ОС, оптимизация скорости обработки запросов.

В отличие от frontend-разработки, где упор сделан на визуальную часть интерфейса, backend-технологии почти всегда работают в неявной для пользователя форме. Задачей разработчика является автоматизация операций передачи данных между клиентом и сервером, а также защита потока данных от возможной подмены в процессе отправки запросов клиента и ответа сервера.

Backend-разработка находит применение не только в веб-технологиях. Знания backend-разработчика востребованы в различных отраслях:

- построении финансовых сервисов;
- программировании промышленного оборудования и умных домов;
- автоматизированных систем в работе транспортных средств.

Современный backend-разработчик должен владеть множеством языков программирования, уметь работать с СУБД, строить сложные и эффективные алгоритмы обработки данных, применять фреймворки и паттерны программирования.

Среди наиболее важных компонент backend-разработки выделяют три:

- базы данных, хранящие большие объемы структурированной информации и оптимизированные определенными типами связи между ними.
- серверы, отвечающие за обработку запросов клиента и хранящие данные в базах данных;
- приложения, которые предоставляют пользователю дополнительный интерфейс взаимодействия с системой или веб-ресурсом.

Обязанности разработчика

- Frontend-разработчик проектирует архитектуру веб-ресурса и формирует ядро веб-сайта.
- Создание собственной библиотеки или платформы функций и классов, обеспечивающей функционирование системы.
- Построение алгоритмов обработки больших объемов данных и оптимизация затрачиваемых ресурсов сервера и интернет-траффика.
- Автоматизация рутинных задач и обеспечение безопасной работы с интерфейсом; резервное копирование данных из СУБД.
- Контроль версий сайта, работы СУБД и интеграции.

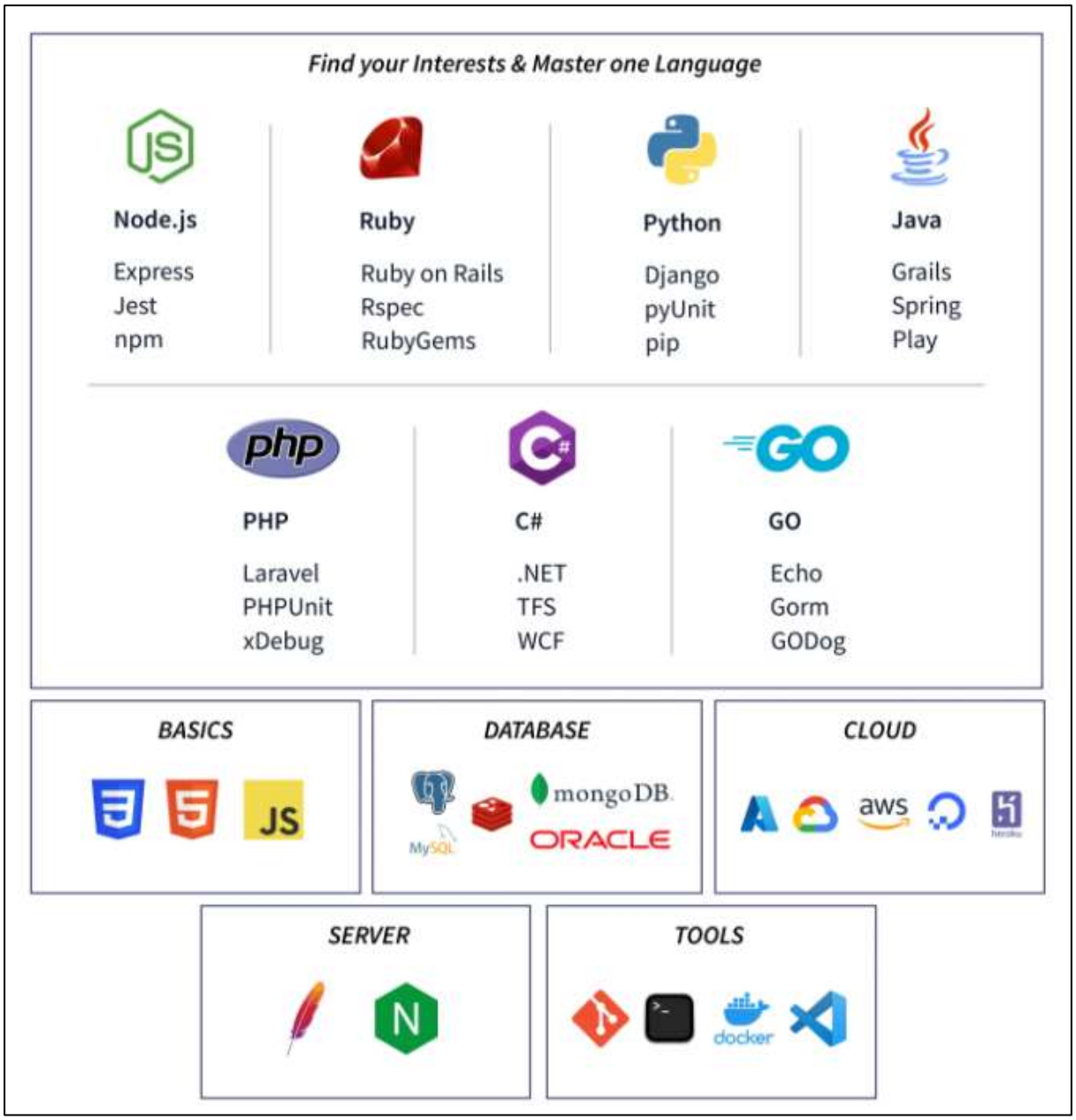


Рис. 1.67. Некоторые востребованные в backend разработке технологии

Fullstack

Назначение

Fullstack-разработка отражает способность веб-разработчика комплексно решать задачи frontend и backend-разработки в едином проекте, а также его администрировать. Несмотря на смешанность и многообразие технологий в настоящее время fullstack – это одно из наиболее прибыльных направлений на рынке веб-разработки.

Появление fullstack-разработки связано с потребностью компаний в универсальных разработчиках, способных заменить нескольких

узко профилированных специалистов. Fullstack-разработчик участвует во всех этапах разработки веб-сервиса, начиная от верстки пользовательского интерфейса и программировании логики, и заканчивая управлением серверной частью.

Компетенции fullstack-разработчика предполагают владение frontend и backend-разработкой, project-менеджмент и DevOps (методология взаимодействия IT-специалистов, направленная на поддержку нормального функционирования информационных систем).

Fullstack-разработка требует от специалиста большого опыта работы с разными веб-технологиями. Обычно это разработчики, которые имеют богатый опыт верстки веб-страниц, создания серверных и мобильных приложений. Fullstack-разработчик может реализовать любую задачу в проекте, сохранив дизайн на должном уровне и оптимизировав работу ресурса.

Fullstack дает возможность заранее определить эффективность использования стека технологий для заданного проекта. Однако широкий спектр задействования технологий ограничивает возможности разработчика реализовать частные задачи, требующие более детальной проработки (например, дизайна, расширенного функционала и т.д.). Это связано с тем, что у fullstack-разработчика обычно недостаточно времени, чтобы изучить различные возможности используемых технологий и погрузиться в решение проблемы.

Обязанности разработчика

- Осуществление адаптивной и кроссбраузерной верстки и дизайна с использованием HTML, CSS и какого-либо фреймворка (например, Bootstrap).
- Программирование на JavaScript, знание jQuery и хотя-бы одного из фреймворков (Angular, React, Vue), способность структурировать код.
- Программирование на стороне сервера, владение Node.js, Python, Ruby, PHP.
- Работа с базами данных и СУБД: SQL, NoSQL, умение масштабировать базы данных.
- Работа с системами контроля версий Git: GitHub, GitLab, SoursceForge.
- Хостинг и администрирование сервера.

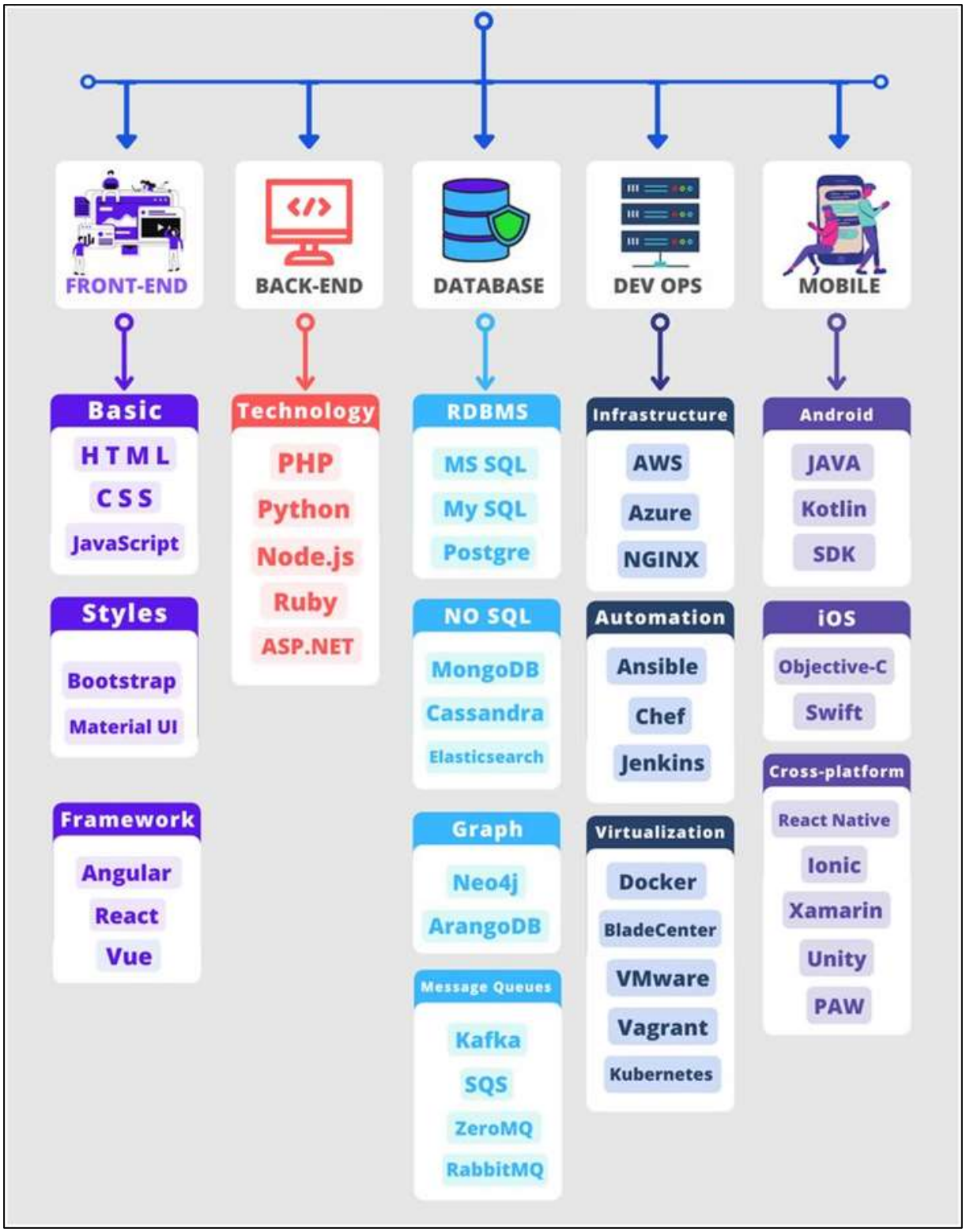


Рис. 1.68. Важные направления fullstack-разработки

Вопросы для самопроверки

1. В каких отраслях в настоящее время используется веб-разработка и с чем связана ее высокая популярность?
2. Опишите основные этапы веб-разработки.
3. Почему важно грамотно сформулировать техническое задание реализации будущего проекта?
4. Какие задачи решаются на этапах проектирования и прототипирования веб-сайта?
5. Чем отличается макет сайта от шаблона? Какие программы позволяют детально оформить дизайн будущей веб-страницы перед тем, как она будет размечена?
6. Что обеспечивает функциональную обработку данных веб-сайта. Чем могут помочь CMS?
7. Для каких целей осуществляется SEO-анализ и SEO-оптимизация?
8. Какими навыками должны обладать frontend-разработчики?
9. Какие задачи решают backend-разработчики.
10. В чем состоит сложность fullstack-разработки?

Практикум

Задание 1

1. Пусть требуется разработать веб-сайт, на котором должны размещаться учебно-методические материалы для студентов по языку программирования Python.
2. Заполните (в свободной форме) следующую таблицу:

№	Критерий	Описание
1.	Цель.	
2.	Целевая аудитория.	
3.	Структура основных разделов сайта.	
4.	Содержимое учебных материалов и курсов.	
5.	Преимущества перед аналогичными сайтами.	

Задание 2

1. Изучите рекомендации по структуре и содержимому технического задания (ТЗ) на разработку веб-сайтов.
2. Например, используя следующие источники:
   - «Как составить грамотное ТЗ на разработку сайта»: https://habr.com/ru/articles/593661/.
   - «Техническое задание на разработку сайта или интернет-магазина»: intitle.ru.
   - «Как составить ТЗ на разработку сайта. Чек-лист»: vc.ru/dev.
3. Ориентируясь на предложенные рекомендации, оформите в документе MS Word примерный вариант ТЗ на разработку сайта из первого задания.
4. В описание обязательно включить следующие подразделы:
   - целевая аудитория и цель реализации проекта;
   - требования к структуре (рекомендуется схемой или вложенным списком);
   - требования к дизайну;
   - требования к функционалу и пользовательским сценариям (т.е.);
   - поддерживаемые технологии;
   - требования к хостинг-провайдеру.

1.5. Инструменты разработчика

1.5.1. Фреймворки и API

Назначение API

Определение и роль API

Определение

API *(от англ.* ***A****pplication* ***P****rogramming* ***I****nterface,* ***программный интерфейс приложения****) – это совокупность способов и правил операций с данными и взаимодействия программ.*

API является в некотором смысле *контрактом*, который предоставляет программа разработчику. API определяет форму обращения к программе и то, как это следует делать. Процедура взаимодействия с API осуществляется посредством вызова определенных в нем констант, функций, классов, свойств и методов, структур.

Интерфейс современных веб-сервисов обычно предполагает наличие клиента, запрашивающего данные, и сервера, обрабатывающего запросы клиентов. API позволяет контролировать, какие данные и в какой форме могут быть переданы клиенту, чтобы не нарушить корректность работы и целостность системы.

В современной разработке ПО API получили широкое распространение:

- API определяет сигнатуру вызываемых функций и позволяет с ними работать по методу «черного ящика» (пользователю не нужно знать особенностей реализации функции, достаточно понимать, для чего она необходима и как ее корректно вызвать).
- API помогает управлять потоками данных между приложениями и настройками операционных систем. В частности, ОС Windows предоставляет нативный API, позволяющий программировать под систему оптимальным образом.

- В веб-технологиях разные сервисы предоставляют разработчикам собственный API и официальную документацию Это ускоряет процесс разработки и интеграции приложений с веб-сервисом.

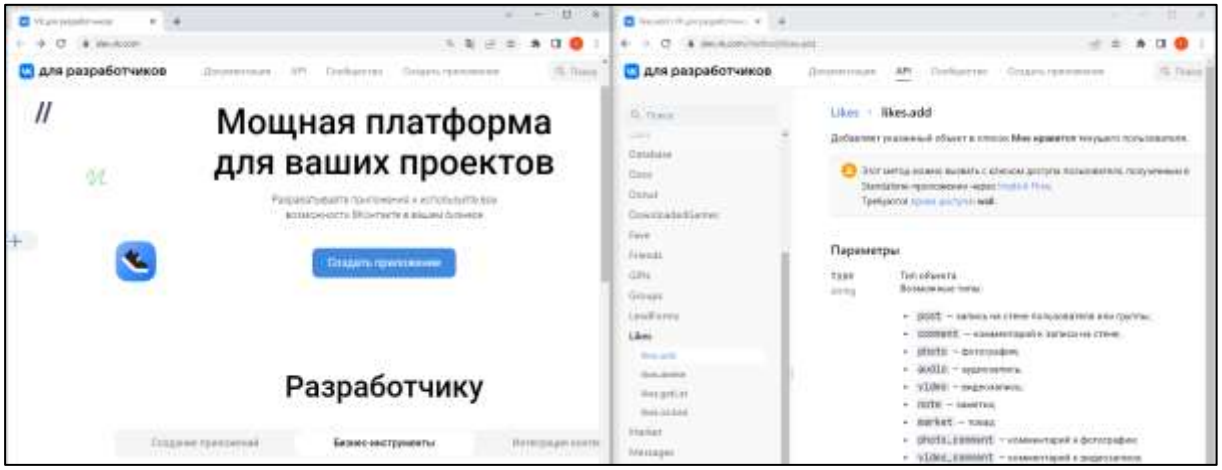


Рис. 1.69. Документация VK API для разработчиков приложений vk.com

API как набор функций

Контракт API может быть представлен в разных формах и описан различными структурами данных. Однако в любом случае он определяет три важных компоненты:

1. операцию, которую можно выполнить;
2. данные, передаваемые в качестве дополнительных параметров и влияющие на сценарий обработки;
3. данные, которые возвращаются на выходе (либо сообщение или код ошибки).

Иными словами, API в этом смысле можно рассматривать в качестве набора функций.

Приведем пример обращения к Web API. Пусть необходимо отправить на сервер HTTP-запрос и передать данные в формате JSON. Реализация на языке Python может выглядеть следующим образом:

```python
import requests
import json

# Помещаем в переменную SOURCE_URL адрес API
SOURCE_URL = "https://tts.api.cloud.yandex.net/speech/v1/tts:synthesize"

# Помещаем в словарь data данные для отправки в API Yandex.SpeechKit
data = {
    "text": "Фрагмент передаваемого текста",
    "lang": "ru-RU",
    "speed": 1,
    "voice": "filipp",
    "emotion": "good"
}

# Преобразуем данные в строку в формате JSON
json_string = json.dumps(data)

# Отправляем данные на сервер и получаем ответ
answer = requests.post(SOURCE_URL, json_string)
```

Рис. 1.70. Пример работы с API

API как интерфейс

С другой стороны, концепция API как работа с контрактами разграничивает взаимодействие двух функциональных систем, каждая из которых скрывает свои процессы от внешнего вмешательства. Принцип разграничения ответственности и черного ящика соответствует важнейшей концепции разработки ПО – использование архитектуры *интерфейсов*.

С точки зрения программирования интерфейс является контрактом, который определяет, что должно войти в структуру данных или процесс. Однако интерфейс не описывает саму реализацию функционала, он лишь указывает требования к структуре. Функционал определяется непосредственно в том элементе, который наследует этот интерфейс, т.е. обязуется соблюдать предложенную интерфейсом структуру.

Продемонстрируем интерфейсный подход к проектированию структуры кода на примере языка C#. Интерфейс `IAnimal` описывает некоторую общую абстрактную сущность, характеризующую животных (рис. 1.71). В описании интерфейса содержится метод `Voice()`.

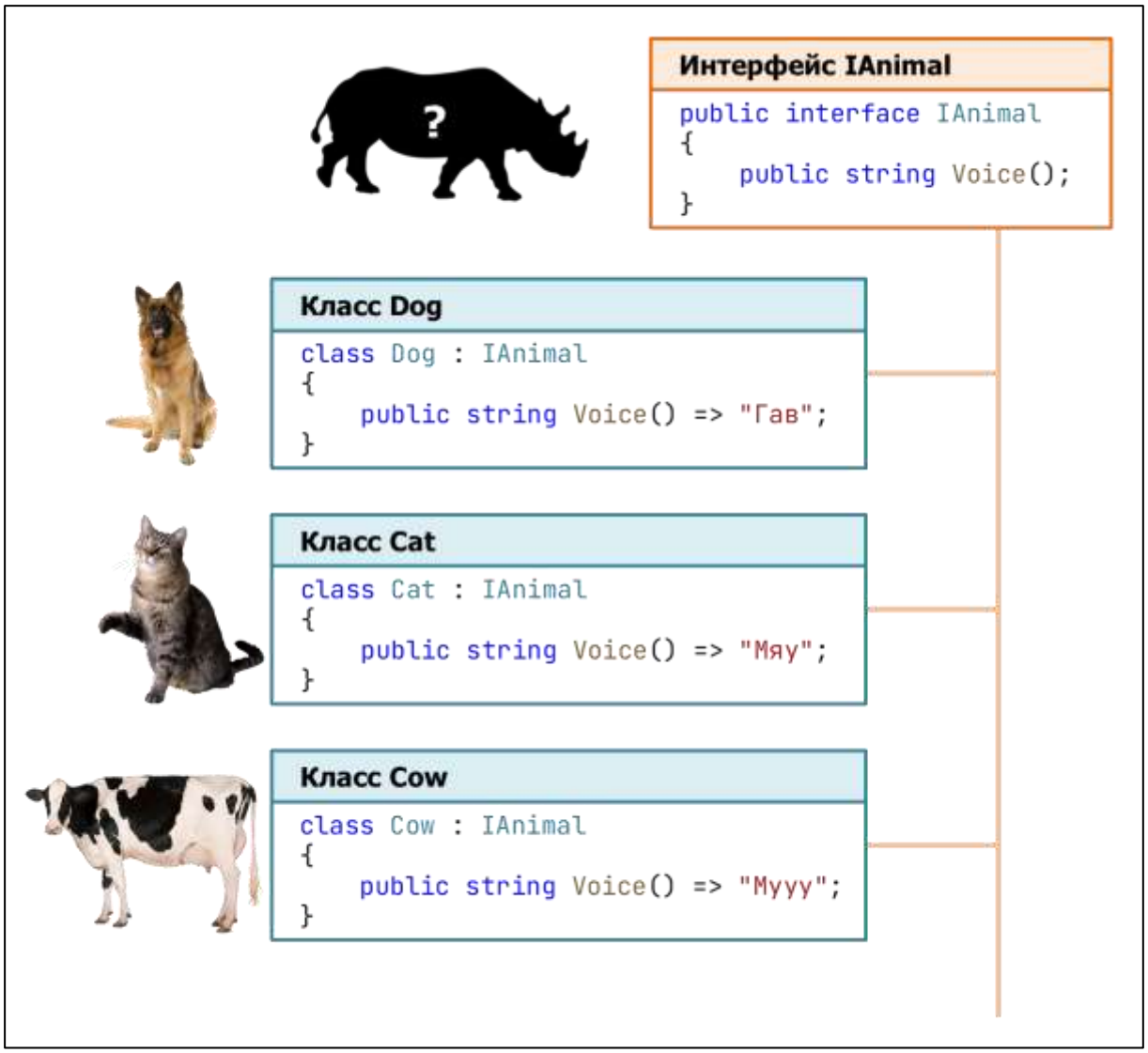


Рис. 1.71. Пример интерфейсного подхода в организации зависимостей при наследовании

Интерфейс `IAnimal` является контрактом, который только определяет сигнатуру метода `Voice()`, но не задает его реализации. Классы `Dog`, `Cat` и `Cow` описывают конкретные виды животных. Каждый из них наследует интерфейс `IAnimal` и задает собственную реализацию метода `Voice()`.

Таким образом интерфейс `IAnimal` определяет требования к структуре классов, а классы, наследующие интерфейс, гарантируют соблюдение указанной структуры.

Разумеется, в общем случае интерфейсы содержат описание множества различных компонент. Более того, классы могут наследовать несколько интерфейсов одновременно (применительно к C#). Это позволяет выстраивать иерархии зависимостей, расширяя возможности новых классов путем цепочек наследования.

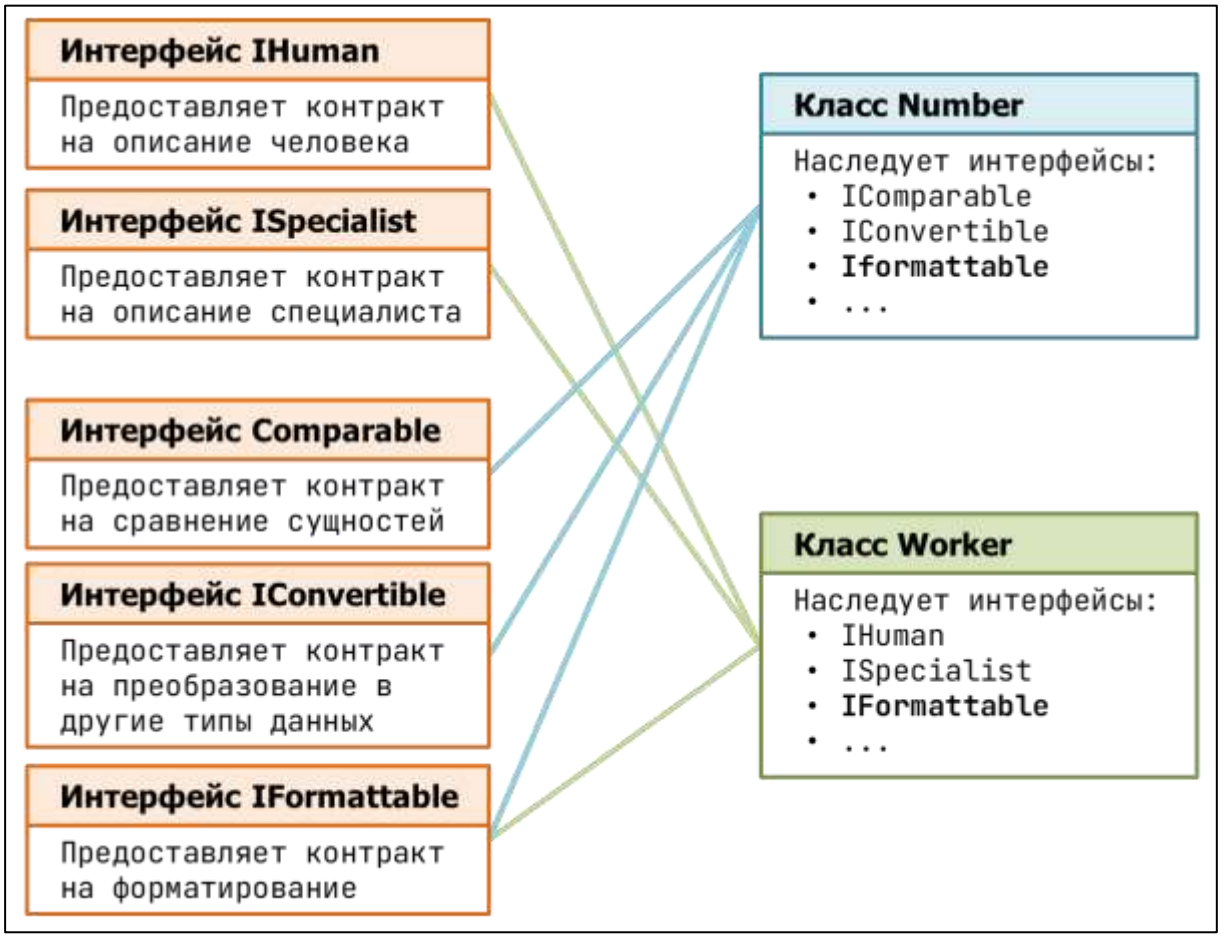


Рис. 1.72. Два разных по сущности класса Number (число) и Worker (работник) наследуют общий интерфейс IFormattable, позволяющий определить логику форматирования строкового представления каждого

Наследование на уровне интерфейсов позволяет разным по сущности и назначению классам иметь похожую структуру. И в отличие от наследования на уровне классов архитектура интерфейсов зачастую более удобна для масштабирования.

Описанный принцип работает и в API: вызовов какой-либо функции API не требует знать ее реализацию, достаточно лишь знать ее назначение и корректно к ней обратиться.

API предоставляет разработчикам ряд возможностей:

- позволяет использовать готовый набор функций;

- защищает данные от нежелательного вмешательства и некорректного использования;
- связывает системы, требующие защиты и шифрования данных (например, API платежных систем и банков);
- упрощают, ускоряют и удешевляют разработку.

Назначение фреймворка

Определение и роль фреймворков

Определение

Фре́ймворк *(от англ.* ***framework*** *— каркас, структура) – это программная платформа, задающая структуру программной системы и предназначенная для упрощения разработки компонентов большого программного проекта.*

Фреймворки используются при создании разных типов проектов: интернет-магазинов, настольных и веб-приложений, блогов, систем с графическим интерфейсом, приложений для мобильных устройств, игр, научных исследований. Фреймворк является в некотором смысле промежуточным решением между разработкой «с нуля» средствами языка программирования и использованием системы управления контентом (CMS). Фреймворк реализует некоторый базовый функционал, однако он имеет четкую архитектуру, удобную для масштабирования проекта и расширения пользовательским кодом.

Это полезно знать!

Фреймворк является программным каркасом, позволяющим корректно выстроить бизнес логику: он требует от разработчика учитывать особенность архитектуры приложения и ограничения.

Обычно фреймворки разбиваются на систему библиотек, реализующих классы (или объекты), методы и константы, необходимые для решения часто возникающих задач. Например, для веб-разработки это могут быть методы валидации данных, работа с базами данных, меха-

низмы авторизации, обработки форм ввода данных. Разработчик использует готовые возможности фреймворка, при необходимости расширяя структуру проекта дополнительной логикой.

Фреймворки также могут содержать инструменты разработки: библиотеки, трансляторы программного кода, текстовые редакторы, сборщики проекта и т.п.

Ярким примером фреймворка является технология .NET, разработанная компанией Microsoft. .NET является не только фреймворком, но и платформой, обеспечивающей работу программ.

Рис. 1.73. .NET поддерживает разработку под разные типы задач, а разработчикам доступна многофункциональная среда разработки Visual Studio

Фреймворки нацелены на упрощение и ускорение процесса разработки приложений и сервисов.

Преимущества использования фреймворков

- *Фреймворки ускоряют разработку больших проектов*. Часто возникающие задачи и повторяющийся функционал не требуется писать с нуля, а достаточно использовать встроенные структура данных и методы их обработки. Некоторые фреймворки позволяют автоматически генерировать базовую структуру проекта, что упрощает изучение концепции и возможностей фреймворка.

- *Фреймворки сокращают количество ошибок в коде и логике проекта*. Многие фреймворки ориентируются на лучшие практики и паттерны программирования, поддерживают механизмы тестирования и выявления ошибок еще до этапа сборки (рис. 1.74).
- *Фреймворки нацелены на потенциальное развитие проекта*. Архитектура библиотек выстраивается таким образом, чтобы разработчик мог использовать базовый функционал и расширять его дополнительными возможностями (например, наследованием классов или расширением объектов-прототипов).
- *Фреймворки обеспечивают безопасное и независимое выполнение программ*. Модули фреймворков обновляются и тщательно тестируются, чтобы исключить потенциальные проблемы в процессе выполнения приложения. Для работы с защищенными данными предусмотрены инструменты идентификации и разграничения доступа. Некоторые фреймворки создают автономную среду выполнения приложения или процесса, что гарантирует стабильность работы и исключает ситуацию, когда ошибка в приложении может привести к зависанию или сбою работы операционной системы. Например, в Java и .NET предусмотрена технология виртуальной машины, изолирующая работу собственных приложений (рис. 1.75).

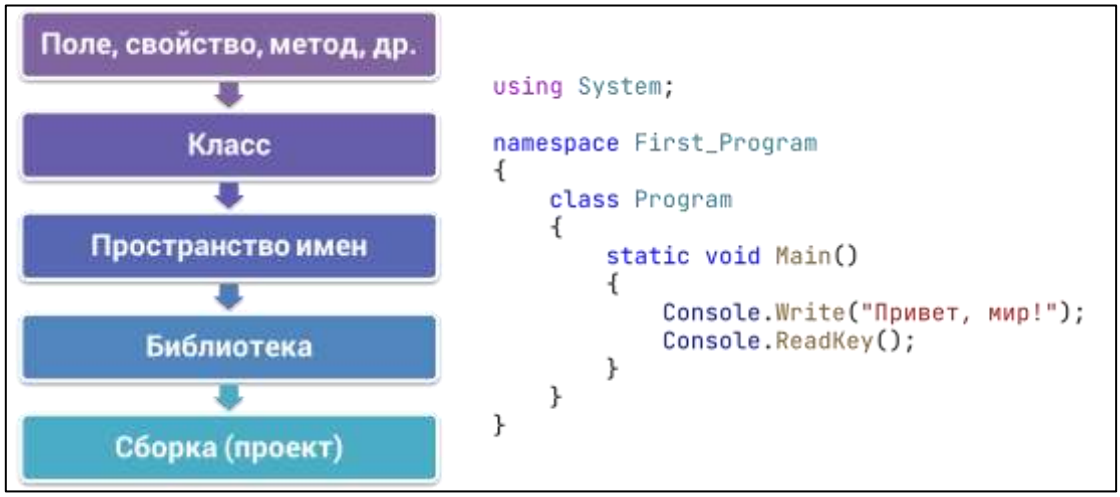


Рис. 1.74. Четкая структура проекта и модульной иерархии на примере простого консольного приложения на языке C# и платформе .NET

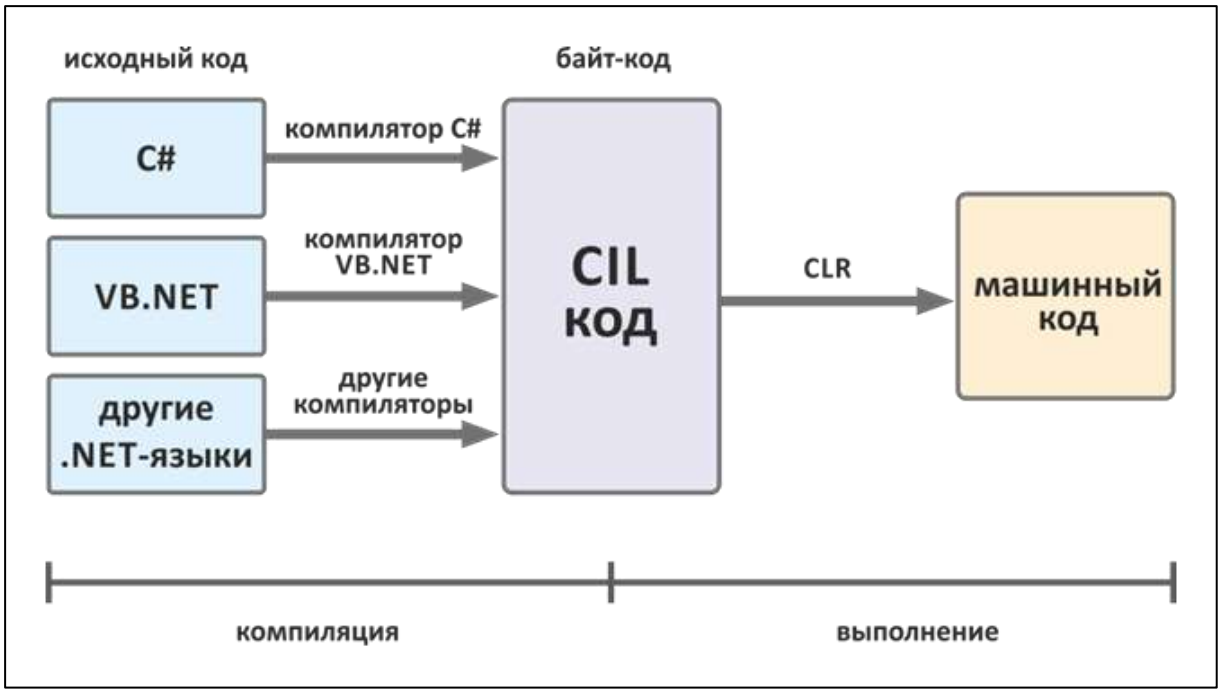


Рис. 1.75. .NET позволяет транслировать код на разных языках программирования в единую форму исполняемого файла

Типы фреймворков для веб-разработки

В зависимости от задач фреймворки для веб-разработки делятся по трем основным разновидностям на frontend-, backend- и fullstack-фреймворки.

Frontend-фреймворки используются для создания и оформления веб-интерфейса и не обычно требуют организации логики работы. Среди наиболее известных фреймворков выделим следующие:

- *Bootstrap* – бесплатный фреймворк с открытым программным кодом, сочетающий работу с технологиями HTML, CSS и языком JavaScript. Bootstrap ориентирован на разработку веб-сайтов с адаптивным дизайном и содержит многочисленные шаблоны для типовых задач.
- *Vue.js* – один из фреймворков для JavaScript, структурирующий создание веб-сайтов со сложным интерфейсом. Достоинством Vue является возможность постепенного наращивания функционала путем подключения дополнительных модулей по мере необходимости.
- *Foundation* похож на Bootstrap и заточен под оформление анимации. Этот фреймворк часто выбирают для веб-разработки под мобильные устройства.

Backend-фреймворки обрабатывают данные на стороне сервера и в целом предоставляют функционал для построения логики приложения. Важным элементом этих фреймворков является защита данных при обработке и отправке клиенту.

В качестве примером отметим следующие backend-фреймворки:

- *Laravel* – это фреймворк для языка PHP, простой в изучении и эффективный построении небольших и средних проектов.
- *Django* – один из мощнейших веб-фреймворков на языке Python. Удобен при создании проектов, требующих масштабируемости.
- *Symfony* используется в разработке Интернет-порталов. Отличается стабильностью и гибкостью работы.
- *Ruby on Rails* используется для разработки сайтов, требующих высокой скорости работы и отказоустойчивости. Фреймворк выбирают при создании сайтов с динамичным веб-интерфейсом и сложной бизнес-логикой.

Fullstack-фреймворки комбинируют задачи, решаемы на стороне клиента и сервера. Среди популярных обозначим следующие:

- *Meteor* реализует работу клиента и сервера на языке JavaScript. Обработка данных производится в режиме реального времени, что важно для динамически обновляемых сервисов.
- *Next.js* решает ряд проблем с отрисовкой, свойственных фреймворку React.
- *Nuxt* gозволяет создавать универсальные приложения на базе Vue и с использованием Node.js.

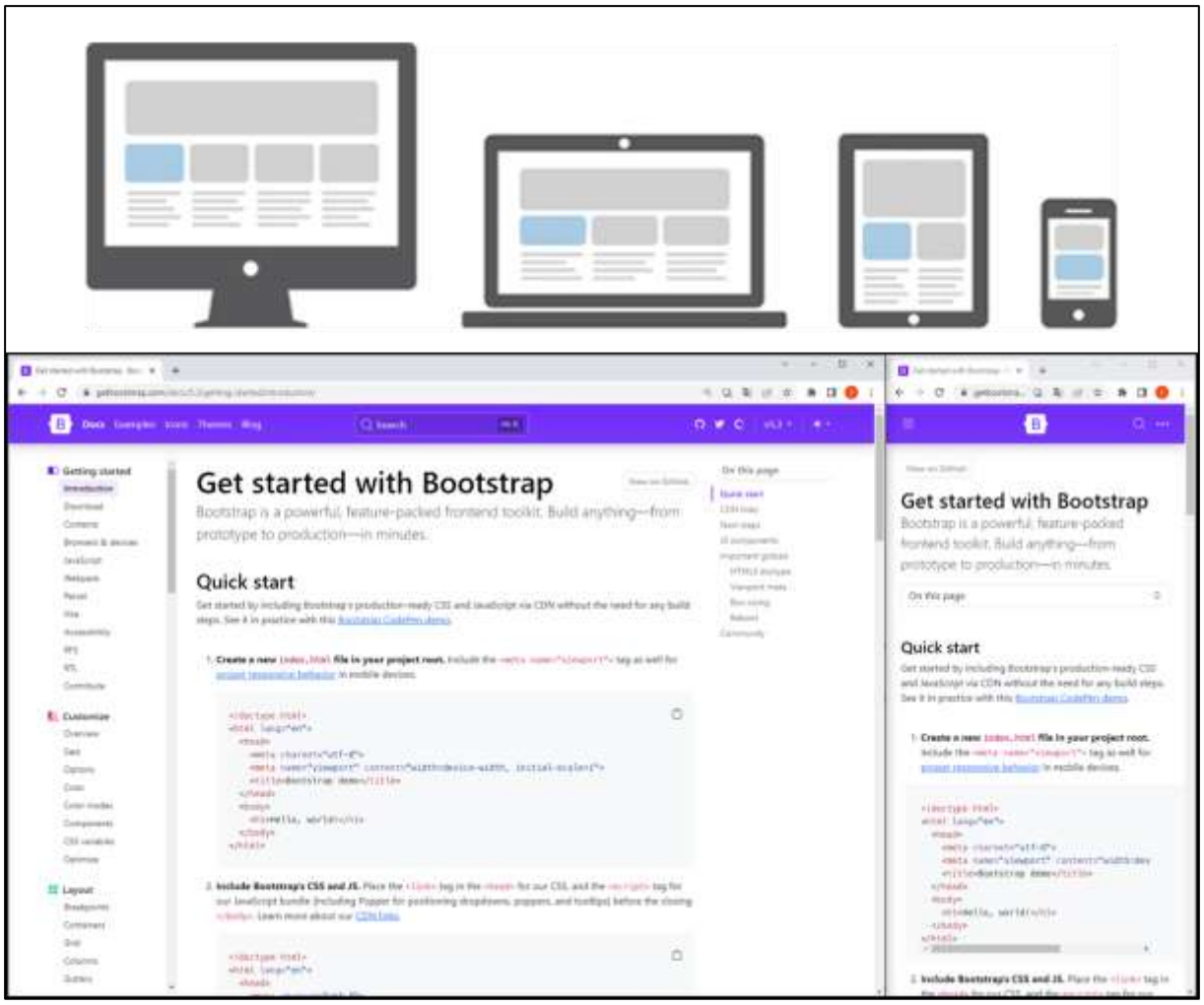


Рис. 1.76. Bootstrap содержит наборы CSS-классов и JS-модулей, обеспечивающих верстку адаптивного дизайна

Архитектура MVC

Архитектура современных фреймворков строится на базе схемы *Model-View-Controller* (*MVC*), которая делит структуру приложения на разные сегменты, каждый из которых отвечает за работу с обработку данных, их отображение и бизнес-логику взаимодействия.

Модель (*Model*) определяет данные и уровни выстраиваемой бизнес-логики, функции и правила доступа этот к данным из базы данных, файлов на жестких дисках или облачных сервисов.

Отображение (*View*) отвечает за организацию взаимодействия с пользователем. Программный код этого сегмента нацелен на обработку отображения данных на экране ПК или смартфона и в целом формирование текстового или графического интерфейса.

Контроллер (*Controller*) связывает элементы модели и отображения, управляя потоками данных и формируя их внешний вид.

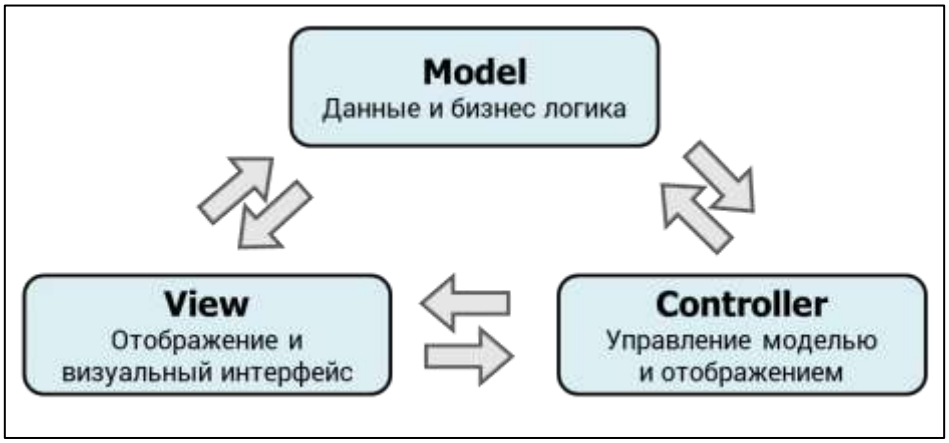


Рис. 1.77. Модель MVC

Обычно в архитектуру MVC в качестве четвертого элемента включается пользователь, который взаимодействует с моделью с помощью средств контроллера и отображения.

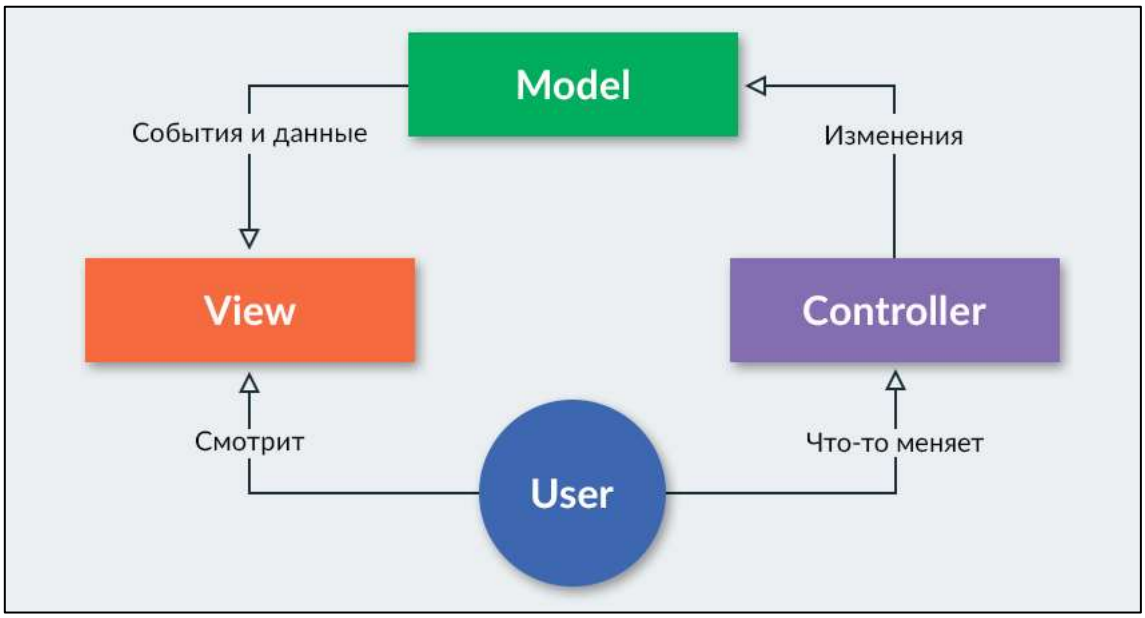


Рис. 1.78. Пользователь в системе MVC

Разделение в MVC позволяет эффективнее организовать архитектуру проекта, отделив бизнес-логику от ее визуализации. MVC упрощает программирование сложных систем, повышает читаемость кода, облегчает масштабирование и техническую поддержку работоспособности системы.

Разница между API и Фреймворками

Зачастую понятие фреймворка, API и программной библиотеки отождествляют, что некорректно.

API позволяет приложению обратиться к внешнему программному коду и обеспечивает интерфейс взаимодействия с независимыми сервисами.

Библиотека представляет собой набор уже реализованных данных, классов, методов и алгоритмов решения типовых задач определенной сферы. Библиотека используется приложением во время его работы.

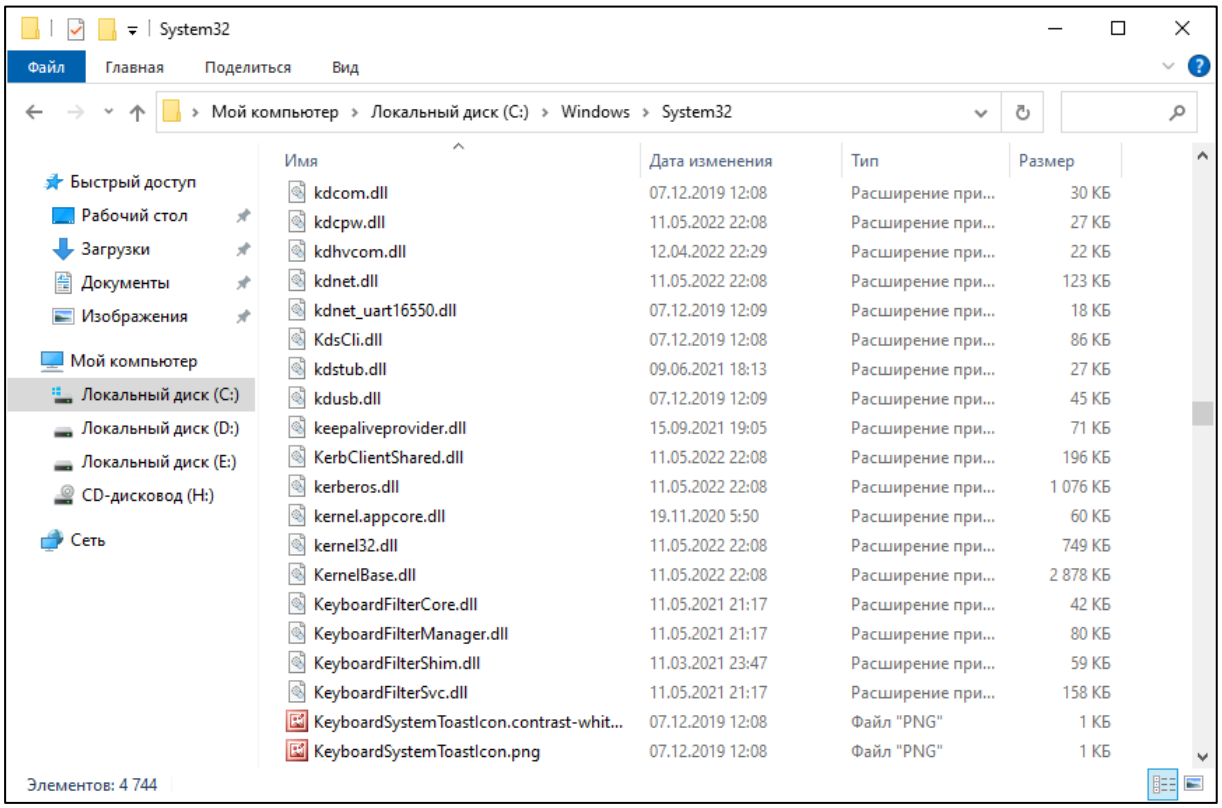


Рис. 1.79. Библиотеки динамической компоновки DLL, используемые операционной системой Windows

Фреймворк предоставляют разработчику шаблоны приложения, которые написаны в соответствии с определенными паттернами проектирования (MVC, MVVC и т.п.). Обычно фреймворк включает в себя библиотеки, однако он выстраивает архитектуру предложения, где модули жестко связаны механизмом интерфейса. Расширение приложения потребует соблюдения правил архитектуры фреймворка.

1.5.2. Интегрированные среды разработки ПО

Использование в разработке

Понятие

Определение

Интегрированная среда разработки*, IDE** *(от англ.* ***I*ntegrated* ***D****evelopment* ***E****nvironment) – комплекс программных средств, предназначенный для разработки ПО.*

IDE предоставляют разработчику комплекс инструментов, необходимых для набора и проверки кода, сборки, запуска и отладки приложений, тестирования программ и многих других задач. Первоочередной целью использования сред разработки является создание эргономичного виртуального рабочего пространства, которое упрощает создание проектов.

Обычно в структуру IDE включены следующие компоненты:

- продвинутый текстовый редактор;
- программа-транслятор (компилятор, интерпретатор);
- инструменты автоматизированной сборки проекта.

Классификация IDE

В зависимости от стоимости IDE делятся на три категории:

- *Бесплатные* обычно используются школьниками, студентами или начинающими разработчиками для реализации частных проектов. При этом исходный программный код среды может быть защищен авторским правом.
- *Условно-бесплатные* IDE предоставляют возможность работы либо в ограниченный ознакомительный период времени, либо имеют ограниченный функционал по сравнению с коммерческой версией. Такой вариант удобен для индивидуальных разработчиков, учебных организаций и небольших компаний. Некоторые производители IDE предоставляют пользователям возможность работы с бесплатной версией среды, чтобы продвигать ее платный вариант.

- *Платные IDE* требуют приобретения лицензии или подписки на определенный срок. Такой вариант использования среды необходим компаниям и частным лицам, занимающимся крупными проектами. По желанию пользователя могут быть добавлены расширенные надстройки и модули (бесплатно или за дополнительную плату).

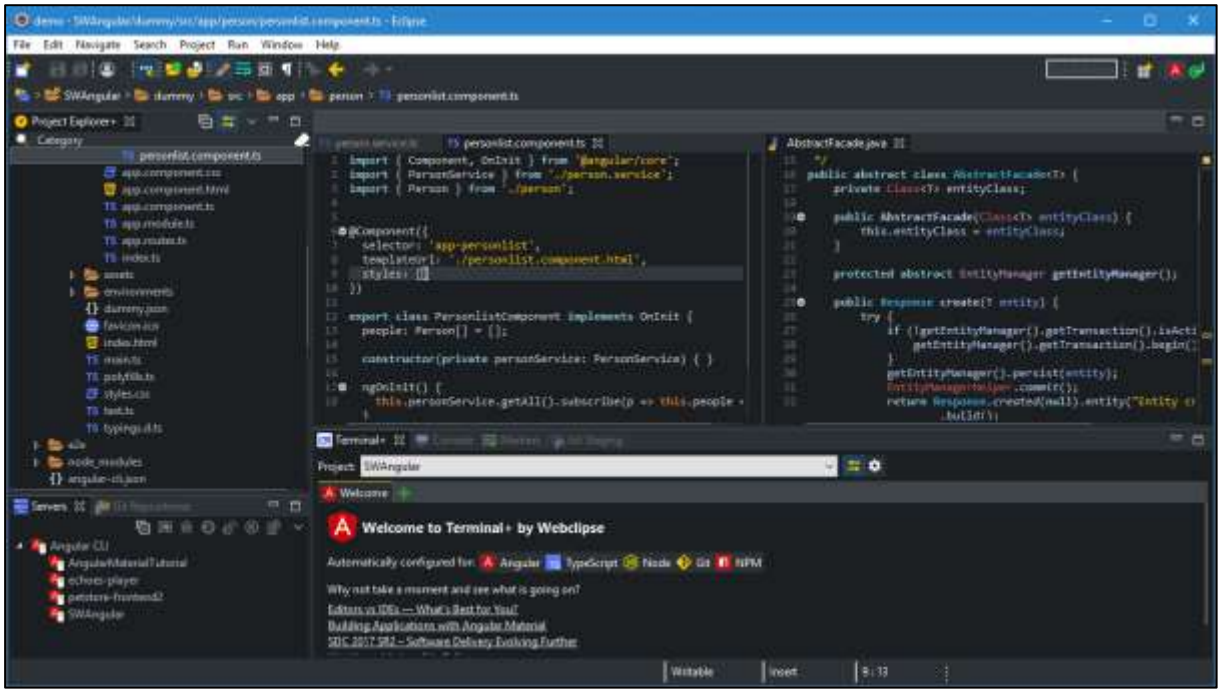


Рис. 1.80. Бесплатно распространяемая Eclipse IDE

В зависимости от универсальности выделяют следующие виды сред разработки:

- *Одноязычные* IDE нацелены и оптимизированы на работу с определенным языком программирования.
- *Мультиязычные* IDE поддерживают работу с несколькими языками программирования. Некоторые среды способны добавлять другие языки программирования и настраивать работу с ними согласно предпочтениям пользователя.

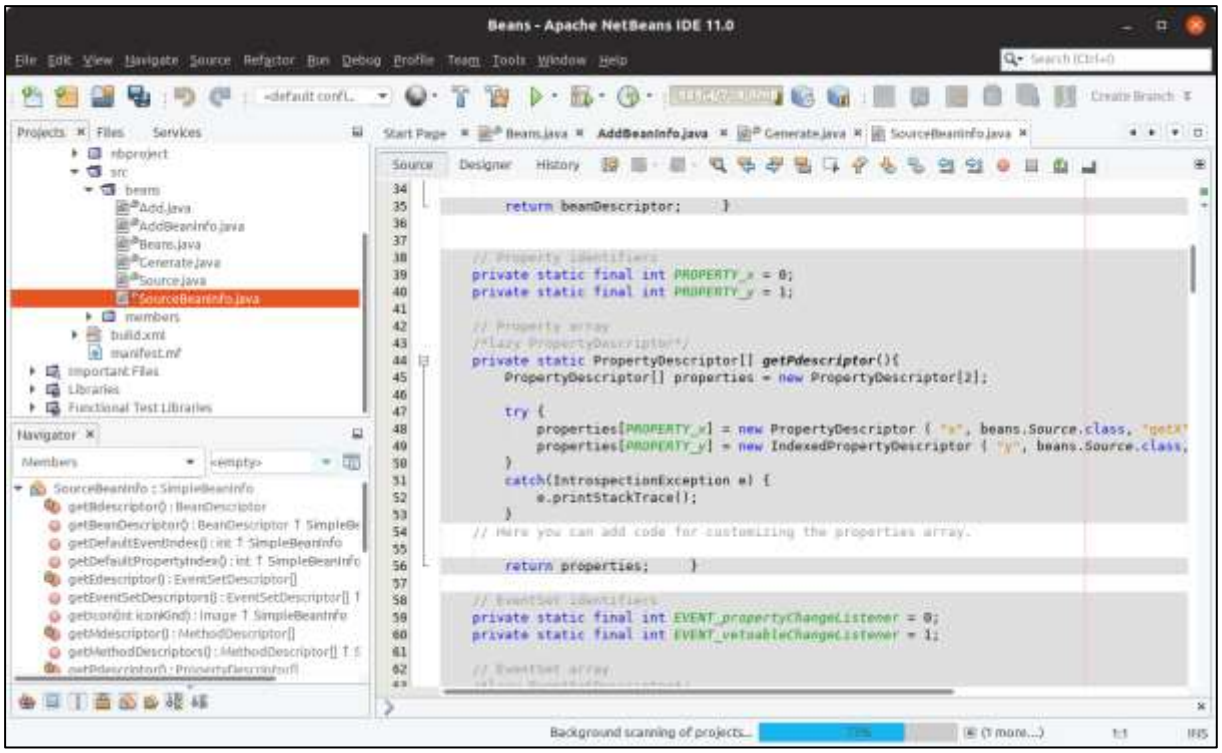


Рис. 1.81. NetBeans IDE позволяет работать с такими популярными языками как Java, C/C++, PHP, JavaScript

Возможности

Современные среды разработки могут являться автономной системой, зачастую позволяющей реализовать весь цикл разработки приложения. Рассмотрим важнейшие инструменты и возможности сред разработки.

1. Встроенный текстовый редактор

Ключевым компонентом любой IDE является встроенный текстовый редактор, поддерживающий функции подсветки синтаксиса, управления областью отображения кода и предоставляющий гибкие инструменты поиска и замены в программном коде.

В отличие от обычных текстовых редакторов, окно редактора IDE интерактивно и позволяет выделять структуру кода, показывать ссылочную связь между его модулями, просматривать графические элементы, осуществлять групповые преобразования фрагментов кода и многое другое.

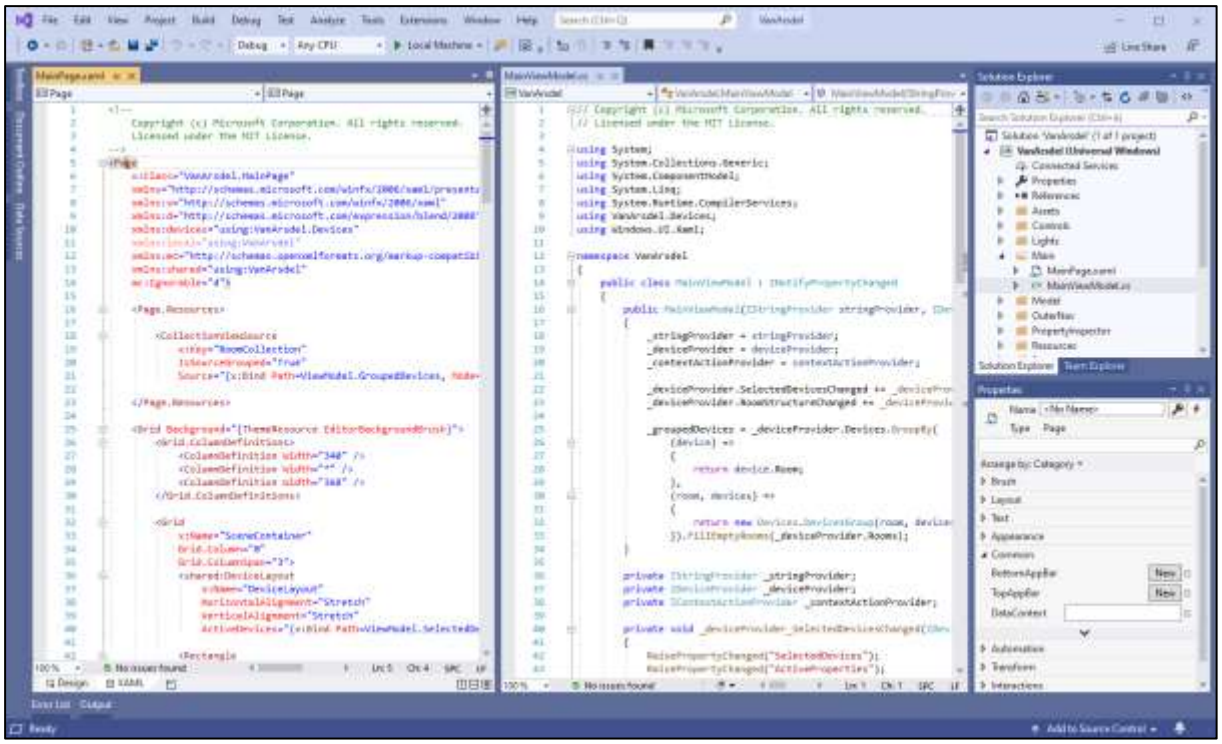


Рис. 1.82. Редакторы кода IDE позволяют делить область на несколько колонок и параллельно редактируемых файлов

2. *Автозавершение кода*

Важным преимуществом использования IDE является технология *IntelliSense* – механизм автоматической подсказки и автозавершения кода одним нажатием. IDE позволяют генерировать стартовые шаблоны для распространенных типов приложений, вставлять фрагменты часто используемых конструкций. Также постепенно в IDE начинают внедрять элементы искусственного интеллекта, который способен предугадывать действия разработчика.

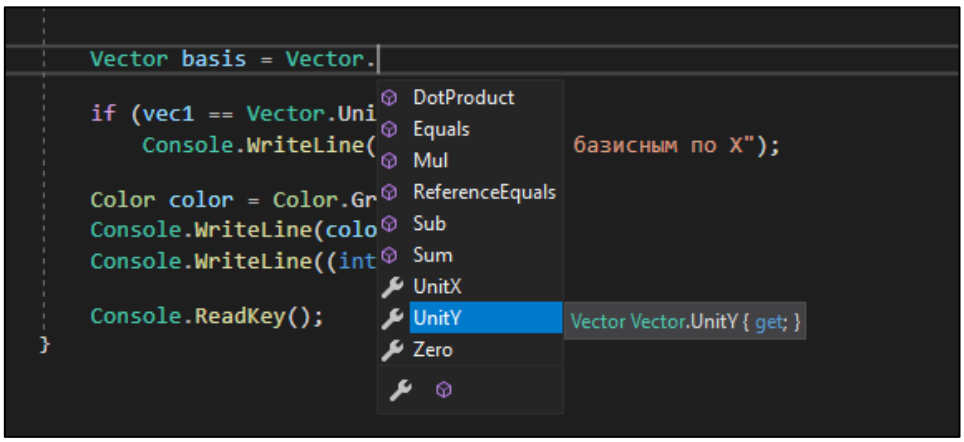


Рис. 1.83. Окно всплывающей подсказки при наборе кода

3. Интерактивный файловый менеджер

IDE содержит собственный файловый менеджер, отображающий структуру проекта, который похож на проводник в операционной системе. В окне проводника в интерактивном режиме осуществляется управление файлами и каталогами проекта, поиск фрагментов в проекте, подключение ссылок на модули и библиотеки.

Встроенный проводник анализирует структуру кода и выделяет логические блоки, помечая их разными символами, что упрощает визуальный поиск необходимого файла.

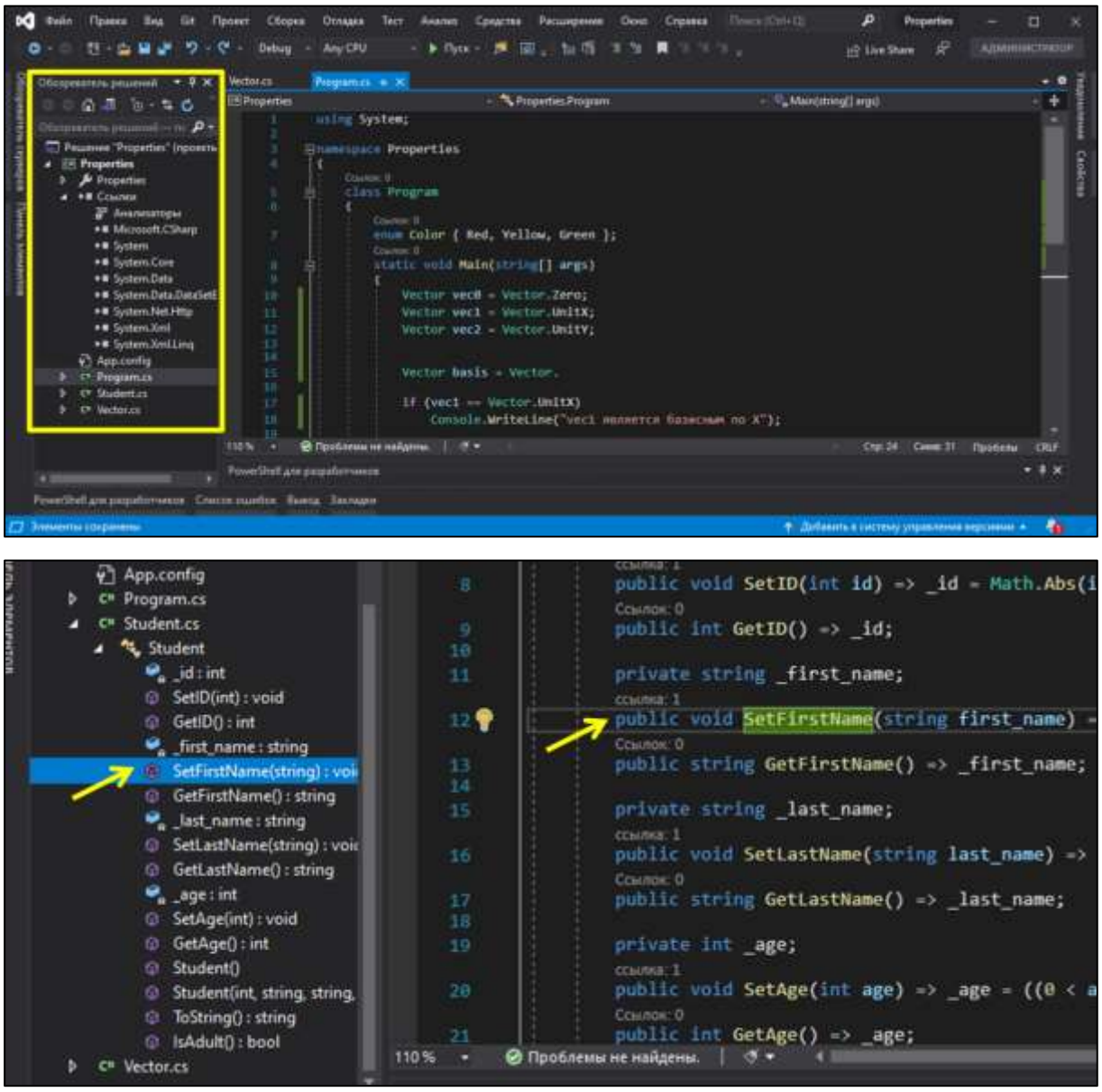


Рис. 1.84. Переход к реализации метода класса в файле с кодом

4. Конструктор визуального интерфейса

Некоторые среды содержат удобный конструктор визуального интерфейса приложений. Разработчику достаточно лишь разметить элементы на форме и настроить их оформление, а IDE автоматически обновит программный код.

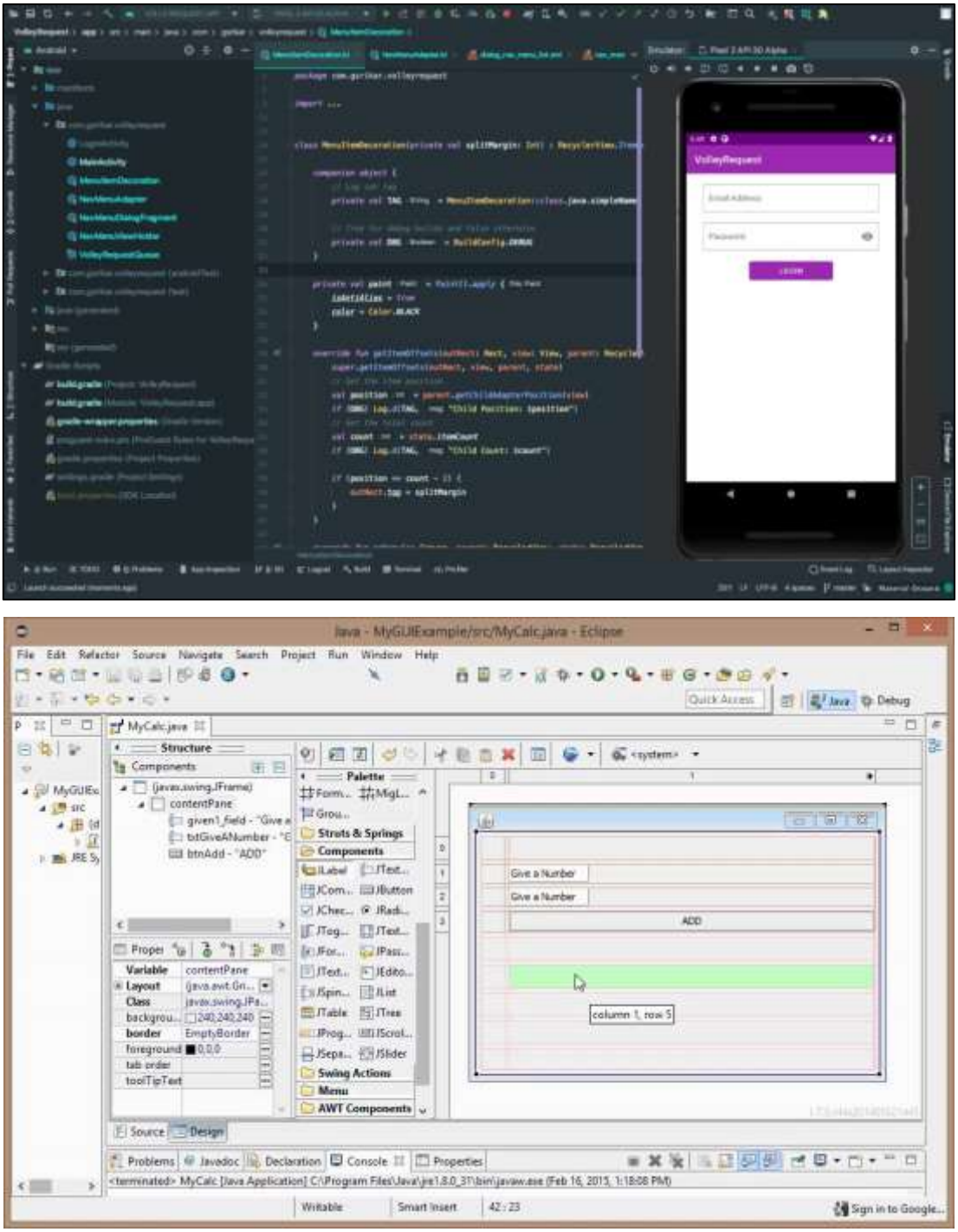


Рис. 1.85. Визуальный конструктор в средах разработки Android Studio и Eclipse

5. *Инструменты рефакторинга*

Рефакторинг кода – это процесс его преобразования, позволяющий упростить чтение, понимание и изменение кода, не изменяя при этом его поведение. Рефакторинг помогает улучшить и систематизировать структуру кода, выстроить его в соответствии под определенные стандарты и паттерны программирования.

Комплексный рефакторинг обычно производится по завершению реализации предварительной версии проекта, когда получается в целом работоспособное приложение. В отличие от оптимизации первостепенной целью рефакторинга является повышение читаемости кода, однако зачастую грамотный рефакторинг может выявлять более эффективные и универсальные алгоритмы реализации фрагментов программы.

Методология рефакторинга включает комплекс действий по преобразованию кода. Современные среды разработки способны автоматизировать часть операций по рефакторингу, например – автоматически менять сигнатуру методов, классов, переводить запись к обновленной версии синтаксиса, извлекать модули и т.д.

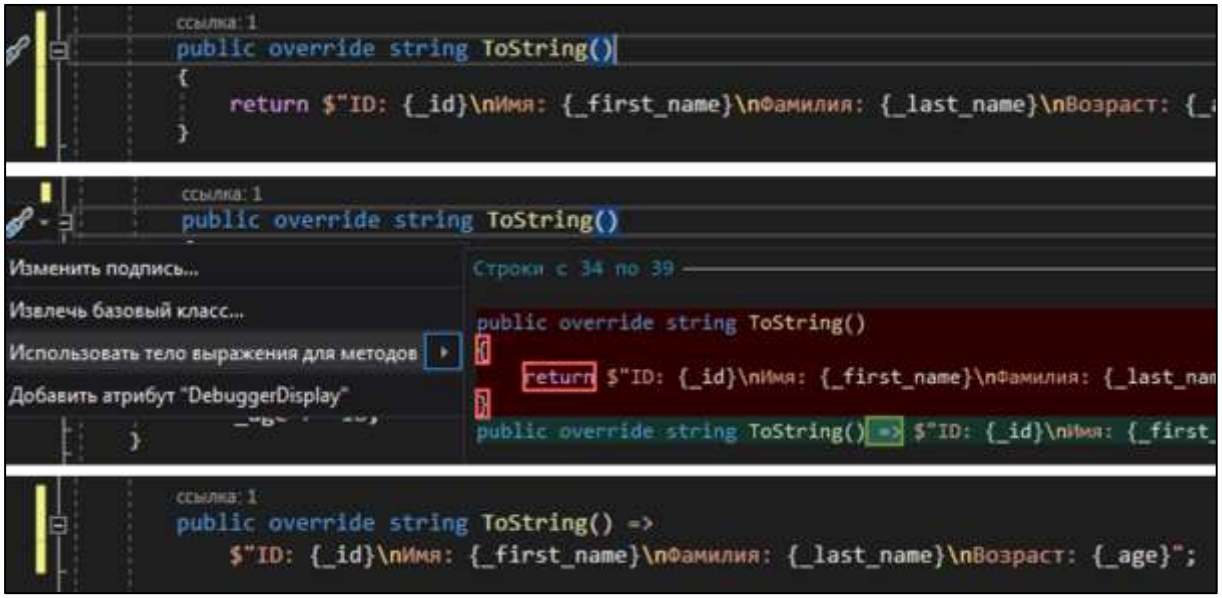


Рис. 1.86. С помощью встроенного инструмента рефакторинга среда преобразует описание метода к более новому и короткому синтаксису (на примере среды Visual Studio и языка программирования C#)

6. Автоматическая компиляция и сборка проекта

Существенно упростить и ускорить разработку помогает автоматизация операций при компиляции программного кода и сборке проекта, состоящего из различных модулей. Здесь среда разработки берет на себя рутинные операции обращения к транслятору (компилятору или интерпретатору), что позволяет запускать приложение в режим отладки нажатием одной кнопки.

Разумеется, для сборки проекта необязательно задействовать IDE. Разработчик может обратиться к транслятору напрямую в командной строке, указав файлы проекта и параметры трансляции в исполняемый файл или параметры выполнения. Частично упростить эту процедуру позволяют bat-файлы и make-файлы, хранящие необходимую для сборки информацию.

Однако IDE универсализирует компиляцию и сборку программы, скрывая от разработчика явные операции с командной строкой. При необходимости параметры компиляции могут быть исправлены в более удобном визуальном интерфейсе среды, а командная строка вызвана непосредственно из редактора.

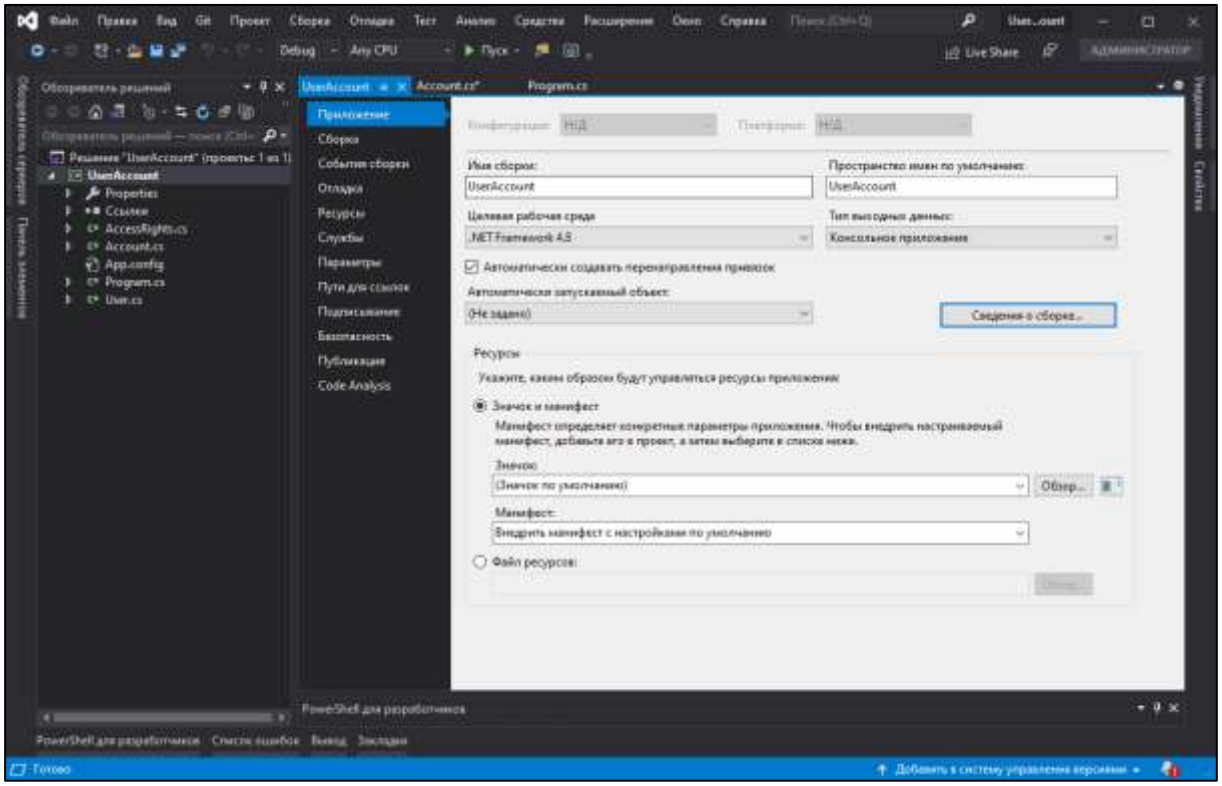


Рис. 1.87. Настройка параметров автоматической компиляции программы

7. Отладка и тестирование

Современные IDE имеют развитый механизм проверки кода на наличие ошибок. Среда способна анализировать синтаксис языка и отслеживать возникающие проблемы в режиме реального времени, до того, как проект будет скомпилирован. Кроме ошибок IDE также показывают предупреждения о нежелательных операциях (не являются ошибками) и предлагают возможную оптимизацию кода.

В процессе проверки корректности работы кода используются возможности встроенных отладчиков. Они позволяют выявлять ошибки, пошагово анализировать выполнение фрагментов кода и отдельных модулей, динамически менять данные и другое.

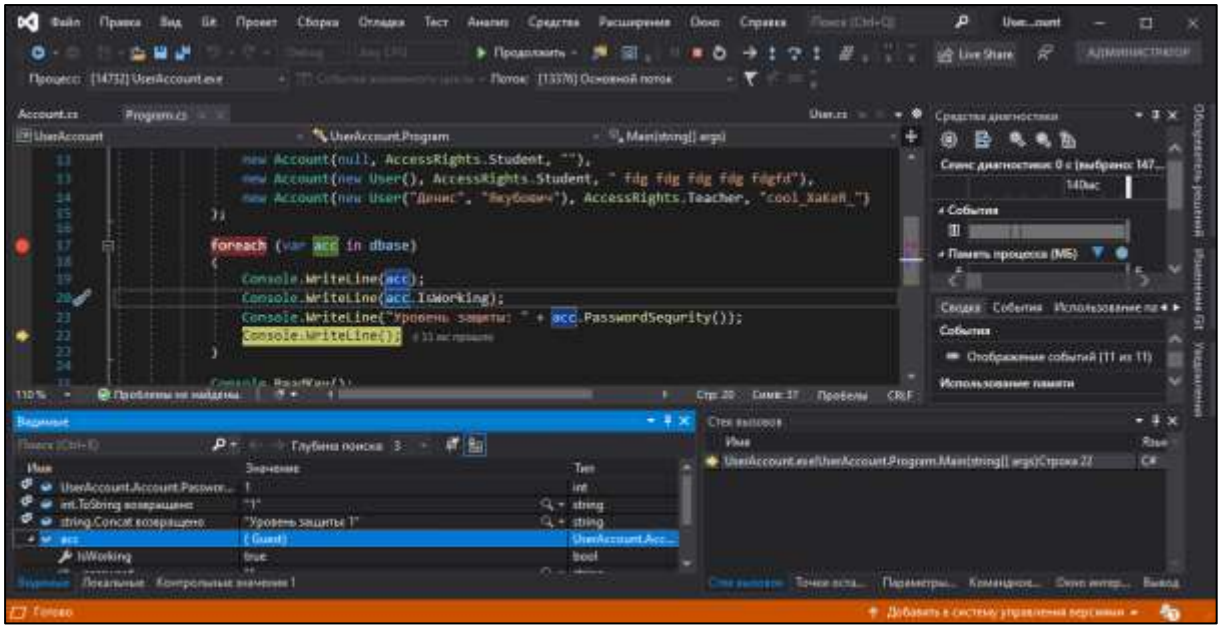


Рис. 1.88. Отладка фрагмента кода: пошаговый анализ данных (переменных)

После того, как синтаксические и логически ошибки в коде устранены, осуществляется комплексное тестирование программного продукта по одной из выбранных методологий.

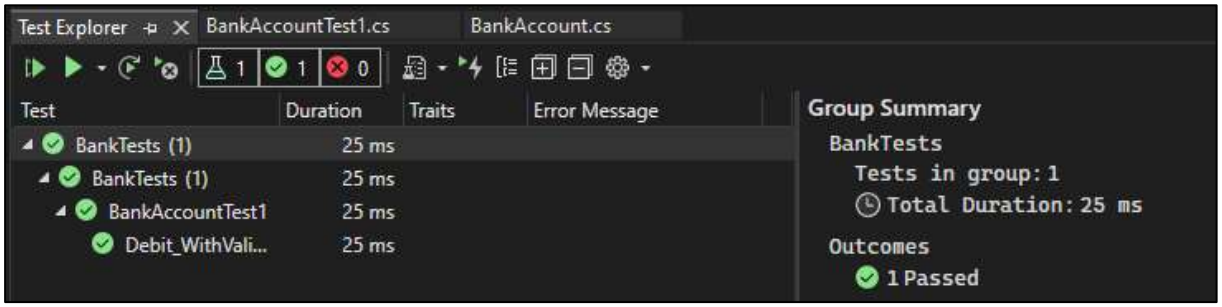


Рис. 1.89. Модульное тестирование приложения средствами Visual Studio

8. Анализ производительности

Востребованной задачей в разработке является анализ особенностей работы приложения: скорость выполнения, потребляемые ресурсы, время отклика и т.д. Для этого IDE предоставляют разработчикам удобные и визуальные средства профилирования.

Профилировщик позволяет анализировать производительность программы и диагностировать потребляемый ресурс памяти и нагрузки центрального процессора, выявлять возникающие проблемы на уровне отладки. Для удобства профилировщик выдает информацию в графической форме, отображая график используемой памяти или загрузки ЦП относительно времени.

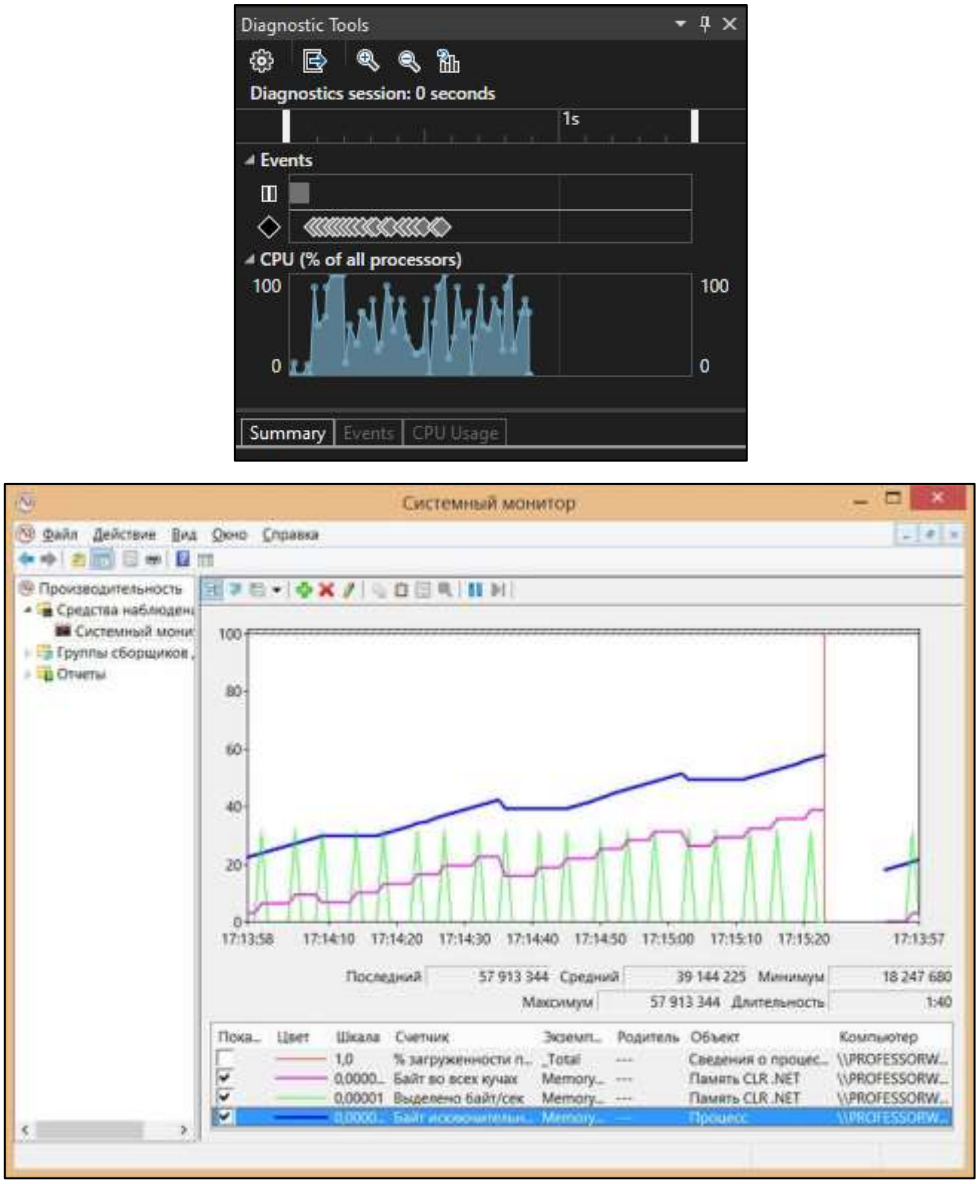


Рис. 1.90. Некоторые инструменты профилирования на примере Visual Studio

9. *Синхронизация с системами контроля версий*

Разработка большого проекта предполагает последовательную и командную работу, на каждом из этапов которой требуется изменение программного кода. Чтобы процесс разработки был гибким, необходимо контролировать процесс изменения проекта и обеспечивать доступ к копиям этого проекта другим разработчикам.

Системы контроля версий позволяют хранить копии кода, создавать отдельные ветви для его изменения и обеспечивать совместную работу над проектом. Работа с системами контроля версии может вестись как отдельно, так и в связке с IDE пользователя.

Наиболее популярные среды разработки (такие как Visual Studio) имеют встроенный инструмент интеграции с технологией Git.

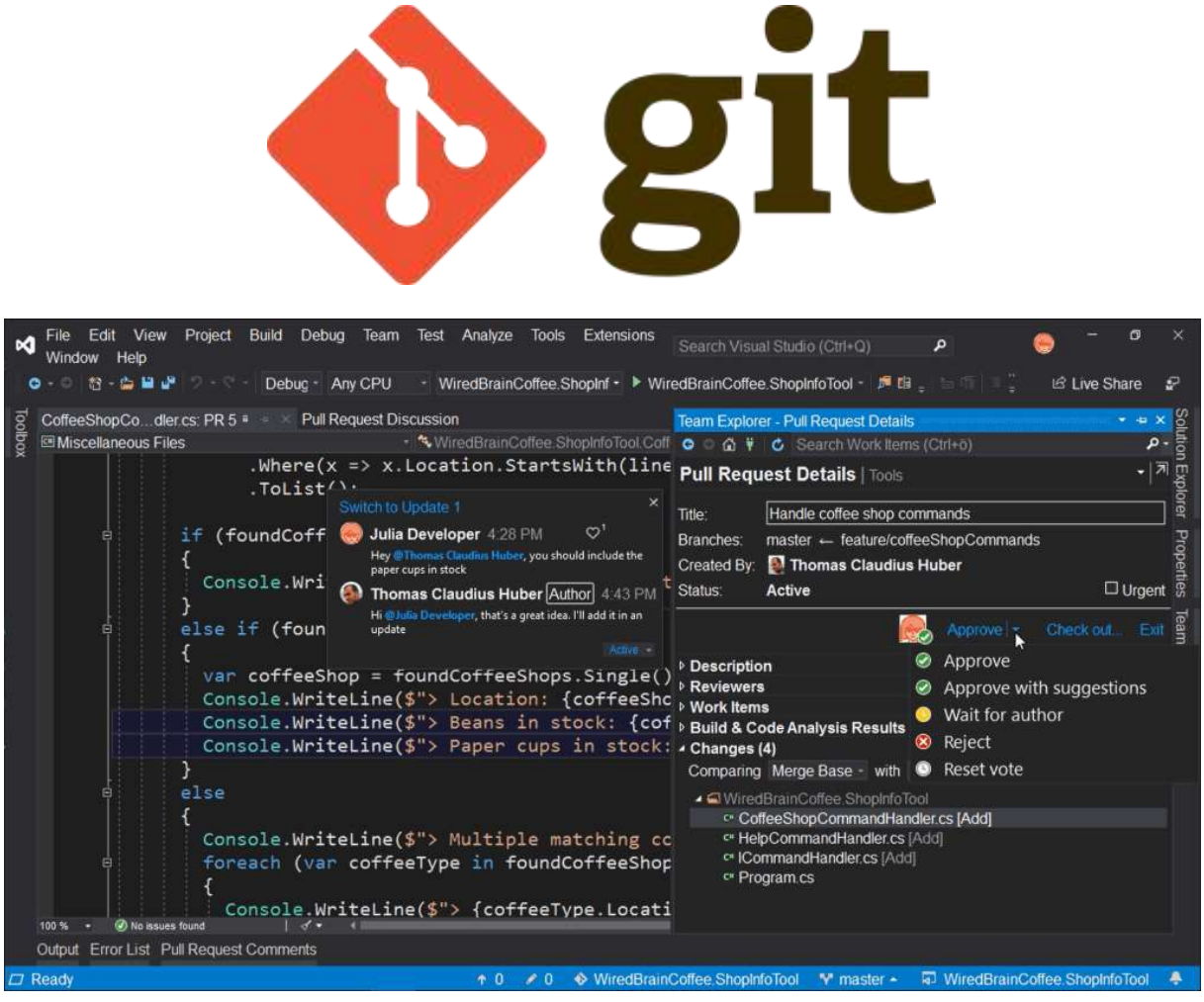


Рис. 1.91. Технология Git позволяет гибко управлять разными версиями приложения и его ветвями

Методология RAD

Концепция быстрой разработки

Определение

***RAD** (от англ. **R**apid **A**pplication **D**evelopment, **быстрая разработка приложений**) — модель организации технологического процесса и разработки ПО, предполагающая максимально быстрое достижение результата при условии жестких ограничений по времени и бюджету с нечётко определёнными требованиями к продукту.*

Концепция RAD предполагает, что в разработке задействована небольшая команда, которая реализует проект в течение 3-4 месяцев. Разработка ведется методом *инкрементного прототипирования* (постепенное наращивание функционала модели-прототипа). В процессе реализации задействуются средства визуального моделирования и разработки. RAD предполагает активное вовлечение заказчика уже на первоначальном этапе разработки.

Высокая популярности RAD-технологий определяется рядом преимуществ, ведущими из которых являются:

- высокая скорость проектирования и разработки;
- низкая стоимость реализации;
- высокое качество итогового проекта.

Традиционная и быстрая разработка

Традиционная разработка базируется на четко установленной последовательности действий. Эту модель часто называют Waterfall (по аналогии с водопадной системой), поскольку она основана на традиционной инженерной модели проектирования и строительства зданий и мостов.

Однако в ряде случаев жесткая последовательность традиционной модели неэффективна. RAD предполагает организацию гибкой разработки, которая учитывает опыт, полученный в рамках процесса жизненного цикла управления проектом.

Рис. 1.92. Традиционный и RAD-подход к разработке

Первая RAD была предложена в 1986 году Б. Боэмом. Концепция предусматривала спиральную модель разработки. Каждый виток спирали является фрагментом или версией проекта. Каждая новая итерация уточняет цели, спецификации и реализацию проекта.

В дальнейшем концепции методологии RAD были развиты Дж. Керром и Р. Хантером. Новая методика получила большое распространение и повлияла на возможности систем разработки.

Принципы RAD

При ограниченных сроках и малых затратах обеспечить высокое качество проекта и достичь компромисса между заказчиком и разработчиками весьма затруднительно. Поэтому в методологии RAD ориентируются на ряд принципов:

- *Минимизация сроков разработки*. Инструменты разработки проекта должны быть эффективны в реализации конкретного проекта и подбираться с требуемого учетом необходимого функционала.
- *Разработка прототипов*. RAD-разработка ведется на базе моделей, которые позволяют оценить проект, осуществить его декомпозицию на отдельные элементы, каждая из которых разрабатывается отдельной группой. Прототипы позволяют заказчику уточнить требования к проекту на каждом этапе.

- *Цикличная разработка*. Каждая последующая версия проекта учитывает оценку заказчика, корректность и эффективность предыдущей версии. Разработка является последовательным и итеративным развитием прототипов, каждый из которых реализует некоторый функционал на текущем этапе развития проекта. Каждый следующий виток разработки расширяет возможности прототипов (изначальный прототип реализует только интерфейс приложения без функциональной обработки).
- *Динамическое тестирование*. Каждая итерация разработки требует комплексной проверки корректности работы всех компонент и системы в целом.
- *Групповая разработка*. Методология RAD эффективна для коллектива разработчиков. Важно тесное взаимодействие и четкое распределение обязанностей.

Это полезно знать!

Принципы RAD эффективны не только в реализации ПО, но и на разных этапах жизненного цикла разработки проекта.

Жизненный цикл разработки в RAD

Выделяют 4 фазы RAD-модели:

1. *Анализ и формулировка требований*. На этом этапе определяются требования к внешнему виду и функционалу проекта, их приоритету. В фазе активно задействуются будущие пользователи. Также оценивается планируемый масштаб проекта, сроки реализации, бюджет, обсуждаются платформы для размещения ПО.
2. *Проектирование*. Заказчик и пользователи участвуют в техническом проектировании системы. Обычно используется техника совместной разработки приложений (JAD) и CASE-технологии, позволяющие достичь потребности пользователей. На этой фазе пользователи определяют, необходимы ли изменения модели. Для более точного представления о будущем проекте создается первоначальный прототип.

3. *Построение проекта*. На этой фазе осуществляется быстрая разработка. Пользователи могут предлагать вносить изменения в функционал отдельных модулей. Тестирование осуществляется динамически, в процессе разработки.
4. *Внедрение*. Предполагает внедрение новой или обновление старой системы, обучение пользователей и тестирование проекта в реальных условиях.

Это полезно знать!

В отличие от классической модели, жизненный цикл проекта в методологии RAD позволяет гибко управлять фазами разработки, упрощая их по возможности.

Примеры сред разработки

1. Visual Studio

Visual Studio – одна из наиболее мощных и многофункциональных IDE для разработки ПО. Разработана компанией Microsoft и ориентирована на операционные системы Windows и macOS.

Visual Studio поддерживает работу с целым рядом языков программирования, в частности C++, C#, Python, JavaScript, PHP, TypeScript, Ruby, Go, HTML и CSS. Большое внимание уделено работе с языком C#, который разработан Microsoft для современной платформы .NET.

Visual Studio содержит комплекс инструментов проектирования и разработки, характерный современным требованиям и тенденциям в IT-индустрии, включая удалённую отладку.

Платформа включает:

- интерактивный редактор кода с технологией IntelliSense;
- инструменты отладки, тестирования и профилирования;
- механизмы совместной разработки и управления версиями;
- средства интеграции с Git;
- инструменты развёртывания на базе Azure.

Основным недостатком Visual Studio является сравнительно высокая цена. Однако пользователям доступна бесплатная версия – Visual Studio Community, обладающая многими функциями профессиональной версии и в целом удобная в разработке коммерческих приложения частными лицами.

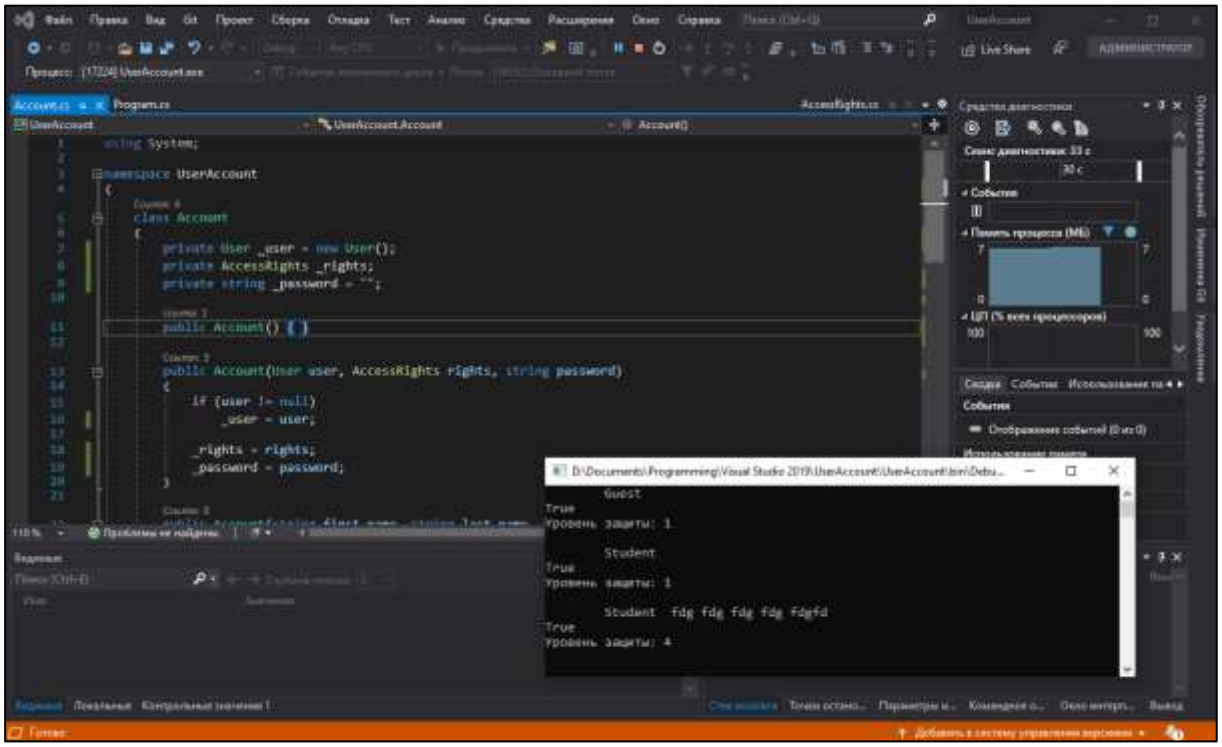


Рис. 1.93. Интерфейс IDE Visual Studio Community 2019

2. IntelliJ IDEA

IntelliJ IDEA – это интегрированная среда разработки приложений, разработанная компанией JetBrains. IntelliJ в первую очередь ориентирована на создание проектов на языке Java. Однако среда поддерживает работу и с другими популярными языками веб-разработки, в частности Kotlin, Scala, JavaScript, Python, PHP, HTML, CSS, Go, Ruby. Среда также является кроссплатформенной и поддерживает работу в операционных системах Windows, macOS и Linux.

IntelliJ является хорошей альтернативой таким крупным проектам, как Visual Studio и предоставляет следующий функционал:

- продвинутый тестовый редактор с технологией автоподсказки и автозавершения IntelliSense;
- автоматизированная отладка, тестирование и профилирование кода;

- инструменты рефакторинга;
- вставка фрагментов на других языках;
- совместная разработка и управления версиями, интеграция с системами контроля версий;
- инструменты развёртывания;
- интеграция с инструментами сборки (например, Webpack).

Среда IntelliJ IDEA распространяется в трех вариантах:

- Community-версия бесплатна, однако она не поддерживает работу с JavaScript и базами данных;
- EDU-версия бесплатна для студентов и преподавателей;
- Ultimate-версия является коммерческой для индивидуальных разработчиков и компаний.

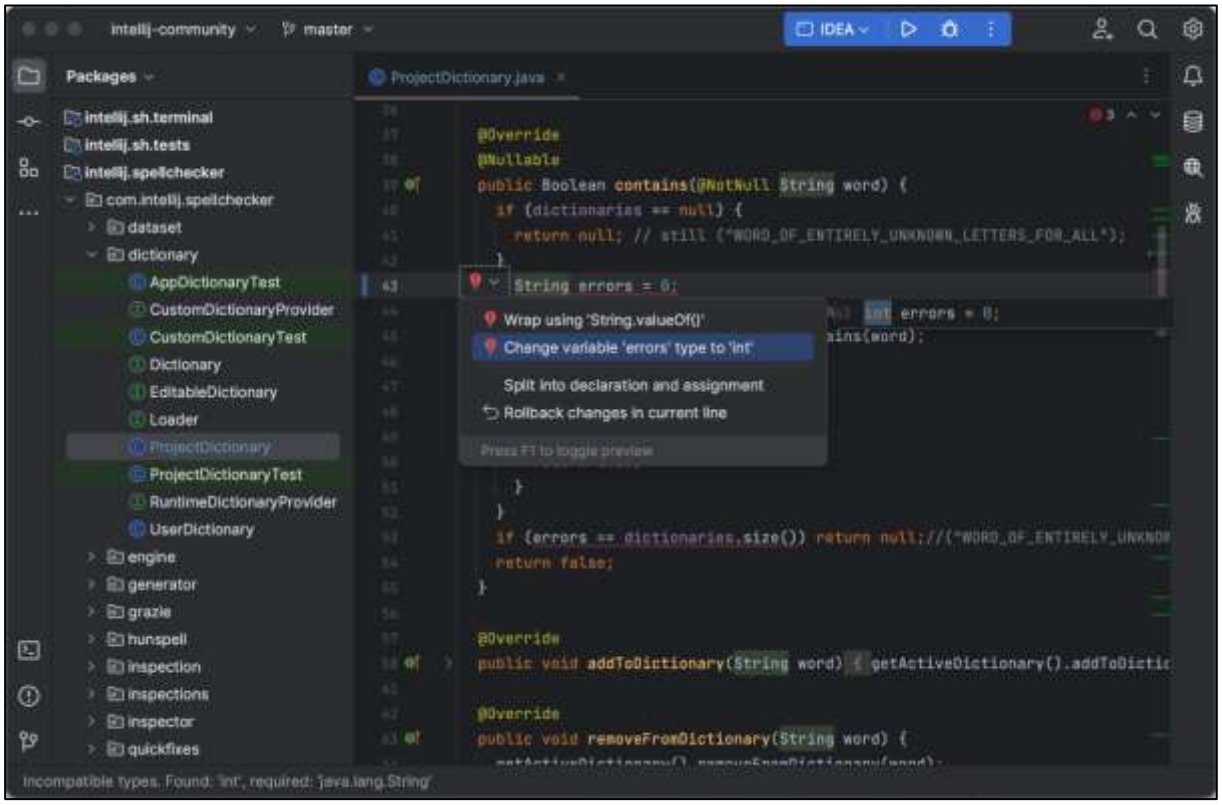


Рис. 1.94. Интерфейс IDE IntelliJ IDEA 2023

3. NetBeans

NetBeans – бесплатная среда разработки с открытым исходным программным кодом от Apache. NetBeans хорошо оптимизирован под разработку на языке Java. Кроме того, среда поддерживает программирование на классическом C/C++, HTML, CSS, JavaScript, PHP,

Python. При необходимости функционал расширяется установкой плагинов.

Как и другие IDE своего класса, NetBeans содержит:

- многофункциональный редактор с подсветкой синтаксиса и режимом автодополнения кода;
- конструктор визуального интерфейса приложений;
- визуальный отладчик;
- инструменты для работы с технологией Java;
- инструменты работы с базами данных;
- средства тестирования приложений;
- доступ к настройкам эргономики среды;
- встроенные инструменты интеграции с Git, Maven и другими платформами.

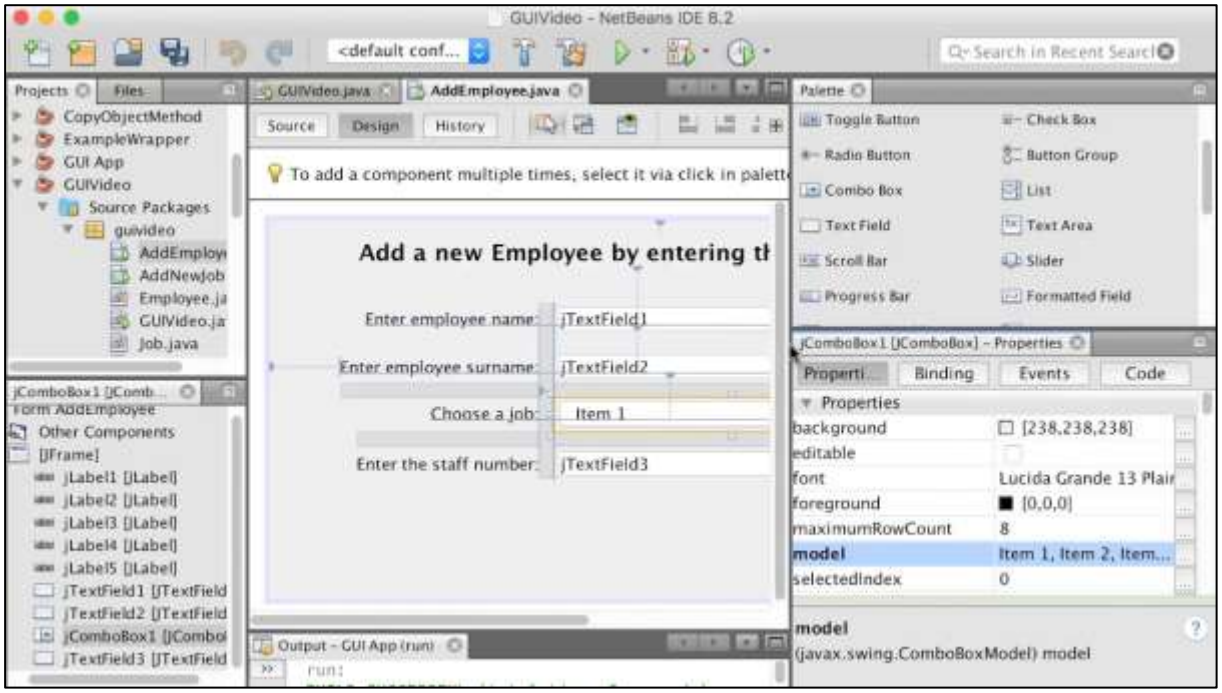


Рис. 1.95. Интерфейс NetBeans IDE 8.2

4. PhpStorm

PhpStorm – один из продуктов компании JetBrains, ориентированный на веб-разработчиков на языке PHP. Также среда поддерживает связанные технологии HTML, CSS и язык JavaScript и способна работать на операционных системах линейки Wondows, Linux и macOS.

Возможности PhpStrom:

- текстовый редактор с функцией автозавершения кода;
- поддержка работы с PHP-фреймворками;
- содержит визуализированный отладчик кода;
- предоставляет необходимый инструмент для работы с frontend-технологиями;
- позволяет синхронизироваться с FTP, FTPS, SFTP;
- работает с базами данных и SQL;
- интегрированы средства контроля версий.

Программа распространяется исключительно как коммерческая среда разработки.

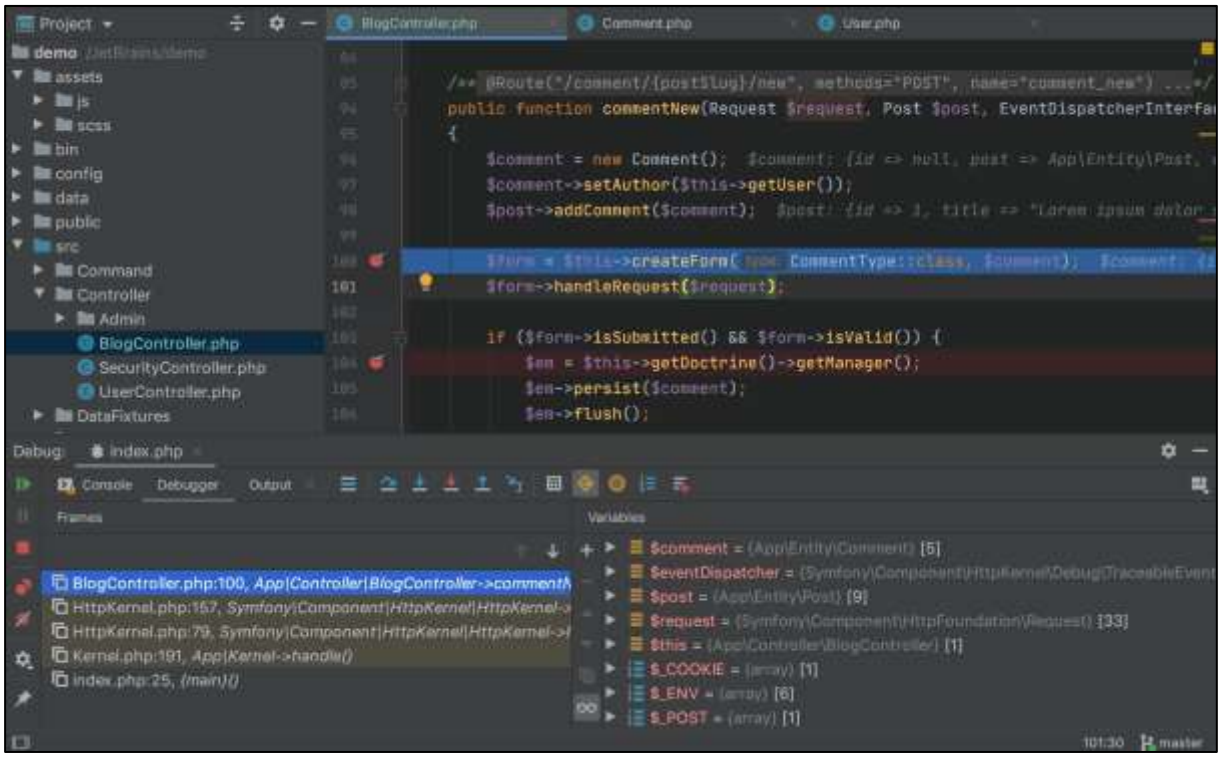


Рис. 1.96. Интерфейс IDE PhpStorm

5. WebStorm

WebStorm – мощная среда для программирования на языке JavaScript и связанные с ним популярные frontend-фреймворки Angular, React, Vue.js и backend-фреймворки Node.js, Meteor.

Среди преимуществ достоинств WebStorm отметим:

- поддерживает автодополнение и анализ кода «на лету»;
- содержит развитые инструменты отладки;

- позволяет интегрироваться с системами управления версиями;
- предоставляет инструменты тестирования Karma, Mocha, Protractor и Jest;
- может работать в режиме Live Edit (обновляет редактируемую страницу в окне браузера в реальном времени).

Как и PhpStorm, компания JetBrains позиционирует WebStorm в качестве платного продукта.

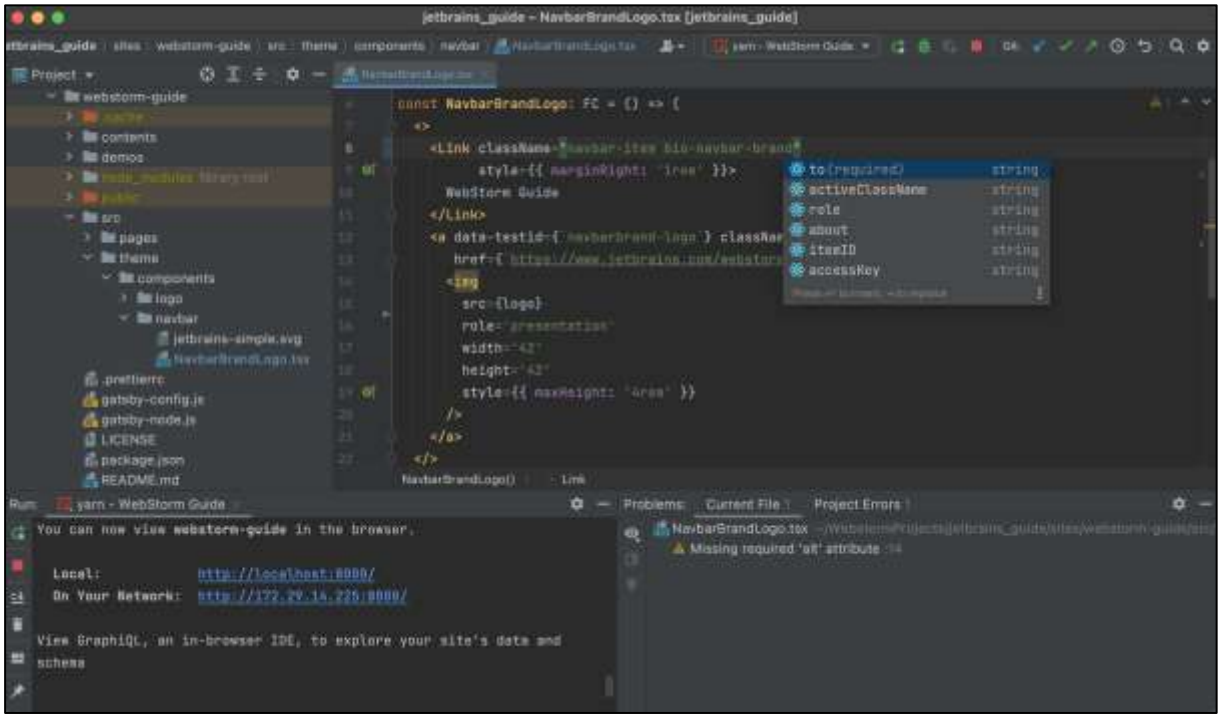


Рис. 1.97. Интерфейс IDE WebStorm

6. Android Studio

Android Studio является бесплатной IDE для разработки на базе Android. Среда доступна для платформ Windows, macOS и GNU/Linux. В Android Studio включена поддержка работы с Java, Kotlin, C++.

Android Studio позиционируется как инструмент для разработки приложений для смартфонов и планшетов на базе системы Android. Среда является удобным инструментом как для небольших команд разработчиков, так и крупных организаций.

Основные возможности среды разработки:

- предоставляет шаблоны макетов и компонентов Android;

- содержит редактор макетов UI, создание интерфейса приложения с помощью технологии Drag-and-Drop;
- позволяет вести отладку приложений, осуществлять рефакторинг кода;
- содержит статический анализатор кода Lint, анализирующий проблемы производительности и несовместимости версий;
- осуществляет сборку проектов на базе Gradle.

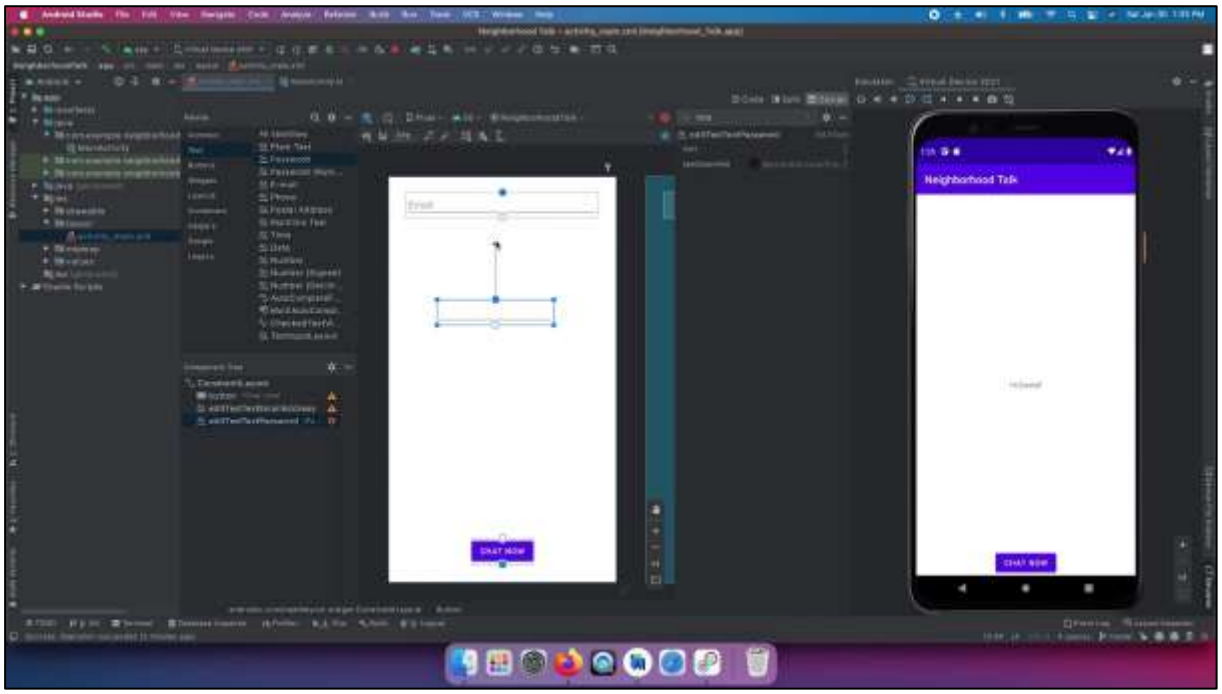


Рис. 1.98. Интерфейс IDE Android Studio

1.5.3. Текстовые редакторы разработчика

Возможности и отличие от IDE

Наряду с интегрированными средами разработки программисты используют профессиональные текстовые редакторы, функционал которых не ограничивается лишь редактированием программного кода (как, например, в приложении Блокнот). Подобные редакторы ускоряют набор кода и минимизируют частоту ошибок, работают с документами, используя встроенный проводник, управляют кодировками файлов, автоматизируют некоторые процессы компиляции программ.

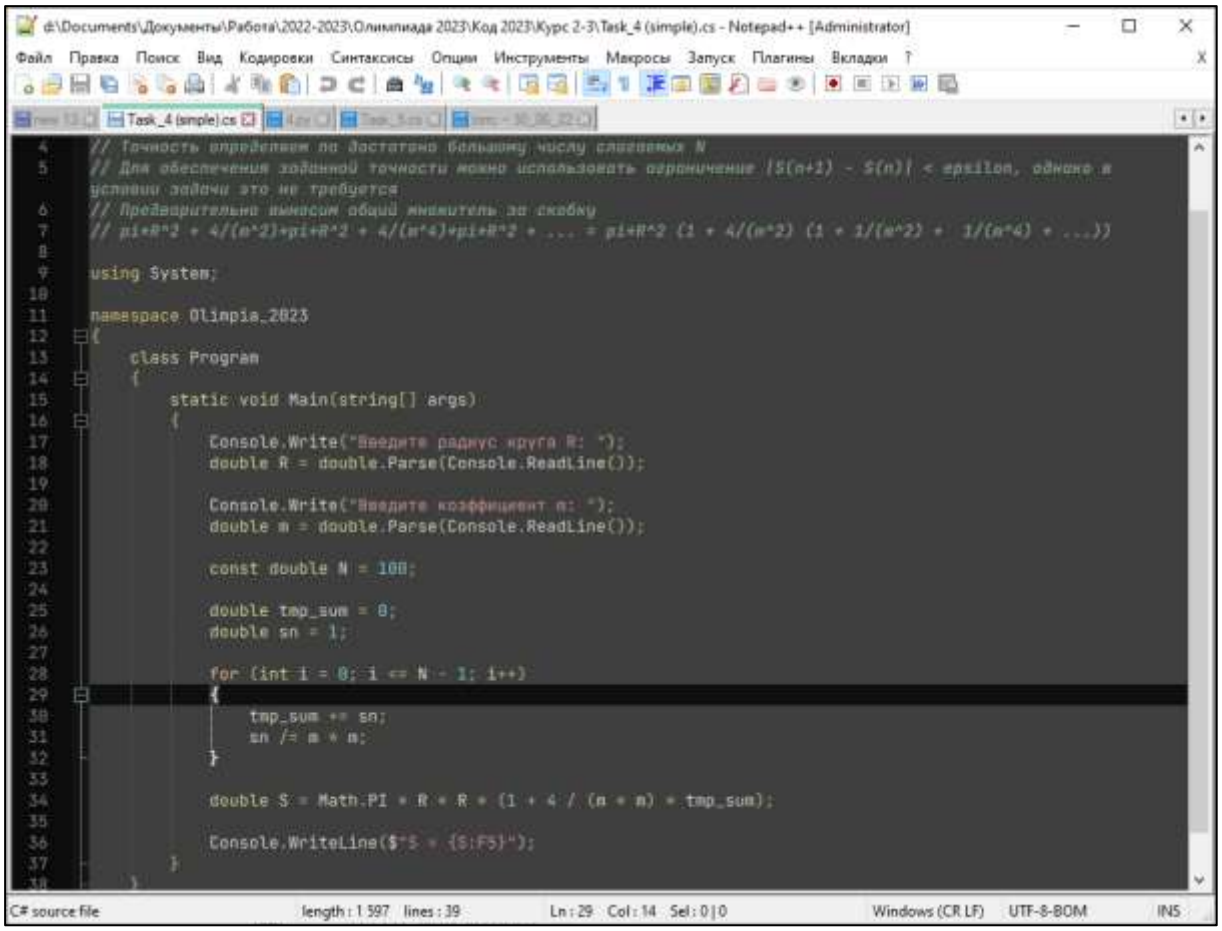


Рис. 1.99. Текстовый редактор Notepad++

В отличие от сред разработки, текстовые редакторы не требовательны к ресурсу ПК, быстрее подгружают процессы и в целом более стабильны в работе. При этом зачастую функционал редактора может незначительно уступать IDE. Опытные пользователи могут настроить текстовый редактор, сделав из него полноценную систему разработки.

Это полезно знать!

Говоря далее о текстовых редакторах, мы подразумеваем программы, которые работают с обычными текстовыми файлами как последовательностями символов.
Текстовые редакторы и процессоры, предназначенные для создания электронных документов с последующим форматированием (например, настольные офисные системы) в этом курсе не затрагиваются.

Выделим основные возможности, которыми обладают многие современные текстовые редакторы:

- подсветка синтаксиса команд и управление отступами;
- всплывающие подсказки и автозавершение кода;
- деление области на несколько колонок;
- мини-карта файла с кодом;
- настройка интерфейса и шрифта;
- встроенный проводник (обозреватель проекта);
- возможность подключения сторонних плагинов;
- контроль версий;
- плагин быстрого набора Emmet;
- консоль логов и работы с командной строкой;
- возможность подключения отладчиков;
- настройка горячих клавиш.

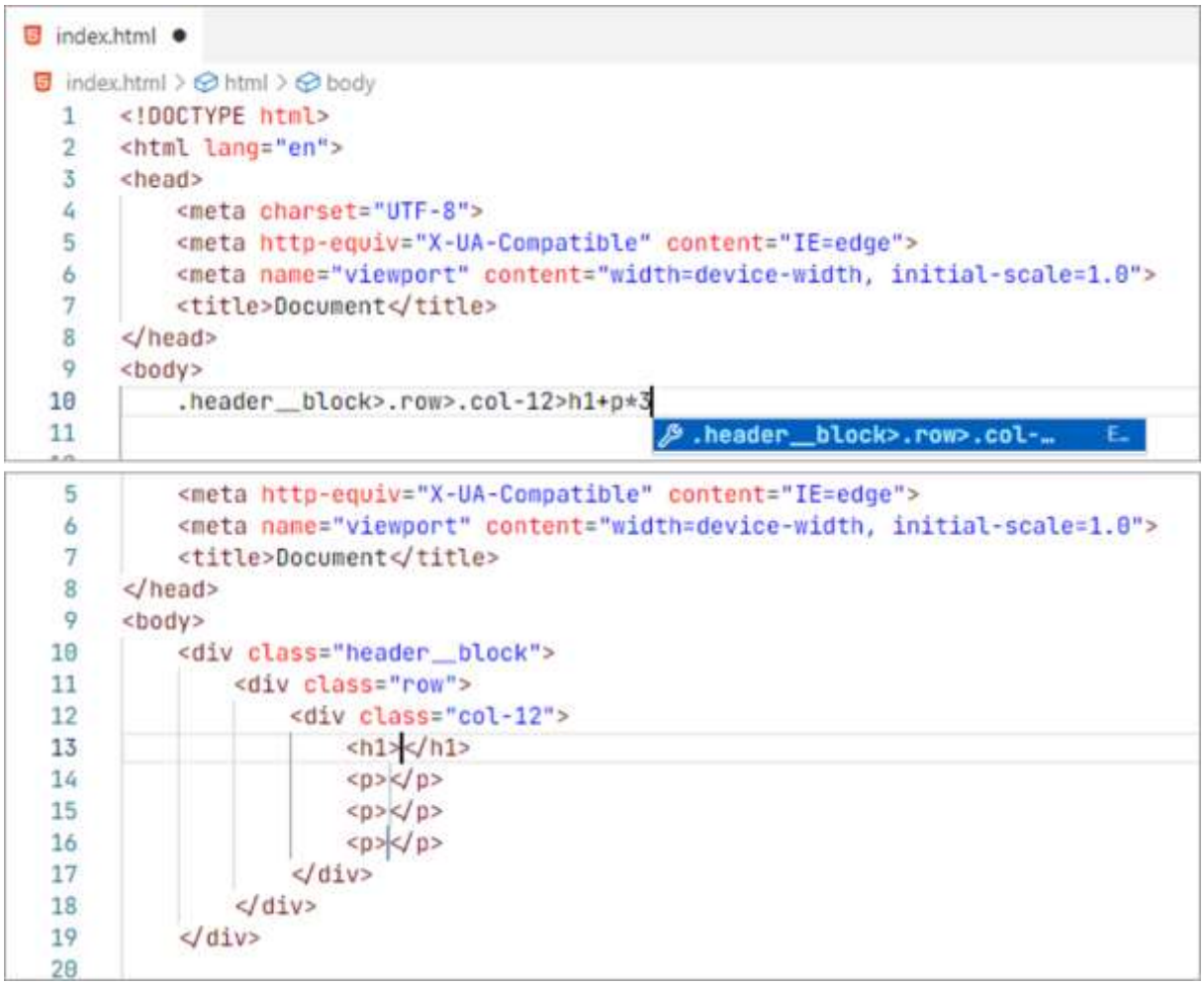


Рис. 1.100. Плагин Emmet для быстрой разметки на языке HTML

Примеры текстовых редакторов для разработки

1. Visual Studio Code

В отличие от среды разработки, редактор *Visual Studio Code* отличается небольшим потребляемым ресурсом ПК и богатой функциональностью Важным преимуществом является его бесплатное распространение. Редактор можно использовать на платформах семейства Windows, macOS, Linux.

Visual Studio Code поддерживает подсветку синтаксиса многих используемых языков программирования, имеет встроенную технологию IntelliSense и плагин Emmet. Редактор является отличным инструментом для веб-разработчиков (в частности, для работы с HTML, CSS и JavaScript установка дополнительных плагинов не требуется).

Visual Studio Code дает разработчику гибкие возможности настройки интерфейса и функционала, горячих клавиш, имеет встроенные возможности подключения к Git-репозиториям, содержит инструменты рефакторинга. Расширить возможности Visual Studio Code позволяют многочисленные плагины, скачиваемые и устанавливаемые автоматически.

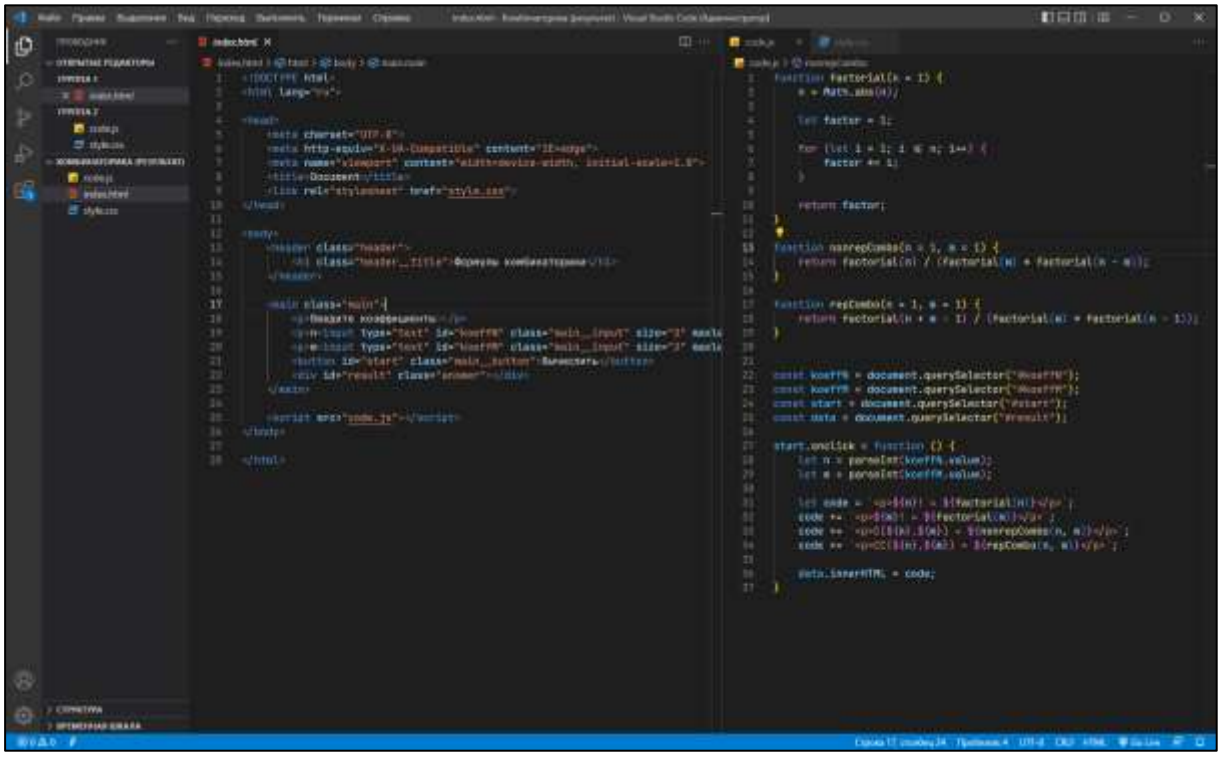


Рис. 1.101. Редактор Visual Studio Code в веб-разработке

2. Sublime Text

Sublime Text – редактор для веб-разработчиков, написанный на языках C++ и Python. Поддерживается на платформах Windows, Linux и Mac. Разработчикам доступна бесплатная ознакомительная версия. Для коммерческого использования приобретается лицензия.

Как и Visual Studio, Sublime Text подсвечивает синтаксис большинства используемых языков программирования.

Sublime Text является популярным редактором по ряду причин:

- имеет простой и интуитивно понятный интерфейс, а для опытных пользователей доступна гибкая настройка с помощью Package Control и работа с терминалом;
- предоставляет плагин Emmet;
- поддерживает различные опции быстрого и множественного выделения и редактирования;
- позволяет создавать сниппеты (фрагменты вставляемого кода);
- отображает мини-карта кода (для перемещения по большим файлам);
- хорошо оптимизирован и быстро загружается;
- может быть расширен дополнительными плагинами.

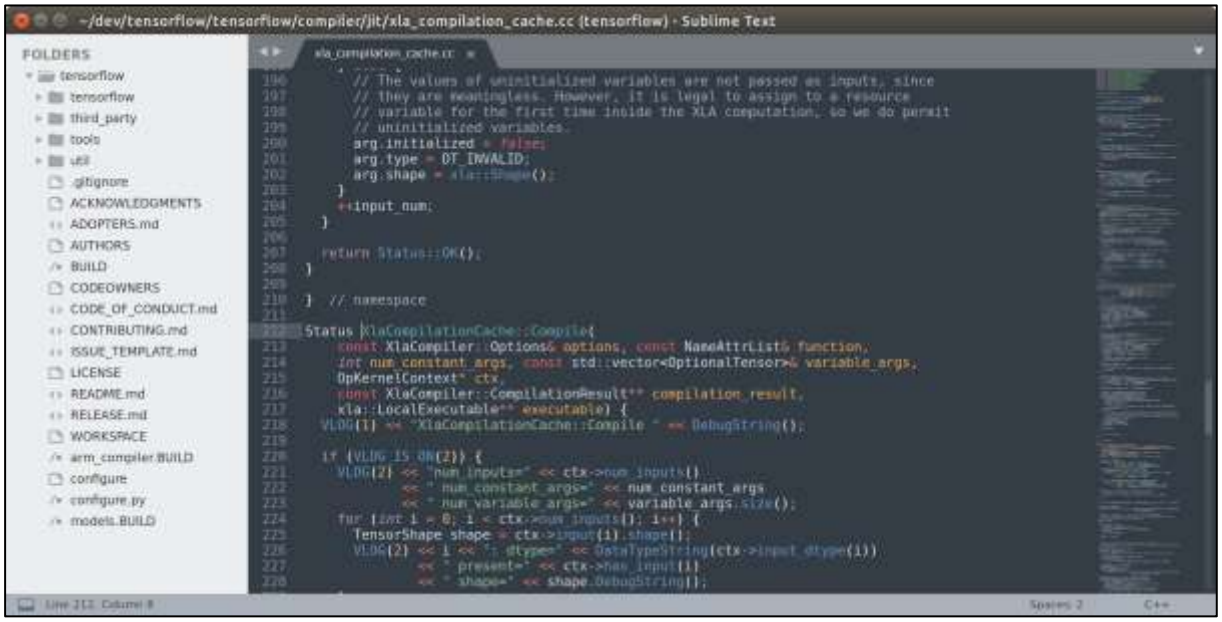


Рис. 1.102. Редактор Sublime Text

3. Atom

Atom – это бесплатный текстовый редактор от сообщества GitHub с открытым программным кодом. Редактор реализован на базе языков C++, JavaScript, HTML и CSS. Доступен на платформах Windows, Linux и Mac.

Atom поддерживает широкий спектр языков программирования и может стать серьезным подспорьем в работе программиста. Редактор обладает тем же набором функций, что и его ближайший конкурент Sublime Text.

Среди возможностей отметим следующие:

- поддерживает плагины Emmet, Autoprefixer, автоматическое форматирование кода;
- настройки осуществляются в визуальном редакторе, а не текстовом файле формата JSON;
- плагины написаны на Node.js, что удобно при их настройке;
- позволяет писать код нескольким разработчикам совместно в режиме реального времени;
- демонстрирует хорошие показатели быстродействия.

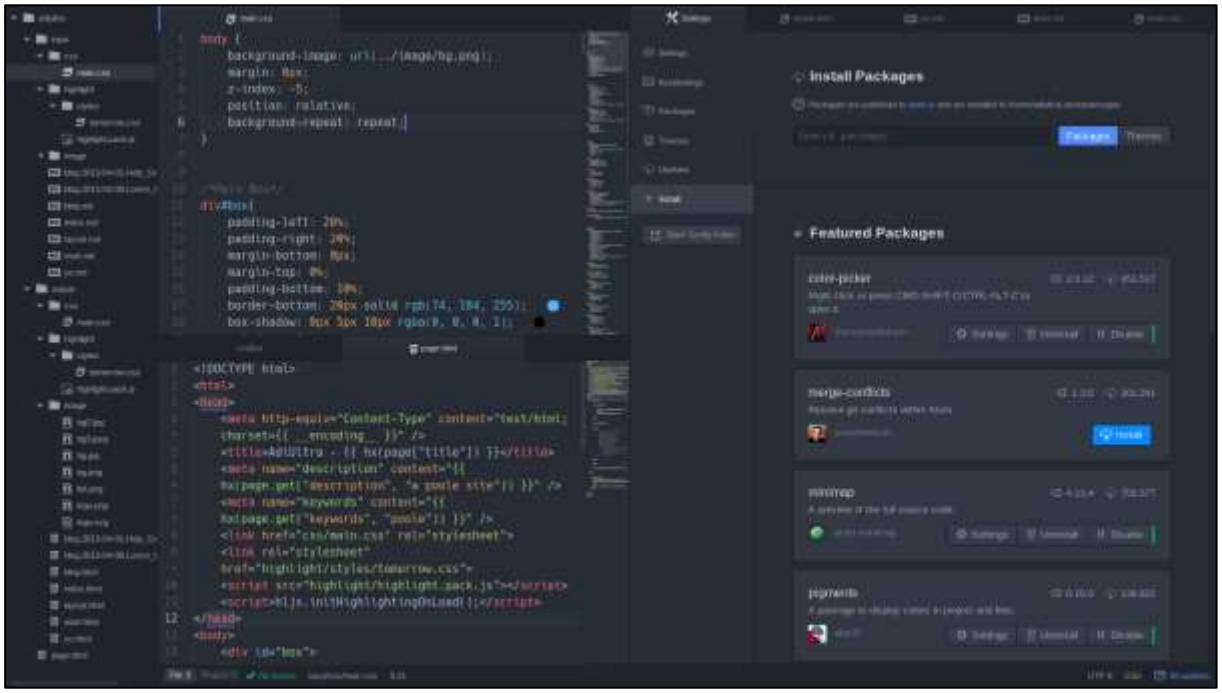


Рис. 1.103. Редактор Atom

4. Coda

Coda – один из самых популярных текстовых редакторов для пользователей Mac. Редактор хорошо оптимизирован и обладает всеми необходимыми в веб-разработке функциями: подсветка синтаксиса, сворачивание блоков кода, поиск и замена, функции автозавершения, интеграция с Git и Subversion.

Среди собственных возможностей Coda отметим:

- подключение по FTP, SFTP, WebDAV;
- упрощенный поиск по файлам с использованием регулярных выражений;
- встроенный терминал и редактор MySQL;
- ряд инструментов для работы с PHP;
- валидация разметки HTML, CSS и кода на JavaScript.

Главным недостатком редактора Coda является его распространение по лицензии.

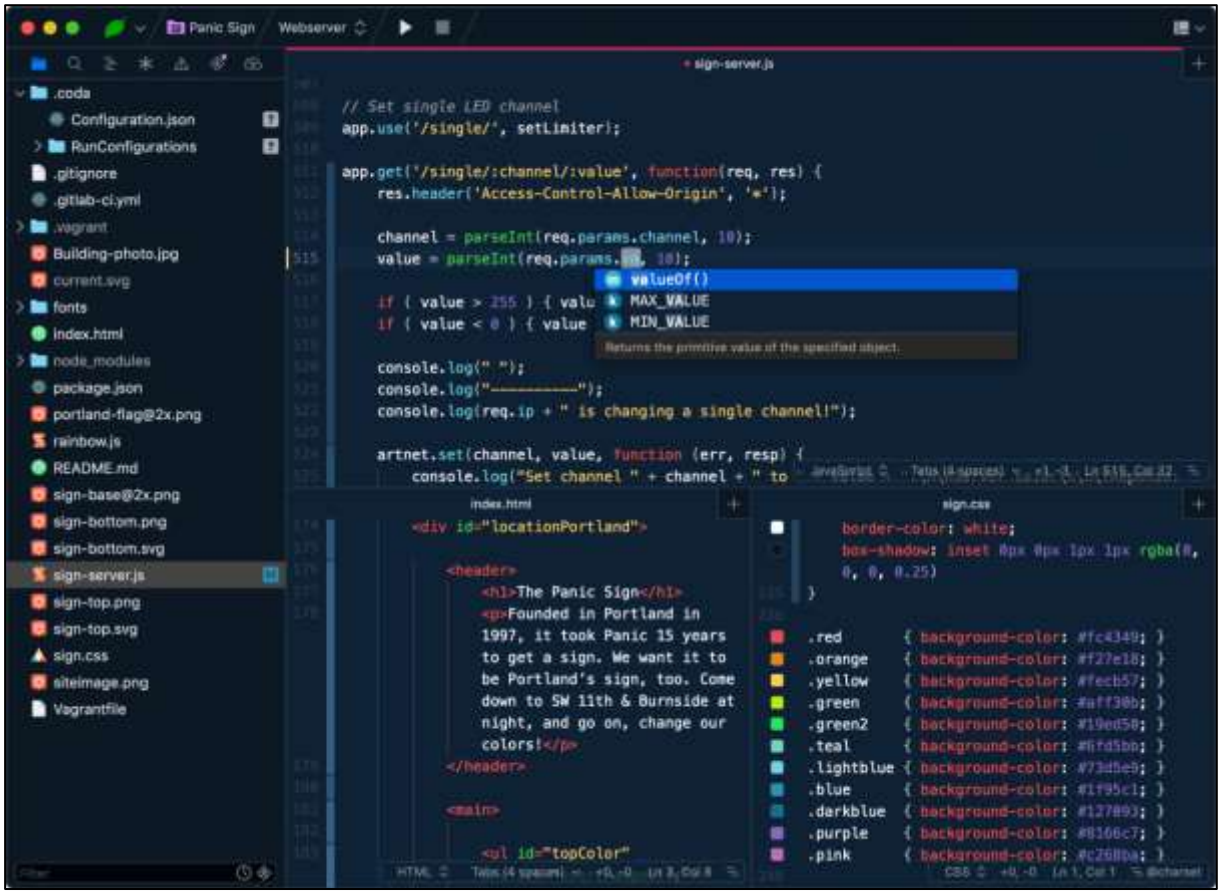


Рис. 1.104. Редактор Coda

5. Brackets

Brackets позиционируется как один из лучший бесплатных редакторов с для веб-разработки. Brackets разработан компанией Adobe Systems в 2014 году и был нацелен на frontend-разработку.

Основными языками в Brackets являются HTML, CSS и JavaScript. Встроенные инструменты редактора позволяют ускорить операции с CSS. Brackets обладает JavaScript-отладчиком Theseus и собственным локальным веб-серверов, необходимым для тестирования проектов.

Brackets синхронизируется с Git и поддерживает установку расширений. Особенно интересными являются средства быстрого редактирования: можно менять несколько элементов кода одновременно.

С помощью плагина Live Preview любые изменения в коде в автоматически обновляют страницу в браузере.

Среди недостатков отмечаются некоторые проблемы с отображением кириллических шрифтов.

Рис. 1.105. Редактор Brackets

6. Vim

Vim является бесплатным текстовым редактором для программистов. В отличие от многих других программ своего класса, Vim специфичен и работа с ним требует подготовки. Почти все операции в редакторе осуществляются с помощью команд клавиатуры.

Vim поддерживает несколько режимов работы, среди которых два основных:

- режим ввода текста, необходимый для набора кода;
- режим команд, предназначенный для управления текстом кода.

Оба режима работают параллельно и переключение между ними осуществляется по нажатию клавиш.

Несмотря на достаточно высокий порог вхождения, Vim предоставляет разработчику множество полезных возможностей:

- подсветка синтаксиса для более чем 200 языков и автозавершение ключевых слов;
- работа с несколькими файлами в параллельных областях;
- сравнение нескольких файлов;
- реализует цикл разработки по схеме «редактирование – компиляция – исправление»;
- интеграция с операционной системой;
- преобразование файлов разных форматов;
- поддержка визуального режима редактирования;
- возможность использования графического интерфейса.

Рис. 1.106. Редактор Vim

7. NotePad++

Notepad++ – это бесплатный текстовый редактор с открытым программным кодом, ориентированный на разработчиков и верстальщиков веб-страниц.

NotePad++ является одной из наиболее продвинутых альтернатив встроенному приложению Блокнот: редактор бесплатен, нетребователен к ресурсу ПК и позволяет осуществлять различные преобразования с текстом и файлами, свойственные профессиональным текстовым редакторам.

Среди основных возможностей NotePad++ отметим:

- подсветка синтаксиса команд множества языков программирования и разметки;
- автоматические завершение команд, закрытие парных тегов, скобок, кавычек;
- сворачивание кода в блоки;
- гибкий поиск;
- работа с макросами;
- мини-карта документа;
- настройка сочетания горячих клавиш;
- многочисленное редактирование;
- подключение и ручная настройка плагинов.

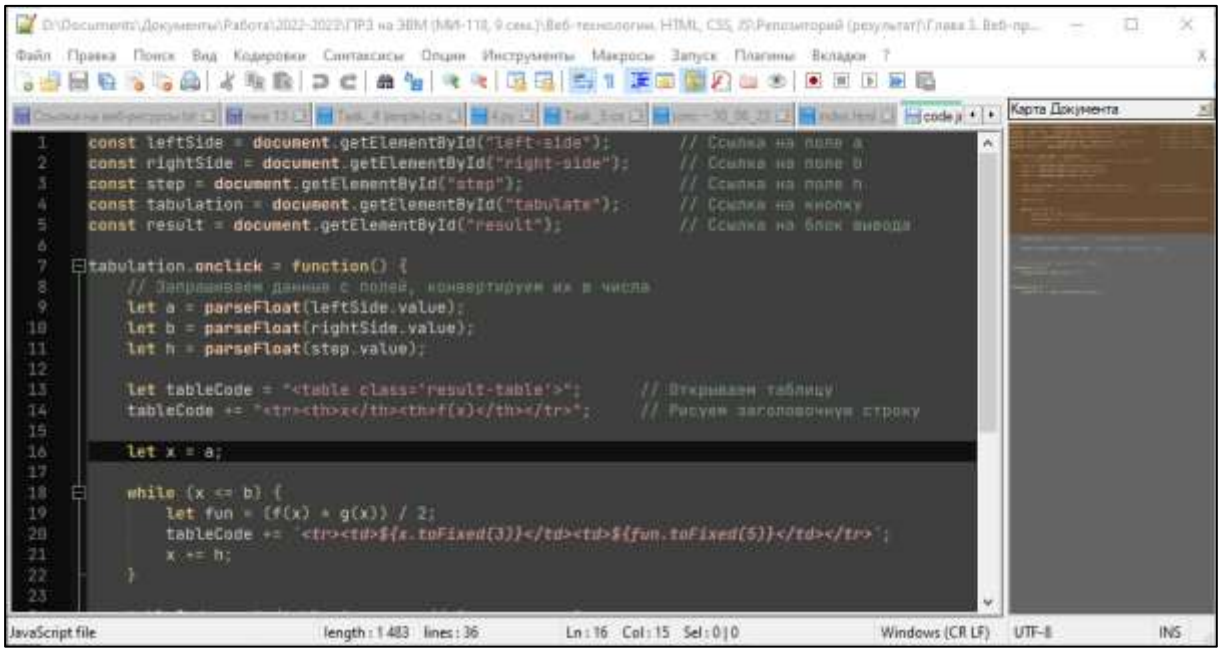


Рис. 1.107. Редактор NotePad++

1.5.4. Графические редакторы

Возможности и значение

Создание дизайна современного сайта требует работы с *графическими редакторами*, которые позволяют детально проработать внешний вид веб-страниц, отдельных графических элементов и даже сгенерировать код разметки.

Редакторы графики необходимы для решения следующих задач веб-дизайна:

- обработка растровой и векторной графики;
- создание и редактирование изображений, графических элементов, наполняющих веб-страницы;
- разработка дизайна прототипов веб-сайтов и отдельных веб-страниц;
- автоматизация ряда рутинных операций по переносу дизайна в программный код.

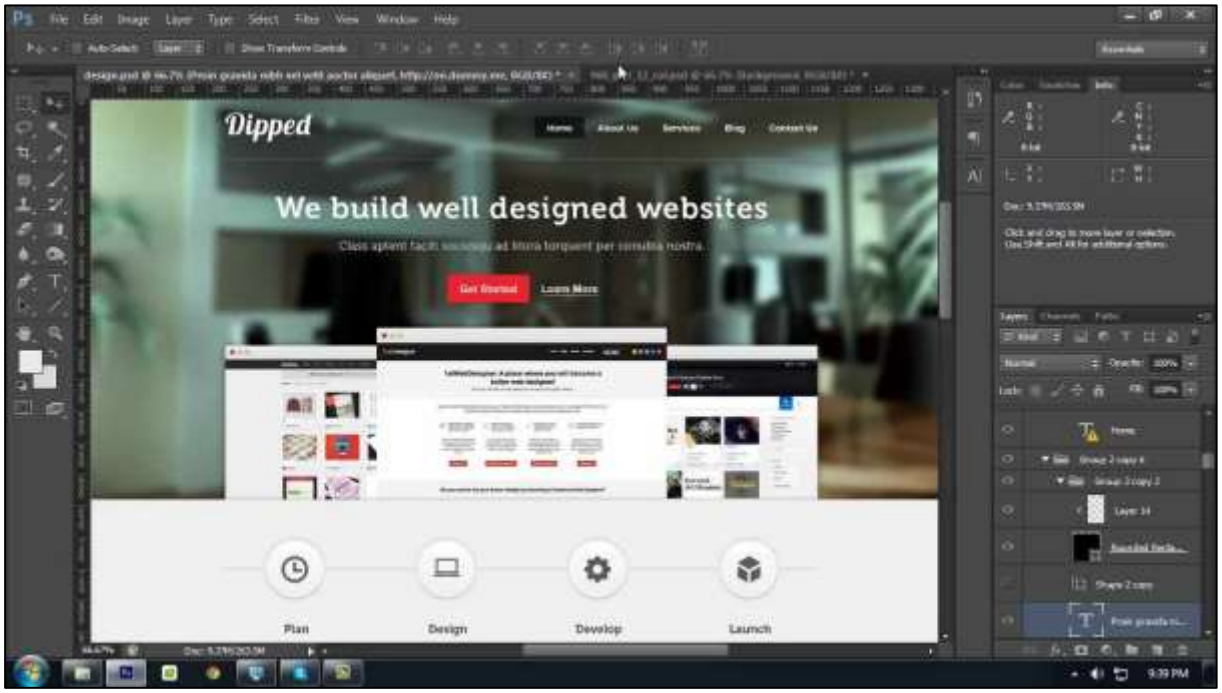


Рис. 1.108. Создание дизайна веб-страницы в графическом редакторе

Примеры графических редакторов для веб-дизайна

Редакторы компании Adobe

Adobe Photoshop является одним из первых редакторов для профессиональной обработки растровых изображений. Благодаря развитому функционалу редактор остается одним из наиболее востребованных, несмотря на достаточно высокий порог вхождения.

Photoshop часто используется для подготовки контента веб-страниц: изображений, фотографий, логотипов, иконок, рекламных баннеров и других графических объектов. Photoshop также имеет большой набор функций для рисования с помощью графического планшета.

Полезной возможностью, которую дает Photoshop веб-дизайнеру, является набор инструментов для разметки прототипов или дизайна будущих веб-страниц.

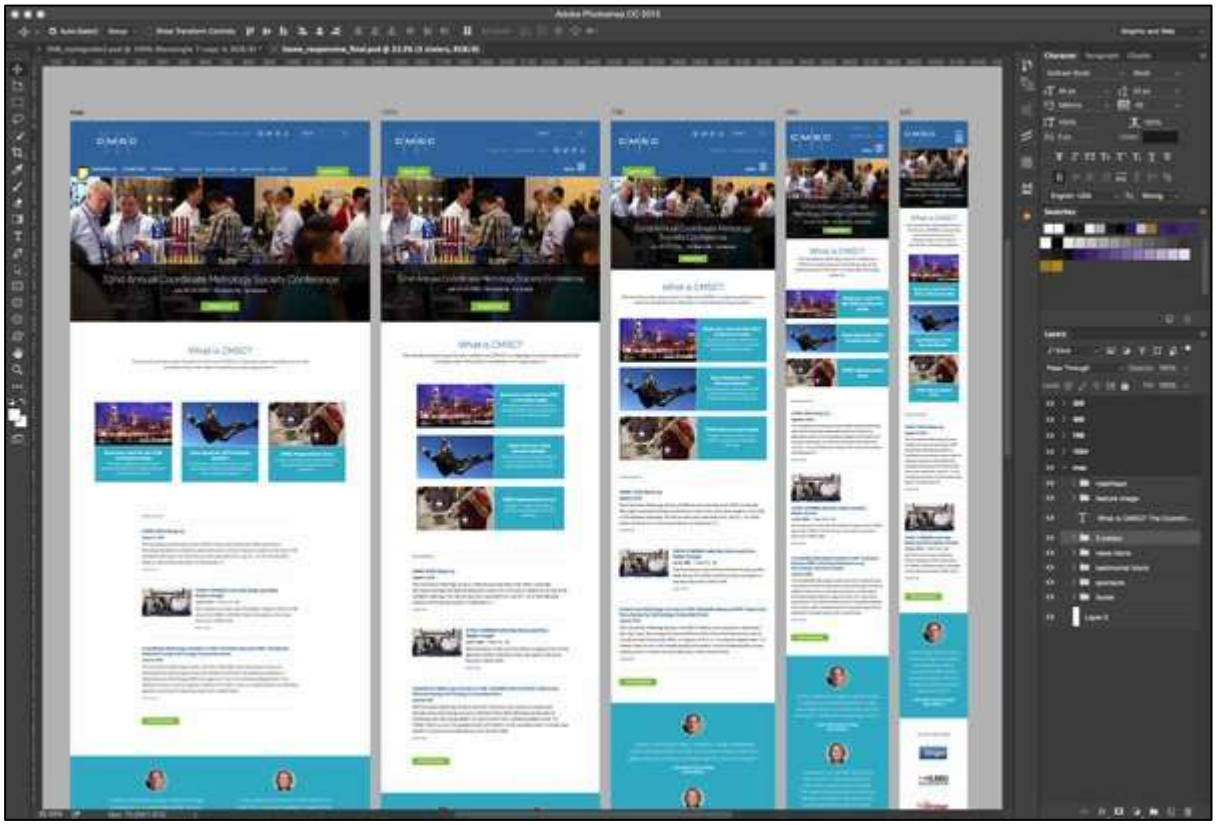


Рис. 1.109. Разметка дизайна веб-страницы с учетом адаптивности

Adobe XD предназначен для работы с векторной графикой, которая необходима при разработке UI/UX веб-дизайна. Приложение Adobe XD удобно в качестве инструмента конструирования каркаса и прототипа веб-сайта. Возможности редактора также часто востребованы при создании компьютерных игр. Adobe XD при необходимости может быть интегрирован с другими программами Adobe.

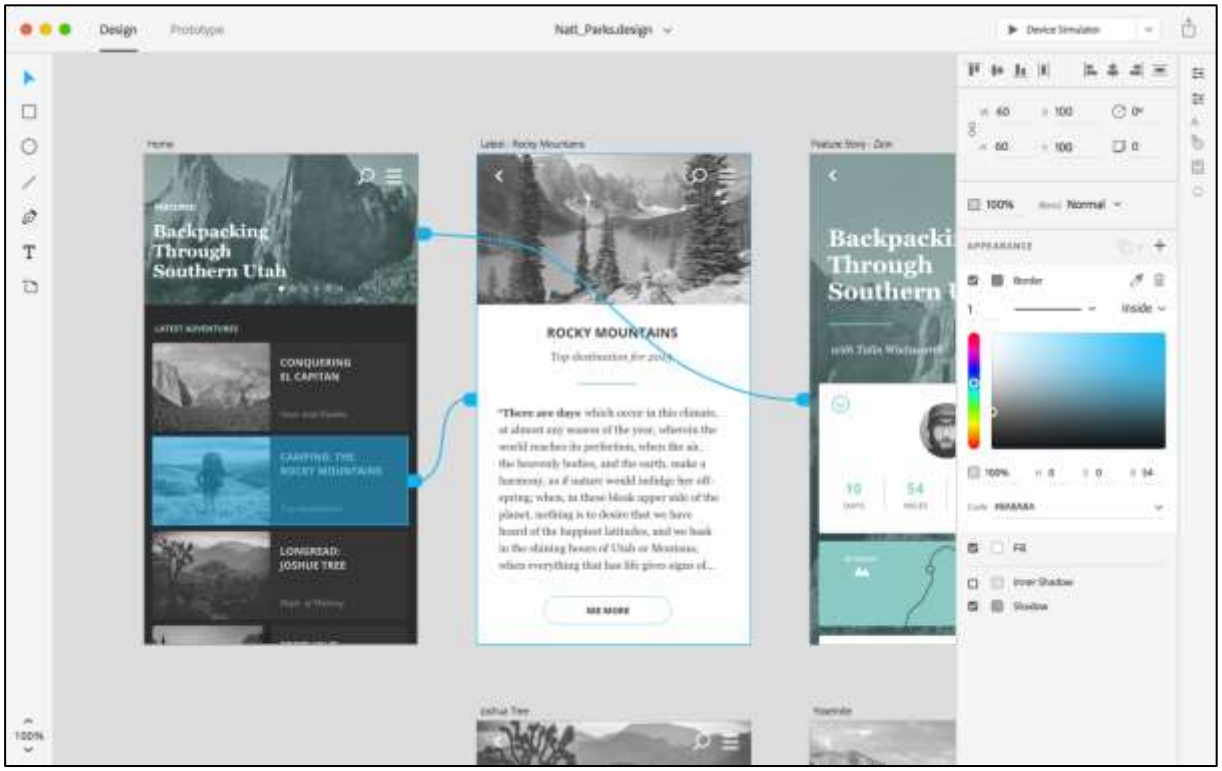


Рис. 1.110. Adobe XD в создании прототипа веб-сайта

Adobe Illustrator является мощным инструментом для создания иллюстраций, которые востребованы при разработке сайтов для оказания коммерческих услуг и социальных сетей. Illustrator поддерживается на iPad и других планшетах, совместим с операционными системами линейки Windows и Mac OS.

Как и многие продукты Adobe, перечисленные программы являются платными. На рис. 1.112 представлены некоторые платные и бесплатные аналоги программ Adobe, имеющие схожий функционал.

Рис. 1.111. Adobe Illustrator в проработке дизайна веб-страницы

Рис. 1.112. Аналоги программ линейки Adobe

Сделать веб-сайт более динамичным позволяет использование 3D-моделирования. Cinema 4D совместно с Adobe Affects позволяют реализовать motion-дизайн и создавать веб-анимации.

Cinema 4D помогает создавать реалистичные или мультипликационные спецэффекты, 3D-персонажей. Программа отличается простым и понятным интерфейсом, для начинающего пользователя доступна система подсказок.

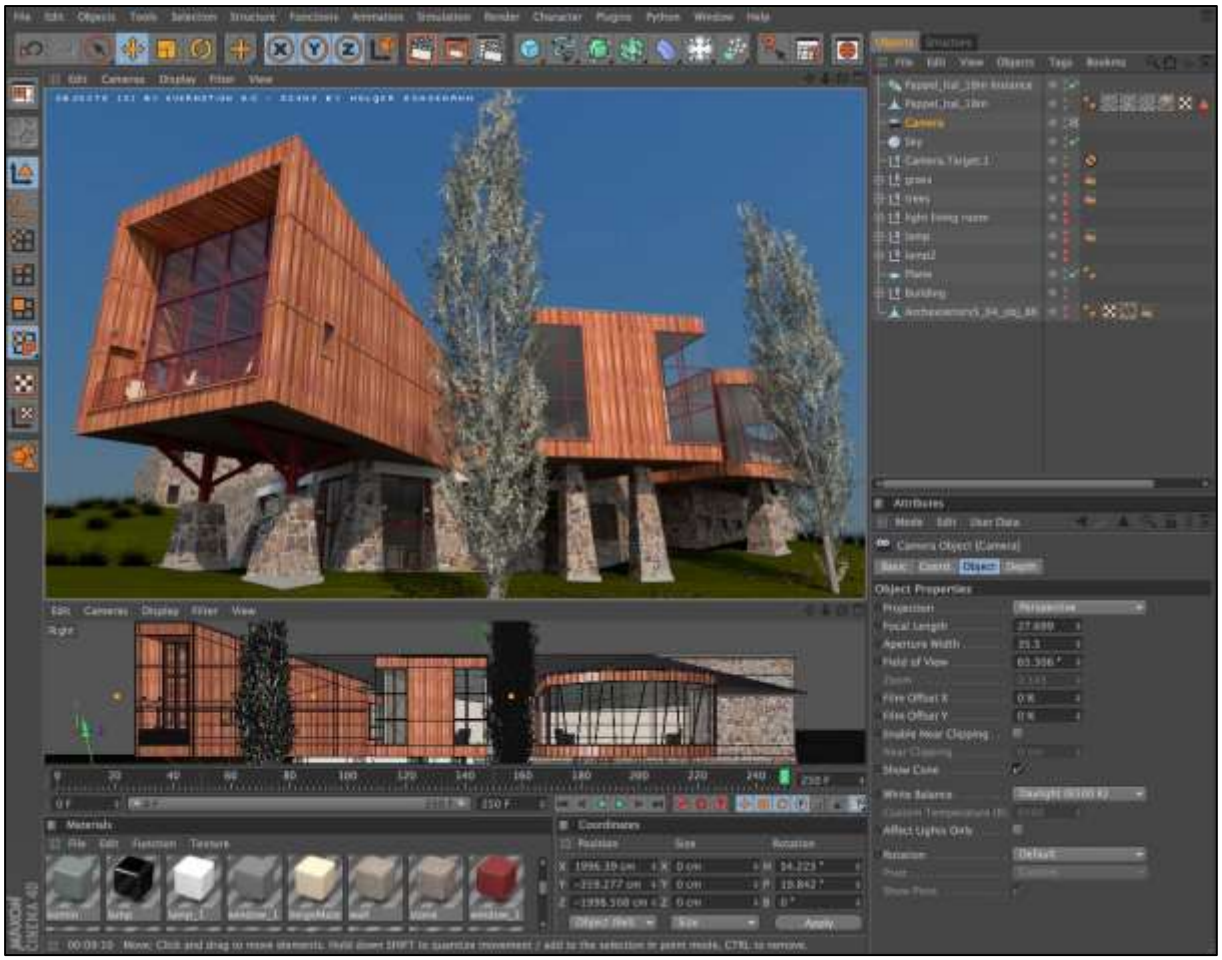


Рис. 1.113. Cinema 4D в реализации 3D-модели

Adobe After Effects поможет объединить созданные модели и динамические изображения в единый ролик и наложить эффекты. Программа осуществляет цветокоррекцию и в целом предоставляет постобработку, предоставляя заготовки превращается в полноценную анимацию или рекламный ролик.

Рис. 1.114. Adobe After Effects в создании анимационных роликов

Редакторы растровой и векторной графики

CorelDRAW – это популярный коммерческий редактор векторной и растровой графики. CorelDRAW часто используется для создания векторных изображений: логотипов, визиток, иконок, художественных элементов и т.д. Последние версии программы ориентированы на операционную систему Windows.

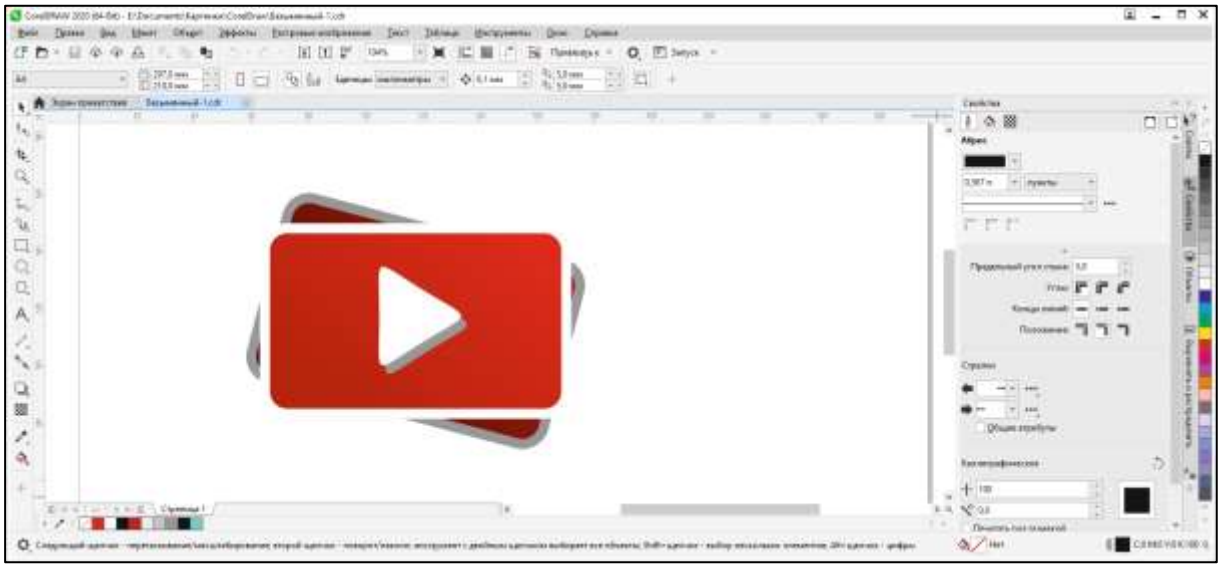


Рис. 1.115. Создание логотипа сайта в CorelDRAW 2020

Inkscape является векторным редактором, позволяющим работать с художественной и технической графикой. Inkscape позиционируется в качестве бесплатного аналога популярным коммерческим приложениям Adobe Illustrator и CorelDRAW. Inkscape находит свое применение и веб-графике.

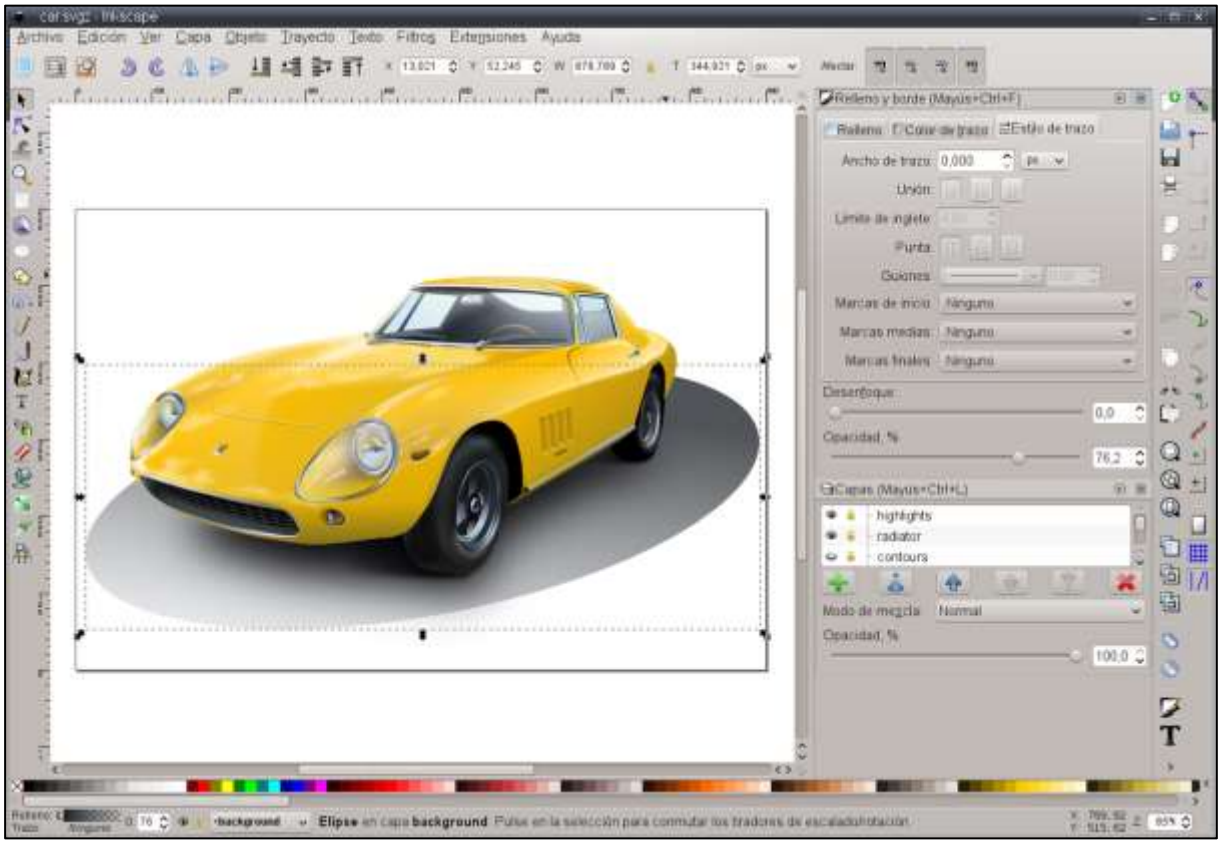


Рис. 1.116. Создание векторного изображения в Inkscape

Графический редактор *GIMP* является бесплатным редактором растровой графики. Он позволяет работать с кистями, шаблонами, фильтрами, поддерживает различные форматы графических файлов. Дополнительные модули позволяют расширять стандартный функционал редактора.

GIMP часто позиционируется в качестве бесплатного аналога Adobe Photoshop. Редактор содержит продвинутый функционал для обработки фотографий и создания графических файлов, логотипов, рисунков. Кроме того, Gimp поддерживает частичную работу с векторной графикой.

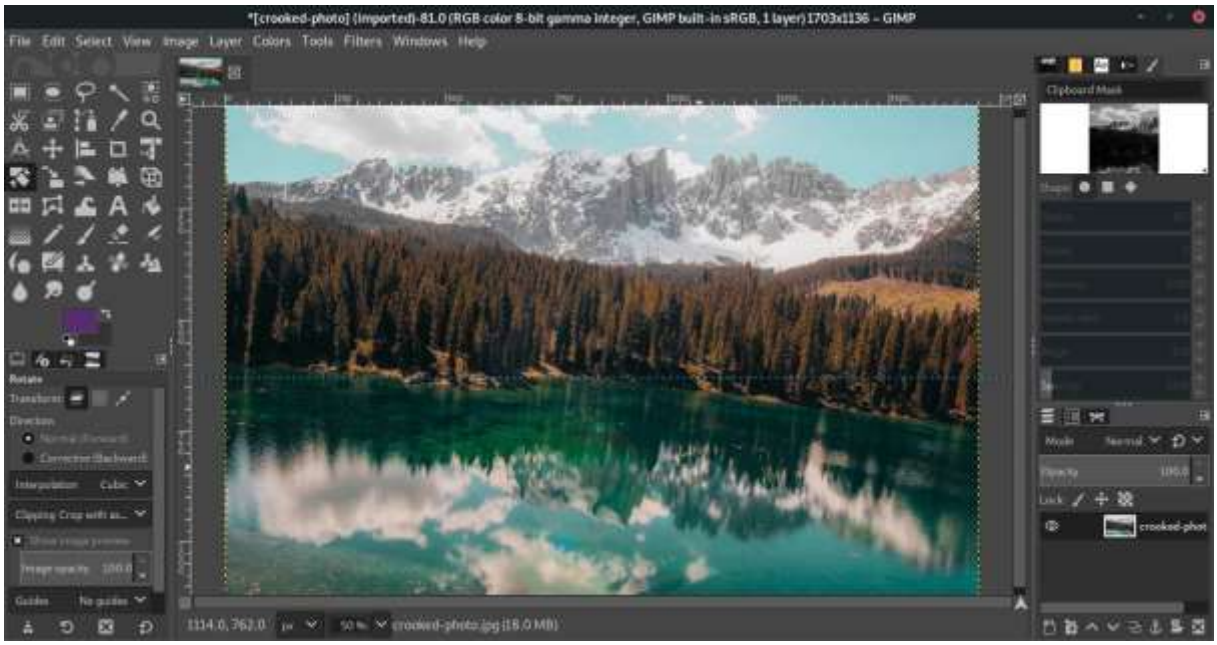


Рис. 1.117. Обработка изображений в GIMP

1.5.5. Инструменты проектирования интерфейса

Инструменты для создания прототипов

Важность прототипирования

Прототипирование лежит в основе проектирования UI и UX (пользовательский интерфейс и функционал). В зависимости от поставленных задач прототипы могут иметь различную степень детализации. Главная цель создания прототипа – предоставить возможность пользователю или заказчику увидеть концепцию дизайна будущего сайта. Для разработчика прототип – это возможность протестировать идеи до того, как они будут реализованы в готовом проекте.

Примеры программ для создания прототипов

Sketch – это программа, помогающая разрабатывать прототипы и веб-интерфейсы. В отличие от обычных графических редакторов, хорошо оптимизирован и ориентирован именно на веб-дизайн. Sketch позволяет быстро оформлять блоки сайта, при необходимости задавая им точное положение и размеры. Недостатками Sketch является совместимость только с Mac OS и платная подписка.

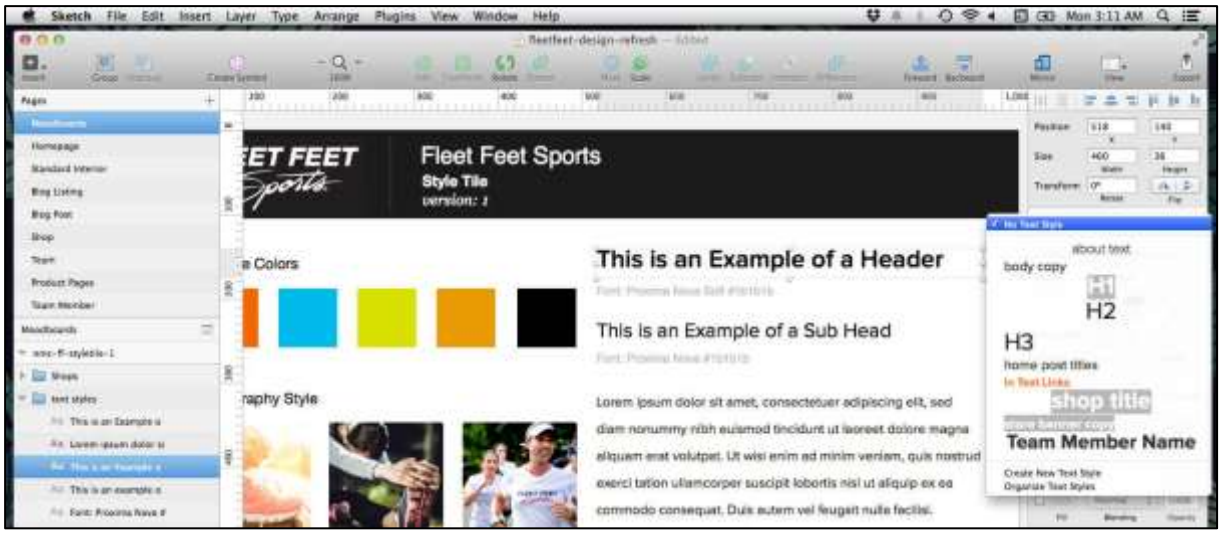


Рис. 1.118. Sketch в разметке блоков веб-страницы

Avocode позволяет переносить дизайн веб-сайта с графического объекта или макета в спецификацию с изображениями и CSS-стилями. Редактор в некотором смысле является мостом между дизайнерами интерфейса и разработчиками, поскольку автоматизирует часть рутинной работы с кодом.

Avocode упростит работу с вёрсткой, стилями, цветами, шрифтами и размерами блоков и элементов, а также и сохранит историю изменений файла. Редактор является платным, однако подписка сравнительно недорогая.

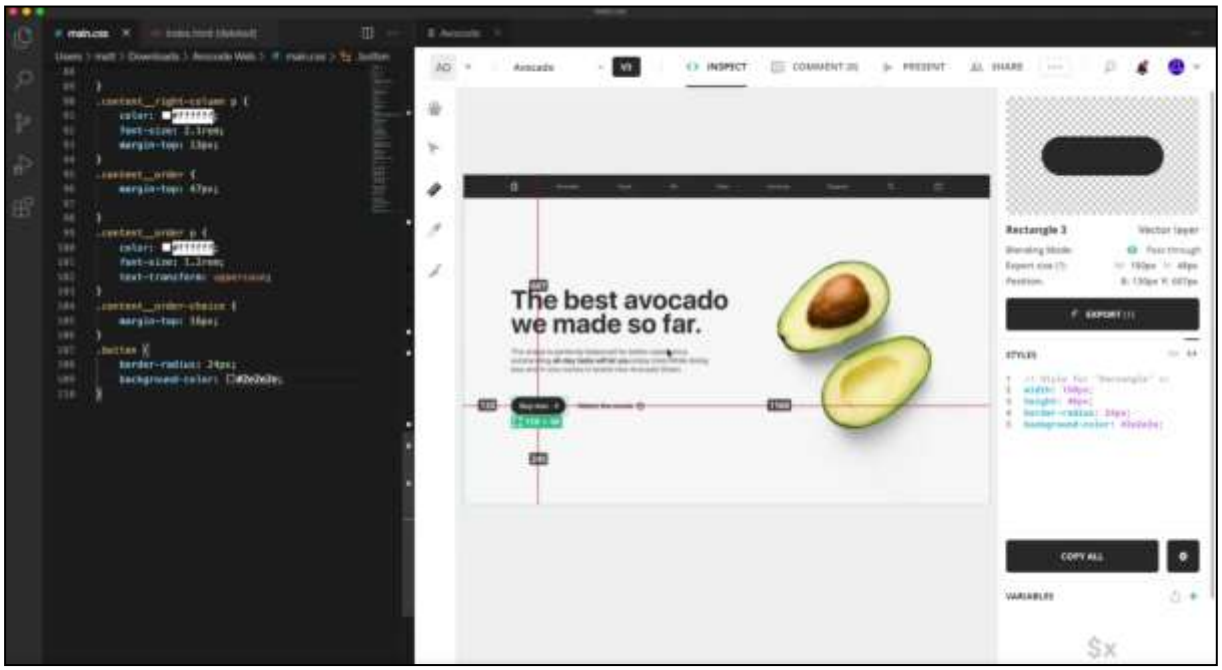


Рис. 1.119. Avocode генерирует CSS-код оформления блоков по макету

Moqups – это веб-платформа, позволяющая разрабатывать дизайн прототипов приложений, веб-сайтов и макетов. Платформа также предоставляет инструменты для «мозгового штурма» и создания интеллектуальных карт.

Прототипы в Moqups можно создавать с нуля или на базе встроенных шаблонов. Доступны средства для совместной работы в реальном времени.

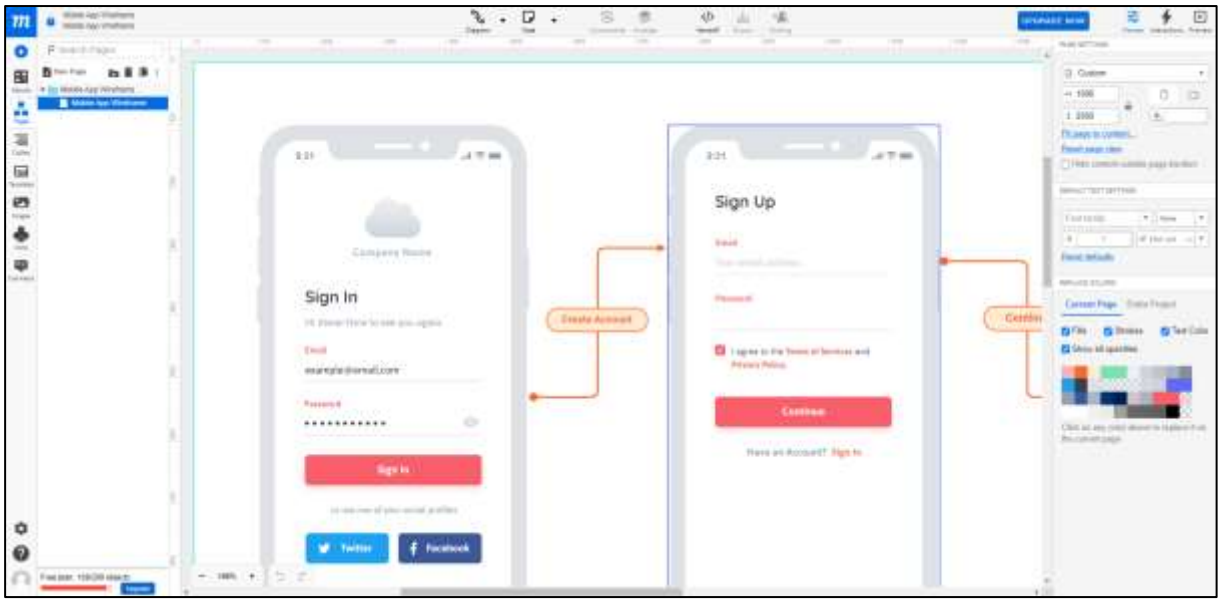


Рис. 1.120. Редактор Moqups в разработке прототипа дизайна мобильного веб-сервиса

Работа с интерфейсом

Обозначим ряд программ, которые упростят подготовку визуального интерфейса веб-сайта или приложения.

Figma – веб-сервис для создания интерфейсов. Редактор Figma предоставляет инструменты конструирования прототипа сайта и интерфейса веб-приложения, а также для участия в совместной работе. Часто Figma используется для подготовки макетов, которые переносятся в будущем на конструкторы сайтов, например, Tilda.

Индивидуальным пользователям доступен бесплатный вариант конструктора. Платная подписка открывает веб-дизайнеру доступ к всему функционалу системы, в частности – средствам коллективной разработки. Figma автоматически сохранять проекты в облаке. Доступна работа с сетками, редактором шрифтов и цвета.

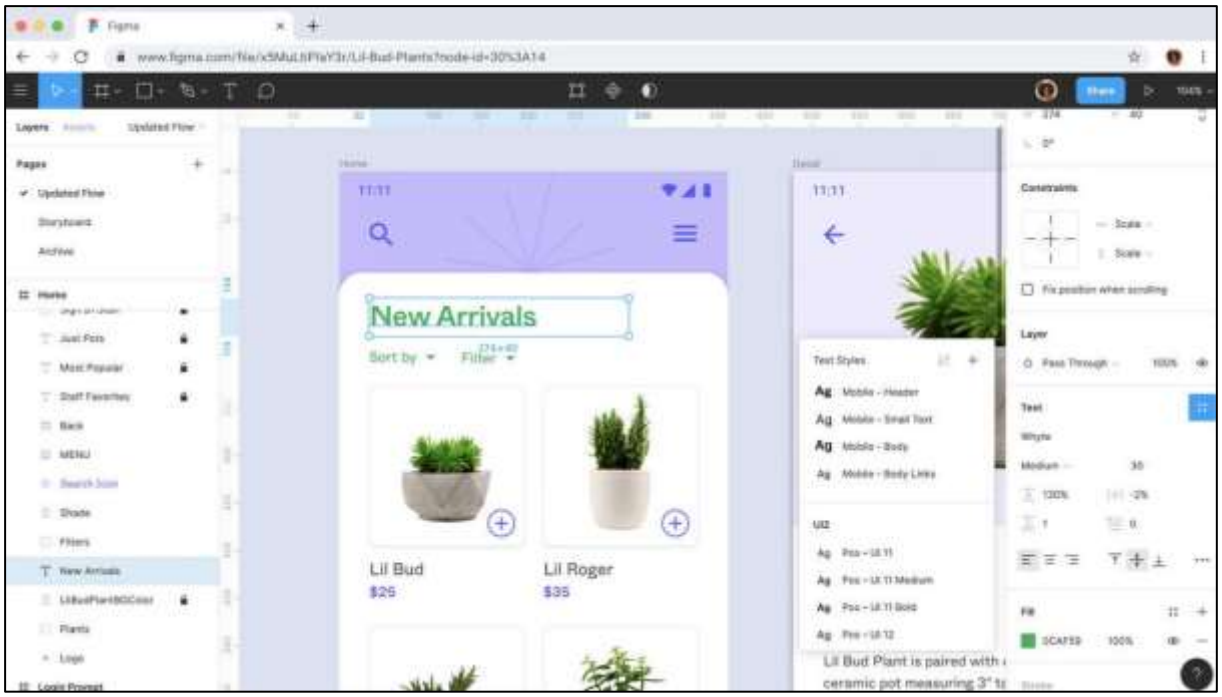


Рис. 1.121. Редактор сервиса Figma

Webflow является не просто редактором, в широком смысле – это онлайн-конструктор сайтов. Webflow объединяет в себе возможности традиционных конструкторов сайтов, систем управления контентом и кодом разметки, набранным веб-программистом. Сервис ориентирован на проекты для маркетинга.

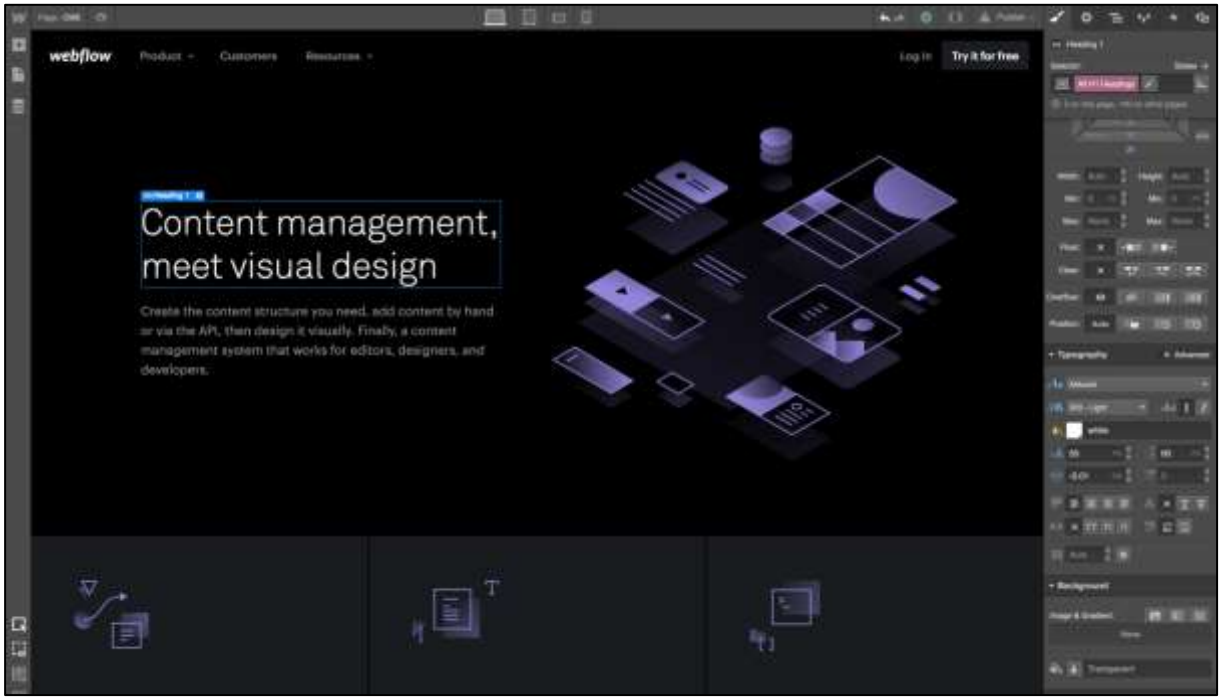


Рис. 1.122. Дизайнер Webflow

1.5.6. Системы контроля версий

Системы контроля версий

Роль систем контроля версий

Определение

Система контроля версий *(**V**ersion **C**ontrol **S**ystems,* ***VCS****) – это система, сохраняющая изменения в файлы проекта в течение определенного времени и позволяющая вернуться к определенной версии предшествовавших изменений.*

Системы контроля версий важны для разработки больших проектов в коллективе разработчиков: они позволяют отслеживать изменения в программном коде с течением времени и при необходимости «откатываться» к предыдущим версиям. Многие популярные IDE и профессиональные текстовые редакторы позволяют синхронизировать инструменты контроля версий с рабочей средой, что существенно упрощает разработку и повышает безопасность развертывания.

Контроль версий позволяет фиксировать все изменения на уровне отдельного разработчика и купировать возникающие проблемы на этапе их возникновения.

Источником возможных проблем обычно являются следующие ситуации:

- изменения в одной части проекта могут быть несовместимы с другой частью ПО;
- часть функционала, реализованная одним разработчиком, несовместима с функционалом другой части проекта;
- в процессе разработки в код внесено изменение, которое содержит ошибки или в целом приводит к возникновению проблем.

Задача ПО для контроля версий – организовать процесс управления версиями проекта, который не ограничивает команду в разработке, но при этом скрыто отслеживает все изменения и сохраняет их на разных этапах. Универсальное ПО для контроля версий может работать с разными платформами и операционными системами, интегрировано с инструментами разработки.

Преимущество использования системы контроля версий

За последние годы VCS-системы претерпели существенные изменения. Наиболее популярной в настоящее время является технология Git – распределенная система контроля версий с бесплатным доступом и открытым программным кодом.

VCS-системы обладают рядом преимуществ:

1. *Хранение истории всех изменений*. VCS-системы хранят историю изменения каждого файла в проекте за длительный период времени. В историю включается информация о изменении кода и удалении файла, сведения об авторе, дате, а также комментарий с указанием цели внесенного изменения. Такая детализация систематизирует подход к анализу причин возникших проблем.
2. *Ветвление и слияние*. Изменения могут носить не только линейный, но и разветвленный характер. На определенном этапе разделение позволяет реализовать несколько отдельных «веток» проекта, каждая независимо друг от друга. Процесс «слияния» позволяет проверить, что внесенные в каждой отдельной ветке изменения не конфликтуют. Наличие нескольких разветвленных проектов позволяет команде разработчиков выбрать наиболее удачно выполненный и запустить его в релиз.
3. *Отслеживаемость*. VCS-система делает процесс разработки и внесения изменений гибким. Каждое изменение комментируется и позволяет понять причины возникновения ошибок, а также получить обоснование реализации фрагмента программы.

Виды систем контроля версий

Локальные системы контроля версий

Самым простым подходом контроля версий разрабатываемого ПО является обычное копирование файлов в отдельный каталог. Для определенности в имени каталога помечается дата редакции или номер версии. Однако этот подход подвержен появлению ошибок: разработчик в силу невнимательности может перепутать каталоги при изменении отдельных файлов.

Подобные проблемы решают VCS с простой базой данных, которая записывает метаданные об изменениях в файлах, что гарантирует контроль ревизий.

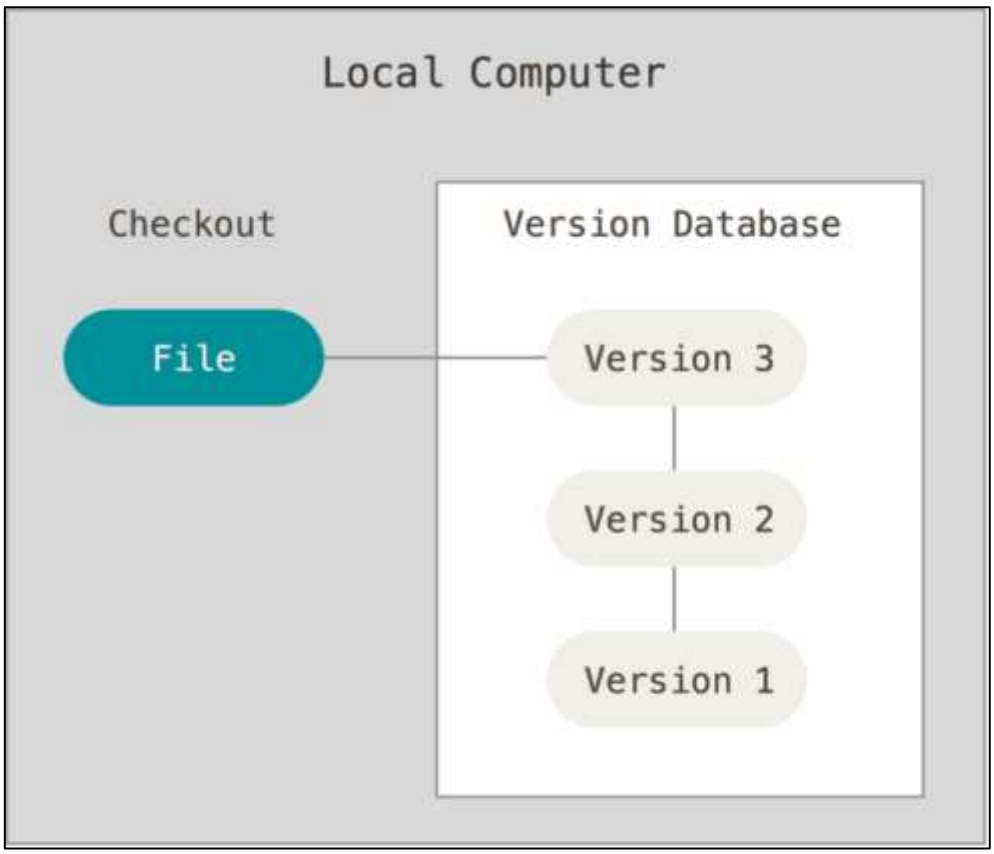


Рис. 1.123. Схема локального контроля версий

Централизованные системы контроля версий

Хранения копий проектов и данных об изменениях далеко не всегда достаточно. Разработчикам в коллективе необходимо непрерывное взаимодействие.

Со временем системы контроля версий были расширены до централизованных CVCS-систем. Как и локальные VCS, CVCS разворачиваются на одном сервере, который содержит все созданные ранее версии файлов. Клиенты получают файлы из централизованного хранилища.

CVCS-системы контроля версий определяли стандарт на протяжении многих лет.

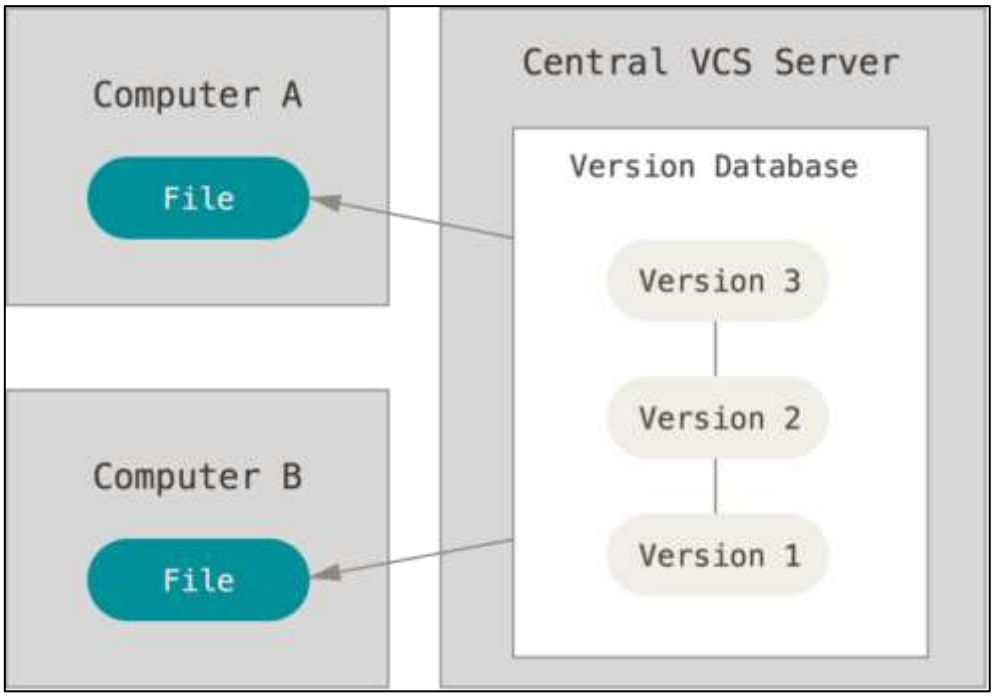


Рис. 1.124. Схема централизованного контроля версий

Централизованная CVCS имеет ряд существенных преимуществ в использовании:

- все разработчики проекта видят разграничение задач и осуществляемую деятельность своих коллег;
- администраторы полностью контролируют права на изменение кода каждого разработчика;
- CVCS удобны в администрировании.

Однако работа с одним сервером порождает серьезный недостаток – система перестает работать в случае отказа сервера. Временный отказ сервера блокирует все операции по сохранению изменений и синхронизации данных разных пользователей. А в случае выхода из строя физических носителей (HDD/SDD) сервера возможна безвозвратная потеря данных о проекте и его версиях. Эта проблема также касается и локальных VCS.

Распределённые системы контроля версий

В распределённых системах контроля версий (DVCS) клиенты работают не просто со «снимком» всех файлов, а получают полные копии репозитория. Такой подход является существенно более безопасным: в случае выхода из строя одного из серверов копии репозитория проекта могут быть получены с машины любого клиента.

Таким образом каждый клиент может обеспечить «бэкап» (восстановление или возврат) всех данных.

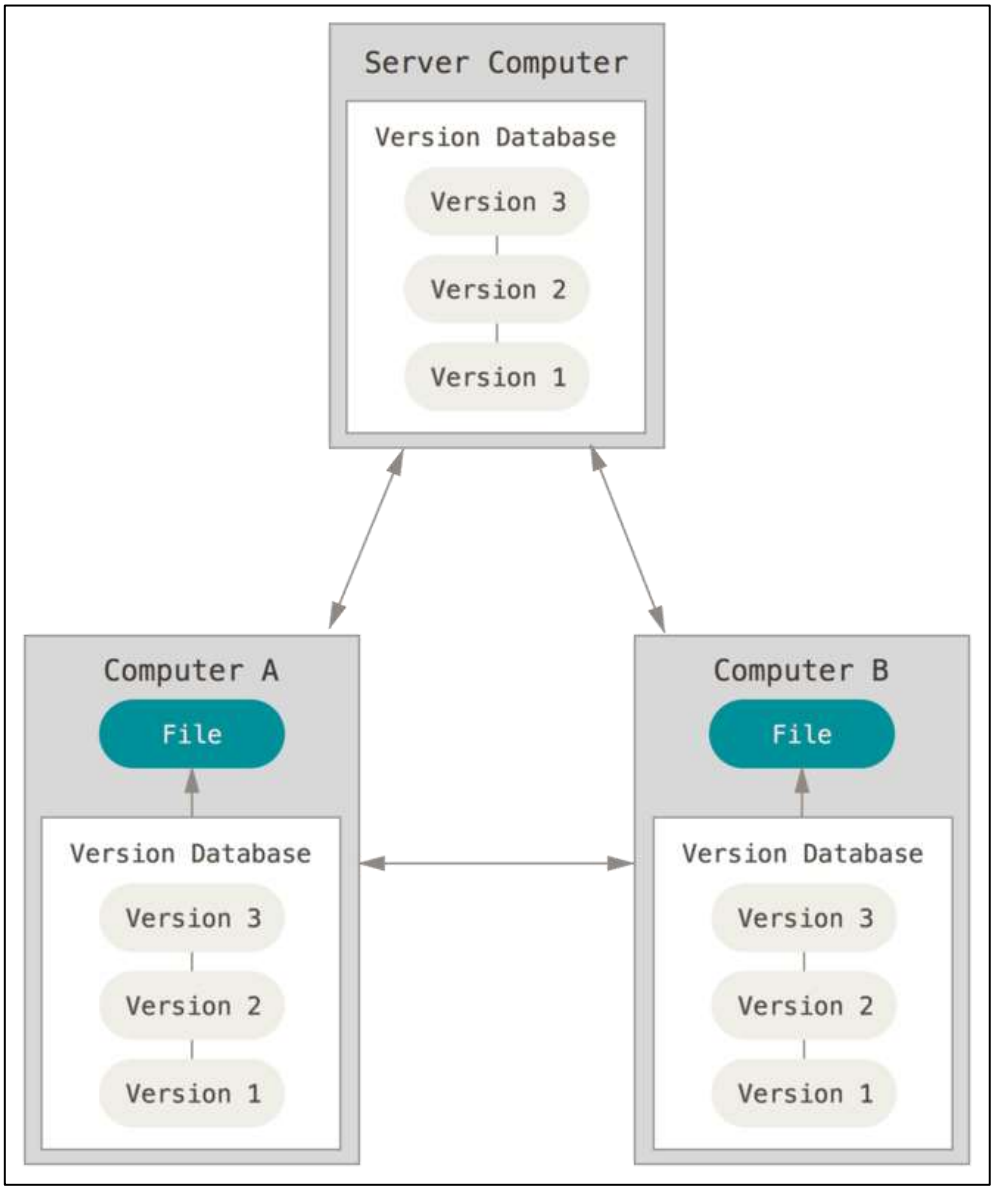


Рис. 1.125. Схема распределённого контроля версий

Кроме того, многие DVCS могут параллельно взаимодействовать с разными удалёнными репозиториями, что позволяет использовать различные подходы в рамках реализации одного проекта. Например, организовать работу с иерархическими моделями, что невозможно достичь в централизованных системах.

Технология Git

Достоинства Git

Определение

Git *– это система управления версиями разрабатываемого ПО, основанная на распределенной архитектуре.*

Git была разработана в 2005 году Линусом Торвальдсом, известного также в качестве создателя ядра операционной системы Linux.

В настоящее время Git – это наиболее популярная технология контроля версий, которая применяется в коммерческих проектах и проектах с открытым исходным кодом. Git работает под управлением различных операционных систем и может быть синхронизирован с известными средами разработок и текстовых редакторов.

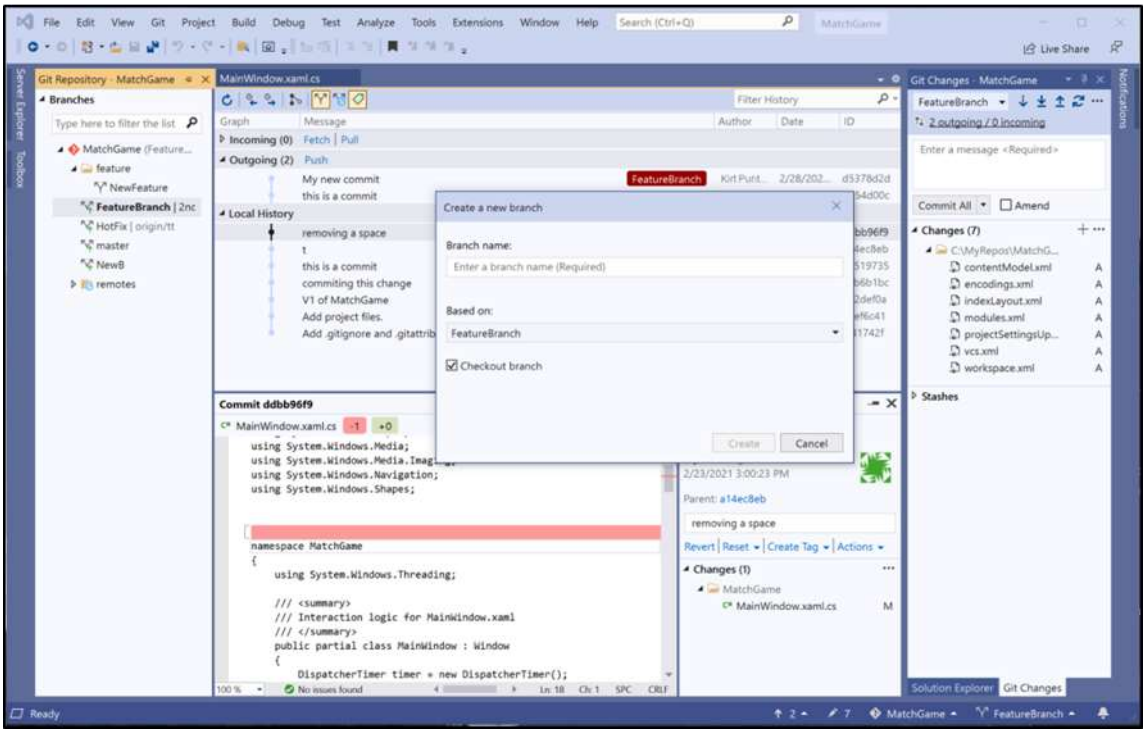


Рис. 1.126. Интеграция Git в Visual Studio IDE

Популярность и высокая распространенность Git связана с рядом причин:

- *Хорошие показатели производительности*. Во многом высокая скорость работы обеспечивается оптимизацией процедуры фиксации *коммитов* (сохраненное изменение), разбиения на ветки, слияние и сравнение предыдущих версий.
- *Высока безопасность*. Git обеспечивает целостность кода проекта. Содержимое файлов, данные об изменениях и операциях с репозиторием защищаются криптографическим алгоритмом хеширования SHA1.
- *Гибкость*. Git позволяет реализовать нелинейные циклы разработки ПО и прекрасно подойдет как для малых, так и для крупных проектов. Далеко не все системы контроля версий позволяют детально отслеживать каждое изменение, как это делает Git.
- *Открытый программный код*. Git бесплатен в использовании и непрерывно поддерживается огромным числом разработчиков. Система обладает подробной документацией.
- *Git как стандарт*. Качественная модель реализации Git и открытый программный код делают технологию образцом в своем классе инструментов контроля версий.

Принципы работы Git

1. *Git работает со «снимками», а не различиями*. Многие VCS хранят информацию в форме списка изменений в файлах.

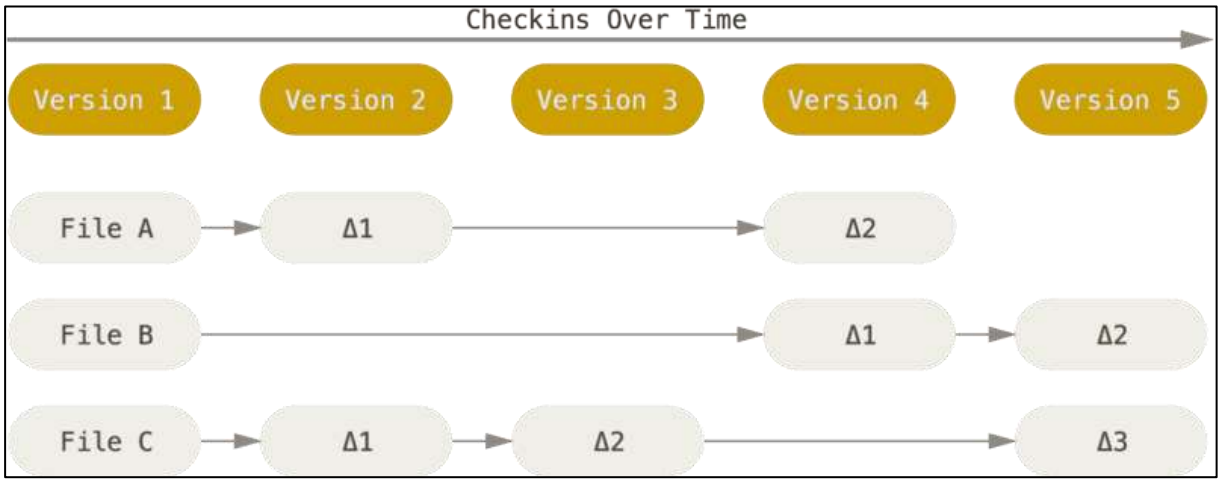


Рис. 1.127. Схема хранения данных в виде набора изменений относительно исходной версии каждого из файлов

Git хранит данные в виде файловой системы. Когда разработчик делает коммит, Git сохраняет состояние файла и ссылку на этот «снимок». Если же изменений не вносилось, то Git копирует ссылку на предыдущую версию файла.

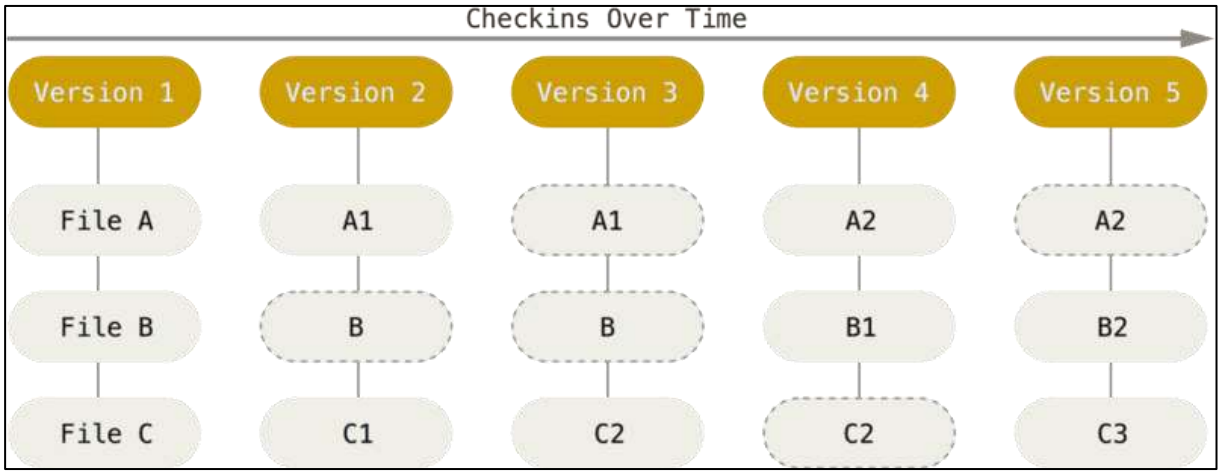


Рис. 1.128. Схема хранения данных об изменениях как «снимков» проекта во времени

2. *Выполнение операций на локальном уровне*. В большинстве случаев операциям Git не требуется информация с других компьютеров сети, а достаточно локальных файлов и ресурсов. История обо всех изменениях хранится на компьютере клиента, поэтому операции с системой достаточно быстрые.

3. *Целостность данных*. Для каждой операции Git вычисляет *хеш-сумма* и только после этого сохраняет информацию о ней. Обращение к сохраненным данным осуществляется по этой хеш-сумме. Таким образом Git фиксирует любые изменения, в частности – нежелательные.

4. *Git добавляет данные*. Любые действия пользователя, требующие изменения состояния версии файла, вносят информацию в базу Git. Поэтому разработчики могут экспериментировать с проектом и не беспокоиться о том, что при необходимости откат к предыдущей версии будет невозможен.

Три основных состояния Git

Git реализует три основных состояния, в которых могут находиться файлы проекта:

1. *Изменён* (*modified*) – файл, который был изменен, но это еще не было зафиксировано.
2. *Индексирован* (*staged*) – изменённый в текущей версии файл, который отмечен для включения в следующий коммит.
3. *Зафиксирован* (*committed*) – файл, который уже сохранён в локальной базе.

Таким образом концепция работы Git состоит в разделении на три секции проекта:

1. рабочая копия (working tree);
2. область индексирования (staging area);
3. каталог Git (Git directory).

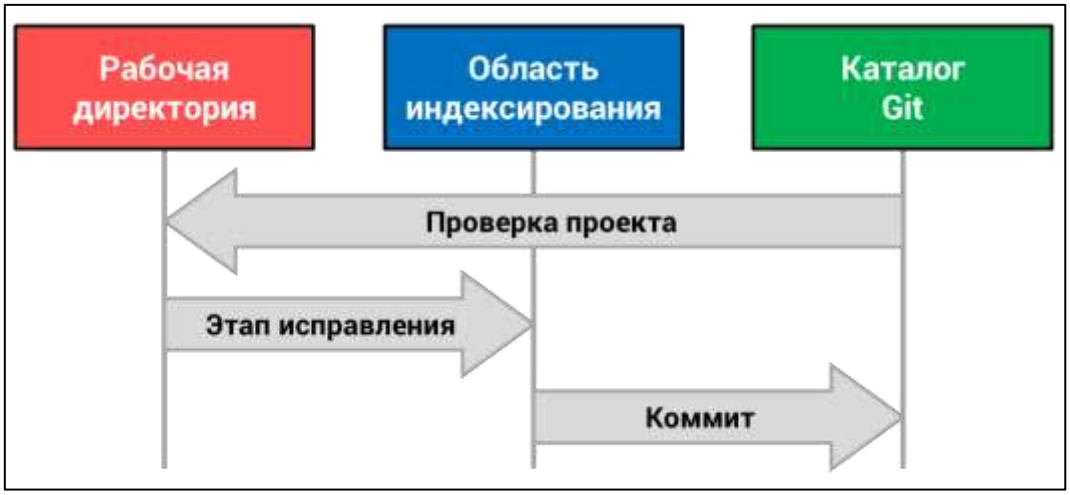


Рис. 1.129. Секции проекта в концепции Git

Рабочая копия – это снимок одной версии проекта. Файлы копии извлекаются из базы данных каталога Git и размещаются на диске для дальнейшего использования или изменения.

Область индексирования является файлом (обычно расположен в каталоге Git), который хранит данные о том, что должно попасть в следующий коммит.

Каталог Git необходим для записи метаданных и базы объектов искомого проекта. Он копируется при каждом клонировании репозитория с другого компьютера.

Упрощенно общий подход в работе с Git можно описать следующим образом:

1. Разработчик вносит изменения в файлы рабочей копии.
2. В индекс выборочно добавляются только те изменения, которые необходимо включить в следующий коммит (т.е. изменения записываются в виде снимка).
3. При создании коммита запрашиваются файлы по соответствующему индексу, новый снимок сохраняется в каталоге Git.

При работе с файлом возможно несколько вариантов:

- если определённая версия файла уже расположена каталоге Git, то она считается зафиксированной;
- если файл был отредактирован и добавлен в индекс, то он считается проиндексированным;
- если с момента последней распаковки в файл были внесены изменения, но он не был добавлен в индекс, то файл числится как модифицированный.

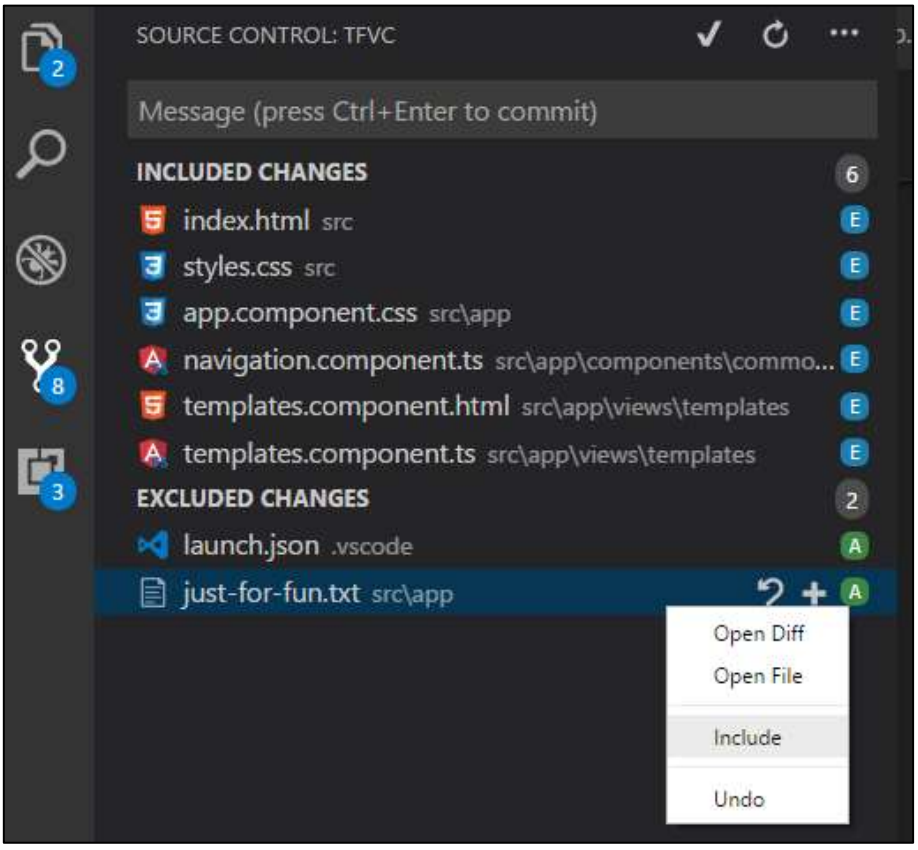


Рис. 1.130. Список сохраненных изменений в проекте (коммиты)

1.5.7. Средства совместной разработки

Роль инструментов совместной разработки ПО

В современных условиях разработка приложений и веб-сервисов зачастую требует командной работы. Парное программирование, в отличие от индивидуальной разработки, позволяет эффективнее отслеживать возникающие проблемы. В частности, один разработчик может писать программный код для реализации задачи, а второй кодер анализировать его, вносить правки, осуществлять рефакторинг и тестирование.

При необходимости взаимодействия нескольких разработчиков важно обеспечить качественную синхронизацию действий. Здесь могут помочь *инструменты совместной разработки*.

Современные средства совместной разработки позволяют решать следующие задачи:

- обеспечивают удаленную работу над проектом;
- предоставляют инструменты для редактирования кода и контроля версий в режиме реального времени;
- способны размещать файлы в открытый доступ.

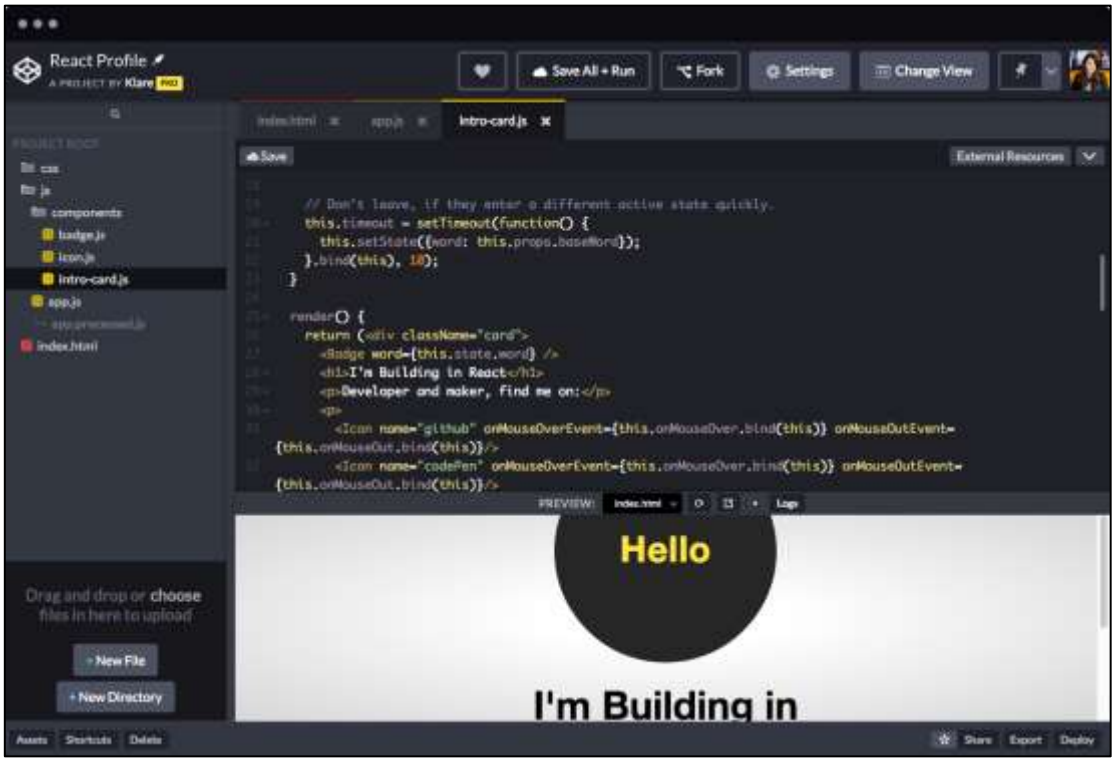


Рис. 1.131. Онлайн «песочница» для веб-разработчиков CodePen

Пример средств совместной разработки ПО

Visual Studio Live

Visual Studio Live – это сравнительно новый проект от Microsoft, призванный обеспечить пользователей Visual Studio и Visual Studio Code излюбленным инструментом совместной разработки.

Для использования службой Live Share необходима учетная запись Microsoft или GitHub. Система связывает соавторов единым сеансом и предоставляет им доступ ко всем файлам проекта. Одновременно может быть обеспечена синхронизация 6 разработчиков. Использование является бесплатным.

Visual Studio Live позволяет:

- отслеживать курсор пользователей в процессе навигации;
- устанавливать точки останова;
- просматривать веб-приложения и базы данных;
- выполнять команды и задачи;
- передавать информацию в текстовом чате, голосовыми сообщениями, видеотрансляцией;
- совместно использовать экран.

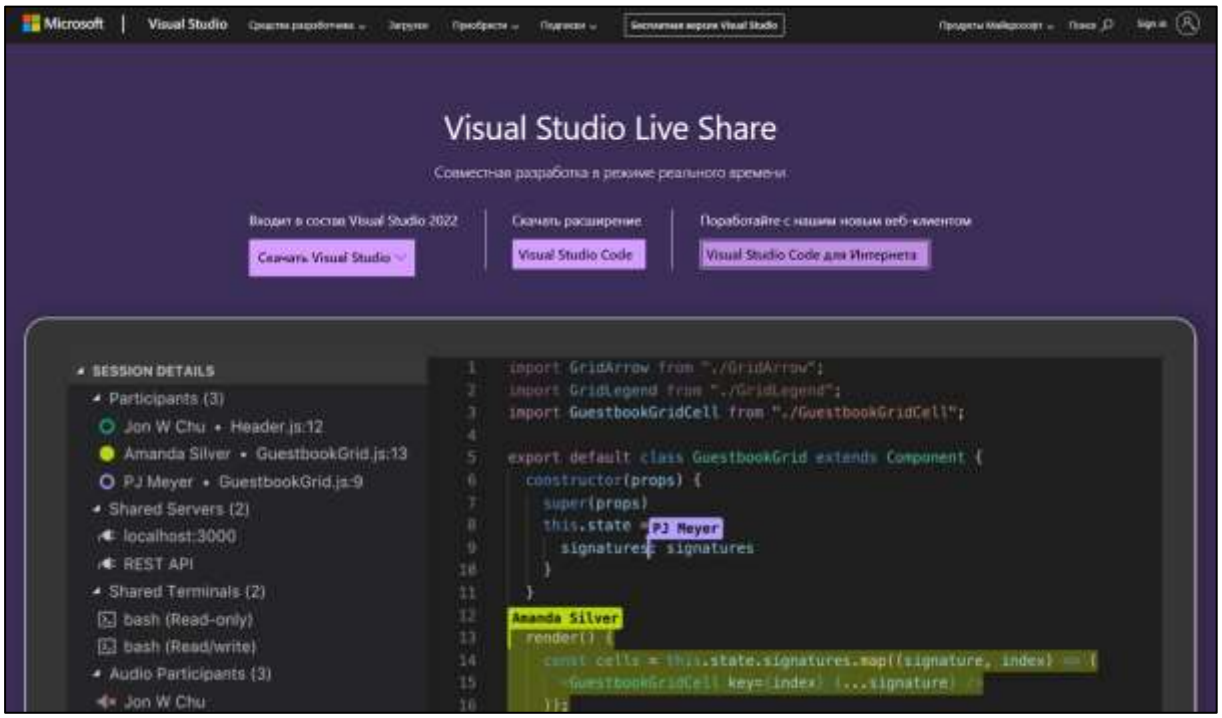


Рис. 1.132. Технология Live Share

Teletype для Atom

Teletype – это бесплатный инструмент от сообщества GitHub разработанный для совместного использования кода в текстовом редакторе Atom.

Teletype работает по следующим принципам:

- создает виртуальное рабочее пространство, похожее по структуре на портал;
- гость получает возможность редактировать код и следить за активным рабочим пространством;
- обмен данными осуществляется по зашифрованному соединению WebRTC;
- серверы не видят файлов и процесса их редактирования, что позволяет сохранить конфиденциальность.

В настоящее время Teletype позволяет передавать только текстовые сообщения, однако в будущем разработчики планируют добавить и средства голосовой связи.

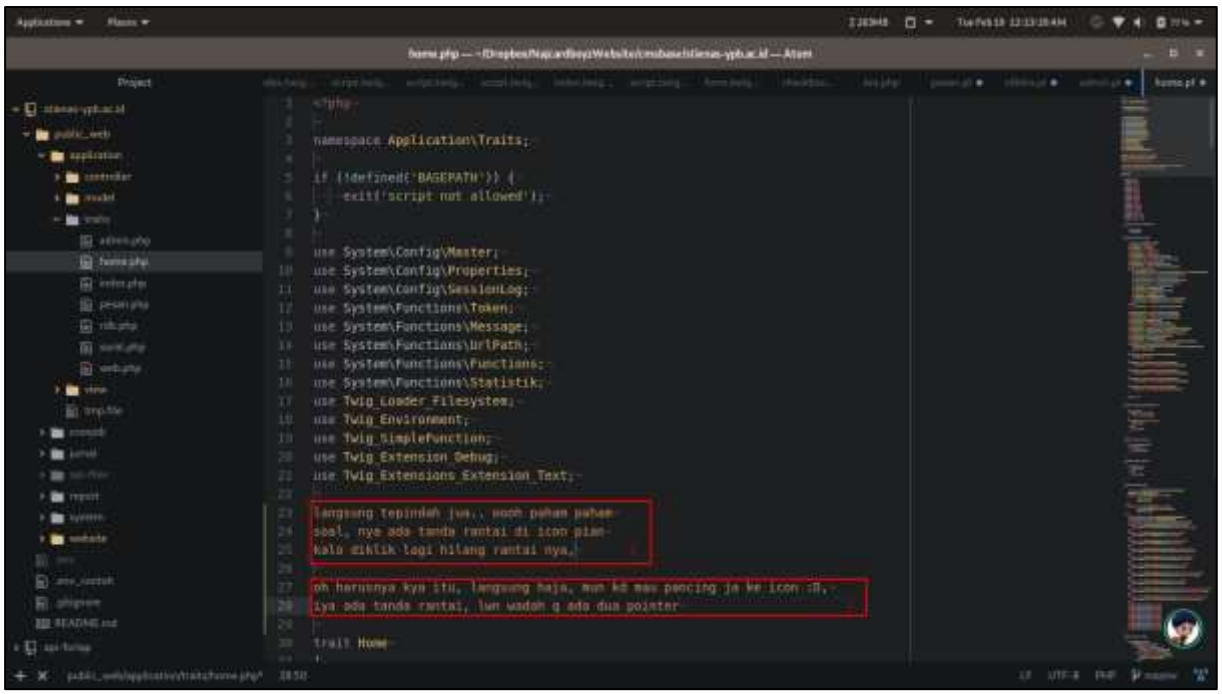


Рис. 1.133. Teletype в редакторе Atom

AWS Cloud9

AWS Cloud9 представляет собой инструмент для совместной работы от Amazon, позволяющий кооперироваться нескольким программистам в единый поток. AWS Cloud9 поддерживает многие популярные языки программирования, например, C#, Go, Python, Java, JavaScript, PHP, Ruby.

Возможности AWS Cloud9 во многом аналогичны современным средам разработки приложений. Пользователи получают удобный инструмент разработки, используя только браузер.

Cloud9 позволяет:

- писать, запускать и вести отладку кода в браузере;
- управлять средой посредством терминала;
- разрабатывать бессерверные приложения;
- взаимодействовать разработчикам в режиме реального времени, проводить сеансы и отслеживать активность;
- работать в групповом чате.

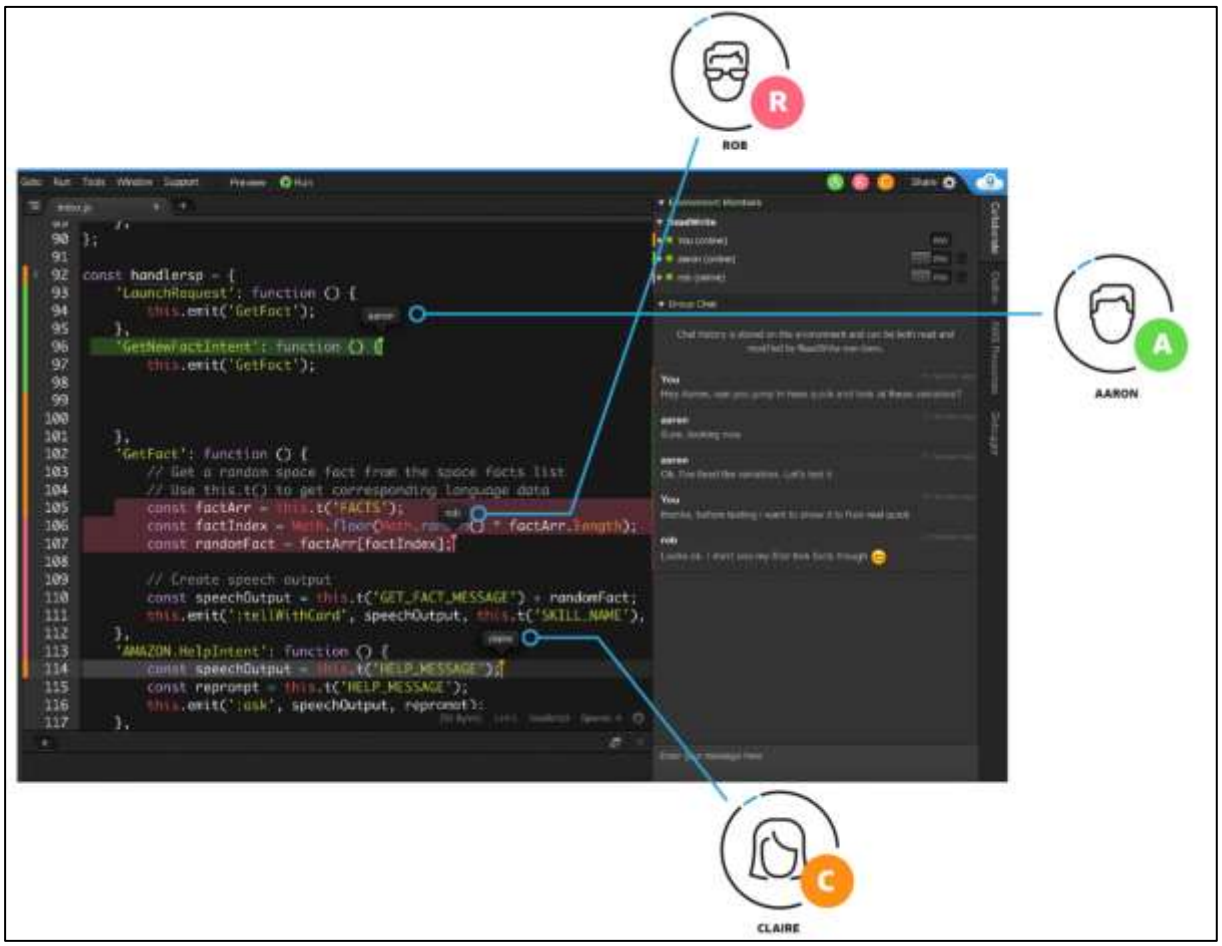


Рис. 1.134. Совместное редактирование в AWS Cloud9

CodePen

CodePen представляет собой веб-редактор для frontend-разработки. Пользователям предоставлена возможность работы с HTML, CSS и JavaScript. CodePen используется в качестве «песочницы», где пользователи могут разместить фрагменты своих проектов или кода, поделиться ссылкой с коллегами, проверить результат верстки веб-страницы и работу скриптов.

Веб-сервис CodePen можно использовать бесплатно. Дополнительно Pro-пользователи получают возможность работы в режиме «Collab Mode», где несколько удаленных разработчиков имеют возможность работать над одним и тем же проектом одновременно и в реальном времени.

Возможности CodePen:

- редактирование HTML, CSS и JavaScript как по отдельности, так и совместно;
- обмен сообщениями в чате;
- публикация кода в открытом доступе;
- гибкая настройка;
- подписка на определенных пользователей;
- режимы работы для студентов и преподавателей.

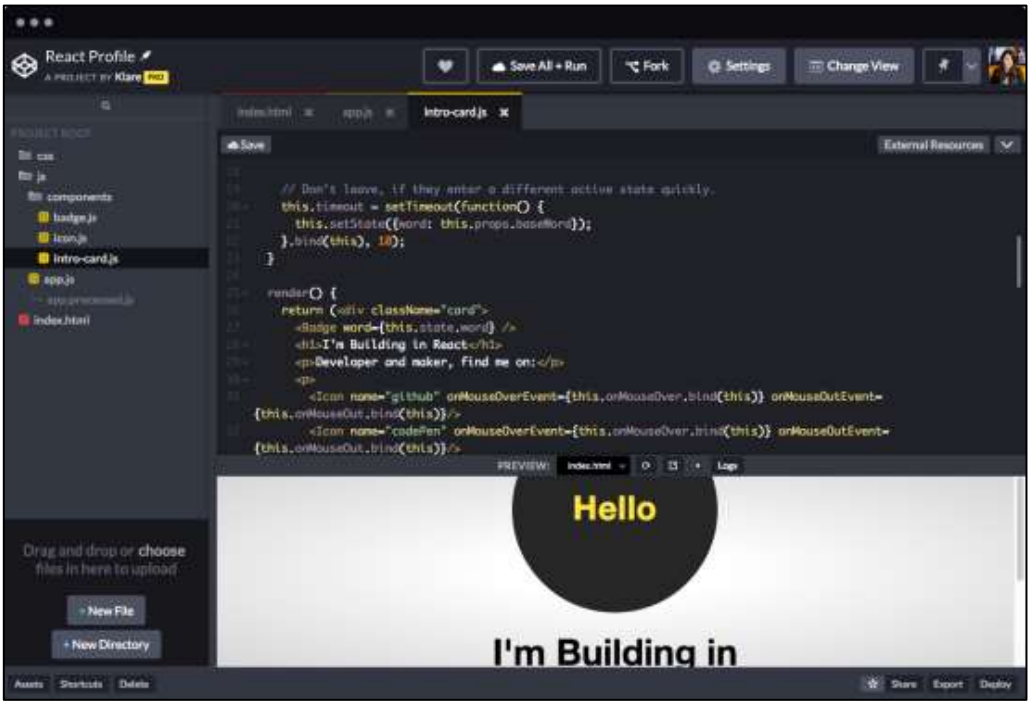


Рис. 1.135. Веб-сервис CodePen

Codeanywhere

Codeanywhere – это редактор кода, позволяющий настраивать среду совместной разработки. Редактор крайне популярен у программистов компаний Adobe, Oracle, Intuit, Cisco и других.

Совместная работа с веб-редактором проста и похожа на отправку ссылки на общий ресурс сотруднику. Другие участники получают возможность предварительного просмотра на сайте Codeanywhere. Можно делиться отдельными файлами или проектами целиком. Интерфейс редактора интуитивно понятен и удобен для пользователя. поделиться целыми проектами, файлами или папками с любым другим разработчиком в любой точке мира.

Codeanywhere обладает следующими возможностями:

- удаленное парное программирование в реальном времени (с двумя и более участниками);
- редактор кода с поддержкой терминала;
- контроль изменений;
- встроенный отладчик;
- поддержка HTML, CSS, JavaScript, Python, Go и т. д.

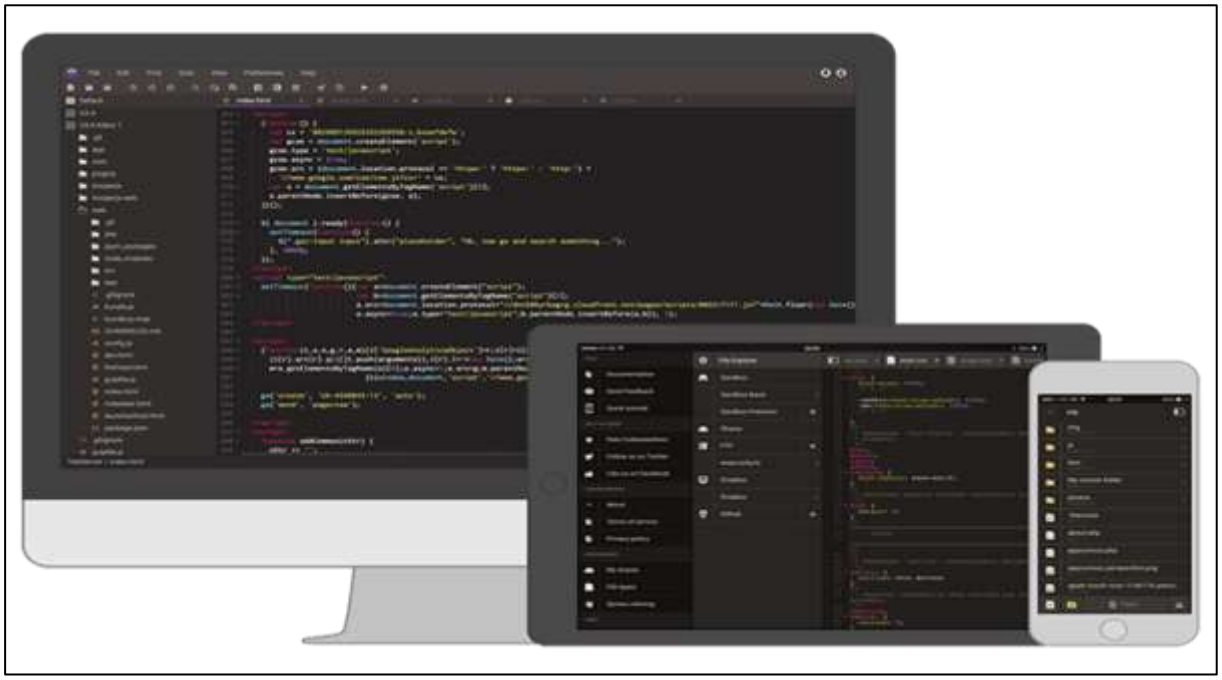


Рис. 1.136. Codeanywhere адаптирован под разные платформы и устройства

Replit

Replit является онлайн-платформой для разработки ПО и хостинга приложений. Платформа позволяет создавать и запускать приложения на таких языках программирования, как Python, Java, Ruby и многие другие.

Replit создавался в качестве облачного инструмента, который позволяет упростить процесс разработки и совместной работы, что снимает с программиста проблемы по установке и настройке рабочего окружения.

Replit предоставляет следующий функционал:

- удобный редактор кода;
- поддержка популярных языков программирования
- командная работа над проектом;
- обмен кодом и комментариями;
- хостинг приложений;
- интеграция с GitHub;
- возможность работы с нейросетями;
- работа с терминалом и утилитам для управления сервером.

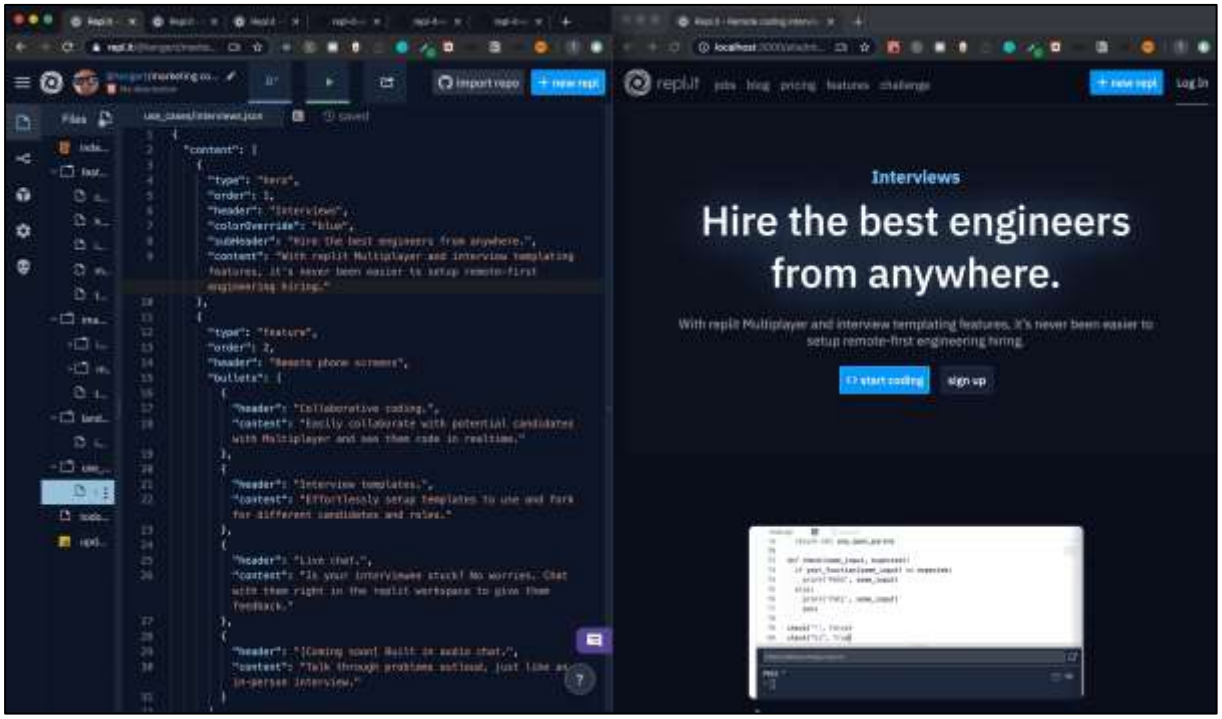


Рис. 1.137. Replitd в качестве онлайн редактора кода

1.5.8. CMS

Использование и возможности CMS

Структура CMS

Определение

Система управления содержимым *(**C**ontent **M**anagement **S**ystem, **CMS**) – это программное обеспечение, позволяющее создавать, редактировать и управлять содержимым веб-сайта.*

CMS используется как инструмент, который упрощает наполнение веб-страниц информацией. Это избавляет программистов от большого объема рутинной работы по изменению кода разметки и скриптов. Для удобства CMS реализует визуальный интерфейс, в котором происходит управление содержимым страниц сайта.

CMS-движок позволяет конструировать сайт на основе типовых функциональных блоков. Для простых веб-сайтов CMS даже не требуют от пользователя специальных знаний в области веб-программирования. Поэтому системы управления содержимым зачастую также являются тем инструментом, который передается заказчику для дальнейшего самостоятельного администрирования сайта.

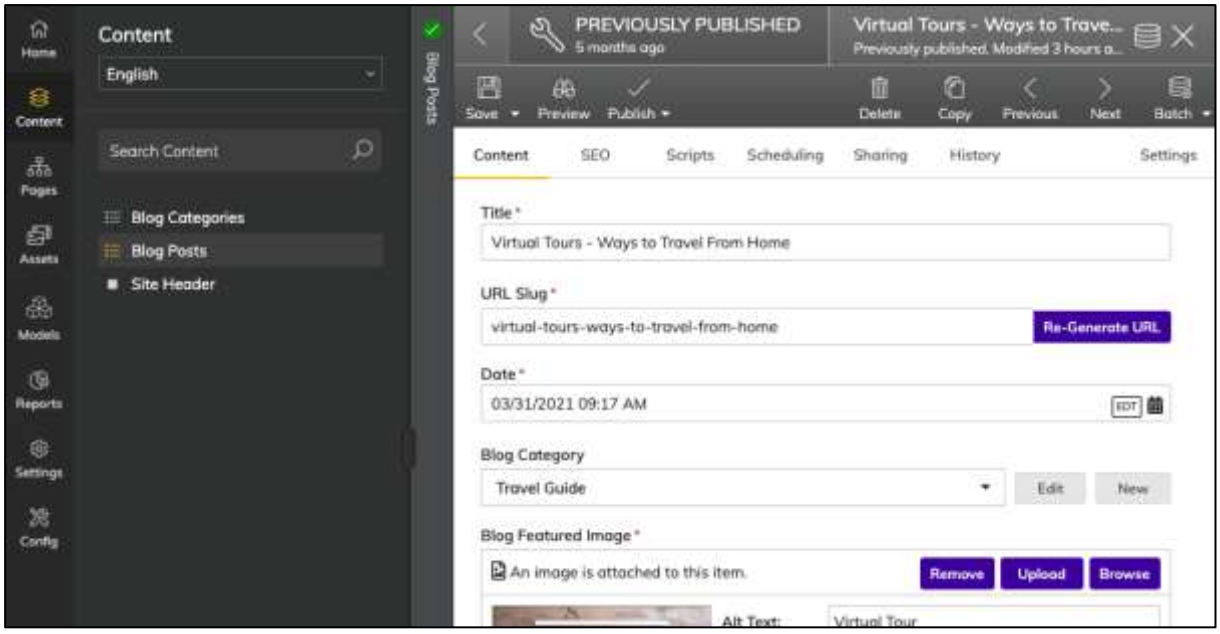


Рис. 1.138. CMS предоставляет конструктор для заполнения блоков сайта

CMS состоит из следующих основных компонентов:

- *Подсистема для хранения баз данных*. В нее включаются данные о пользователях веб-сайта, его содержимом и других параметров.
- *Подсистема для хранения объектов интерфейса*. С этими элементами взаимодействуют пользователи.
- *Подсистема конструктора*. Редактор с визуальным интерфейсом, в котором осуществляется заполнения блоков веб-сайта текстовым и мультимедийным контентом.
- *Другие модули*. Могут быть настроены по желанию пользователя, в том числе собственная разметка и скрипты.

Использование CMS

CMS-системы обычно являются оптимальным вариантом для организаций, веб-сайты которых имеют типовой функционал (онлайн-визитки, веб-магазины, порталы, блоги и прочее).

Движок CMS используется для следующих задач:

- наполнение веб-сайта контентом;
- модификация структуры блоков, содержимого и администрирование веб-ресурса;
- создание новых сайтов по типовым макетам;
- расширение функционала веб-страниц;
- адаптация веб-интерфейса к новым требованиям.

Достоинства и недостатки CMS

Системы управления содержимым сайта обладают целым рядом преимуществ (по сравнению с ручной версткой):

- наполнение страниц содержимым не требует специальных знаний в области веб-разработки;
- быстрая разработка и наполнение веб-ресурса;
- дизайн выстраивается по шаблону;
- удобство администрирования.

Однако CMS-инструменты обладают важным недостатком: они не позволяют разрабатывать сайты с уникальным и нестандартным дизайном и функциональностью.

Примеры CMS

Виды CMS

При выборе CMS необходимо определить возможности, ограничения и условия использования их движка. Среди CMS условно выделяют:

- открытые и бесплатно распространяемые системы;
- закрытые и платные системы (искомый код не допускает модификации).

Кроме тог, CMS по типу формируемых веб-страниц делятся на:

- статичные (веб-страницы не требуют изменения);
- динамические (для создания интерактивных вебсайтов).

Примеры

WordPress – это один из лидеров в категории CMS. Изначально использовался для создания интернет-блогов. Со временем возможности WordPress были существенно расширены.

В настоящее время система позволяет решать обширный спектр задач, требующих веб-программирования.

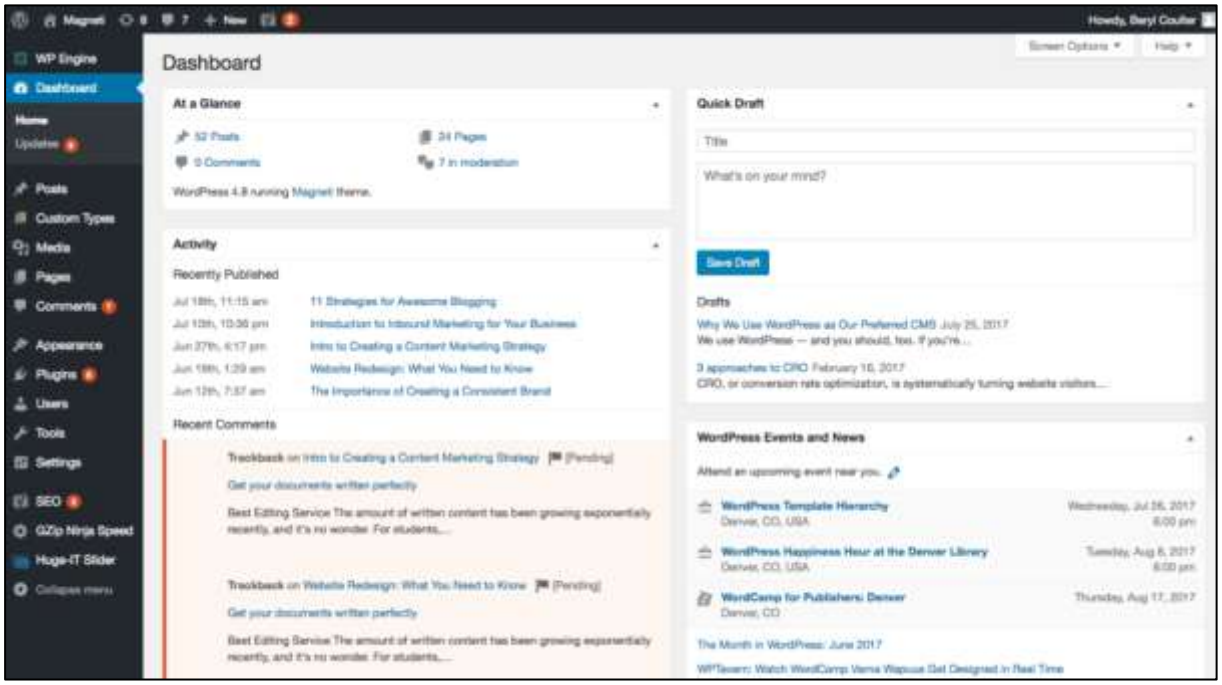


Рис. 1.139. Интерфейс CMS WordPress

1С Битрикс – коммерческая система с богатым функционалом. Обычно ее выбирают для реализации больших веб-систем.

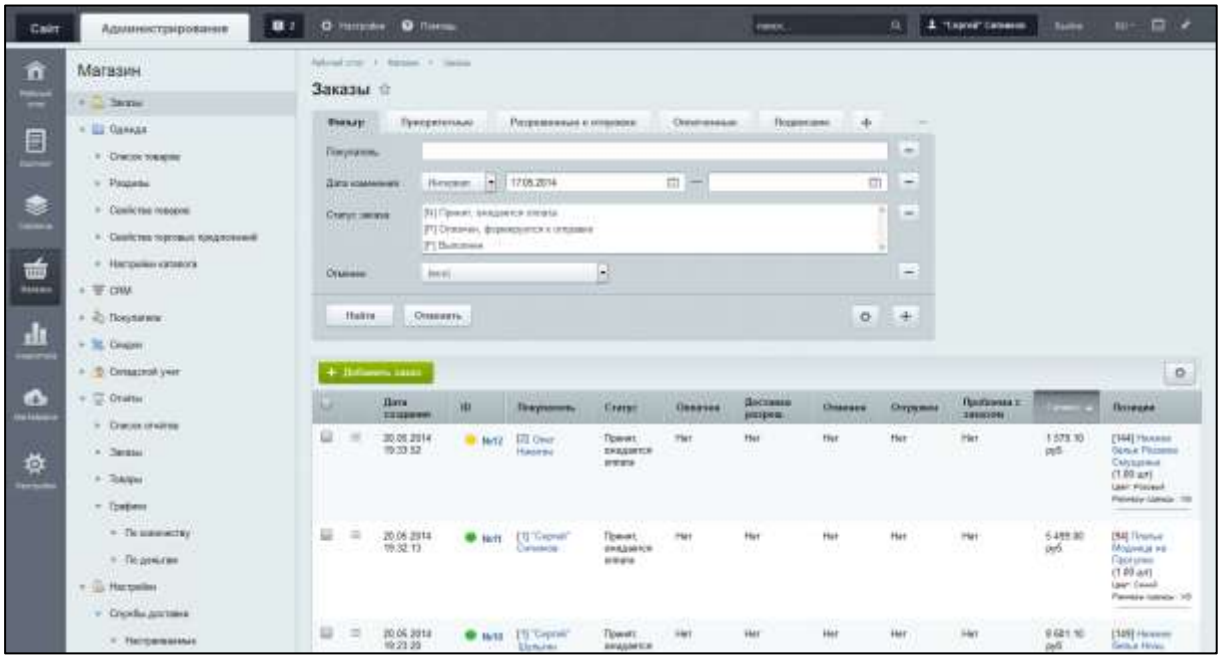


Рис. 1.140. Интерфейс CMS 1С Битрикс

Joomla –это бесплатная CMS для разработки разнообразных веб-сайтов. Обладает большим набором шаблонов, хорошо оптимизирована и может быть расширена дополнительными модулями.

Среди недостатков отмечается отсутствие официальной технической поддержки.

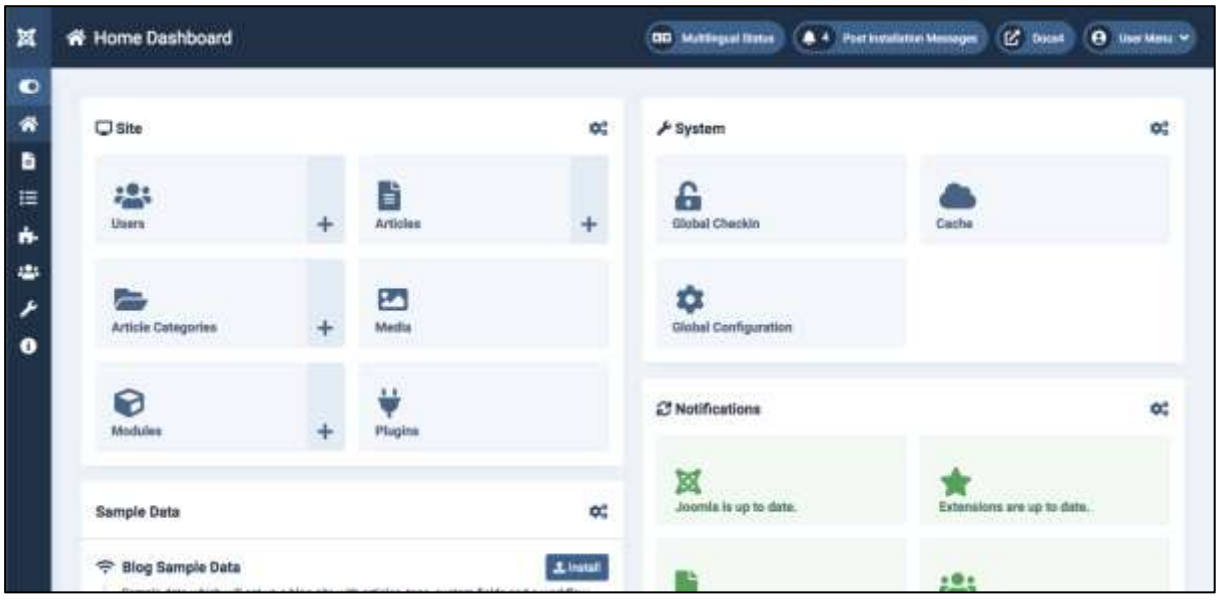


Рис. 1.141. Интерфейс CMS WordPress

Drupal – это гибких и многофункциональный фреймворк, который будет хорошим решением при создании блога, форума, корпоративного сайта, интернет-магазина, сайта услуг или социальной сети. Drupal сложна для новичков и требует компетенций в области веб-программирования.

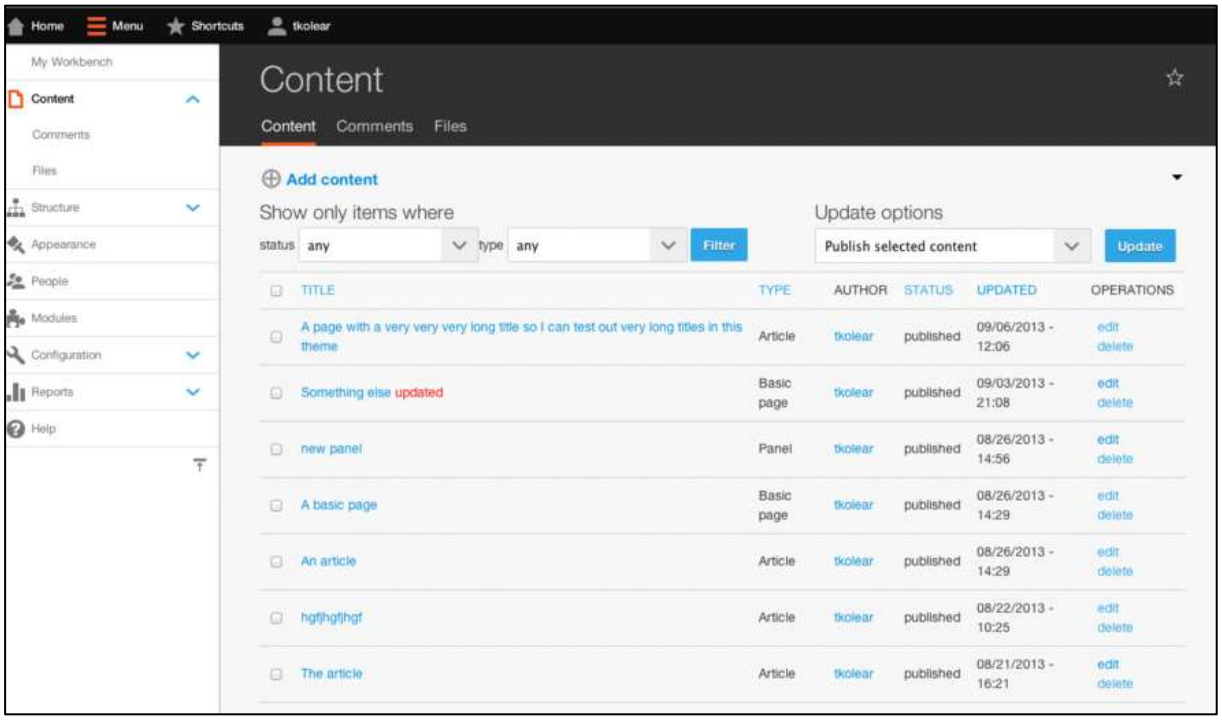


Рис. 1.142. Интерфейс CMS Drupal

CMS.S3 подходит для реализации сайтов-визиток или интернет-магазинов. Однако доработка сайтов под требования поисковых систем будет затруднительной, поскольку техподдержка не всегда работает оперативно.

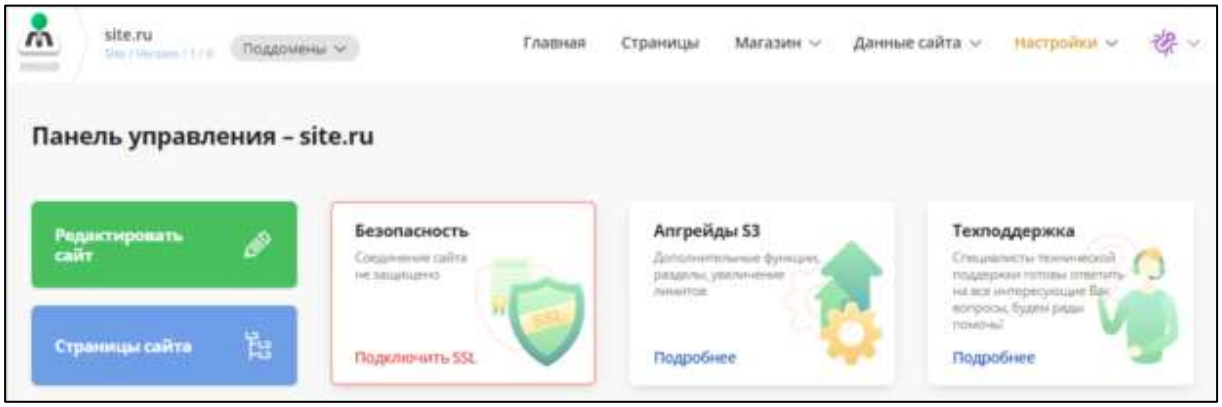


Рис. 1.143. Интерфейс CMS CMS.S3

Modx – это бесплатная CMS и фреймворк для разработки веб-приложений. Modx практически не ограничивает пользователей в создании в веб-сайтов любого типа. Возможности фреймворка позволяют модифицировать панель администрирования. Однако система не отличается большим числом предлагаемых шаблонов и в целом потребует комплексной настройки.

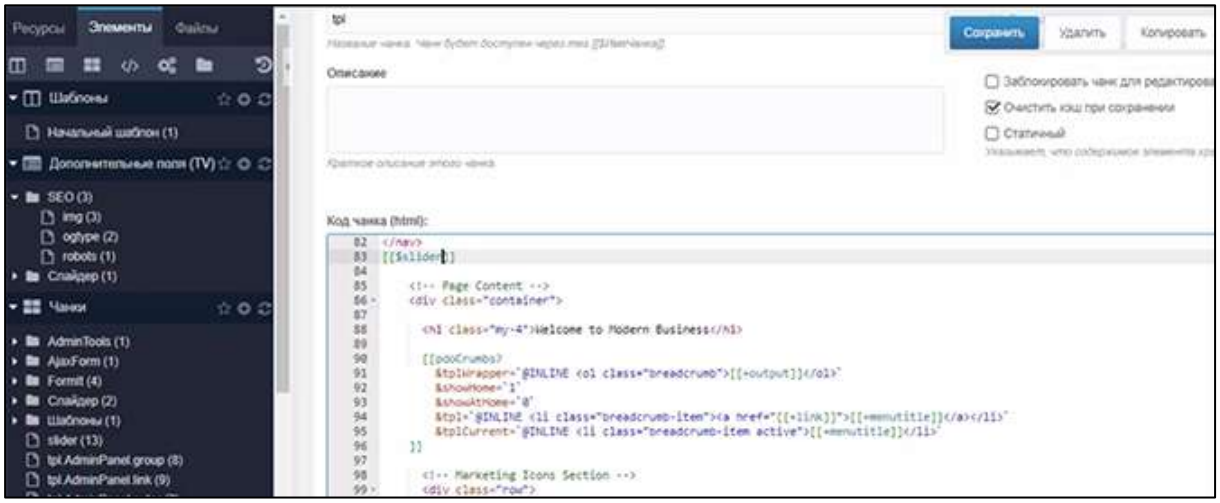


Рис. 1.144. Интерфейс CMS Modx

OpenCart является бесплатной CMS с открытым исходным кодом. Это отличный выбор для создания онлайн-магазинов, где требуется обеспечить безопасный обмен данными в сети. Среди недостатков отмечаются проблемы при установке сторонних модулей.

Рис. 1.145. Интерфейс CMS OpenCart

Вопросы для самопроверки

1. Для чего необходимы API и какие возможности они предоставляют разработчику?
2. Почему в концепции API важное значение имеет понятие интерфейса?
3. Что представляют собой фреймворки и в чем преимущество их использования при разработке ПО?
4. Опишите основные компоненты модели архитектуры MVC.
5. В чем принципиальная разница между фреймворком и API? Существует ли связь между ними?
6. Перечислите возможности, которые обычно реализованы в современных средах разработки ПО.
7. Что представляет собой технология IntelliSense и как она упрощает работу с кодом?
8. Какую цель и задачи преследует рефакторинг кода?
9. Опишите важнейшие особенности методологии RAD. Чем могут помочь современные IDE при ее использовании?
10. Перечислите некоторые популярные текстовые редакторы для веб-разработчиков.
11. Может ли текстовый редактор полностью заменить среду разработки?
12. Какие элементы веб-сайта могут быть созданы или обработаны в графических редакторах?
13. Для чего создаются прототипы веб-сайта? Приведите примеры программ и веб-сервисов для прототипирования.
14. Опишите причины, по которым работа с системами контроля версий ПО более безопасна.
15. Перечислите основные виды систем контроля версий и особенности их работы.
16. В чем преимущество технологии Git в сравнении с другими системами контроля версий?
17. Какие задачи стоят перед CMS-системами?
18. Почему создание сайтов с уникальным интерфейсом и функционалом не может быть реализовано средствами CMS?

Практикум

Задание 1

1. Проверьте, поддерживается ли работа приложений Java на вашем компьютере. Для этого перейдите в *Панель управления / Программы и компоненты* (рис. 1.146).
2. Осуществите запуск любого Java-приложения. Если запуск не удался, обновите пакет Java SE.
3. В чем отличие Java JRE от Java JDK? Можно ли разрабатывать приложения для Java с помощью JRE?

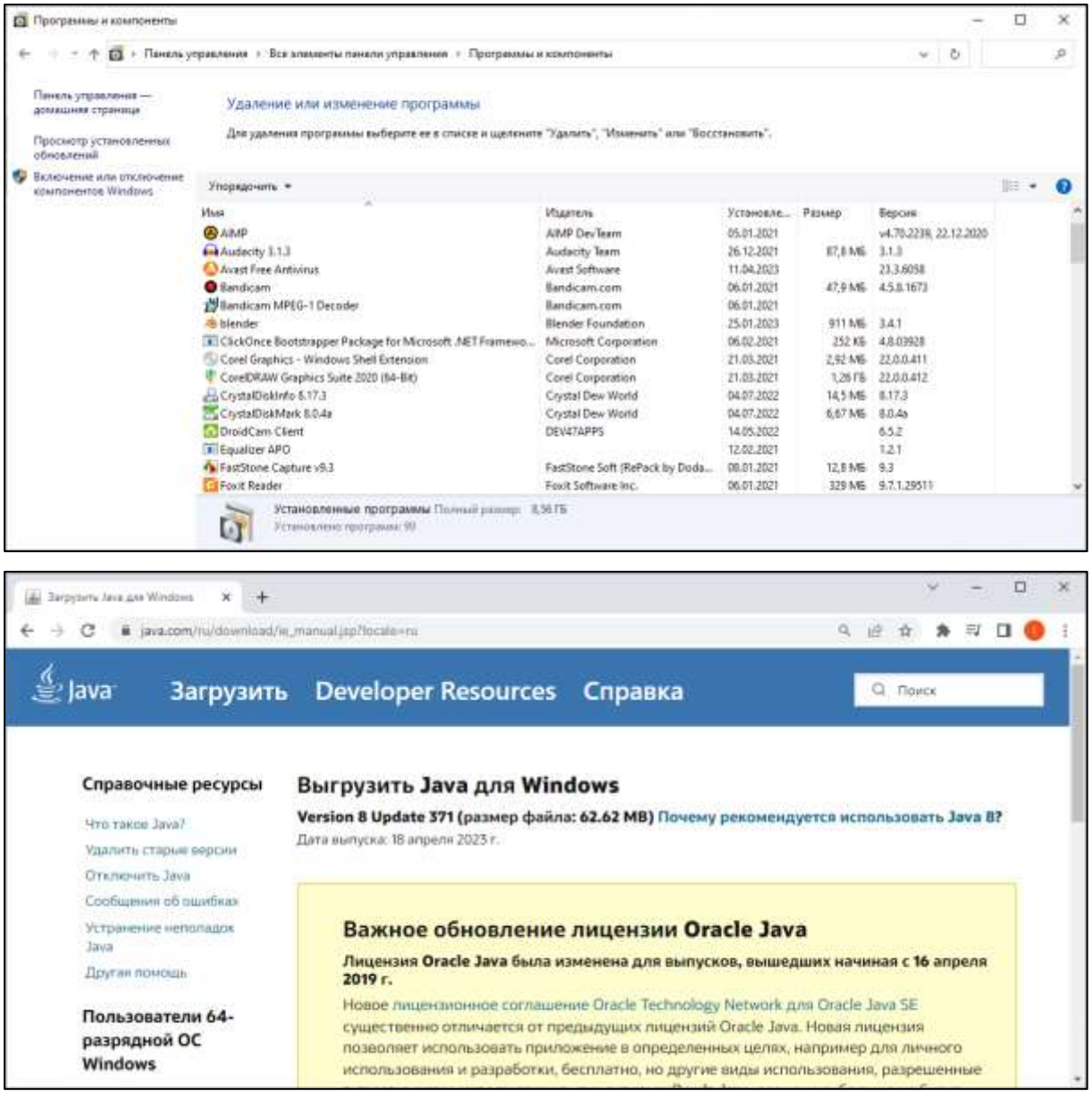


Рис. 1.146. Установленные приложения и загрузка среды выполнения Java

Задание 2

1. Что представляет собой платформа .NET / .NET Framework?
2. Проверьте поддержку выполнения приложений на базе платформы .NET на своем ПК (*Панель управления*).
3. Установите последнюю актуальную версию пакета разработчика .NET SDK и язык программирования C#. Воспользуйтесь официальным сайтом Microsoft: https://dotnet.microsoft.com/en-us/download.
4. По завершению процесса установки проверьте наличие пакета в списке установленных программ (рис. 1.147).
5. Запустите командную строку и с помощью команды

   ```
   dotnet --version
   ```

   узнайте версию .NET (рис. 1.148).

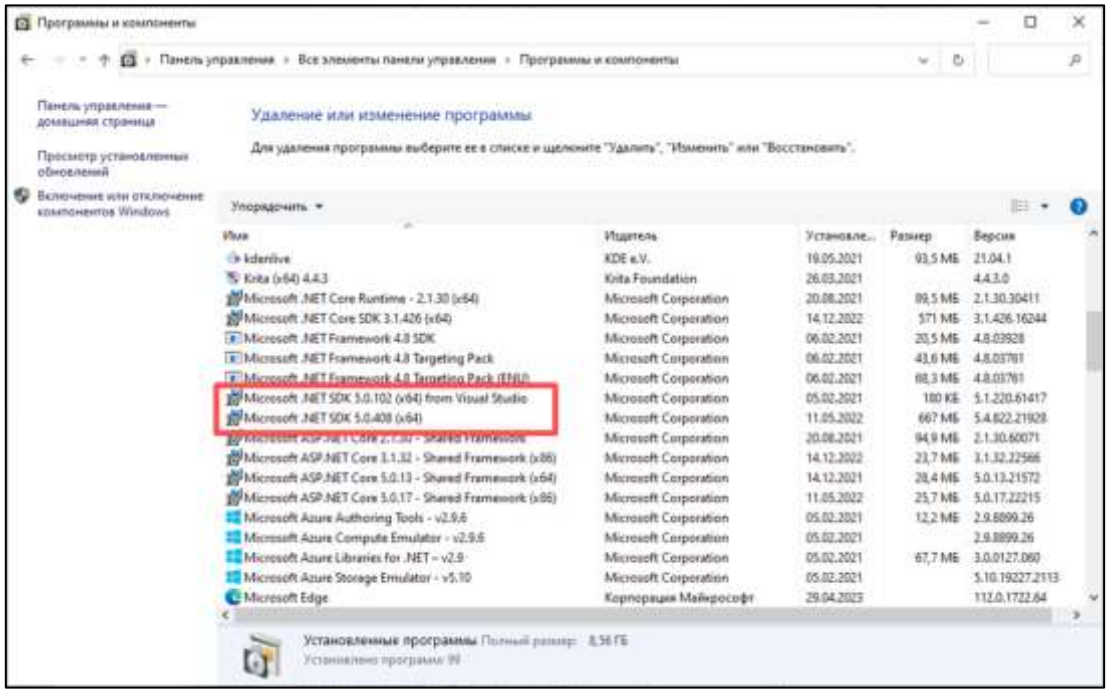


Рис. 1.147. Установка пакета .NET SDK

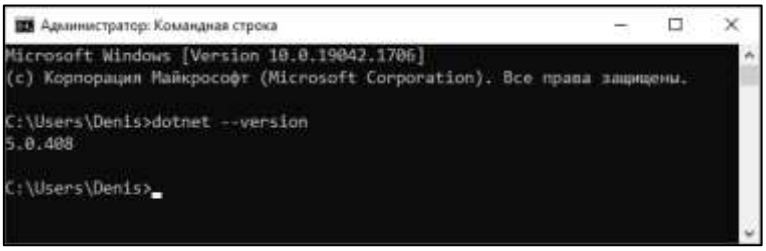


Рис. 1.148. Проверка версии .NET

Задание 3

1. Какие среды разработки вы используете (или установлены на вашем ПК)?
2. Является ли учебная среда Pascal ABC средой разработки?
3. Загрузите Pascal ABC и на примере некоторого программного кода протестируйте основные возможности среды. Определите, поддерживает ли она:
   a. подсветку синтаксиса;
   b. автоматическое форматирование кода;
   c. режим IntelliSense;
   d. анализ данных в процессе отладки программы;
   e. точки останова и инструменты отладки кода;
   f. конструктор приложений с визуальным интерфейсом;
   g. другие инструменты.
4. Можно ли использовать среду Pascal ABC для программирования на других языках?

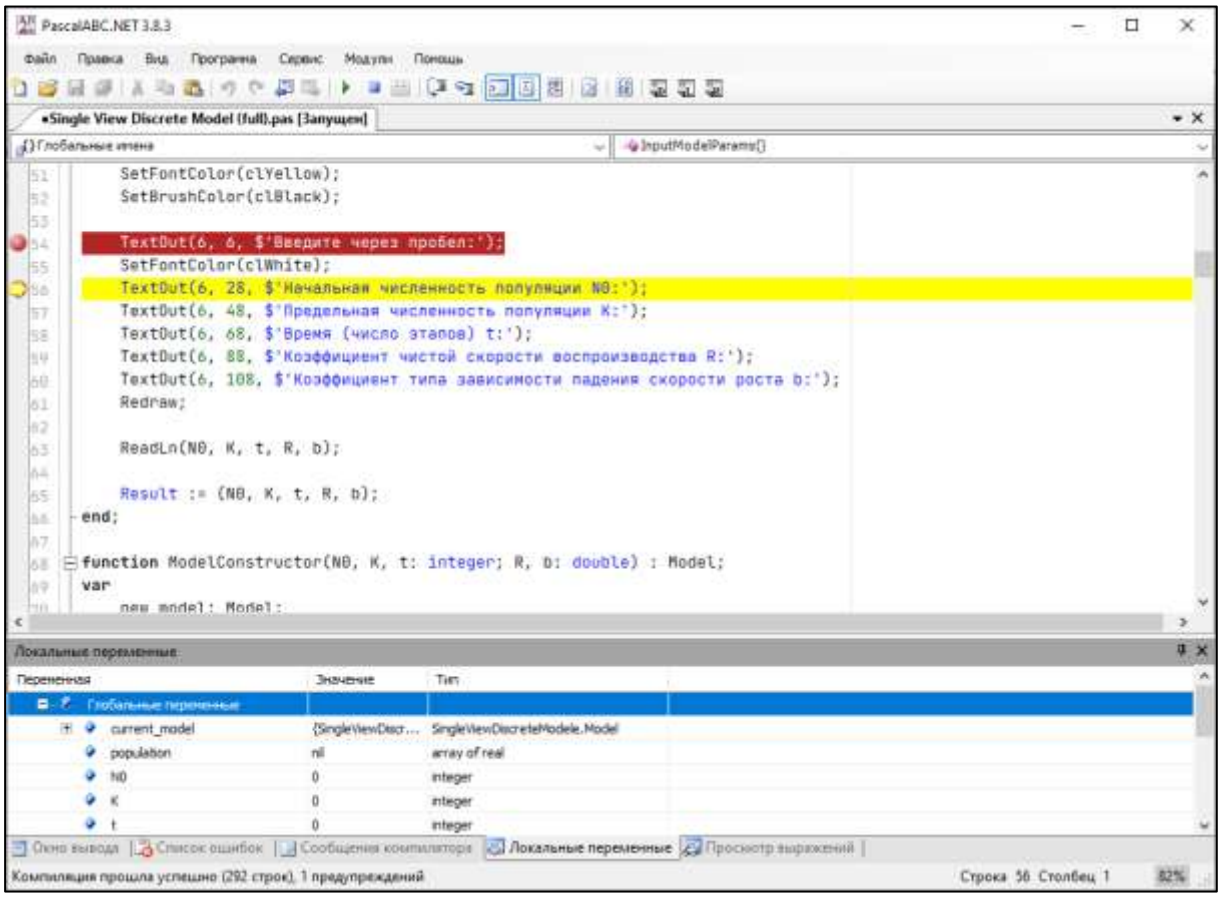


Рис. 1.149. Среда разработки Pascal ABC.NET

Задание 4

1. Перейдите на официальный сайт Microsoft и скачайте редактор Visual Studio Code: https://code.visualstudio.com (рис. 1.150).
2. Далее принимаем условия соглашения, указываем при необходимости каталог для установки редактора.
3. В появившемся окне *Выберете дополнительные задачи…* следует поставить опции для второго и третьего пунктов, чтобы редактор был доступен в контекстном меню *ПКМ*. Осталось дождаться установки и запуска редактора (рис. 1.151).
4. Нажмите на кнопку *Расширения*. В строку поиска введите *Russian Language Pack* – это плагин, которые отвечает за русификацию интерфейса редактора (рис. 1.152). После установки обязательно перезагрузите редактор, чтобы изменения вступили в силу.
5. Далее установите расширения для работы с языком программирования C#: *C# for VSC*. Задача этого расширения – связать .NET с редактором и режимом IntelliSense. Расширения скачиваются из сети Интернет автоматически (рис. 1.152).
6. Для создания нового проекта необходимо в любом месте вашего диска создать новый каталог и перетащить его в область *Проводника* редактора (Рис. 1.153).
7. Visual Studio Code поддерживают работу с командной строкой и многие операции по сборке проекта осуществляются обычно через нее.
8. В панели меню нажмите на *Вид / Терминал*. В нижней части редактора появится окошко терминала, ожидающее инструкции пользователя (рис. 1.154).
9. Введите следующую команду:

   ```
   dotnet new console
   ```

   Эта инструкция обращается к компилятору .NET и требует создать консольное приложение (рис. 1.155). Через несколько секунд в окне терминала появится информация сборке проекта (рис. 1.156), а в обозреватель добавятся файлы

проекта, в том числе стартовый шаблон программы (рис. 1.157).

10. Если нажать на файл с кодом программы, то на вкладке *Выходные данные* можно увидеть, как загружаются некоторые дополнительные данные для работы с проектом в редакторе (рис. 1.158). Необходимо дождаться, чтобы загрузился сервер OmniSharp, отвечающий за корректную работу плагина (рис. 1.159-рис. 1.160).
11. Кроме того, через время появится уведомление, предлагающее добавить недостающие ресурсы для сборки и отладки приложения, нажимаем *Да* (если оно пропало, то нажмите на колокольчик в нижнем правом углу). Можно приступать к редактированию кода (рис. 1.160).
12. Внесите некоторые изменения в код, например, добавьте работу с переменными. Сохраните файл (рис. 1.161).
13. Необходимо скомпилировать проект и запустить исполняемый файл. Для этого в терминал вводим команду

    ```
    dotnet run
    ```

14. Инструкция компилирует проект и запускает приложение, если не возникло ошибок компиляции (рис. 1.162).
15. После выполнения программы в окне терминала отобразится результат ее работы. Далее можно вносить изменения в код программы, сохранять код в файле и повторно вызывать команду

    ```
    dotnet run
    ```

    для перекомпиляции и запуска проекта.
16. В случае успешной компиляции проекта в каталоге *bin* генерируется исполняемый файл приложения. Его можно запустить как автономное приложение (рис. 1.163).
17. Когда работа с проектом завершена, нажимаем *Файл / Закрыть папку* (рис. 1.164). По аналогии создаются другие проекты.

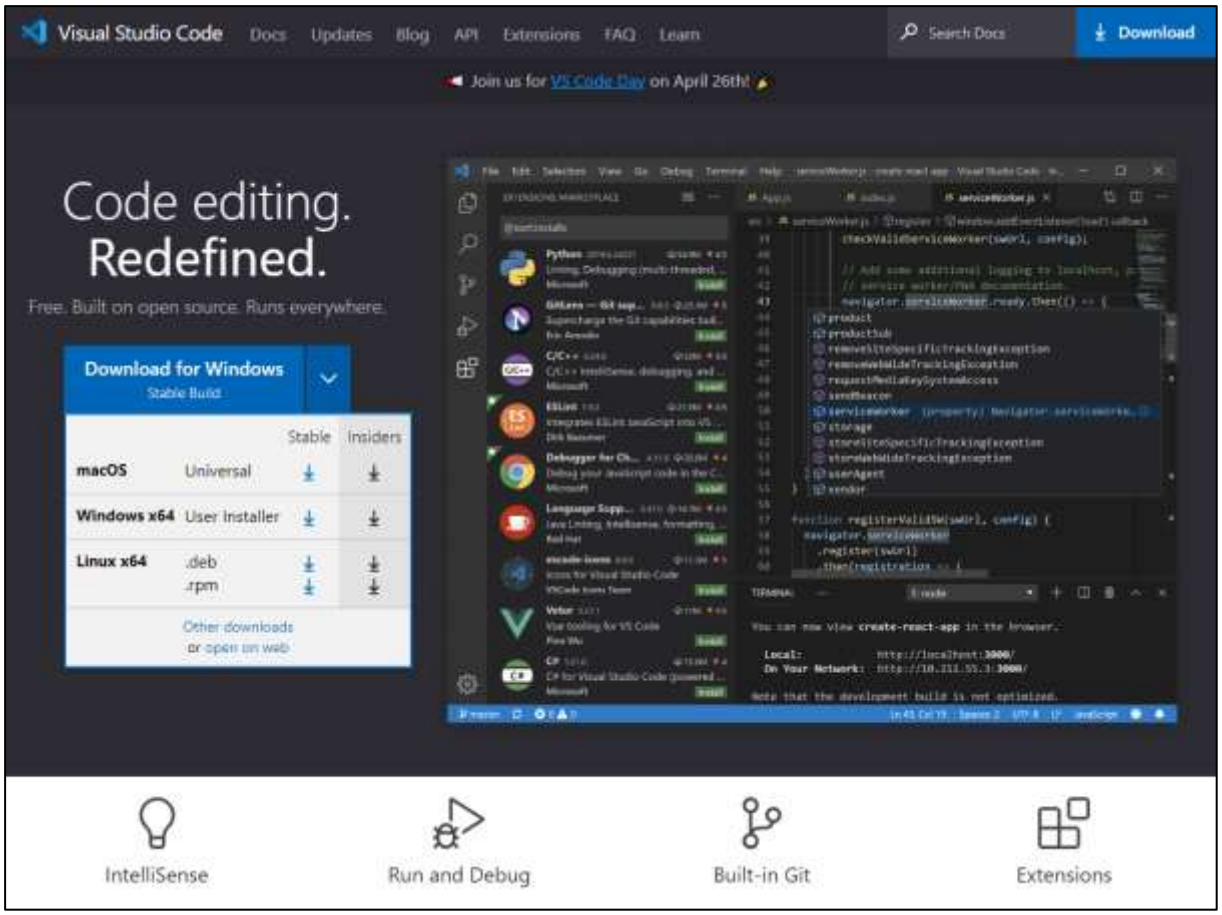


Рис. 1.150. Скачивание файла установки редактора Visual Studio Code с официального сайта

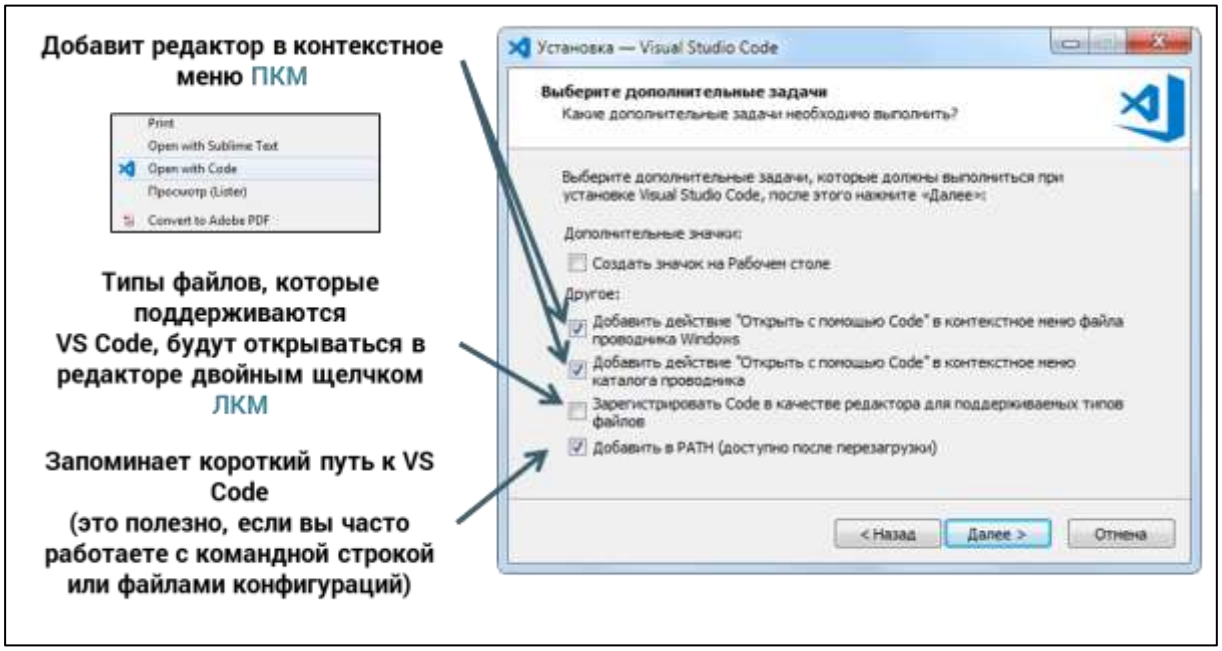


Рис. 1.151. Активация дополнительных параметров редактора при установке

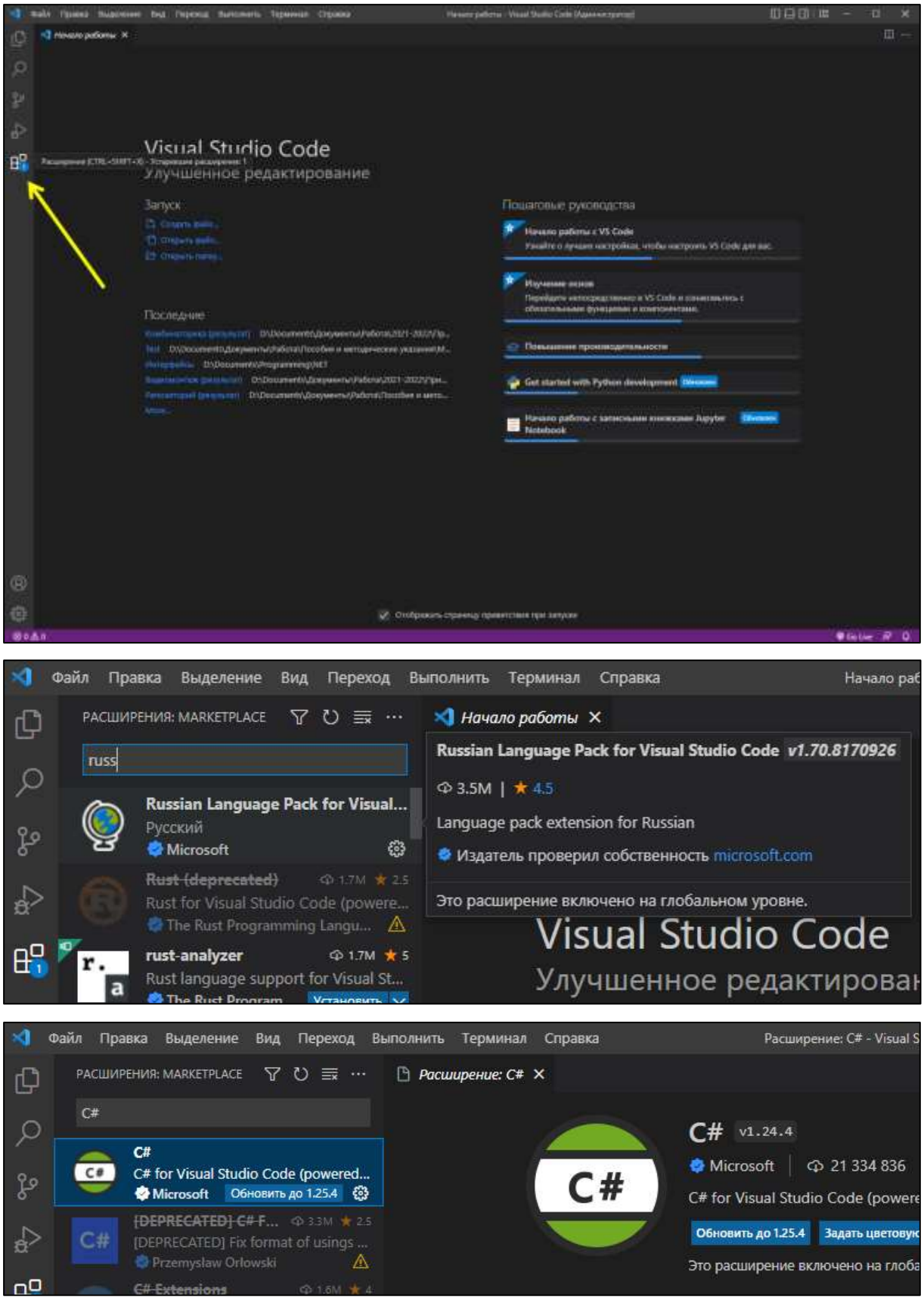


Рис. 1.152. Установка дополнительных расширений: русификация интерфейса редактора и плагин для работы с .NET и языком C#

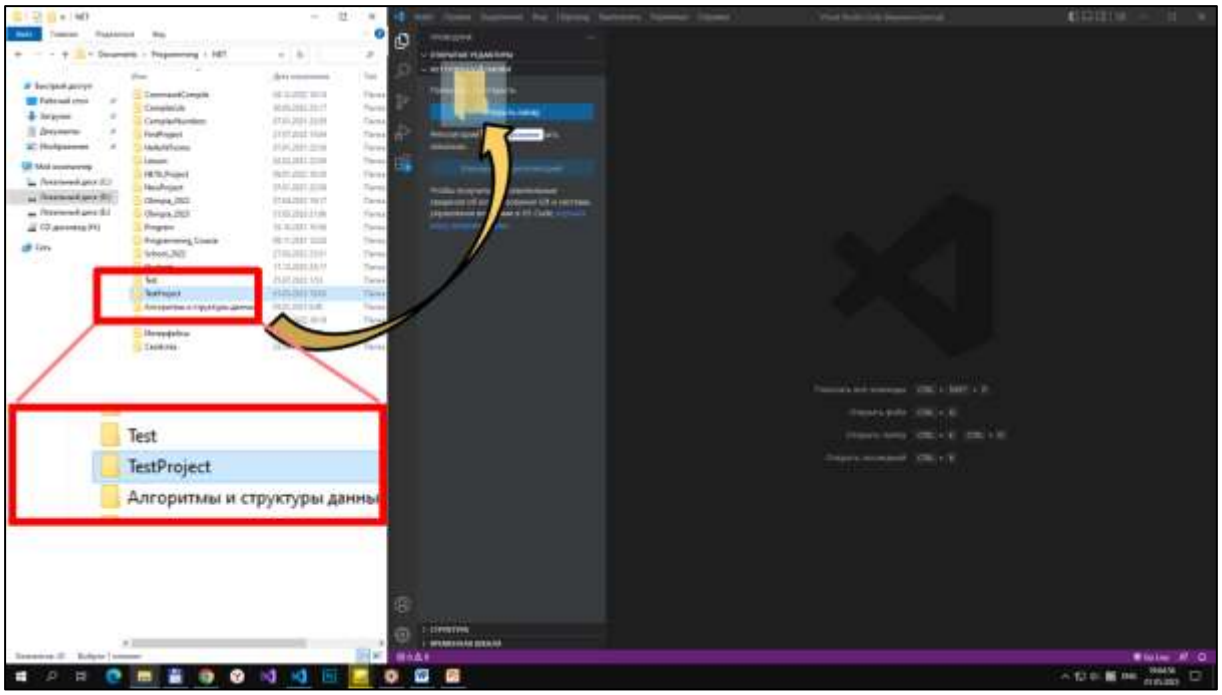


Рис. 1.153. Загрузка коренного каталога проекта

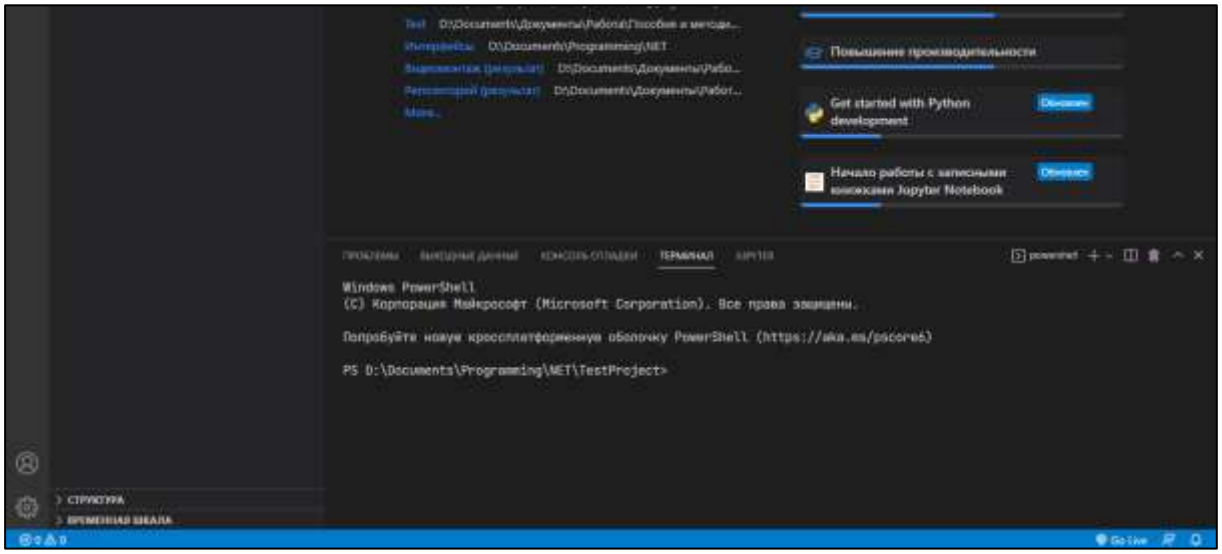


Рис. 1.154. Работа с командной строкой внутри редактора

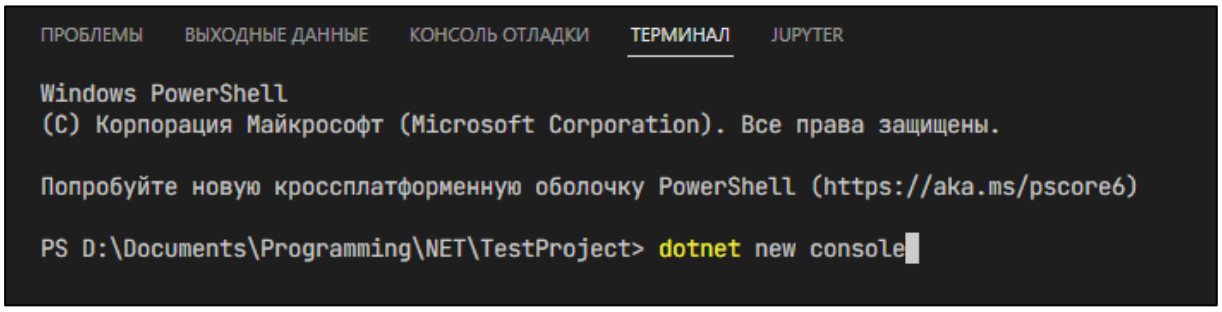


Рис. 1.155. Создание нового консольного приложения на языке C#

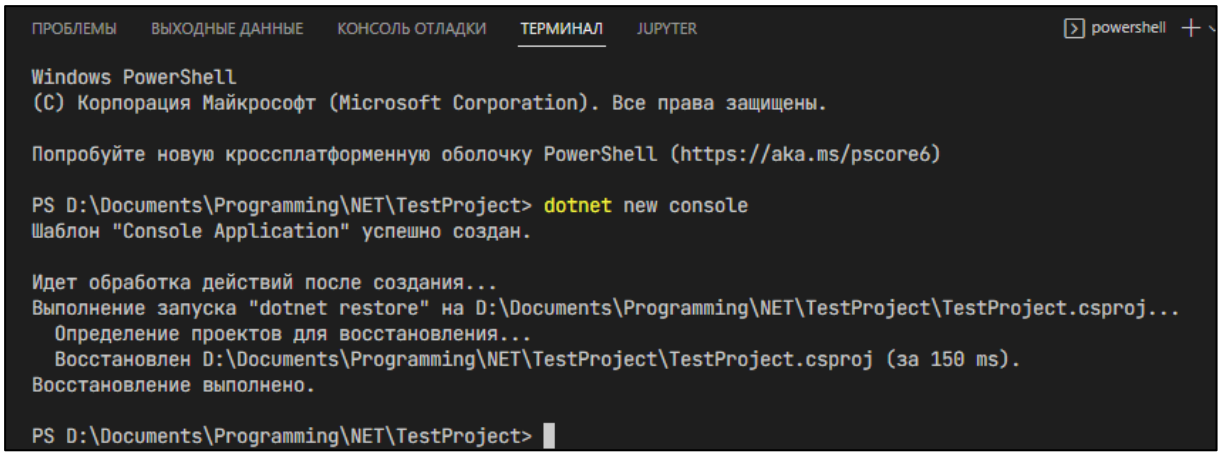


Рис. 1.156. Статус создания стартового шаблона и дополнительных файлов

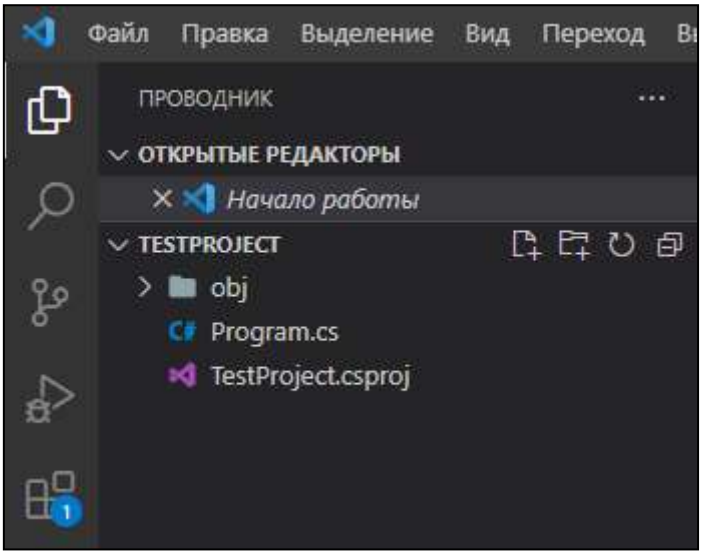


Рис. 1.157. Сгенерированные файлы проекта в проводнике

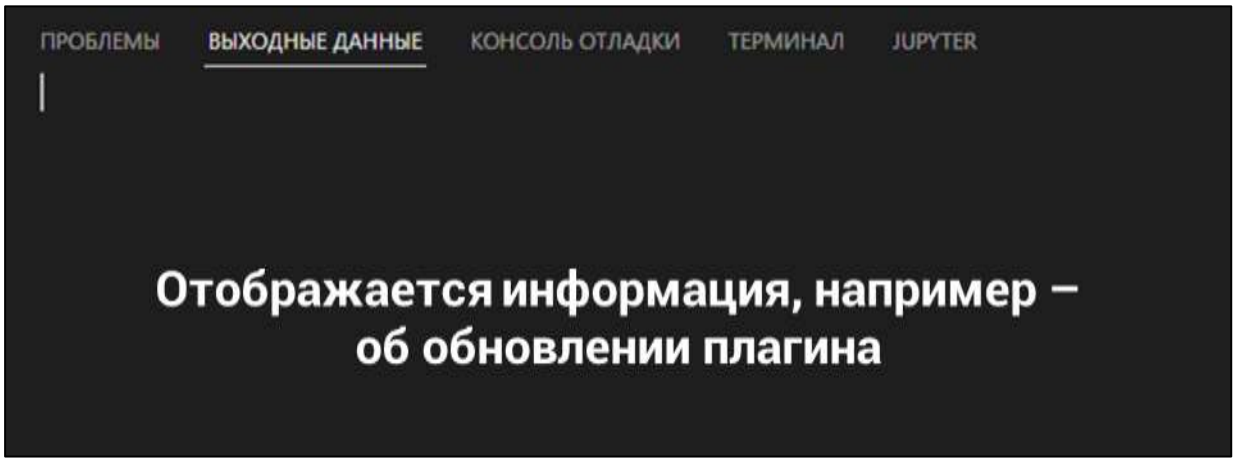


Рис. 1.158. Область вывода дополнительной информации

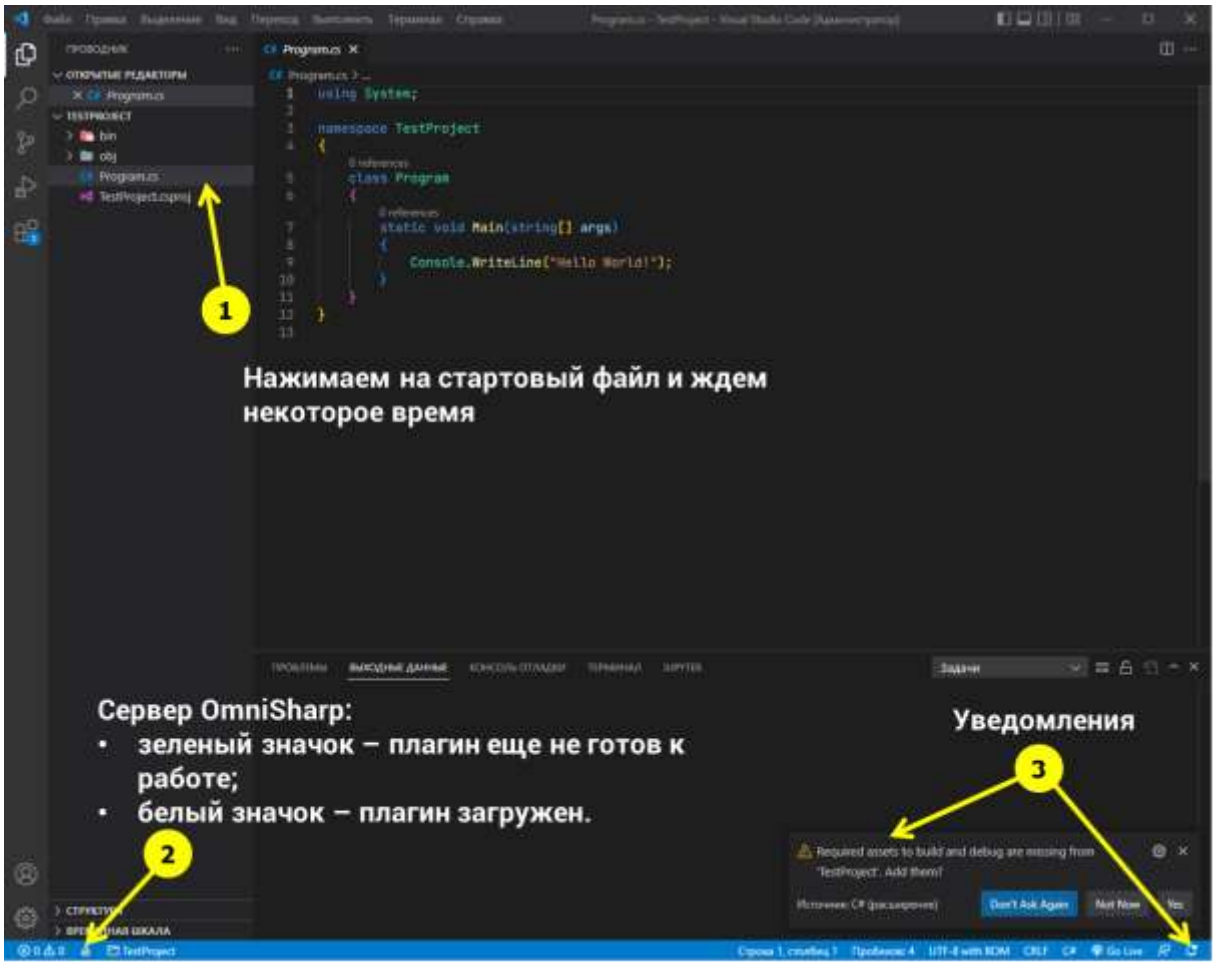


Рис. 1.159. Загрузка рабочих процессов для работы с .NET и C#

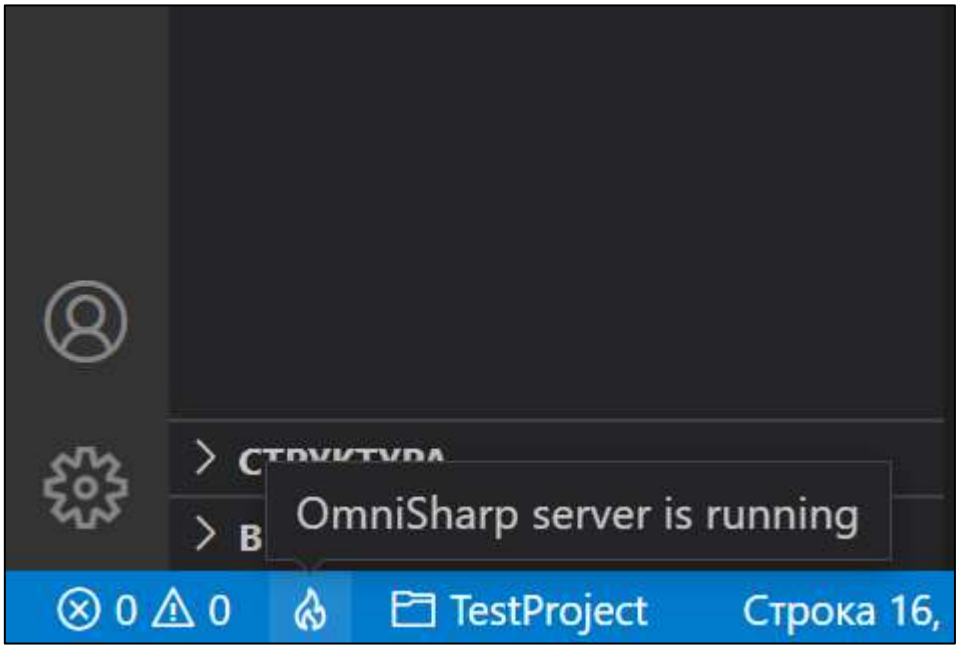


Рис. 1.160. Окончательная загрузка плагина OmniSharp

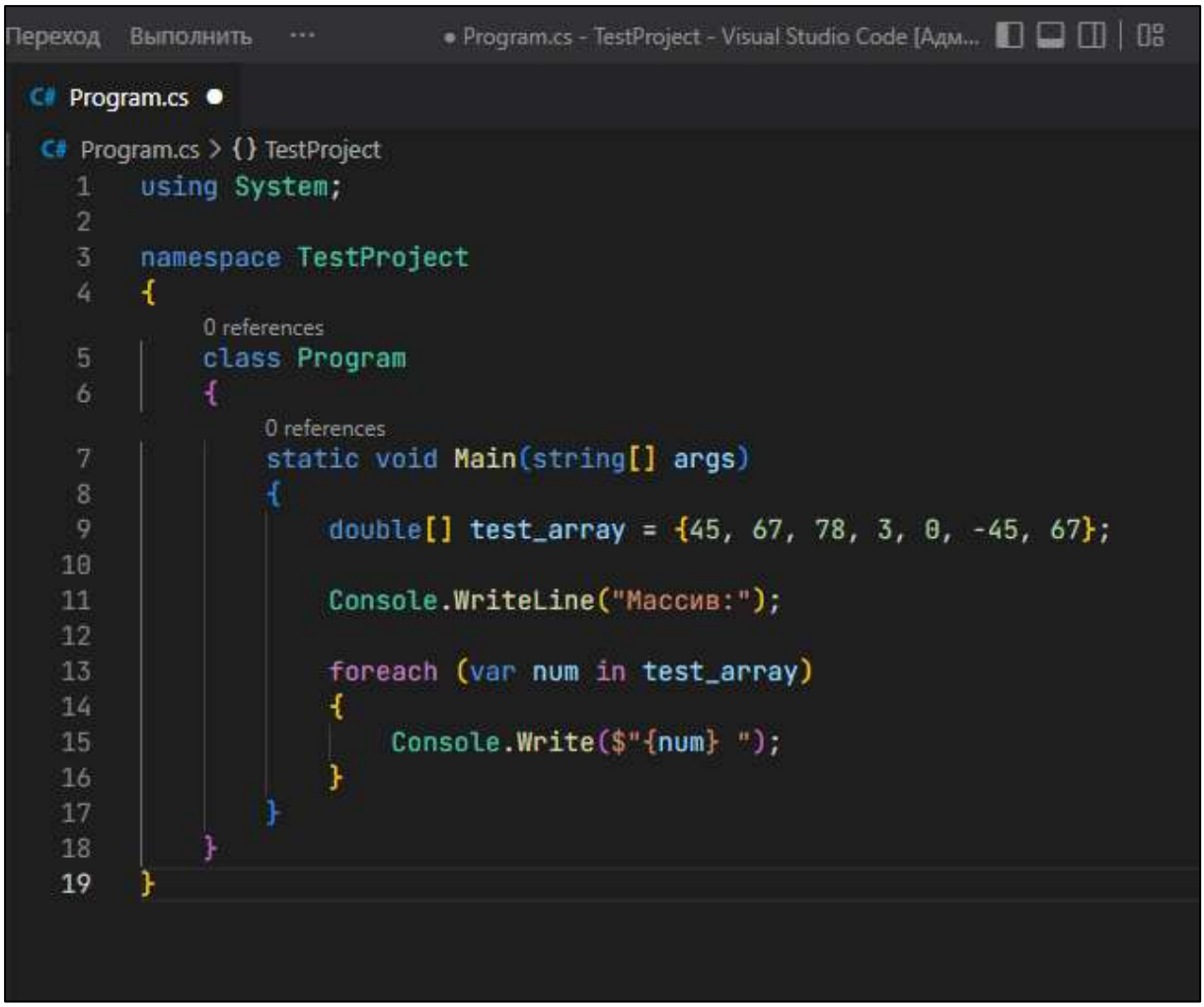


Рис. 1.161. Редактирование кода

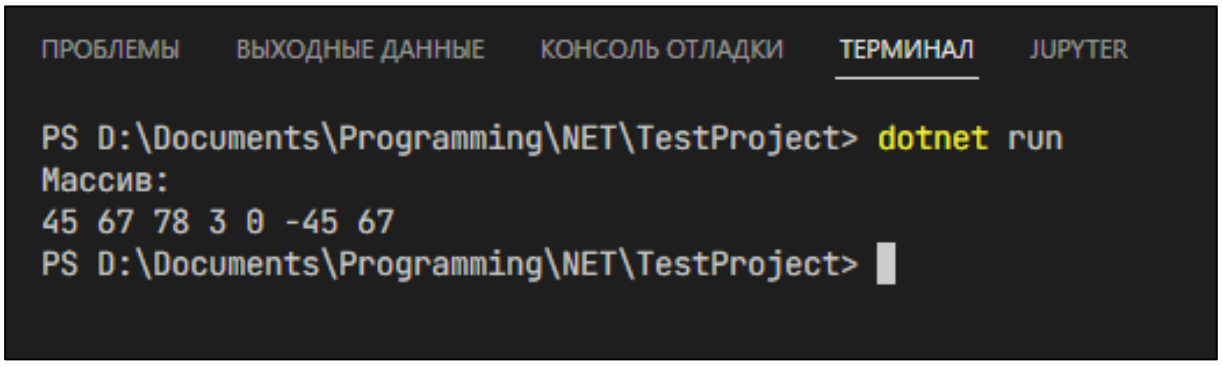


Рис. 1.162. Компиляция и запуск программы

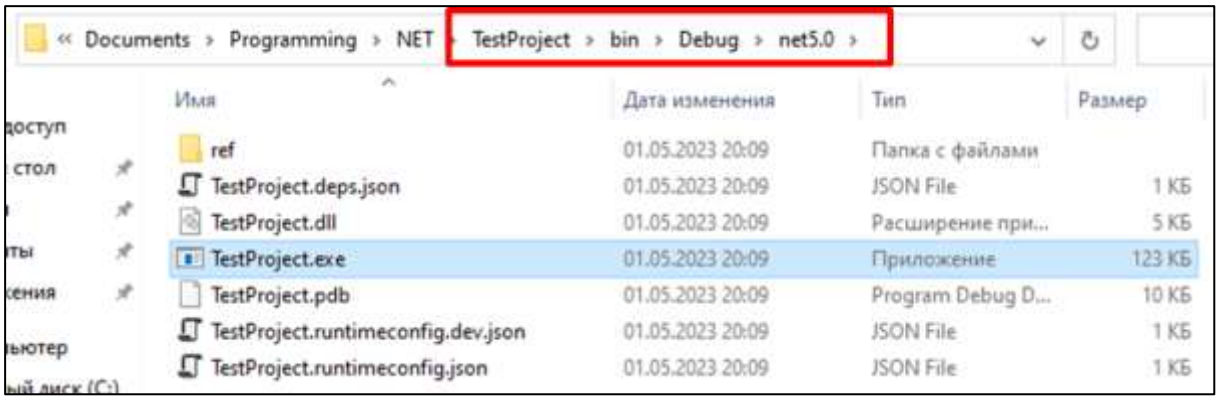


Рис. 1.163. Исполняемый файл приложения, может быть перемещен и запущен как автономная программа

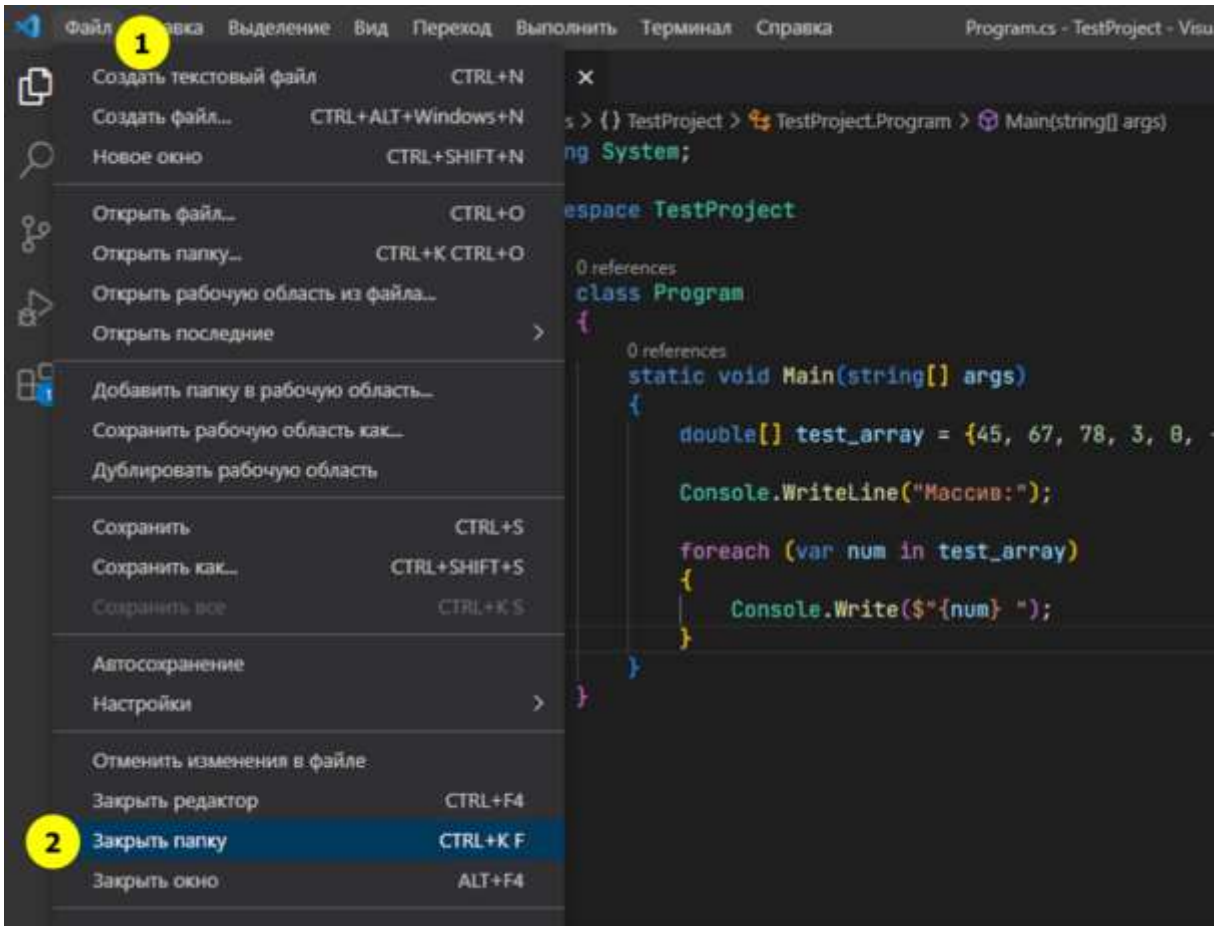


Рис. 1.164. Выгрузка каталога проекта из рабочего пространства

Задание 5

1. Протестируйте некоторые возможности технологии IntelliSense.
2. Начните ввод команды с первых нескольких символов:

   ```
   Console.Write("Привет!");
   ```

   Редактор отобразит выпадающий список возможных команд, попадающих бод указанное начало (рис. 1.165, п. 1).
3. С помощью стрелок клавиатуры можно сместиться выше или ниже по списку.
4. Если необходимая команда уже подсвечена, то нажмите *Tab*, редактор автоматически допишет название класса *Console*.
5. Поставьте точку (рис. 1.165, п. 2). Вновь сплывает автоматическая подсказка, но уже относительно компонент этого класса (константы, свойства, методы или другое).
6. Введите несколько первых символов метода *Write* и нажмите *Tab* (рис. 1.165, п. 3-4).
7. Убедитесь, что Visual Studio Code автоматически пишет парные скобки и кавычки. Также обратите внимание, что слева от каждой команды в списке указан значок, обозначающий роль элемента (рис. 1.165, п. 5).
8. Ниже начните ввод цикла *foreach*. В списке подсказки выберите вариант команды со значком прямоугольника и нажмите *Tab* (рис. 1.166, п. 1). Будет вставлен каркас цикла *foreach* (*сниппет*), как на рис. 1.166, п. 2.
9. Повторно нажимая *Tab* можно переместиться к некоторым элементам сниппета, чтобы изменить их названия на типы данных и переменные. (рис. 1.166, п. 3)
10. В случае, если подсказка пропала, нажмите комбинацию *Ctrl + Пробел*.

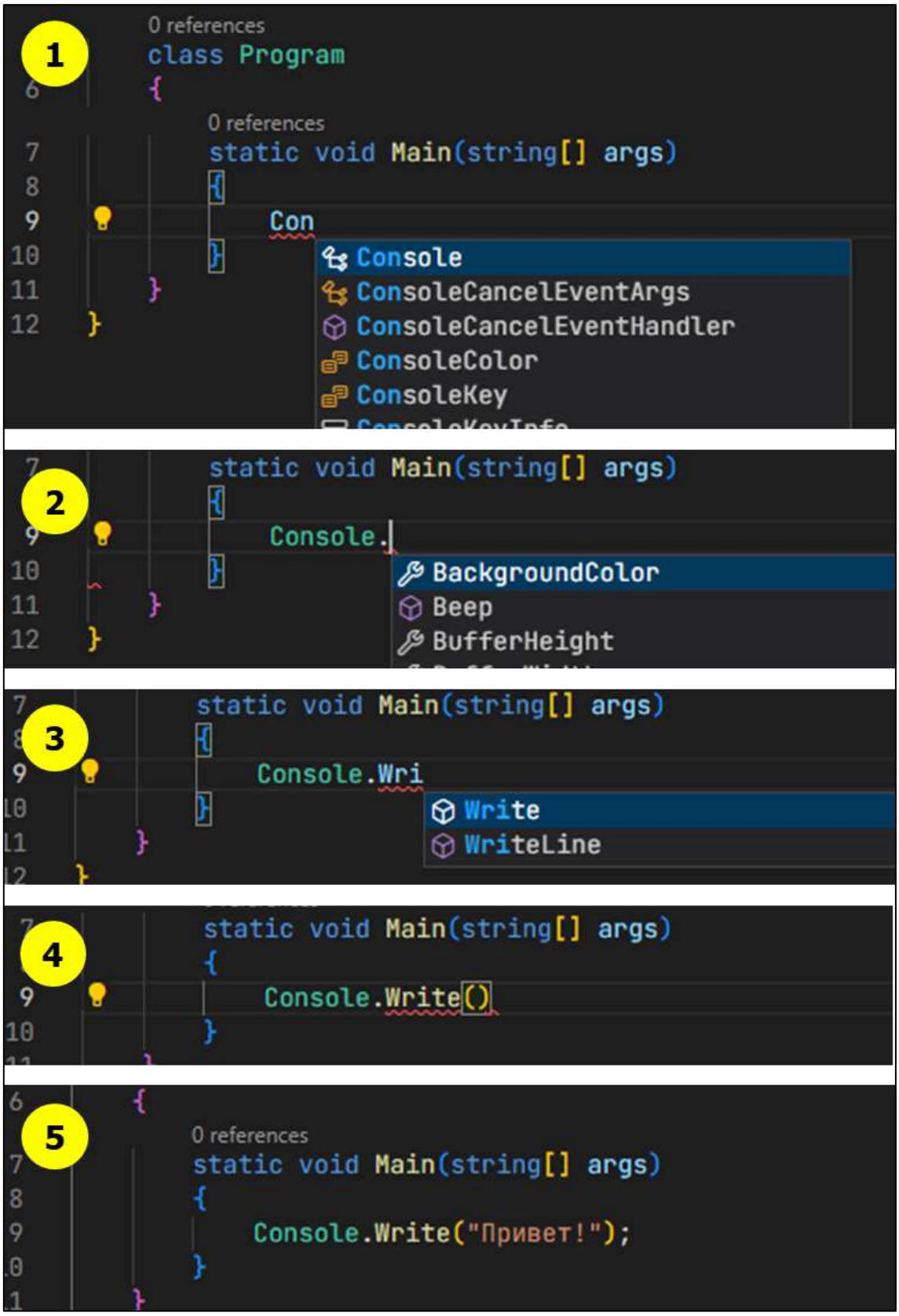


Рис. 1.165. Автоматическое дополнение инструкций

Рис. 1.166. Быстрая вставка шаблона инструкции

Задание 6

1. Наберите следующий программный код:

```
using System;

namespace TestProject
{
    class Program
    {
        static void Main(string[] args)
        {
            double[] test_array = {
                45, 67, 78, 3, 0, -45, 67
            };
```

```
            PrintArray(test_array);
        }

        static void PrintArray(double[] arr)
        {
            foreach (double num in arr)
            {
                Console.Write($"{num} ");
            }
        }
    }
}
```

2. Поставьте курсор на команду *foreach*. Слева появится символ лампочки. Щелкните на нем и выберете *Преобразовать в for*. Это пример операции, обратной рефакторингу: цикл *for* в текущем контексте менее рационален и приводит к увеличению объема кода.
3. Повторите процедуру и вернитесь к циклу *foreach*.

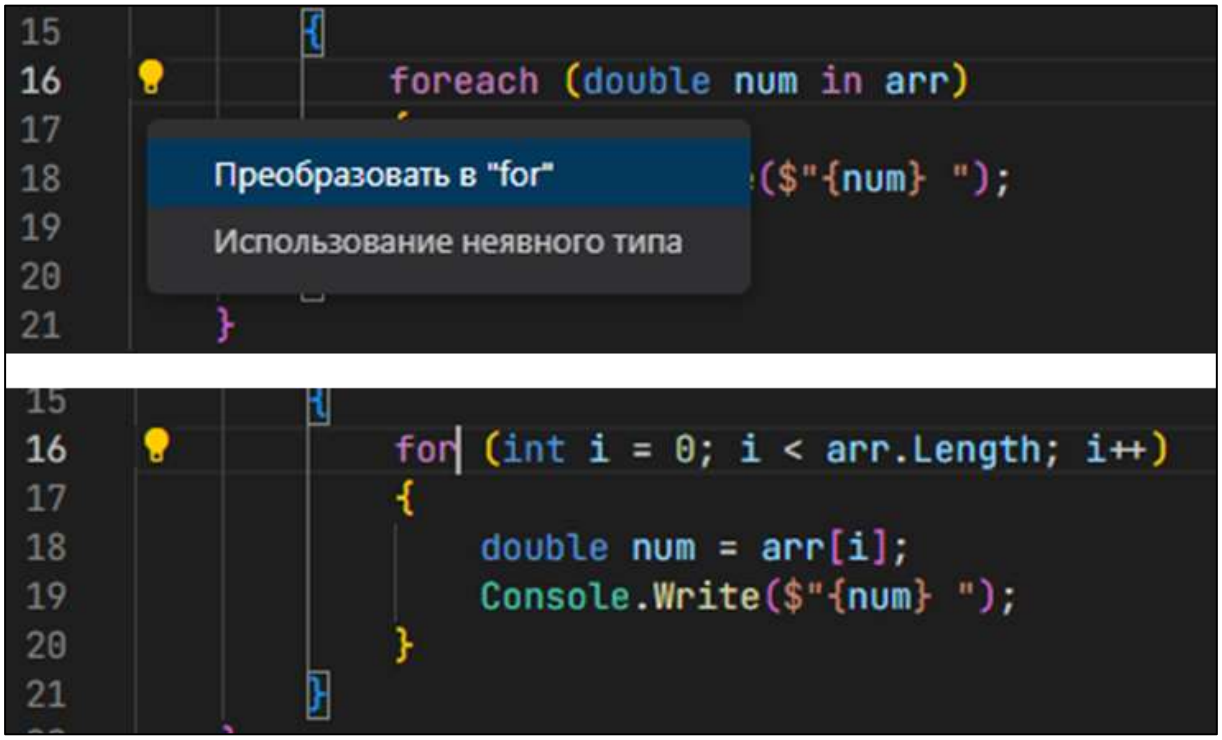


Рис. 1.167. Пример автоматического рефакторинга средствами редактора

1.6. Технологии разработки веб-сайтов

1.6.1. Frontend-разработка

Языки программирования и разметки

HTML

Определение

HTML *(от англ.* ***H****yper****T****ext* ***M****arkup* ***L****anguage) — это язык гипертекстовой разметки веб-страницы.*

Концепции будущего HTML закладывались еще в 1986 году, когда был разработан язык SGML, необходимый для разметки текста и оформления веб-страниц. Однако SGML не предусматривал разделения структурных элементов разметки в зависимости от определенных условий. В нем был реализован принцип, который далее лег в основу HTML – наличие *тегов* в разметке.

Требовалось создать язык разметки, который:

- четко устанавливает правила и условия применения тегов для разных элементов;
- имеет фиксированный набор тегов, позволяющих описать любой веб-документ и его структуру;
- будет универсальным и интерпретируемый разными программами.

В 1989 году Тим Бернерс-Ли предлагает концепцию HTML, которая из частного проекта CERN со временем перерастает в самый популярный и распространенный язык разметки современного веб.

HTML задает структуру разметки веб-страниц, разбивая его с помощью тегов на блоки, текстовые абзацы, заголовки, списки, таблицы, изображения, гиперссылки, мультимедийные элементы, формы и прочее. Каждый тег определяет роль элемента в разметке и правила его базового оформления. За форматирование документа отвечает технология CSS, которая оформляет элементы разметки согласно заданным стилям и правилам их иерархии.

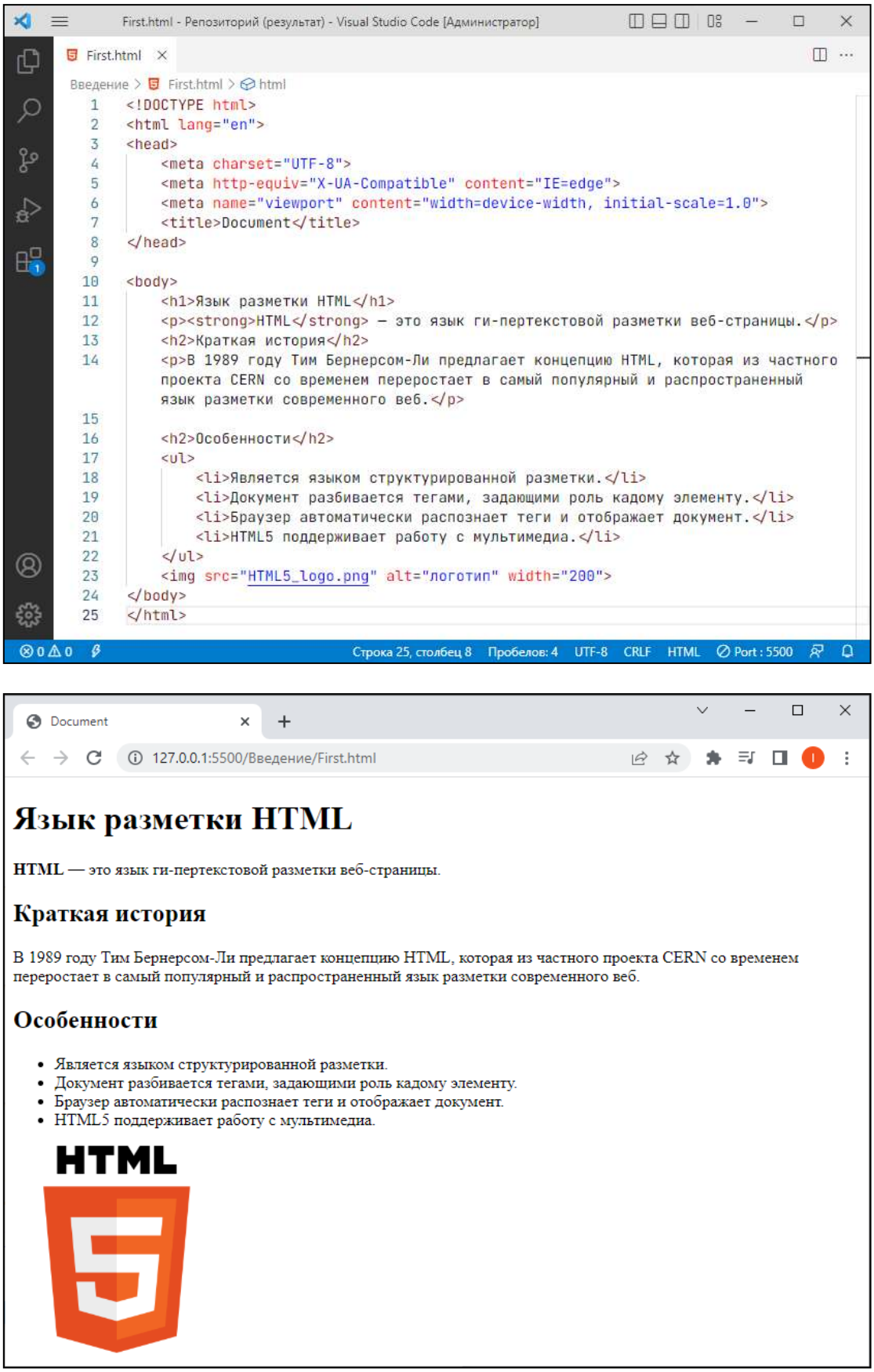


Рис. 1.168. HTML-разметка и отображение документа в браузере

Стоит отметить, что длительное время HTML содержал теги и атрибуты, которые позволяли физически форматировать внешний вид документа, что несколько нарушало концепцию HTML как языка исключительно логической разметки и структуризации.

С приходом стандарта HTML5 неоднозначность была снята и HTML «очистили» от тегов физического форматирования. Тем не менее браузеры продолжают поддерживать работу со старыми сайтами и версиями HTML.

Особенности и применение HTML:

- Является *семантическим* языком, т.е. теги задают определенную роль элементам, которые они размечают.
- Теги HTML не отображаются в браузере.
- Браузер автоматически распознает теги и отображает документ в надлежащей форме.
- Язык удобен как для автоматической обработки, так и достаточно естественен для человека.
- Обладает высокой степенью универсальности при описании веб-документов несмотря на ограниченное число тегов.
- Поддерживается всеми современными браузерами.
- Это один из простейших языков в изучении.

Это полезно знать!

Говоря о языке HTML его следует понимать именно в качестве языка разметки. HTML не предоставляет каких-либо инструментов построения логики функционирования, как это свойственно языкам программирования.
Тем не менее HTML обладает строгим синтаксисом, семантикой и лексикой, что позволяет его отнести к классу декларативных языков программирования.

CSS

Определение

CSS *(от англ.* ***C****ascading* ***S****tyle* ***S****heets) – это формальный язык описания внешнего вида веб-страницы, описанной с использованием языка разметки.*

Зарождение концепции CSS было связано с необходимость стандартизировать инструменты веб-разработки. Впервые CSS предложил использовать Хок Виум Ли в 1994 году. 17 декабря 1996 года появилась первая спецификация каскадных таблиц стилей CSS1, которая была рекомендована к использованию Консорциумом Всемирной паутины (W3C).

Последний актуальный стандарт – CSS3, который позволяет не только форматировать и позиционировать элементы разметки, но и работать с анимацией, фильтрами, трансформацией и прочими эффектами, которые ранее приходилось программировать на JavaScript.

CSS позволил отделить разметку структуры документа от его оформления, что позволило систематизировать подход к верстке веб-страниц и упростить их дальнейшую модификацию. Современные HTML5 и CSS3 работают совместно (хотя CSS этого не требует).

В CSS используется понятие *селектора* как элемента, к которому может быть привязан стиль оформления. В качестве селектора могут выступать теги, атрибуты тегов и определенные значения атрибутов. Описание стилей CSS единообразно и состоит из пар «ключ : значение». Каждый ключ является некоторым фиксированным свойством CSS, которое может принять одно из нескольких допустимых постоянных либо произвольных числовых значений. Одному селектору можно назначить несколько свойств.

```
h1 {
    font-family: Tahoma;
    font-size: 2.0em;
    color: brown;
}
```

Рис. 1.169. Описание стиля селектора тега заголовка `<h1></h1>`

CSS обладает следующими возможностями:

- Независим от разметки и может подключаться как сторонний файл стилей, влияющих на внешний вид HTML-документа.
- Простой и единообразный синтаксис.
- Возможность быстрого изменения дизайна веб-страницы с помощью подключения других каскадных таблиц стилей. С другой стороны один CSS-файл может раздельно использоваться несколькими HTML-страницами (рис. 1.170).
- CSS3 поддерживает мультимедийные эффекты и работу с адаптивным дизайном (рис. 1.171).
- Ускоряют загрузку страниц и позволяют экономить трафик при обновлении данных.
- Хорошая совместимость с разными платформами в силу соответствия современным и единым стандартам.

Несмотря на то, что стандарт CSS3 используется уже длительное время, в некоторых браузерах наблюдается проблема интерпретации новых свойств. Однако постепенно она решается обновлением функционала браузеров.

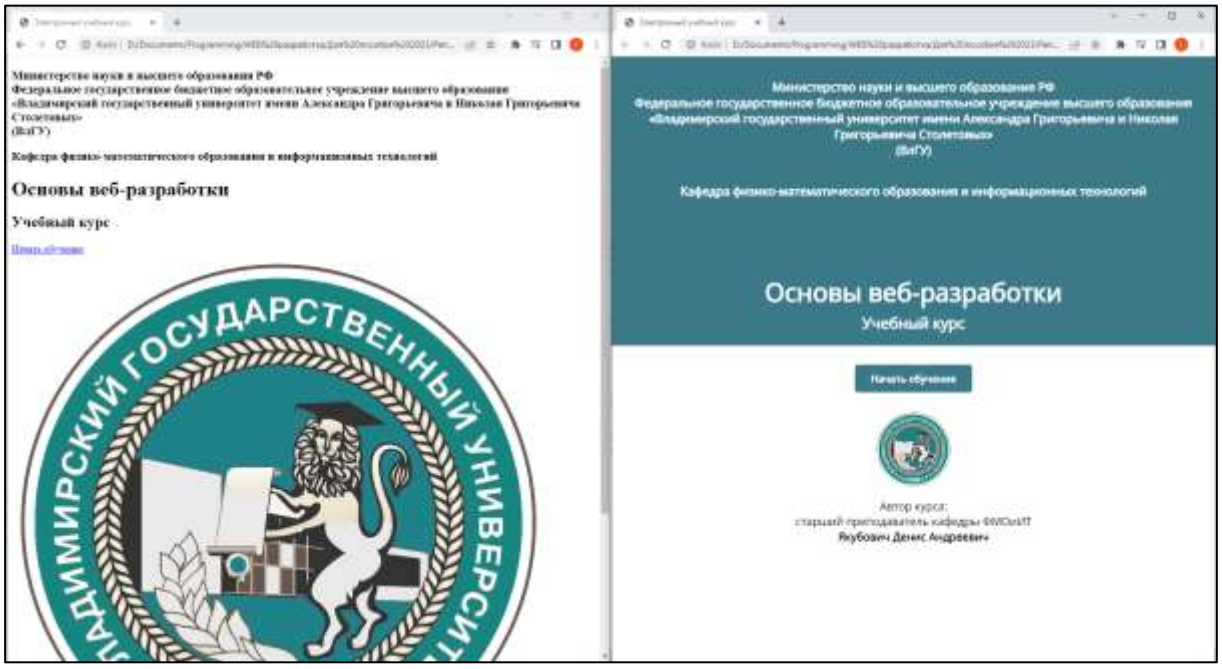


Рис. 1.170. HTML-документ без CSS и с CSS-стилями (структура HTML в обоих случаях одна и та же)

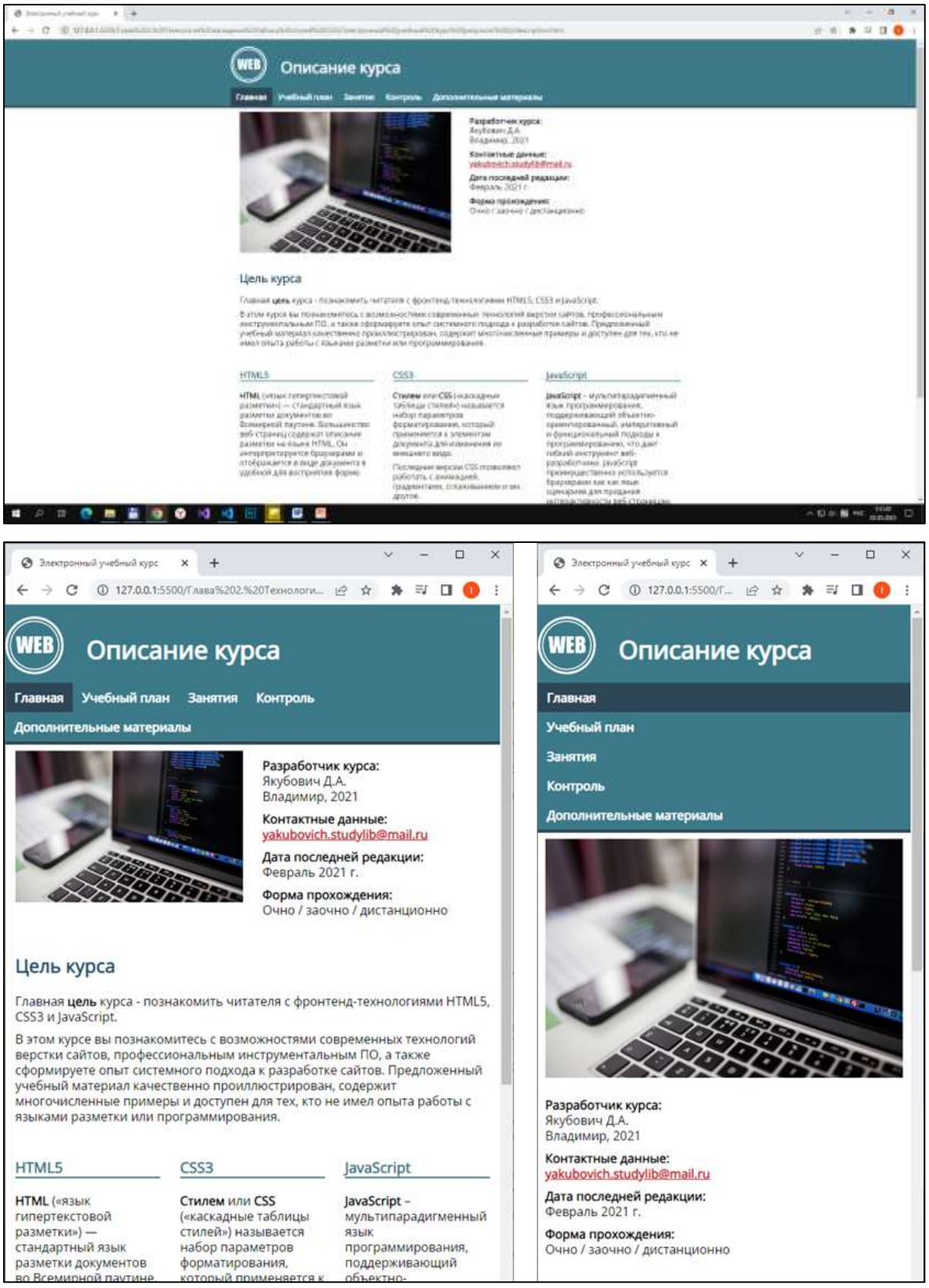


Рис. 1.171. Пример адаптивного дизайна, подстраивающегося под ширину экрана (реализовано возможностями CSS3)

JavaScript

Определение

JavaScript *– язык, объединяющий несколько парадигм программирования и используемый как встраиваемый язык для программного доступа к объектам приложений.*
Обычно используется как язык сценариев для придания интерактивности веб-страницам.

JavaScript является одним из самых востребованных языков программирования современности и прочно занимают лидирующие позиции в веб-программировании. JavaScript отвечает за функциональную обработку данных и управляющих элементов веб-страницы, т.е. за функционал на стороне клиента. Без преувеличения JavaScript является фундаментом и основным языком программирована frontend-разработки.

Модель JavaScript предполагает работу с событиями – определенные условия, при возникновении которых необходим ответ со стороны системы (нажатие клавиши, клик мыши, изменение размера, положения окна и прочее). JavaScript базируется на стандарте ECMAScript, который описывает правила построения скриптовых языков.

JavaScript используется в программировании широкого класса задач:

- разработка веб-сайтов и веб-приложений (рис. 1.172);
- плагины для браузеров;
- приложения для мобильных устройств;
- серверный функционал сайта или программы (требуется движок Node.js);
- браузерные игры.

Высокая популярность языка связана с его необычайной гибкостью и постоянным развитием (рис. 1.173). Синтаксис JavaScript во многом похож на C++, C#, Java, однако концептуально и функционально от них существенно отличается.

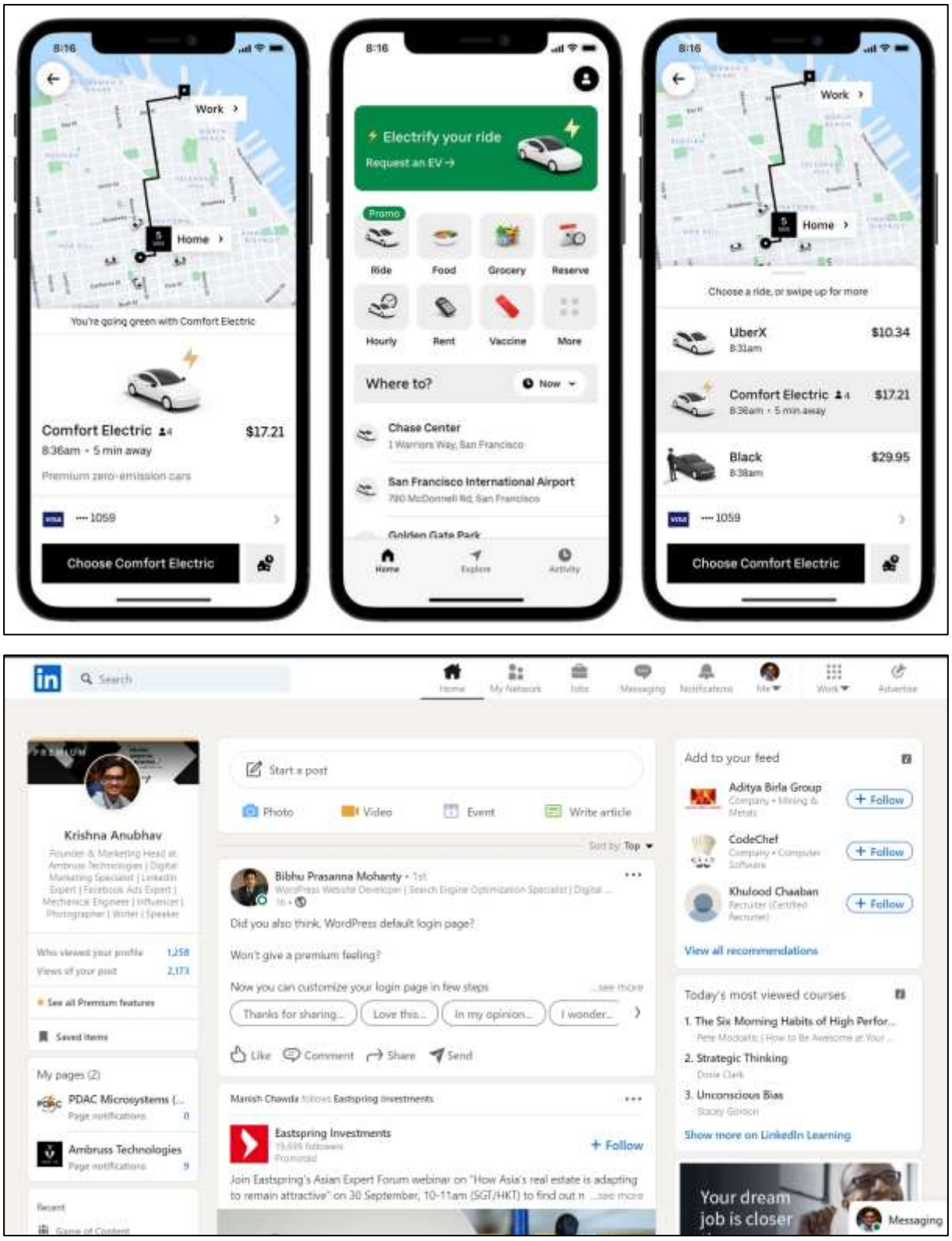


Рис. 1.172. Uber и LinkedIn – популярные приложения, разработанные на базе JavaScript

К особенностям языка JavaScript относят следующие:

- JavaScript является *интерпретируемым*. Это ускоряет разработку, поскольку интерпретатор уже встроен в браузер.
- Язык является *мультипарадигменным*. Он поддерживает объектно-ориентированное, функциональное и императивное программирование.
- JavaScript *динамически типизированный* язык, что не требует явного указания типа переменной. Более того, в процессе работы приложения переменная может принимать значения разного типа и даже автоматически преобразовываться к требуемому, если это допустимо.
- Язык качественно интегрирован с HTML и CSS. Он содержит встроенные функции для работы с разметкой и стилями, что и позволяет гибко управлять динамикой изменения содержимого веб-страниц.
- JavaScript – основа многих фреймворков и библиотек, которые упрощают создание крупных проектов.

Несмотря на высокую гибкость, JavaScript зачастую используется в обучении программированию, как первый профессиональный язык разработки ПО. Базовый синтаксис языка достаточно прост и позволяет изучить особенности управляющих конструкций, объектную модель, структуры данных, иерархию связей. Разработчики на JavaScript без особого труда смогут перейти к изучению «сиобразных» языков: C++, C#, Java, PHP (и наоборот).

Кроме того, JavaScript поддерживается многими средами разработки и текстовыми редакторами.

JavaScript в силу политики безопасности может управлять вкладками браузера только при определенных ограничениях. Также он не может читать и записывать файлы на диск, обрабатывать видеопоток с веб-камеры и получать данные с сервера без разрешения пользователя.

Расширить возможности языка позволяют технологии Node.js и React Native.

```
const fullname = document.getElementById("fullname");
const studentRating = document.getElementById("rating");
const getData = document.getElementById("getData");
const studentMark = document.getElementById("student-mark");

getData.onclick = function () {
    let fullName = fullname.value;
    let points = studentRating.value;
    let mark = 0;

    if (points ⩾ 90)
        mark = 5;
    else if (points ⩾ 75)
        mark = 4;
    else if (points ⩾ 61)
        mark = 3;
    else
        mark = 2;

    studentMark.innerHTML = `<p>Баллы студента ${fullName}: ${points}</p>`;
    studentMark.innerHTML += `<p>Оценка: ${mark}</p>`;
}
```

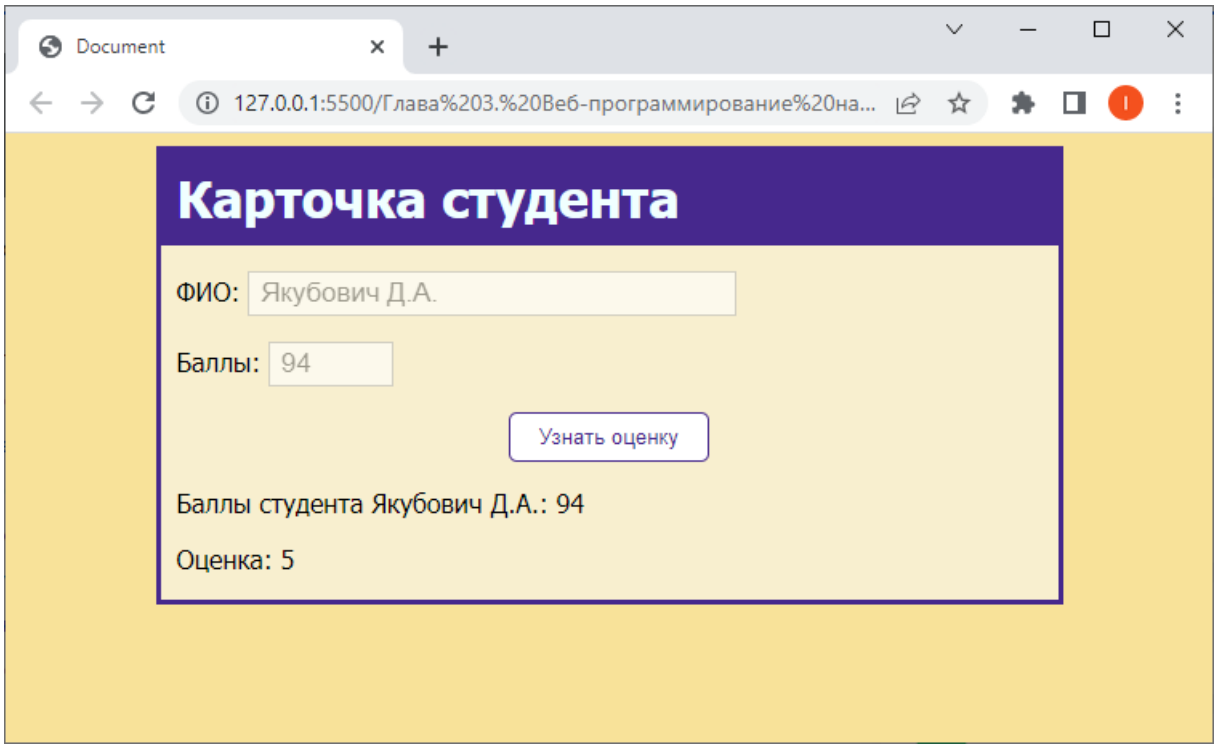


Рис. 1.173. Пример учебного проекта. JavaScript позволяет динамически управлять HTML-разметкой и CSS-стилями веб-страницы, а также программировать функционал

TypeScript

Определение

TypeScript *– это язык программирования от Microsoft, позиционируемый как расширение JavaScript.*

TypeScript был разработан в 2012 году Андерсом Хейлсбергом, который является родоначальником не менее известных языков Turbo Pascal, Delphi и C#.

Необходимость в создании TypeScript была связана с многочисленными проблемами, которые возникали у разработчиков. Гибкость JavaScript зачастую усложняла разработку больших проектов и выстраивание архитектур со сложными иерархическими зависимостями.

TypeScript включает поддержку стандарта ECMAScript и расширяет синтаксис понятиями классов и интерфейсов, что свойственно классическим объектно-ориентированным языкам.

TypeScript обратно совместим с JavaScript, т.е. перед выполнением код на TypeScript автоматически компилируется в код на JavaScript и уже далее интерпретируется браузером.

Важным отличием TypeScript является работа с явно типизированными данными (рис. 1.174). При описании переменным указывается определенный тип данных, что повышает безопасность операций и в целом уменьшает вероятность допустить ошибки в реализации скрипта.

Разработчики TypeScript предполагают, что в будущем JavaScript станет в некотором смысле *ассемблером* для веб-технологий, а более современные языки программирования будут использовать его в качестве промежуточного при трансляции программ в машинные коды.

Планируется, что в силу полной обратной совместимости адаптация существующих приложений на новый язык программирования может происходить поэтапно, путём постепенного определения типов. В частности, некоторые популярные библиотеки JavaScript постепенно переводятся и в формат TypeScript.

Перечислим особенности TypeScript:

- *Строгая типизация данных*. Тип переменной указывается явно и ограничивает допустимые с ней операции.
- Работа с *классами* и *интерфейсами*. Подобные абстракции востребованы в построении иерархических структур данных и масштабировании проектов.
- Возможность работать с библиотеками JavaScript.
- Активное развитие языка компанией Microsoft.

code.ts

```ts
const num1 = <HTMLInputElement>document.getElementById("num1");
const num2 = <HTMLInputElement>document.getElementById("num2");
const result = document.getElementById("result");
const btn = document.getElementById("btn");

btn.addEventListener("click", (e) => { Sum(); });

function Sum()
{
   let sum : number = num1.valueAsNumber + num2.valueAsNumber;
   result.innerHTML = sum.toString();
}
```

code.js

```js
var num1 = document.getElementById("num1");
var num2 = document.getElementById("num2");
var result = document.getElementById("result");
var btn = document.getElementById("btn");

btn.addEventListener("click", function (e) { Sum(); });

function Sum() {
   var sum = num1.valueAsNumber + num2.valueAsNumber;
   result.innerHTML = sum.toString();
}
```

Рис. 1.174. Код на TypeScript (code.ts) и соответствующий ему код на JavaScript (code.js). В первом случае используется явная типизация данных

Среди недостатков TypeScript отмечают увеличение времени на разработку, некоторые несовершенства классовой модели (по сравнению с теоретической) и необходимость преобразования кода в JavaScript перед запуском.

TypeScript эффективен в написании комплексных решений, которые легче развивать и тестировать, чем на JavaScript. TypeScript уже используется в ряде крупных фреймворков, например – в Angular.

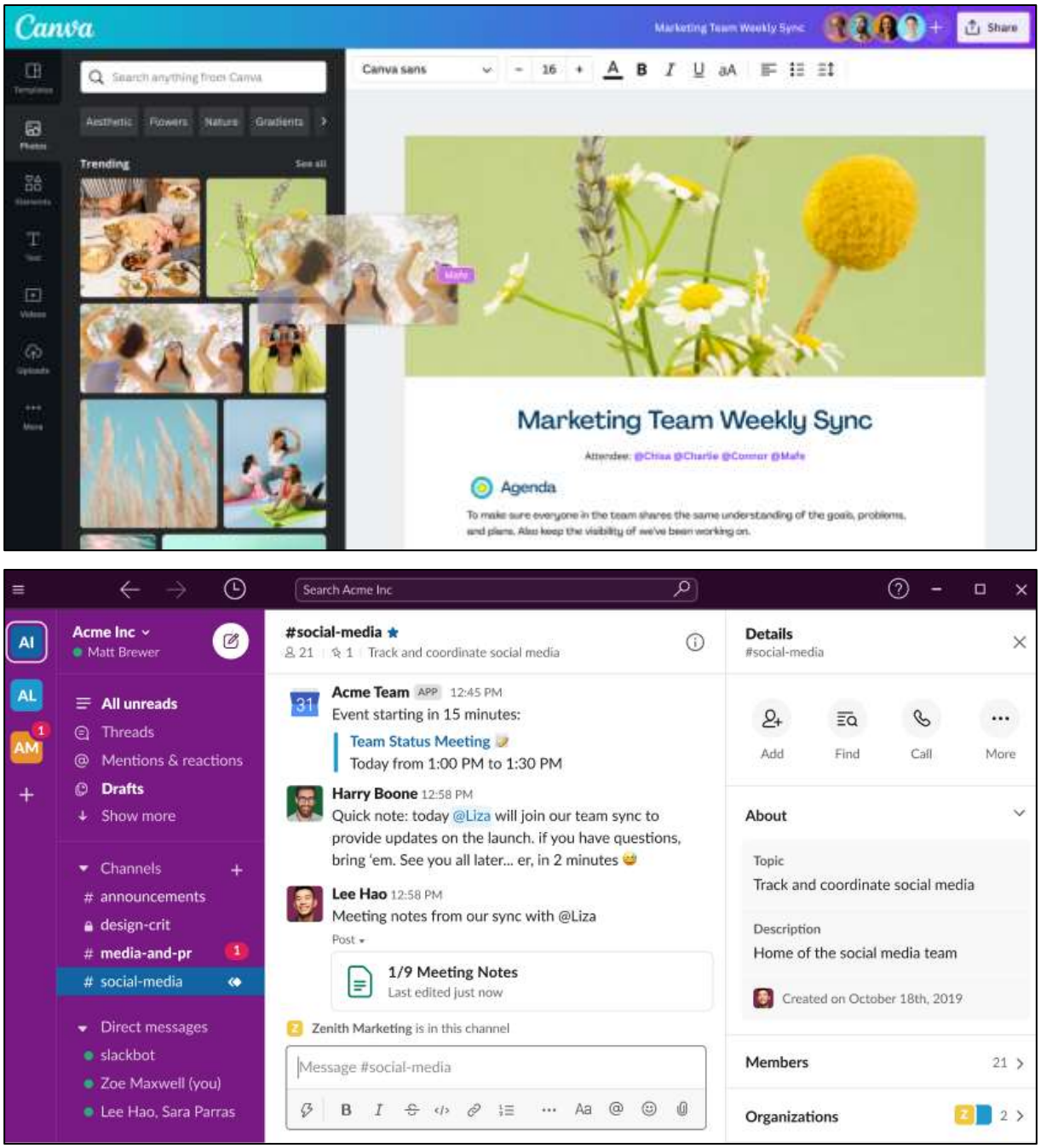


Рис. 1.175. Canva и Slack разработаны на базе TypeScript

Swift

Определение

Swift *– это мультипарадигменный компилируемый язык общего назначения, разработанный компанией Apple.*

Swift был представлен в 2014 году как альтернатива языку программирования Objective-C. Последний имел множество проблем и в целом считается устаревшим. Swift используется для frontend-разработки приложений под iOS, tvOS, macOS и watchOS.

Swift относится к классу компилируемых языков. Для разработчиков доступна удобная IDE XCode, предназначенную для macOS.

Язык программирования Swift используется в разработке:

- приложений для операционных систем macOS, iOS, устройств Apple;
- мобильных приложений под iOS (рис. 1.177);
- игр для системы Apple;
- веб-сайтов;
- оболочек, обеспечивающих работу серверной части сайтов и веб-приложений.

Характерными особенностями Swift являются:

- мультипарадигменность;
- статическая типизация (т.е. тип переменной определяется изначально и не может быть изменен);
- необязательное объявление типа.

Swift имеет ряд преимуществ:

- обеспечивает высокую скорость работы;
- автоматически управляет оперативной памятью;
- обладает достаточно простым для использования синтаксисом (рис. 1.176);
- обеспечивает хорошую безопасность;
- является проектом с открытым исходным кодом;
- поддерживается специалистами от Apple.

Среди недостатков Swift выделяют узкую специализацию и проблемы с поддержкой старых устройств.

code.swift

```swift
class User {

    var age: Int
    var fullname: String

    init(){
        age = 20
        fullname = "Tom"
    }

    func getUserInfo(){
        print("Имя: \(fullname). Возраст: \(age)")
    }
}

var tom: User = User()
tom.getUserInfo()
```

Рис. 1.176. Пример описания класса на языке Swift

Рис. 1.177. Swift использован в разработке мессенджера WhatsApp

Фреймворки и библиотеки

React

Определение

React.js *– это библиотека на основе языка программирования JavaScript, имеющая открытый исходный код и предназначенная для разработки пользовательских интерфейсов.*

Библиотека React используется в разработке одностраничных веб-сайтов и разработки крупных приложений. React удобен при решении следующих задач:

- создание интерактивных веб-интерфейсов;
- реализация веб-страниц или их компонентов;
- разработка структур со сложной архитектурой;
- доработки функциональности веб-сервиса;
- разработка мобильных приложений;
- обработка серверных данных.

Особенность React заключается в работе с HTML-разметкой документа. React создает виртуальное DOM-дерево (объектная модель документа), которая более эффективна в обработке. Это позволяет рационально и быстро («реактивно») обновлять отдельные элементы и не перезагружать всю страницу.

Концепция React основана на работе с компонентами, которые качественно защищают собственные данные и методы их обработки.

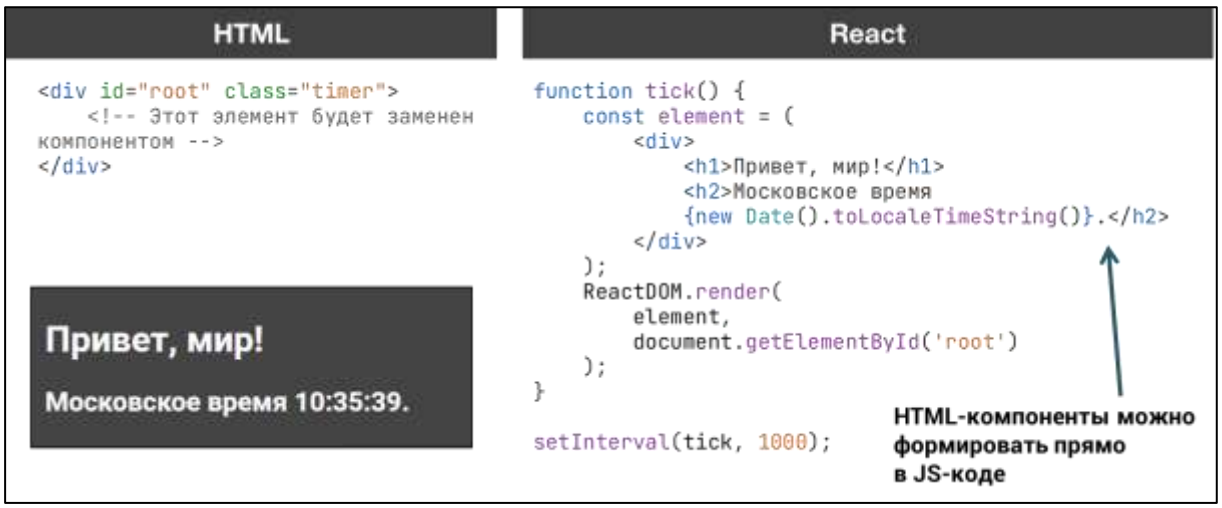
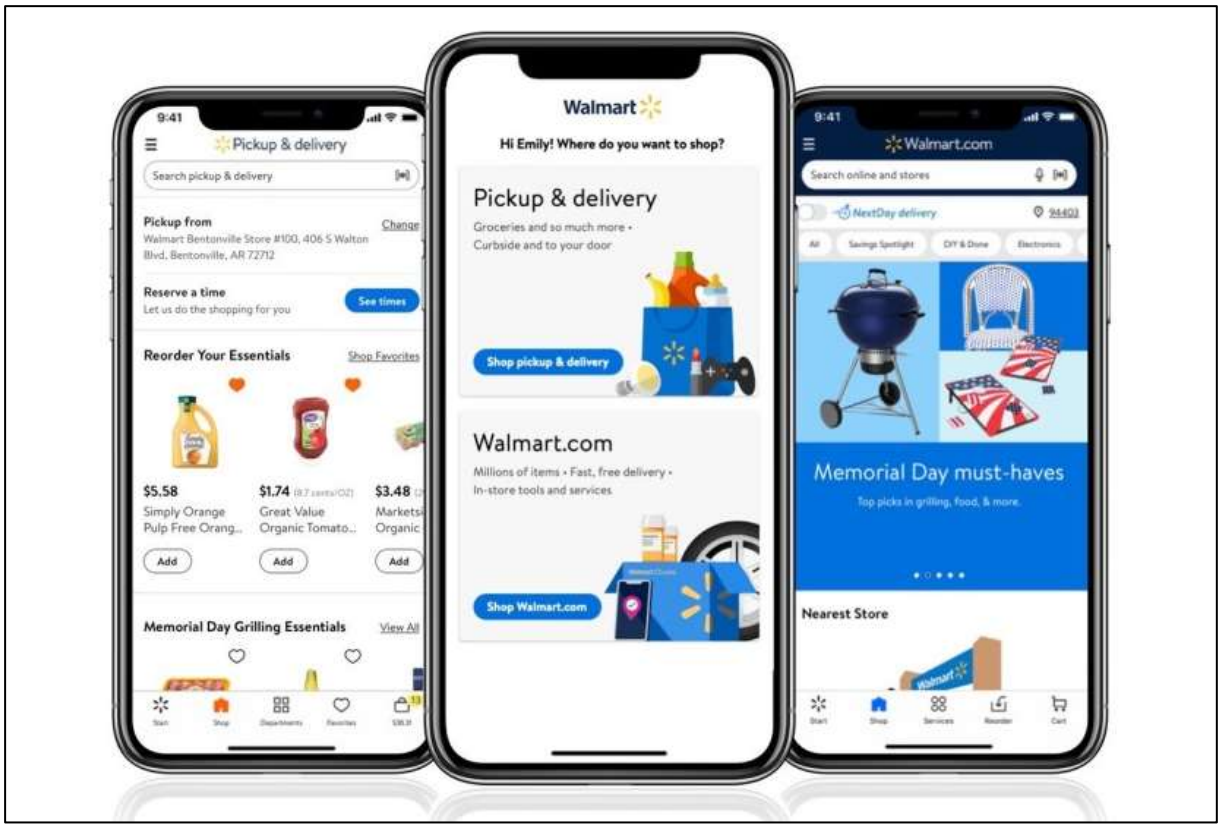


Рис. 1.178. Пример работы с компонентом React



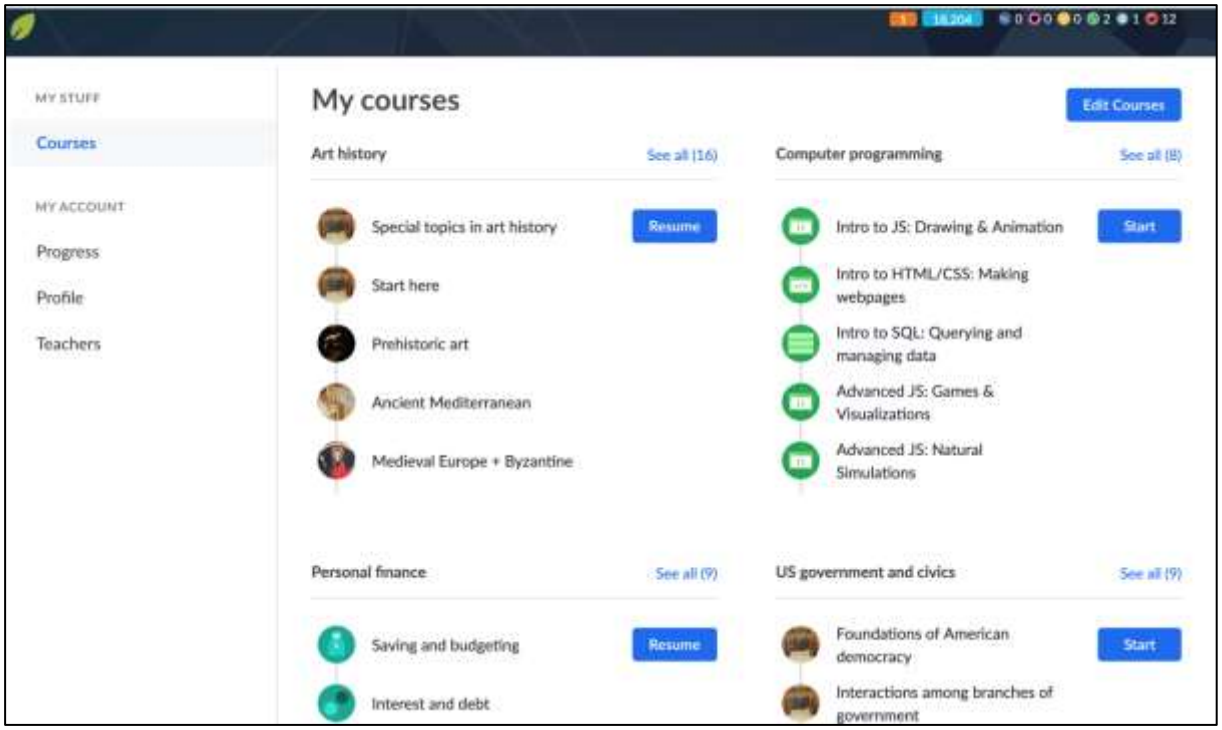


Рис. 1.179. Сервисы Walmart и Khan Academy написаны с использованием библиотеки React.js

Vue

Определение

Vue.js *— это фреймворк на языке JavaScript, применяемый для создания веб-интерфейсов с использованием архитектуры MVVM (Model-View-ViewModel).*

Впервые Vue.js был представлен в 2015 году, после чего она многократно обновлялась. В отличие от других популярных фреймворков (например, Angular, React), Vue не подвержен корпоративным тенденциям и поддерживается сообществом независимых разработчиков.

Vue является эффективным инструментом в разработке небольших проектов, которым необходимо расширить функционал, требующий работы с технологией AJAX. Vue позволяет легко масштабировать проекты и сохранять высокую скорость работы. Особенно Vue полезен в тех задачах веб-разработки, где требуется работа с внешними API.

Как и React, во Vue важную роль играю компоненты. Они могут использоваться многократно и настраиваться по усмотрению разработчика.

Другие характерные особенности Vue:

- работа с шаблонами;
- «реактивность»;
- работа с URL;
- развитые механизмы программирования графики, переходов, анимации.

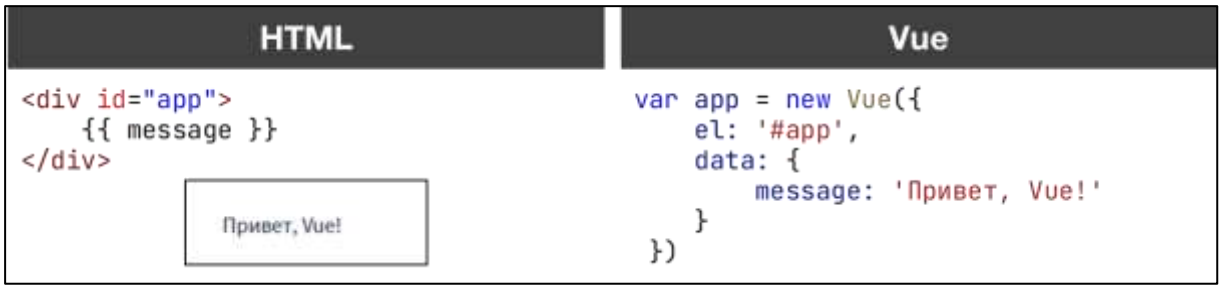


Рис. 1.180. Пример работы с компонентом Vue

Рис. 1.181. Сервис Chess.com разработан с помощью Vue

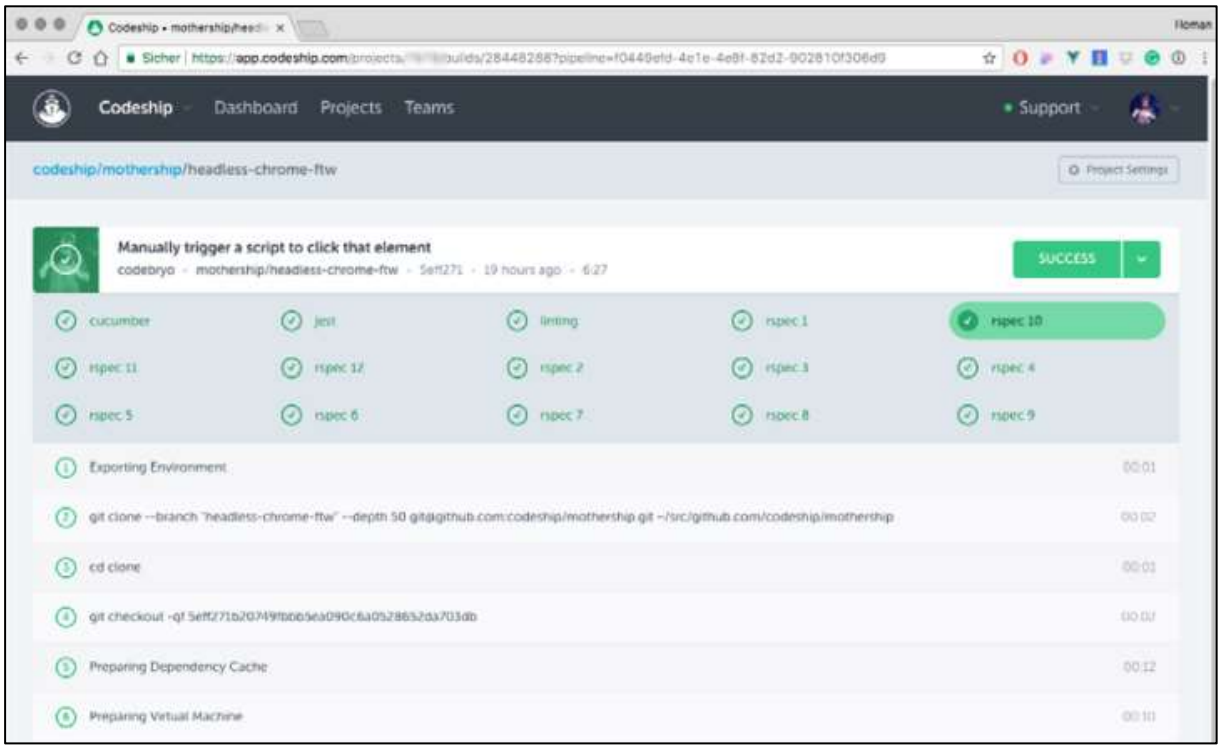


Рис. 1.182. Платформа Codeship разработана на основе Vue

Angular

Определение

Angular *– это фреймворк, разработанный компанией Google и предназначенный для создания одностраничных веб-приложений с использованием языков программирования TypeScript, JavaScript и Dart.*

Важной особенностью Angular является то, что он основан на TypeScript, однако требует и хорошего владения JavaScript, что зачастую несколько осложняет его изучение.

Фреймворк Angular достаточно универсален и не ограничивается возможностью создания веб-приложений. Допускается разработка функционала, который может быть адаптирован в другую среду. В частности, адаптация приложения под мобильную систему. С помощью Angular также создаются приложения для дополненной реальности.

Опишем некоторые важные элементы его концепции:

- Разбивает элементы приложения или сайта на *компоненты*, которые являются блоками ресурса. Каждый компонент описывается отдельно и может быть разбит на файлы с HTML-шаблоном, CSS-стилями и логикой поведения.
- *Модули* необходимы для управления компонентами.
- Содержит готовые шаблоны для *форм*, элементы которых адаптируются к возникающим у разработчика задачам.
- *Сервисы*, реализующие специальную логику: хранят глобальное состояние приложения и используются в качестве источников данных.
- *Директивы* отвечают за изменение структуры или поведения страницы. Компоненты также являются директивами.

Для работы с Angular необходимо установить платформу Node.js, менеджер пакетов Npm и обеспечить поддержку языка TypeScript.

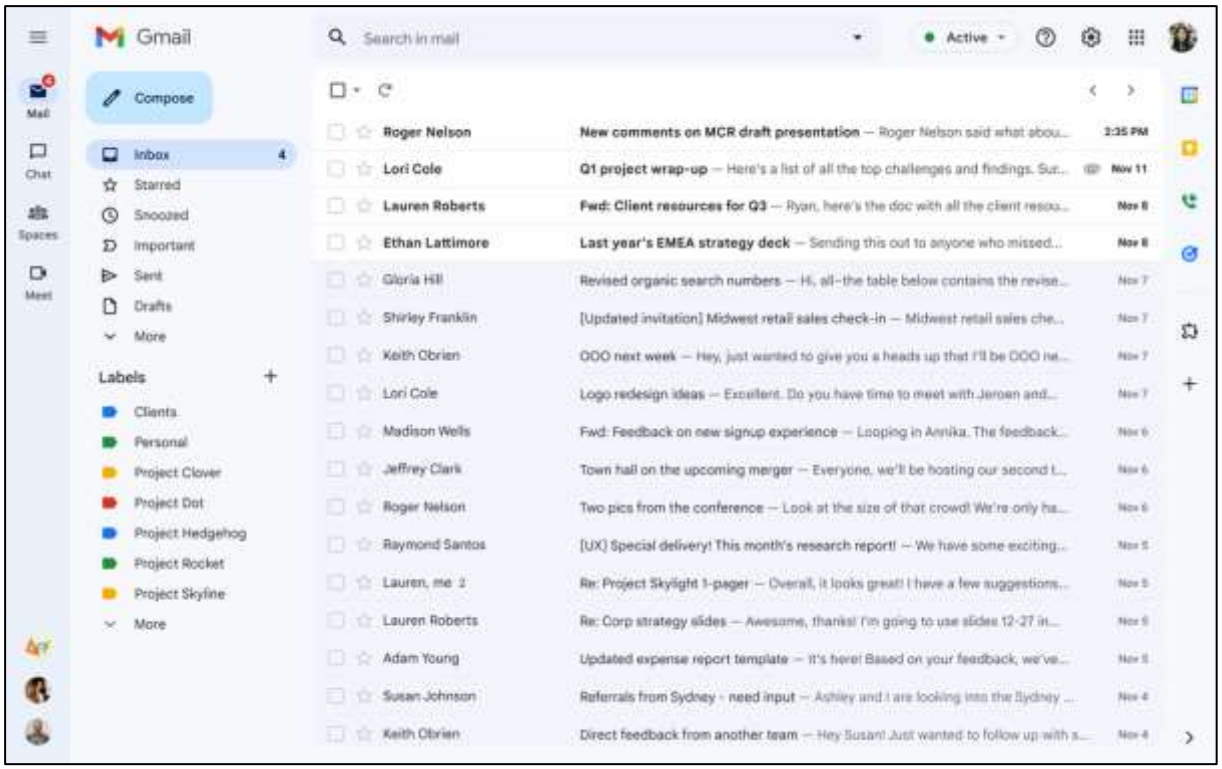


Рис. 1.183. Почтовый сервис Gmail от Google использует возможности Angular

Рис. 1.184. Сервисы Weather.com также построен с использованием Angular

jQuery

Определение

jQuery *– это библиотека на языке JavaScript, которая упрощает управление структурой документа и универсализирует работу с разными браузерами.*

jQuery представляет собой один файл с JavaScript-кодом, который предоставляет пользователям большой объем полезных в работе функций. Изначально библиотека создавалась для повышения гибкости работы с JavaScript и упрощения синтаксиса.

Особое внимание при разработке jQuery было уделено стабильной работе скриптов в разных браузерах и операционных системах (за исключением старых браузеров по типу Internet Explorer 6, которые в настоящее время не актуальны).

jQuery распространяется по открытой лицензии MIT и имеет некоторые дополнительные ветви развития jQuery UI и jQuery Mobile, которые расширяют возможности организации работы со сложными веб-интерфейсами.

Библиотека jQuery упрощает решение следующих задач:

- управление HTML-разметкой на основе модели DOM;
- обработка CSS-селекторов и их стилей;
- работа с событиями;
- интеграция с технологией AJAX (обмен данными с сервера в фоновом режиме);
- добавление анимационных эффектов.

jQuery зачастую позволяет существенно упрощать код и делать его более компактным, однако при реализации простых операций его возможности в целом избыточны.

Это полезно знать!

В настоящее время востребованность jQuery существенно упала, поскольку сложные задачи обычно решают с помощью фреймворков.

JavaScript

```
let menu = document.getElementById("menu");
let menuItems = menu.children;

for (var i = 0; i < menuItems.length; i++) {
    menuItems[i].onclick = function(){
        this.style.background = "yellow";
    };
}
```

JavaScript

```
$(function() {
    $("#menu > li").click(function() {
        this.style.background = "yellow";
    });
});
```

Список

- Пункт 1
- Пункт 2
- Пункт 3

Рис. 1.185. Элементы списка окрашиваются по щелчку мыши: реализация на «чистом» JavaScript и jQuery

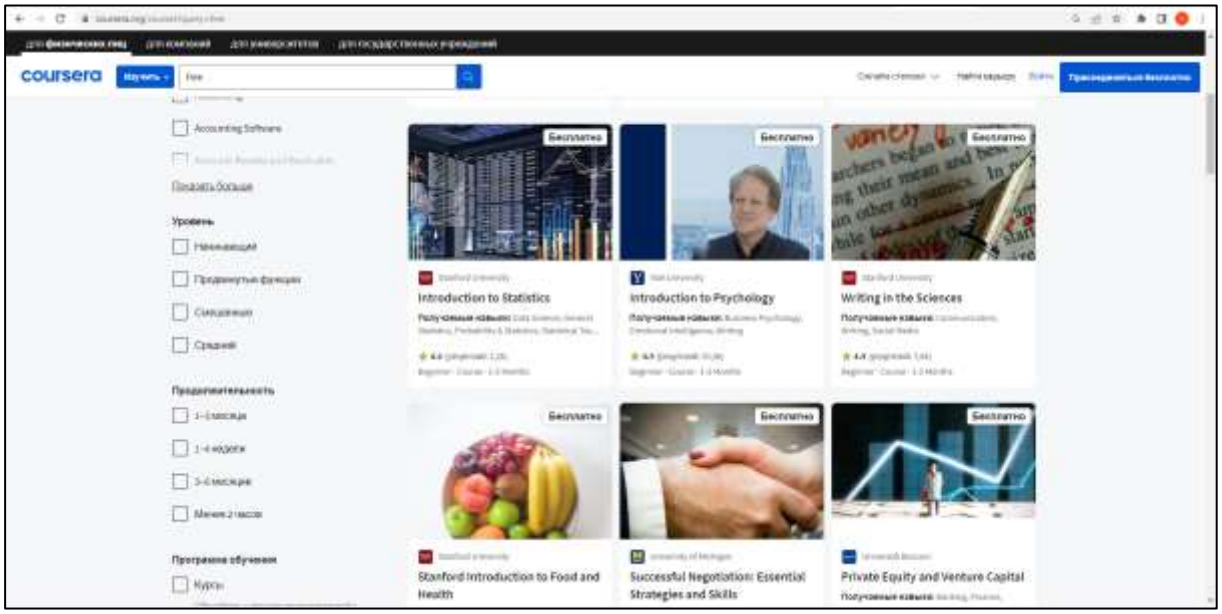


Рис. 1.186. Сервис для размещения электронных учебных курсов Coursera.org разработан с использованием jQuery

1.6.2. Backend-разработка

Языки программирования

PHP

Определение

PHP *– это скриптовый объектно-ориентированный язык общего назначения, используемый в веб-разработке на стороне сервера и клиента.*

PHP остается одним из наиболее востребованных языков веб-разработки. На заре развития веб-технологий язык активно использовался как один из немногих инструментов, который позволял динамически менять структуру разметки веб-страниц. В настоящее время подобные операции обычно осуществляются другими языками программирования и фреймворками, однако PHP до сих пор продолжает использоваться и совершенствоваться.

PHP обладает следующими возможностями:

- динамическая типизация;
- отправка и обработка данных на сервере;
- поддержка встроенных средств работы с различными СУБД, в частности – языка запросов SQL;
- работа с cookies, сессиями, XForms.

Язык PHP упрощает множество рутинных операций кодирования, достаточно прост в изучении и лоялен к ошибкам. Однако разработчики отмечают недостатки в его архитектуре, сложность в масштабировании проектов и непредсказуемость результатов в ряде случаев.

Рис. 1.187. Фрагмент PHP-кода будет обработан на стороне сервера и отобразится в браузере уже в виде тега абзаца

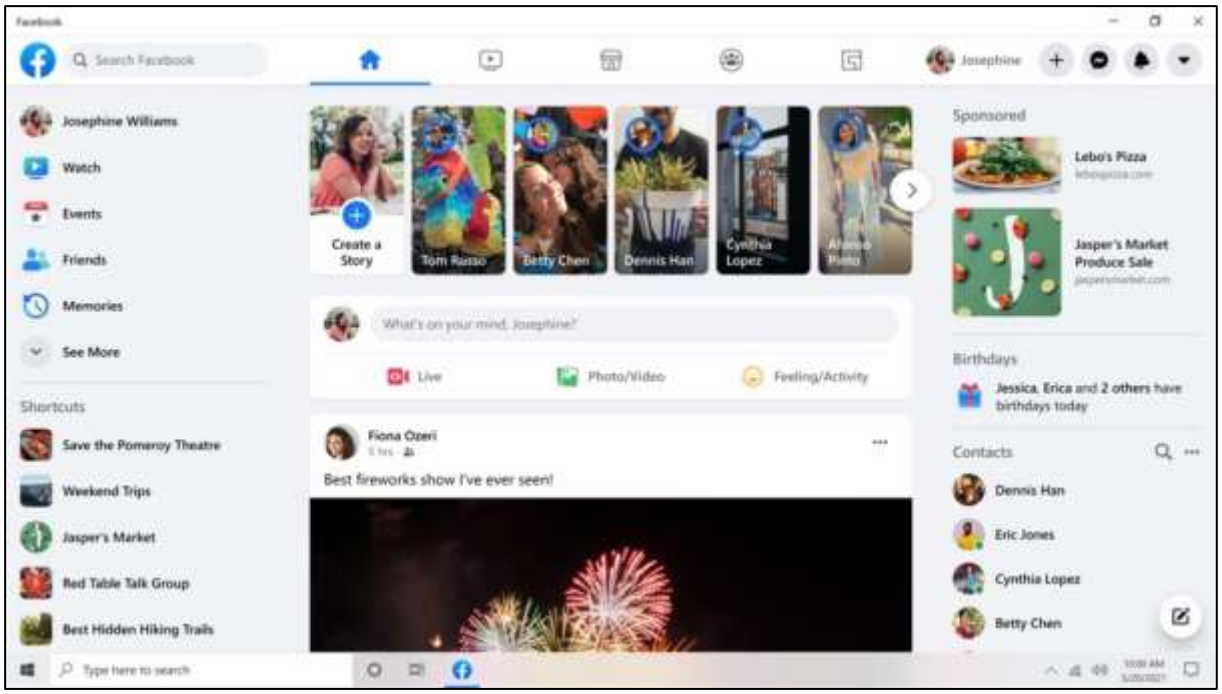


Рис. 1.188. PHP использовался в создании социальной сети Facebook

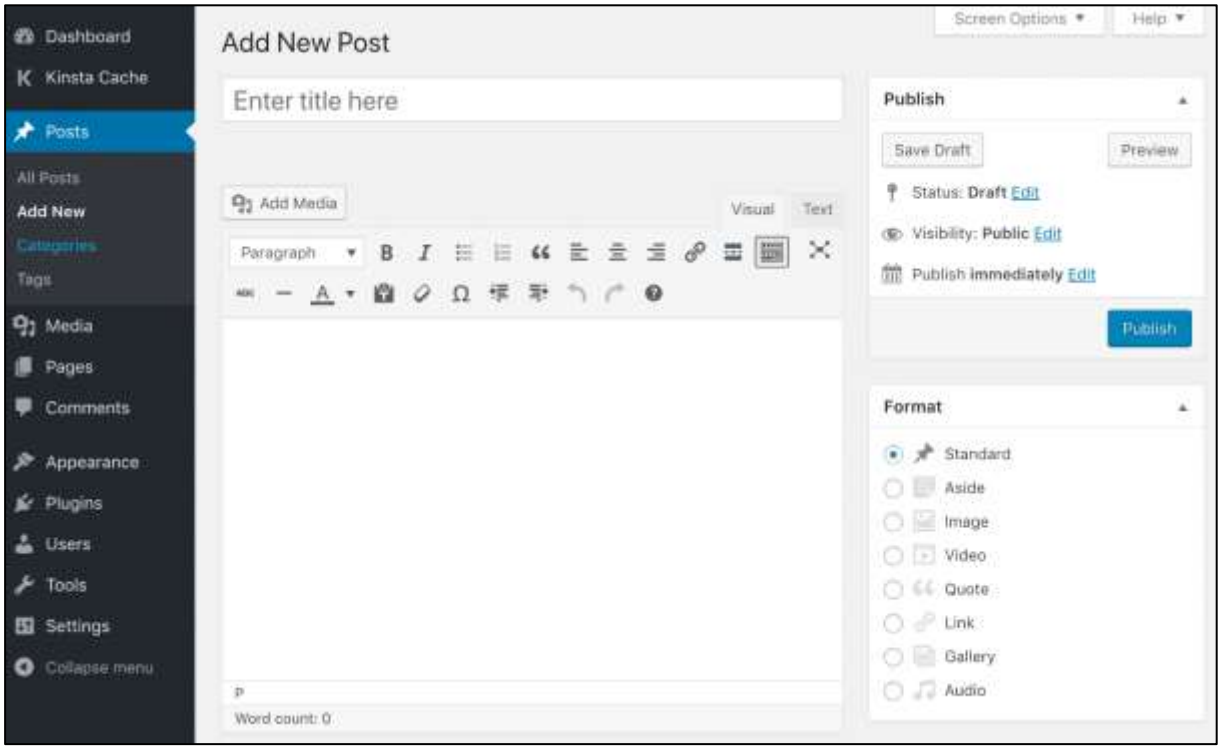


Рис. 1.189. Популярный конструктор сайтов WordPress использует PHP как один из языков, определяющий логику работы сайта

Java

Определение

Java *– это объектно-ориентированный язык программирования со строгой типизацией данных. Используется в разработке веб-приложений.*

Технология языка Java получила высокую популярность не только благодаря удобному синтаксису и встроенным библиотекам, но и его кроссплатформенности. Универсальность Java обеспечивается использованием байт-кода, в который транслируется программа. Java является эффективным языков в реализации проектов, требующих сложного функционала, высоких нагрузок и надежности.

Java активно используется при разработке следующих сервисов:

- приложений для банков;
- социальных сетей;
- маркетплейсов и интернет-магазинов;
- онлайн-кинотеатров и видеохостингов;
- игровой индустрии;
- приложений для Android;
- проекты для крупных компаний.

Особенности Java:

- реализует строгую типизацию данных;
- доступно множество библиотек и фреймворков;
- универсальность выполнения приложений благодаря работе под виртуальной машиной;
- развитие языка поддерживает открытое сообщество.

Язык Java в силу хорошей структуризации кода является неплохим инструментом для изучения программирования. Синтаксис языка очень похож на C# имеет много общих черт с C++.

Среди недостатков Java стоит отметить требовательность к ресурсам ПК.

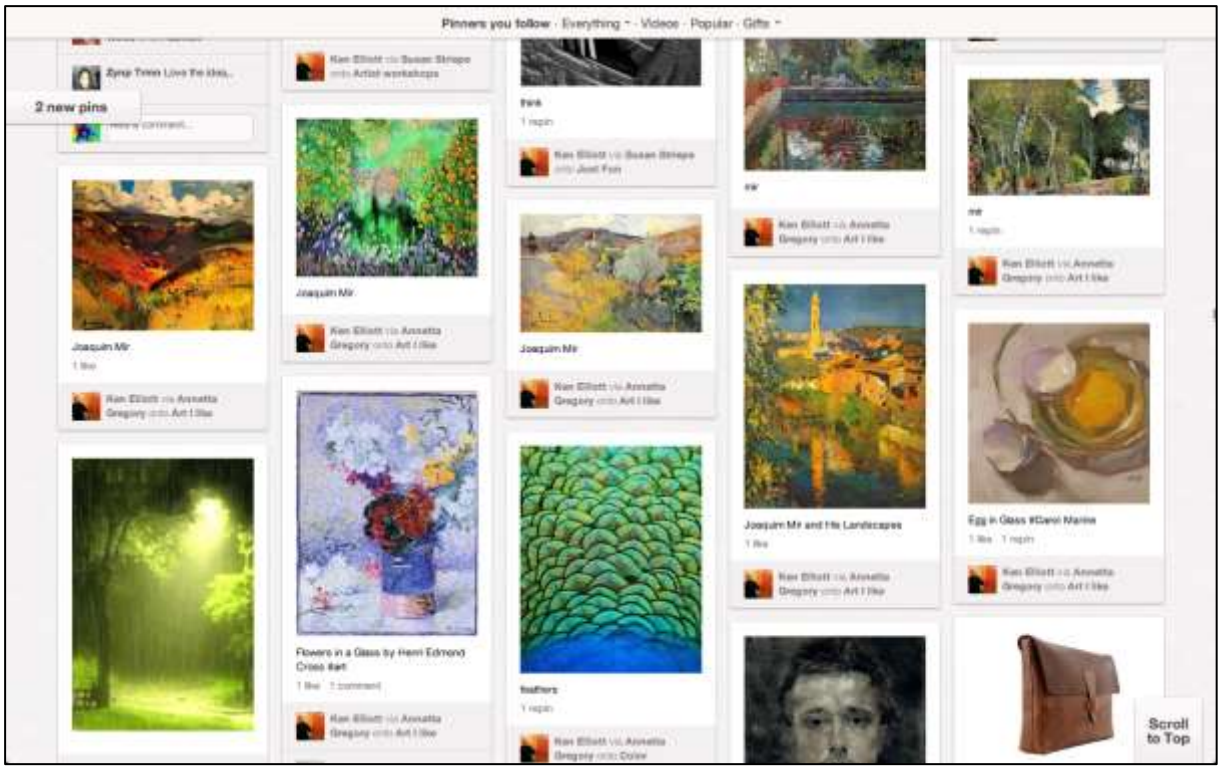


Рис. 1.190. Java использования для создания фотохостинга Pinterest

Рис. 1.191. Одна из популярнейших игр Minecraft разработана на базе Java

Python

Определение

Python *– это высокоуровневый объектно-ориентированный язык общего назначения, имеющий динамическую строгую типизацию данных и механизм автоматического управления оперативной памятью.*

Python стал одним из самым популярных языков программирования благодаря лаконичности синтаксиса. Язык активно используется в организации математических расчётов, программировании нейросетей, систем искусственного интеллекта. Однако сфера его использования очень широка: от разработки настольных приложений до администрирования веб-сервисов.

Python реализует разные парадигмы программирования, обладает прозрачной модульной структурой кода и является одним из лучших языков в обучении программированию.

Основные достоинства языка Python:

- минималистичный и лаконичный синтаксис;
- обширный набор встроенных библиотек и различных фреймворков;
- способность работать с XML/HTML-файлами, HTTP-запросами, веб-сценариями, технологией FTP, приложениями с графическим интерфейсом, изображениями, аудио и видео файлами;
- универсальность и открытость.

Python относится к интерпретируемым языка программирования, поэтому скорость выполнения его скриптов может быть несколько ниже, чем у других языков его класса. Также в Python сложно реализовать асинхронное выполнение задач и многопоточные вычисления.

Тем не менее Python крайне востребован в администрировании серверов.

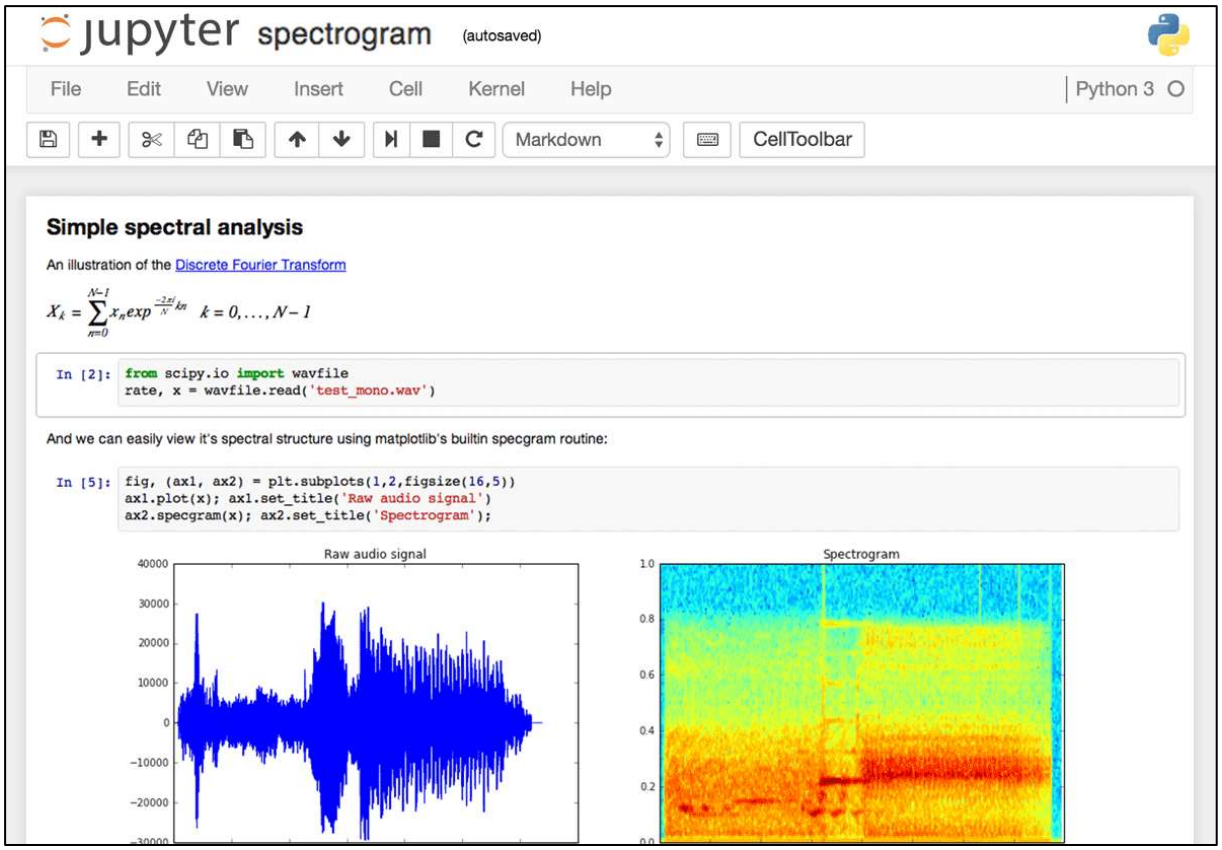


Рис. 1.192. Jupyter – интерактивная веб-платформа для работы с Python

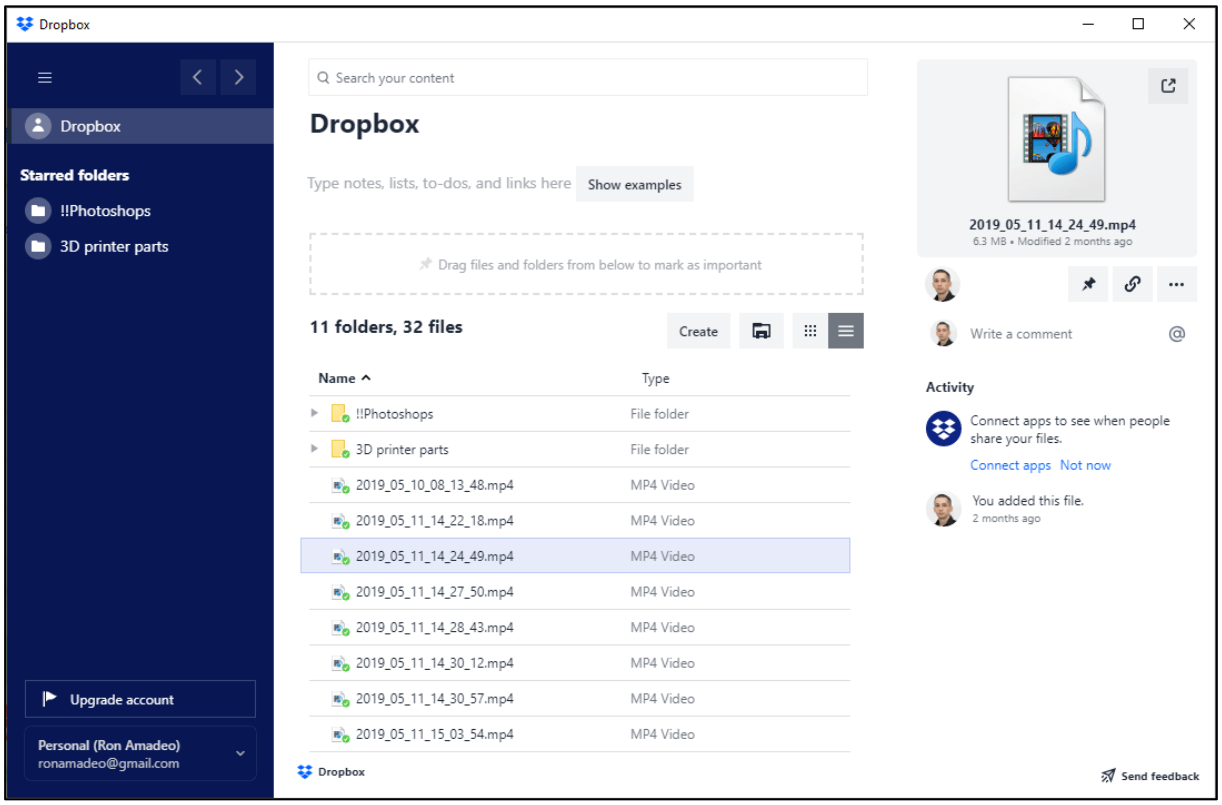


Рис. 1.193. Файловый хостинг DropBox написан с использованием Python

Ruby

Определение

Ruby *— высокоуровневый объектно-ориентированный язык программирования общего назначения.*

Ruby был создан в 1995 году японским программистом Юкихиро Мацумото. Язык получил широкое распространение лишь в 2000-х, когда документация была переведена на другие языки.

Обычно Ruby используют в качестве серверного языка, однако он универсален и может решать множество других задач разработки. Ruby создавался как язык, удобный для разработчика. Философия работы с этим языком предполагает автоматизацию многих рутинных операций. Кроме того Ruby дает возможность программисту выбирать удобный для него способ реализации решения оной и той же задачи. Во главу угла в Ruby ставится прозрачность и изящество решения задачи, если производительность не столь критична.

Ruby популярен в силу ряда причин:

- предсказуемость поведения программы;
- ориентированность на разработчика;
- многофункциональность;
- востребованность;
- активное сообщество.

Перечислим основные особенности языка:

- Ruby – *интерпретируемый* язык, поэтому поддерживается разными платформами.
- Язык является *объектно-ориентированным*.
- Важным элементом Ruby являются *блоки* – это произвольные фрагменты кода, который можно передать в функцию или метод.
- Ruby относится к *динамически типизируемым* языкам программирования.
- Язык также поддерживает *многопоточное выполнение*.

В силу особенностей работы интерпретатора Ruby он не подходит для систем со слабыми характеристиками.

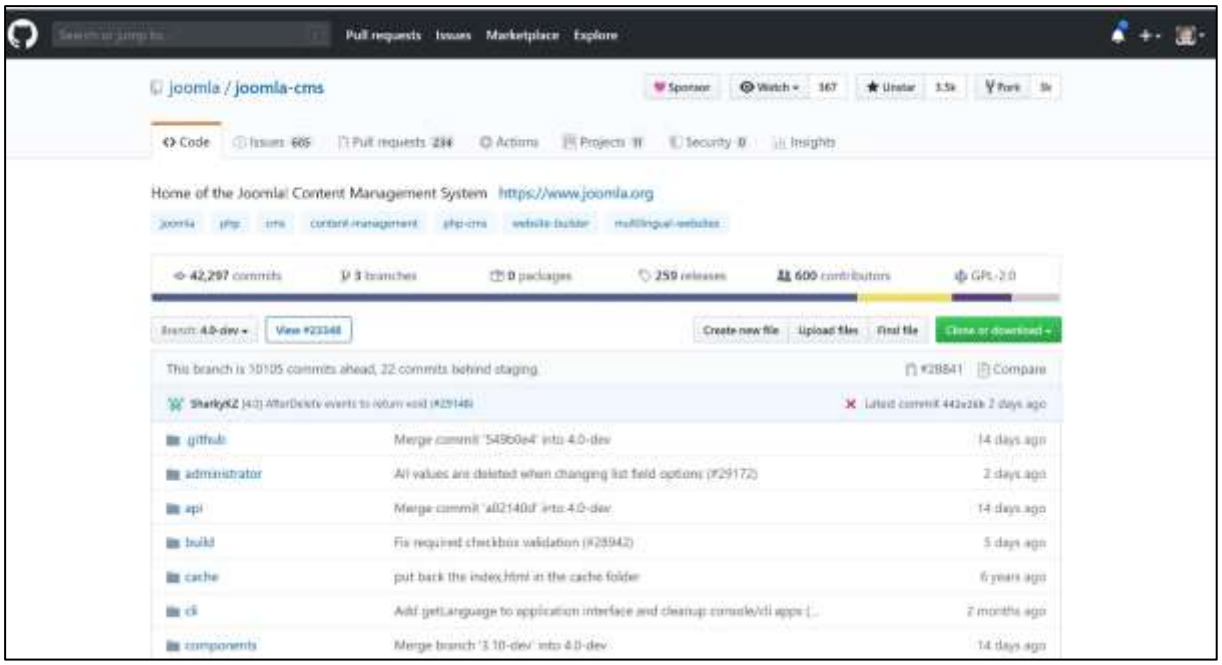


Рис. 1.194. GitHub – крупнейший сервис для хостинга IT-проектов, написан с использованием Ruby

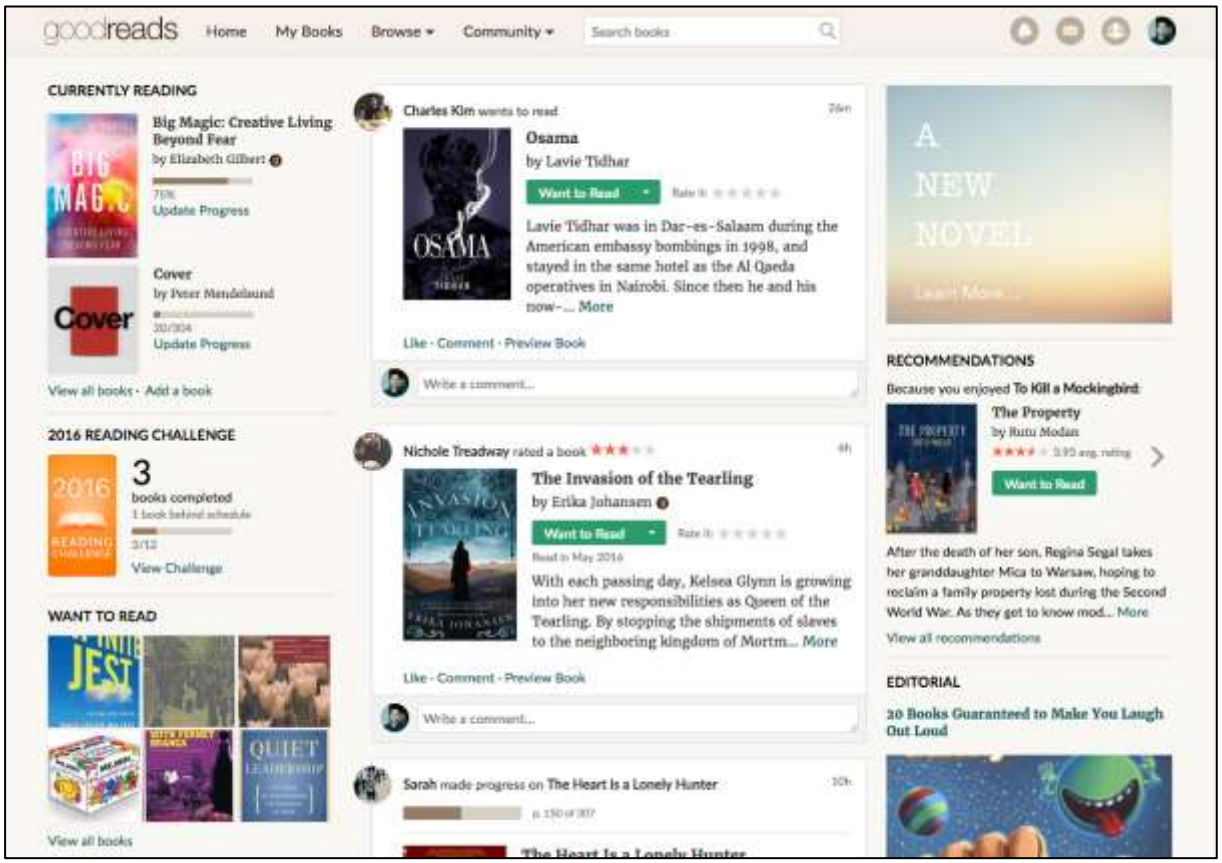


Рис. 1.195. Сервис Goodreads является социальной сетью для читателей книг. Также разработан с использованием Ruby

Golang

Определение

Go *(также* ***Golang****) – это язык программирования общего назначения от компании Google, поддерживающий многопоточные вычисления и объектно-ориентированный подход.*

Язык Golang был представлен компанией Google в 2009 году в 2009 году. Разработчики Go Роб Пайк и Кен Томпсон пытались создать современный язык в качестве альтернативы C и C++, который бы упрощал и ускорял разработчику ПО для Google.

Язык Go заимствовал лучшие практики из языков C и C++, Pascal, Python, Oberon и Modula. Изначально Golang использовался только внутри корпорации Google, однако со временем стал переноситься и в общее использование.

Важные черты языка Go:

- Относится к классу компилируемых.
- Способен организовывать многопоточные вычисления.
- В языке поддерживаются *горутины* — параллельное выполняемые операции, работающие независимо от функции, в которой они запущены.
- Работает с многопоточностью и позволяет создавать микросервисы.
- Является статически строго типизированным языком.
- Работает с кодировкой Unicode.
- Обладает простым синтаксисом.
- Имеет встроенный механизм *сборки мусора*.
- Кроссплатформеный и переносимый, поддерживается на системах Windows, Linux, macOS, Android. Также может работать с UNIX-системами FreeBSD, OpenBSD и другими.

Однако в плане реализации объектно-ориентированного подхода язык не работает с классами и наследованием. Это необходимо для упрощения кода и повышения его быстродействия.

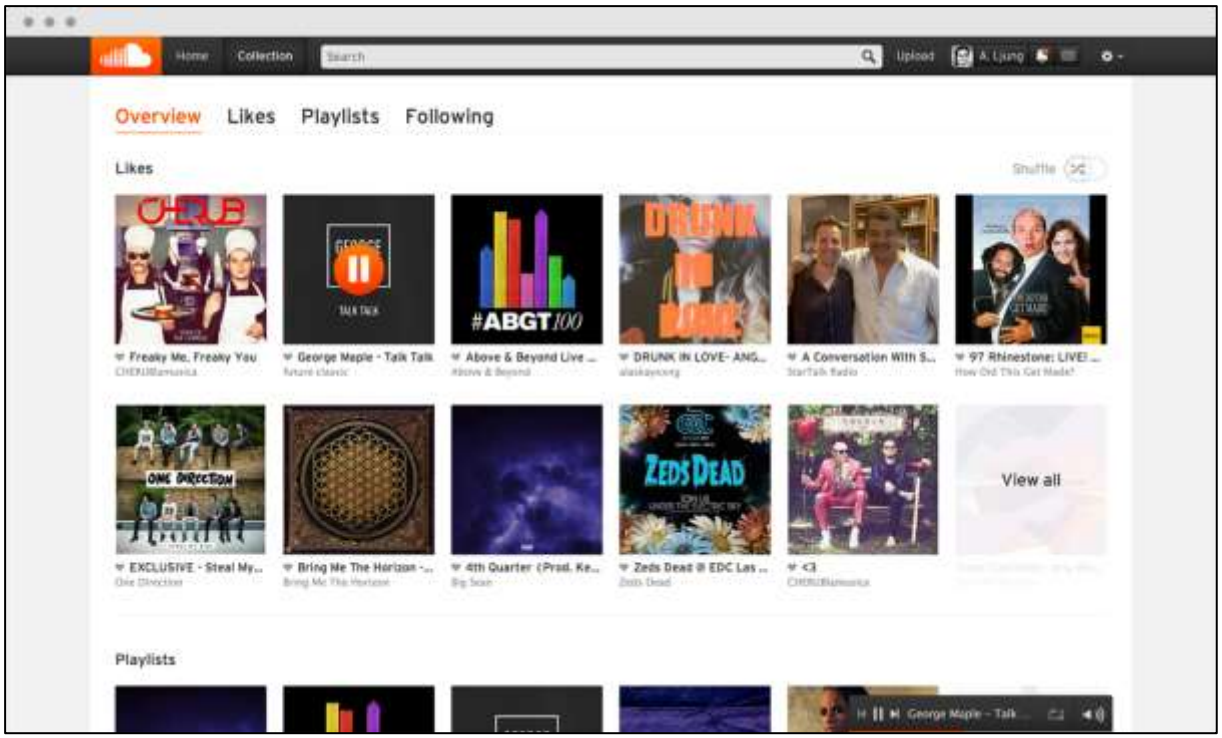


Рис. 1.196. SoundCloud – одна из крупнейших платформ для распространения звукозаписей, написана с использованием Go

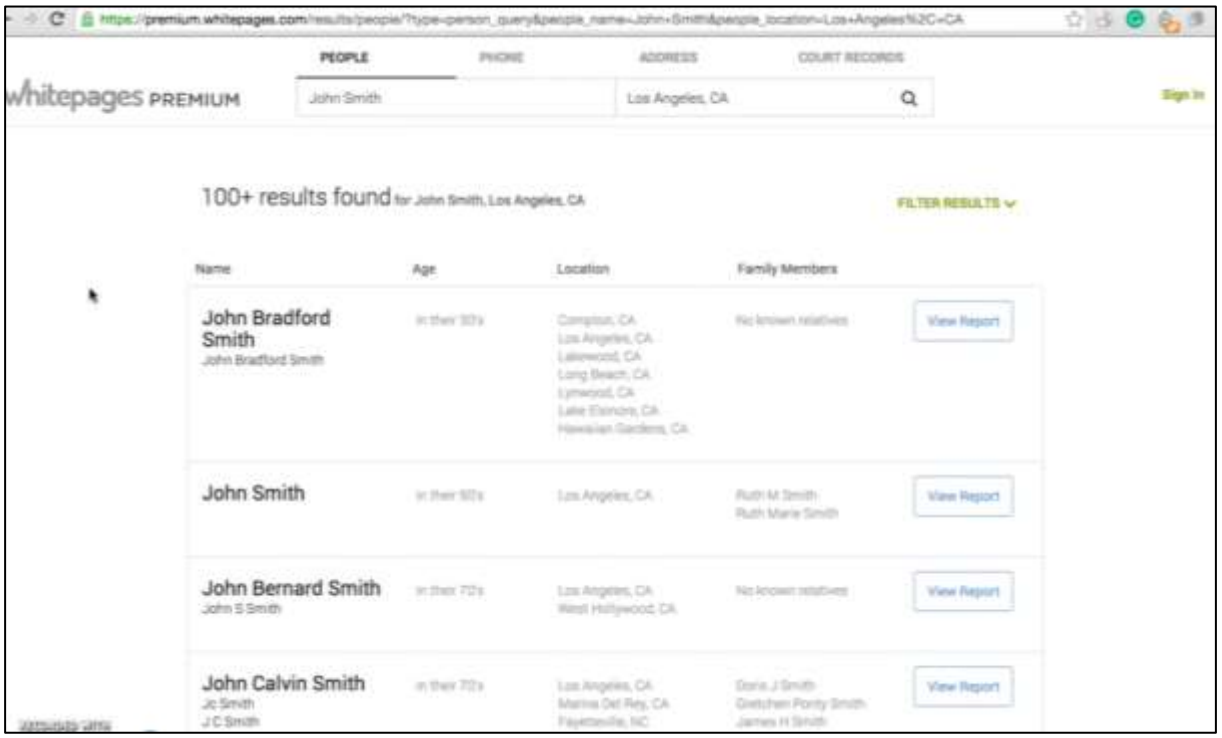


Рис. 1.197. Сервис Whitepages для проверки биографических данных личности

Perl

Определение

Perl *– интерпретируемый язык высокого уровня, предназначенный для кодирования задач общего назначения.*

Perl был разработан американским программистом Ларри Уоллом в 1987 году, который был также специалистом в области лингвистики. В 1988 году в язык добавлена поддержка регулярных выражений, а в 1989 – обработка данных в двоичной форме.

Современные версии Perl поддерживают работу со сложными типами данных и объектной моделью.

Язык Perl является эффективным инструментом в решении задач администрирования UNIX- и Windows-систем. Perl реализует большой объем задач самостоятельно и не требует привлечения для этих целей других средств. Язык помогает управлять компьютерами и системами с помощью инструкций командной строки, контролировать выполнение программ и осуществлять иные операции администрирования систем.

Отметим основные преимущества Perl:

- встроенные механизмы обработки сложных структур;
- многогранный синтаксис, т.е. возможность решать одну задачу несколькими способами;
- обширный набор встроенных библиотек (модулей);
- гибкие возможности в обработке строк, в частности - поддержка регулярных выражений;
- простая обработка больших объемов данных;
- поддержка объектно-ориентированного и функционального стилей программирования;
- кроссплатформенность.

Несмотря на общую направленность Perl, обычно он используется в решении специфических задач. В ряде случае отмечается, что многочисленные модули языка избыточны при создании простых проектов.

Perl используется в электронной почте, веб-узлах, выполнении CGU-сценариев (обработка введенных на HTML-странице данных).

Рис. 1.198. DuckDuckGo – анонимная поисковая система. Написана с использованием Perl

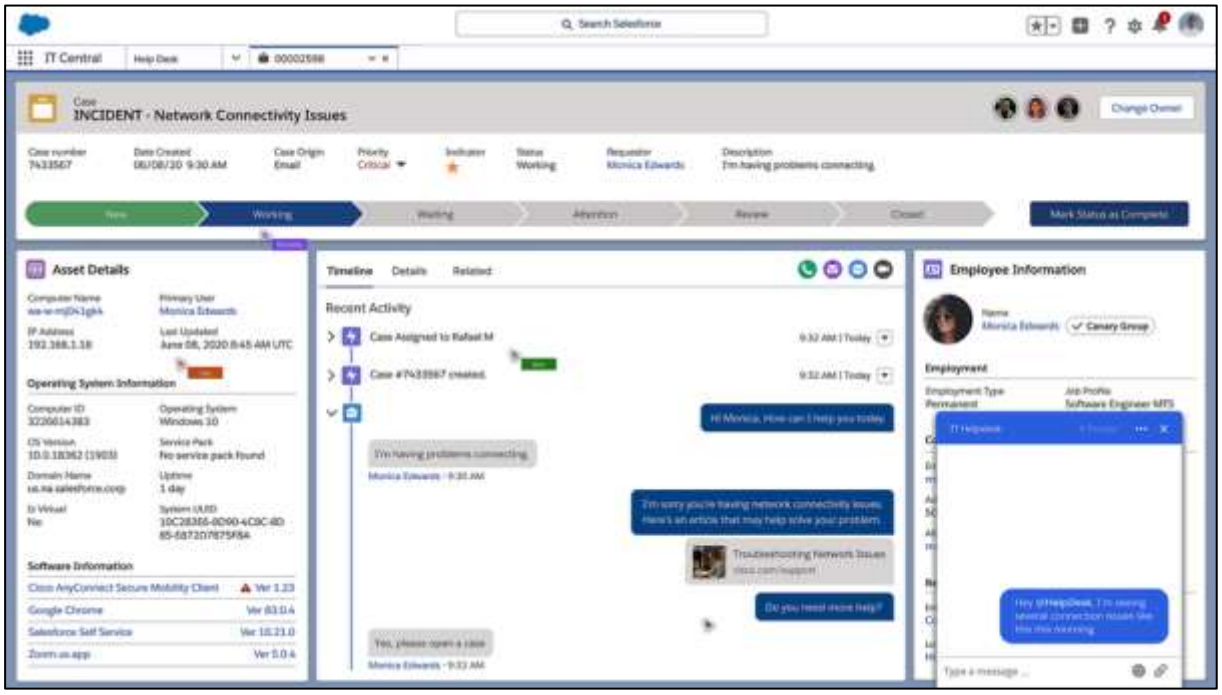


Рис. 1.199. Tanium – компания, занимающаяся вопросами кибербезопасности и системным управлением

SQL

Определение

SQL *представляет собой структурированный язык запросов, используемый для взаимодействия с базами данных.*

SQL не является языком программирования для разработки ПО. Он предоставляет формализованный синтаксис, позволяющий описывать запросы в виде команд, параметров и ограничений.

SQL уже долгое время остается одним из наиболее востребованных языков. Его используют аналитики, маркетологи, менеджеры, разработчики и тестировщики.

Язык SQL тесно связан с понятием реляционных баз данных, которые выстраиваются по принципу оптимизации операций на основе работы с ключами, которые позволяют исключить избыточность и дублирование данных при работе со связанными таблицами.

Запросы SQL представляют собой логически выстроенные последовательности из ключевых слов, параметров и операторов. В отличие от многих других языков, SQL является декларативным, т.е. близким к естественному человеческому языку описанию проблемы, а не алгоритма ее решения.

ID	FullName	City	...	...
34	Nicolson L. H.	Madrid	...	...
23	Edison E. A.	Dresden	...	...
...	...	...	...	...

```
select * from Customers
where City NOT IN ('Madrid', 'Berlin', 'Bern')
```

Рис. 1.200. Запрос отбирает из таблицы Customers все записи (строки), для которых значение в колонке City не совпадает с указанными. Иными словами, отбираются данные о заказчиках, которые не находятся в Мадриде, Берлине и Берне

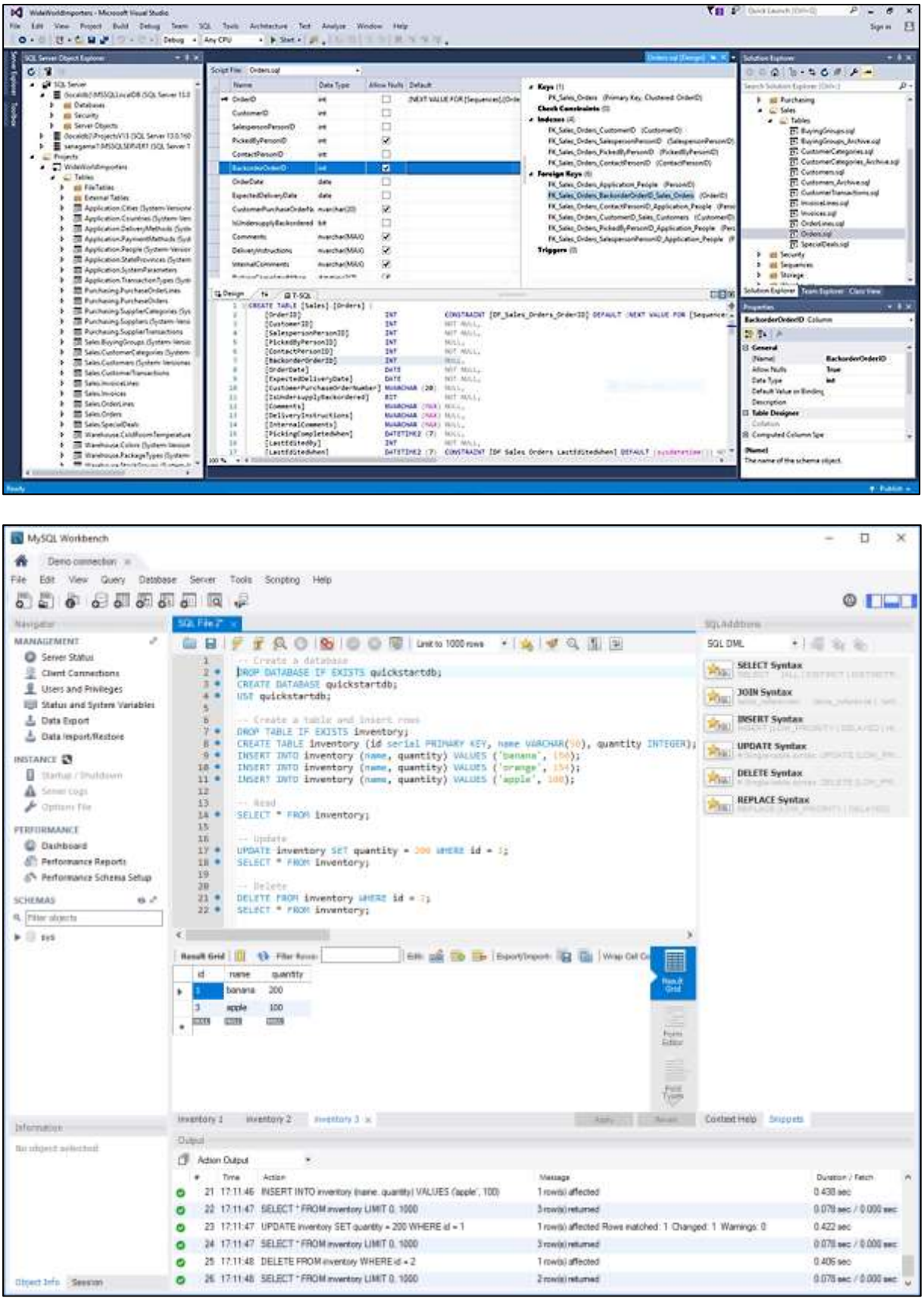


Рис. 1.201. MS SQL Server и MySQL – системы управления реляционными СУБД

Фреймворки и платформы

Django

Определение

Django *– свободный распространяемы фреймворк на языке Python, предназначенный для разработки быстрых и безопасных веб-приложений и сайтов.*

Django был создан в 2003-2005 годах Адрианом Головати и Саймоном Уиллисон. Работа фреймворка базируется на шаблоне проектирования MVC.

Как и другие фреймворки, Django выполняет роль программного каркаса, который берется за основу разработчиком расширяется сценариями, графическим интерфейсом, библиотеками, базами данных и другими модулями.

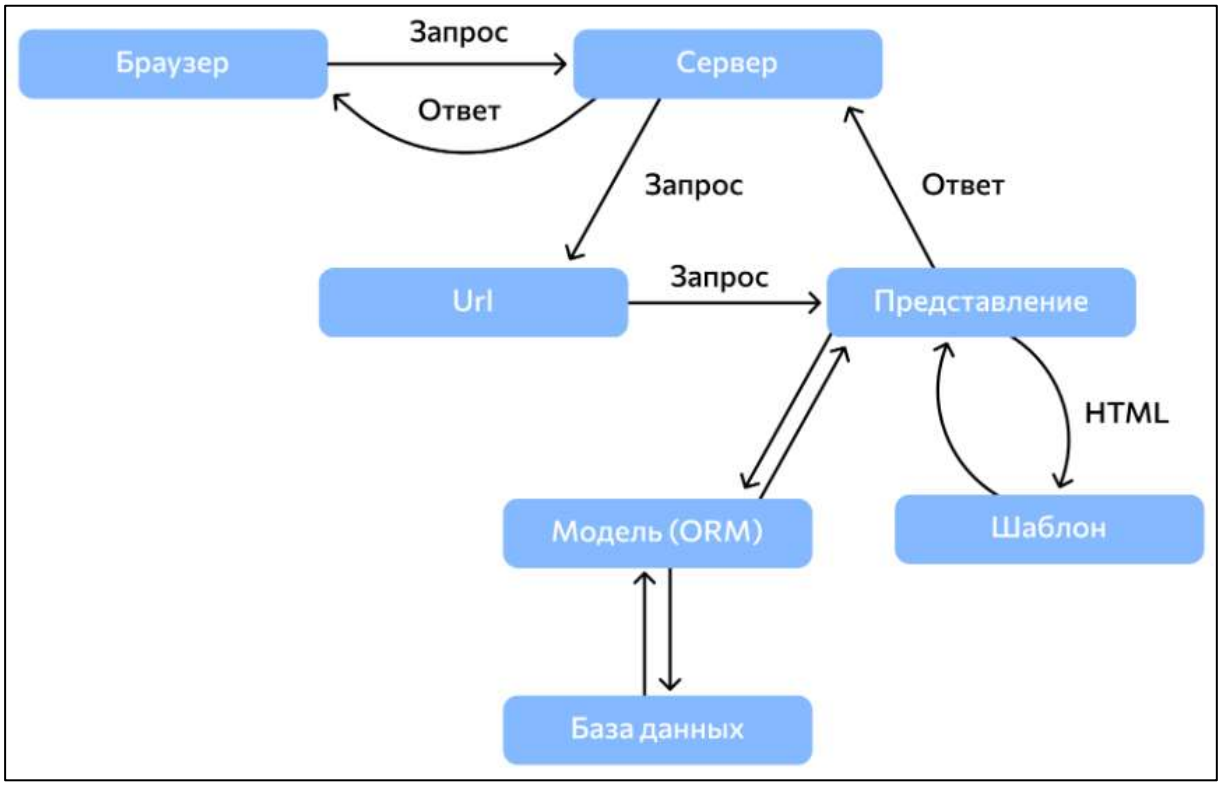


Рис. 1.202. Схема работы приложения, построенного на базе Django

В структуру Django включены следующие компоненты:

- URL-маршрутизаторы, перенаправляющие HTTP-запрос веб-клиента в форму представления;
- представление, обрабатывающее запрос и запрашивающее у модели данные из базы данных;
- модель (менеджер базы данных, ORM), извлекающую требуемые данные из базы данных и передающая их представлению;
- HTML-шаблоны, которые формируются представлением для отображения пользователю.

Кроме того, Django содержит следующие модули и функции:

- веб-сервер, необходимый в разработке и тестировании проекта;
- собственный диспетчер, обеспечивающий обмен событиями между компонентами приложения с помощью установленных сигналов;
- система интернационализации по языковым особенностям;
- встроенные модули, обеспечивающие аутентификацию, авторизацию, подключение внешних блоков;
- фильтры обработки запросов (операций сжатия, кеширования, перенаправления данных и т.д.);
- интерфейс администратора для управления содержимым приложения или сайта;
- интерфейс платформы для тестирования кода на Python;
- модуль защиты.

Фреймворк Django имеет следующие особенности:

- Полнота реализации функциональных модулей позволяет использовать Django в реализации большого числа типовых задач.
- Универсальность платформы Django: подходит для разработки веб-сайтов и приложений разных типов: новостных агрегаторов, видеохостингов, социальных сетей и другое.
- Надежность Django, которая обеспечивается сообществом разработчиков.
- Возможность эффективно масштабировать проекты.
- Хорошая сопровождаемость кода.

- Обеспечение защиты данных в рабочих модулях.
- Переносимость приложений на разные платформы: Windows, Mac OS X и Linux.
- Открытость исходного кода платформы.
- Обширная экосистема и возможность подключения дополнительных модулей.
- Высокая гибкость, обеспечивающая вариативность в выборе приемлемых решений одной задачи.

Однако Django обладает рядом недостатков:

- Монолитная архитектура, которая требующая синхронного взаимодействия разных технологий. Это осложняет обновление платформы.
- Работает с устаревшим ORM.
- Платформа не поддерживает многозадачность.
- Шаблон маршрутизации URL требует владения приемами работы с регулярными выражениями.

Фреймворк Django является эффективным инструментом разработки проектов в следующих случаях:

- требуется обеспечить стабильную работу сервиса и надежную защиту данных;
- проект планируется расширять в будущем, подключая дополнительный ресурс баз данных;
- необходима поддержка ORM;
- в проекте участвуют несколько разработчиков;
- планируется задействовать инновационные технологии (искусственный интеллект, нейросети);
- нет достаточных компетенций в разработке веб-приложений или веб-сайтов «с нуля»;
- установлены жесткие ограничения на время развертывания и бюджет;

В случае реализации простых проектов, либо наоборот – проектов с разветвленной или специфический структурой следует обратиться к другим фреймворкам.

Рис. 1.203. Панель администратора в Django

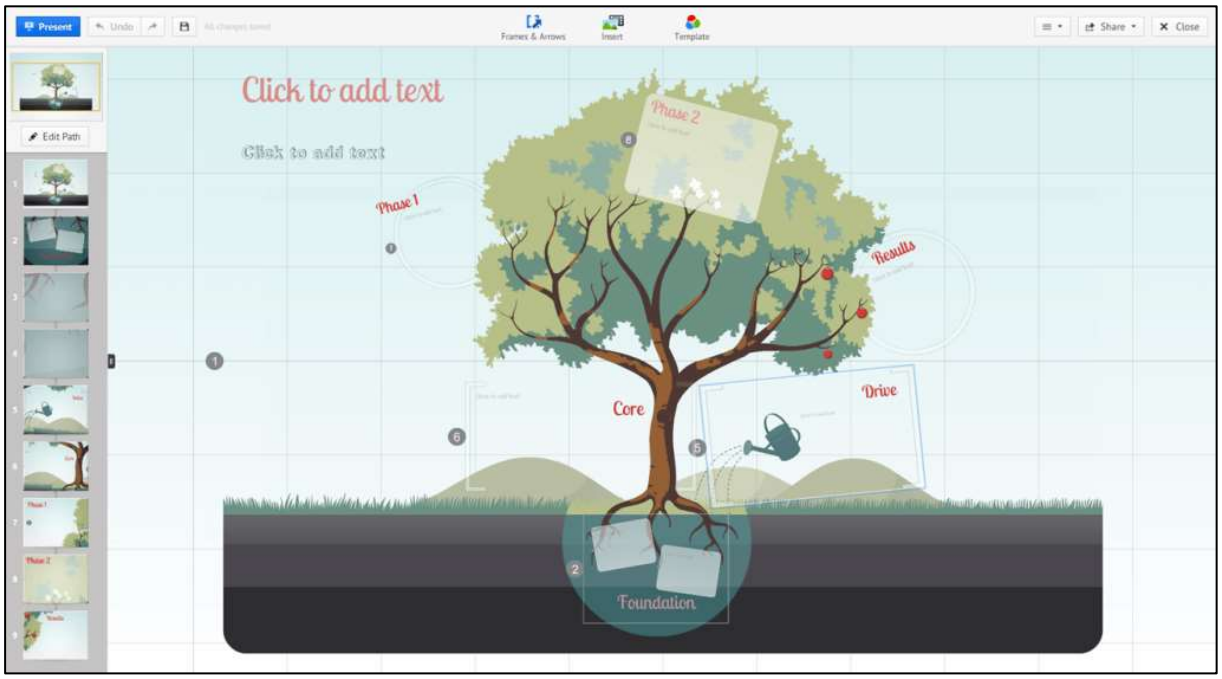


Рис. 1.204. Популярный веб-сервис для создания презентаций создан с использованием Django

ASP.NET

Определение

ASP.NET *– это свободно распространяемая платформа компании Microsoft на базе .NET Framework, нацеленная на разработку динамических сайтов и веб-приложений.*

APS.NET появилась в 1997 году и стала развитием технологии ASP. Первая версия платформы была запущена одновременно с другой популярной платформой .NET Framework и позволяла разрабатывать веб-приложения и сайты на разных языках программирования, которые поддерживались в .NET. ASP.NET легла в основу многочисленных проектов, среди которых и сайт компании Microsoft.

Кроме того, существует модификация ASP.NET Core, которая имеет несколько иную архитектуру и является уже фреймворком с открытым программным кодом. В этой платформе акцент сделан на модульность и производительность.

Появление платформы .NET во многом было связано со стремлением разработчиков Microsoft создать аналог Java. Платформа .NET использует *общеязыковую среду исполнения CLR*, которая позволяет разработчику выбрать более удобный для него язык (из стека доступных). Среди наиболее известных и востребованных – C#.

CLR делает приложения независимыми от особенностей выбранного языка программирования, поскольку компиляция осуществляется в два этапа:

- в начале высокоуровневый код компилируется промежуточный универсальный язык IL (в некотором смысле высокоуровневый ассемблер);
- далее сборка на IL компилируется в низкоуровневые инструкции, которые выполняются непосредственно перед стартом приложения.

Отметим некоторые особенности компиляции в ASP.NET. Трансляция веб-приложения в промежуточный IL-код необходима только один раз. Если в код были внесены изменения, то осуществляется повторная компиляция. При этом машинный код кэшируется в системном каталоге.

Подходы к компиляции в IDE и при выполнении в браузере также отличаются. В частности, Visual Studio преобразует IL-код относительно всего проекта. А код веб-сайта компилируется уже постранично. При этом в обоих случаях преобразование IL-кода в машинные инструкции происходит при первом выполнении.

ASP.NET использует модель объектно-ориентированного программирования. Разработчик имеет доступ к единому набору классов .NET Framework, что обеспечивает:

- четкую архитектуру, возможность наследования и расширения классов;
- стандартизацию кода на базе интерфейсного подхода.

В ASP.NET качественно реализован принцип инкапсуляции данных для серверных компонентов. Разработчику нет необходимости писать HTML-код: вместо это работа ведется с более высокоуровневыми абстракциями объектов и структур данных.

Фреймворк ASP.NET предлагает веб-разработчикам три автономных инструмента, которые имеют собственные подходы к программированию.

1. *ASP.NET Web Forms* позволяет конструировать веб-сайты в визуальном конструкторе путем перетаскивания элементов и настройки событий.
2. *ASP.NET MVC* использует работу с шаблонами и необходима для создания динамических веб-сайтов, особенно мобильных и одностраничных приложений.
3. *ASP.NET WEB API* является фреймворком для работы с HTTP-службами для разных браузеров и мобильных устройств. Этот фреймворк удобен в случае создания легко масштабируемых приложений на базе .NET Framework.

Укажем важные преимущества технологии ASP.NET:

- хорошие показатели производительности;
- эффективный контроль потенциальных ошибок;
- поддержка нескольких языков;
- поддержка в Windows, MacOS и Linux;
- многофункциональность кода и библиотек.

В качестве недостатков APS.NET отметим зависимость платформы от политики Microsoft и ухудшение производительности при реализации небольших проектов.

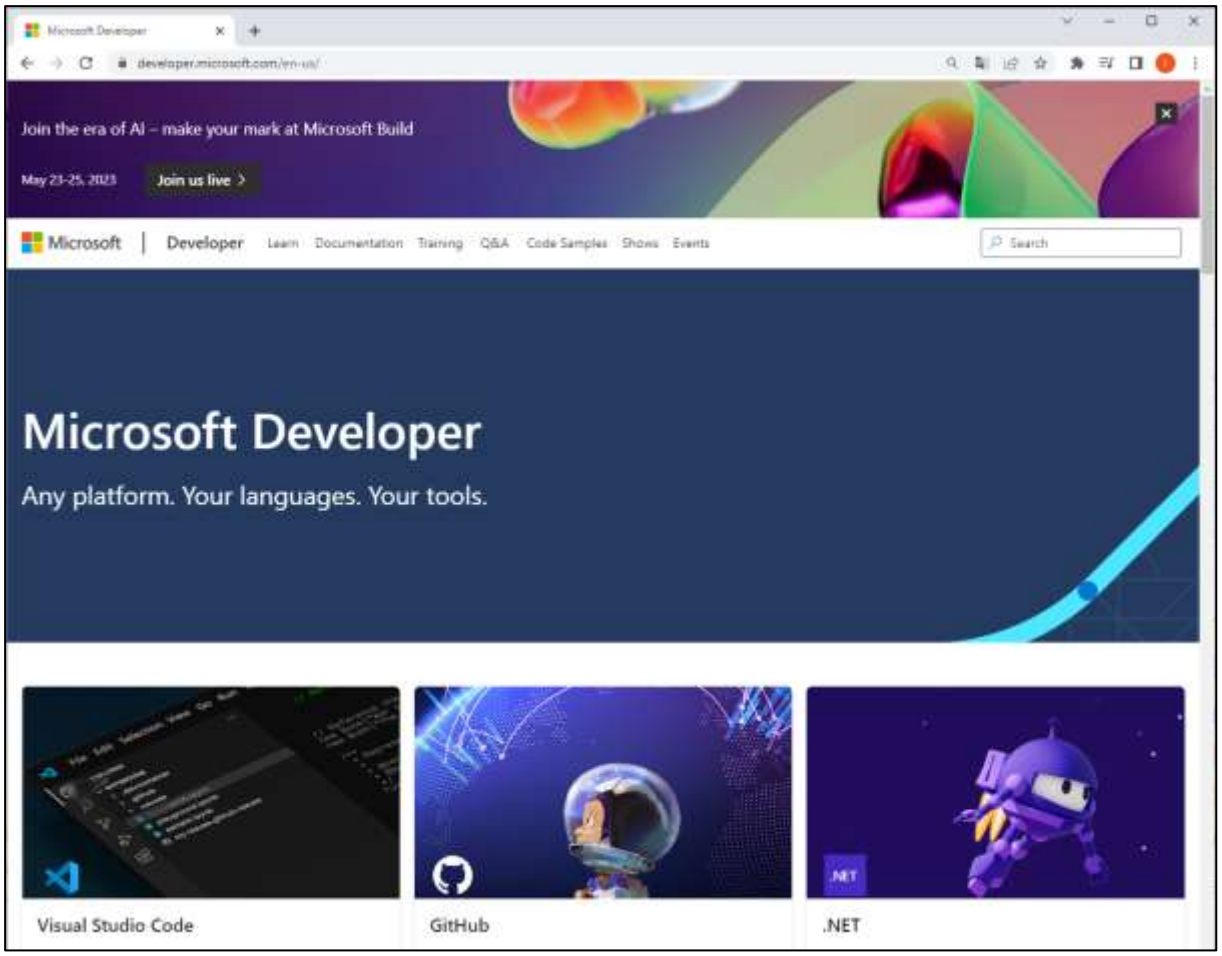


Рис. 1.205. Сайт Microsoft реализован на базе ASP.NET

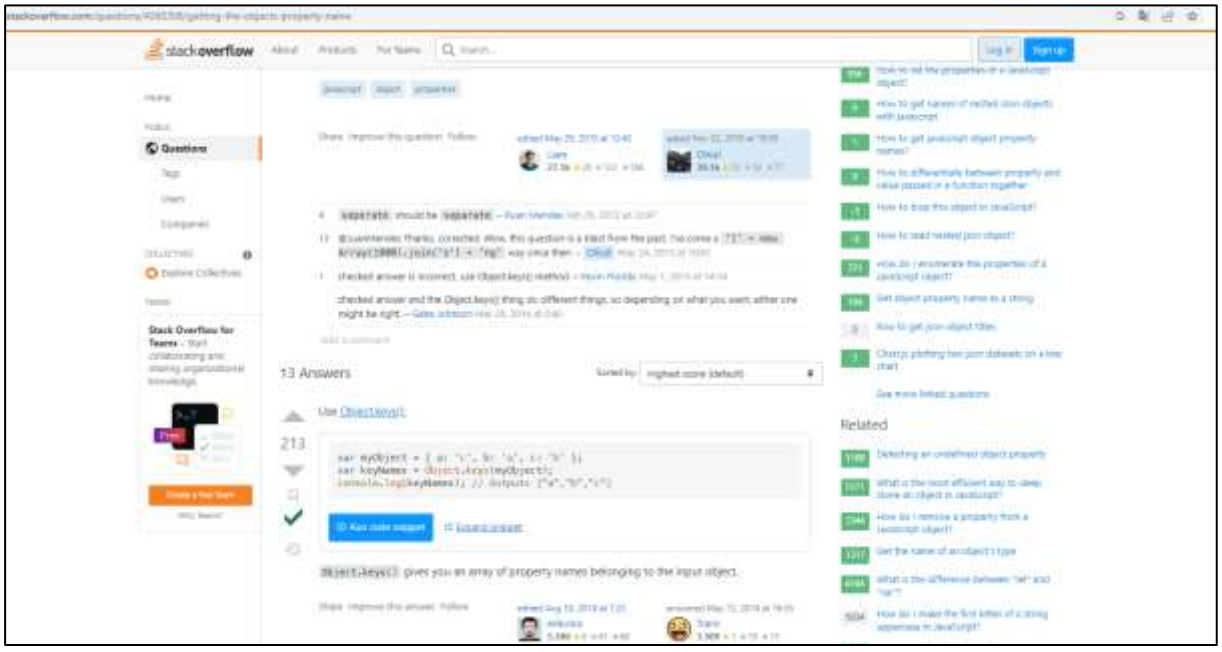


Рис. 1.206. Stack Overflow – крупнейший сервис вопросов и ответов по разнообразным технологиям IT. Построен с использованием ASP.NET

Ruby on Rails

Определение

Ruby on Rails *– это объектно-ориентированный фреймворк на базе языка программирования Ruby, имеющий открытый программный код.*

В Ruby on Rails используется уже обозначенный ранее принцип MVC, а идея работы состоит в организации связи моделей с базой данных и их взаимодействие с объектами приложений. Общий механизм работы предполагает отправку сервером запросов к пользователю, используя контроллер. Сформированные данные извлекаются из модели и возвращаются пользователю в форме файлов представления (HTML-разметок).

Для соблюдения единого подхода к разработке в Ruby on Rails следуют принципам RESTful-роутинга. Для этого функции контроллера имеют названия ActionController, ActiveRecord и ActionView, где:

- ActionController анализирует данные и готовит ответ;
- ActiveRecord отвечает за предоставление данных и бизнес-логики;
- ActionView компилирует отклик;

Первая версия Ruby on Rails была запущена Дэвидом Ханссоном в 2004 году и в 2005 году она стала общедоступной. Ханссон обозначил ряд принципов, которые определили доктрину Ruby on Rails:

1. Оптимизация ради комфортного программирования.
2. Соглашения вместо конфигурации.
3. Выбор по принципу Омакасе (от японского «полагаюсь на вас»).
4. Парадигма не принадлежит кому-либо.
5. Элегантный код первостепенен.
6. Интегрированные системы крайне важны и необходимы.

7. Возможность жертвовать стабильностью работы ради прогресса.
8. Расширение сообщества Ruby-разработчиков только приветствуется.

Ханссон большое внимание всегда уделял роли единомышленников, которые активно участвовали в развитии проектов, продвигали новые идеи и внедряли их в работу.

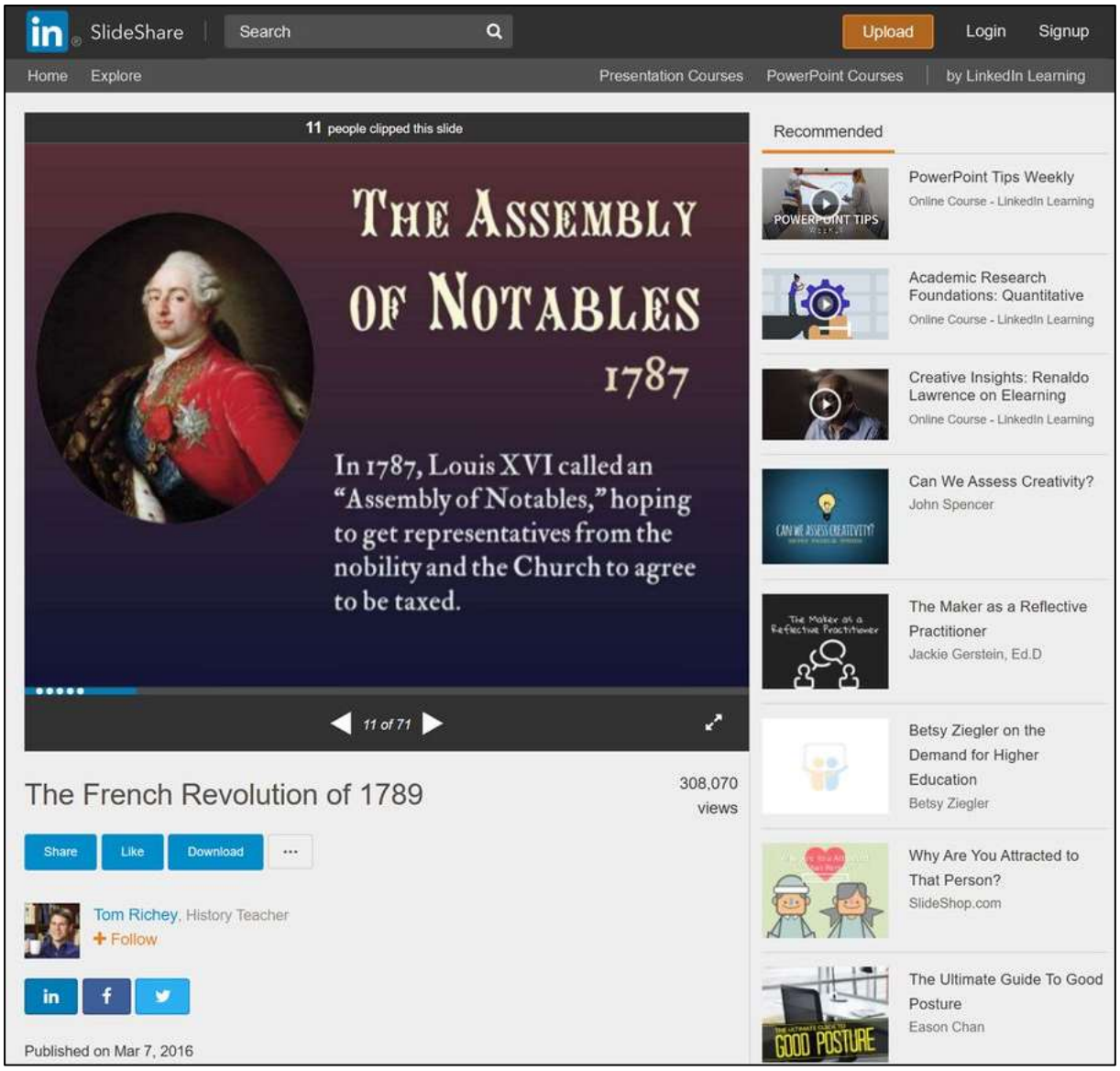


Рис. 1.207. SlideShare – крупный сервис для публикации презентаций реализован с использованием фреймворка Ruby on Rails

Laravel

Определение

Laravel *– это PHP-фреймворк с открытым программным кодом, который был разработан для помощи в реализации сложных сайтов и веб-приложений.*

Фреймворк Laravel содержит готовые инструменты для организации процесса аутентификации, маршрутизации, сессий, кэширования данных, построения архитектуры приложения и взаимодействия с базой данных.

Laravel построен на модели архитектуры MVC.

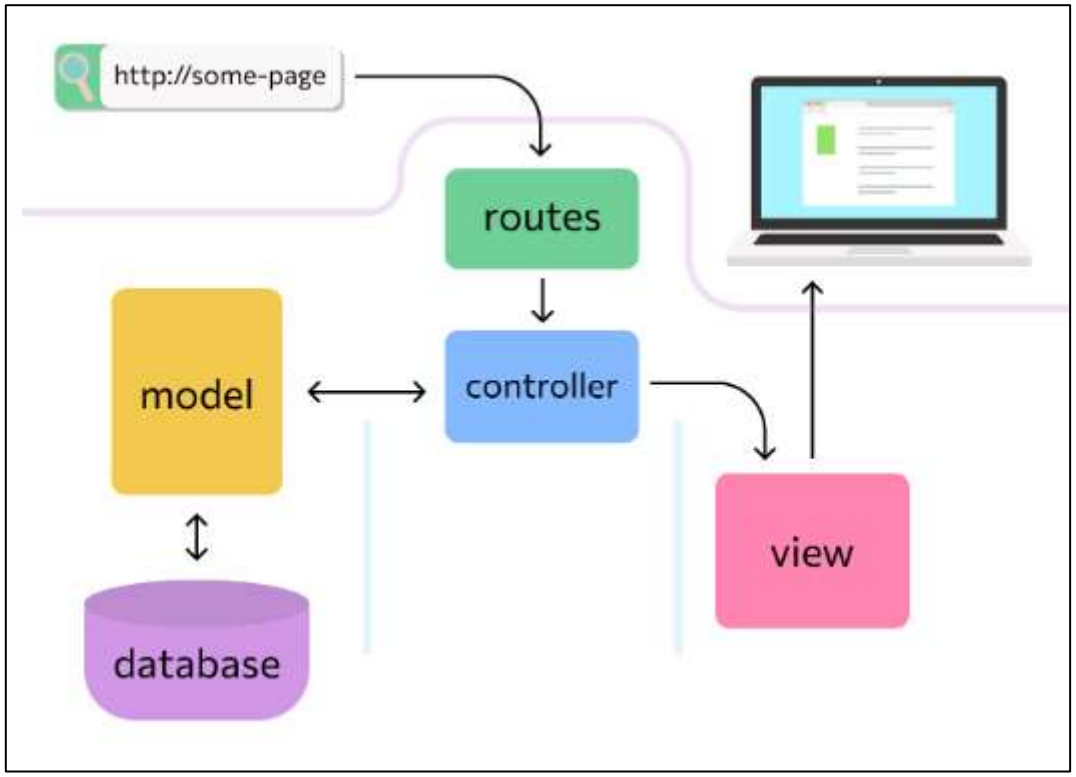


Рис. 1.208. Модель MVC в контексте Laravel

Фреймворк Laravel предоставляет разработчикам комплекс возможностей:

- Консоль Artisan для работы с командной строкой. Она необходима для генерации моделей, тестов, контроллеров и получения логов.

- Eloquent ORM позволяет эффективно связывать PHP с базами данных и языком программирования. Каждой таблице ставится в соответствие класс, с которым удобно работать. Также Eloquent ORM имеет хорошую защиту от нежелательных операций.
- Конструктор Fluent позволяет быстро создавать запросы к базе данных; он полностью совместим с ядром Eloquent ORM.
- Шаблонизатор Blade преобразует HTML-шаблоны в готовые страницы.
- Валидация данных.
- Работа с системой контроля версий баз данных.
- Проведение модульного тестирования. Laravel имеет собственную систему тестирования PHPUnit.
- Безопасная аутентификация через социальные сети.

Перечислим характерные особенности Laravel:

- развитая экосистема, которая включает в себя активное сообщество, цифровые материалы, программное обеспечение и международные конференции;
- высокие показатели производительности благодаря поддержке noSQL и настраиваемому кэшированию;
- собственные механизмы обеспечения безопасности;
- удобный для чтения и анализа синтаксис;
- гибкие механизмы маршрутизации;
- нетрудоемкость организации миграции;
- поддержка разных языков;
- многочисленные пакеты и библиотеки.

Несмотря обилие возможностей, Laravel требует от программиста большой объем работы. С одной стороны это позволяет гибко настроить возможности создаваемого ПО, а с другой – замедляет процесс разработки. Кроме того, Laravel сложнее ближайших конкурентов в плане изучения.

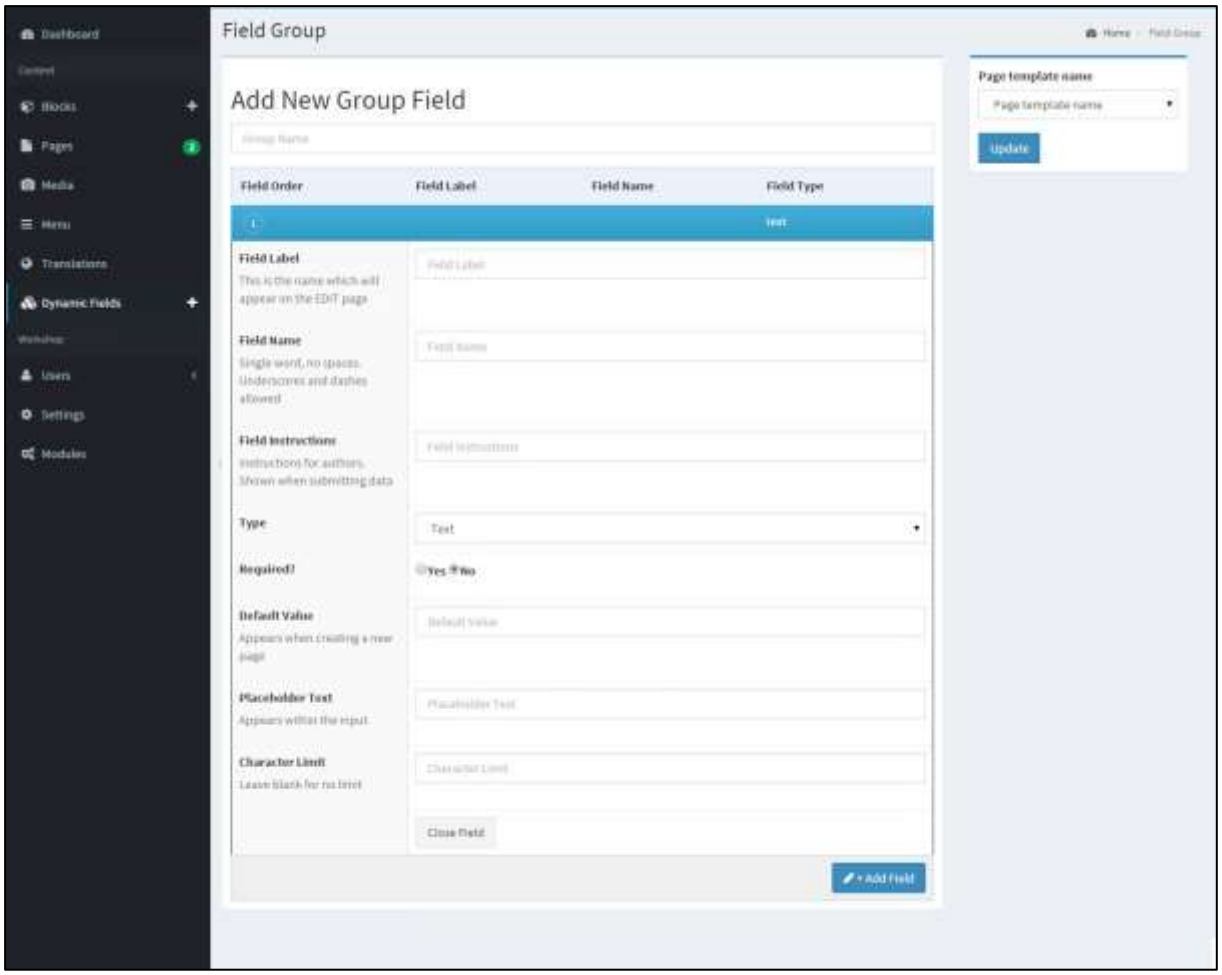


Рис. 1.209. Asgard CMS разработана с помощью Laravel

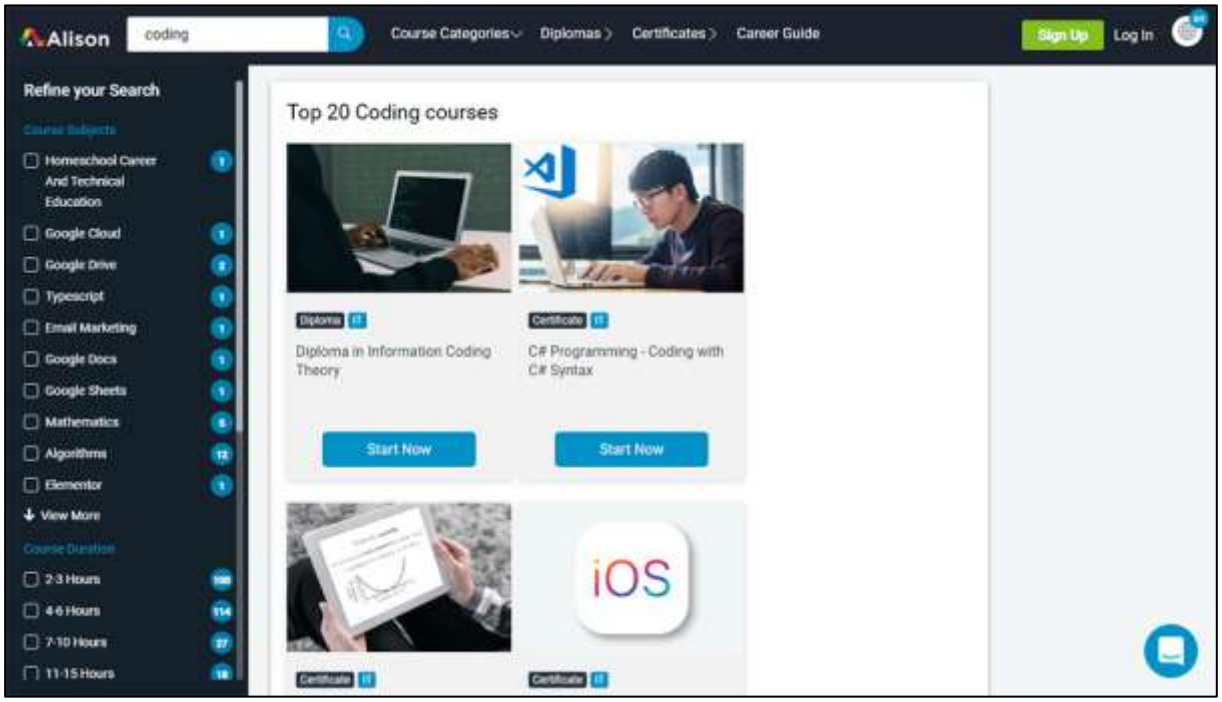


Рис. 1.210. Alison – платформа для электронного обучения

Spring Framework

Определение

Spring Framework *– это фреймворк с открытым программным кодом для разработчиков на языке Java.*

Обычно Spring Framework (далее Spring) используют в реализации крупных корпоративных проектов. Он позволяет разработчикам получить свободу в проектировании и реализации ПО.

Фреймворк Spring написан на языках Java, Kotlin и Groovy. Это мощный фреймворк для Java-разработчиков, который позволяет создавать настольные, мобильные и веб-приложения. Spring обеспечивает условия для быстрой и гибкой разработки, построения зависимостей, а также решения задач, связанных с обеспечением связи между компонентами разных приложений.

Spring имеет модульную структуру и состоит из мини-фреймворков. Отметим некоторые наиболее важные модули:

- IoC является контейнером, который отвечает за управление зависимостями и конфигурацию компонентов.
- Модуль аспектно-ориентированного программирования (АОП), который обеспечивает сквозную функциональность, т.е. такую функциональность, которую нельзя выделить в отдельные сущности средствами объектно-ориентированного программирования (ООП).
- Модуль доступа к данным взаимодействует с СУБД по стандарту JDBC, оптимизированному под Java. Кроме JDBC, Spring поддерживает и ORM.
- Модуль транзакций, который объединяет запросы к СУБД в единые блоки (*транзакции*). Spring может работать с разными видами транзакций и обеспечивать их безопасность.
- Модуль MVC, разделяющий архитектуру на модель данных, отображение и контроллер. Spring имеет собственную реализацию MVC с четким разбиением между слоями.
- Модуль авторизации и аутентификации.
- Другие дополнительные модули и возможности.
- Дочерние фреймворки (Spring Roo, Spring Integration).

Фреймворк Spring обладает целым рядом преимуществ:

- комплексное использование и универсальность;
- упрощение и ускорение работы;
- широкий спектр возможностей;
- большое и активное сообщество;
- открытость фреймворка.

Тем не менее от разработчика Spring потребует немало времени на конфигурирование проектов (может быть упрощено с помощью Spring Boot) и в целом на освоение особенностей программирования под эту платфрму.

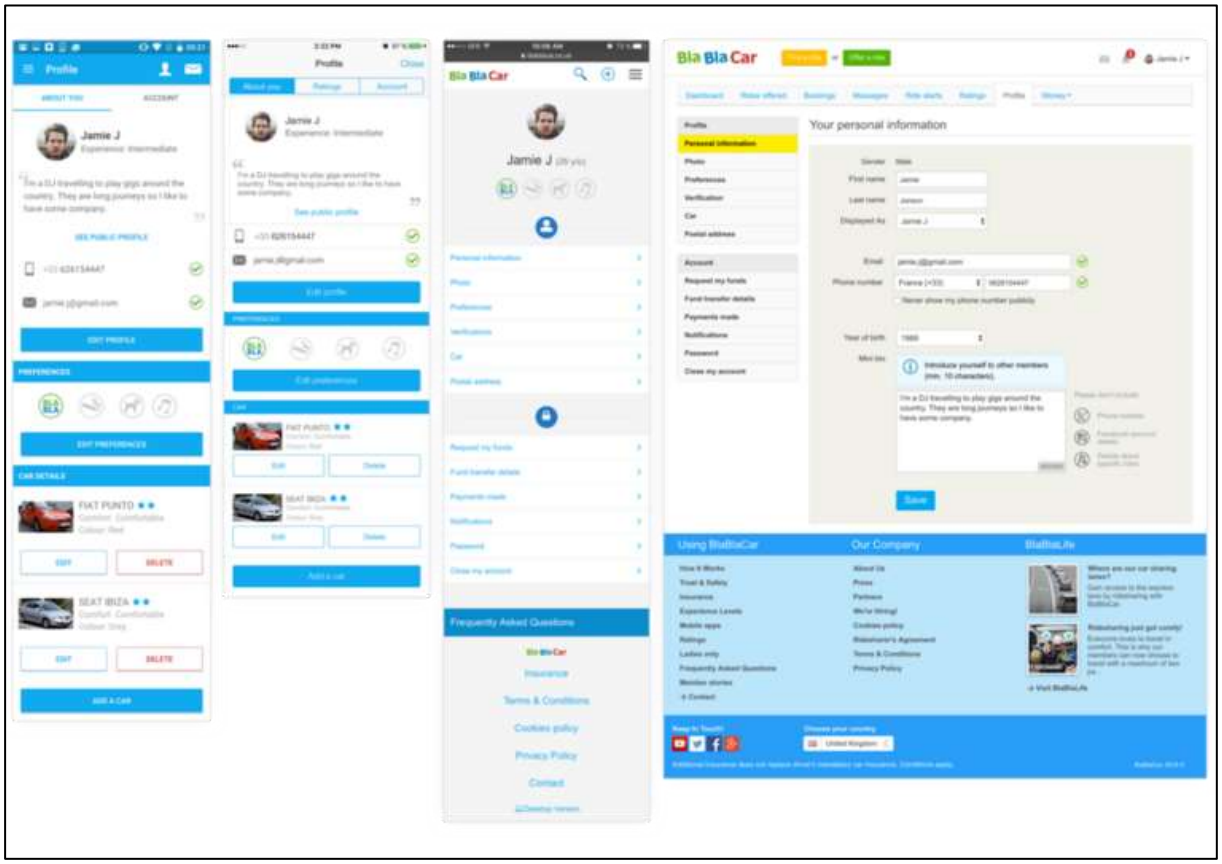


Рис. 1.211. BlaBlaCar – популярный сервис для поиска автомобильных попутчиков, реализовано с использованием фреймворка Spring

Вопросы для самопроверки

1. Почему HTML часто также называют языком разметки структуры документа?
2. Какую роль играют теги в разметке документа?
3. Каким образом технология CSS помогла изменить подход к верстке веб-страниц?
4. Перечислите характерные особенности языка JavaScript.
5. Какие ограничения имеет JavaScript?
6. В чем преимущество TypeScript по сравнению с JavaScript?
7. Какие проекты разрабатываются на языке программирования Swift?
8. Чем интересная библиотека React и какую в ней роль играют компоненты?
9. Опишите возможности фреймворка Vue.
10. Что представляет собой фреймворк Angular.
11. Почему библиотека jQuery в настоящее время теряет свою актуальность?
12. Перечислите основные преимущества языка PHP по сравнению с другими объектно-ориентированными языками backend-разработки.
13. С чем связана высокая популярность языка Java в веб-разработке?
14. Опишите основные компоненты архитектуры MVC. С чем связаны причины ее использования?
15. Какие возможности дает язык Python веб-разработчику?
16. Приведите примеры веб-сервисов, разработанных с использованием возможностей Ruby.
17. Укажите основные сферы использования языка Go.
18. Для решения каких задач язык Perl наиболее эффективен?
19. Перечислите основные особенности фреймворка Django.
20. Опишите особенности платформы .NET и фреймворка ASP.NET.
21. Перечислите компоненты фреймворка Laravel.
22. Укажите достоинства фреймворка Spring.

1.7. Особенности макетов и техники верстки веб-страниц

1.7.1. Классификация макетов сайта

Выбор макета

Разработка интерфейса сайта начинается с создания его макета. Макет веб-сайта определяет способ организации информации на странице. В современном веб-дизайне выделяют множество различных типов макетов, которые можно систематизировать по разным критериям, например, по ширине, количеству колонок, способу компоновки элементов, устройствам и т.д.

Наиболее важными критериями при построении макета являются допустимые значения ширины контента и количество колонок.

- Параметр ширины необходимо контролировать, чтобы содержимое не расползалось на широкоформатных мониторах и не сжималось в узкую полосу на старых классических.
- Верстка колонками обеспечивает простой подход к компоновке блоков на странице. Колонки также являются одним из важнейших компонентов адаптивной разметки.

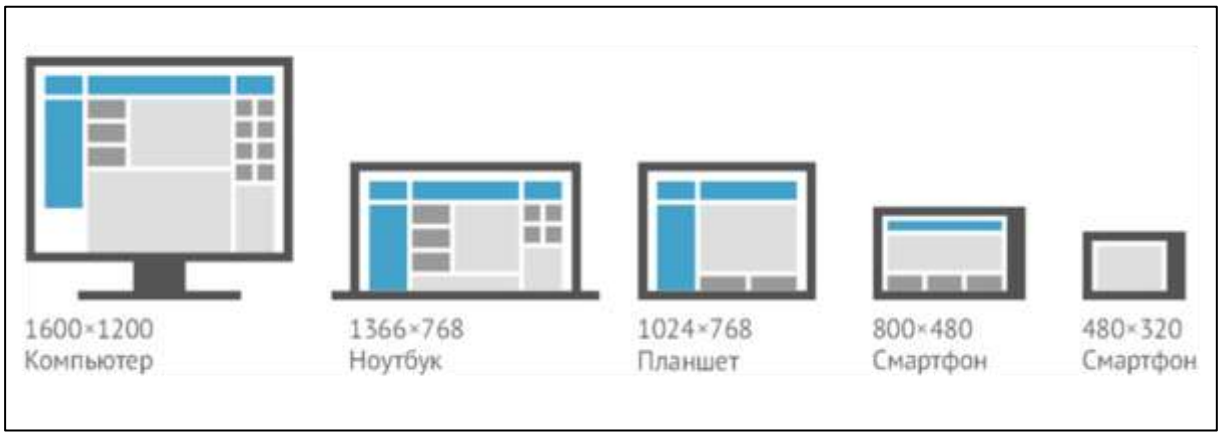


Рис. 1.212. Для разных устройств одинаковый макет может быть неприемлем

1. Фиксированный макет

Для макета с *фиксированной шириной* (или статичного макета) основная область контента задается постоянной и не зависит от ширины экрана устройства потенциального пользователя.

Обычно эта величина задается равной 720, 760, 960, 1200 пикселей, в зависимости от ширины монитора. Выбор этих чисел связан с тем, что их величины близки к наиболее распространённым значениям ширины современных мониторов и в большинстве случаев не потребуют работы с полосой горизонтальной прокрутки. С другой стороны, указанные числа делятся нацело на 2, 3, 4, 5, 6, 8, 10, 12, что позволяет организовать простую разметку в указанное число колонок.

Особенности фиксированного макета

- Основной блок с содержимым обычно выравнивается по центру, его ширина постоянная.
- Ширина контента не меняется в зависимости от размеров окна браузера или устройства.

Достоинства фиксированного макета

- Размеры объектов заранее известны, что удобно для быстрого построения макета и верстки сайта.
- Оптимальны для сайтов, содержащих страницы с большим объемом текста.
- Не вызывают проблем с браузерами.
- Практически одинаково отображаются на разных экранах.

Недостатки фиксированного макета

- На широкоформатных экранах сайт смотрится плохо: много свободного пространства остается слева и справа от основного блока (в случае выравнивания по центру).
- На небольших экранах появляется горизонтальная полоса прокрутки
- В целом неэффективно используется свободное пространство экрана.

В настоящее время макеты с фиксированной шириной используются все реже. Обычно такой подход можно найти на старых сайтах, которые не были адаптированы под новые спецификации HTML и CSS.

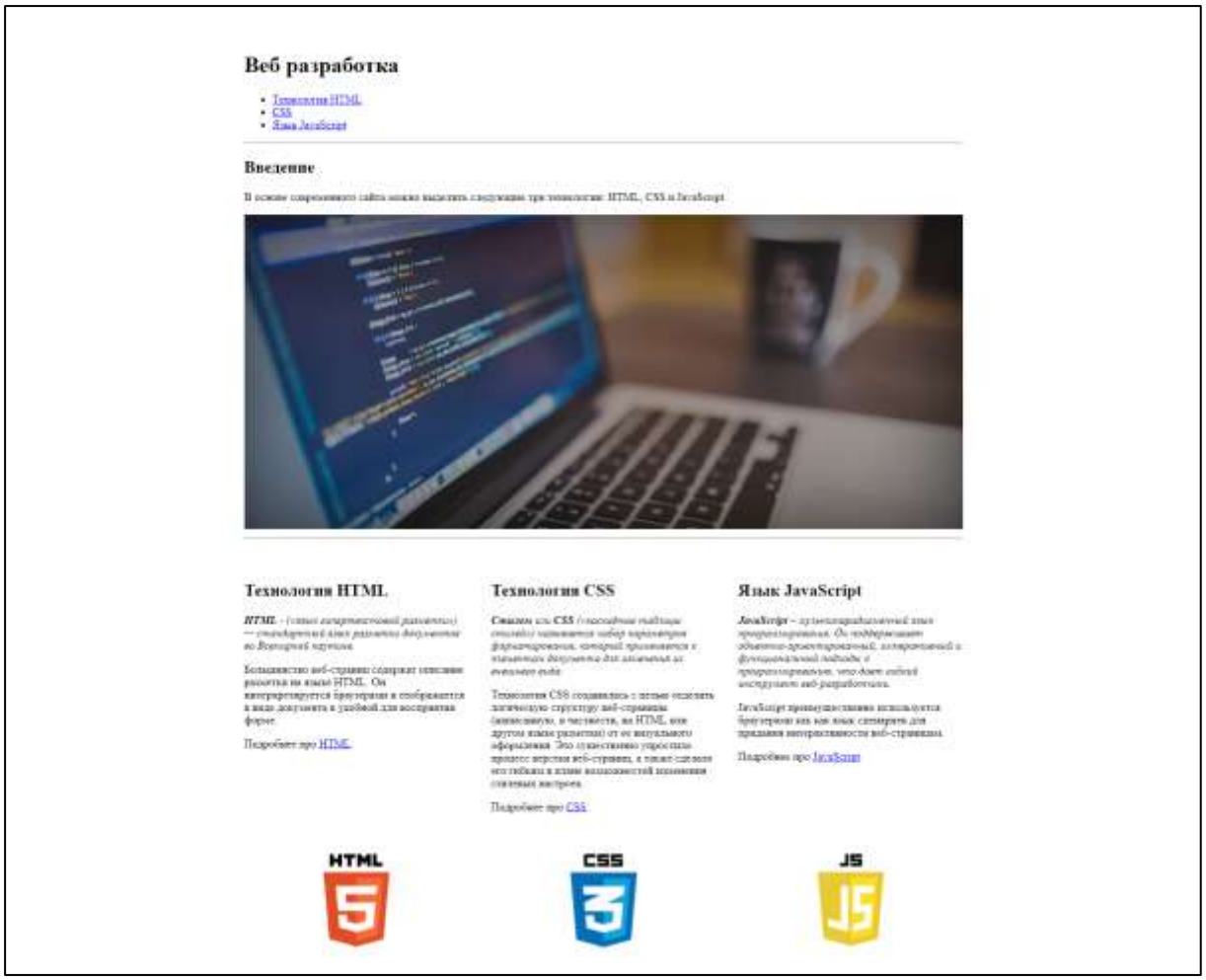


Рис. 1.213. Примеры веб-страниц с фиксированной шириной

2. Резиновый макет

Для *резинового макета* основному блоку значение ширины не задается в явной форме, поэтому его ширина будет зависеть от текущей ширины окна браузера (т.е. считаться равной 100%). Иными словами, если не брать во внимание границы и отступы блока, то содержимое сайта занимает всю доступную ширину окна браузера.

В случае изменения ширины окна браузера или на устройстве с другими размерами ширина блока с содержимым меняется автоматически и подгоняется уже под новое значение. При этом горизонтальная полоса прокрутки не появляется.

Особенности резинового макета

- Страница занимает всю доступную ширину окна браузера или экрана устройства.
- Ширина блоков и элементов на странице задается относительными величинами (в процентах), поэтому сохраняются пропорции.

Достоинства резинового макета

- Используется вся область веб-страницы.
- Удобен для печати: адаптируется под любой размер листа.
- Страница удачно смотрится на разных устройствах.

Недостатки резинового макета

- На широкоформатных мониторах основной блок расползается по всей доступной ширине, что приводит к существенному увеличению размеров внутренних элементов (если их размеры также заданы в процентах).
- Текстовые строки получается чрезмерно длинными, что осложняет восприятие информации.
- Усложняется процесс верстки: необходимо подобрать соотношения вложенных блоков так, чтобы содержимое гармонично смотрелось на различных устройствах.

Резиновые макеты – это простейший подход, позволяющий рационально использовать доступное пространство экрана устройства. Однако они не позволяют равнозначно учитывать всевозможные размеры экранов, с которых будут просматривать веб-сайт.

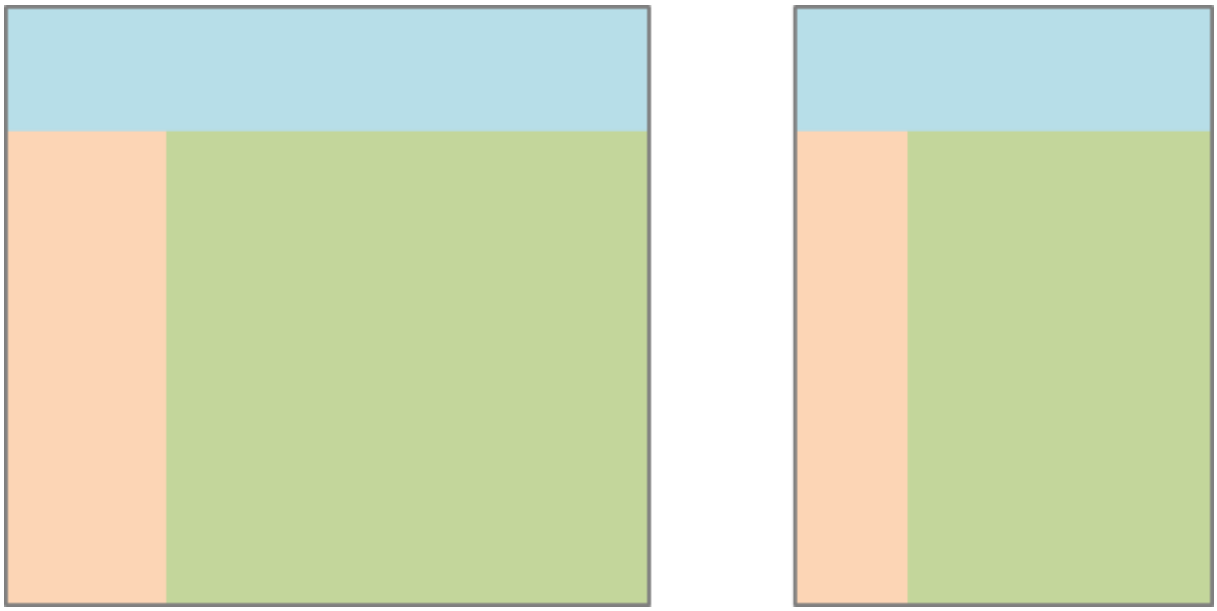


Рис. 1.214. Схема простого резинового макета

Рис. 1.215. Пример резиновой верстки веб-сайта

3. Эластичный макет

Эластичный макет обычно представляет собой развитие концепции резиновой верстки. В статичном состоянии содержимое эластичной веб-страницы растянуто по всей доступной ширине. Однако в случае изменения размера шрифта изменятся и элементы страницы.

Эластичный макет связан с размером шрифта, который задается в специальной относительной единице измерения – *EM* («*мут*», высота букв относительно текущего размера шрифта).

Особенности эластичного макета:

- В основе лежит фиксированный или резиновый макет.
- При изменении размеров окна размер элементов также меняется и сохраняет пропорции.
- Размер элементов задается в *em*, который привязан к размеру шрифта.

Достоинства эластичного макета:

- Макет в целом или отдельные его части легко масштабировать, сохраняя при этом пропорции.
- Макет одинаково хорошо отображается на разных устройствах и ОС.

Недостатки эластичного макета:

- Современные браузеры поддерживают функцию масштабирования веб-страницы, поэтому в эластичном макете нет особой необходимости.
- Сложность верстки макета: требуется тестирование на множестве устройств.
- Ограниченность применения.

Эластичные макеты в современной верстке веб-страниц не имеют широко распространения. Вместо них лучше использовать резиновый или адаптивный макет.

При работе с эластичным макетом следует ограничить максимально возможную ширину основного блока, чтобы размер шрифта и элементов не получился слишком большим для широкоформатных мониторов.

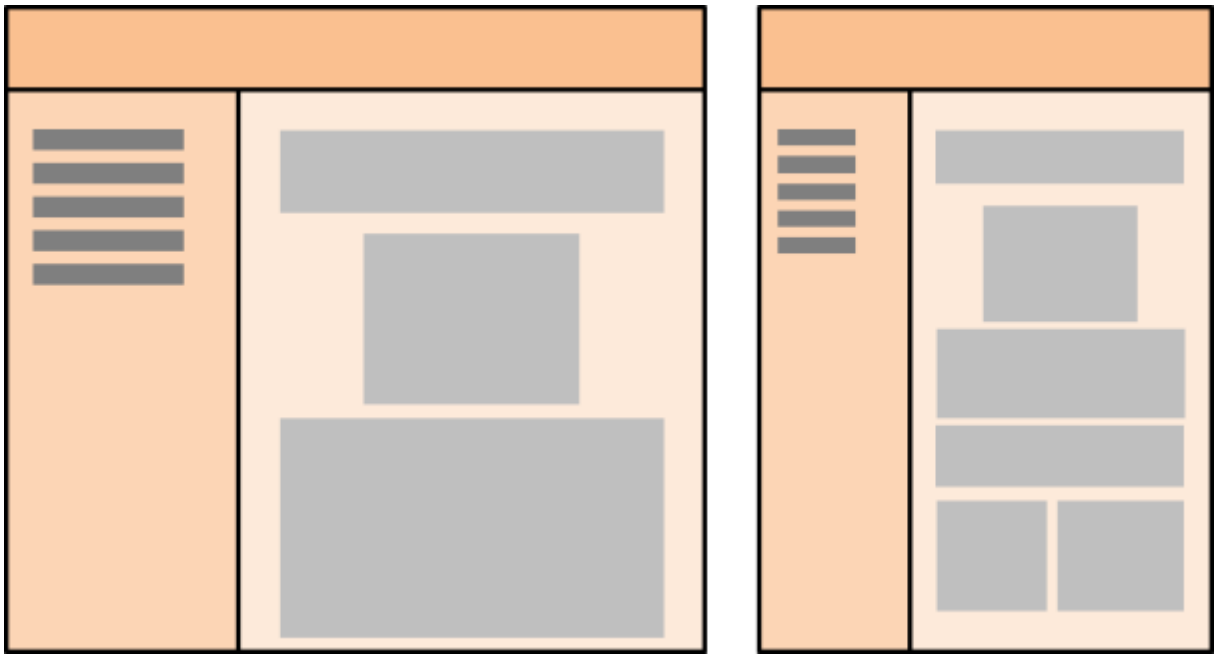


Рис. 1.216. Схема реализации эластичного макета

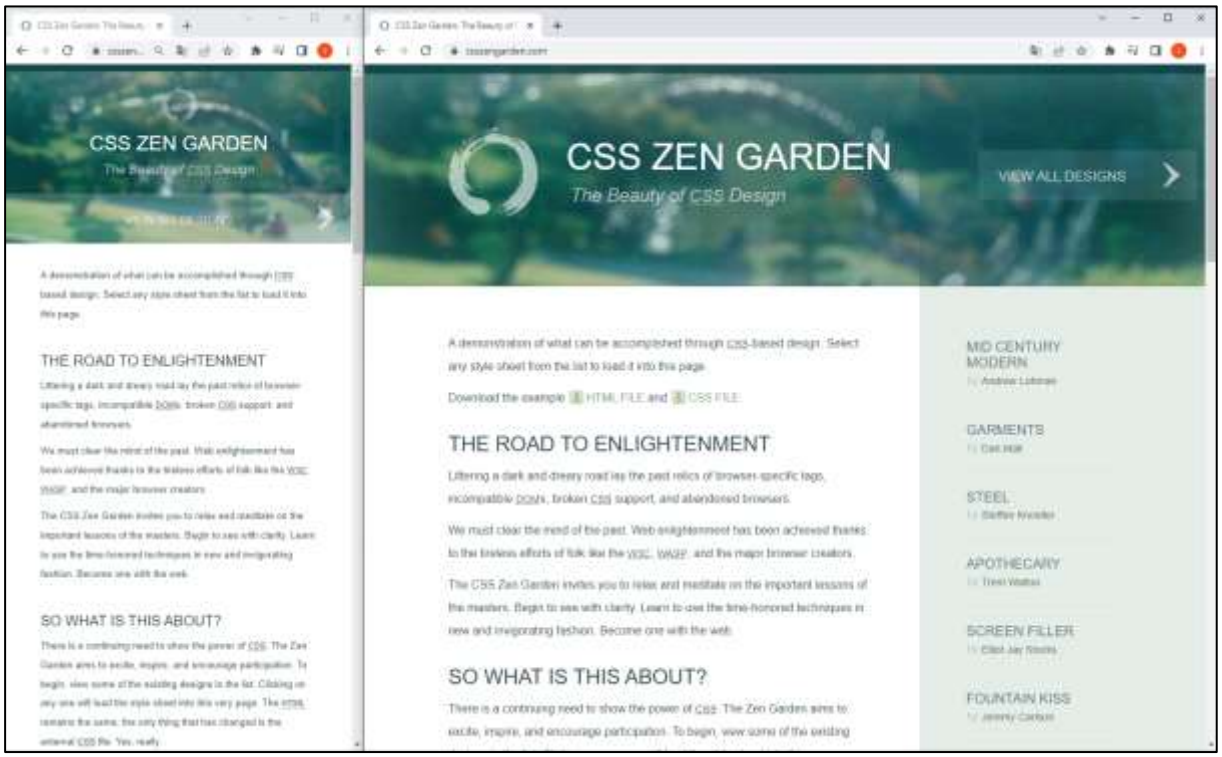


Рис. 1.217. Пример реализации эластичного макета (с элементами адаптивного дизайна)

4. Адаптивный макет

Адаптивный макет автоматически подстраивается под разрешение монитора и ширину окна браузера. При необходимости макет меняет позиции объектов, соотношение размеров разных блоков и элементов, количество колонок и даже оформление. Адаптивная верстка учитывает размеры экрана или окна браузера, ориентацию экрана, устройства вывода и взаимодействие пользователя со страницей.

В современных условиях многие возможности адаптивного дизайна достигаются уже с помощью встроенных средств спецификации CSS3. Большую гибкость и отзывчивость реализуют с использованием программных средств, обычно – JavaScript.

Особенности адаптивного макета

- Макет автоматически подстраивается под разрешение устройства и окна браузера.
- Используется набор из нескольких файлов CSS-стилей, которые срабатывают при определенных условиях.
- Выбор правил обработки может осуществляться с помощью скриптов (*адаптивный макет*) или на базе сеточной разметки и медиа-запросов CSS3 (*отзывчивый макет*).

Достоинства адаптивного макета

- Рационально используется доступное пространство.
- Адаптируется под разную ширину и экраны устройств.
- Разработчикам доступны многочисленные модули и фреймворки для ускорения верстки (в частности – адаптивные сетки).

Недостатки адаптивного макета

- Требует много времени на верстку и тестирование.
- Трудно учесть особенности отображения на всевозможных устройствах.
- Процесс разработки более затратный.

Адаптивная и отзывчивая верстки наиболее востребованы, поскольку пользователи обращаются к веб-сайтам с разных платформ и устройств. Адаптивный дизайн обеспечивает адекватный внешний вид сайта в разных условиях.

Рис. 1.218. Особенность поведения адаптивного макета на разных устройствах

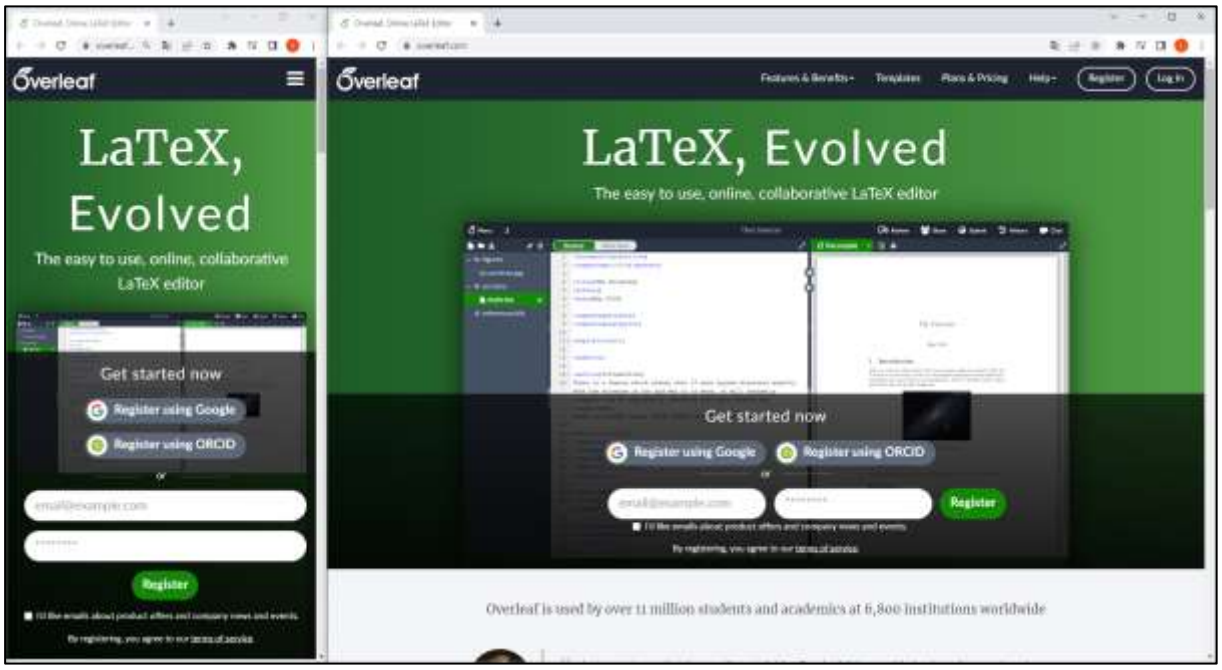


Рис. 1.219. Пример веб-сайта с адаптивным макетом

5. Макет под мобильное устройство

Мобильная верстка – это один из вариантов реализации макета сайта, который оптимальным образом адаптирует его к мобильному устройству. Мобильная верстка делится на две категории:

- *отзывчивая* предполагает способность макета автоматически растягиваться и сжиматься в разных условиях;
- *адаптивная* разделяет дизайн на несколько отдельных макетов, где каждый ориентирован на определенное разрешение экрана.

Мобильная верстка была востребована во времена активного развития телефонов и первых поколений смартфонов. Сейчас она становится мало актуальной, поскольку ее возможности полностью покрываются технологиями адаптивной верстки.

Особенности мобильного макета

- Является отдельной версией веб-сайта, с собственным дизайном, верстkой, содержимым и URL адресом.
- Заточен под особенности мобильных ОС.

Достоинства мобильного макета

- Используется вся область веб-страницы.
- Объекты достаточно крупные, что удобно для управления пальцами.
- Загрузка страниц требует меньшего трафика по сравнению с основным сайтом.

Недостатки мобильного макета

- Дизайн более простой.
- В целях экономии пространства на маленьком экране мобильного устройства могут быть скрыты некоторые текстовые или графические элементы, а также урезан функционал ресурса.
- Требуется дополнительный объем работы, связанный с созданием отдельной копии сайта, адаптированной под мобильную платформу.

Как было отмечено ранее, развитие технологий адаптивной вёрстки делает работу с мобильной верстkой мало актуальной.

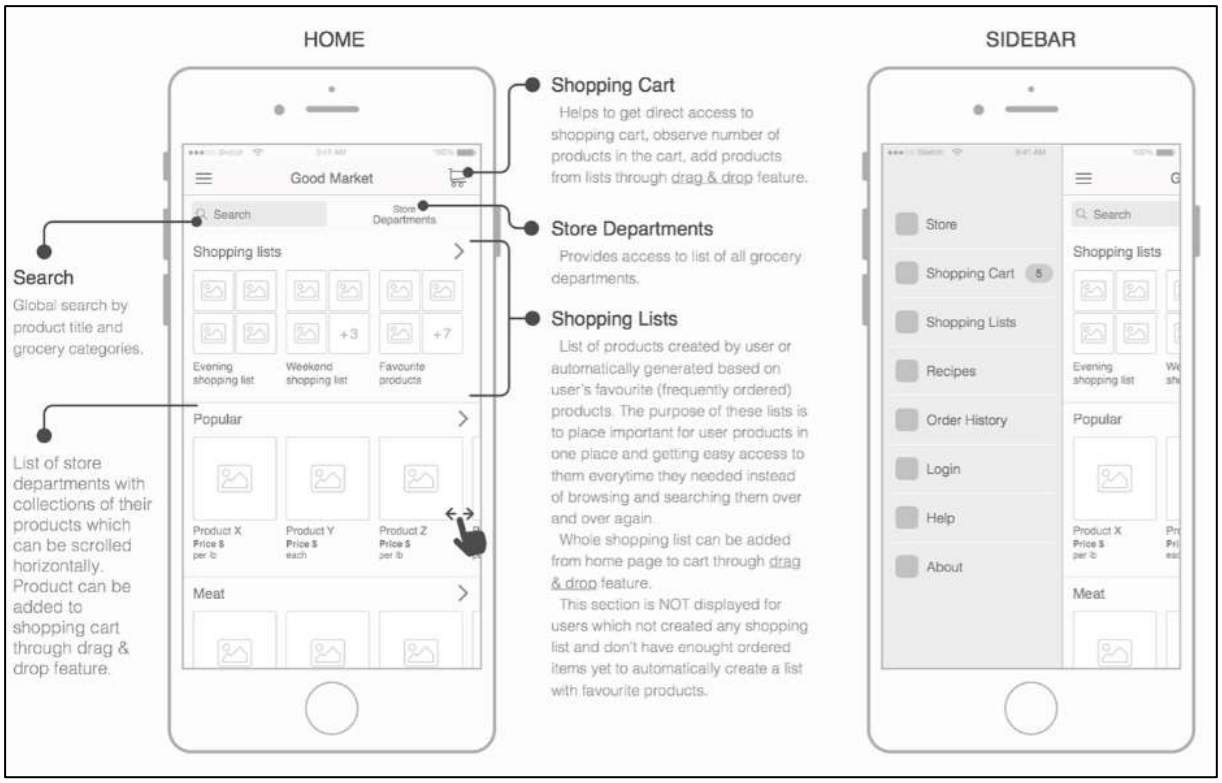


Рис. 1.220. Разработка дизайна под мобильные платформы

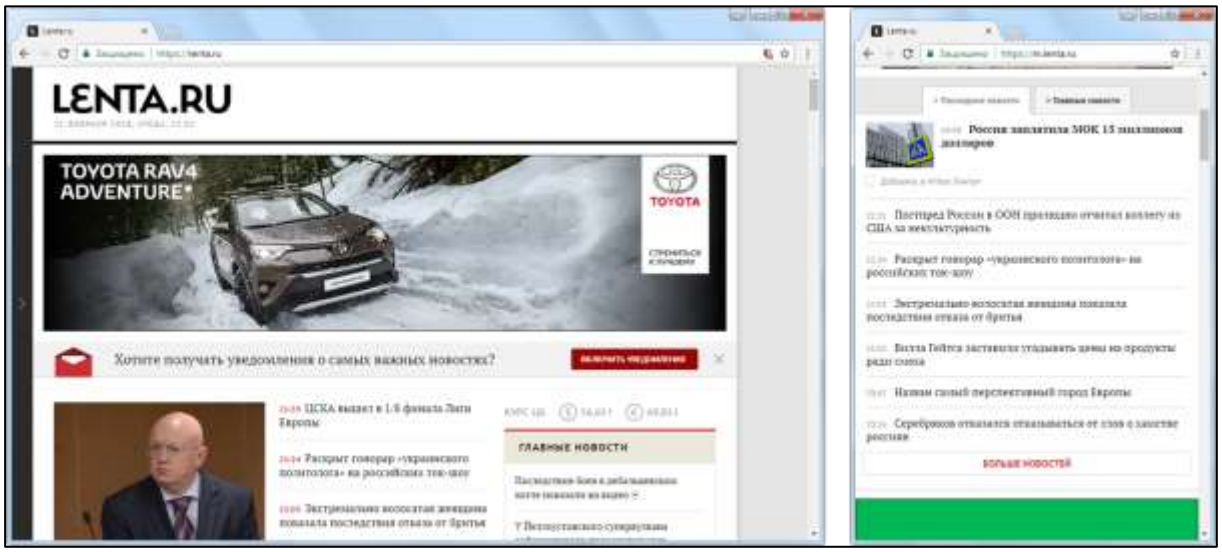


Рис. 1.221. Пример полной и мобильной версии сайта

1.7.2. Техника верстки веб-страниц

1. Табличная верстка

Табличная верстка структуры веб-страницы предполагает использование таблицы в роли каркаса (тега `<table></table>`). Содержимое размещается в ячейках таблицы. При этом границы таблицы не отображаются.

Длительное время табличная верстка была основной техникой разметки веб-страниц. Она обеспечивала простой способ размещения контента и не требовала глубоких познаний HTML.

Особенности табличной верстки

- Сетка макета строится на базе таблиц.
- Ячейки содержат контент или вложенные таблицы.

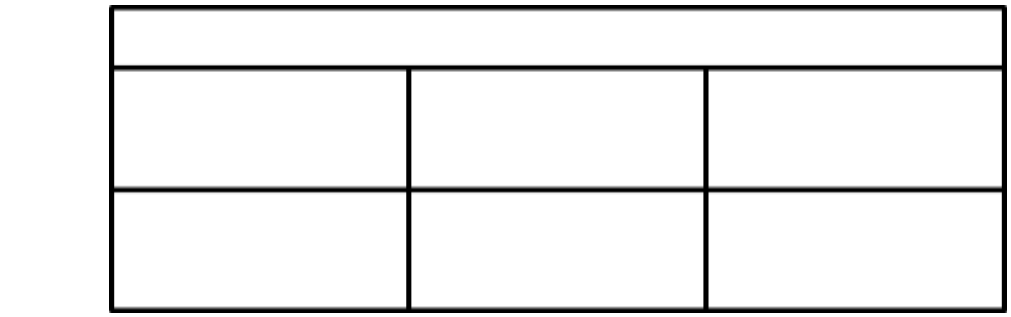
Достоинства табличной верстки

- Простая техника и высокая скорость верстки.
- Удобно разбивать содержимое на колонки.
- Таблицы формируются практически идентично всеми браузерами, в том числе старыми, поэтому исключается «конфликт браузеров».
- Удобна для макетов фиксированной ширины и резиновой верстки.

Недостатки табличной верстки

- Устаревший подход.
- Плохая индексация контента для поисковиков.
- Противоречит концепции HTML.
- Контент не загружается до тех пор, пока вся таблица не прорисована.
- Существенное ограничение возможностей оформления.
- Разрастание кода при большом объеме элементов.

В настоящее время техника табличной верстки является устаревшей и не рекомендуется к использованию даже при создании простых веб-страниц. Ее еще можно обнаружить в реализации старых веб-сайтов.



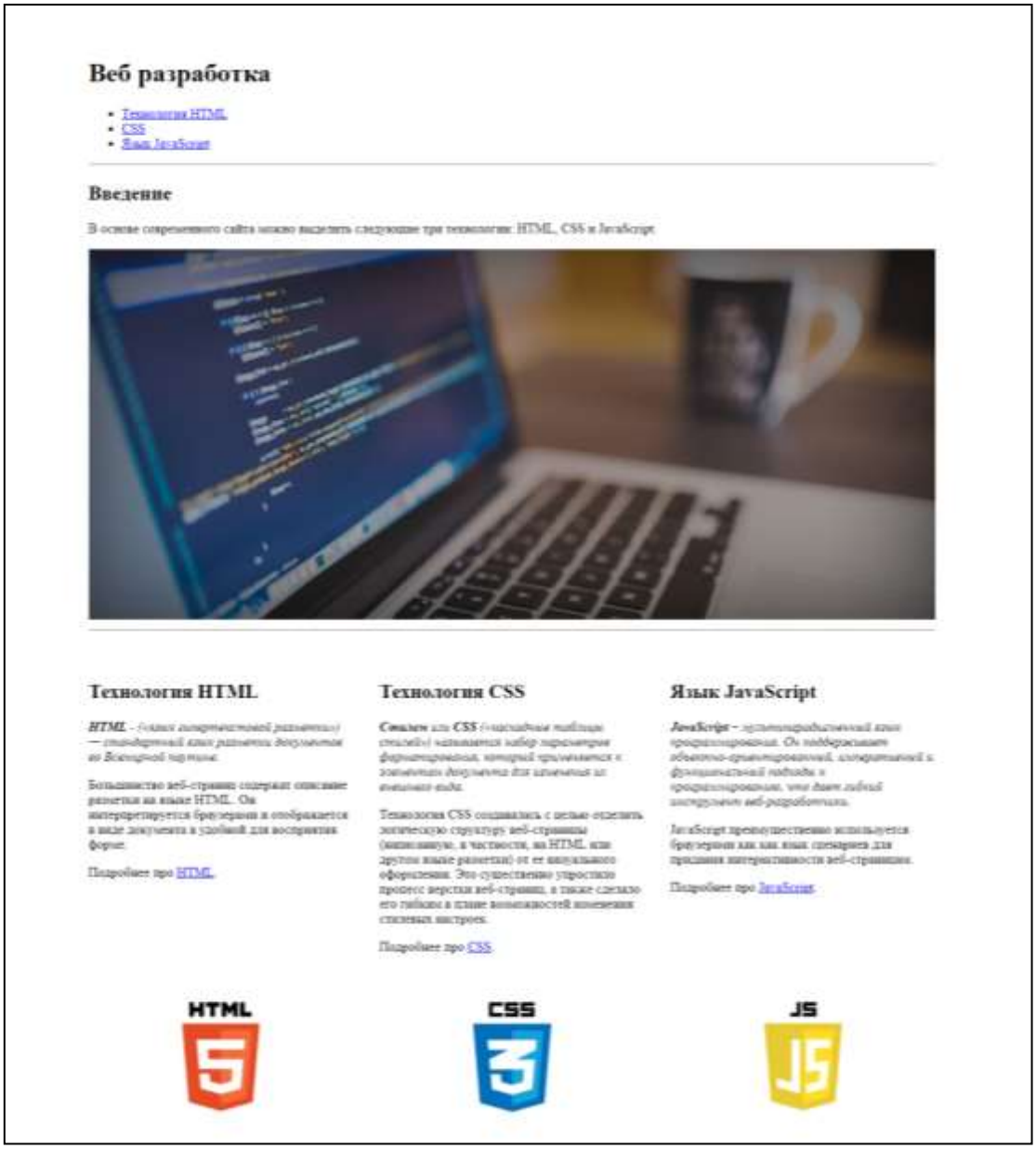


Рис. 1.222. При верстке веб-страницы использовалась следующая таблица из трех строк и трех колонок. Верхняя строка объединена в одну ячейку

2. Блочная верстка

Блочная верстка пришла на смену табличной и в настоящее время определяет ведущий подход к разметке веб-страниц. Основным структурным элементом блочной разметки выступает парный тег `<div></div>`, выполняющий роль абстрактного контейнера. Оформление блока, его позиционирование и влияние на содержимое определяется CSS-классами, которые подключаются к блоку.

Использование блоков позволяет делает верстку очень гибкой, что недоступно обычным таблицам.

Особенности блочной верстки

- Конструируется на базе блоков `<div>` (блочный элемент) и `<span>` (строковый элемент), которые вложены друг в друга. Также используются блочные новые теги HTML5.
- Настройка оформления и позиционирования блоков задается через классы и правила каскадирования стилей.

Достоинства блочной верстки

- Более компактна и создает меньшую нагрузку на сервер.
- CSS-стили отделены от HTML-разметки.
- Возможность реализации адаптивного дизайна.
- Более приемлема для SEO-оптимизации.
- Возможна реализация сложного и уникального дизайна.

Недостатки блочной верстки

- Сложность реализации и более высокий порог вхождения (требуется детально изучить возможности и особенности CSS).
- Труднее достичь одинакового результата в разных браузерах, т.к. не все новые CSS-свойства поддерживаются в надлежащей форме.

Несмотря на трудности работы с блочной версткой и необходимости детального погружения в ее специфику, техника разметки блоками является основным подходом к созданию современных интерактивных веб-страниц.

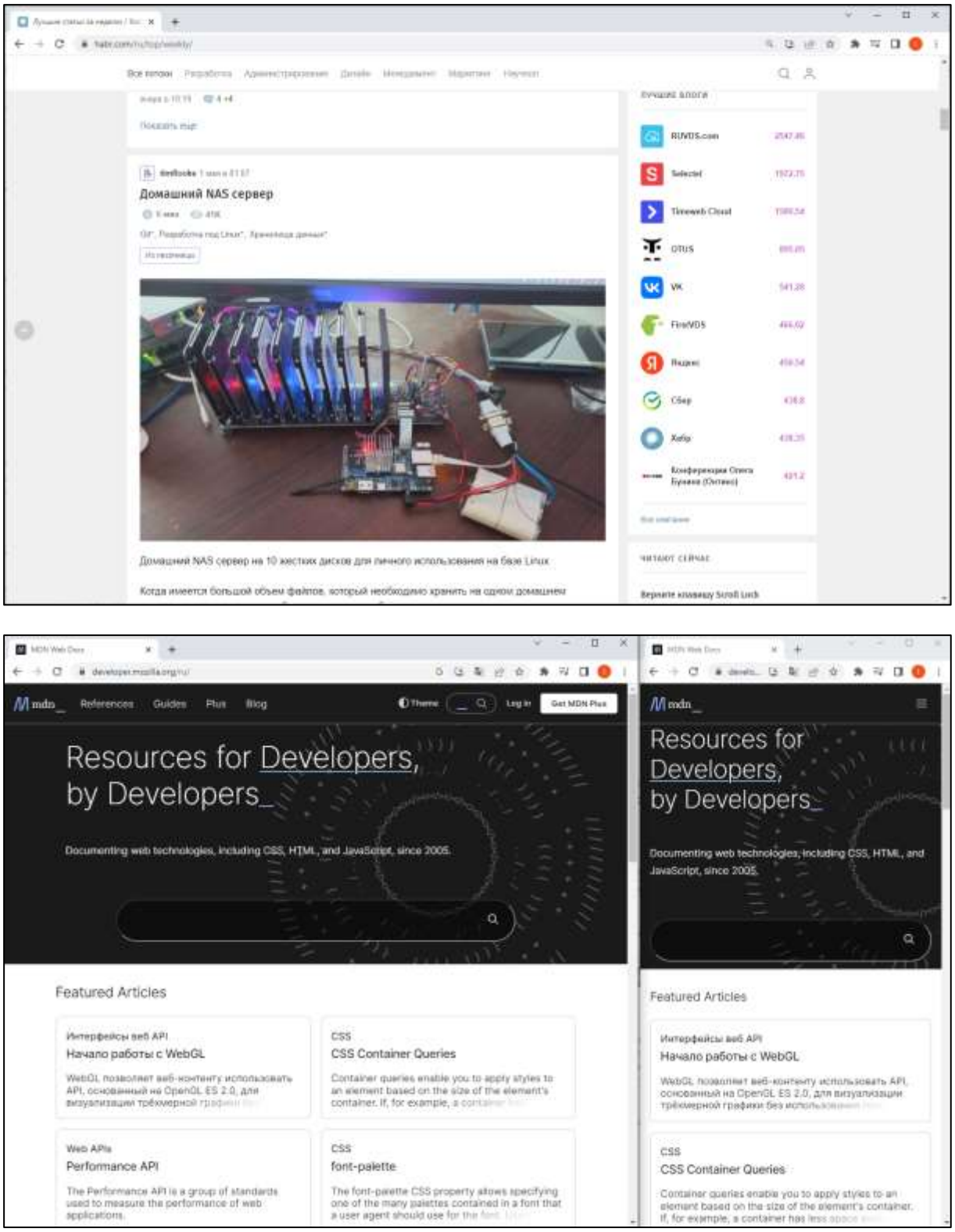


Рис. 1.223. Примеры веб-сайтов с блочной верстkой (поддерживают адаптивный дизайн)

3. Верстка слоями

Верстка слоями предполагает работу с HTML-элементами, которые расположены на веб-странице методом наложения друг на друга с пиксельной точностью. Управление форматированием и позицией слоёв осуществляется с помощью языков JavaScript и VBScript, которые также позволяют достичь некоторых эффектов анимации.

Использование слоев в разметке в свое время позволило отойти от табличной техники верстки. Однако с развитием возможностей CSS значимость разметки слоями стала падать и вытесняться более универсальной техникой блочной верстки с помощью лишь HTML и CSS.

Особенности верстки слоями

- Слои внедряются в страницу сайта наложением друг на друга с пиксельной точностью.
- Управление слоями производится с помощью JavaScript и VBScript.

Достоинства верстки слоями

- Разметка быстро обрабатывается браузерами.
- Позволяет реализовать необычные эффекты анимации.
- Точное позиционирование элементов и сопоставление их в разных слоях.

Недостатки верстки слоями

- Высокий порог вхождения: необходим опыт веб-программирования на JavaScript, VBScript, владение CSS.
- Подход не стандартизирован, возможна неоднородная работа в различных браузерах.

Вёрстка слоями позволяет проявить верстальщику свои творческие способности в дизайне страниц. Однако сегодня она также теряет актуальность по целому ряду причин, в частности:

- язык VBScript устарел, в то время как JavaScript по возможностям ушел далеко вперед;
- отображения страниц браузерами может отличаться.

Рис. 1.224. Каждый слой имеет определенную разметку и стилизацию

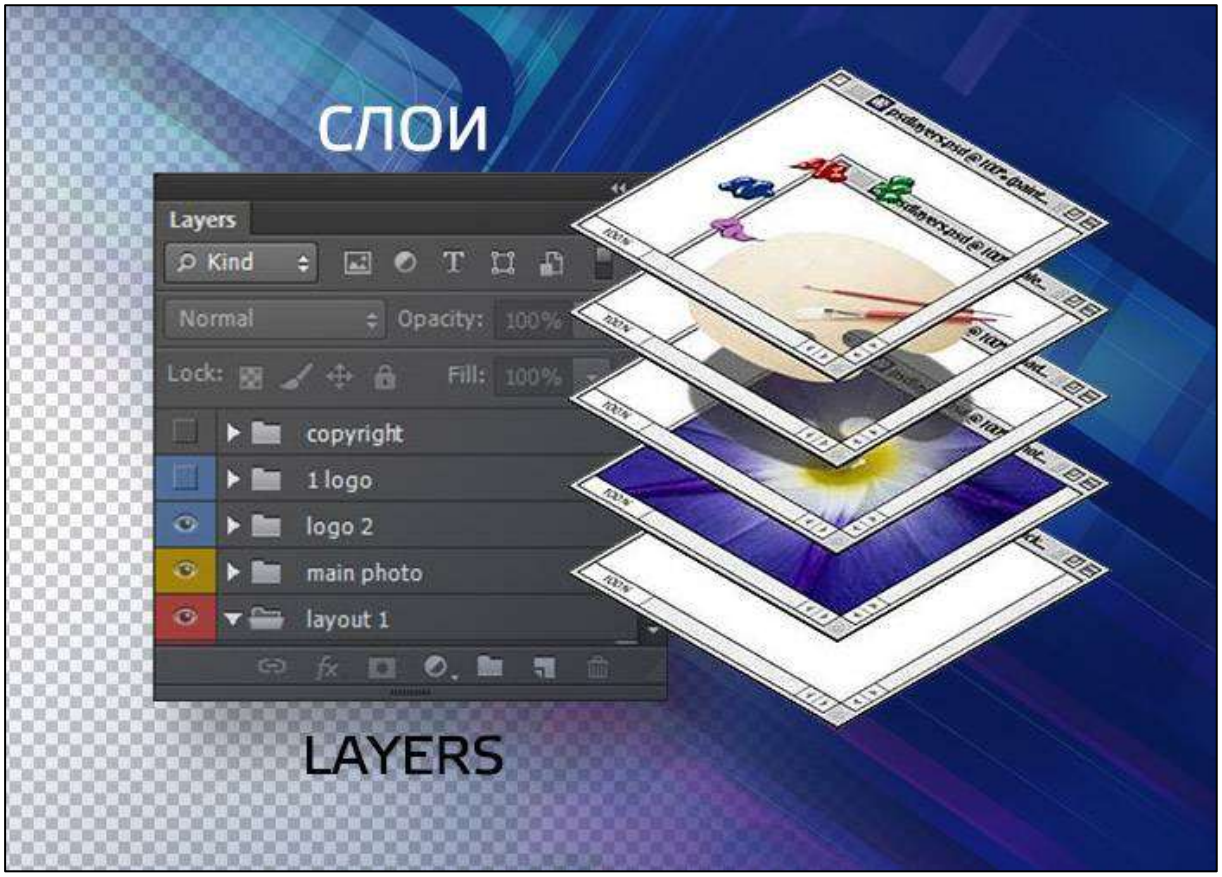


Рис. 1.225. Принцип верстки слоями очень близок к работе со слоями в графических редакторах

4. Гибкая верстка

Гибкая верстка или *flex-верстка* стала продолжением техники блочной верстки. Технология Flexbox, появившаяся в CSS3, даёт верстальщику очень удобный способ расположения и выравнивания блоков с учетом их динамичного поведения.

Flexbox позволяет менять ширину и высоту элементов, учитывая оптимальную позицию внутри родительского контейнера. Это делает отображение flex-разметки практически универсальной и снимает проблему работы с разными типами браузеров и экранов.

Особенности гибкой верстки

- Вначале применяется техника блочной верстки, а затем блоки с помощью классов получают flex-свойства.
- В стилях элемента указывают `display: flex`.
- Элементу можно указать направление главной оси и выравнивание.
- Flexbox не зависит от направления, в отличие от обычных блочных шаблонов.

Достоинства гибкой верстки

- Быстрая настройка «резиновых» блоков и их выравнивание.
- Элементы могут автоматически выстраиваться в несколько строк или столбцов.
- Сравнительно простой CSS-синтаксис и небольшой набор требуемых свойств.

Недостатки гибкой верстки

- Иногда возможности flex-блоков излишни.
- В некоторых браузерах имеются проблемы корректного отображения гибких блоков.

Технология Flexbox расширяет возможности блочной верстки и позволяет достичь отзывчивый дизайн более простыми способами. В частности она упрощает работу с позиционированием блоков, что в обычной блочной технике разметки достигается с помощью работы с плавающими блоками и режимами позиционирования. Кроме того, в последние годы популярные браузеры стабильно поддерживают работу с Flexbox.

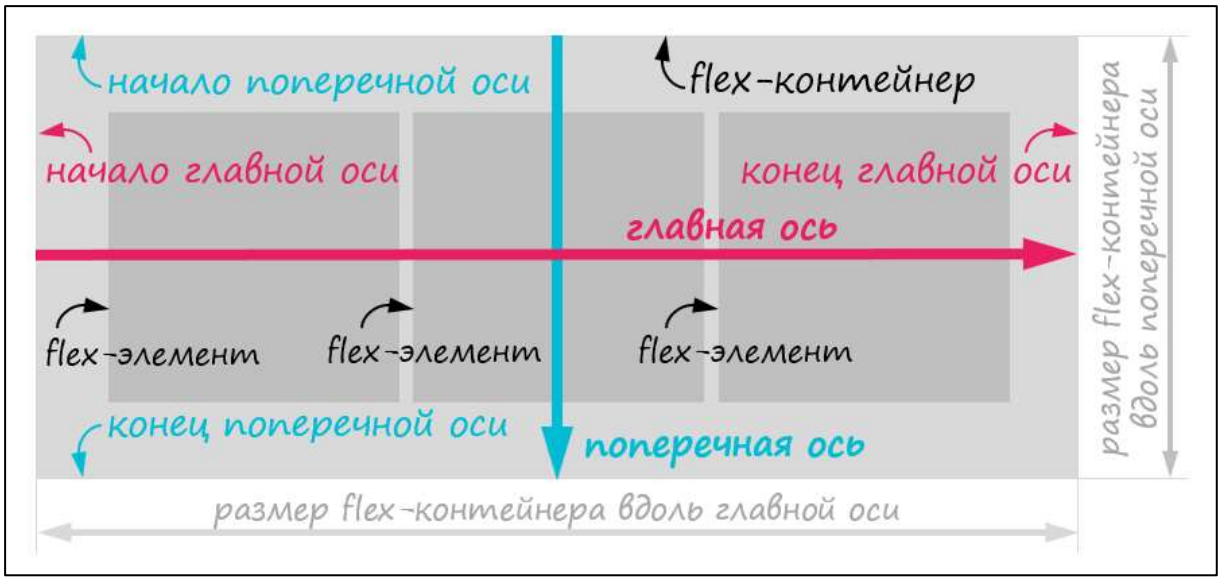


Рис. 1.226. Flex-верстка позволяет управлять расположением блоков, изменяя лишь некоторые общие параметры

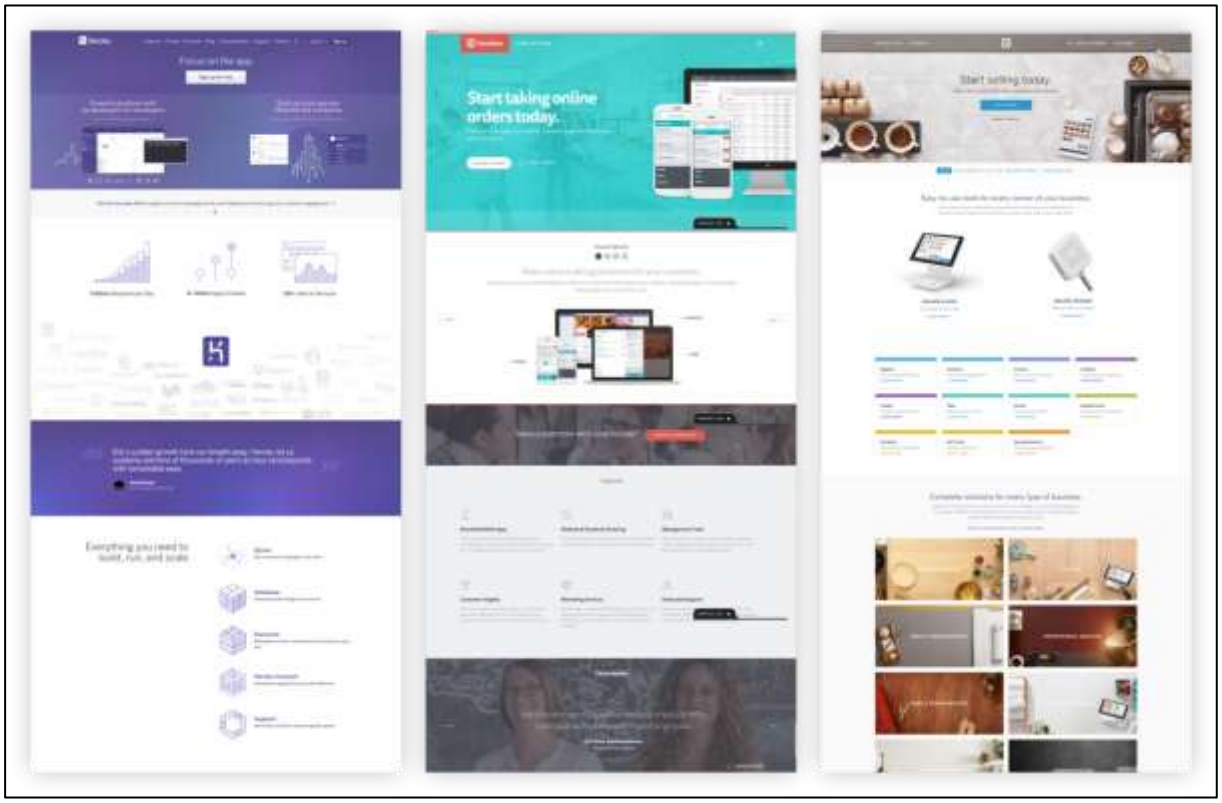


Рис. 1.227. Гибкая верстка делает веб-страницу адаптивной к разным экранам

5. Другие типы верстки

Верстка фреймами

Фрейм представляет собой блок, внутри которого отображается независимая от других HTML-страница. Для его разметки используется парный тег `<frame></frame>`, который располагается внутри тега `<frameset></frameset>`. Фрейм делит окно браузера на отдельные области. При этом каждый фрейм может иметь и собственный URL.

Работа с фреймами имеет ряд преимуществ:

- позволяет исключить дублирование разметки и разбить ее на отельные файлы;
- распределяет содержимое в отдельные окна;
- могут работать с формами ответа.

Однако фреймы имеют существенные недостатки:

- это устаревший подход к разметке;
- страницы хуже индексируются;
- требуют настройки для мобильных платформ.

В современной веб-разработке использовать фреймы настоятельно не рекомендуется.

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Frameset//EN" "http://www.w3.org/TR/html4/frameset.dtd">
<html>
    <head>
        <meta http-equiv="Content-Type" content="text/html; charset=utf-8">
        <title>Фреймы</title>
    </head>

    <frameset cols="100,*">
        <frame src="menu.html" name="MENU">
        <frame src="content.html" name="CONTENT">
    </frameset>
</html>
```

Рис. 1.228. Разбиение области веб-страницы на два фрейма

Семантическая верстка

Семантическая верстка ставит целью осуществить разметку страницы с использованием тегов, задающих содержимому определенную роль. Это одно из важнейших требований, обеспечивающих грамотную SEO-оптимизацию. Семантические теги необходимы поисковым системам, чтобы верно проанализировать запросы пользователя и подобрать наиболее релевантные ответы в виде ссылок на веб-страницы.

Поисковая система в целях оптимизации не сканирует вначале всю веб-страницу, а пытается найти требуемую информацию по ряду ключевых слов – *метатегов*. Метатеги определяют структуру и тип документа, вложенность заголовков, связи между контентом.

Это полезно знать!

Важно заметить, что спецификация HTML5 построена с учетом значимости семантической разметки. В нее были добавлены новые теги, определяющие основные подразделы сайта и их роль в разметке.

Работа с семантической разметкой имеет ряд существенных преимуществ:

- связь между элементами разметки более прозрачная;
- сайт более релевантен в выдаче поисковиков;
- семантические теги стандартизированы.

Рис. 1.229. Семантическая разметка также уменьшает число используемых классов (которые не несут какого-либо смысла для поисковика)

Валидная верстка

Валидная вёрстка представляет собой HTML-разметку, которая соответствует конкретной спецификации (версии) HTML и написана с учетом принятых стандартов. За стандартизацию языков разметки и веб-программирования отвечает Консорциум Всемирной Паутины World Wide Web Consortium (W3C).

Валидность гарантирует, что:

- веб-страница будет корректно отображаться в разных браузерах;
- в коде разметки отсутствуют ошибки (в названии тегов, непарные кавычки или скобки и т.п.);
- HTML-код набран в строгом соответствии со спецификацией, а также не содержит устаревших тегов и их атрибутов.

Веб-разработчик может отслеживать валидность самостоятельно, руководствуясь своим опытом и глубоким познанием HTML. Однако на практике объемы веб-страниц и их число может быть огромным, поэтому следует использовать специальные программы – валидаторы.

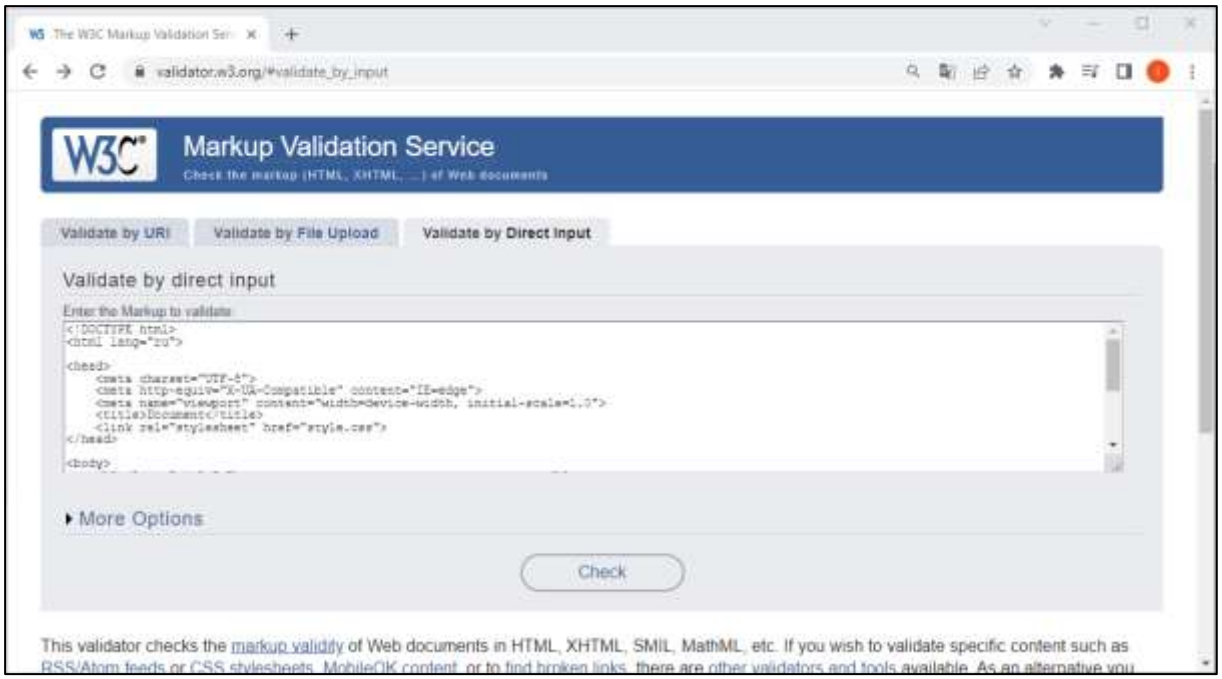
Валидатор анализирует синтаксис разметки, выявляет синтаксические ошибки в ключевых словах и некорректную структуру вложения тегов. Несмотря на то, что современные браузеры весьма лояльны к потенциальным ошибкам в разметке и зачастую их могут игнорировать, валидация крайне важна.

Перечислим некоторые примеры онлайн валидаторов:

- https://validator.w3.org/;
- https://h5validator.appspot.com/dcm/asset;
- https://en.rakko.tools/tools/58/.

Современные среды разработки и текстовые редакторы отслеживают ошибки в коде и устаревшие теги, что избавляет от разработчика от потенциальных проблем уже на этапе набора кода.

Кроме того, валидацию кода можно осуществлять и средствами браузера Google Chrome: для этого необходимо установить плагин Web Developer.



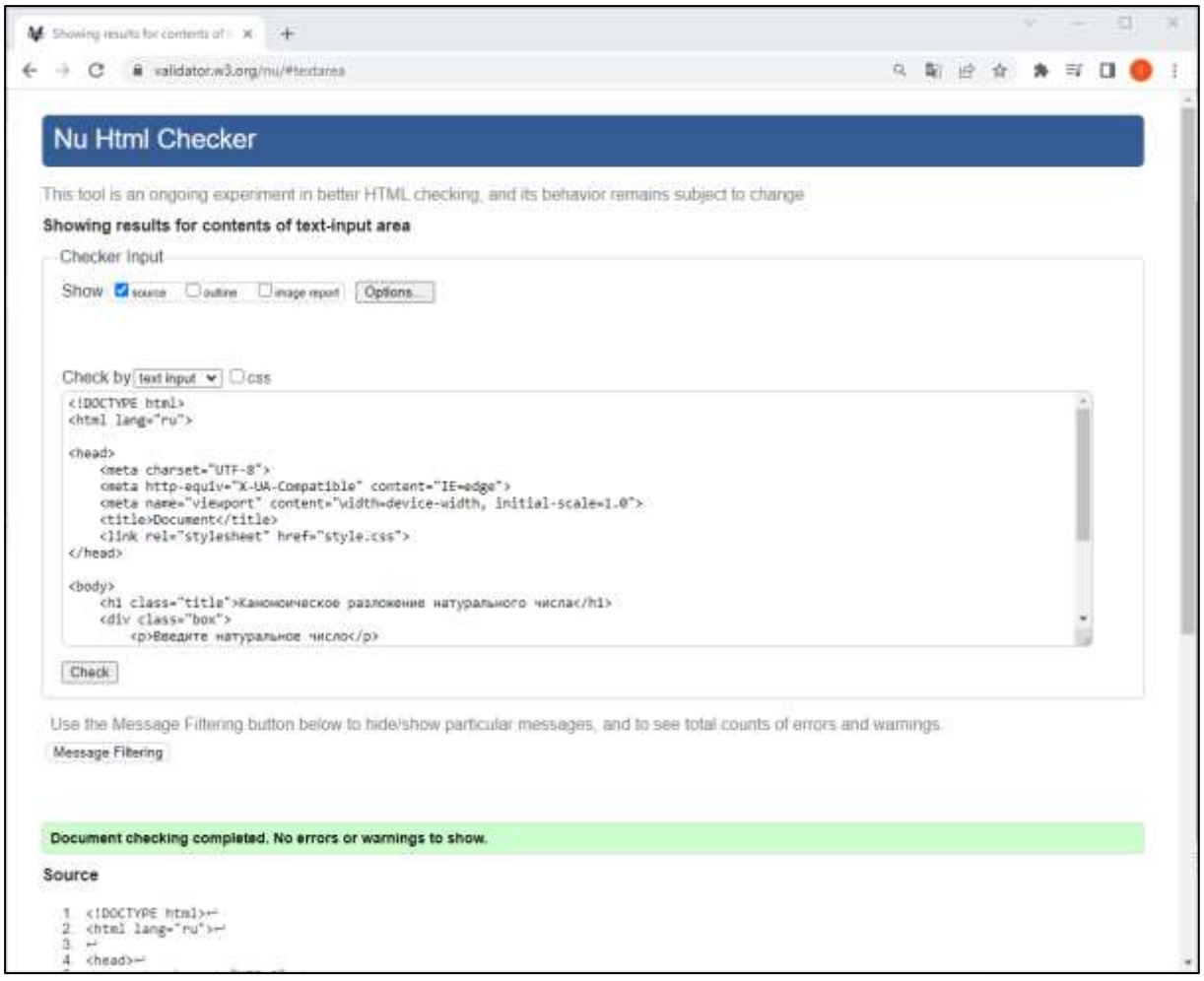


Рис. 1.230. Проверка веб-страницы: валидация пройдена успешно, ошибок и недочетов не обнаружено

Кроссбраузерная верстка

Кроссбраузерность – это способность веб-страницы корректно и одинаково отображаться в браузерах от разных производителей.

Проблема кроссбраузерности имеет давние корни и связана с наличием многочисленных браузеров, которые пользуются популярностью у пользователей. Исторически каждый производитель браузера использовал собственные алгоритмы построения веб-страниц и рендера их внешнего вида. Поэтому пользователи сталкивались с множеством проблем, в частности – одни и те же веб-сайты в разных браузерах отображались совершенно по-разному, а в некоторых случаях – некорректно.

Кроме того, проблемы с обработкой разметок и стилей были связаны с постоянным развитием технологий HTML и CSS. Не все производители своевременно добавляли поддержку новых тегов и свойств CSS.

К настоящему моменту ведущие производители браузеров (таких, как Google Chrome, Mozzila Firefox, Yandex, Edge, Opera и др.) практически полностью поддерживают все нововведения спецификаций HTML5 и CSS3, поэтому проблема кроссбраузерной поддержки постепенно снимается.

Однако веб-разработчик при верстке проектов должен руководствоваться рядом правил, которые помогут уменьшить риск потенциальных проблем с браузерами:

- Следует использовать *сброс* или *нормализацию* стандартных стилей основных тегов. Можно использовать уже проверенные и рекомендованные другими разработчиками. Например: Reset.css и Normalize.css.
- Для новых экспериментальных CSS-свойств следует использовать *вендорные префиксы*.
- Поддержку новых CSS-свойств следует проверять на сайте caniuse.com, который позволяет узнать, какие версии браузеров способны с ними работать.
- Важно тестировать веб-сайт в разных браузерах и некоторых предыдущих версиях.
- Следите за актуальными новостями на сайтах W3C и MDN о последних нововведениях и стандартах.

```
p {
    -webkit-transition-duration: 0.75s;
    -moz-transition-duration: 0.75s;
    -o-transition-duration: 0.75s;
    -ms-transition-duration: 0.75s;
    transition-duration: 0.75s;
}
```

Основное свойство

-webkit-
-moz-
-o-
-ms-

Вендорные префиксы

Рис. 1.231. Вендорные префиксы указывают на браузер, который потенциально может поддерживать экспериментальное свойство

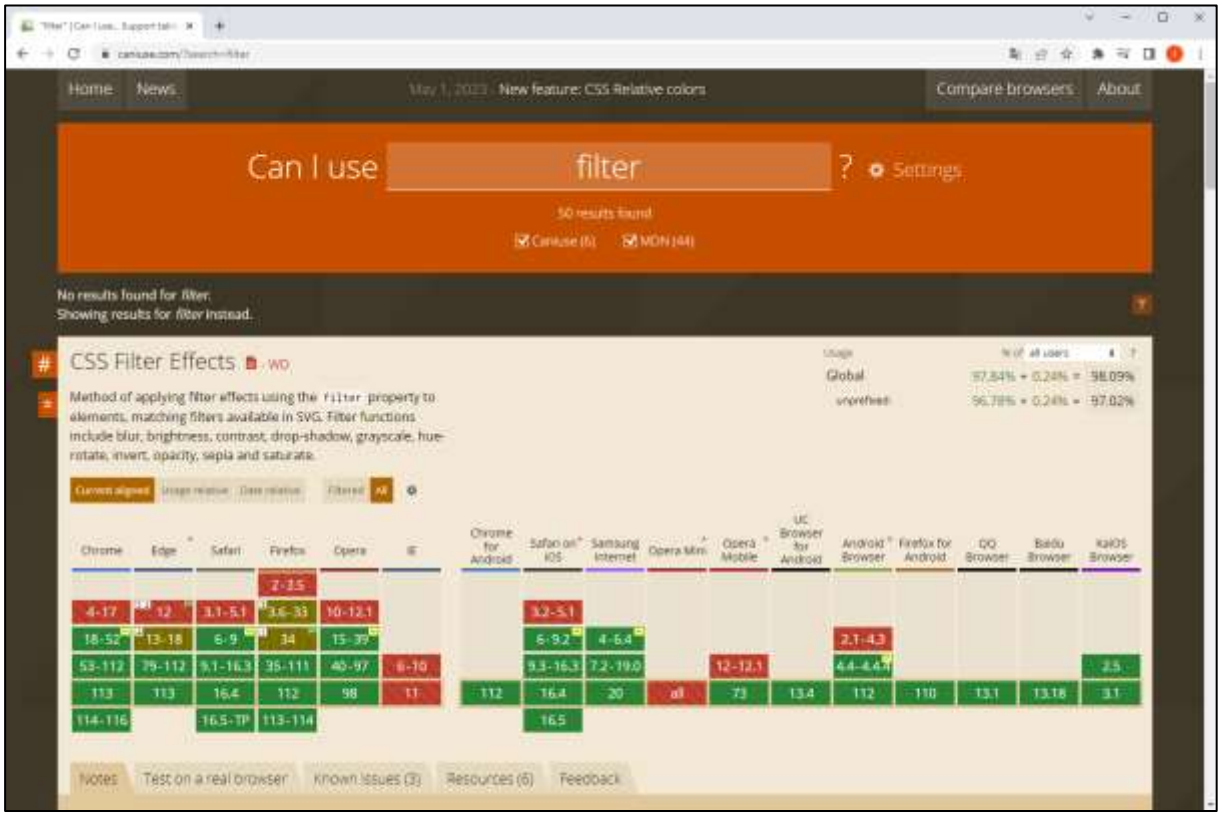


Рис. 1.232. Проверка поддержки CSS-свойств в разных браузерах

Вопросы для самопроверки

1. Перечислите основные типы макетов сайта.
2. Для каких типов сайтов можно использовать фиксированный макет?
3. В чем преимущества и недостатки резинового макета?
4. Что нужно учитывать при верстке резинового макета?
5. Чем отличается эластичный макет от резинового?
6. Почему в настоящее время сайты с адаптивным макетом самые востребованные?
7. На сколько актуальна сегодня вёрстка сайтов под мобильные устройства?
8. Почему в современных условиях табличная верстка является нежелательной?
9. Возможно ли обеспечить адаптивную верстку сайта на основе таблиц?
10. Какие проблемы и трудности возникают при вёрстке вебстраниц слоями?
11. В чем состоит универсальность техники блочной верстки?
12. Опишите возможности и роль тега `<div>` в разметке.
13. Возможно ли осуществить разметку контента в несколько колонок, используя блочную верстку?
14. Какие новые возможности привносит технология Flexbox в блочную разметку?
15. Что представляют собой фреймы и почему от практики их использования отказываются?
16. Для чего необходима семантическая верстка и можно ли ее обеспечить в любой из перечисленных техник разметки?
17. Для каких целей используются валидаторы разметки?
18. Перечислите инструменты, которые можно использовать для валидации HTML-кода.
19. Опишите проблемы, которые решает кроссбраузерная верстка.

Практикум

1. Работа с макетами

Задание 1

1. Проанализируйте веб-сайт вашего вуза.
2. Какой макет и подход к верстке был использован при его создании? По каким признакам вы это определили?

Задание 2

1. Для выполнения задания используйте сеть Интернет.
2. Найдите 2-3 примера веб-страниц, реализованных с помощью фиксированного макета.
3. Проанализируйте корректность отображения содержимого с экрана ПК и мобильного устройства? Какие недостатки или проблемы вы зафиксировали?

2. Работа с разметкой

Задание 1

1. Найдите 2-3 примера сайта с блочной верстkой.
2. Используются ли в разметке семантические теги?
3. Используются ли в разметке новые теги основной структуры спецификации HTML5.

Задание 2

1. Задан следующий фрагмент разметки.
2. Перепишите ее код, используя, там где возможно, семантические теги.

Глава 1\Семантика.html

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta http-equiv="X-UA-Compatible" content="IE=edge">
    <meta name="viewport" content="width=device-width,
initial-scale=1.0">
    <title>Document</title>
</head>
```

```
<body>
    <div class="main-box container">
        <div class="site__header col-12"></div>
        <div class="site__menu col-3"></div>
        <div class="site__content col-9"></div>
        <div class="site__footer col-12"></div>
    </div>
</body>
</html>
```

Задание 3

1. Задан следующий код разметки.
2. Осуществите его валидацию по спецификации HTML5.
3. Исправьте ошибки и недочеты.

Глава 1\Валидация.html

```
<!DOCTYPE html>
<html lang="ru">
<head>
    <meta charset="UTF8">
    <meta http-equiv="X-UA-Compatible" content=IE=edge>
    <meta name="viewport" content="width=device-width,
initial-scale=1.0>
    <title>Document</title>
</head>
<body>
    <h1>Валидаторы кода</h1>
    <p><strong>Валидатор</strong> - это программа,
    которая анализирует синтаксис разметки, выявляет
    синтаксические ошибки в ключевых словах и
    некорректную структуру вложения тегов</p>
    <p><emВалидность</em> кода гарантирует, что:</p>
    <ul>
        <li>веб-страница будет корректно отображаться
        в разных браузерах;</li>
        <li>в коде разметки отсутствуют ошибки
        (в названии тегов, непарные кавычки или скобки
        и т.п.);</li>
        <li>HTML-код набран в строгом соответствии
        со спецификацией, а также не содержит устаревших
        тегов и их атрибутов.</li>
    </ul>
</body>
</html>
```

1.8. Верстка веб-страниц

1.8.1. Особенности верстки

Верстка

Определение

Вёрстка веб-страниц *– это процесс разметки структуры веб-страницы с помощью HTML и ее оформление CSS-стилями согласно разработанному макету.*

Термин «верстка» позаимствован из сферы издательской деятельности и означает разметку текстовым и графическим содержимым страниц в газетах, журналах, книгах, плакатах и других печатных и электронных изданиях. Современные цифровые технологии, используемые издательствами, существенно упрощают верстку материала. В частности, позволяют использовать редакторы с графическим и интерактивным интерфейсом, а также языки разметки (например, LaTeX).

Основной задачей верстальщика веб-страниц является набор кода, который переводит дизайн-макет с графического представления в веб-документ на языках HTML и CSS.

Однако веб-верстка существенно отличается от типографической, поскольку веб-разработчик должен учитывать особенности отображения элементов в разных браузерах, размеры экранов или окна браузера, версии браузера и множество других факторов.

Это полезно знать!

Веб-верстка осложняется тем, что разработчику нужно учитывать возможную динамику изменения содержимого веб-страницы.

Требования к верстке современных сайтов

1. *Адекватная требованиям скорость загрузки веб-сайта*. Необходимо обеспечить высокую скорость загрузки содержимого страниц. Если в документе содержатся интерактивные элементы, требующего длительного времени на загрузку, следует частично отобразить интерфейс страницы.
2. *Соответствие стандартам HTML*. Все веб-страницы должны быть проверены валидатором, исключены все устаревшие теги, атрибуты и CSS-свойства.
3. *Адекватность отображения в браузере*. Корректность отображения страниц веб-сайта должна быть проверена во всех наиболее популярных браузерах, в частности – на мобильных платформах.
4. *Соответствие требованиям поисковых систем*. HTML-разметка должна быть выстроена с использованием правил семантической разметки и SEO-оптимизации.
5. *Адаптивность под различные экраны и устройства пользователя*. Сайт следует верстать с учетом возможности организации адаптивного и отзывчивого дизайна. На любом устройстве пользователю должен быть обеспечен минимально необходимый для работы функционал.
6. *Минимум кода*. Структура разметки должна быть максимально оптимизирована. Блочные теги следует по возможности заменить на семантические, убрать необязательные промежуточные теги в структуре. Чем меньше файл, тем меньше требуется интернет-трафик и выше скорость загрузки страницы.

Определение

Отзывчивый дизайн *– это проектирование веб-сайта с определенными значениями свойств, которые позволяют макету адаптироваться под разные типы устройств.*
Адаптивный дизайн *– это проектирование веб-сайта на основе нескольких макетов, которые автоматически выбираются в соответствии с используемым устройством.*

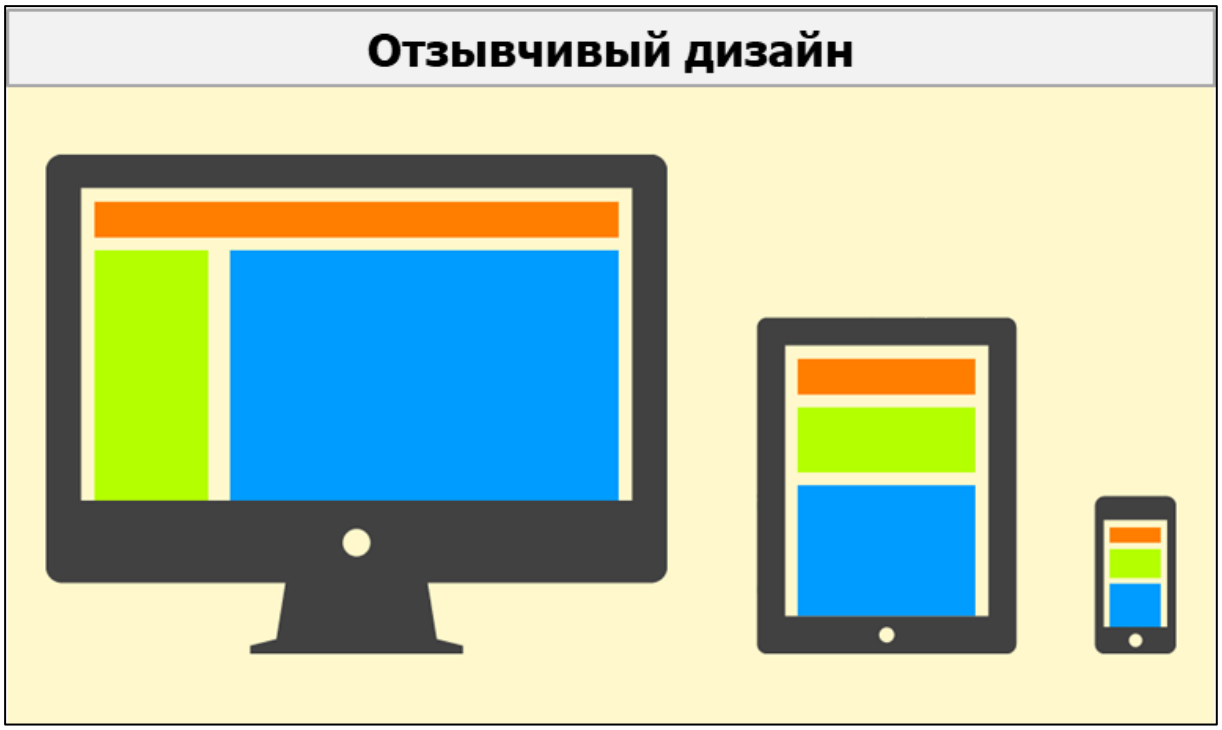


Рис. 1.233. Отзывчивый дизайн адаптируется под доступную ширину окна браузера или экрана устройства. За управление его свойствами отвечает CSS

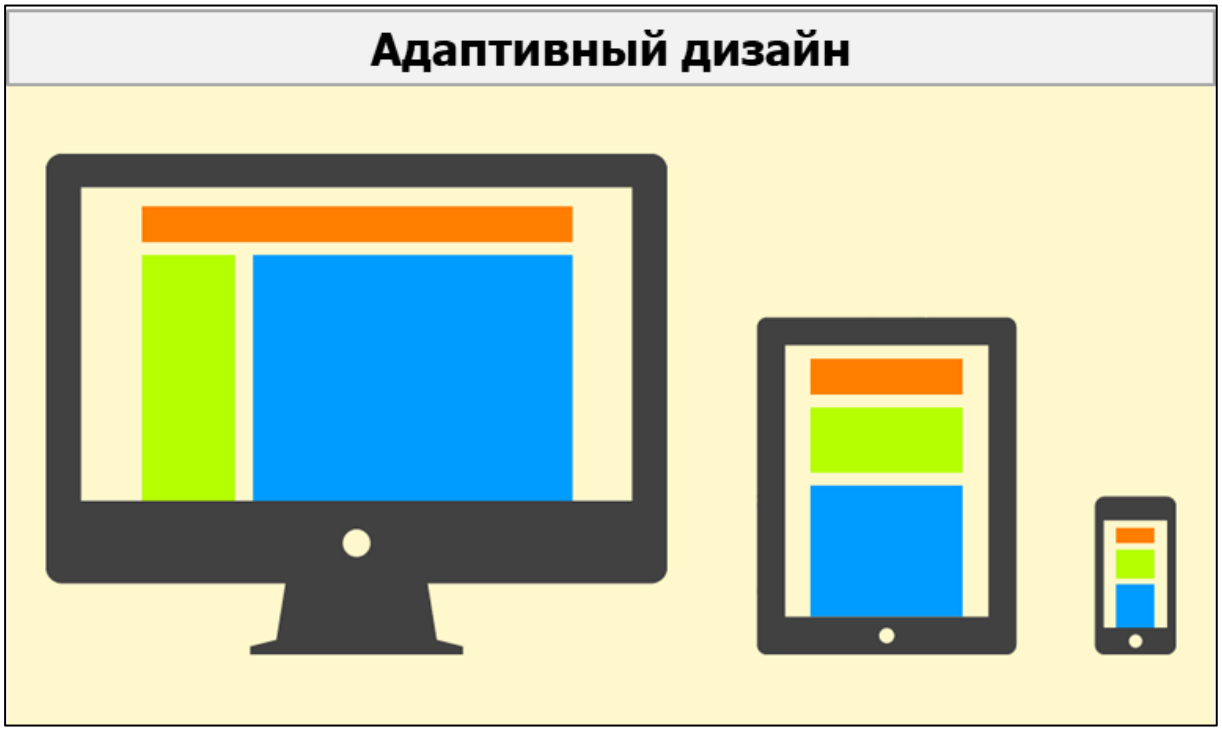


Рис. 1.234. Адаптивный дизайн предполагает работу с различными версиями сайта, ориентированные под разные устройства. Устройство автоматически определяется с помощью используемых скриптов

Этапы верстки

Общие этапы

Этап 1. Разработка чернового варианта макета

Это первоначальный этап, на котором предполагается примерная концепция внешнего вида веб-сайта. Для создания макета можно использовать как специальные онлайн-редакторы, так и рисовать черновик вручную.

В макете схематично размечаются основные блоки сайта (шапка, боковые панели, область контента, подвал сайта) и некоторые основные элементы. Глубокой детализации в макете не требуется, однако можно отдельно прорисовать некоторые детали конкретных блоков.

Рис. 1.235. Схематичная зарисовка макета будущего сайта

Этап 2. Уточнение макета и создание дизайна

На следующем этапе осуществляется уточнение макета и детальная прорисовка внешнего вида веб-страниц. Обычно здесь требуется работа с графическими редакторами или программами, которые предназначены для проектирования прототипов и интерфейсов.

Внешний дизайн потребует вставки текстового и графического материала в основные блоки (логотип, иконки, надписи в меню и гиперссылках на основные разделы сайта и прочее).

Особое внимание необходимо обратить на типографику: используемые шрифты, их размер, начертание. Если требуются специфические шрифты, то необходимо их установить на компьютер, чтобы они стали доступны в графических редакторах.

Мультимедийные файлы следует изначально группировать по каталогам, чтобы проект имел модульную структуру.

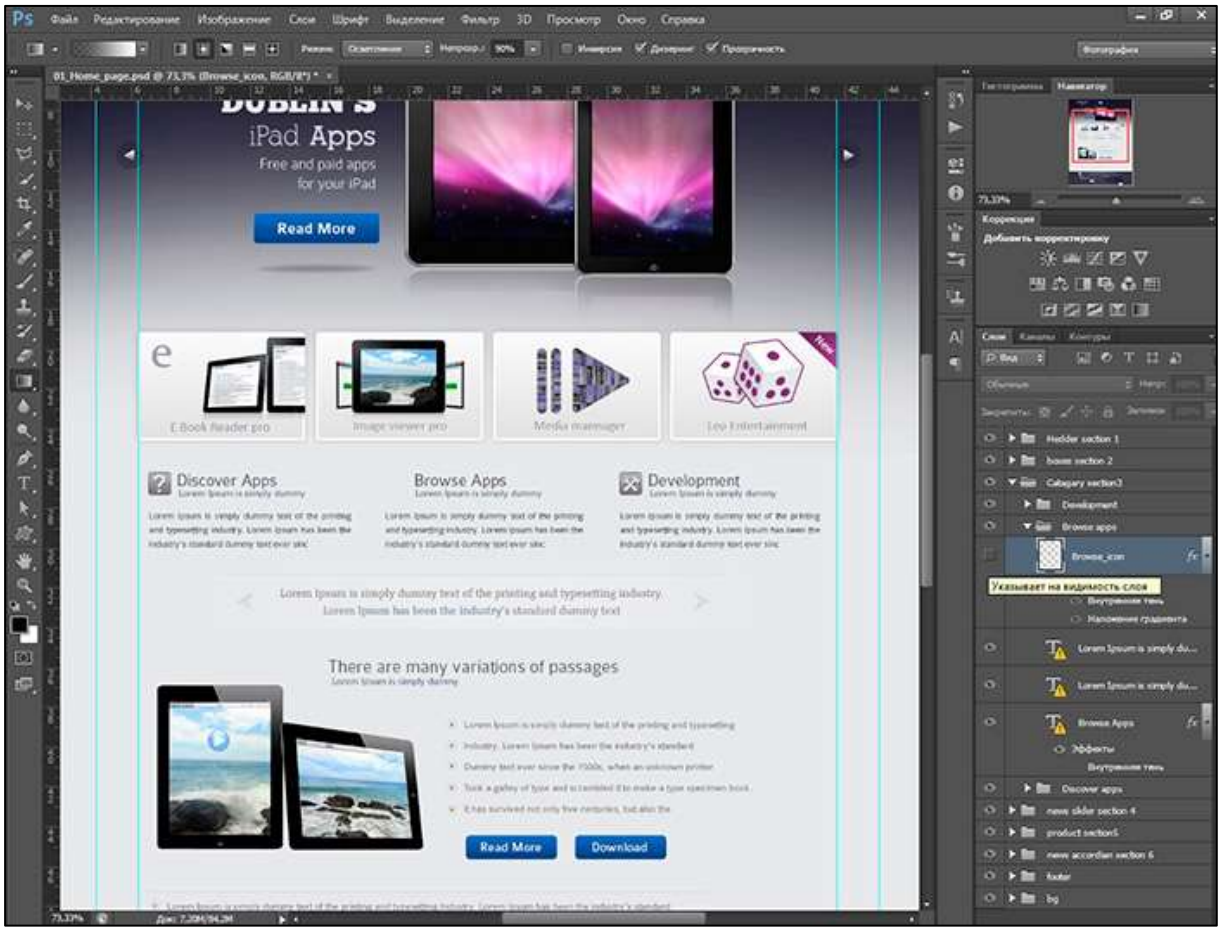


Рис. 1.236. Создание дизайна интерфейса веб-сайта в Adobe Photoshop

Этап 3. Вёрстка HTML-разметки

Важный этап, на котором осуществляется перенос графического представления дизайна в HTML и CSS код. Здесь важно учитывать структуру веб-сайта и связи между отдельными страницами. Как и в случае мультимедийных элементов, следует разбивать файлы по отдельным каталогам.

Корневую веб-страницу принято называть *index.html*. Остальные ссылки выстраиваются относительно ее положения (в зависимости от выбранного типа адресации).

Грамотная структура разметки предполагает, что:

- стили подключаются внешними файлами с использованием тега `<link>`;
- скрипты на JavaScript аналогично подключаются внешним образом.

Разделение разметки, стилей и скриптов повышает гибкость управления структурой веб-сайта.

Обычно верстка ведется по методу «сверху вниз», т.е. от реализации каркаса до последовательной (рекурсивной) детализации вложенных блоков и элементов. Верстальщик описывает HTML-структуру, параллельно указывая тегам:

- CSS-классы, которые в дальнейшем будут использоваться для стилизации этих блоков и элементов;
- идентификаторы, с помощью которых JavaScript может получать ссылку на элемент и управлять его свойствами.

Важно придерживаться определенных правил именования классов и идентификаторов, принятых в компании или сообществе разработчиков (например, по методологии БЭМ, см. ниже).

Чтобы ускорить процесс верстки структуры, используются возможности IDE или текстовых редакторов. Обычно наиболее мощным в этом плане является плагин Emmet, который на базе коротких аббревиатур способен автоматически развертывать большие объемы HTML-разметки.

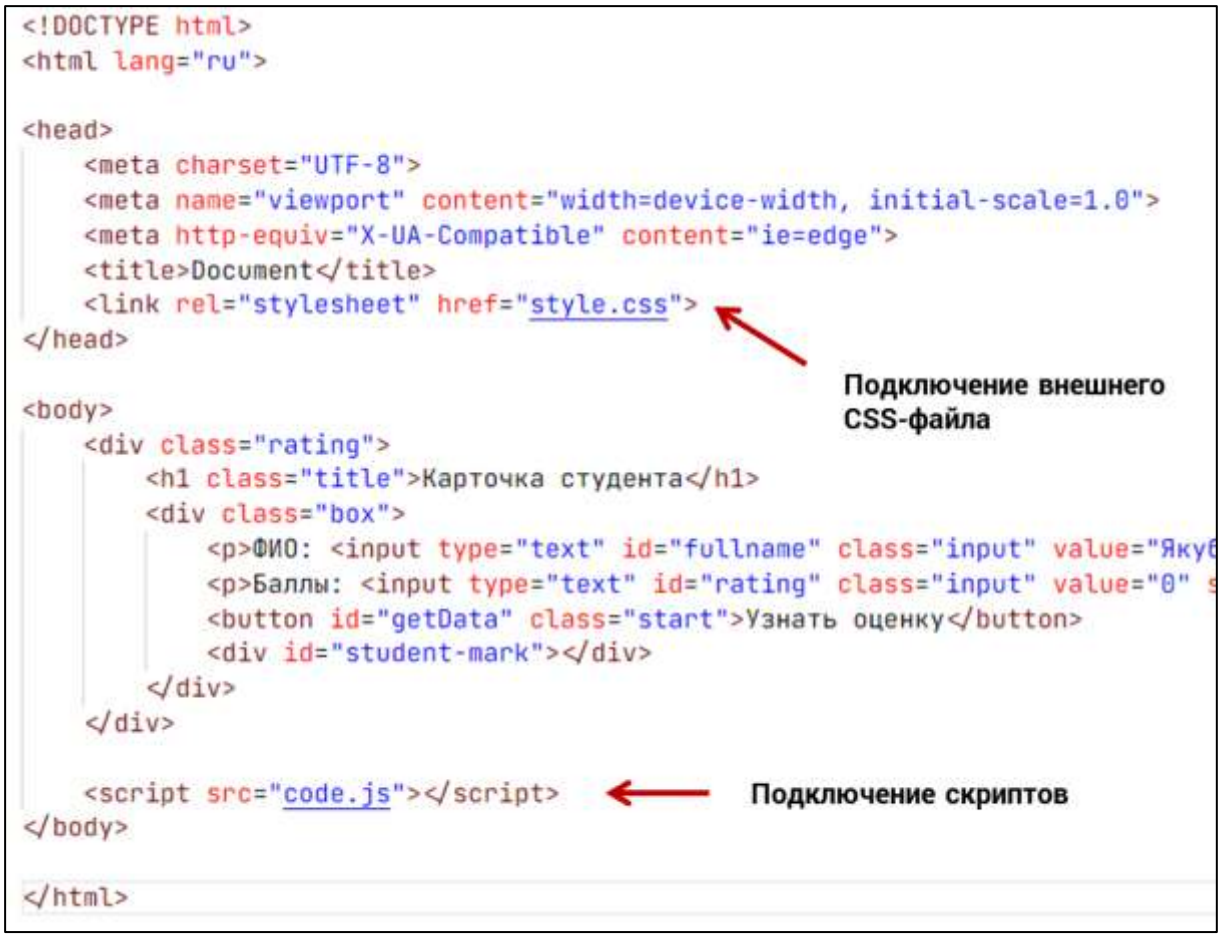


Рис. 1.237. Отделяем разметку от стилей и скриптов

Рис. 1.238. Плагин Emmet предназначен для быстрой вертски благодаря использованию альтернативного синтаксиса. Поддерживается многими IDE и текстовыми редакторами для веб-разработки

Этап 4. CSS-стилизация

После формирования HTML-структуры осуществляется описание стилей блоков и элементов. На практике обычно процесс описания стилей идет параллельно с разметкой, поскольку достаточно трудно задать структуру всего документа, не имея перед глазами результат его предварительного форматирования.

С другой стороны, разметку и стили удобно разбивать в редакторе на отдельные колонки, внося постепенные правки в код и просматривая полученный результат в окне браузера.

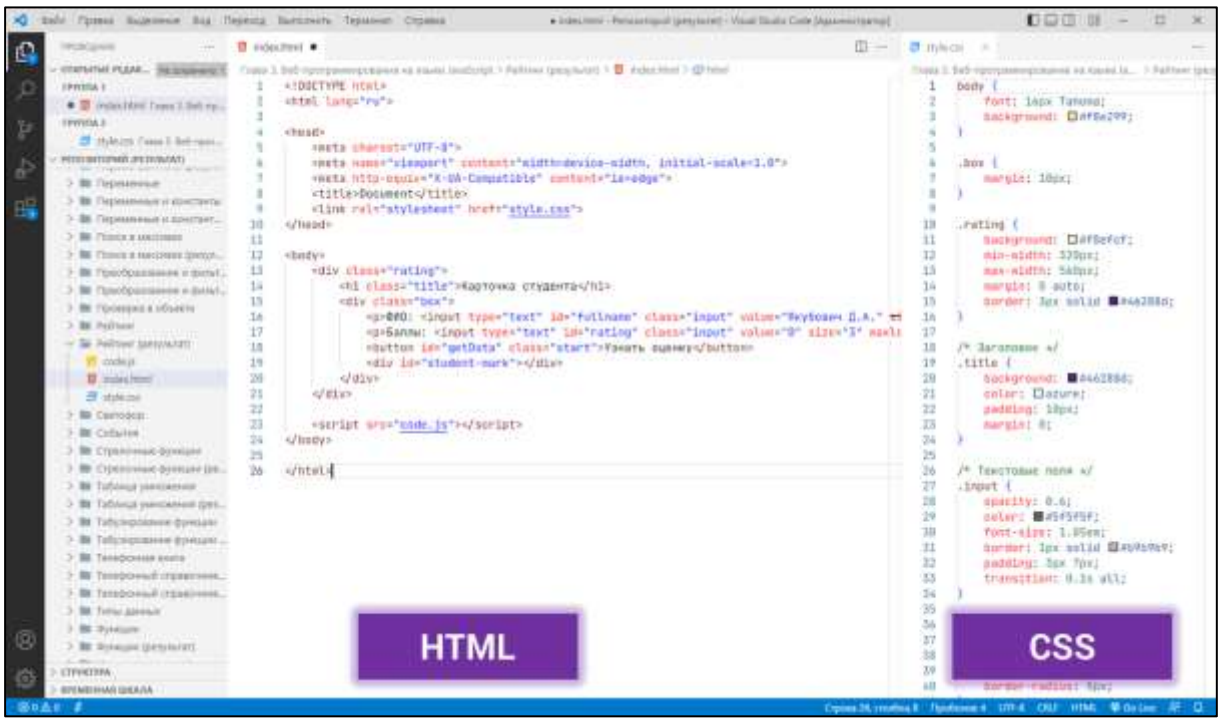


Рис. 1.239. Параллельная работа с HTML и CSS

Как и в случае HTML-разметки, стилизацию следует осуществлять в нисходящем порядке: от основных блоков и элементов к вложенным. В начале задаются свойства блоков, которые отвечают за отзывчивый дизайн (например, разбивают разметку на «слои» и «колонки»). Далее настраивается типографика элементов, свойства фона, границ, полей, отступов, различных эффектов.

При этом каждый блок детализируется последовательно и целиком. Например, если оформляется блок навигации, то следует описать его оформление, оформление внутренних элементов, проработать детали и анимацию каждого.

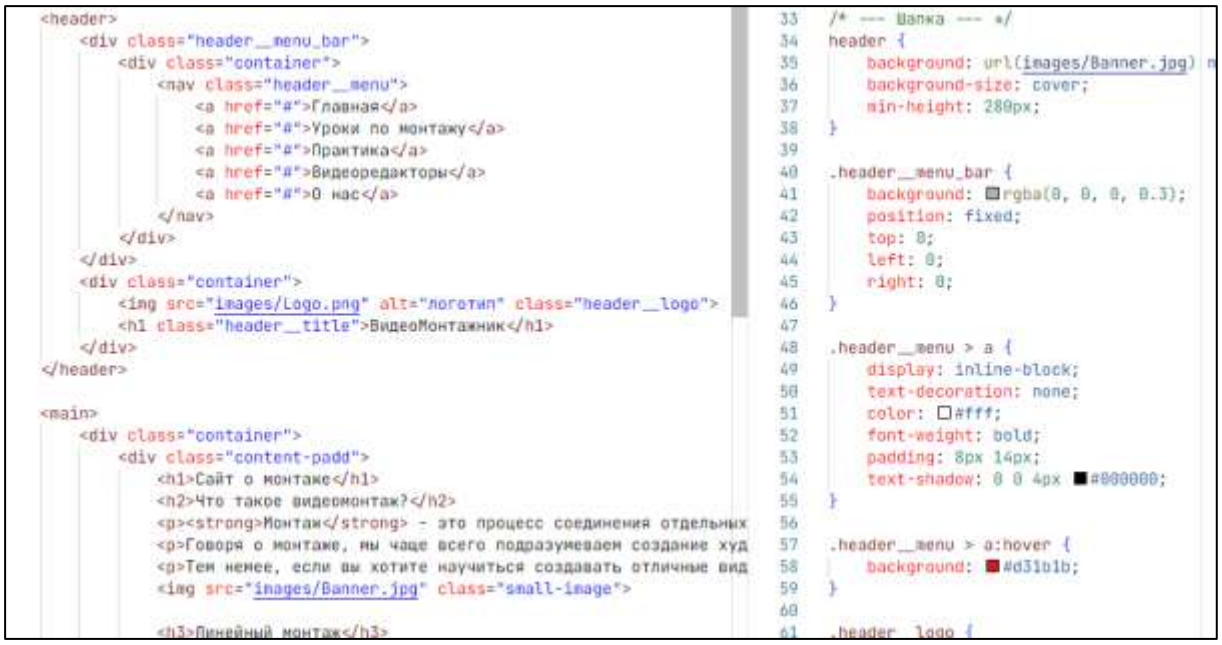


Рис. 1.240. Описание стилей CSS желательно выстраивать согласно разметке, чтобы упростить чтение и редактирование кода. В данном примере оформление элементов шапки сайта сгруппировано

Существенно ускорить верстку позволяет использование CSS-фреймворков, таких как Bootstrap или Bulma. Они реализуют множество классов и адаптивных сеток, которые достаточно лишь подключить в качестве классов и при необходимости добавить собственные элементы стилизации.

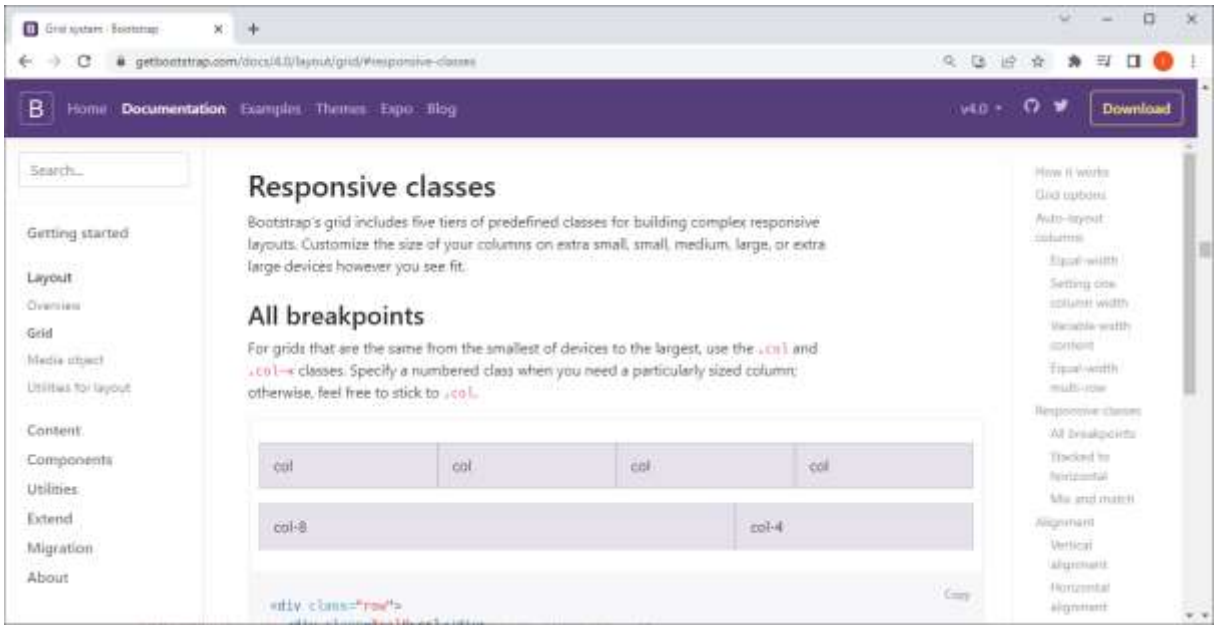


Рис. 1.241. Система колонок в Bootstrap

Этап 5. Программирование функционала (DHTML)

HTML и CSS определяют только внешний вид веб-сайта. Однако если пользователю необходимо с ним взаимодействовать, то управляющие элементы (кнопки, текстовые поля, флажки, ползунки, элементы форм и прочее) должны быть запрограммированы. Подобное часто также называют *динамическим HTML* (*DHTML*).

Основным языком веб-разработки на стороне клиента (браузера) является JavaScript. Верстальщик должен:

- хорошо знать синтаксис этого языка, особенности наиболее актуальных версий, уметь работать с библиотеками;
- иметь опыт работы с одним из наиболее востребованных фреймворков: React, Vue, Angular, Node.js и т.п;
- понимать особенности взаимодействия с серверными технологиями, в частности – подготовки данных к отправке из форм, принципы HTTP, функционирование баз данных.

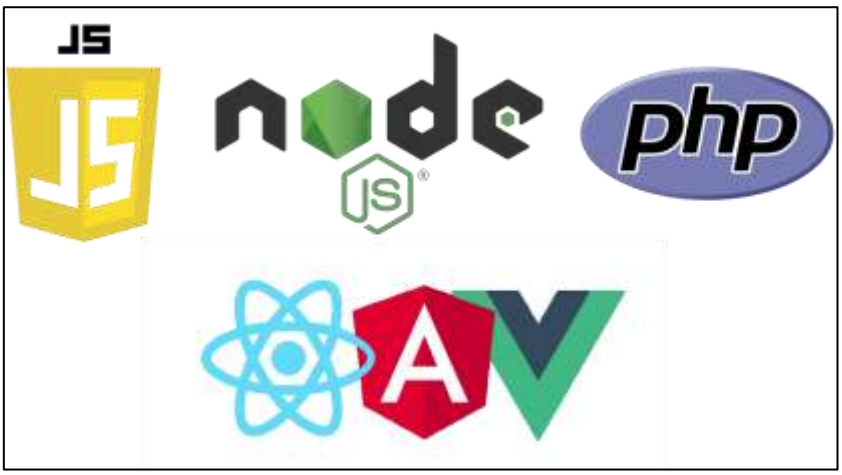


Рис. 1.242. Веб-разработка требует знания синтаксиса языков веб-программирования и связанных с ними фреймворков

Как и в случае стилей, в современной разработке скрипты рекомендуется подключать внешними файлами, чтобы разграничить разметку от функционала. Для этого используется тег `<script>` с атрибутом `src`, который указывает на подключаемый JS-скрипт:

```
<script src="code.js"></script>
```

Этап 6. Валидация

Завершающим этапом верстки является валидация разметки и проверка других файлов на наличие ошибок в синтаксисе. Здесь на первом этапе обычно достаточно возможностей текстового редактора, который анализирует код и выдает список ошибок и недочетов:

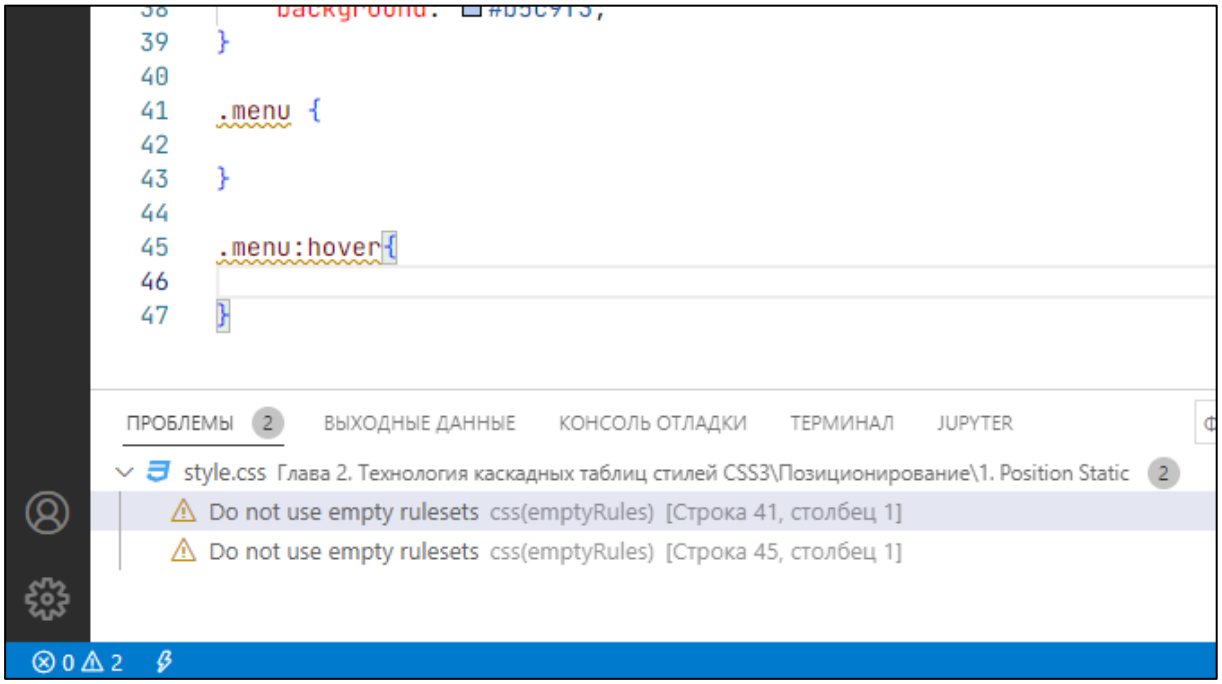


Рис. 1.243. Visual Studio Code предупреждает, что имеются два класса без реализации стилей (это не ошибка, однако следует внести исправления)

Комплексный анализ HTML-кода осуществляется с помощью валидаторов, в частности https://validator.w3.org/.

Верстка в рамках группы разработчиков

1. Верстальщик получает от дизайнера утверждённый дизайн-макет страницы.
2. Макет делится на линии (*полосы*) – «этажи». Каждый «этаж» анализируется отдельно и разбивается на прямоугольные блоки — *колонки*.
3. Разбиение рекурсивно повторяется, пока не реализована каждая компонента.
4. Готовая страница проверяется на кросс-платформенность.
5. Критические исправления вносятся в документ и проверка повторяется с самого начала.

Базовые правила

1. Вёрстка должна быть кроссбраузерной для наиболее популярных браузеров: Mozilla Firefox, Opera, Safari, Google Chrome, Internet Explorer / Edge. Сайт должен быть протестирован на разных разрешениях мониторов (в т.ч. мобильных устройств), начиная от 1024x768.

Рис. 1.244. Верстка сайта должна стабильно отображаться на ведущих браузерах

2. Внешний вид страницы должен максимально соответствовать дизайну макета: допускается отклонение в пределах не более 3-6 пикселей.

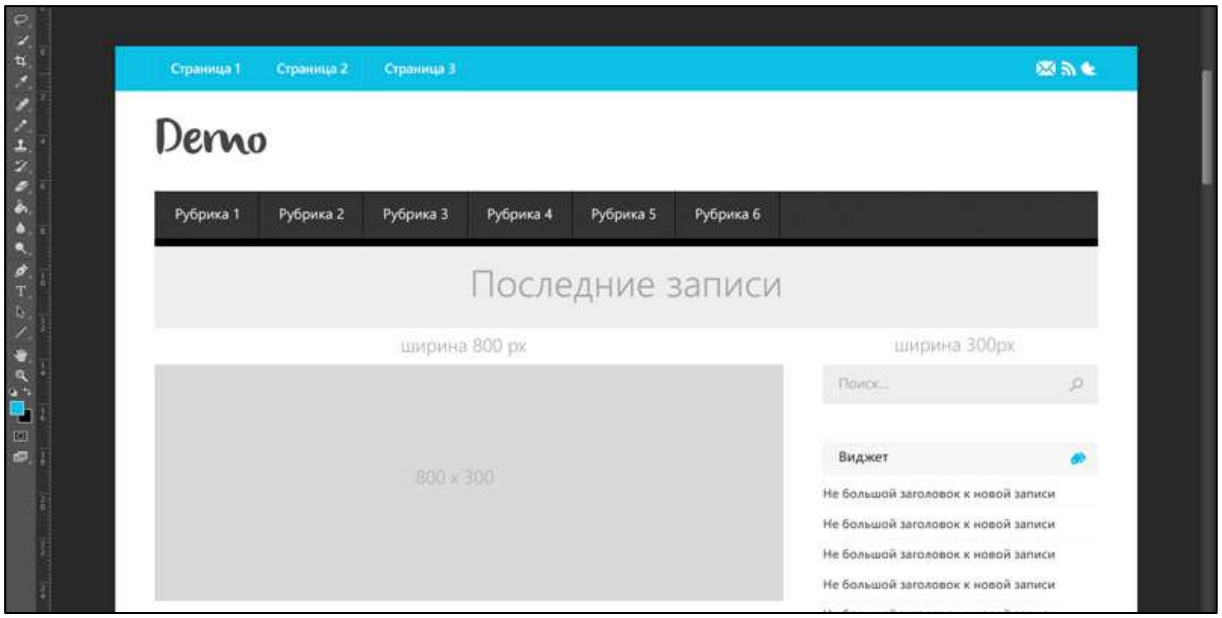


Рис. 1.245. Дизайн переносится в разметку с высокой точностью детализации

3. Вёрстка абсолютно всех страниц сайта должна пройти валидацию. В коде необходимо исключить все синтаксические ошибки.

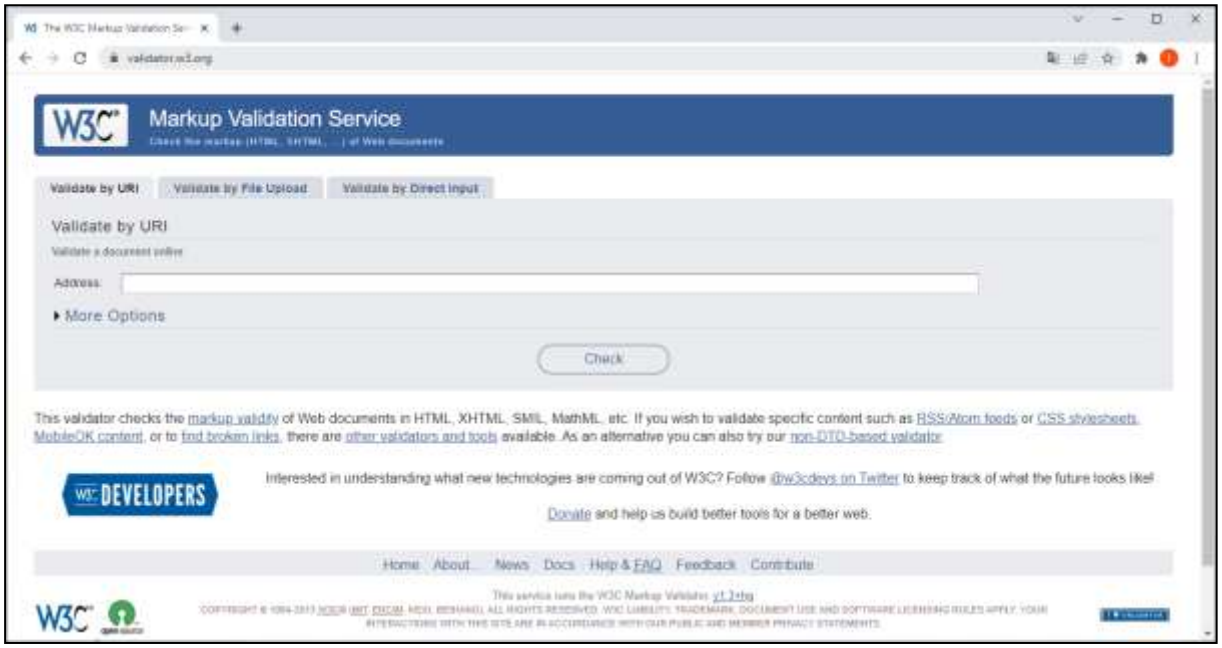


Рис. 1.246. Валидация кода на сайте консорциума W3C

4. CSS стили вынесятся в отдельный файл(ы).

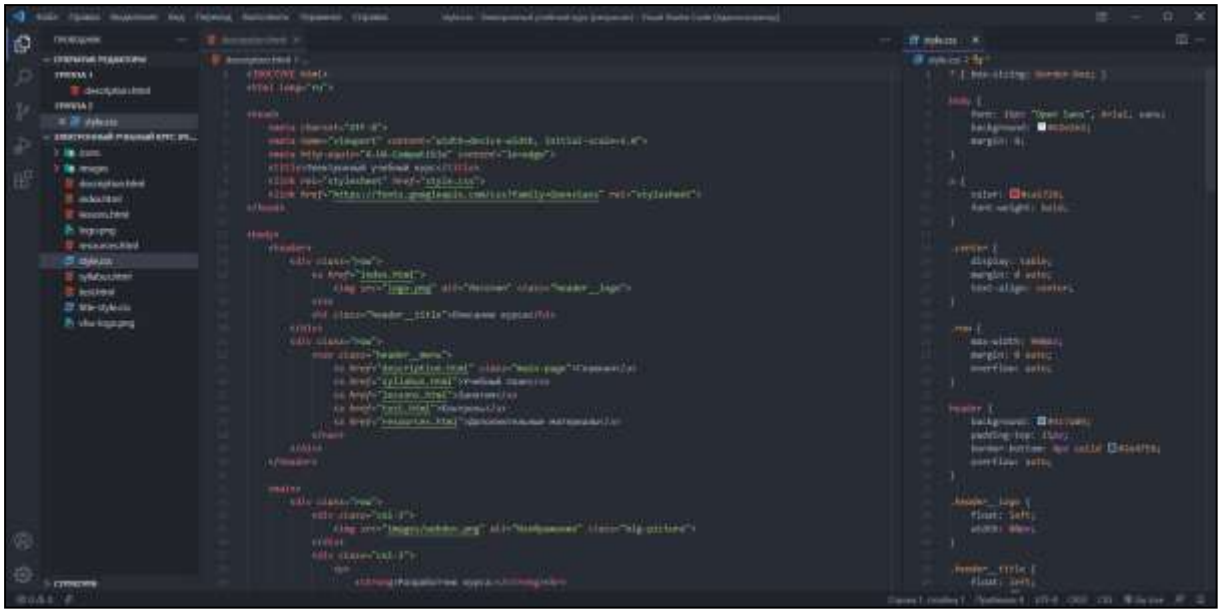
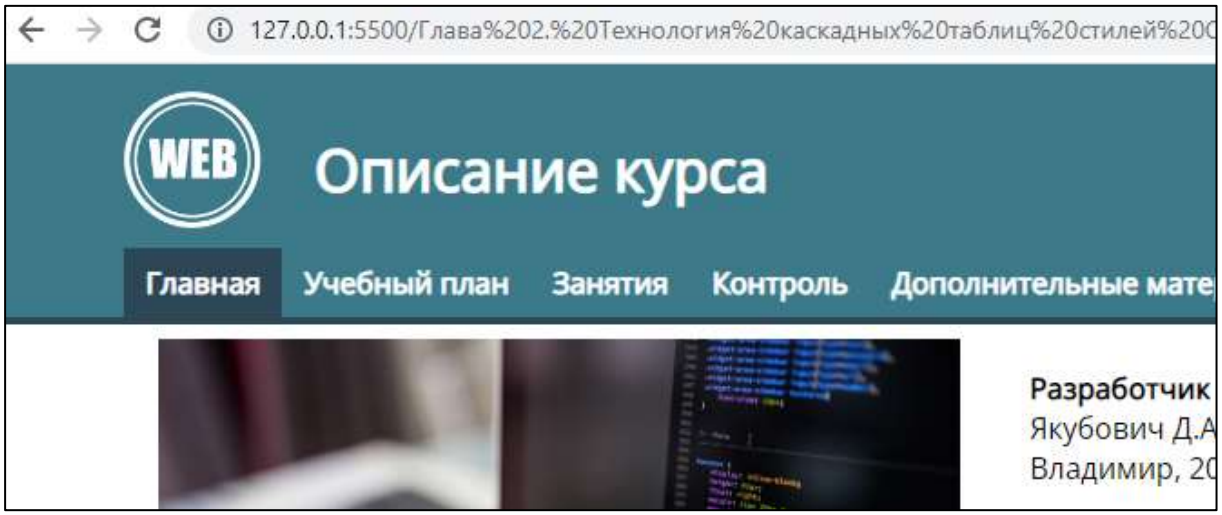


Рис. 1.247. Разграничение разметки и стилей

5. Логотип веб-сайта делают ссылкой на главную страницу.




```
<a href="index.html">
    <img src="logo.png" alt="Логотип" class="header__logo">
</a>
```

Рис. 1.248. Логотип помещен в гиперссылку на главную страницу

6. В HTML коде допустимо только присутствие идентификаторов и классов.

```
<header>
    <h1 class="header__title" id="title">Базы данных</h1>
    <nav class="header__menu">
        <a href="#">MS Access</a>
        <a href="#">MS SQL Server</a>
        <a href="#">MySQL</a>
        <a href="#">PHPMyAdmin</a>
        <a href="#">MySQL Workbench</a>
        <a href="#" class="about-ref">О ресурсе</a>
    </nav>
</header>
```

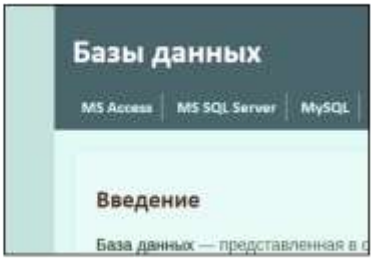


Рис. 1.249. Теги содержат классы, которые отвечают за оформление, и идентификаторы, чтобы можно было обратиться к элементу в скриптах

7. В таблицах стилей настоятельно рекомендуется использовать одинаковые единицы измерения для всех величин (шрифт, размеры блоков, поля, отступы, рамки и др.).

Разные единицы измерения

```
.container {
    display: block;
    background: #E9FBF7;
    border: 1px solid #597e75;
    border-radius: 5px;
    margin: 20px;
    padding: 15px 25px;
}
```

Одинаковые единицы измерения

```
.container {
    display: block;
    background: #E9FBF7;
    border: 0.063em solid #597e75;
    border-radius: 0.313em;
    margin: 1.25em;
    padding: 0.938em 1.563em;
}
```

Рис. 1.250. Единый подход к использованию единиц измерения величин

8. Надписи на кнопках должны быть написаны на одном языке, а также либо прописными буквами, либо начинаться с заглавной.

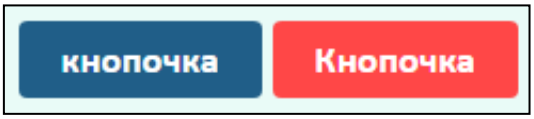


Рис. 1.251. Надписи на кнопках

9. Кнопки должны иметь стандартное оформление и быть либо графическими элементами, либо стилизованы с помощью таблиц стилей.

Рис. 1.252. Оформление кнопок

10. HTML и CSS код должен быть минимизирован. Избегайте лишних классов и идентификаторов, используя возможности наследования и каскадирования CSS-свойств. Для выработки корпоративного стиля разработки также следует изучить одну из методологий верстки, например – БЭМ.

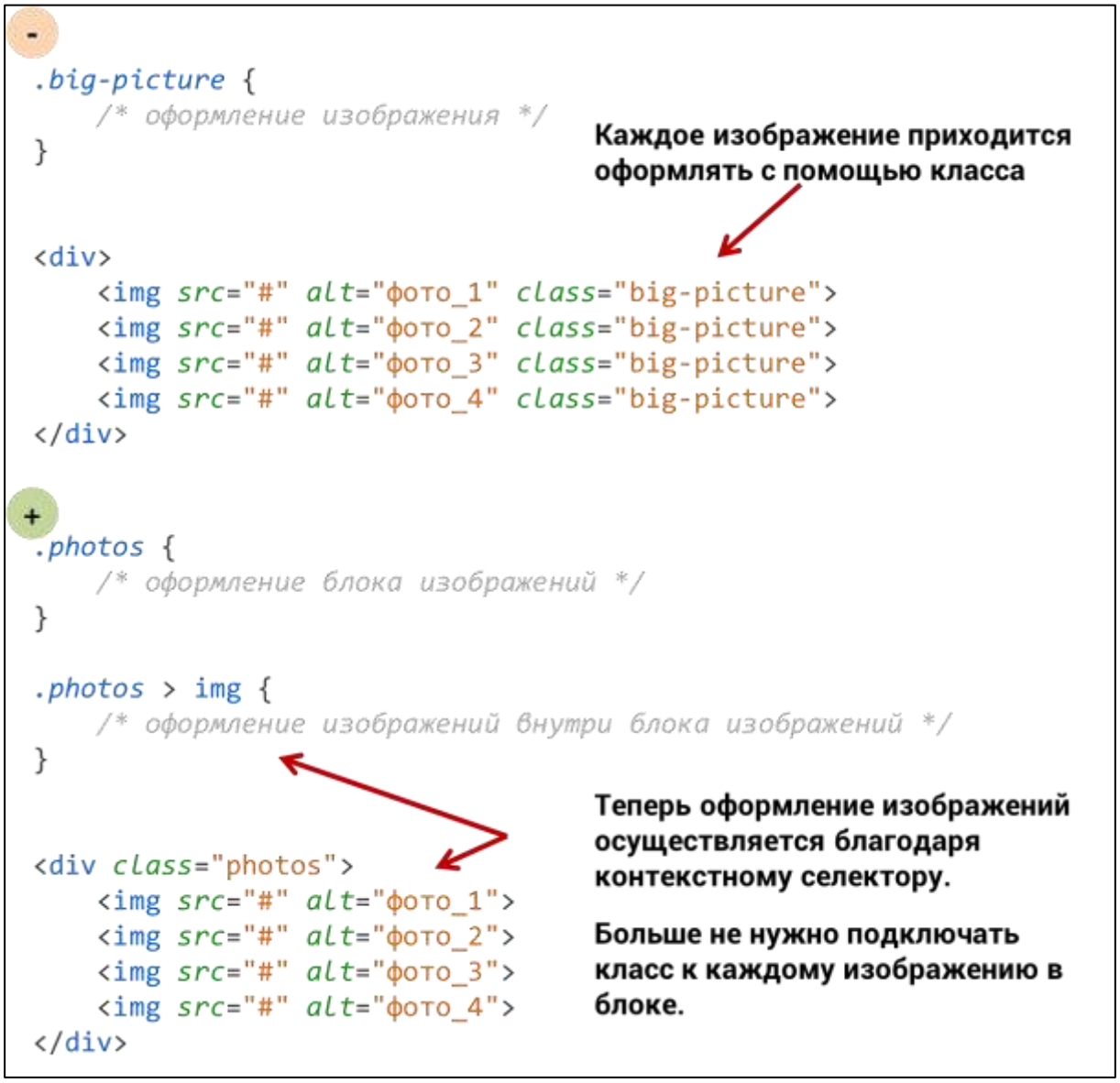
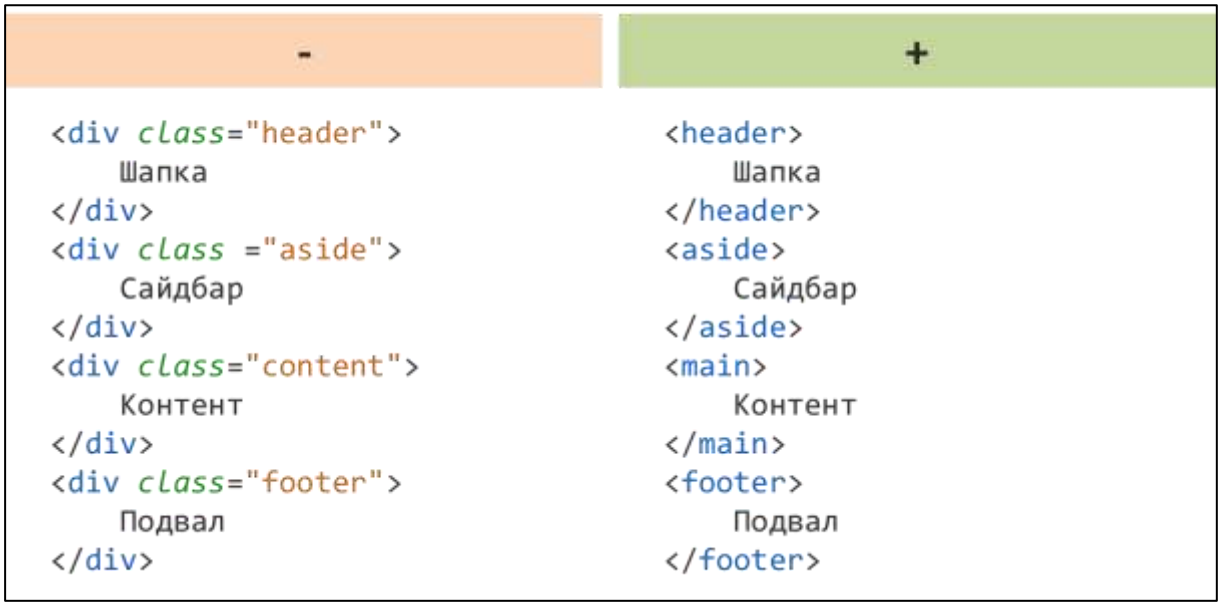


Рис. 1.253. Старайтесь писать как можно более лаконичный и прозрачный код

11. Работайте с тегами семантической разметки там, где это возможно. Используйте по максимуму новые теги спецификации HTML5.



-

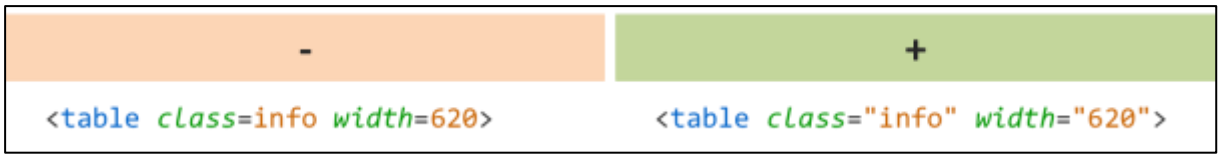
```
<div class="header">
    Шапка
</div>
<div class ="aside">
    Сайдбар
</div>
<div class="content">
    Контент
</div>
<div class="footer">
    Подвал
</div>
```

+

```
<header>
    Шапка
</header>
<aside>
    Сайдбар
</aside>
<main>
    Контент
</main>
<footer>
    Подвал
</footer>
```

Рис. 1.254. Семантические теги - первостепенны

12. Атрибуты всех тегов должны быть заключены в кавычки.



-

```
<table class=info width=620>
```

+

```
<table class="info" width="620">
```

Рис. 1.255. Следует строго соблюдать синтаксис описания тегов, даже если браузеры могут обработать страницу с недочетами в разметке

Тестирование верстки сайта

Следующий список можно использовать как чек-лист:

1. Проверка соответствия сверстанного веб-сайта с запланированным дизайном.
2. Проверка корректности работы в разных браузерах.
3. Просмотр веб-страниц в разных разрешениях и ширине окна браузера.
4. Проверка отзывчивости и/или адаптивности на разных устройствах.
5. Валидация HTML-разметки и CSS-стилей.
6. Анализ JavaScript-кода на наличие ошибок.

7. Анализ скорости загрузки страниц
8. Проверка корректности подключения шрифтов, в т.ч. работа в разных операционных системах.
9. Проверка функционирования управляющих элементов страниц.
10. Проверка работоспособности и доступности ссылок.
11. Проверка орфографии, грамматики, оформления текста.

1.8.2. Методология БЭМ

Особенности БЭМ

Понятие БЭМ

Определение

Методология БЭМ (Блок, Элемент, Модификатор) *– это универсальный компонентный подход к веб-разработке, предполагающий принцип разделения интерфейса на независимые блоки. БЭМ систематизирует подход к верстке интерфейсов любой сложности и позволяет повторно использовать код, избегая копирования-вставки.*

Методология БЭМ была разработана в компании Яндекс с целью формирования единого корпоративного подхода к разработке проектов. Однако она оказалось востребованной у разработчиков по всему миру. Это связано не только универсальным подходом к структуре проекта и оформлением кода, но и удобной интеграцией с JavaScript (многие другие подходы и соглашения верстки ограничиваются лишь ограничениями на HTML и CSS).

БЭМ использует специальное соглашение по именованию CSS-классов, что делает их максимально информативными, уникальными и позволяет многократно использовать компоненты. БЭМ упрощает коллективную разработку, масштабирование и поддержку веб-сайтов.

Достоинства методологии БЭМ

- Обеспечивает стабильный, предсказуемый и хорошо читаемый код.
- Позволяет повторно использовать верстку.
- Допускает безболезненную частичную замену верстки в проекте на уровне компонент.
- Позволяет легко переносить готовую верстку из одного проекта в другой.
- Сокращает время на отладку проекта.

Преимущества БЭМ перед другими методологиями

- БЭМ качественно разделяет разметку и функционал отдельных компонент, поэтому стили разных элементов получаются взаимозависимыми.
- БЭМ определяет универсальный подход к именованию классов, что позволяет создавать неограниченное число уникальных стилей.
- БЭМ эффективно организует самодокументирующийся код: в классе содержится информация об элементе.
- БЭМ разбивает проект на отдельные компоненты, которые могут быть сгруппированы с соблюдением обычной файловой структуры.

Пример разметки без БЭМ

Предварительно приведем пример разметки, которая не предусматривает работу с БЭМ.

Пусть в разметке присутствует форма поиска, описанная следующим образом (содержит заголовок, текстовое поле и кнопку):

```
<form class="search">
    <h2 class="title">Поиск</h2>
    <input class="input">
    <button class="button">Найти</button>
</form>
```

Описание классов этом случае зададим следующим образом:

```
.search { }
.search .title { }
.search .input { }
.search .button { }
```

Здесь при описании CSS-стилей мы использовали контекстное вложение классов, чтобы обеспечить однозначный приоритет стилей для каждого элемента.

Даже на таком простом примере можно выявить ряд серьезных недостатков обычного подхода к разметке:

- На странице может быть несколько форм, имеющих как общие, так и собственные стили оформления. Это потребует добавлять элементам несколько классов.
- В каждой форме может быть свой заголовок, элемент ввода, кнопка или иные повторяющиеся элементы. Когда элементов в документе много, возникнет проблема с именованием классов.
- Для оформления разных форм и их элементов потребуется многократно описывать классы, используя правила каскадирования.

Компоненты БЭМ

Блок

Определение

***Блок** представляет собой функционально автономный компонент веб-страницы, который можно использовать повторно. В HTML-разметке блоки определяются атрибутом `class`. Блок может содержать элементы и другие блоки.*

Блок может включать в себя шаблоны, CSS-стили, JavaScript-код, документацию в формате XML, а также другие необходимые в его работе технологии.

Блоки связывают в себе связанные по смыслу элементы документа. В силу независимости блоки можно перемещать в рамках веб-страницы или проекта в целом. При этом их компоненты продолжат корректную работу и сохранят внешний вид, а вносить изменения в стили и скрипты не потребуется.

Блоки обычно создаются для фрагментов, которые могут использоваться самостоятельно.

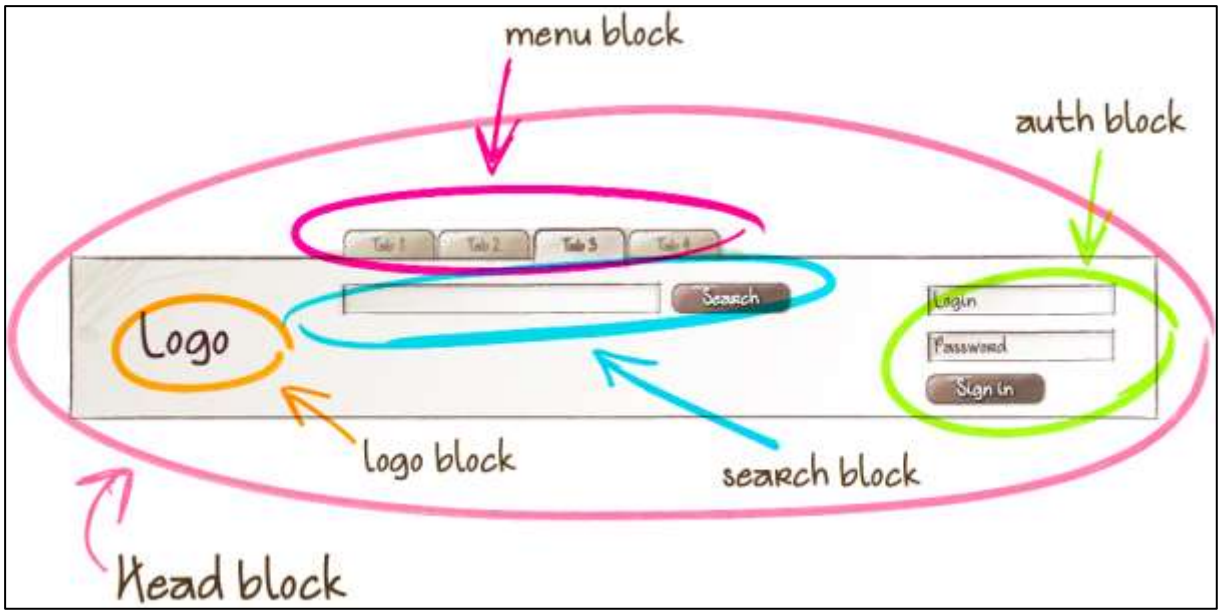


Рис. 1.256. Блоки в разметке

Блоки в БЭМ имеют ряд особенностей:

- название блока характеризует смысл, а не состояние объекта;
- блок не влияет на свое окружение, т. е. блоку не следует задавать внешнюю геометрию (поля, отступы, границы) и позиционирование.

В приведенным выше примере разметки формы поиска в качестве блока выступает сама форма поиска:

```
<form class="search">
    <h2 class="title">Поиск</h2>
    <input class="input">
    <button class="button">Найти</button>
</form>
```

Элемент

Определение

Элемент *является составной частью блока, которая не должна использоваться в отрыве от него. Элементы всегда привязаны к какому-либо блоку и только к одному. Также один элемент может входить в состав другого.*

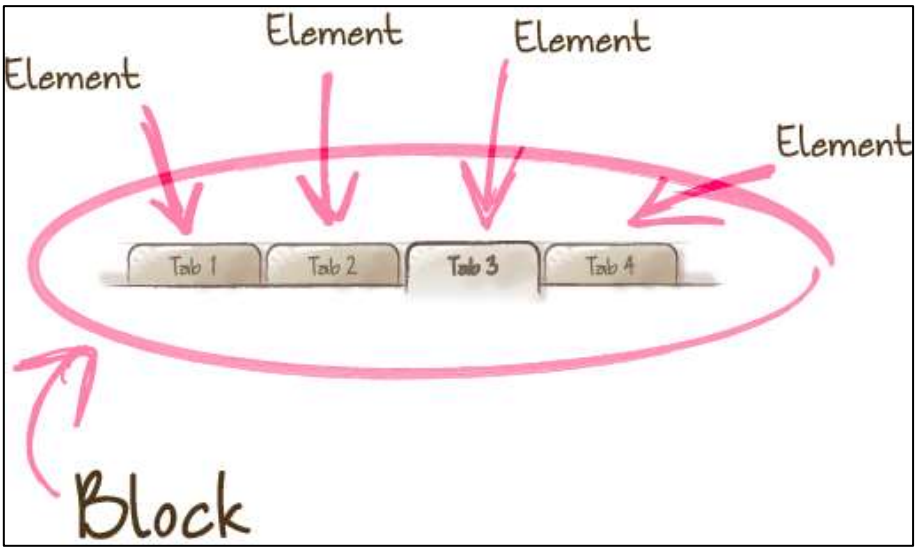


Рис. 1.257. Элементы привязаны к определенному блоку

Элементы относятся к необязательным компонентам: иногда можно ограничиться малыми блоками без необходимости вложения в них элементов.

Для работы с элементами задаются ряд правил на его имя:

- название элемента характеризует смысл или роль, а не его состояние.
- название элемента задается символом двойного подчеркивания, где сначала указывается имя его блока:

```
имя_блока__имя_элемента
```

Модифицируем разметку формы поиска с учетом обозначенных правил:

```
<form class="search">
    <h2 class="search__title">Поиск</h2>
    <input class="search__input">
    <button class="search__button">Найти</button>
</form>
```

В приведенной разметке по названиям классов четко прослеживается, что форма является блоком, а заголовок, поле ввода и кнопка – элементы этой формы.

Изменения затронут и описание селекторов классов:

```
.search { }
.search__title { }
.search__input { }
.search__button { }
```

Здесь прослеживается важное преимущество БЭМ: удалось избавиться от контекстного вложения селекторов и перейти к «плоскому» дизайну классов. Стили элементов будут независимы, при этом в названиях классов разработчик видит смысловую связь между ними (привязку к форме поиска).

Модификатор

Определение

Модификатор *представляет собой сущность, которая определяет внешний вид, состояние или поведение блока, либо элемента.*

Модификатор позволяет задать блоку и элементу разное оформление: блок или элемент просто подключает модификатор как CSS-класс. В некотором смысле модификаторы аналогичны атрибутам тегов в HTML.

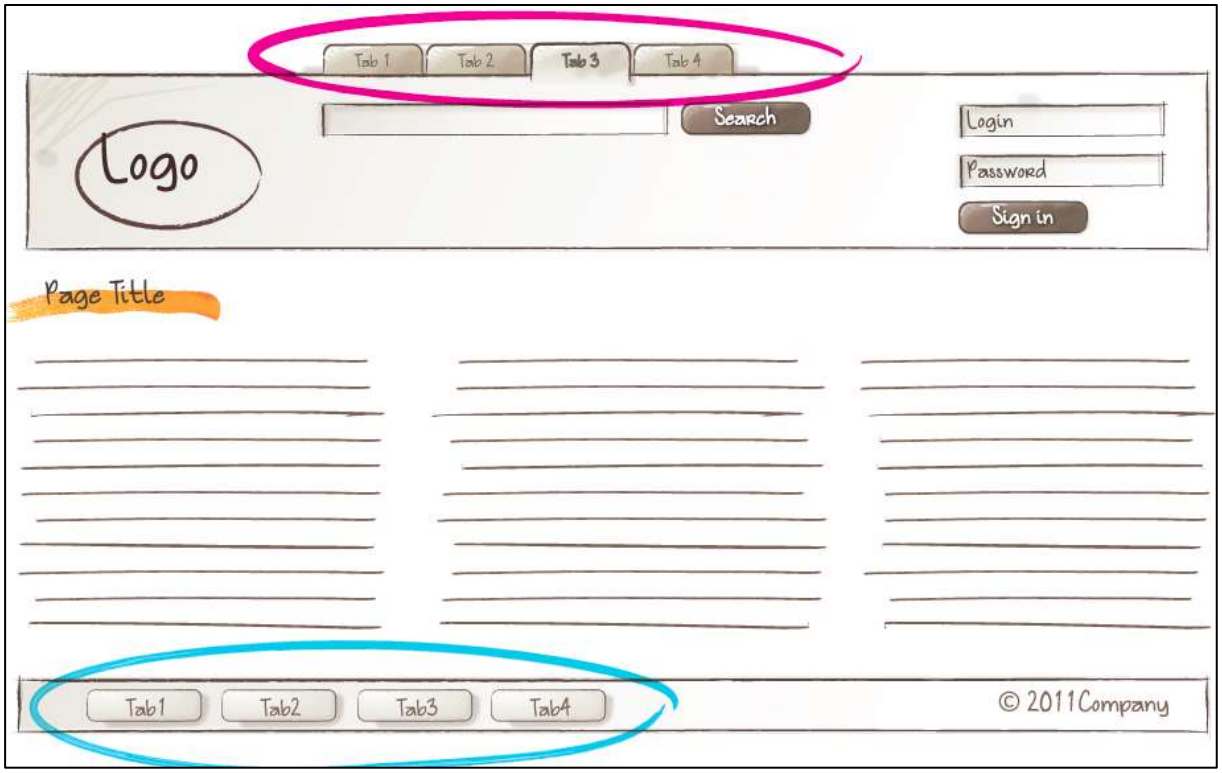


Рис. 1.258. Одинаковые модификаторы могут использоваться в разных частях разметки, придавая элементам или блокам некоторые общие характеристики

Модификаторы делятся на две категории:

- *логические* отмечают факт наличия какого-либо свойства;
- модификаторы в формате *ключ-значение*, когда имя отражает формат или преобразование элемента.

Элементы и блоки могут содержать любое число модификаторов, которые управляют их внешним видом, состоянием и расположением.

Для модификаторов используют следующие соглашения по именованию:

- название модификатора характеризует внешний вид, состояние и поведение блока или элемента;
- название модификатора отделяется от имени блока или элемента символом подчеркивания:

```
имя-блока_имя-модификатора
имя-блока__имя-элемента_имя-модификатора
```

Перепишем разметку формы, подключив к блоку и элементам классы оформления:

```
<form class="search search_dark-theme">
    <h2 class="search__title
search__title_light">Поиск</h2>
    <input class="search__input search__input_size-m">
    <button class="search__button search__button_size-
m">Найти</button>
</form>
```

Видно, что плоский дизайн классов, отвечающих за описание модификаторов, также сохраняется:

```
.search_dark-theme { }
.search__title_light { }
.search__input_size-m { }
.search__button_size-m { }
```

Файловая структура проекта

Важным достоинством БЭМ является системный подход к построению структуры проекта. Каждый компонент в БЭМ можно разбить на отдельные части, которые отвечают за его структуру, стиль оформления и функционал.

Подобный подход позволяет:

- выборочно подключать компоненты;
- вести независимую разработку компонентов;
- переносить блоки внутри проекта или между разными проектами.

Каждый блок в разметке можно описывать в отдельной директории; в поддиректориях располагаются файлы элементов и модификаторы, а также другие необходимые файлы: изображения, видео, шрифты, скрипты.

```
<form class="search search_dark-theme">
    <h2 class="search__title search__title_light">Поиск</h2>
    <input class="search__input search__input_size-m">
    <button class="search__button search__input_button-m">Найти</button>
</form>
```

```
search/
    search.css
    search.js
    __title/
        search__title.css
        search__title.js
    __input/
        search__input.css
        search__input.js
    __button/
        search__button.css
        search__button.js
    . . .
. . .
_dark-theme/
    search_dark-theme.css
    search_dark-theme.js
_light/
    search__title_light.css
    search__title_light.js
_size-m/
    search__input_size-m.css
    search__input_size-m.js
    search__button_size-m.css
    search__botton_size-m.js
```

Рис. 1.259. Разбиение описания формы на отдельные каталоги и подкаталоги

Это полезно знать!

В этом параграфе мы раскрыли лишь базовую часть методологии БЭМ. Ее изучение требует времени и определенного опыта в верстке веб-страниц.
Несмотря на кажущуюся избыточность, методология БЭМ существенно улучшает эргономику проекта. Однако на начальном этапе изучения frontend-разработки можно ограничиться лишь правилами именования классов.

1.8.3. Фреймворки для быстрой верстки

Возможности

В современных условиях верстка даже небольших проектов «с нуля» является достаточно трудоемким процессом. Поэтому в разработке обычно использует фреймворки, предоставляющие упрощенный подход к верстке структуры и оформлению элементов.

Фреймворки на базе CSS и JavaScript предоставляют следующие возможности:

- упрощают и ускоряют разработку веб-сайтов;
- реализуют классы для разметки адаптивных сеток и колонок, поддерживают фиксированные, резиновые или адаптивные макеты;
- настраивают элементы типографики страницы;
- содержат готовые классы для разметки и форматирования элементов форм, меню, навигации т.д;
- предоставляют JavaScript-модули для обработки данных форм и анимации;
- учитывают современные тенденции в веб-дизайне.

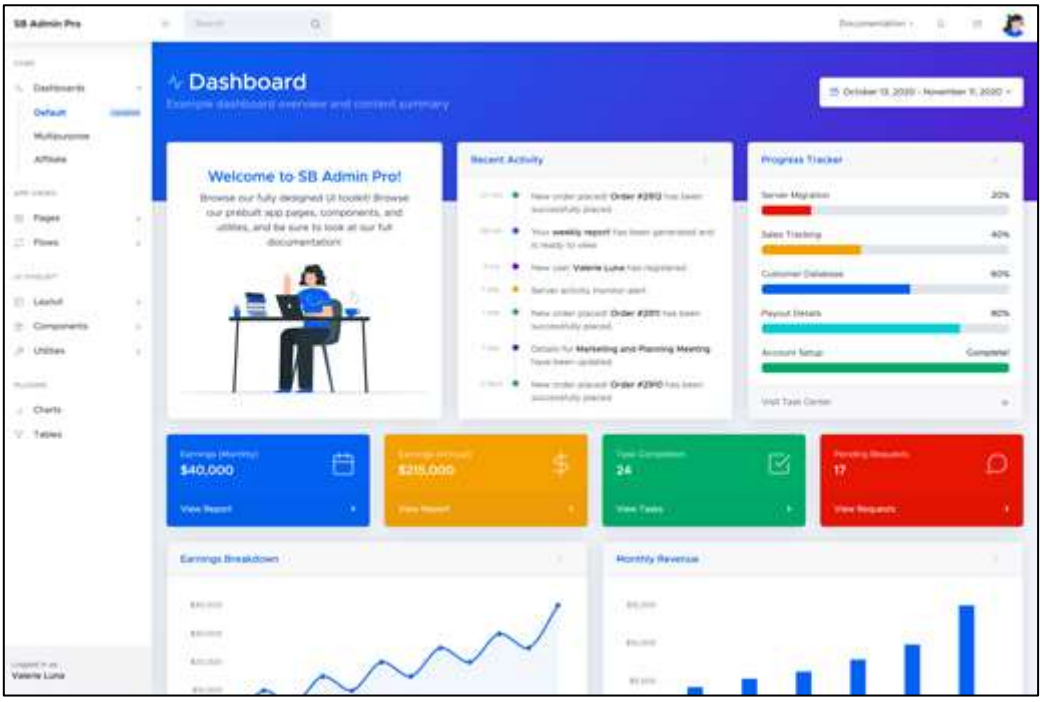


Рис. 1.260. Фреймворки ускоряю верстку сложных проектов

Примеры фреймворков

Bootstrap

Bootstrap является одним из самых востребованных открытых CSS-фреймворков, который предоставляет мощные инструменты для разработки адаптивных сайтов и мобильных сервисов.

На момент написания пособия последней актуальной версией является Bootstrap 5.

Возможности

- Гибкий инструмент для настройки адаптивных сеток и изображений, работы с flex-моделью.
- Реализация классов для оформления компонентов: форм, панелей навигации, интерактивных списков меню, индикаторов и прочее.
- Оформление всплывающих окон.
- Поддержка JavaScript.
- Работа с шаблонами.
- Подробная документация.
- Активное сообщество разработчиков.

Недостатки

- Сложен для изучения новичкам.
- Большие размеры загружаемых файлов, что особенно нерационально для небольших проектов, требующих малого трафика и большой скорости загрузки страниц.

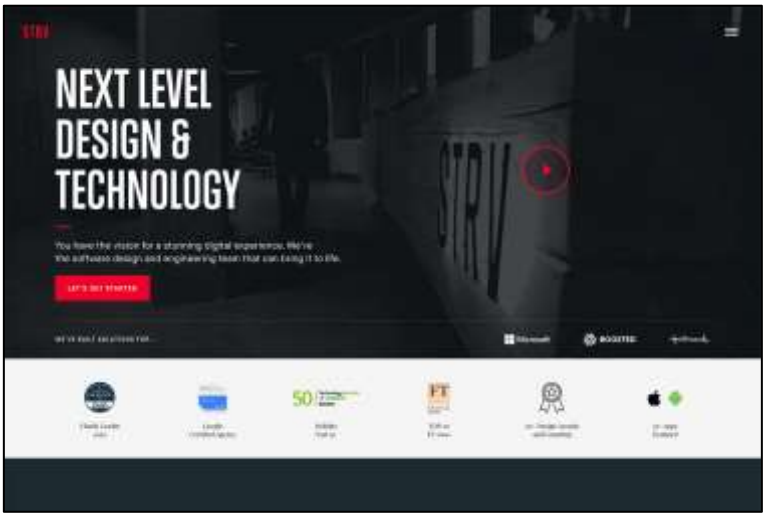


Рис. 1.261. Пример сайта, разработанного с помощью Bootsrap

Bulma

Bulma является бесплатным CSS-фреймворком, построенном на базе реализованных классов. Фреймворк появился в 2016 году и стал одним из наиболее популярных в frontend-разработке.

Bulma использует модель гибкой компоновки блоков Flexbox, которая упрощает создание адаптивного дизайна.

Возможности

- Предоставляет инструменты для разметки адаптивных сеток и обеспечения адаптивного дизайна.
- Имеет большой набор классов с настроенной CSS-типографикой.
- Реализует настройку компонентов форм.
- Работает с JavaScript-компонентами и плагинами.
- Поддерживает синтаксис SASS и SCSS.
- Можно комбинировать с другими фреймворками (Angular, React).
- Имеет модульную структуру.

Недостатки

- Как один из крупнейших фреймворков, требует изучения.
- При активном использовании возможностей фреймворка интерфейс сайта получается типовым.

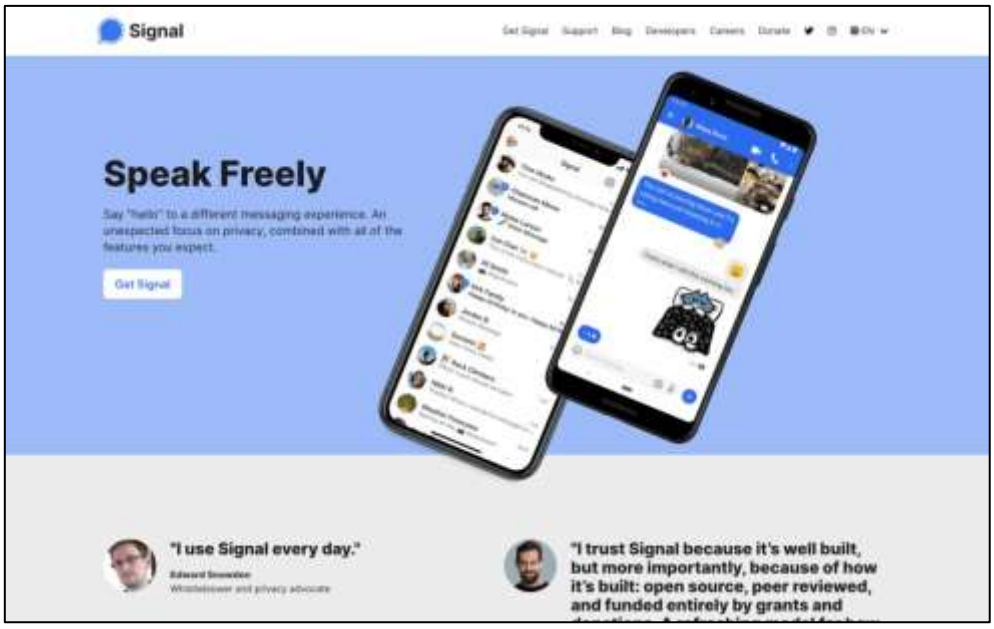


Рис. 1.262. Пример сайта, разработанного с помощью Bulma

Foundation

Foundation – это фреймворк для адаптивной верстки веб-интерфейсов, позволяющий работать с адаптивной сеткой и компонентами пользовательского интерфейса. Фреймворк имеет достаточно обширные возможности, близкие к Bootstrap, Tailwind и Bulma, однако в настоящее время не столь распространен и продолжает завоевывать популярность среди программистов и верстальщиков.

Foundation определяет модульную структуру и работает со стилями SASS. Основой является адаптивная сетка в 940 пикселей.

Foundation является удобным инструментом для опытных разработчиков, которым требуется свобода в реализации уникальных идей.

Возможности

- Множество инструментов верстки и гибкость управления.
- Хорошо оптимизирован.
- Адаптивная сетка, панели навигации, разные типы контейнеров, HTML-шаблоны.
- Технология Foundation for Emails для создания адаптивных шаблонов электронной почты.
- Переходы и анимация на базе Motion UI.

Недостатки

- Сложность освоения множества функций.
- Использует jQuery (считается устаревающей JS-библиотекой).

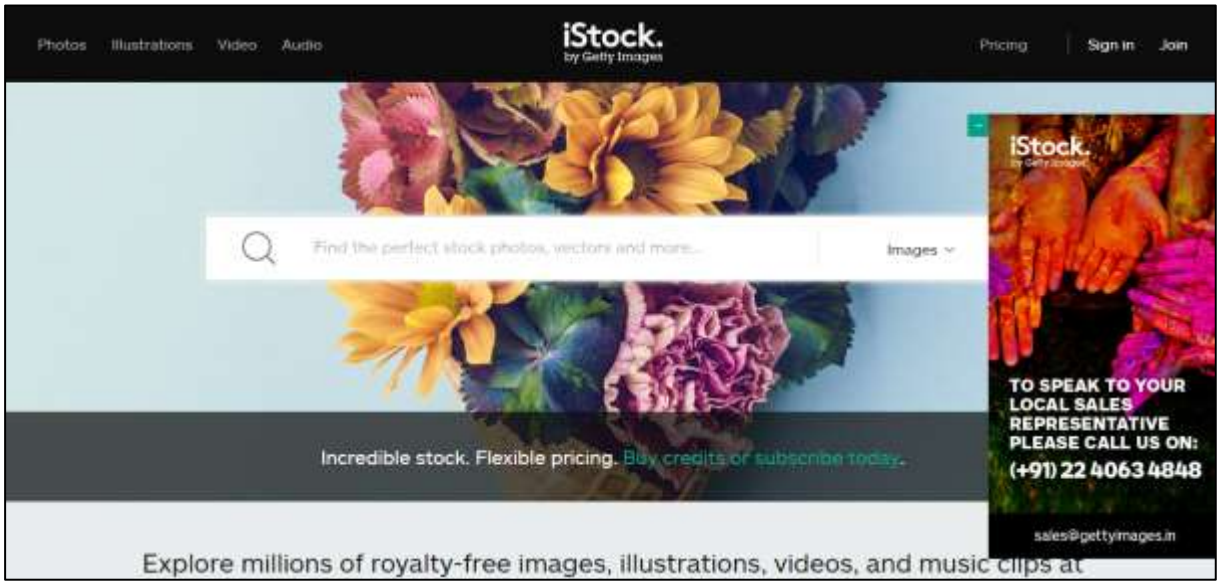


Рис. 1.263. Пример сайта, разработанного с помощью Foundation

Простые фреймворки

Для реализации простых проектов обычно не требуется сложного функционала. Иногда достаточно ограничиться адаптивной сеткой и простым, но современным интерфейсом. В этом случае следует подбирать легковесные фреймворки, которые быстро загружаются и позволяют использовать только необходимые компоненты.

Обозначим некоторые из таких.

UIkit – это небольшой фреймворк с акцентом на минимализме в дизайне. Это удобный фреймворк для различных приложений, особенно – для мобильных платформ. В UIkit включен комплект реализованных компонент для различных блоков сайта.

Однако UIkit не может быть расширен или изменен. Поэтому не получится интегрировать его, например – с React.

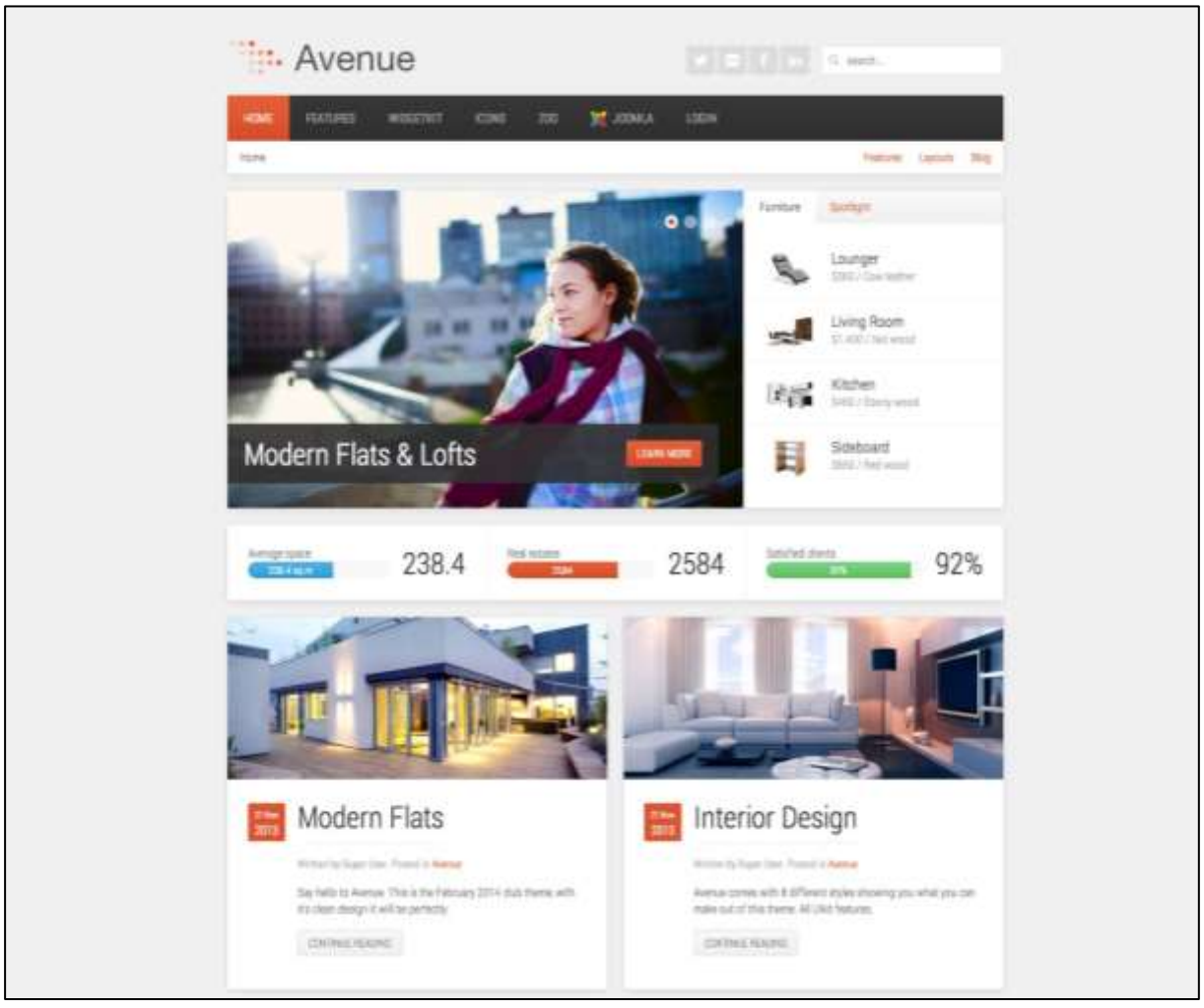


Рис. 1.264. Пример веб-сайта на UIkit

Skeleton – это фреймворк, использующий CSS и JavaScript. Он позволяет верстать отзывчивые сайты для различных платформ и устройств. В пакет фреймворка входит все самое необходимое для верстальщика: настроенные адаптивные сетки, стильная типографика, настроенные элементы форм, вкладки и другие востребованные в работе компоненты.

Официальный сайт: http://getskeleton.com/.

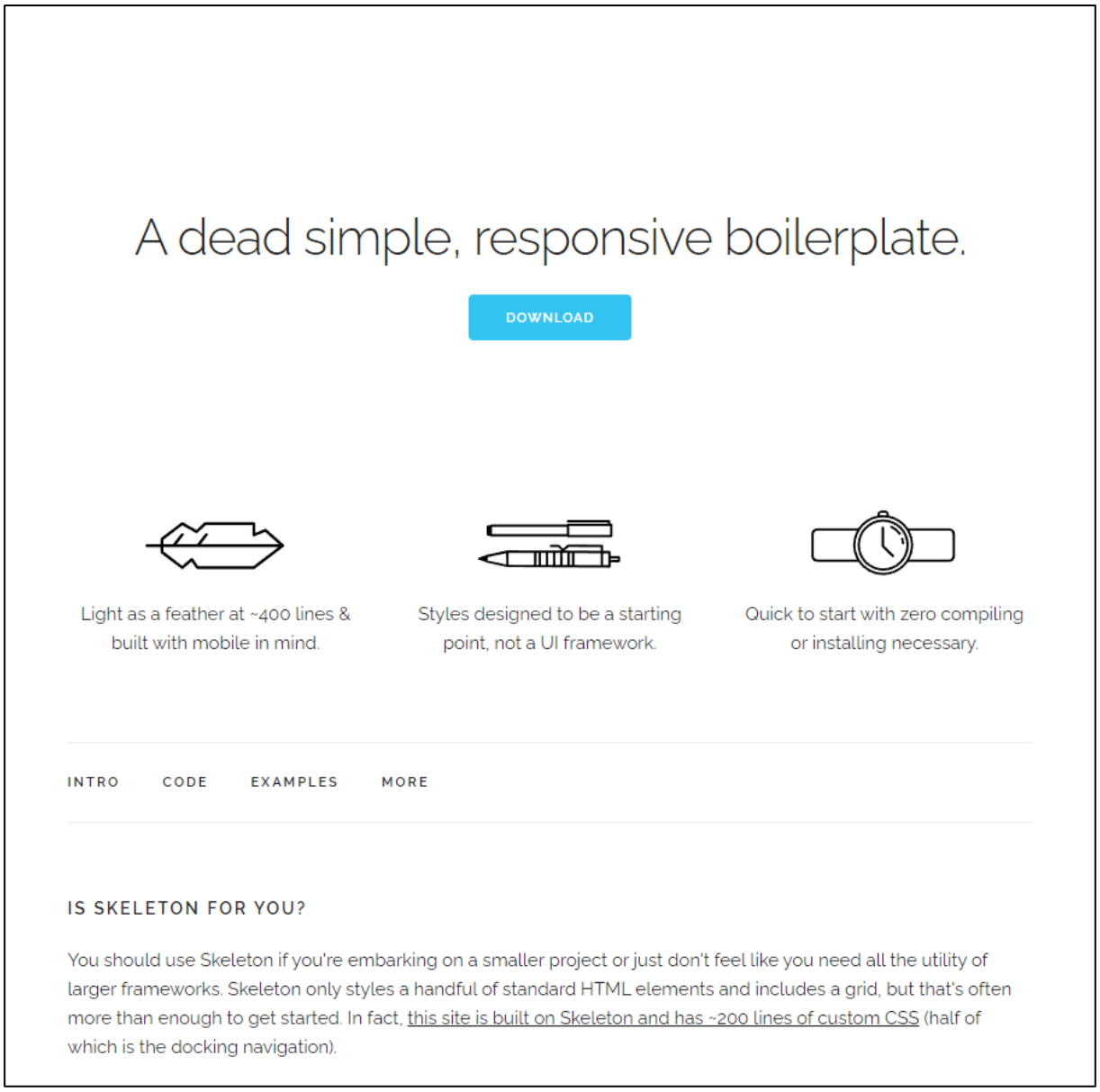


Рис. 1.265. Сайт Skeleton написан на нем самом

Pure.css – простой и легковесный фреймворк, созданный для разработки под мобильные устройства. Несмотря на малый размер, разработчику дополнительно предоставляется возможность загрузить лишь определенные модули фреймворка, необходимые в работе. В пакет входит нормализация стилей и классы для лаконичной и современной стилизации компонентов веб-страницы.

Разработчику Pure.css предлагает подробную документацию с многочисленными примерами. Официальный сайт: https://purecss.io/.

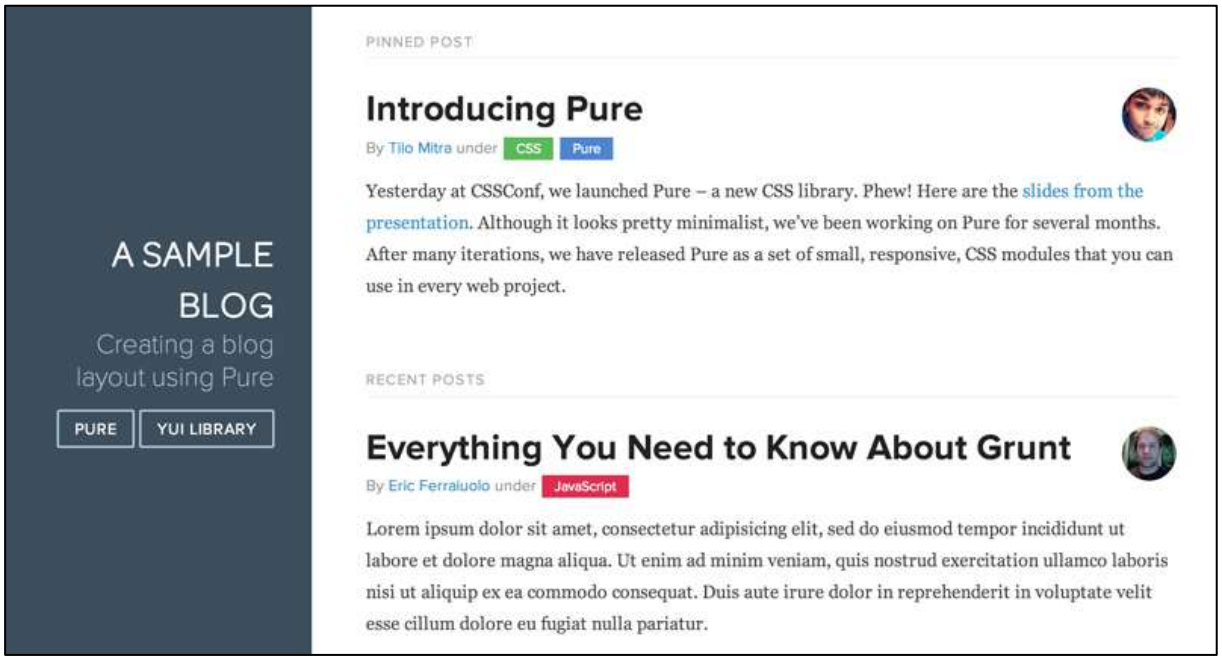


Рис. 1.266. Фрагмент веб-страницы, сверстанной с использованием Pure.css

Milligram – это очень простой и легкий фреймворк на базе CSS. В его состав входит адаптивная сетка и простая стилизация компонент. При необходимости разработчик без каких-либо ограничений может определить собственную стилистику и функционал.

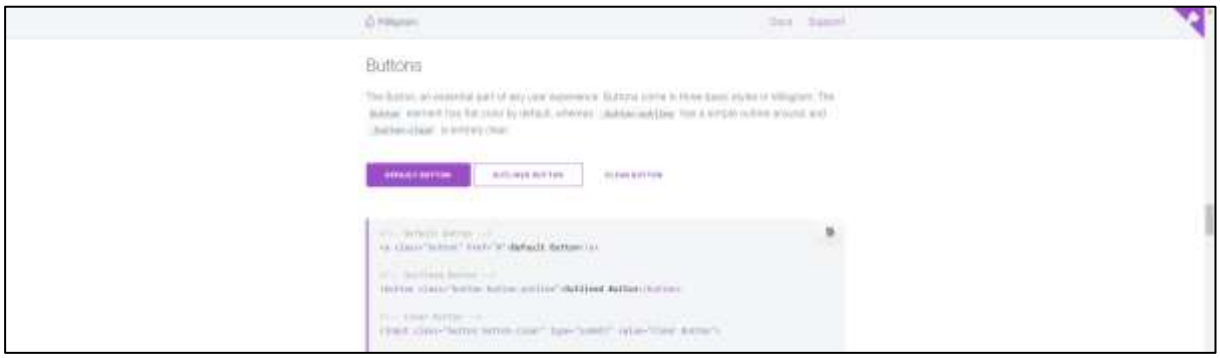


Рис. 1.267. Фреймворк Milligram

Milligram в сжатом виде занимает всего лишь 2 КБ. Для начала работы с ним достаточно беглого ознакомления с документацией на официальном сайте: https://milligram.io/.

Вопросы для самопроверки

1. Что общего и в чем разница между верстkой бумажных изданий и верстkой веб-сайтов?
2. Какие требования предъявляются к верстке современных веб-сайтов?
3. Опишите основные этапы верстки веб-сайта.
4. Что важно протестировать после завершения верстки веб-страниц?
5. В чем заключается сущность методологии БЭМ?
6. Опишите особенности работы с блоками, элементами и модификаторами в БЭМ.
7. Почему БЭМ позволяет писать универсальный и хорошо масштабируемый код?
8. Опишите общие возможности фреймворков для верстки.

Практикум

Задание 1

1. Проанализируйте на примере 1-2 веб-сайтов, насколько их разметка соответствует описанным в работе правилам.
2. В качестве примера можно взять новостные порталы.

Задание 2

1. Перед выполнением задания рекомендуется изучить дополнительный материал:
   a. БЭМ. Быстрый старт.
   b. БЭМ. Как писать классы БЭМ?
2. Пусть задана приведенная ниже разметка.
3. Используя полученные в текущем параграфе знания о методологии БЭМ, подключите к тегам классы для оформления блоков и элементов.

4. Исходите из того, что в шапке сайта расположена форма авторизации. Отразите зависимости в названиях классов, согласно правилам именования блоков и элементов БЭМ.
5. По аналогии добавьте некоторые модификаторы (в предположении, что блоки и элементы далее будут описаны).
6. Описывать стили CSS-классов не требуется.

Глава 1\БЭМ\index.html

```
<!DOCTYPE html>
<html lang="ru">

<head>
    <meta charset="UTF-8">
    <meta http-equiv="X-UA-Compatible" content="IE=edge">
    <meta name="viewport" content="width=device-width,
initial-scale=1.0">
    <title>Document</title>
    <link rel="stylesheet" href="style.css">
</head>

<body>
    <header>
        <img src="#" alt="">
        <form action="">
            <label for="login">Логин: </label>
            <input type="text" id="login">
            <label for="userPass">Пароль: </label>
            <input type="password" id="userPass">
            <button>Войти</button>
        </form>
    </header>

    <main>
        <div>
            <!-- Здесь разметка содержимого -->
        </div>
    </main>
</body>

</html>
```

Задание 3

1. Перед выполнением задания рекомендуется закрепить изученный материал:
   a. БЭМ. Быстрый старт.
   b. БЭМ. Организация файловой структуры. (достаточно структуры классической Nested).
2. Сделайте полную копию каталога выполненного задания 2.
3. Согласно выделенным блокам разбейте описание их стилей и скриптов на отдельные каталоги (см. пример организации иерархии в п. 1.8.2). Соблюдайте правила именования БЭМ.
4. Описывать стили и скрипты не требуется.
5. Замечание. В качестве основы используйте основной стилевой файл *style.css*, который размещен в корневом каталоге. Чтобы из этого файла перейти к вложенным, можно воспользоваться функцией импорта стилей. Например, тело документа согласно разметке можно условно разбить на два блока: *header.css* (шапка сайта) и *main.css* (контент). Тогда в корневом файле стилей *style.css* можно подключить вспомогательные следующим образом:

```
@import url('header/header.css');
@import url('main/main.css');
```

6. Далее для каждого из двух блоков *header* и *main* аналогичным образом подключаются вложенные в них блоки и элементы. Названия каталогов, стилей и скриптов должны также соответствовать правилам именования в БЭМ.

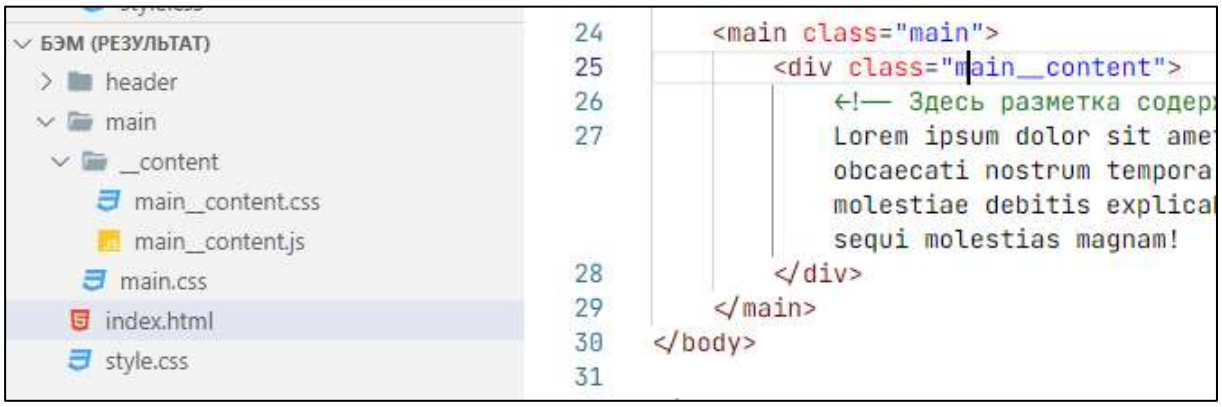


Рис. 1.268. Пример того, как может быть организовано вложение блоков и элементов для раздела `<main>`

ГЛАВА 2.
ТЕХНОЛОГИЯ HTML5

2.1. Технологии frontend-разработки

2.1.1. Порядок работы с курсом

Цель практического курса

Следующие главы текущего учебного пособия посвящены трем важнейшим технологиям frontend-разработки: HTML, CSS и JavaScript. Материалы глав изложены таким образом, чтобы читатель мог подробно изучить фундаментальные возможности каждой технологии на основе многочисленных примеров. Для формирования опыта работы с перечисленными технологиями каждый параграф сопровождается задачами.

В этих главах читатель познакомится с последними стандартами HTML5, CSS3, JavaScript и изучит возможности использования этих технологий в качестве инструмента решения практических задач. Мы рассмотрим возможности их применения в практике учителей математики и информатики, изложим фундаментальные особенности верстки веб-страниц. И чтобы процесс изучения был динамичным, читатель будет работать с профессиональным текстовым редакторов Visual Studio Code.

Цель этого курса – сформировать и систематизировать полученные знания в области верстки веб-страниц, объектно-ориентированного программирования и сформировать опыт их применения в реализации практически значимых задач учителя математики и информатики. Читатель убедиться, насколько широким потенциалом обладают технологии HTML, CSS и JavaScript.

Важное замечание!

В главе 1 подробно изложен материал по особенностям работы сети Интернет и описаны возможности технологий веб-разработки. Предполагается, что изучение первой главы ведется параллельно изучению последующих глав.

Веб-технологии в работе учителя

Работу современного специалиста сферы образования сложно представить без использования веб-технологий. В частности, педагоги все чаще создают блоги и веб-сайты для организации работы со своими учащимися, используют веб-сервисы для коммуникации и решения учебных задач.

Однако современные технологии веб-разработки также являются мощным инструментом, который находит применение в решении практических задач педагога: визуализации, автоматизации рутинных процессов, создании программ-тренажеров, обучении школьников математике и информатике.

Это полезно знать!

В этом курсе мы хотим показать, что современному учителю, особенно математики и информатики, важна компетентность не только в использовании прикладных программ, но и инструментальных средств разработки ПО. Технологии HTML, CSS и JavaScript идеально сочетают в себе гибкий инструмент для создания привлекательного и современного дизайна и программирование функционала ресурса.

Чем интересны рассмотренные в курсе технологии?

- Закрепляют и систематизируют компетенции, полученные студентом при изучении основ программирования и разработки ПО.
- Помогают развивать эффективный подход к исследованию и решению практических задач.

- Формируют и развивают навыки поэтапного планирования и проектирования.
- Дают свободу выбора инструмента для реализации задач профессиональной деятельности.
- Развивают опыт в сфере создания и настройки пользовательского интерфейса учебного ресурса согласно собственным потребностям.
- Веб-разработка – это большой потенциал возможностей.

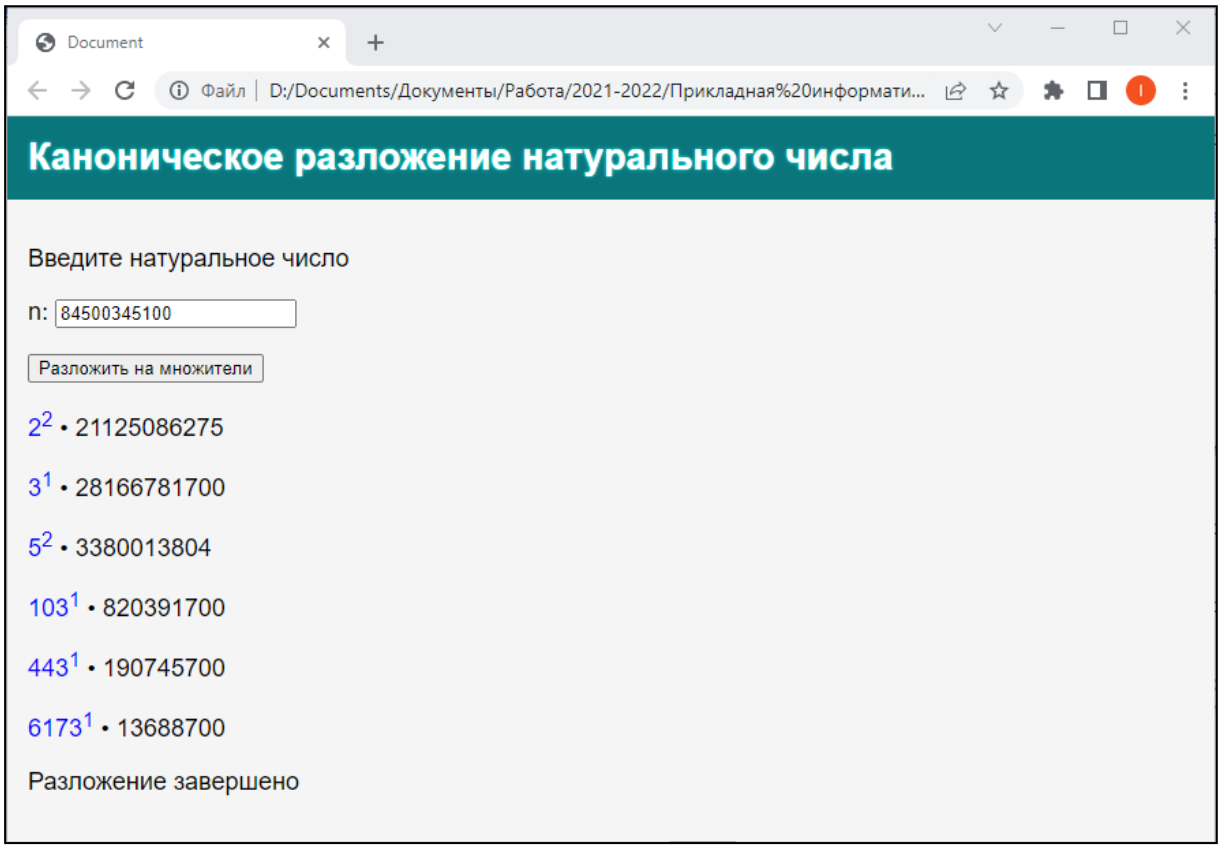


Рис. 2.1. Пример простой формы на JavaScript для демонстрации разложения числа на множители

С чем познакомится читатель в этом курсе?

- Вы узнаете, что такое frontend и что входит в его основу.
- Познакомитесь с языком разметки HTML5.
- Изучите фундаментальные возможности и особенности стилизации веб-страниц с помощью CSS3.
- Научитесь верстать веб-страницы с использованием профессионального текстового редактора.

- Познакомитесь с современным и гибким языком программирования JavaScript.
- Отработаете свои навыки в программировании задач и реализации небольших проектов.

2.1.2. Модель «клиент-сервер»

Особенности архитектуры

Компоненты

Архитектура «клиент-сервер» определяет основную модель работы сети Интернет и предполагает распределению систему обработки и передачи данных между поставщиками услуг и пользователями. За контроль корректности операций в сети отвечают *протоколы*.

В клиент-серверной архитектуре можно выделить следующие важные компоненты:

- *клиент* (устройство, подключенное к сети, в частности пользователь);
- *сервер* (отвечает за обработку запросов одного и более клиентов сети);
- *сеть* (осуществляет передачу данных между узлами);
- *приложения* (программное обеспечение, участвующее в формировании, обработке и передачи данных в сети).

Особенности работы

Сервер способен работать со множеством клиентов одновременно, ориентируясь на приоритеты и очередность запросов. При этом обработка данных происходит в скрытом для пользователя режиме.

Обычно взаимодействие пользователя с сервером осуществляется посредством браузера или веб-приложений, которые предоставляют визуальный интерфейс ресурса. Сервер может использоваться в роли:

- аппаратного обеспечения;
- кластера серверов;
- локальной машины.

Корректность процесса обмена данными обеспечивается сетевыми протоколами, в частности – HTTP (подробнее разделе 1.1.2).

Программные технологии

Работа сети зависит от возможностей аппаратуры, программного обеспечения клиента и сервера.

Кратко опишем схему работы.

Пользователь с помощью браузера (клиент) отправляет серверу запрос на получение данных с веб-ресурса. Сервер обрабатывает запрос и высылает ответ пользователю.

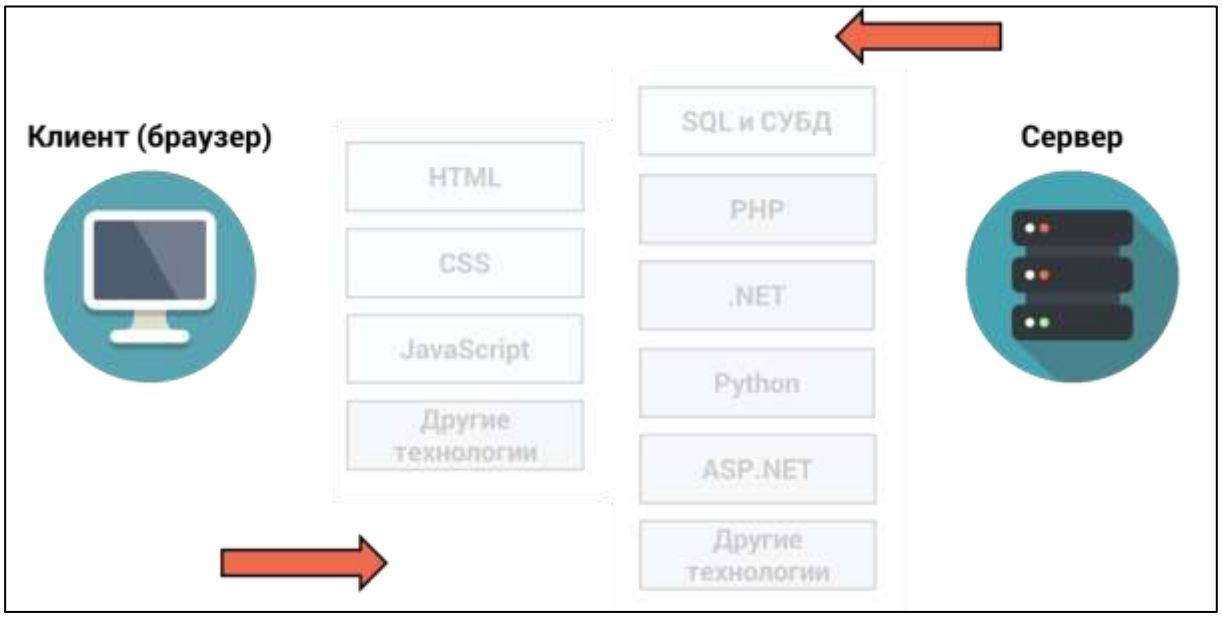


Рис. 2.2. Взаимодействие сервера и клиента

Технологии, реализующие работу в веб-ресурсом на стороне клиента, называют *frontend-технологиями* («фронтенд»).

В основе frontend лежат три технологии: HTML, CSS и JavaScript. Остальные являются либо их ветвями, либо предоставляют дополнительный функционал.

Это полезно знать!

Основной задачей frontend-разработки является создание визуального интерфейса веб-сайта или веб-приложения, а также программирование его функционала на стороне клиента (анимация, управление, обработка данных, подготовка данных к отправке серверу и т.п.).

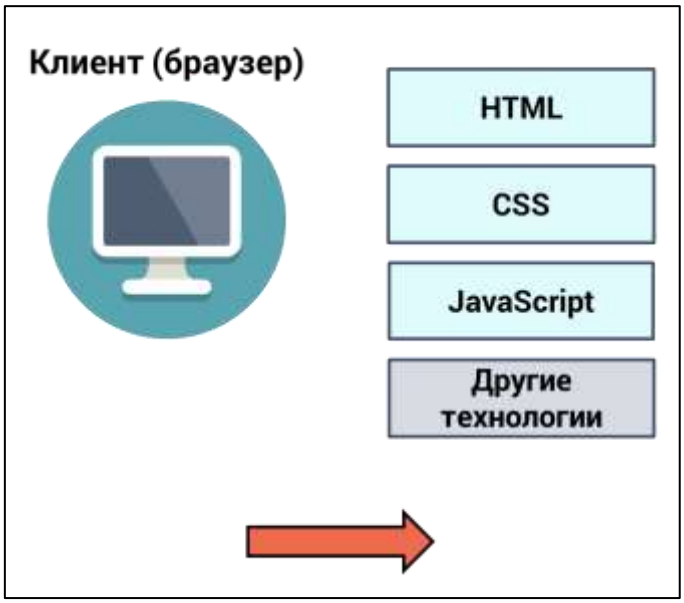


Рис. 2.3. Frontend-технологии

Технологии, реализующие работу в веб-ресурсом на стороне сервера, называют *backend-технологиями* («бэкенд»).

Обычно сервер представляет собой систему компьютеров высокой вычислительной мощности. Программную основу сервера составляют базы данных.

Сегодня backend реализуют разные технологии, которые могут работать как вместе, так и по отдельности решать свой круг задач.

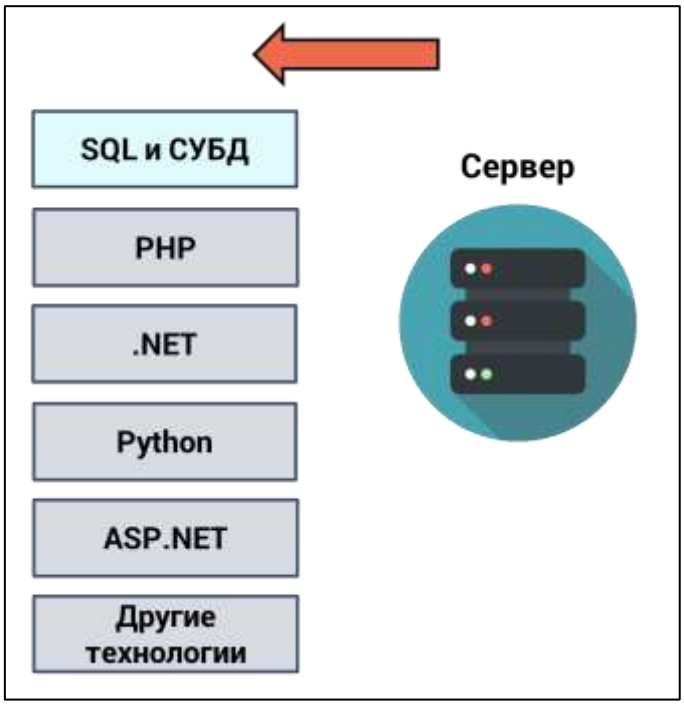


Рис. 2.4. Backend-технологии

Это полезно знать!

Главной задачей backend-разработки является обработка данных на стороне сервера (управление СУБД, защищенная передача данных клиенту, обработка запросов, формирование отчетов и т.д.).

Важное замечание!

В этом практическом курсе мы рассматриваем только frontend-технологии. Подробное описание их вариаций и связанных вопросов вы найдете в главе 1.

2.1.3. Статические и динамические страницы

Статические веб-страницы

Статические веб-сайты состоят из неизменяемых страниц. Содержимое и внешний вид таких страниц остается постоянным, т.е. сохраняется HTML-структура страницы. С помощью CSS и JavaScript обеспечивается некоторые эффекты анимации, однако это не влияет на структуру. Статической странице также не требуется изменение, формируемое севером на запрос пользователя.

Браузер получает от сервера одну страницу за один запрос и в силу постоянности содержимого обеспечивается достаточно быстрая загрузка страницы в браузер.

Статические сайты часто используются в простых проектах: документация, сайты-визитки, каталоги продуктов и услуг, формы отчетов и т.п. В ряде случаев на статическую страницу можно добавить отдельные динамические элементы (например, строку поиска).

Рис. 2.5. Сайты с статическими веб-страницами

Динамические веб-страницы

Динамические веб-сайты могут динамически изменять свою структуру и оформление. В отличие от статических веб-страниц, они не хранятся на сервере как сформированные документы, а создается по запросу пользователя или автоматически. В начале сервер ищет требуемую страницу и отправляет ее интерпретатору, который выполняет HTML-код. Далее документ отправляется на сервер и отображается в браузере в виде сформированной веб-страницы. Динамическая интерпретация производится на стороне сервера и с использованием возможностей языков программирования Java, PHP, Ruby, различных фреймворков и .т.п.

Иными словами, данные такой веб-страницы могут обновляться в процессе работы и фрагментарно обновляться, не требуя перезагрузки всей страницы. Часто этот подход обозначают аббревиатурой *DHTML* (Dynamic HTML)

Динамические страницы можно создавать, используя также *системы управления контентом* (*CMS*). Многие современные типы сайтов предполагают работу с динамическими веб-страницами.

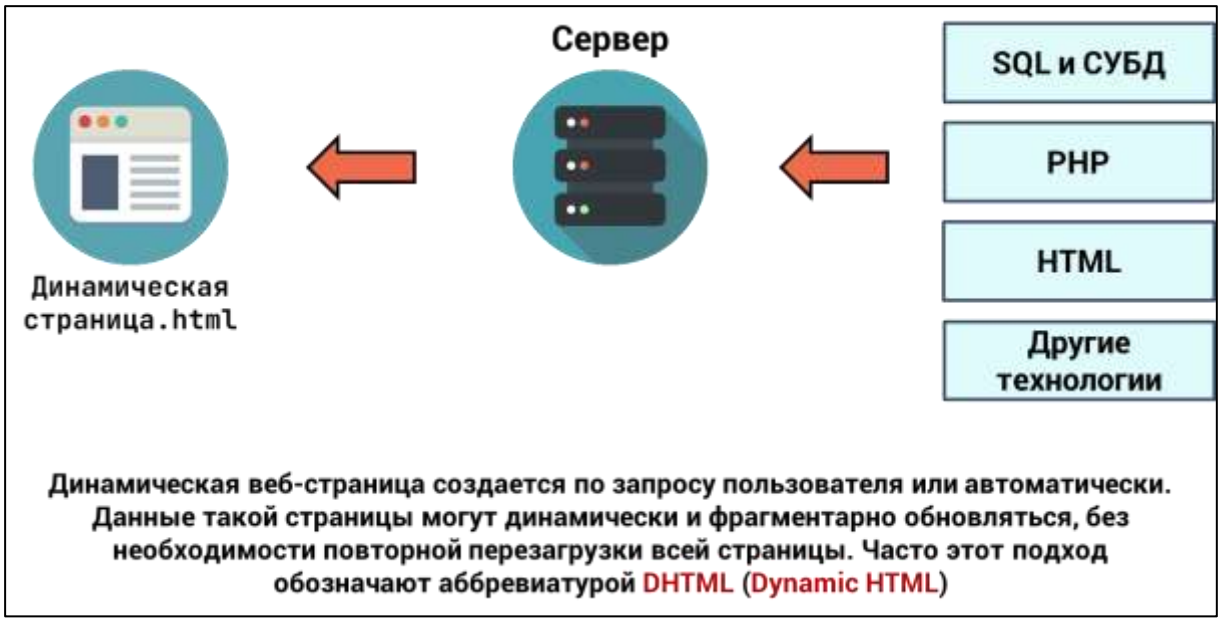


Рис. 2.6. Сайты с динамическими веб-страницами

Смешанный вариант

На практике часто возможности статических и динамических страниц комбинируют. Это позволяет эффективнее обеспечить высокую скорость загрузки статических страниц и функционал пользователя за счет динамических страниц.

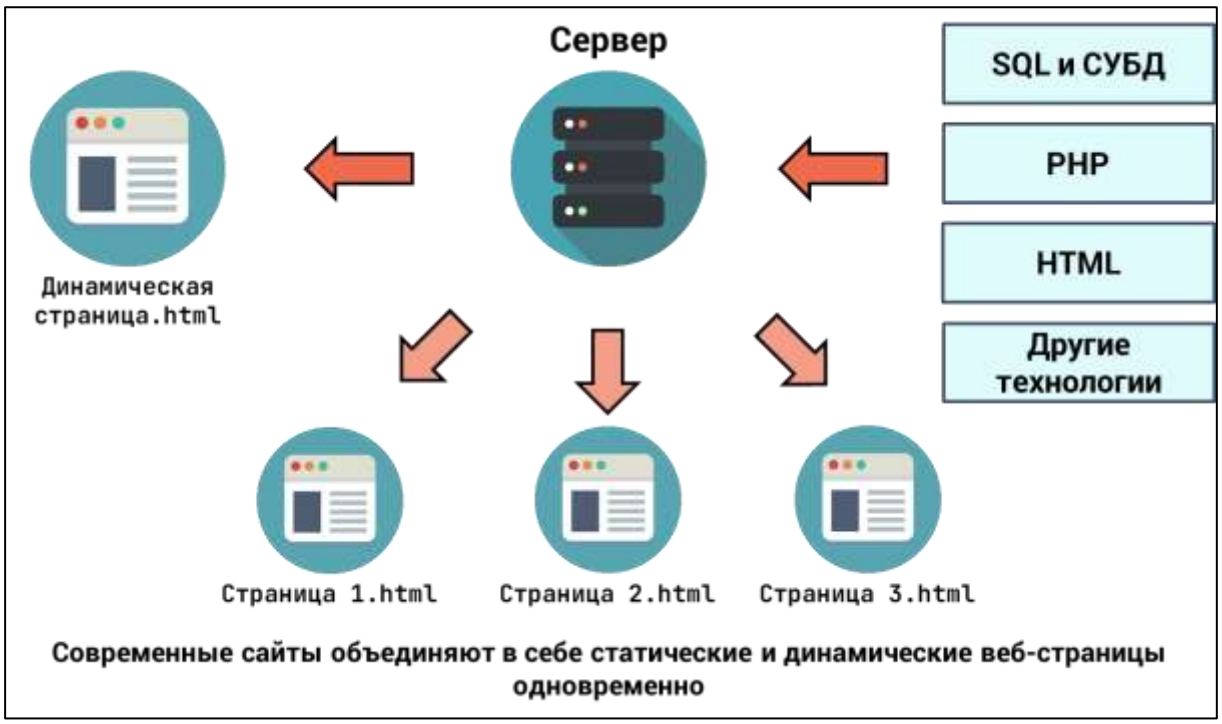


Рис. 2.7. Статические и динамические страницы в одном сайте

2.1.4. Технологии HTML, CSS, JavaScript

Роль технологий

Прежде чем приступить к детальному изучения перечисленных технологий в последующих главах, обозначим кратко их роль и значимость в frontend-разработке.

1. *HTML* – это язык гипертекстовой разметки текстового и графического содержимого на веб-странице. Он определяет структуру документа и роль элементов в нем.
2. *CSS* – это технология каскадных таблиц стилей, которые предназначены для визуального оформления документа. Обычно они используются совместно с HTML.
3. *JavaScript* – это язык программирования, который позволяет управлять HTML-структурой документа и динамически менять CSS-свойства.

Это полезно знать!

Несмотря на самодостаточность каждой из трех технологий, их обычно используют совместно.

Рис. 2.8. HTML, CSS и JavaScript: образное сравнение роли технологий на примере станции метро

Возможности использования

В сочетании HTML, CSS, JavaScript дают разработчику мощный инструмент реализации его идей:

- Возможность разрабатывать любой дизайн проекта: от классического «оконного» интерфейса с привычными элементами управления до современного и стильного веб-сайта с мультимедийными компонентами.
- Доступность удобных и бесплатных инструментов разработки: мощные среды разработки, продвинутые текстовые редакторы, библиотеки, фреймворки и платформы.
- Отсутствие необходимости устанавливать программы и системы на свой ПК: для работы с веб-страницами необходим только браузер.

Кроме того, стек веб-технологий постоянно развивается, предоставляя разработчику множество готовых решений.

Одним из важных преимуществ стека технологий из HTML, CSS и JavaScript является высокая гибкость решения прикладных задач. Frontend-разработчик получает достаточно универсальный инструмент, позволяющий одновременно работать с дизайном и функциональной частью приложения. А использование фреймворков и технологий backend-разработки позволяет выйти за рамки браузера.

При этом изучение основ HTML, CSS и JavaScript сравнительно простое: начинающий разработчик получает фундаментальное представление о процедуре разработки ПО.

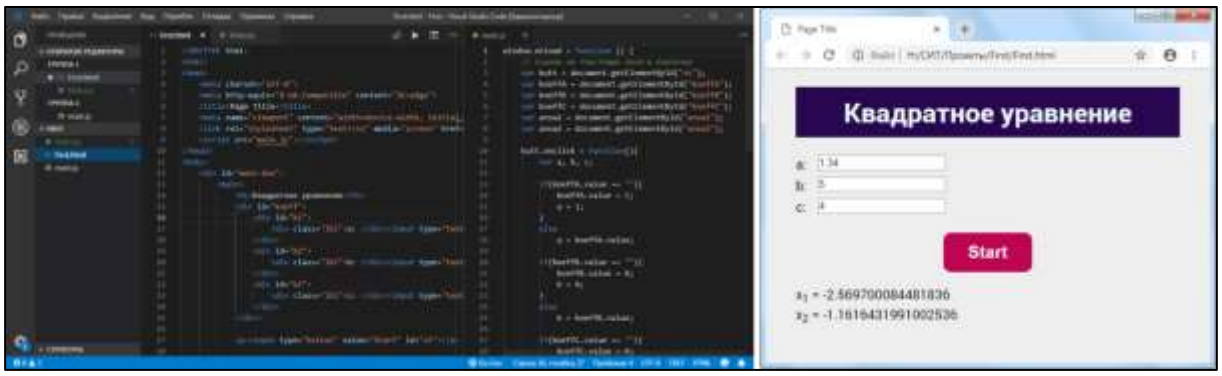


Рис. 2.9. Работа с frontend-технологиями предполагает возможность модульного построения, где каждая часть реализуется отдельно, но в рамках единого проекта

Вопросы для самопроверки

1. Какие возможности открывают современные веб-технологии учителю?
2. Что входит в модель клиент-сервер и как она работает?
3. Перечислите основные задачи, которыми занимается fronted- и backend разработчик?
4. Опишите отличие между статическими и динамическими веб-страницами.
5. Почему технологии HTML, CSS и JavaScript дают разработчику большой потенциал в реализации проектов?

Практикум

Задание 1

1. Приведите примеры ситуаций, когда статические веб-страницы являются оптимальный вариантом реализации веб-ресурса.
2. Найдите в сети Интернет примеры подобных статических веб-страниц.
3. Как можно определить, что страница именно статическая?
4. Приведите примеры веб-сайтов, которые используют динамические и статические веб-страницы.
5. Возможно ли определить, с помощью какого языка программирования или фреймворка реализовано динамическое поведение?

Задание 3

1. Проанализируйте некоторые учебники по информатике на предмет качества и полноты изложения материала по теме «Разработка веб-сайтов». Используйте Федеральный перечень учебников: https://fpu.edu.ru/.
2. В качестве критериев используйте следующие:
    a. полнота изложения материала;
    b. изучение возможностей HTML, CSS и JavaScript;
    c. актуальность используемых технологий;
    d. наличие примеров и качественных иллюстраций, работа с ЭОР.

2.2. Знакомство с редактором Visual Studio Code

2.2.1. Текстовые редакторы в веб-разработке

Возможности

Определение

Текстовый редактор разработчика *– это редактор текстовых файлов с расширенными возможностями, поддерживающий полный или частичный функционал интегрированных сред разработки.*

Современные редакторы кода в большинстве своем обладают следующими возможностями:

- подсветка синтаксиса разных языков программирования;
- многочисленные функции редактирования и навигации по тексту;
- возможность работы в нескольких окнах и отдельных вкладках одновременно;
- поддерживают операции с кодировками файлов;
- позволяют устанавливать дополнительные плагины или надстройки.

Для более удобной работы редактору необходимо поддерживать дополнительные возможности:

- развитую систему IntelliSense и snippet (интеллектуальная подсказка, автозавершение команд и генерация фрагментов кода разметки одним нажатием);
- управление сборкой проекта;
- инструменты отладки;
- работа с системой контроля версий Git.

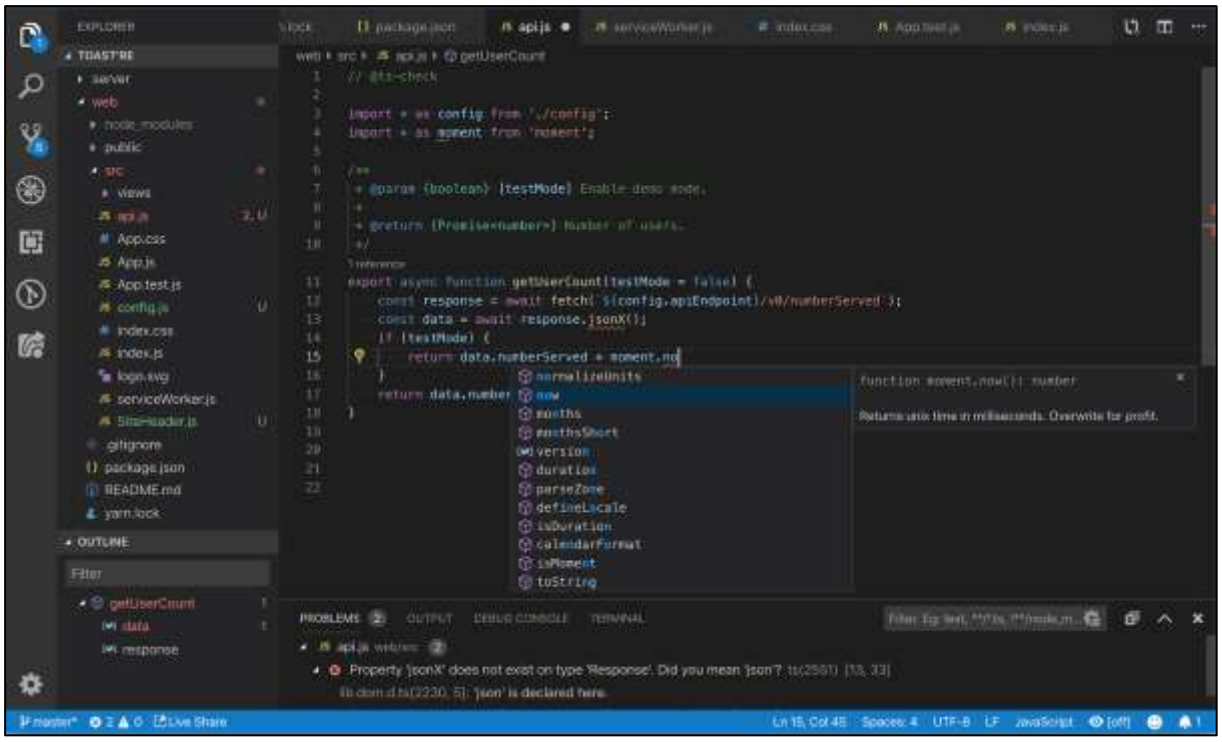


Рис. 2.10. Редактор Visual Studio Code – мощный бесплатный инструмент для frontend-разработчика

Примеры

В настоящее время разработчикам доступен широкий выбор бесплатных и платных текстовых редакторов. Многие из них позволяют уже изначально поддерживают работу с HTML, CSS и JavaScript, прежде всего – режим *IntelliSense* и плагин быстрой верстки *Emmet*. Подробно возможности некоторых популярных редакторов описаны в п. 1.5.3.

При выборе редактора для веб-разработки следует обращать внимание на такие детали, как:

- доступные возможности;
- поддержку языков веб-программирования;
- эргономику редактора;
- возможность интеграции системы контроля версии Git;
- права использования (платный или бесплатный).

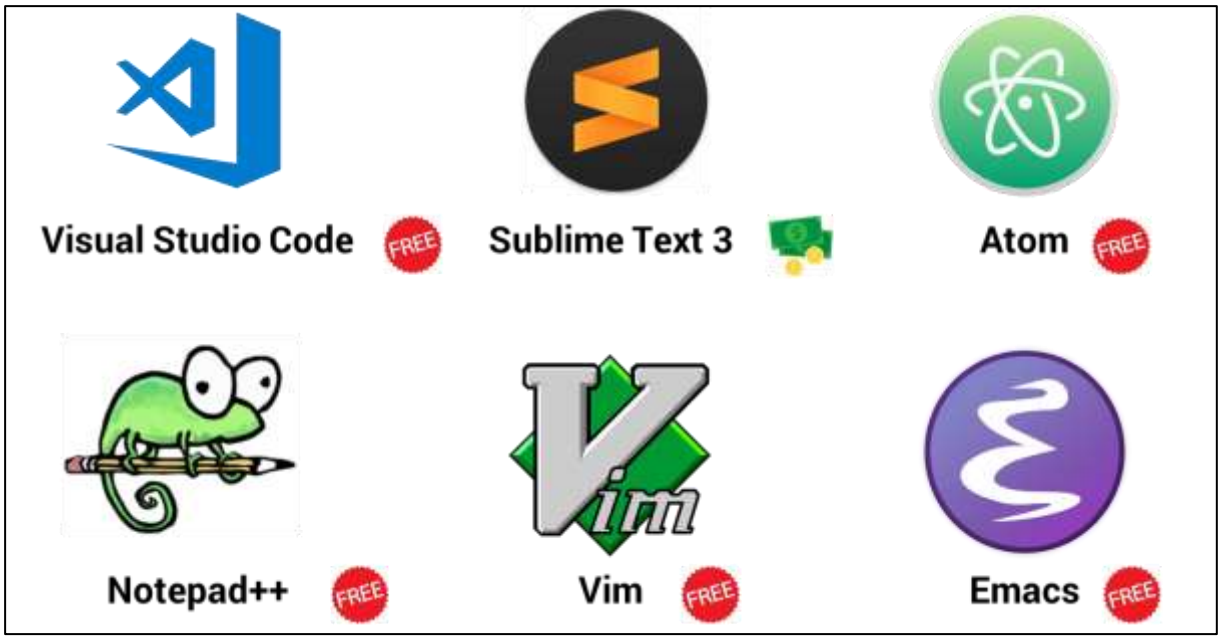


Рис. 2.11. Примеры текстовых редакторов, которые можно использовать в веб-разработке

Выбор редактора для новичка

В текущем курсе мы будем использовать бесплатный редактор Visual Studio Code. Он является наиболее оптимальный выбором для новичка, поскольку весьма прост и удобен в работе.

С другой стороны, опытный пользователь сможет настроить Visual Studio Code согласно своим предпочтениям, превратив в мощную среду разработки (и не только веб-сайтов).

Важное замечание!

Далее читатель будет постепенно знакомиться с возможностями Visual Studio Code. Настоятельно рекомендуем запоминать приемы работы с редактором и практиковать их в решении предложенных задач.

2.2.2. Редактор Visual Studio Code

Определение

Visual Studio Code *– это бесплатно распространяемый профессиональный текстовый редактор от Microsoft, поддерживающий разработку на большинстве популярных современных языков программирования.*

Рис. 2.12. Логотип редактора

Visual Studio Code предоставляет все самое необходимое для разработки под HTML, CSS и JavaScript:

- подсветка синтаксиса множества языков программирования и разметки;
- режимы IntelliSense и плагин Emmet;
- возможность работы в окнах («колонках») одновременно;
- встроенный терминал;
- гибкие возможности конфигурации;
- возможность синхронизации с системой контроля версий;
- удобная установка плагинов из сети Интернет.

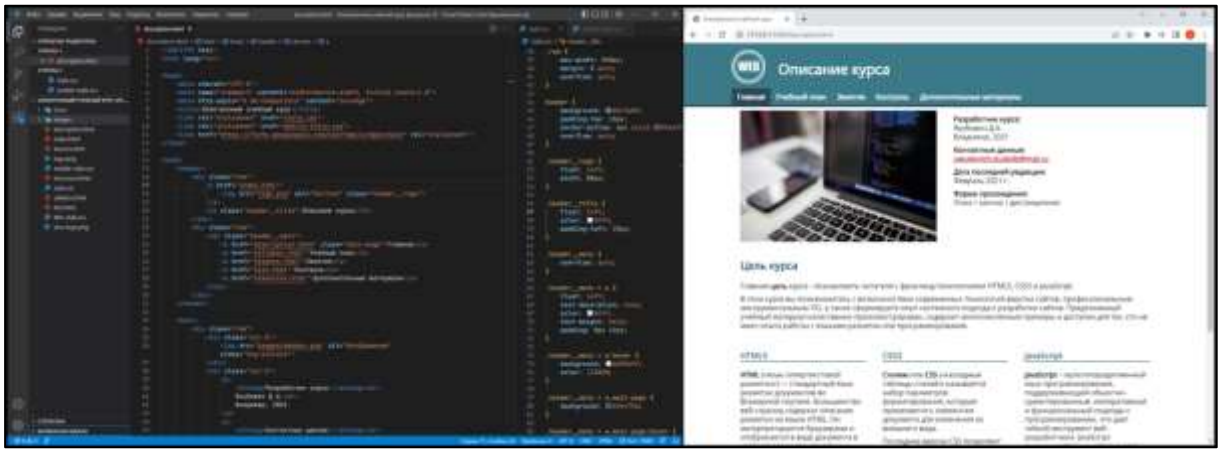


Рис. 2.13. Использование Visual Studio Code в верстке веб-сайта

2.2.3. Установка и конфигурирование редактора

Скачивание и установка

Для скачивания редактора перейдите на официальный сайт Visual Studio Code и нажмите на кнопку *Download*: https://code.visualstudio.com/.

По нажатию на выпадающее меню можно выбрать файл установки под вашу операционной систему. Следует выбирать Stable-версию (стабильную).

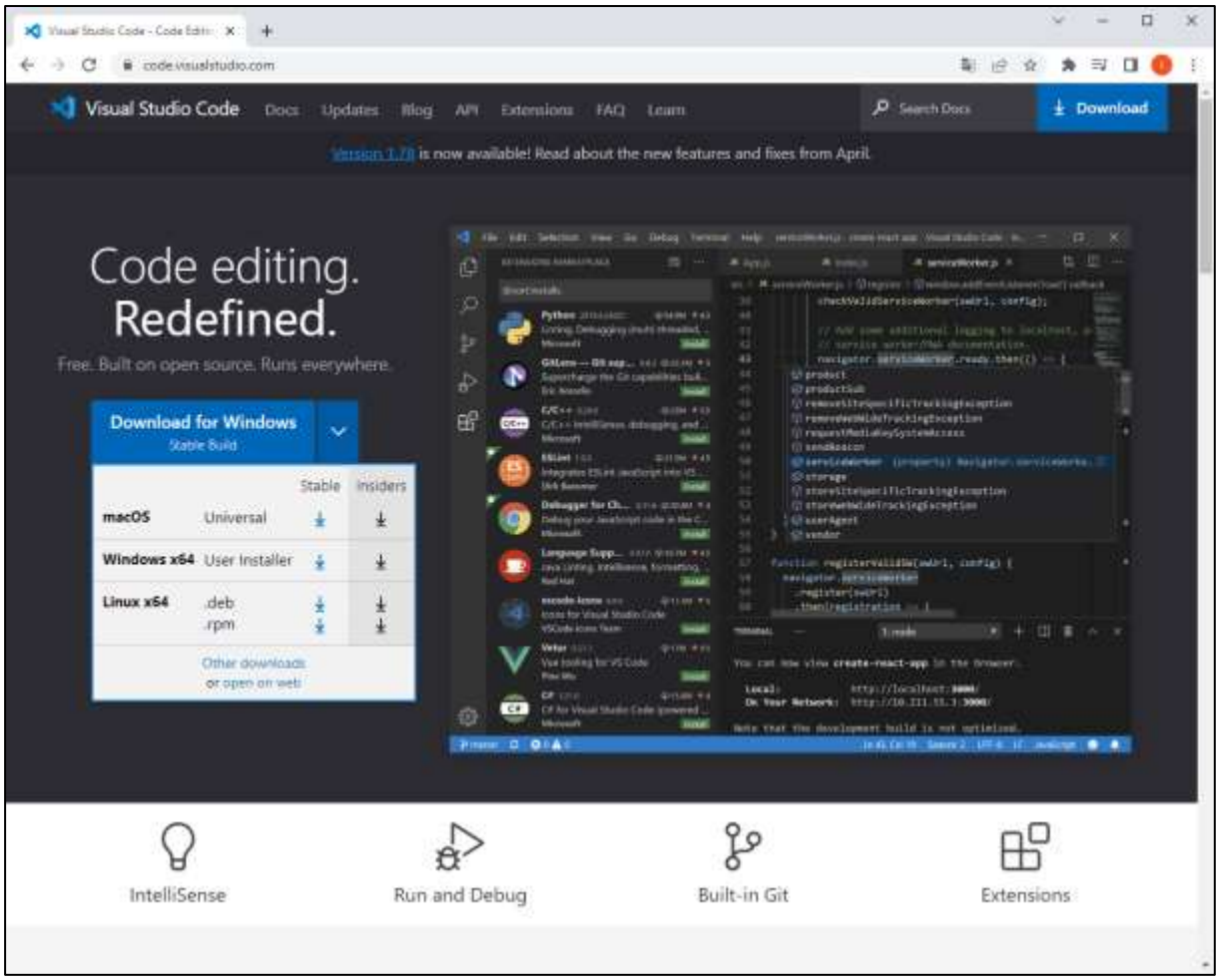


Рис. 2.14. Скачивание Visual Studio Code

Процедура установки редактора происходит в автоматическом режиме. Запустите инсталлятор. Осуществите предложенные действия: примите лицензионное соглашение, при необходимости измените каталог установки и название в меню Пуск:

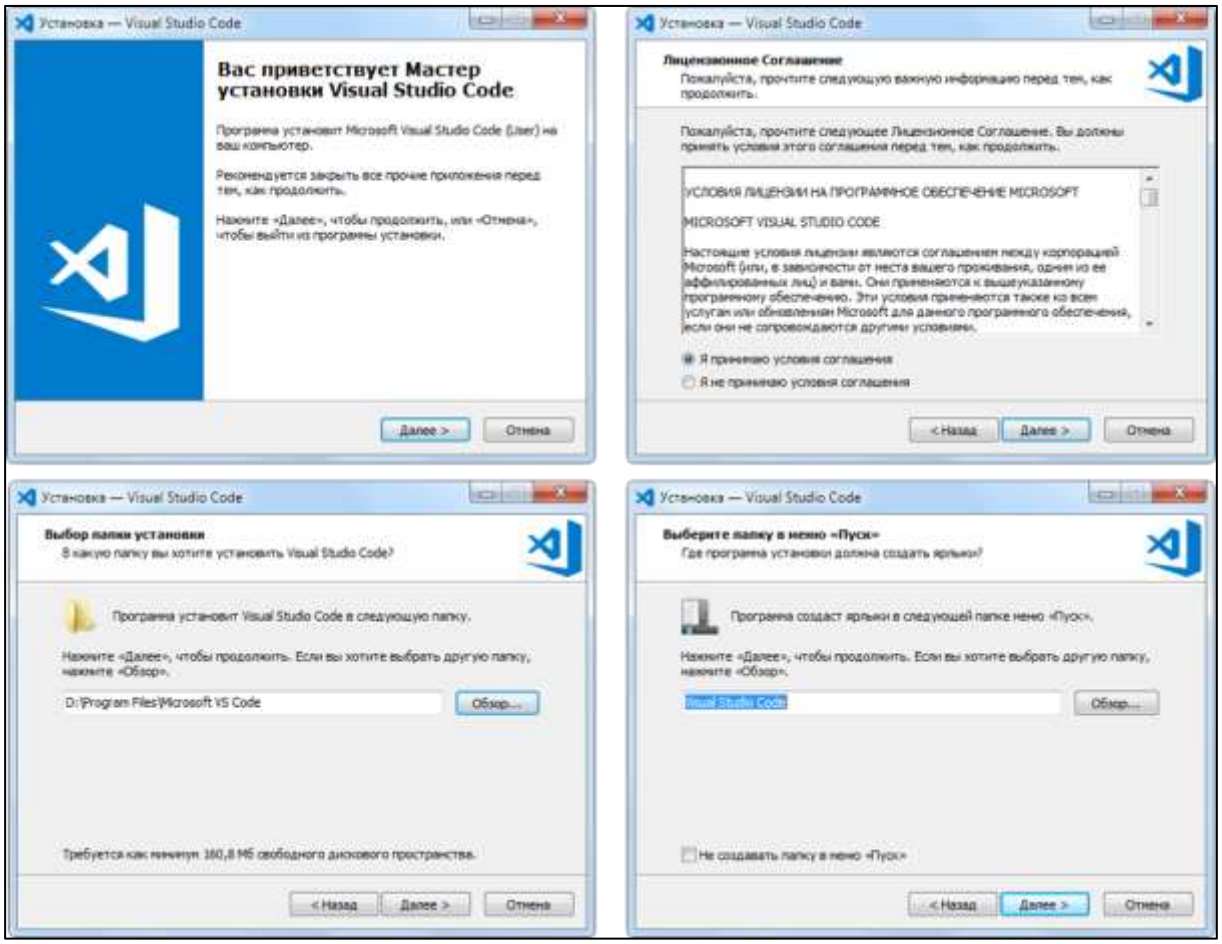


Рис. 2.15. Установка VSC: подготовительная фаза

Далее появится окно *Выберите дополнительные задачи*: установка первых двух галочек внесет редактор в контекстное меню правой кнопки мыши (далее *ПКМ*), что позволит открывать любые текстовые файлы с помощью Visual Studio Code.

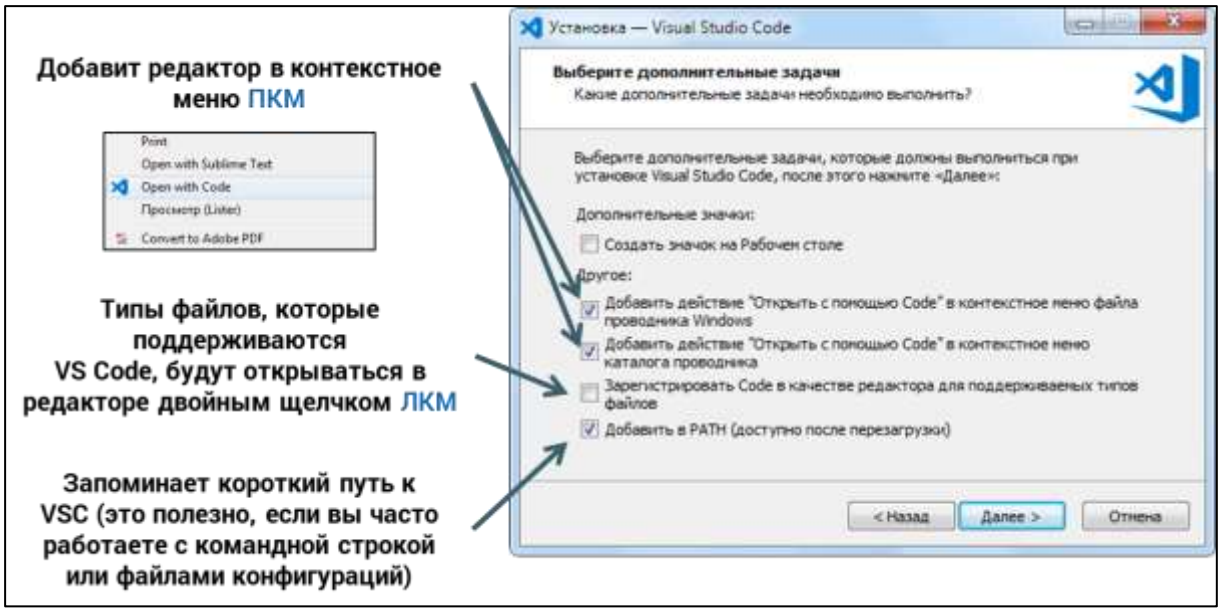


Рис. 2.16. Установка VSC: дополнительные опции

Далее начнется процесс установки. Дождитесь его окончания:

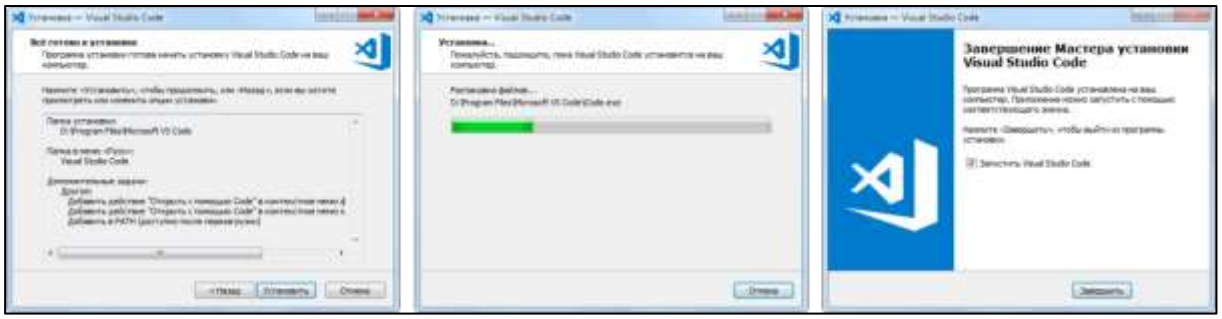


Рис. 2.17. Установка VSC: распаковка файлов на компьютер

Запуск редактора

После завершения процесса установки запустите редактор. Стартовая страница редактора выглядит, как изображено на рис. 2.18.

В начале установите руссификацию интерфейса редактора. Нажмите на кнопку *Расширения* (рис. 2.19), вбейте в строке поиска слово «russian» и выберете плагин *Russian Language Pack for Visual Studio Code*. Нажмите на кнопку *Install* и дождитесь установки пакета (расширения скачиваются из сети Интернет). Обязательно перезагрузите редактор (самостоятельно или по предложенному в углу уведомлению).

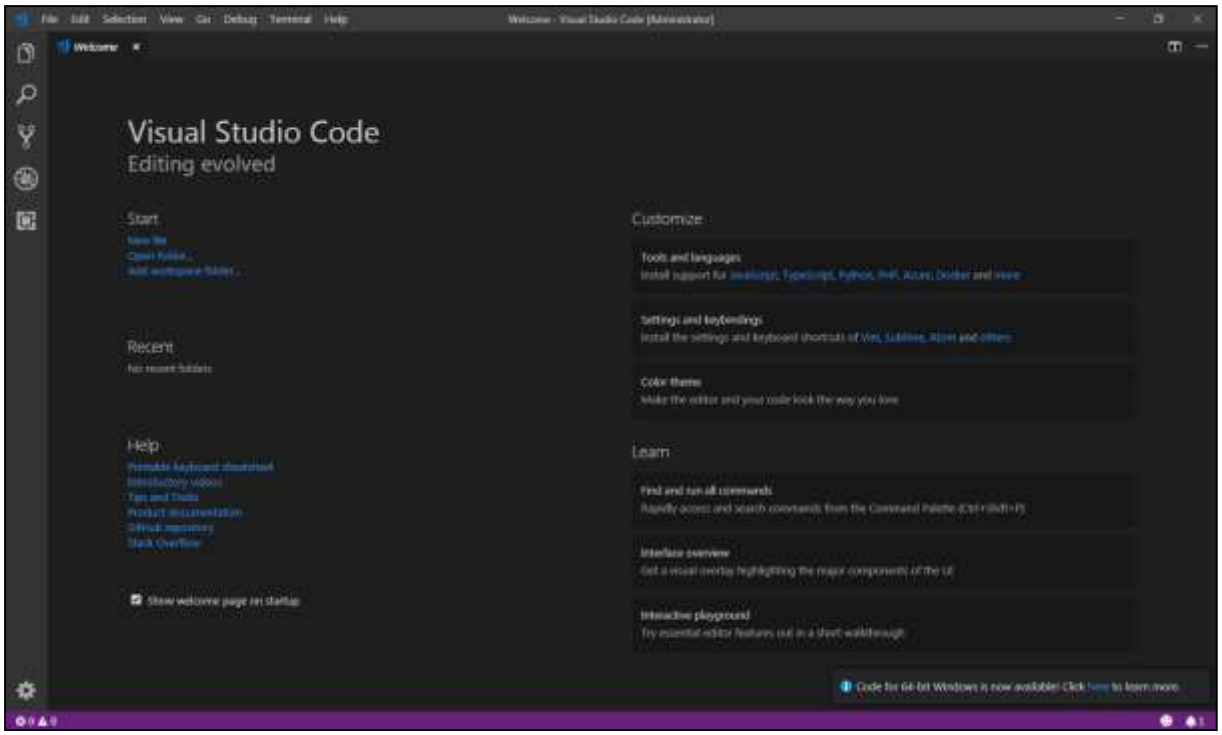


Рис. 2.18. Стартовая страница

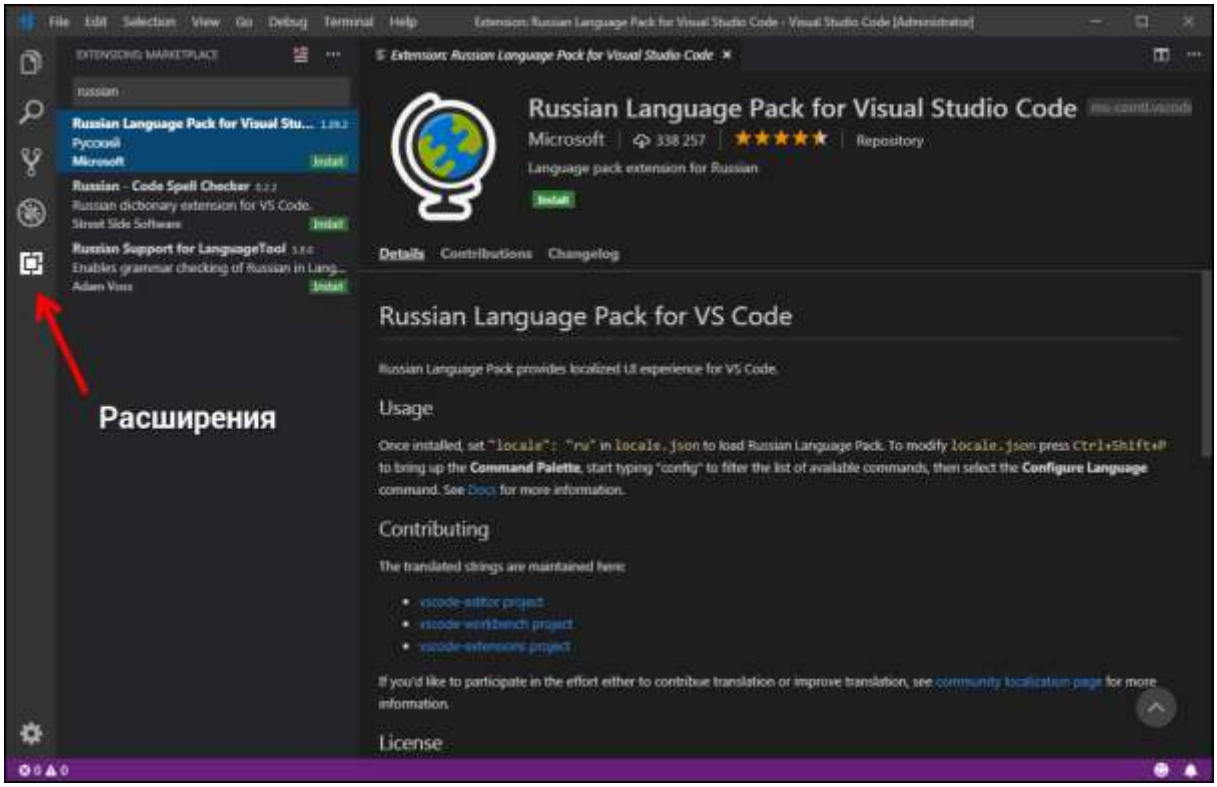


Рис. 2.19. Установка руссификации для интерфейса редактора

Создание нового файла

Чтобы создать новый файл, в верхней панели меню нажмите *Файл / Создать текстовый файл...*:

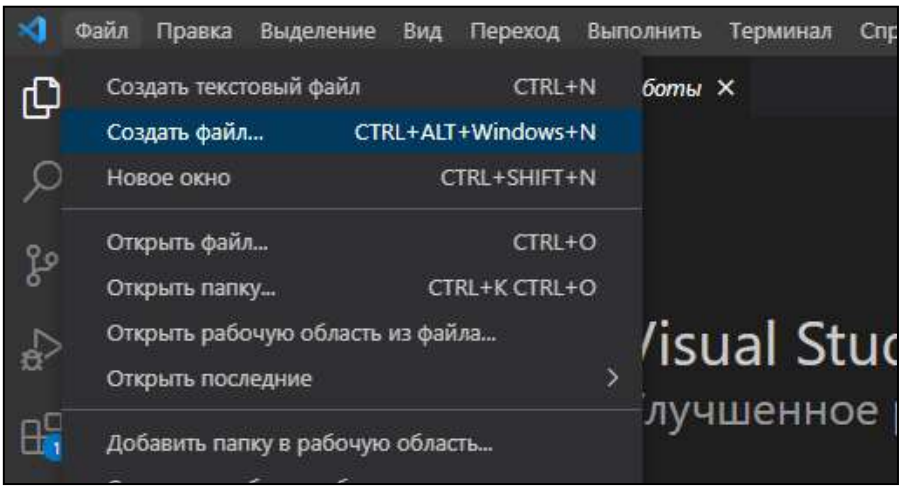


Рис. 2.20. Создание нового файла

Visual Studio Code создаст новую вкладку, где предложит выбрать язык программирования (или разметки), для которого нужно включить подсветку синтаксиса:

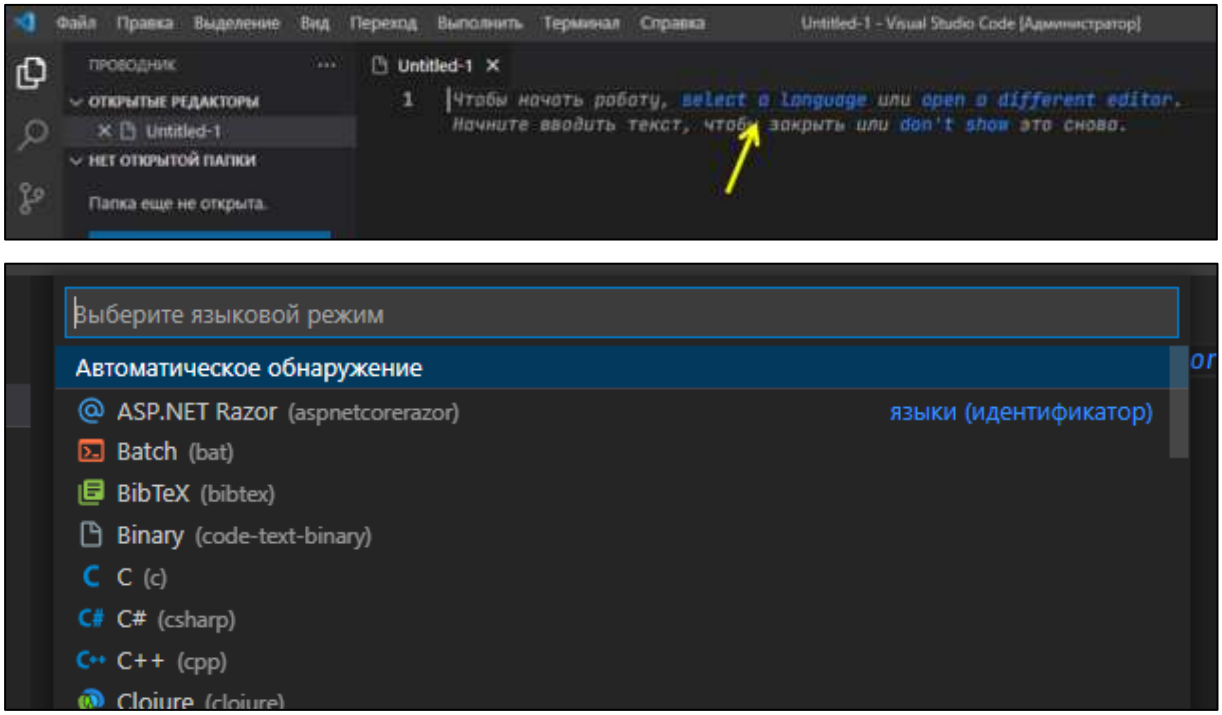


Рис. 2.21. Выбор языка для подсветки синтаксиса его команд

Из предложенного списка выберете HTML: в текущей вкладке редактор переключится в режим подсветки команд этого языка, а на вкладке появится значок, символизирующий HTML:

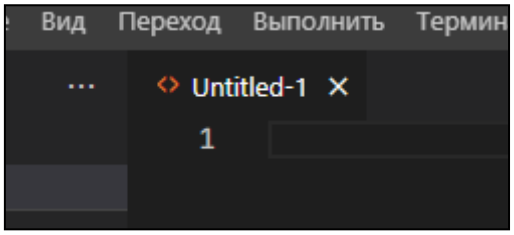


Рис. 2.22. Выбор языка подсветки синтаксиса

Далее в процессе работы вы можете изменить язык подсветки вне зависимости от того, является ли это обычный текстовый файл или листинг кода на определенном языке. Для этого в нижней панели нажмите на кнопку *Выберите языковой режим* и укажите язык:

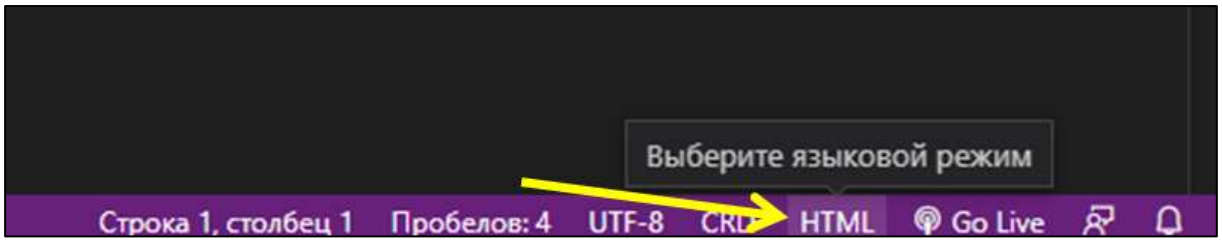


Рис. 2.23. Смена режима подсветки

Осуществите ввод следующего кода HTML-разметки (не используя пока режимы автоматического дополнения):

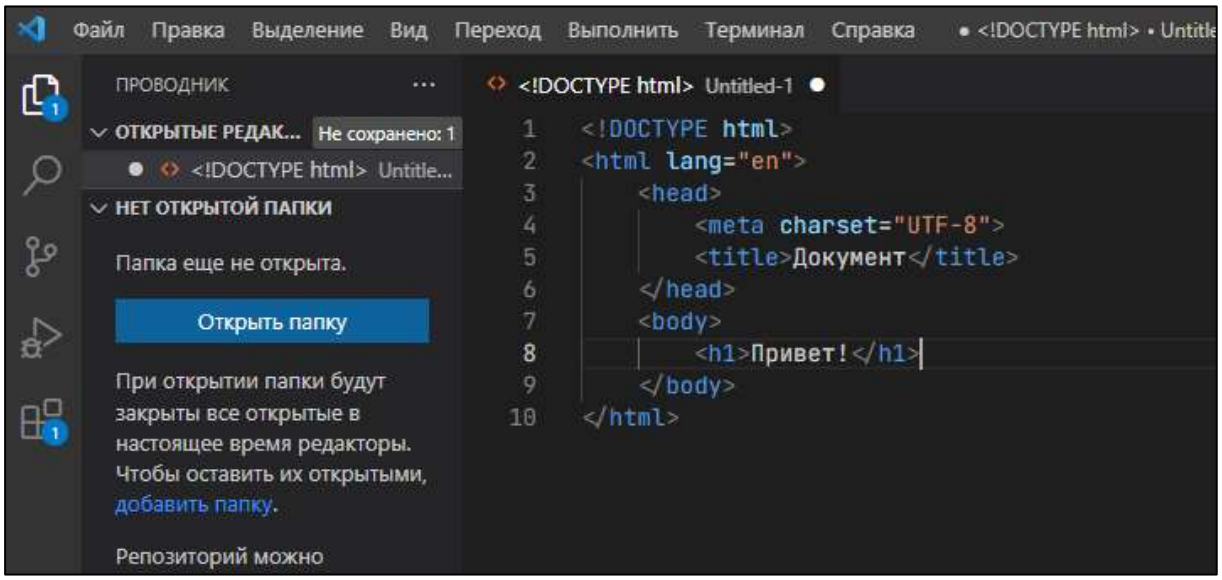


Рис. 2.24. Ввод кода разметки

Сохранение файла

Поскольку файл еще не сохранен, редактор на иконке *Проводника* отображает кружок с числом «1». Это означает, что имеется одна вкладка с несохраненными изменениями. Кроме того, на вкладке вместо крестика также отображается круг:

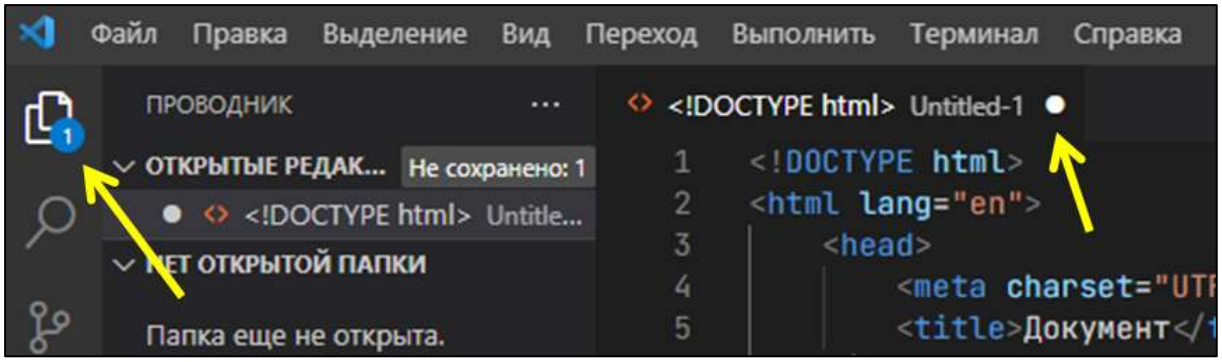


Рис. 2.25. Файл еще не сохранен

Сохраните файл одним из двух способов:

- *Файл / Сохранить*;
- комбинацией клавиш *CTRL + S*.

Если файл не был сохранен ранее, то в появившемся диалоговом окне укажите его название и каталог на вашем ПК, где требуется сохранить новый документ. При необходимости здесь также можно поменять и его формат в разделе *Тип файла*:

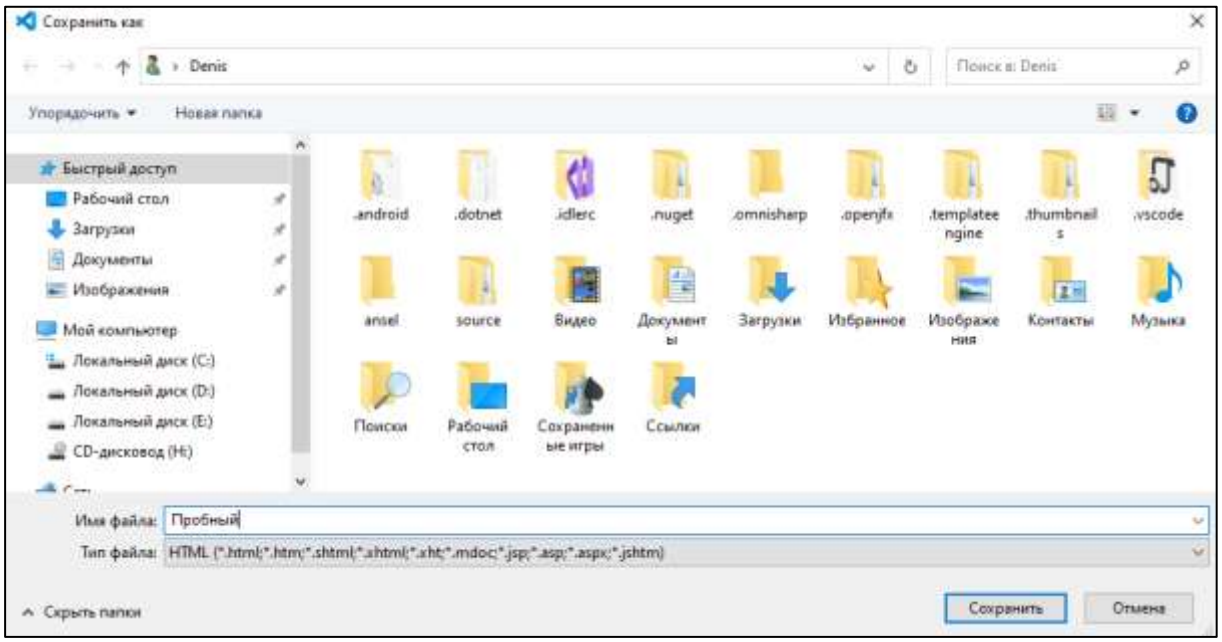


Рис. 2.26. Сохранение файла

Открытие созданных файлов

Открыть созданный файл также можно несколькими способами:

- с помощью меню *Файл / Открыть файл…*;
- путем перетаскивания файла в окно редактора:

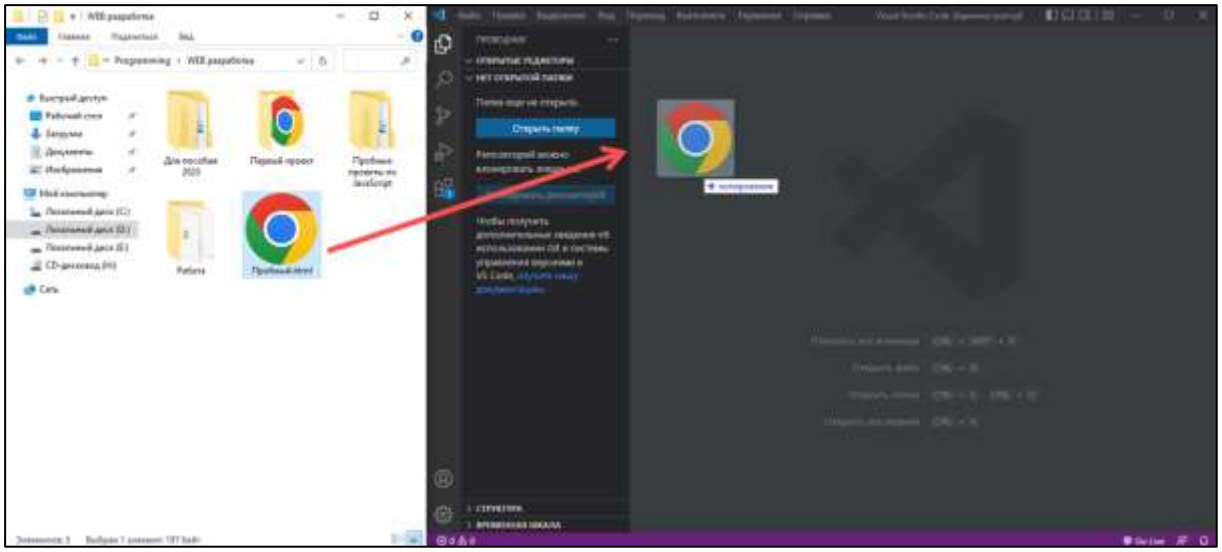


Рис. 2.27. Открытие файлов в редакторе

Если вы редактируйте HTML-файл, то его вкладку можно также перетащить и в окно браузера, чтобы увидеть веб-страницу с заданной разметкой:

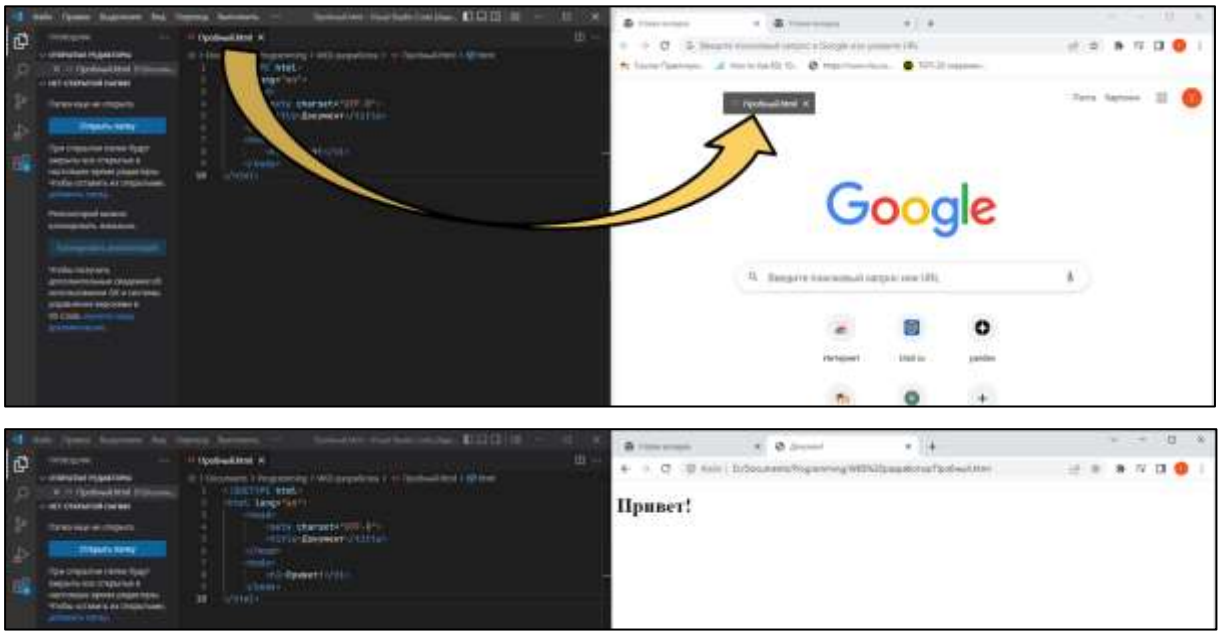


Рис. 2.28. Перетаскивание HTML-документа для просмотра в браузере

Настройка редактора

Основные

Настройки редактора осуществляются по нажатию на кнопку *Управление* в левой панели меню:

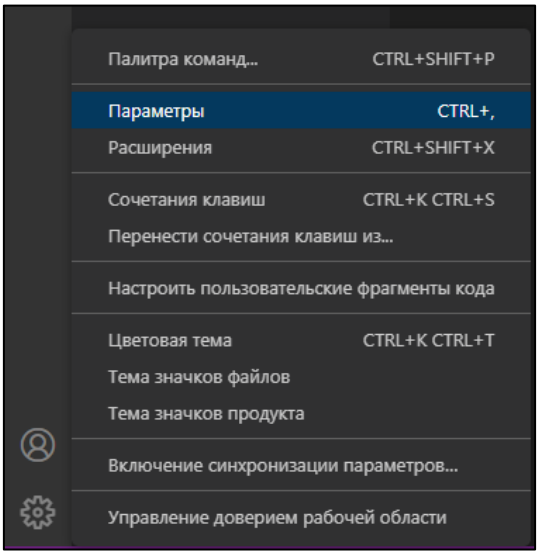


Рис. 2.29. Настройка редактора

Основным пунктом меню настроек является *Параметры*. В открывшейся вкладке многочисленные параметры интерфейса и функционирования редактора для удобства разбиты по категориям:

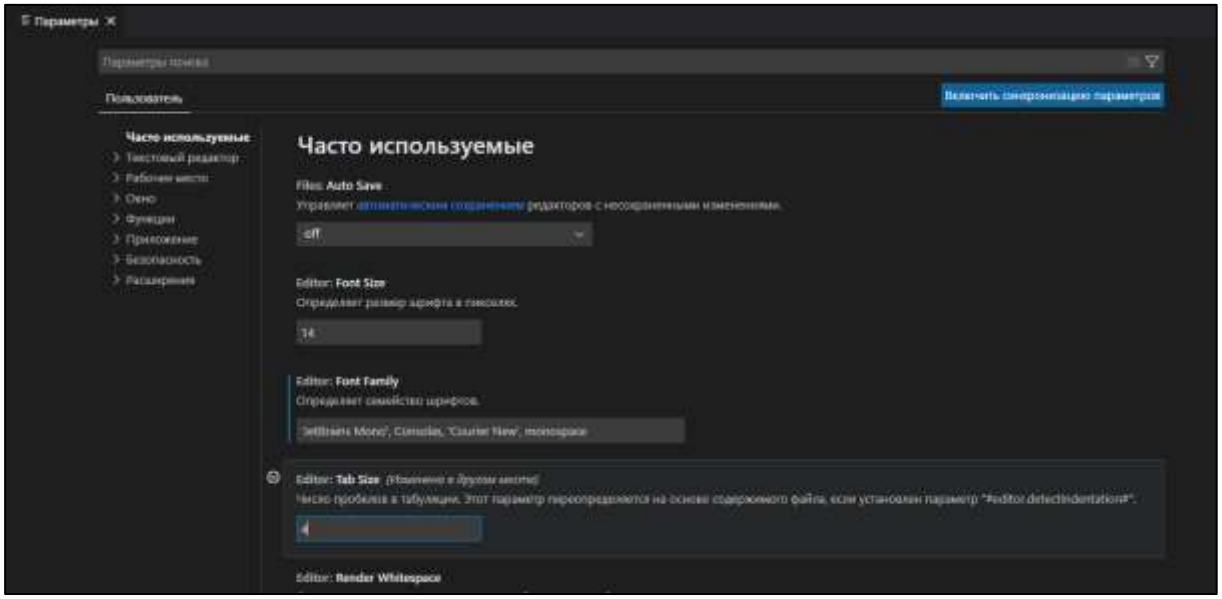


Рис. 2.30. Управление параметрами редактора

С другой стороны, все перечисленные настройки редактор хранит в служебном текстовом JSON-файле. Изменения параметров, которые вносятся (переопределяются) пользователем, можно просмотреть, нажав на кнопку *Открыть параметры (JSON)*:

Рис. 2.31. JSON-файл с настройками

Приемы работы с Visual Studio Code

Чтобы увеличить / уменьшить масштаб интерфейса в окне редактора, зажмите комбинацию клавиш ***Ctrl*** *и* **+ / -**.

Другие пункты меню настройки

Пункт *Палитра команд* вызывает окно для поиска и выполнения некоторой команды.

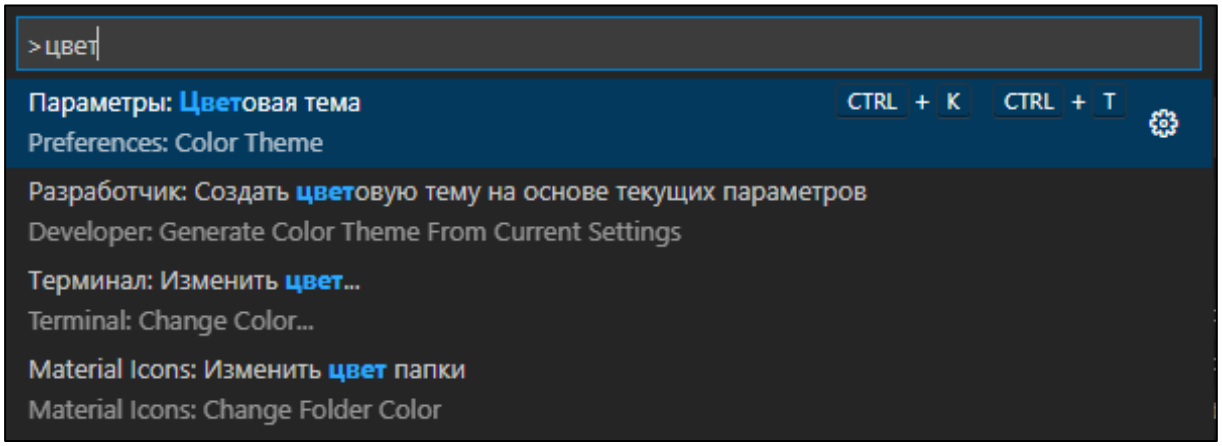


Рис. 2.32. Работа с палитрой команд

Пункт *Расширения* переходит к окну поиска и установки дополнительных расширений.

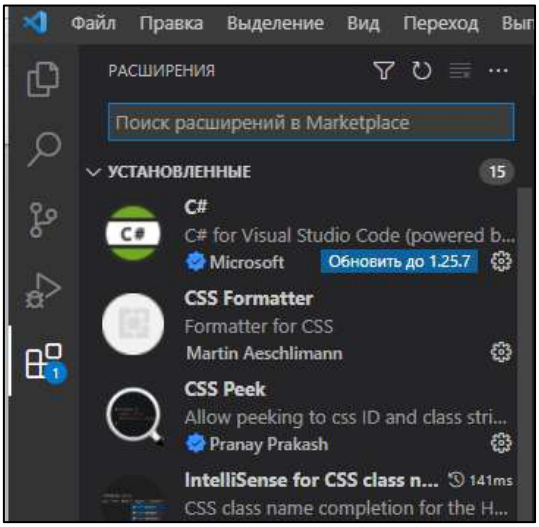


Рис. 2.33. Дополнительные расширения

Пункт *Сочетания клавиш* позволяет настроить комбинации клавиш для многочисленных функций редактора.

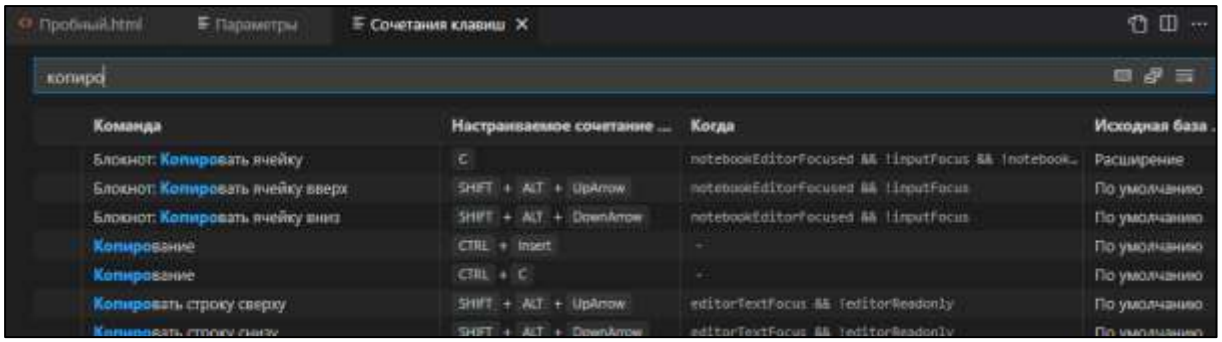


Рис. 2.34. Настройка комбинации горячих клавиш

Пункт *Цветовая тема* позволяет выбрать общее цветовое оформление окна редактора и подсветки кода. Обычно их делят на светлые и темные.

При желании пользователь может подобрать не только встроенные темы, но и доступные в сети:

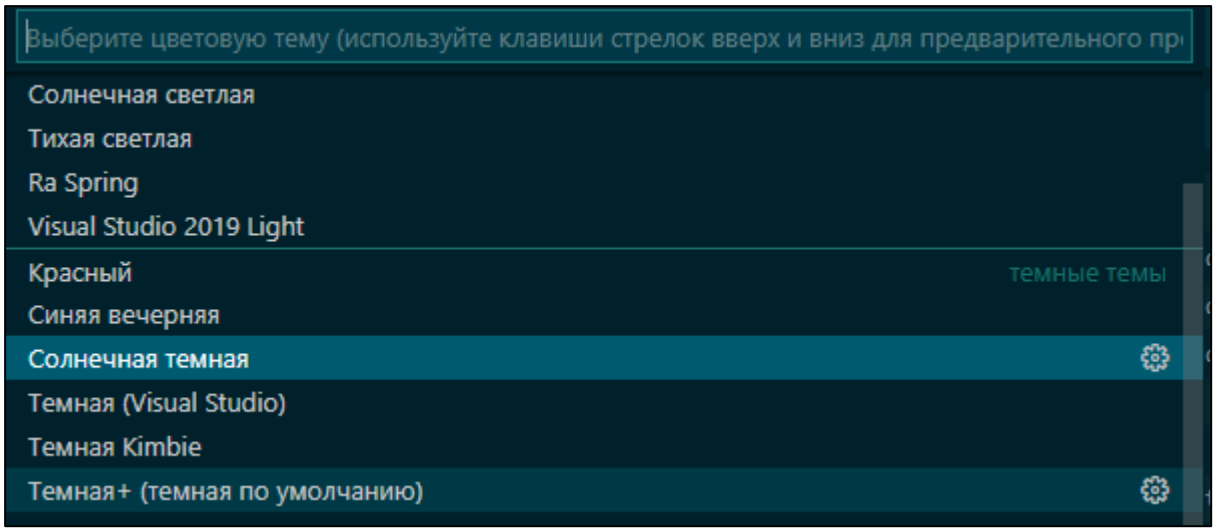


Рис. 2.35. Выбор цветовой темы оформления и подсветки

Параметр *Тема значков файлов* меняет иконки для обозначения файлов разных форматов (позволяет визуально отслеживать их в проводнике). Чтобы сменить их, нажмите *Установить дополнительные темы значков файлов* и подберите понравившуюся (используйте стрелки клавиатуры).

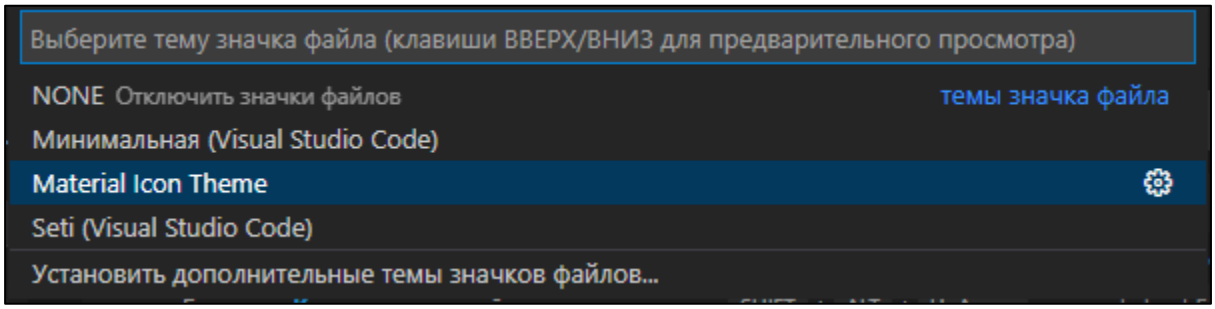


Рис. 2.36. Настройка значков файлов

Цветовая тема

Выбор цветовой темы интерфейса и окна редактора обычно является делом вкуса разработчика. И зачастую многие делают выбор в пользу темной темы. Во многом эта традиция была связана особенностью работы предыдущих поколений мониторов (ЭЛТ), которые давали заметный глазу эффект мерцания на белом фоне.

Для современного поколения жидкокристаллических мониторов эта проблема нивелирована. Поэтому при выборе цветовой гаммы следует учитывать ряд особенностей.

Так, для темной темы:

- напряжение глаз в целом меньше, особенно при условиях низкой освещенности помещения;
- оказывается меньшее влияние на нарушение цикла сна;
- лучше фокусировка на содержании;
- более эстетичный вид.

Для светлой темы:

- более четкое восприятие контраста между текстом и фоном;
- отсутствие засвета, как на темном фоне (если источник света направлен на монитор);
- в отличие о темной темы, светлая меньше побуждает пользователя работать в ночное время, т.е. не перенапрягать глаза чрезмерной ежедневной нагрузкой.

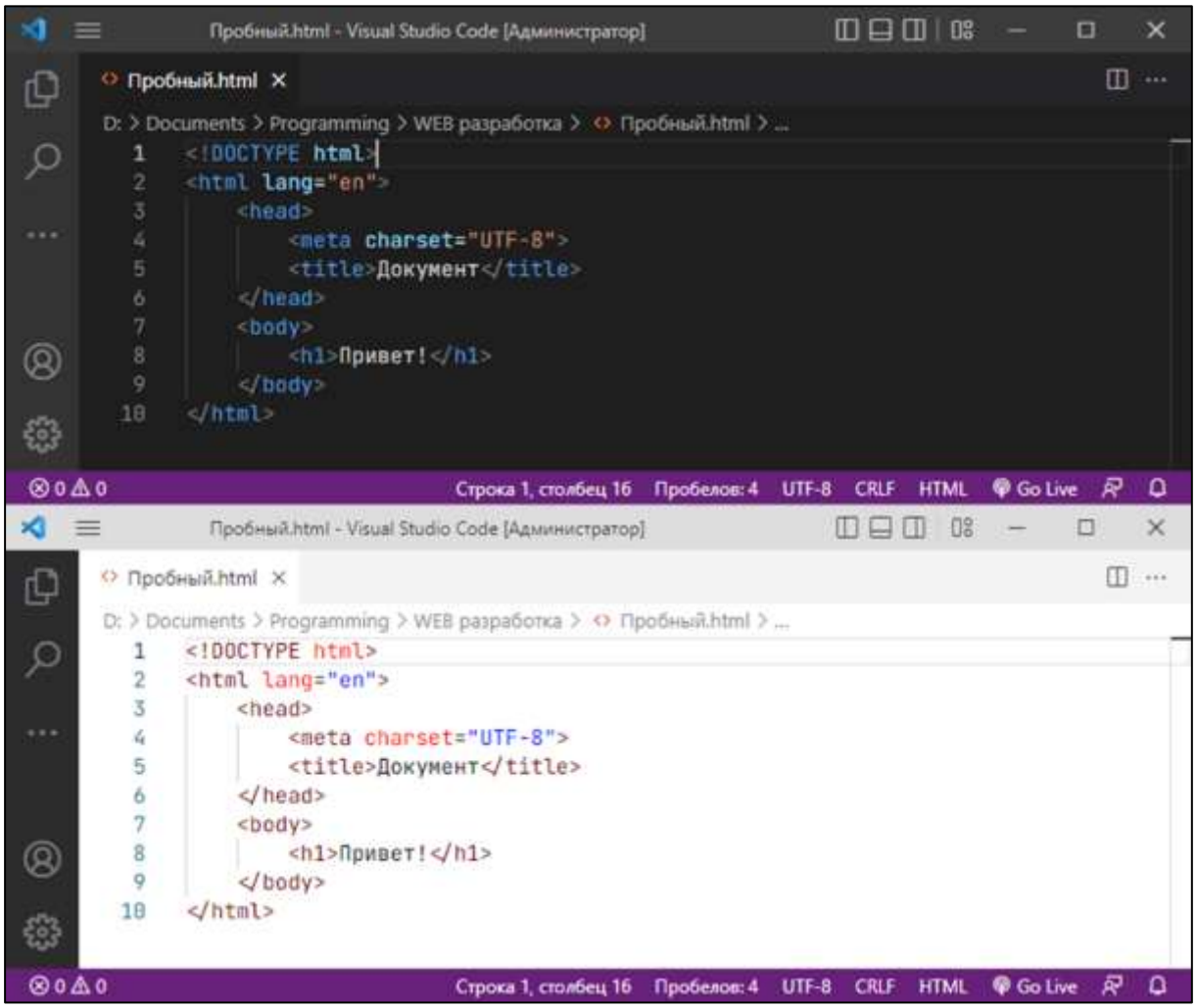


Рис. 2.37. Разница между темной и светлой темой

Загрузка корневого каталога

Обычно веб-проект состоит из множества файлов (разметки, стилей, скриптов, графических файлов и т.д.) и вложенных каталогов. Поэтому все файлы необходимо сгруппировать в единый каталог. Кроме того, когда разработчик загружает в редактор каталог, что для него формируется рабочее окружение, которое активирует дополнительный функционал.

Создайте новый каталог (папку) и перетащите его в область редактора:

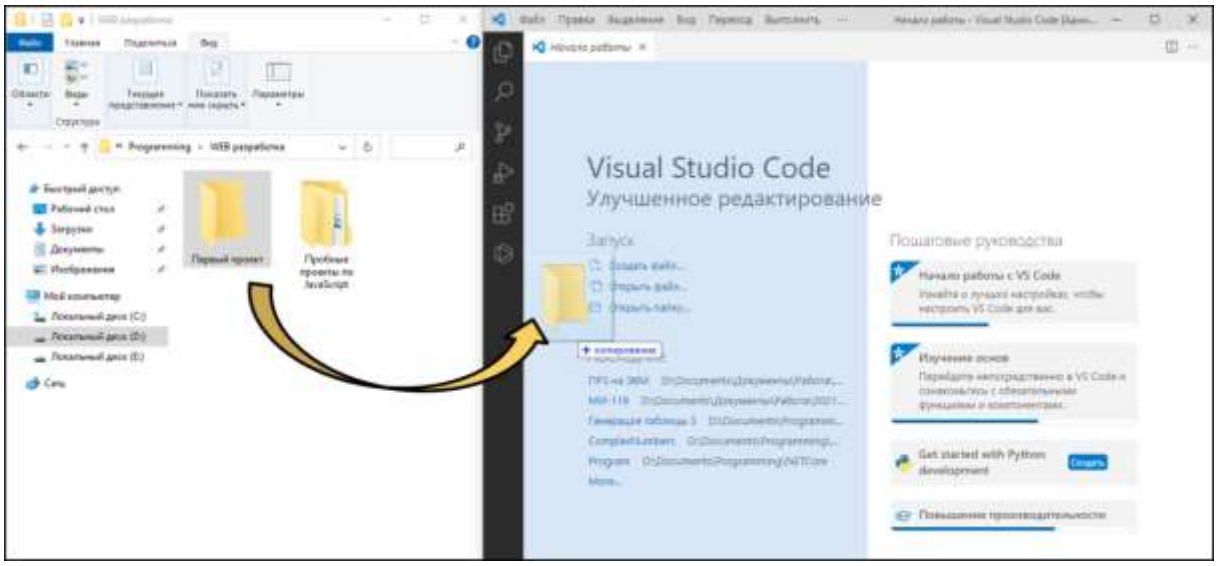


Рис. 2.38. Загрузка рабочего каталога в Проводник

Созданный каталог отобразится в *Проводнике*:

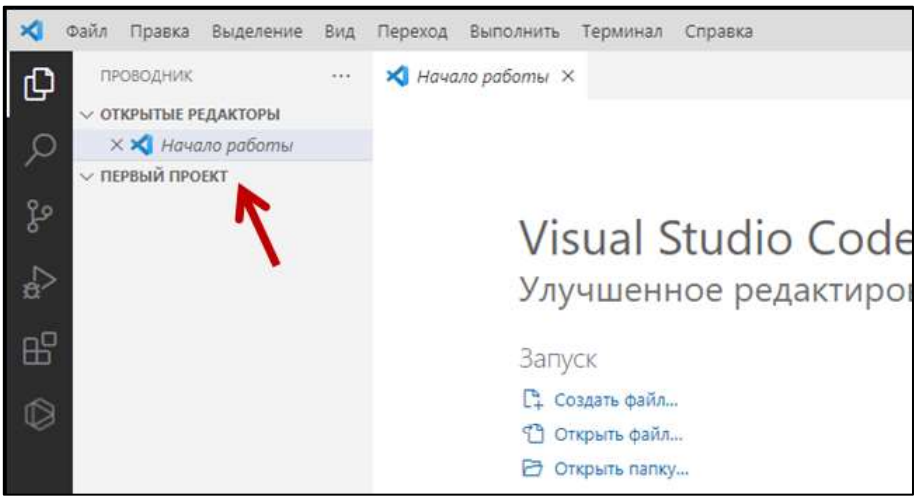


Рис. 2.39. Каталог проекта

Обратите внимание, что цвет нижней панели редактора изменился с фиолетового на синий: это говорит о том, что в редактор загружен каталог.

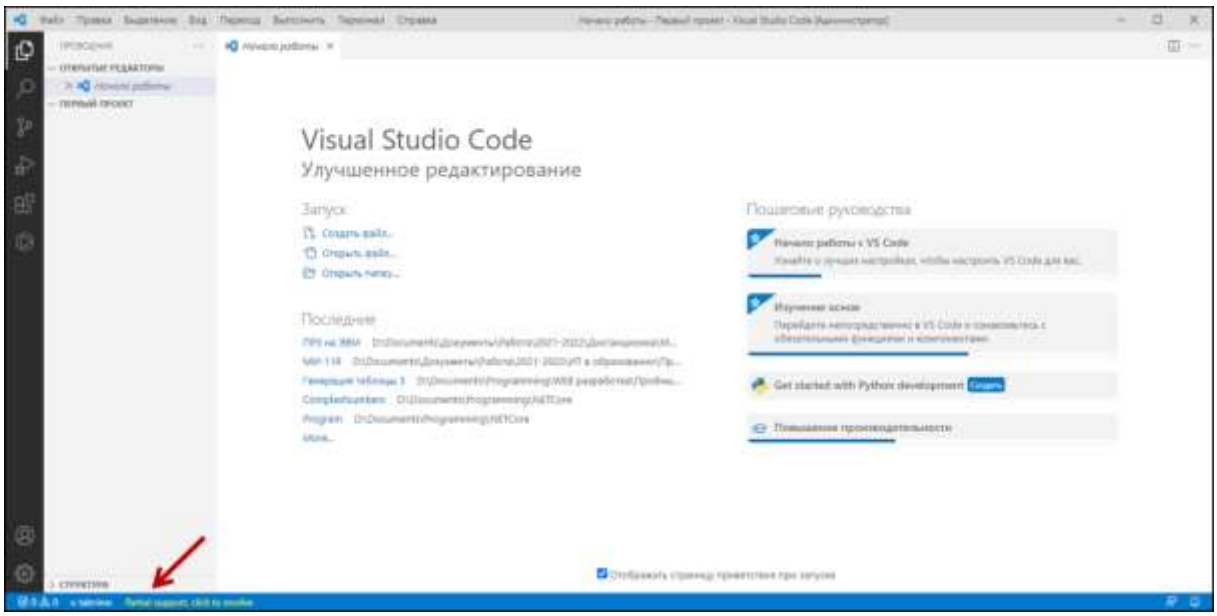


Рис. 2.40.

Загруженный каталог является корнем проекта. В нем допускается создание и редактирование файлов и каталогов. При этом в процессе создания нового файла можно указать его формат в названии, Visual Studio Code автоматически переключится в режим подсветки синтаксиса искомого языка.

Примечательно, что любые изменения в проводнике редактора напрямую влияют на файлы этого каталога, которые хранятся на жестком диске ПК (равно как и обратно).

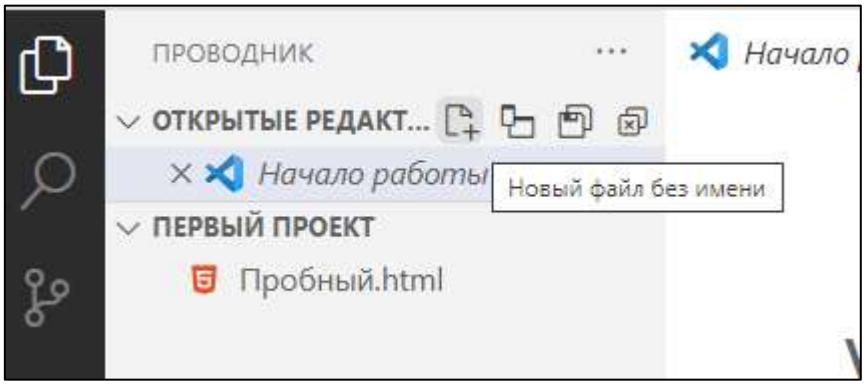


Рис. 2.41. Кнопки для управления файлами

Дополнительные кнопки доступны в контекстном меню *ЛКМ*.

Режим IntelliSense

Начальный шаблон HTML-файла

Одно из главных достоинств Visual Studio Code – поддержка автозавершения кода. Для HTML, CSS и JavaScript эта возможность уже встроена в редактор и не требует установки плагинов.

Загрузите пустой каталог в редактор. Создайте новый HTML-файл. Начните ввод команды *html*, сработает выпадающая подсказка. Выберете (стрелками клавиатуры или мышкой) пункт *html:5*, нажмите клавишу *TAB* или *Enter*, редактор вставит фрагмент разметки (*сниппет*):

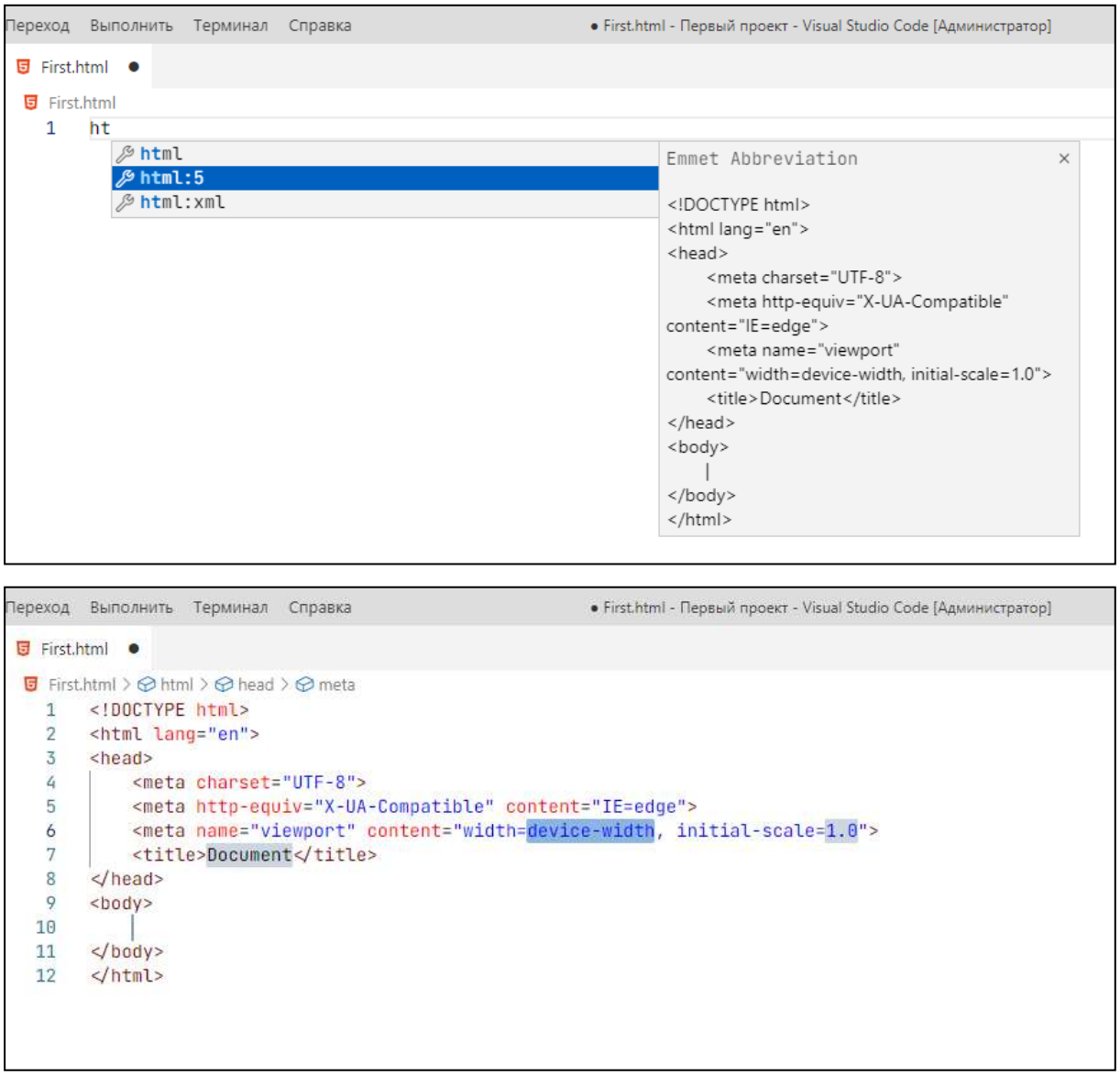


Рис. 2.42. Режим IntelliSense: вставка начальной разметки документа

Автозавершение тегов

Перейдите в раздел `<body></body>`.

Откройте скобку < тега и начните набор первых букв его названия. Нажмите *TAB* (*Enter*) для автозавершения, закройте скобку > тега и редактор автоматически допишет парный закрывающий тег:

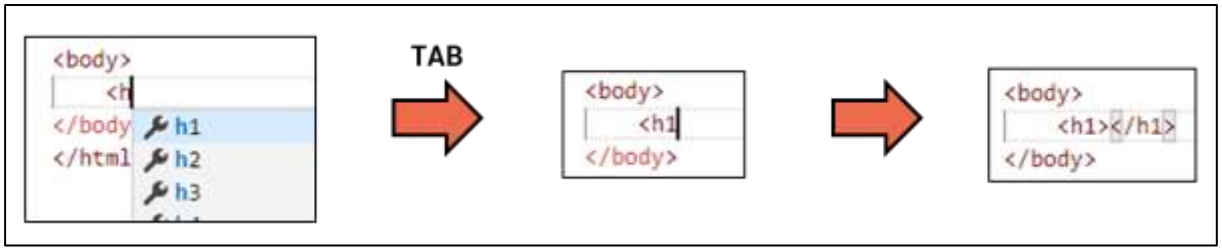


Рис. 2.43. Автозавершение тегов

Более эффективный и быстрый подход к набору тегов основан на плагине Emmet (подробнее в п. 1.5.3 и далее в курсе). Начните ввод имени тега без угловой скобки, стрелками клавиатуры подберите нужный и нажмите клавишу *TAB* (*Enter*):

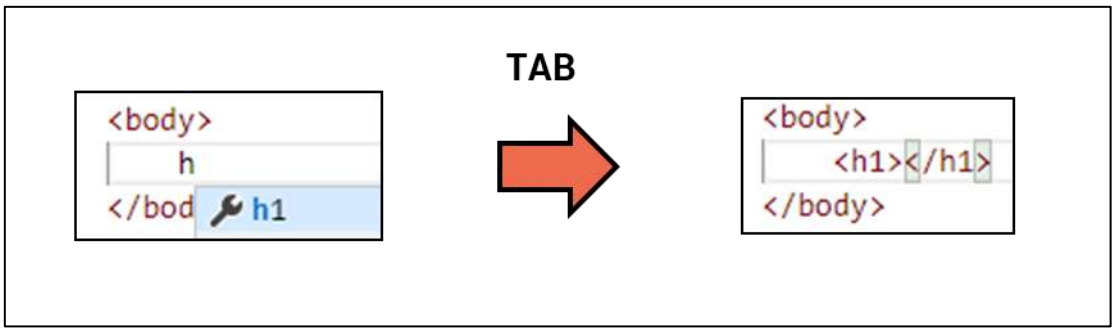


Рис. 2.44. Автозавершение тегов с помощью возможностей плагина Emmet

Важное замечание!

Используйте именно этот подход! Он существенно быстрее и должен быть отработан до автоматизма.

Приемы работы с Visual Studio Code

Если окно подсказки вдруг пропало, нажмите ***Ctrl*** *+* ***Пробел****.*

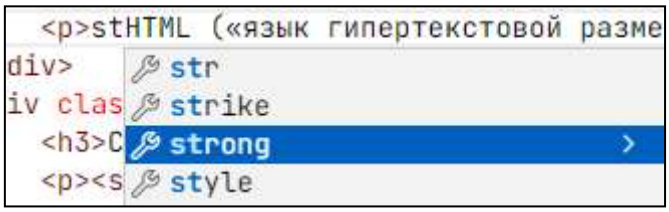


Рис. 2.45. Восстановление окна подсказки

Процедура верстки и просмотр результата

Обычный подход

При верстке веб-страниц обычно удобнее сразу просматривать полученный результат в окне браузера.

После любого изменения код необходимо сохранять (*CTRL + S*), а веб-страницу – обновлять (*F5*).

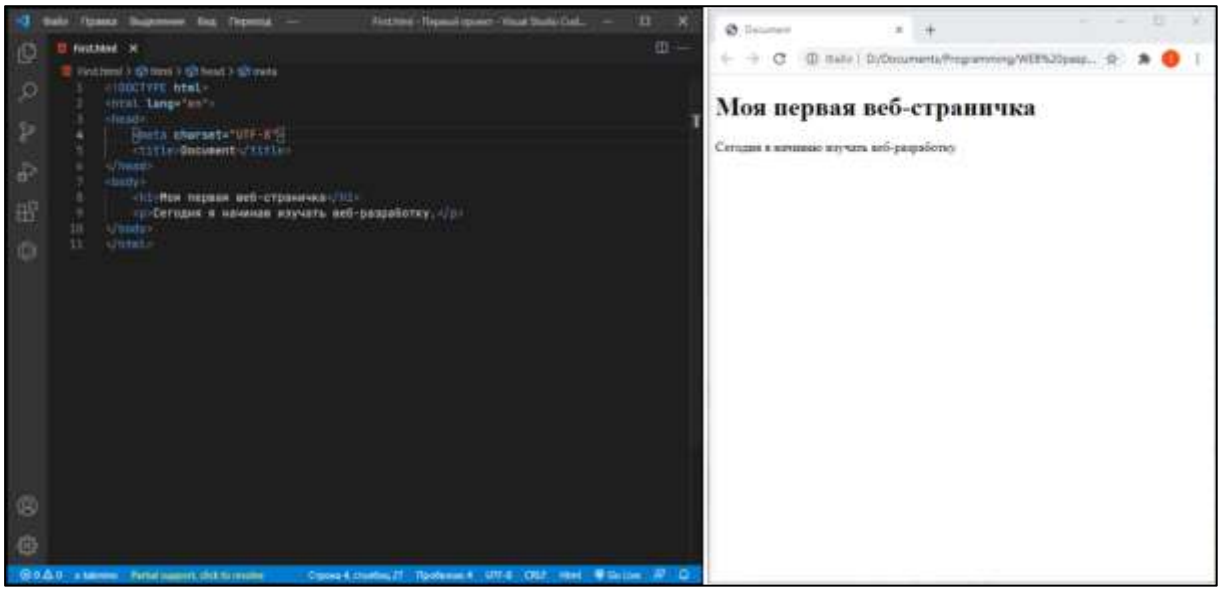


Рис. 2.46. Процесс верстки веб-страницы и просмотр результата в браузере

Расширение Live Server

Обновлять веб-страницу после каждого изменения кода разметки крайне неудобно.

Чтобы автоматизировать процесс обновления, установите расширение *Live Server*. Оно эмулирует работу локального сервера на компьютере.

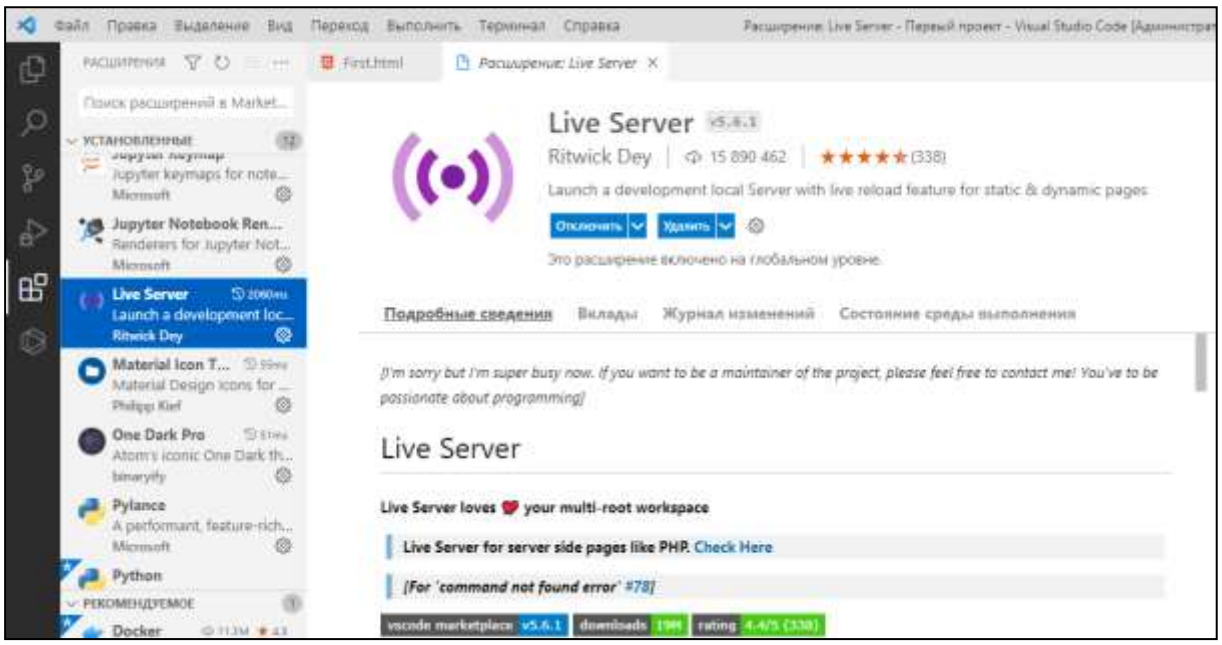


Рис. 2.47. Установка плагина Live Server

Для активации плагина в HTML-файле нажмите *ПКМ / Open with Live Server*. Страница откроется в браузере:

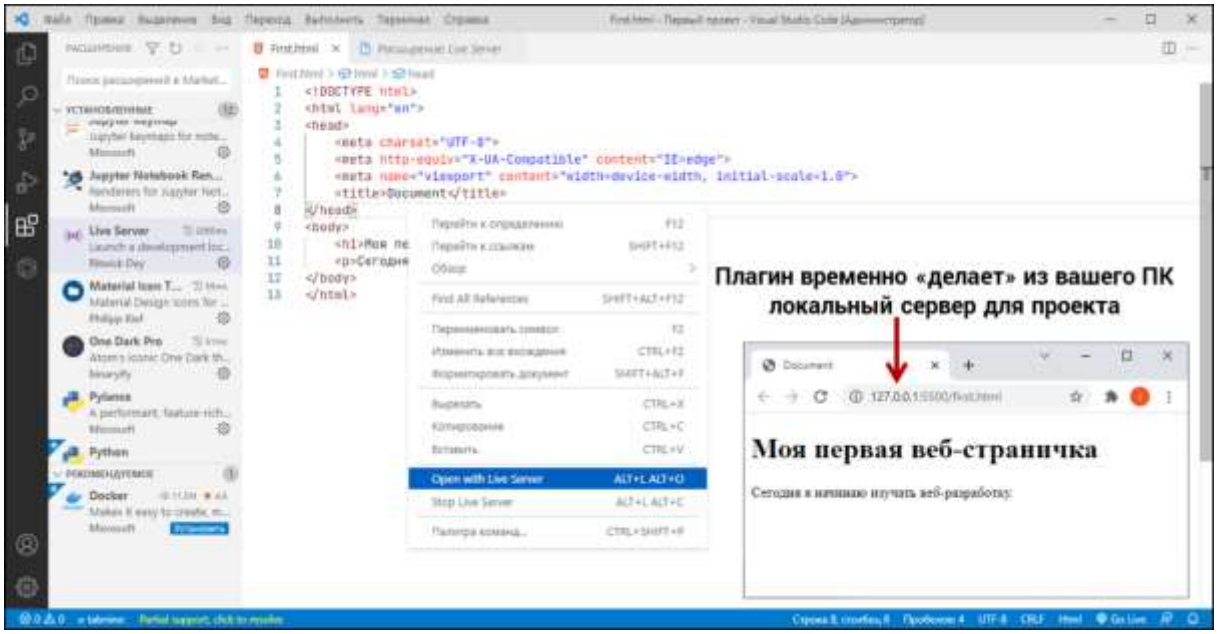


Рис. 2.48. Запуск страницы под эмуляцией локального сервера ПК по IP-адресу 127.0.0.1:500

Теперь после изменения HTML-кода файл достаточно сохранить и веб-страница обновится автоматически:

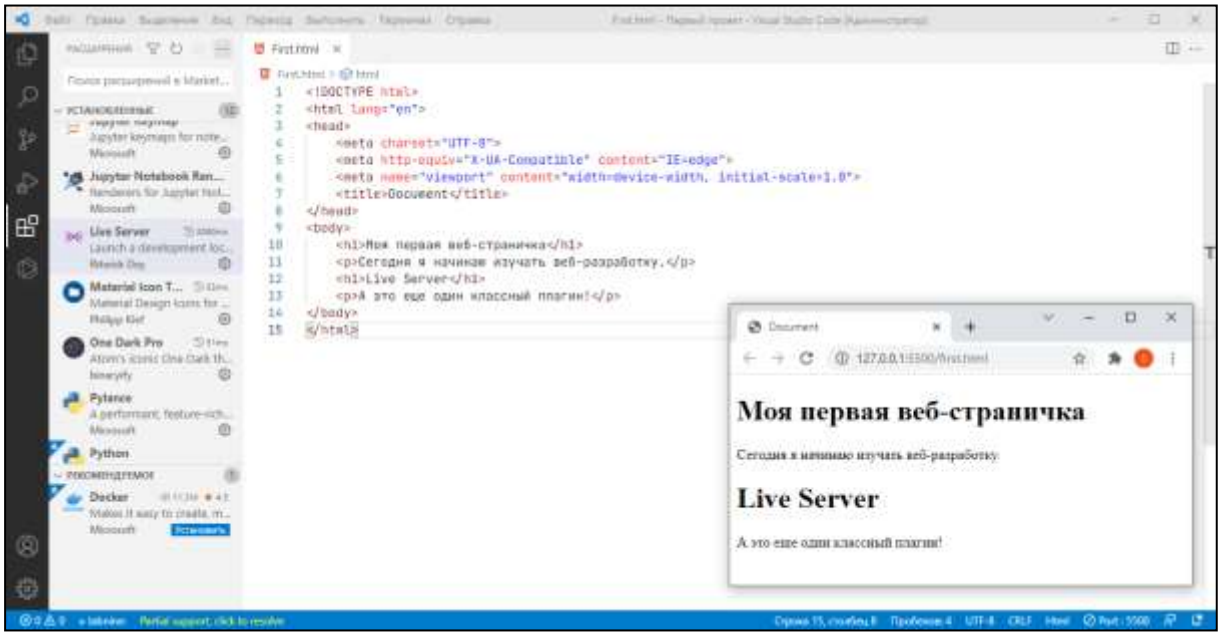


Рис. 2.49. Live Server обновит страницу сразу после сохранения файла

Если работа с проектом завершена, его каталог следует выгрузить из редактора командой *Файл / Закрыть папку*:

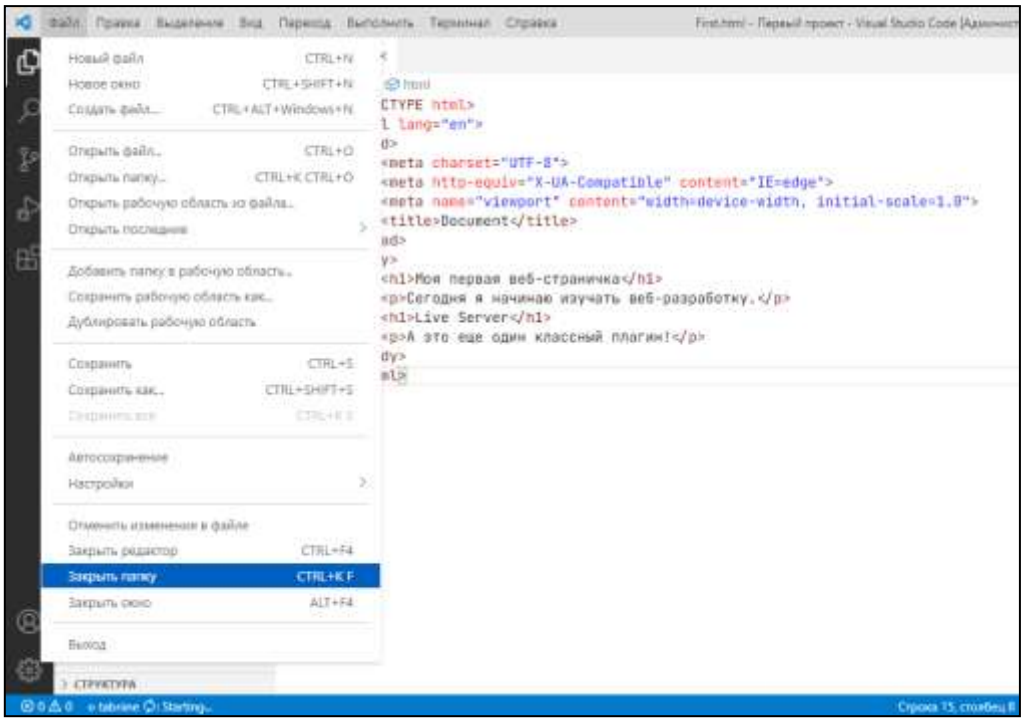


Рис. 2.50. Выгрузка проекта

Это полезно знать!

Читатель наверняка замечал, что веб-разработчики обычно используют несколько мониторов. Даже в настоящее время, когда появились широкоформатные мониторы, разработчику удобно использовать 2 или даже 3 монитора. Они позволяют визуально разграничить работу с системой разработки ПО, графическим интерфейсом ресурса, информацией об отладке и тестировании.

Особенно это актуально для веб-разработки, когда на одном экране открыт редактор кода, а на другом отображается дизайн формируемой веб-страницы. Тем более что современные операционные системы позволяют настроить рабочий стол ПК по желанию пользователя.

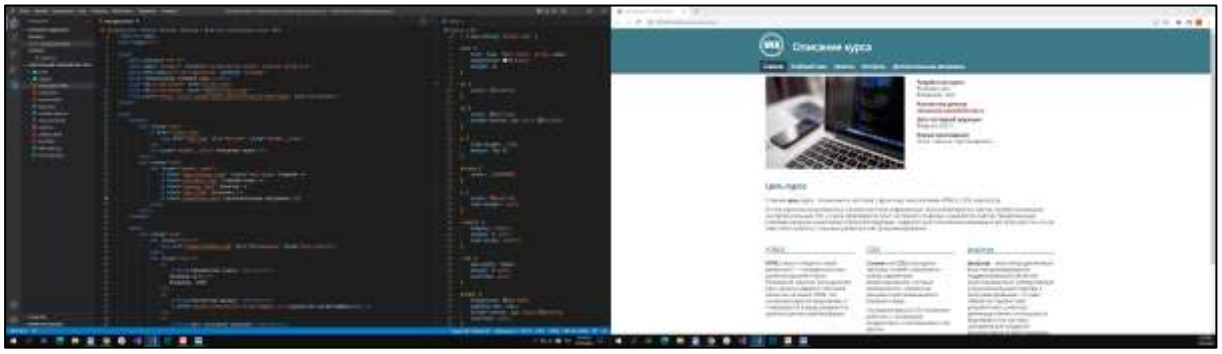


Рис. 2.51. Левый экран используется для верстки сайта, правый – для просмотра текущего результата

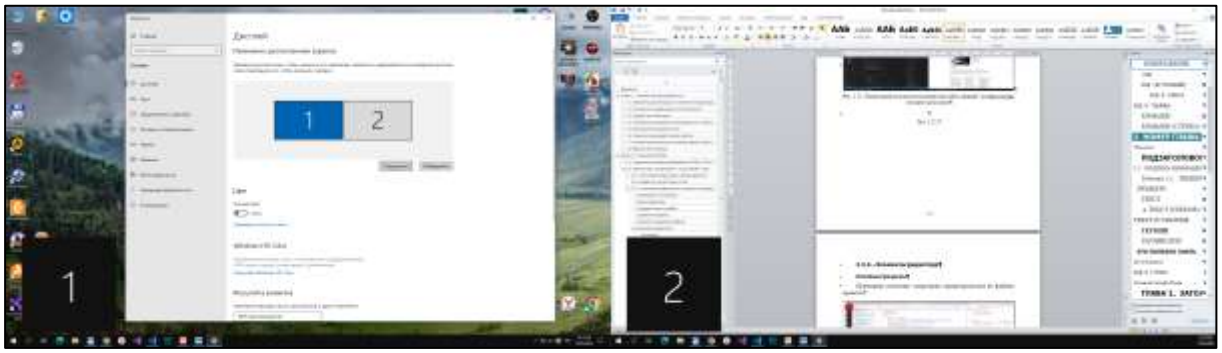


Рис. 2.52. В настройках экрана можно уточнить порядок отображения содержимого на рабочем столе: обычно выбирается режим расширения на несколько экранов

2.2.4. Элементы редактора

Основные разделы

Проводник позволяет оперативно ориентироваться по файлам проекта. Он отображает структуру документа и отдельно открытые в редакторе вкладки:

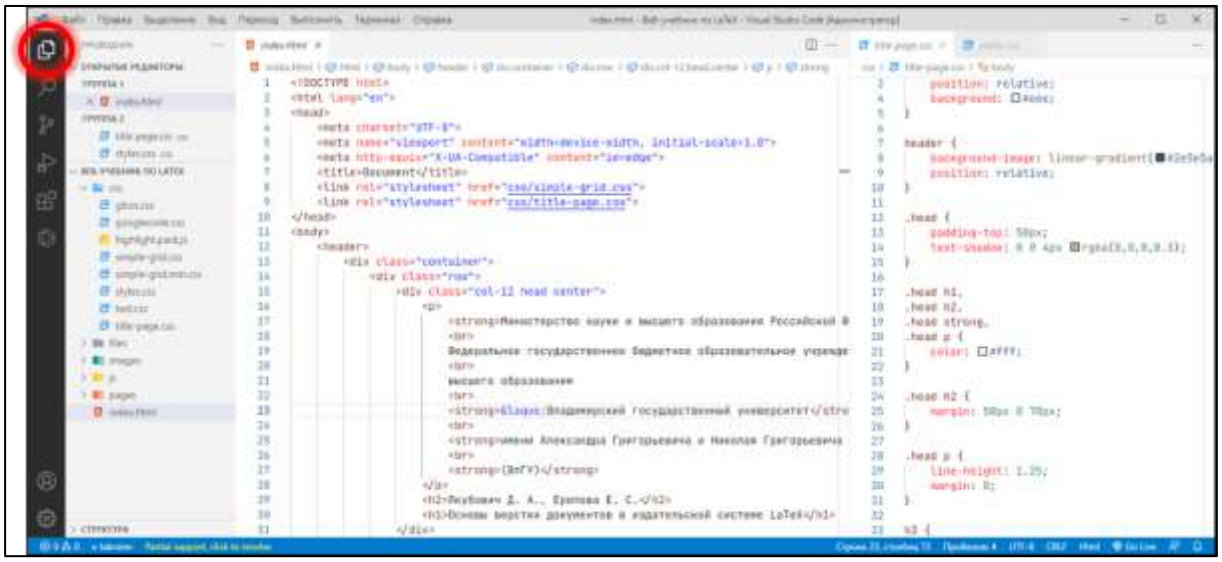


Рис. 2.53. Редактор VSC: проводник проекта

Поиск поможет найти совпадения по тексту и заменить его во всех файлах:

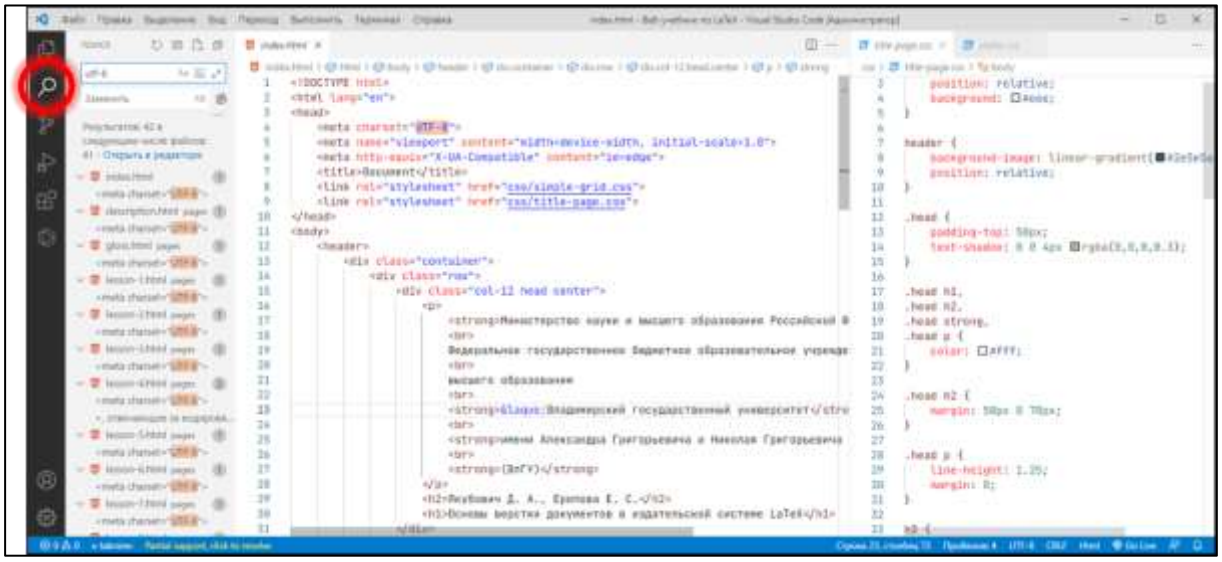


Рис. 2.54. Редактор VSC: поиск и замена

Система управления версиями позволяет контролировать этапы и разные версии разработки программного продукта. VSC может быть синхронизирован с Git:

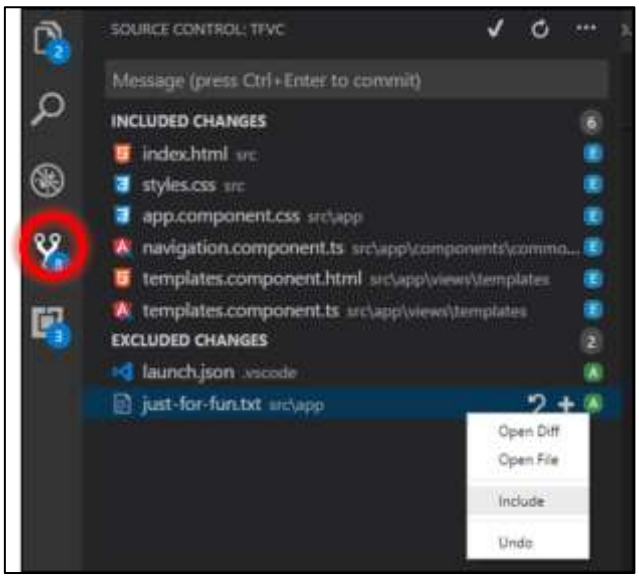


Рис. 2.55. Редактор VSC: контроль версий по технологии Git

Запуск и отладка позволяет осуществлять пошаговую отладку программного кода: устанавливать контрольные точки, отображать содержимое переменных, стека, пошагово выполнять инструкции и т.п. В зависимости от языка программирования может требоваться установка дополнительных расширений:

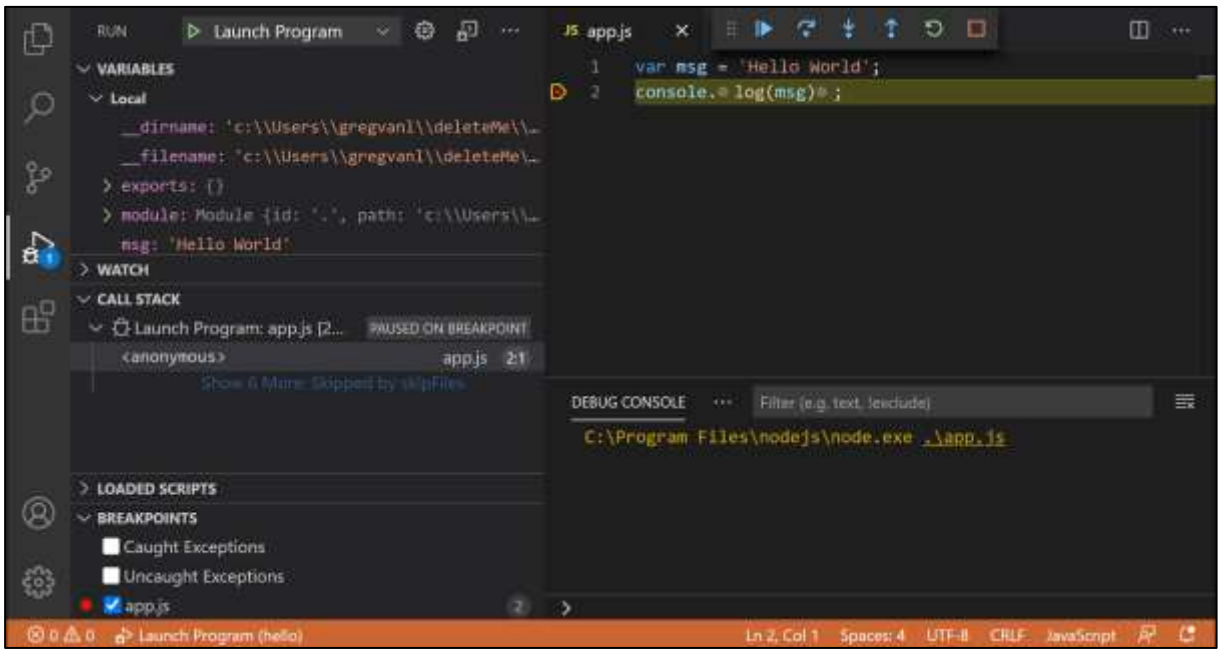


Рис. 2.56. Редактор VSC: окно отладки кода

Расширения – дополнительные плагины из сети Интернет, повышающие эргономику редактора. При активации отображается список уже установленных расширений. По каждому расширению представлена краткая информация, оценка пользователей и ссылка на сайт разработчика:

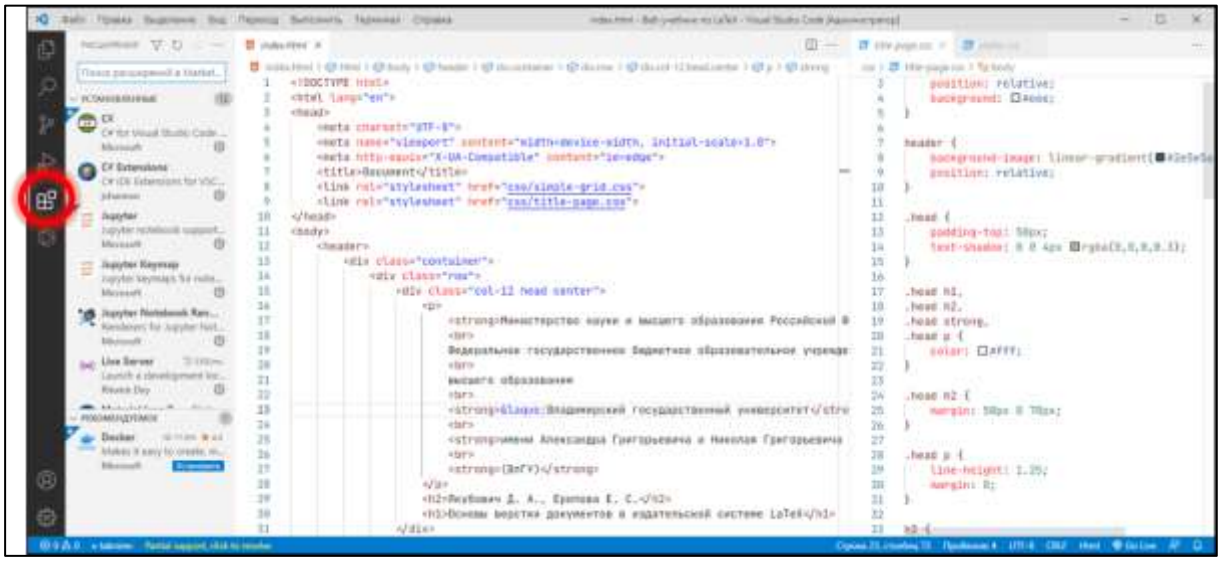


Рис. 2.57. Редактор VSC: дополнительные расширения

Дополнительные элементы, значки которых могут добавляться пользовательскими плагинами:

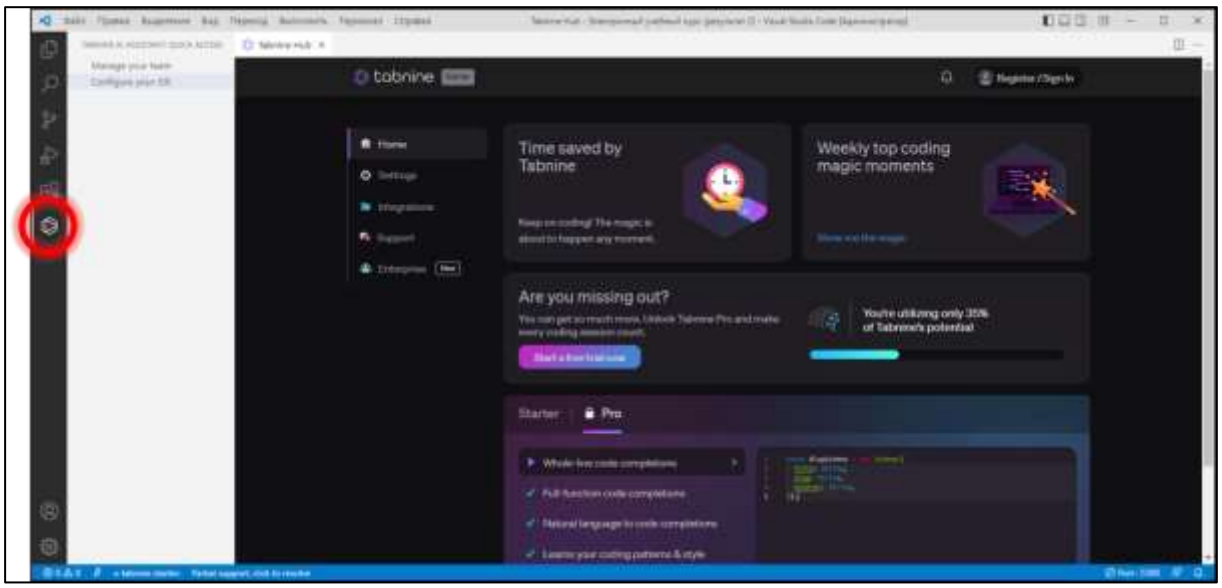


Рис. 2.58. Редактор VSC: дополнительные элементы боковой панели

Окно редактора

Проводник – отображает файлы и каталоги проекта. Позволяет перемещать файлы зажатой *ЛКМ*. В верхней части отображены открытые вкладки.

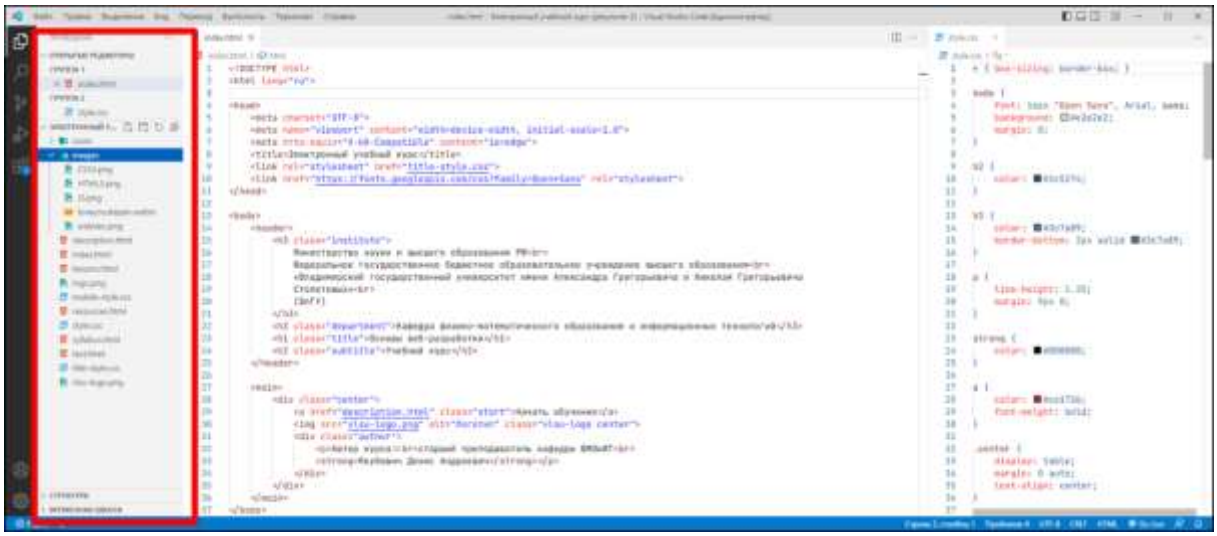


Рис. 2.59. Редактор VSC: панель проводника

Колонки – это отдельные области редактора, в которых допускается параллельное редактирование или просмотр файлов. Для веб-разработки деление редактора на несколько колонок крайне удобная возможность: это позволяет отдельно редактировать код разметки, стилей и скриптов, визуально отслеживая связанные элементы.

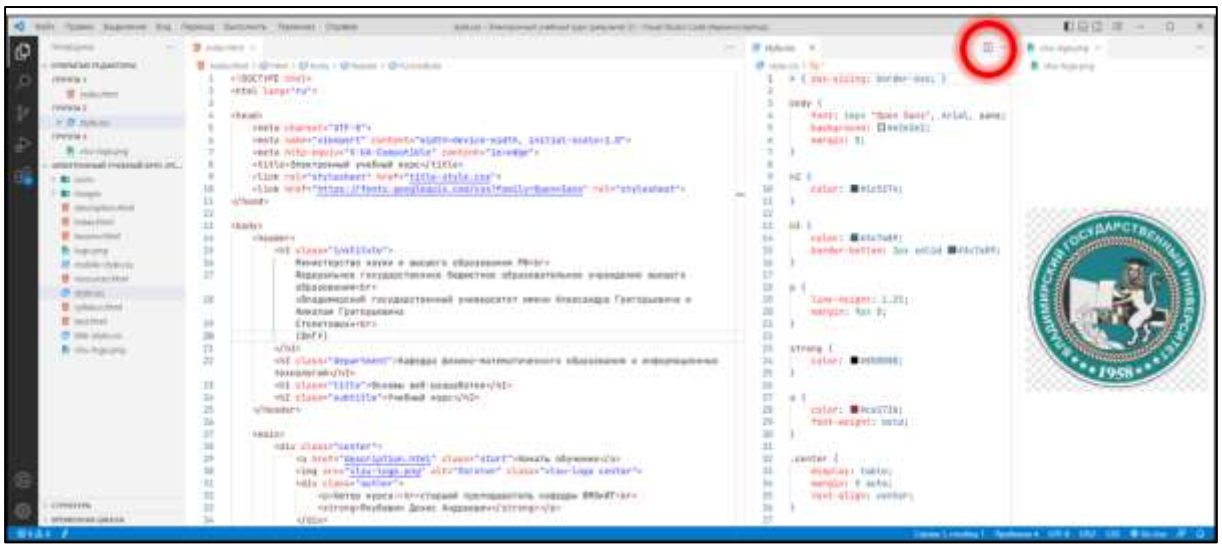
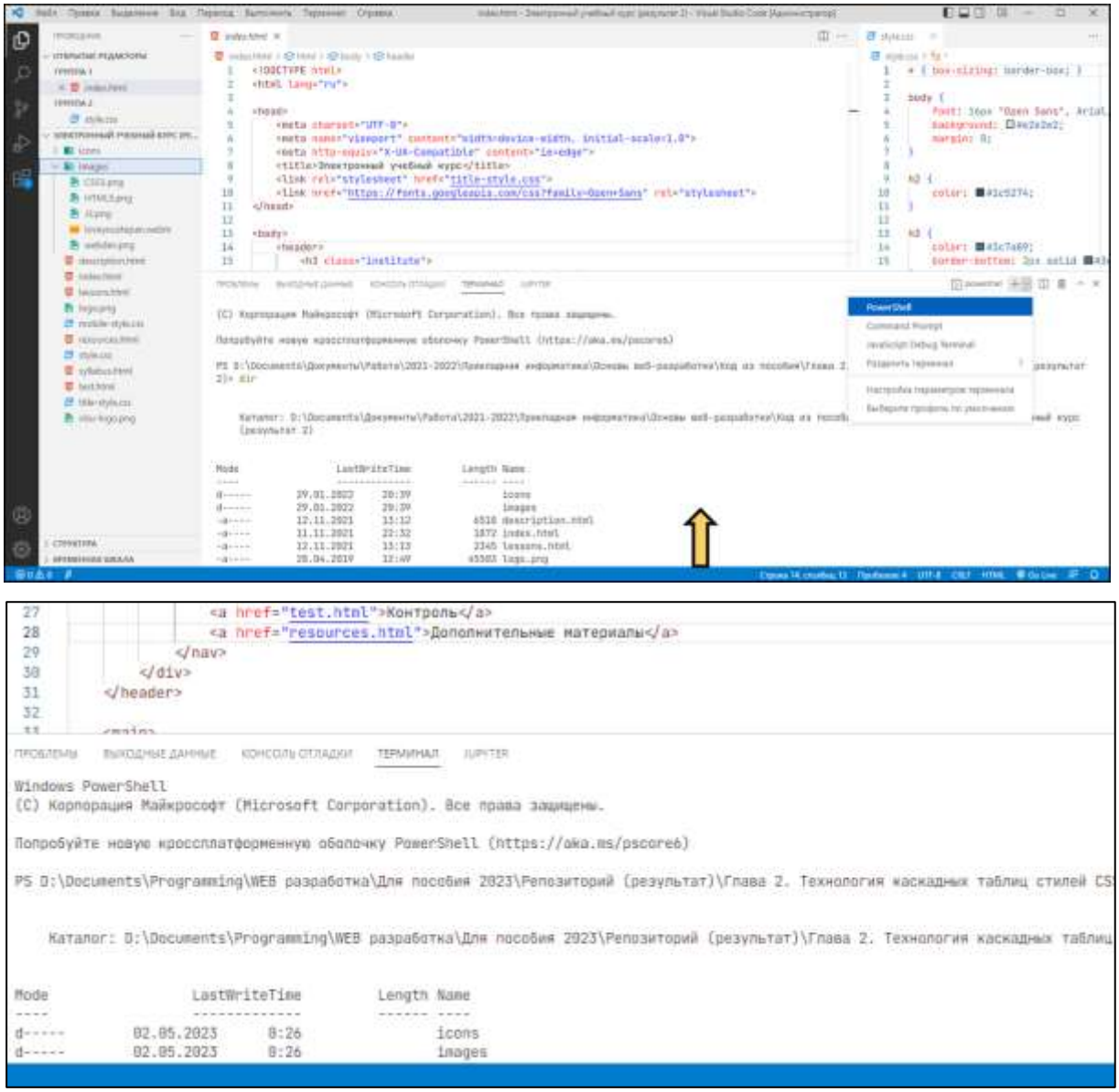


Рис. 2.60. Редактор VSC: деление области редактора на вертикальные колонки (в одной может быть открыто несколько файлов)

Нижняя панель – отображает информацию об ошибках в коде, кодировке страницы, синтаксисе языка, уведомления о работе процессов и другую информацию.

Рис. 2.61. Редактор VSC: нижняя панель

Терминал – скрытая в нижней части панель работы с командной строкой. Можно выбрать классическую или PowerShell. Также открыть эту панель можно в меню *Вид / Терминал*.



```
        <a href="test.html">Контроль</a>
        <a href="resources.html">Дополнительные материалы</a>
      </nav>
    </div>
  </header>

ПРОБЛЕМЫ  ВЫХОДНЫЕ ДАННЫЕ  КОНСОЛЬ ОТЛАДКИ  ТЕРМИНАЛ  JUPYTER

Windows PowerShell
(C) Корпорация Майкрософт (Microsoft Corporation). Все права защищены.

Попробуйте новую кроссплатформенную оболочку PowerShell (https://aka.ms/pscore6)

PS D:\Documents\Programming\WEB разработка\Для пособия 2023\Репозиторий (результат)\Глава 2. Технология каскадных таблиц стилей CS

    Каталог: D:\Documents\Programming\WEB разработка\Для пособия 2023\Репозиторий (результат)\Глава 2. Технология каскадных таблиц

Mode                 LastWriteTime         Length Name
----                 -------------         ------ ----
d-----        02.05.2023      8:26                icons
d-----        02.05.2023      8:26                images
```

Рис. 2.62. Редактор VSC: работа с терминалом и логами (панель также можно «поднять» из нижней области)

Вопросы для самопроверки

1. Опишите основные и дополнительные возможности профессиональных текстовых редактора для разработчиков ПО.
2. Перечислите достоинства редактора Visual Studio Code.
3. Каким образом использовать автоматическую подсказку и автозавершение кода в Visual Studio Code?
4. Для чего необходимо загружать каталог проекта в проводник редактора?
5. Перечислите основные элементы редактора Visual Studio Code и опишите их назначение.

Практикум

Задание 1

1. Установите редактор Visual Studio Code на свой компьютер. Ориентируйтесь на описанную в пункте 2.2.3 процедуру.
2. Установите плагины руссификации и Live Server.
3. Создайте каталог *Введение* и загрузите его в редактор.
4. Создайте в нем файл *First.html*: для этого воспользуйтесь кнопками в проводнике (см. рис. 2.63). Редактор по названию файла определит расширение и надлежащий язык для подсветки синтаксиса.
5. Сгенерируйте автозавершением стандартный стартовый шаблон разметки (как на рис. 2.42).
6. Потренируйтесь в создании новых веб-страничек, работе с подсказками и автозавершении тегов.
7. Приложите созданный каталог с файлами к отчету.

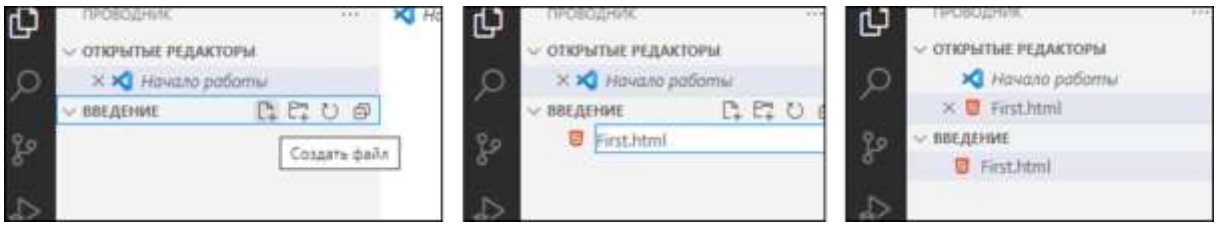


Рис. 2.63. Создание нового файла в Проводнике

Задание 2

1. Зайдите в раздел *Параметры редактора*.
2. Найдите параметры, которые отвечают за размер и семейство шрифта.
3. Временно установите шрифт Courier и размер 16, сохраните файл.
4. В чем отличие настройки этих параметров от увеличения масштаба редактора?
5. Верните исходные параметры.

Задание 3

1. Перейдите в блок *Расширения*.
2. В поле поиска запросите плагин *Material Icon Theme*.
3. Установите плагин.
4. В настройках смените тему значков файлов. Проверьте, что они отличаются от стандартных.

Задание 4

1. Откройте окно терминала.
2. Введите последовательно команды *CLS*, *DIR*, *ipconfig /all*.
3. Покрутите стрелочками *вверх* / *вниз*: PowerShell восстанавливает из кэша предыдущие команды.
4. Нажмите на символ корзины, чтобы удалить текущий сеанс терминала.

Задание 5

1. Создайте и загрузите в редактор пустой каталог с названием *Второй проект*.
2. Создайте HTML-файл с названием *index.html*. Введите в него следующий код разметки, используя возможности автозавершения тегов и вставки сниппета начального шаблона разметки:

Глава 2\Второй проект\index.html

```
<!DOCTYPE html>
<html lang="en">

<head>
    <meta charset="UTF-8">
    <meta http-equiv="X-UA-Compatible" content="IE=edge">
```

```
    <meta name="viewport" content="width=device-width,
initial-scale=1.0">
    <title>Второй проект</title>
    <link rel="stylesheet" href="style.css">
</head>

<body>
    <div class="main-box">
        <h1 class="title">Текущая дата</h1>
        <p class="date-box" id="date"></p>
    </div>

    <script src="code.js"></script>
</body>

</html>
```

3. Наведите курсор на тег `<link>` и название файла внешней таблицы CSS-стилей. Чтобы перейти к ней, зажмите *Ctrl* + *ЛКМ*. Поскольку файл еще не создан, то редактор предложит создать его с таким именем.

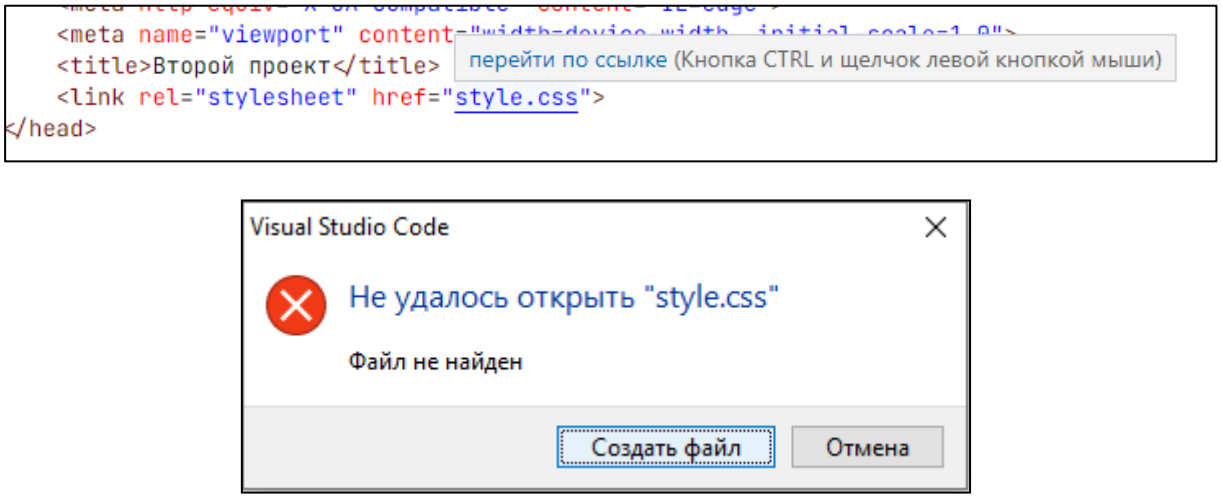


Рис. 2.64. Создание файла по ссылке

4. Наберите следующий ниже код стилей, сохраните файл.

Глава 2\Второй проект\style.css

```
body {
    background: #e6e6e6;
    font-family: Arial, Helvetica, sans-serif;
    font-size: 1.0em;
}
```

```
.main-box {
    max-width: 560px;
    background: #fff;
    margin: 0 auto;
    padding: 10px;
    border-radius: 8px;
    box-shadow: 0 0 10px #00000056;
}

.title {
    border-bottom: 3px solid #c9c9c9;
}

.date-box {
    font-family: Cambria, 'Times New Roman', serif;
    font-size: 1.1em;
}
```

5. По аналогии перейдите к тегу `<script>` и создайте новый файл для скрипта на языке JavaScript. Наберите в нем указанный код, сохраните файл.

Глава 2\Второй проект\code.js

```
const timer = document.getElementById('date');

const currentDate = new Date()
const year = currentDate.getFullYear();
const month = currentDate.getMonth();
const day = currentDate.getDate() + 1;

timer.innerText = "Сегодня " + day + ":" + month + ":" +
year;
```

6. Откройте веб-страницу в браузере. Если все выполнено без ошибок, то отобразится следующий результат:

Рис. 2.65. Результат выполнения задания: на странице выводится текущая дата

2.3. Знакомство с HTML

2.3.1. Понятие HTML

HTML как язык разметки

Определение

HTML (HyperText Markup Language, язык гипертекстовой разметки) *– это стандартный язык разметки документов во Всемирной паутине.*

Концепция HTML была предложена в 1989 году Тим Бернерс-Ли. В настоящее время этот язык разметки остается одним из наиболее востребованных в веб-разработке.

Разметка на языке HTML определяет структуру веб-документа, разбивая его с помощью тегов на блоки, текстовые абзацы, заголовки, списки, таблицы, изображения, гиперссылки, мультимедийные элементы, формы и прочее. Каждый тег задает элементу роль в разметке и правила его базового оформления в браузере.

Семантическая разметка документов

HTML-документ представляет собой обычный текстовый файл, который помимо текста содержит специальные команды (теги), определяющие его логическую структуру. Благодаря тегам разработчик может судить о структуре документа. С другой стороны, HTML-код удобен для автоматической обработки.

Поскольку теги определяют роль элемента в разметке, то ее часто называют *семантической*, т.е. смысловой. HTML является достаточно универсальным языком разметки для описания веб-документов.

HTML как общепринятый стандарт

Большинство современных веб-страниц сверстаны именно на языке HTML. Язык HTML интерпретируется браузерами и отображается в виде документа в удобной для человека форме.

Спецификация HTML5

Определение

HTML5 *– это последний и наиболее современный стандарт языка HTML.*

HTML5 пришел на замену стандартов HTML 4.01, XHTML 1.0 и XHTML 1.1. Он расширяет, улучшает и рационализирует разметку документов, а также добавляет огромный спектр возможностей для работы с мультимедиа (встроенные проигрыватели аудио и видео).

Это полезно знать!

HTML5 – это не просто усовершенствование HTML, а совершенно новая платформа для разработки веб-ресурсов.
С 28 октября 2014 года консорциум W3C официально рекомендует использовать стандарт HTML5.

Новые возможности HTML5

- *Новые теги семантической разметки*. HTML5 улучшает семантику страницы, упрощает код разметки по позволяет работать с мультимедиа.
- *Новые элементы форм*. Теперь некоторые элементы управления на формах не требуют программной реализации средствами JavaScript.
- *Рисование*. HTML5 внедряет тег `<canvas>`, в котором можно работать с растровой графикой: графическими примитивами, диаграммы, изображениями, играми.
- *Встроенная поддержка аудио и видео*. HTML5 больше не нужна работа с Adobe Flash или Microsoft Silverlight. Видео и аудио ролики сопровождаются простым интерфейсом плейера.
- *Кроссплатформенность*. HTML5 реализован как универсальная технология, поддерживаемая современными браузерами и портативными устройствами.

- *Лояльность к ошибкам*. Браузеры зачастую способны корректно интерпретировать разметки, содержащие незначительные ошибки и недочеты.
- *Поддержка геолокации*. Приложения, использующие HTML5, могут получать данные о метаположении устройства по IP-адресу, подключению к беспроводной сети, данными о сотовом операторе, а также посредством GPS оборудования.

Рис. 2.66. Логотип технологии HTML5

2.3.2. Теги и атрибуты тегов

Текст без разметки

Чтобы понять важность специальных команд в разметке текста, приведем практически пример.

В простейшей форме любой текст представляет собой набор символов. Например, есть следующий фрагмент электронного документа:

Глава 2\Роль тегов\index.txt

```
Технология HTML

HTML5

Большинство современных веб-страниц содержат описание
разметки именно на языке HTML. Язык HTML интерпретируется
браузерами и отображается в виде документа в удобной для
человека форме.
HTML5 — последний и наиболее современный стандарт языка
```

```
HTML.

<<Логотип HTML>>

Разработка документов с помощью языков разметки широко
распространена в издательствах. Здесь документ
представлен в форме перечня специальных команд,
определяющих его логическую структуру.
Технология верстки документов нашла свое применение и
в области веб-технологий, в частности, разработке сайтов.
Последние открывают широкий спектр мультимедийных
возможностей в сфере образования.

Автор: Якубович Д.
```

Однако такая форма записи не несет браузеру никакой полезной информации о том, как именно должен быть оформлен этот фрагмент, какую роль играет каждый элемент в рамках документа. Иными словами, подобная разметка документа лишена семантики.

Теги в разметке

Определение

Тег *– это правило, согласно которому информация отображается в браузере. Теги окружены символами <>.*

Теги также являются текстовыми элементами, однако браузер не отображает их на странице.

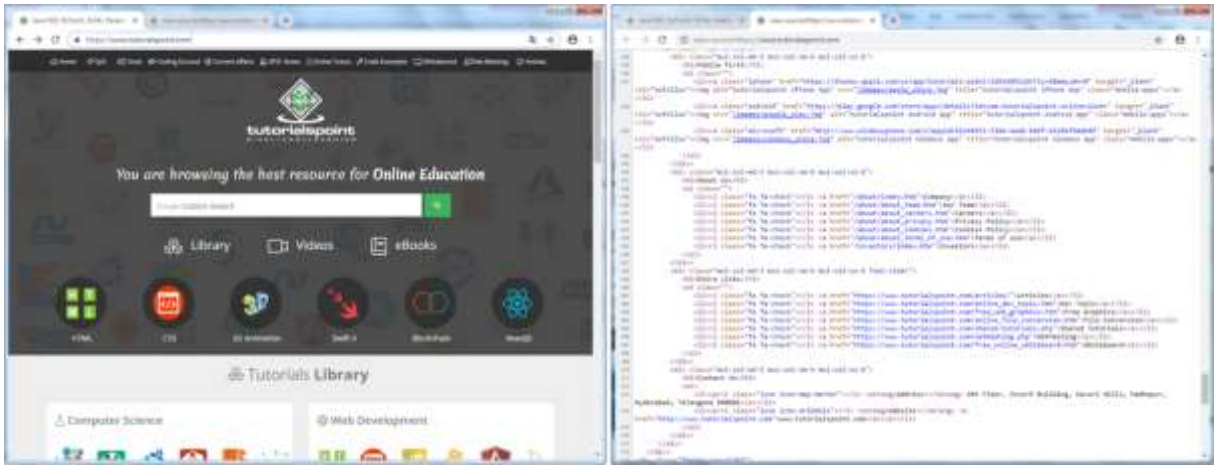


Рис. 2.67. Современный веб-сайт и его HTML-код

Вернемся к предыдущему примеру с текстом. HTML-теги позволяют наделить элементы документа смыслом, т.е. преобразовать текст в семантическую разметку.

Глава 2\Роль тегов\index.html

```
<!DOCTYPE html>
<html lang="ru">

<head>
    <meta charset="UTF-8">
    <meta http-equiv="X-UA-Compatible" content="IE=edge">
    <meta name="viewport" content="width=device-width,
initial-scale=1.0">
    <title>Document</title>
</head>

<body>
    <h1>Технология HTML</h1>
    <h2>HTML5</h2>
    <p>Большинство современных веб-страниц содержат
    описание разметки именно на языке
    <strong>HTML</strong>. Язык HTML интерпретируется
    браузерами и отображается в виде документа в удобной
    для человека форме.</p>
    <p><dfn>HTML5</dfn> – последний и наиболее
    современный стандарт языка HTML.</p>

    <img src="html_logo.png" alt="HTML5" width="120">

    <p>Разработка документов с помощью языков разметки
    широко распространена в издательствах. Здесь документ
    представлен в форме перечня специальных команд,
    определяющих его логическую структуру.</p>
    <p>Технология <strong>верстки</strong>  документов
    нашла свое применение и в области веб-технологий,
    в частности, разработке сайтов. Последние открывают
    широкий спектр мультимедийных возможностей в сфере
    образования.</p>

    <p>Автор: <cite>Якубович Д.</cite></p>
</body>

</html>
```

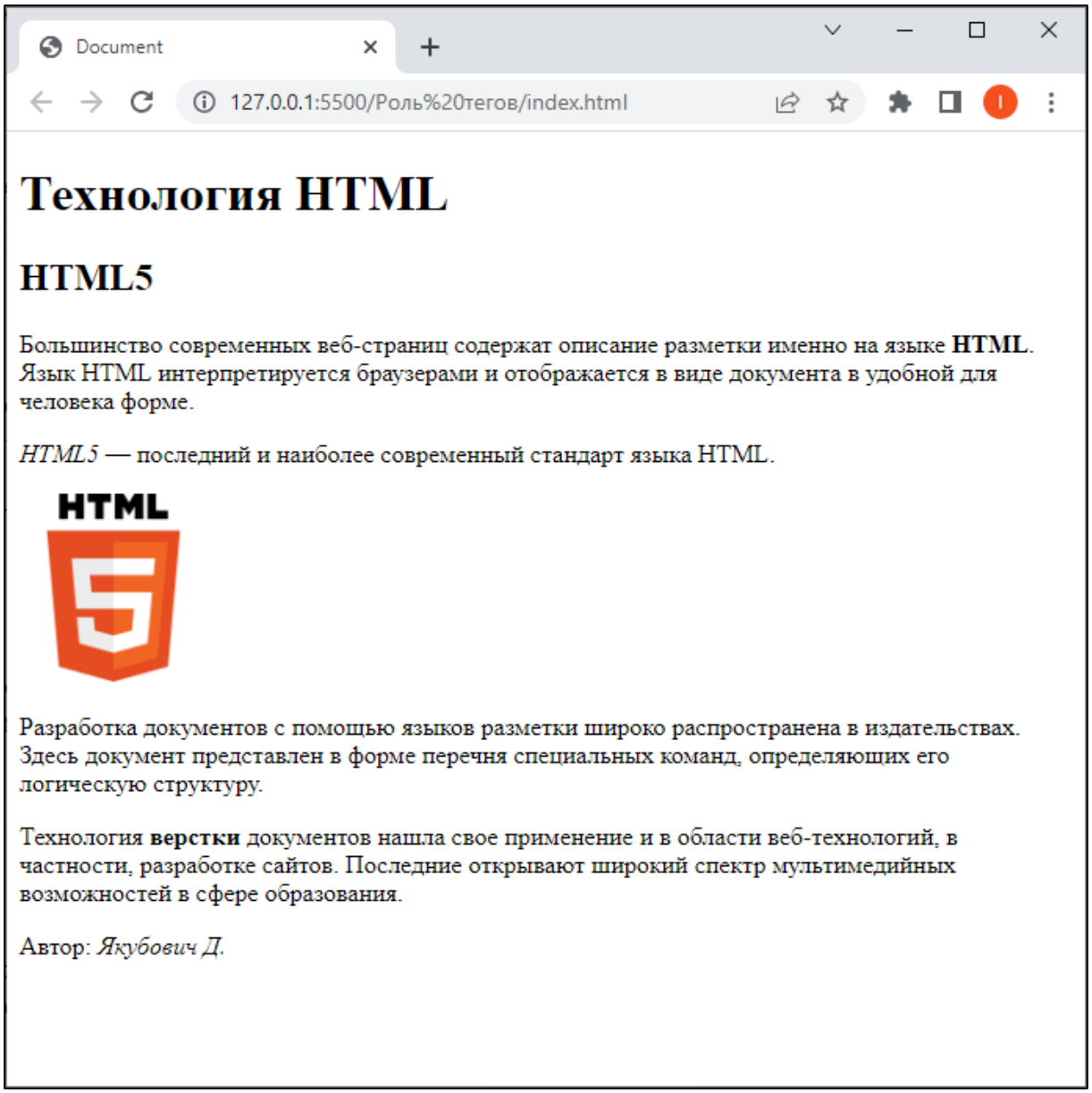


Рис. 2.68. Оформление веб-страницы из примера

Как видно из примера, текст и изображения оформлены в виде тегов, что позволяет браузеру однозначно интерпретировать их.

Рис. 2.69. Элементы тега

Таким образом для тегов справедливы следующие тезисы:

- теги задают структуру веб-страницы, размечая ее основные элементы (заголовки, абзацы, текст, блоки, таблицы, изображения и т.д.);
- большинство тегов являются парными: имеется открывающий и закрывающий теги;
- теги интерпретируются браузером, как специальные команды (они не отображаются в документе, но влияют на отображение содержимого).

Браузер анализирует текст HTML-разметки, определяя теги. Документ и его элементы отображается в окне браузера согласно тегам и закрепленными по умолчанию за ними стилями.

Виды тегов

Парные

Парные представляют собой открывающий и закрывающий теги:

```
<тег> . . . </тег>
```

Обычно парные теги определяют действие над содержимым, т.е. являются *контейнерами*.

Примеры:

```
<h1>Заголовок уровня 1</h1>
<p>Текстовый абзац</p>
<td>Ячейка таблицы</td>
```

Унарные

Унарный тег не имеет закрывающей пары:

```
<тег> (устаревший формат: <тег/>)
```

Обычно унарный тег производит действие, не привязанное к конкретному блоку (разрыв строки абзаца, прорисовка разделительной линии, вставка изображения, описание метаданных, подключение скриптов, стилей и др.).

Примеры:

```
<img src="Котики.jpg"> - вставка изображения
<br> - разрыв строки
<link ...> - подключение внешнего файла
```

Атрибуты тегов

Синтаксис описания

Определение

Атрибуты тега *– это параметры, несущие дополнительную информацию для тега или влияющие на отображение содержимого.*

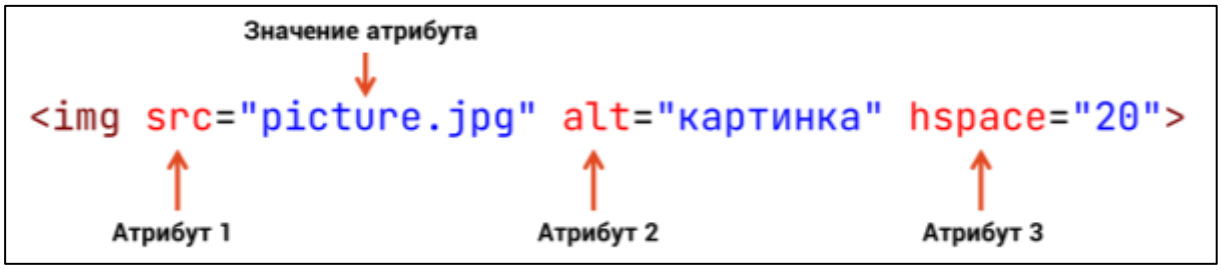


Рис. 2.70. Оформление атрибутов тега на примере разметки изображений

При описании тегов важно руководствоваться следующими правилами:

- атрибуты перечисляются в открывающем теге (если он парный) и через пробел;
- каждый атрибут состоит из пары «атрибут="значение"»;
- значение атрибута всегда пишется в кавычках (даже, если это число);
- порядок перечисления атрибутов неважен.

Виды атрибутов

Атрибуты можно разделить на следующие виды:

- *глобальные*, которые доступны всем тегам (см. далее);
- *собственные*, принадлежащие определенному тегу;
- *логические*, которые играют роль булевого флажка.

Примеры:

```
<h1 class="article-title">Атрибуты тегов</h1>
<img src="images/back.jpg" alt="фон">
<input type="button" value="OK" disabled>
```

Глобальные атрибуты

Перечислим некоторые глобальные атрибуты, которые поддерживают все теги. Новые атрибуты, которые были добавлены в HTML5, отмети значком.

Таблица 2.1. Примеры глобальных атрибутов тегов в HTML5

Атрибут		**Описание**
`accesskey`		Задает клавишу или комбинацию клавиш для установки фокуса на элементе.
`class`		Задает CSS-класс для стилизации элемента.
`contenteditable`		Показывает, допустимо ли редактирование содержимого элемента.
`contextmenu`		Задает контекстное меню для элемента, возникающее при щелчке правой кнопкой мыши.
`data-*`		Хранит пользовательские данные, закрытые для страницы или приложения.
`dir`		Определяет направление текста.
`draggable`		Определяет, допустимо ли перетаскивать элемент.
`dropzone`		Определяет, допустимо ли копирование перемещаемых данных.
`hidden`		Скрывает элемент.
`id`		Уникальный идентификатор элемента (обычно используется JavaScript).
`lang`		Определяет язык содержимого элемента.
`spellcheck`		Показывает, будет ли использоваться проверка правописания в элементе.
`style`		Определяет встроенный CSS-стиль элемента.
`tabindex`		Задает порядок переключения между элементами при нажатии клавиши TAB.
`title`		Устанавливает текст всплывающей подсказки при наведении курсора на элемент.
`translate`		Определяет, необходимо ли переводить содержимое внутри элемента.

Некоторые замечания

Это важно знать!

На первоначальном этапе не следует много внимания уделять запоминанию глобальных атрибутов. В демонстрируемых далее задачах нам потребуются лишь некоторые часто используемые, такие как `class` и `id`.

Это полезно знать!

Не следует использовать атрибуты тегов, которые отвечают за оформление элементов. В спецификации HTML5 они считаются устаревшими и не рекомендуются к применению.
Мы также не будем подробно останавливаться на изучении всех собственных атрибутов каждого тега, а затронем лишь самые важные для каждого из них.
Если требуется уточнить информацию по тегам и их атрибутам, используйте актуальную справочную информацию.

2.3.3. HTML-файлы

Форматы .html и htm

Любой HTML-документ является обычным текстовым файлом, содержащим код разметки. Редактировать разметку HTML-файла можно даже в Блокноте.

Веб-страница имеет расширение *.html* или *.htm*. Ее просмотр осуществляется в браузере. Формат .htm является устаревшим и был унаследован из-за особенности ограничения в старых операционных системах, которые могли обозначать расширение не более чем тремя символами. В настоящее время формат HTML-страниц обозначается именно как .html (хотя браузеры продолжают работать и с .htm).

HTML-файл можно редактировать и сохранять в любом текстовом редакторе, не меняя при этом его расширения (рис. 2.71).

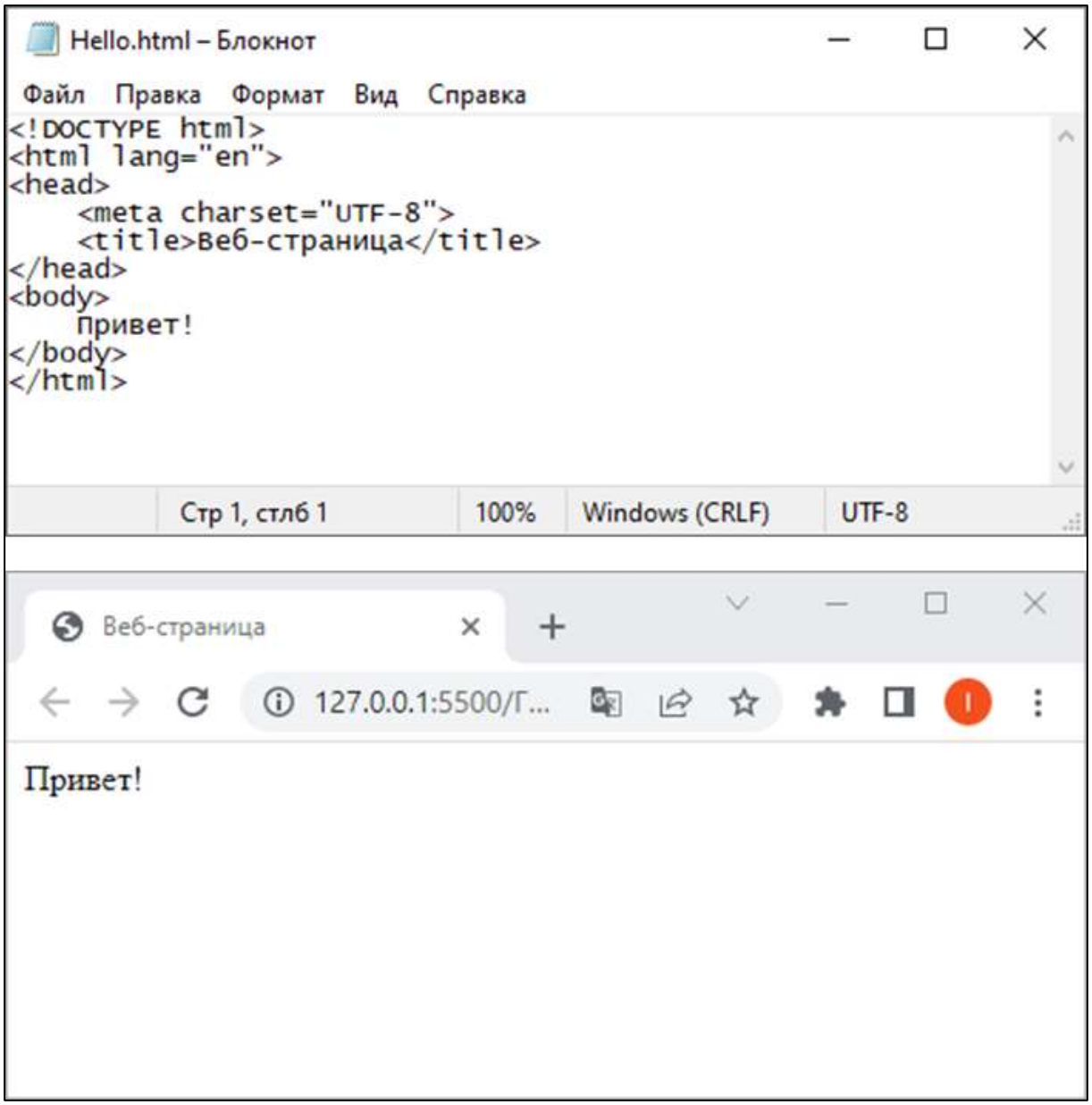


Рис. 2.71. Разметка веб-страницы в Блокноте и ее вид в браузере

Настройка формата

В процессе разработки проекта создаются или используются файлы разного формата. Редактор Visual Studio Code отображает в проводнике структуру вложения файлов и каталогов проекта, а также для удобства пользователя помечает их соответствующими пиктограммами (при условии, что установлена какая-либо тема значков).

С другой стороны, узнать формат файла можно и по его названию: он отделяется точкой в конце файла (рис. 2.72).

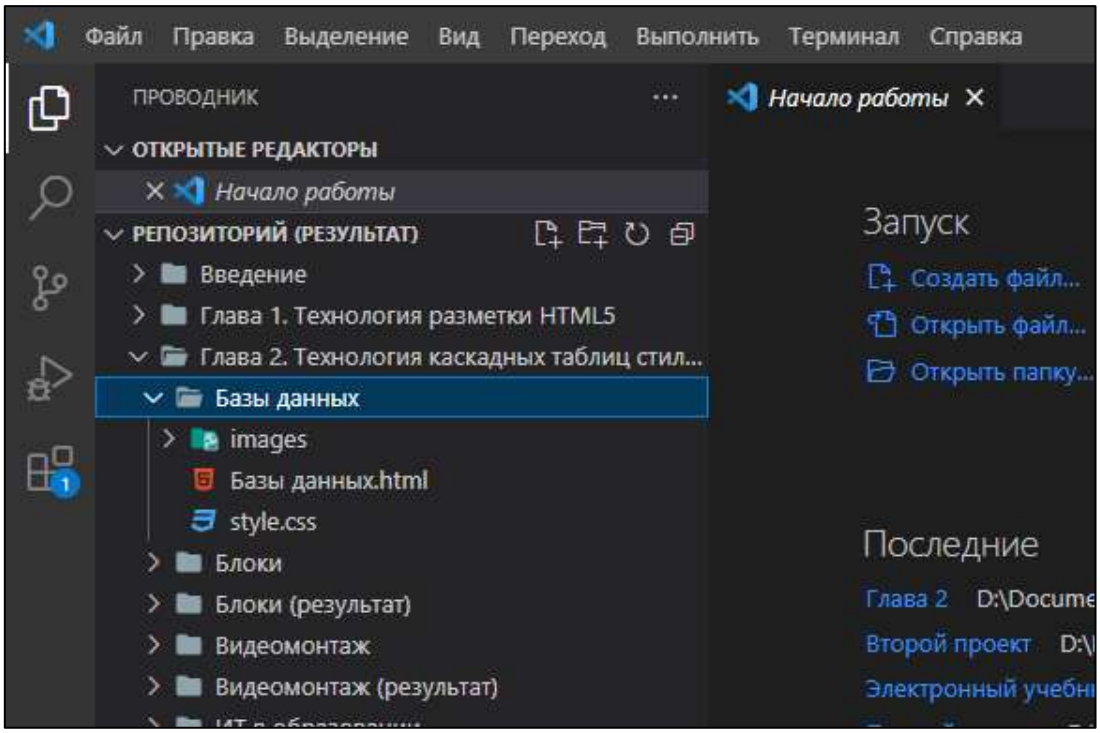


Рис. 2.72. Структура проекта: каталоги и файлы

Похожий принцип интерфейса используется и в операционных системах. Каждый файл помечается пиктограммой, которая обычно ассоциируется с приложением, открывающим файл.

В ряде случаев отображение иконки файла может быть некорректным (например, пользователь пытался открыть файл другим приложением и случайно сохранил его в качестве приложения по умолчанию). В подобных случаях желательно иметь возможность менять формат файла в названии.

Однако в некоторых операционных системах изначально форматы файлов могут скрываться. Покажем на примере операционной системы Windows, как их отобразить.

1. Зайдите в *Панель управления*, в разделе *Категория* выберете *Мелкие значки*:

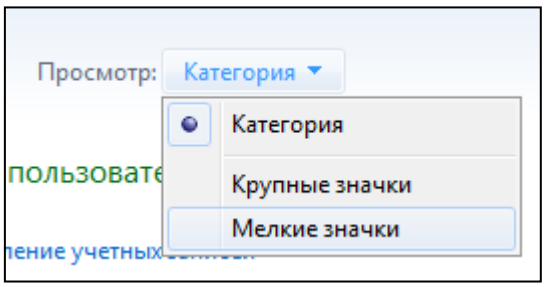


Рис. 2.73. Переход в панель управления

2. Найдите раздел *Параметры папок* или *Параметры Проводника*:

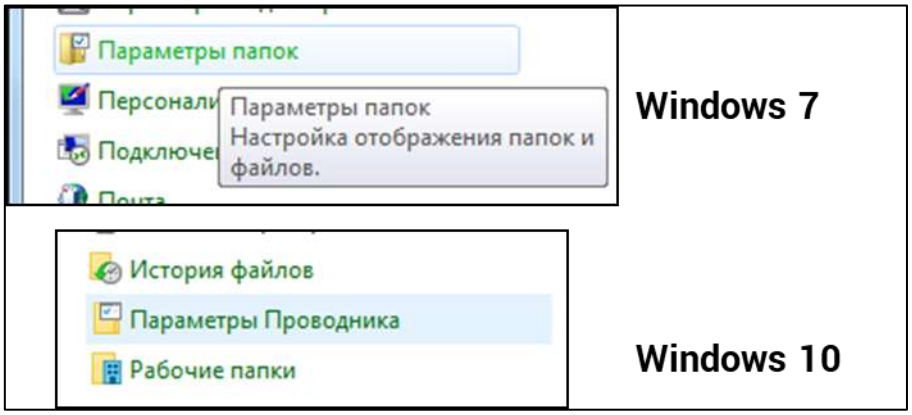


Рис. 2.74. Параметры каталогов

3. На вкладке *Вид* снимите галочку для пункта *Скрывать расширения для зарегистрированных типов файлов* (если она установлена):

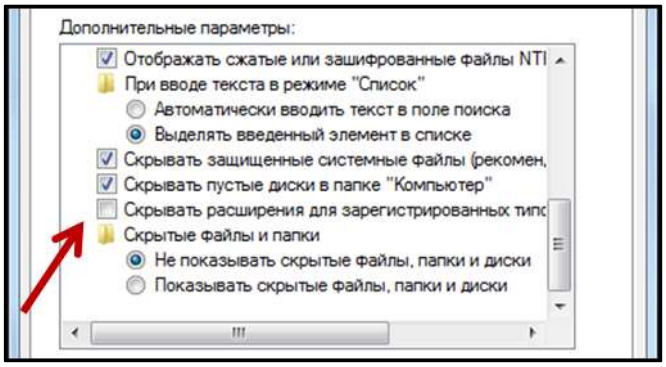


Рис. 2.75. Отображаем формат файлов в имени явно

4. Теперь файлы отображаются вместе с форматом, который при необходимости можно изменить вручную, переименовав файл:

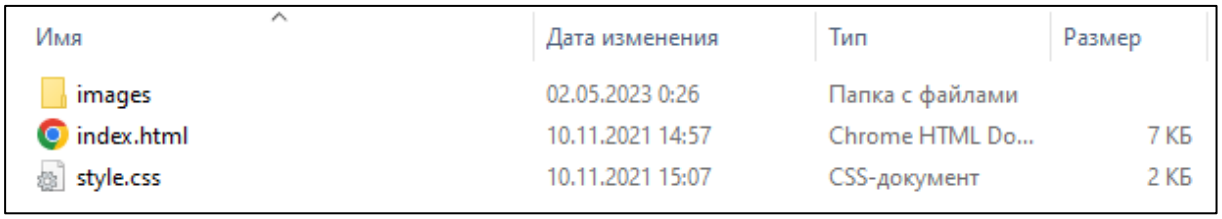


Рис. 2.76. Файлы отображают не только имя, но и формат

Кодировка текстовых файлов

При работе с текстовыми файлами, в том числе листингами разметки или программного кода, важно не забывать следить за кодировкой, в которой они сохранены.

Для HTML-файлов в настоящее время стандартом кодировки является *UTF-8*. Эта кодировка позволяет записывать огромное число различных символов, в частности – на разных языках.

Это важно знать!

Если в текстовом файле символы отображаются «крокозябрами» – это верный признак некорректно установленной кодировки файла.

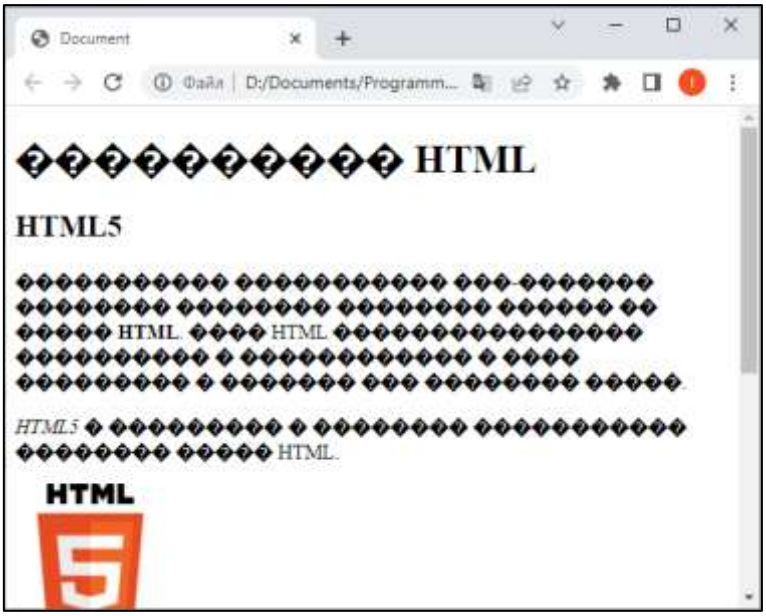


Рис. 2.77. Возникла проблема с кодировкой символов

Узнать и поменять кодировку файла можно как в обычном Блокноте (рис. 2.78), так и в продвинутом редакторе, например, Notepad++ (рис. 2.79) и Visual Studio Code (рис. 2.80).

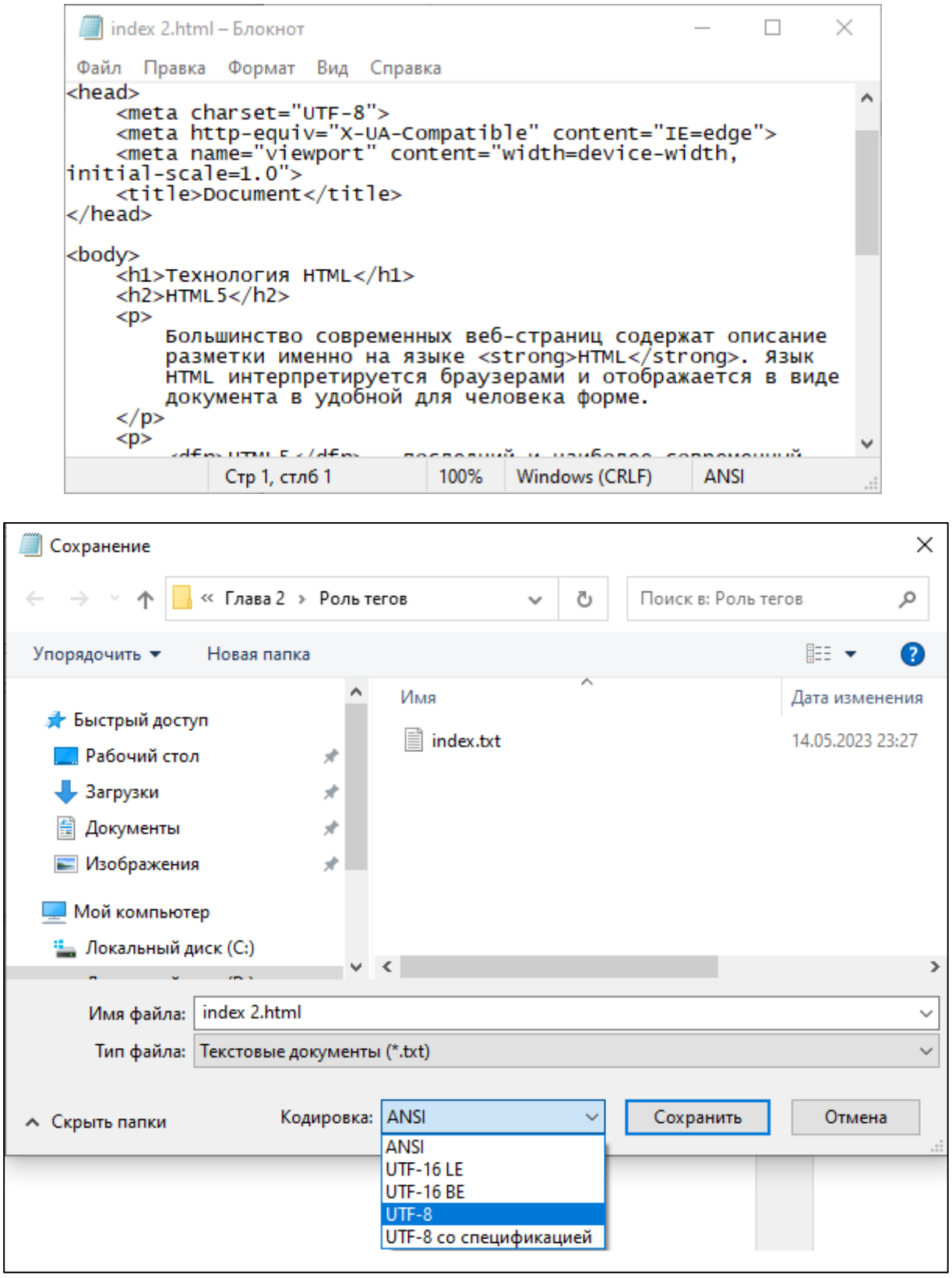


Рис. 2.78. Кодировка файла и ее смена при сохранении в стандартном приложении Блокнот

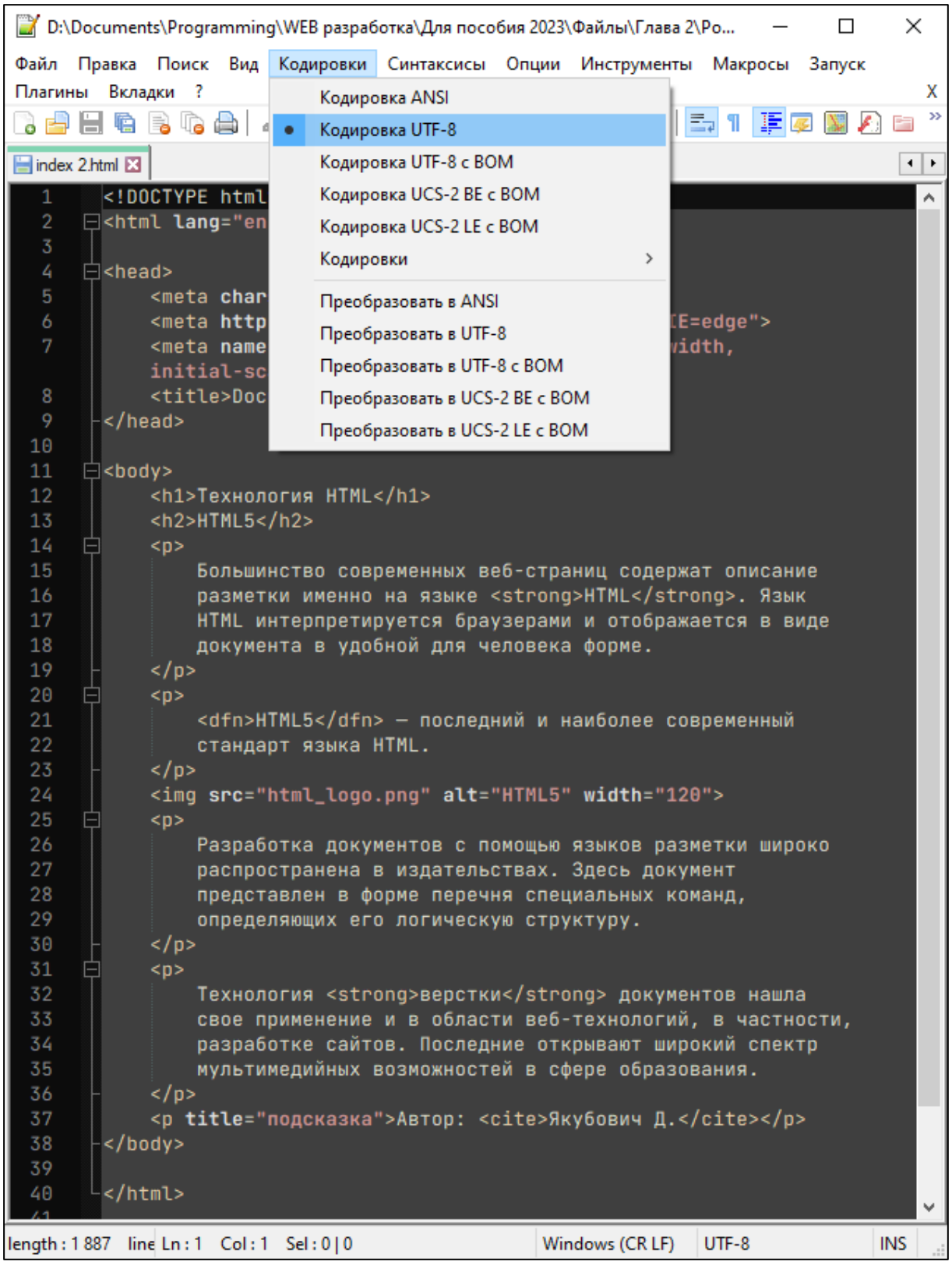


Рис. 2.79. Работа с кодировками на примере Notepad++.
Редактор поддерживает множество различных кодировок
и преобразование в них

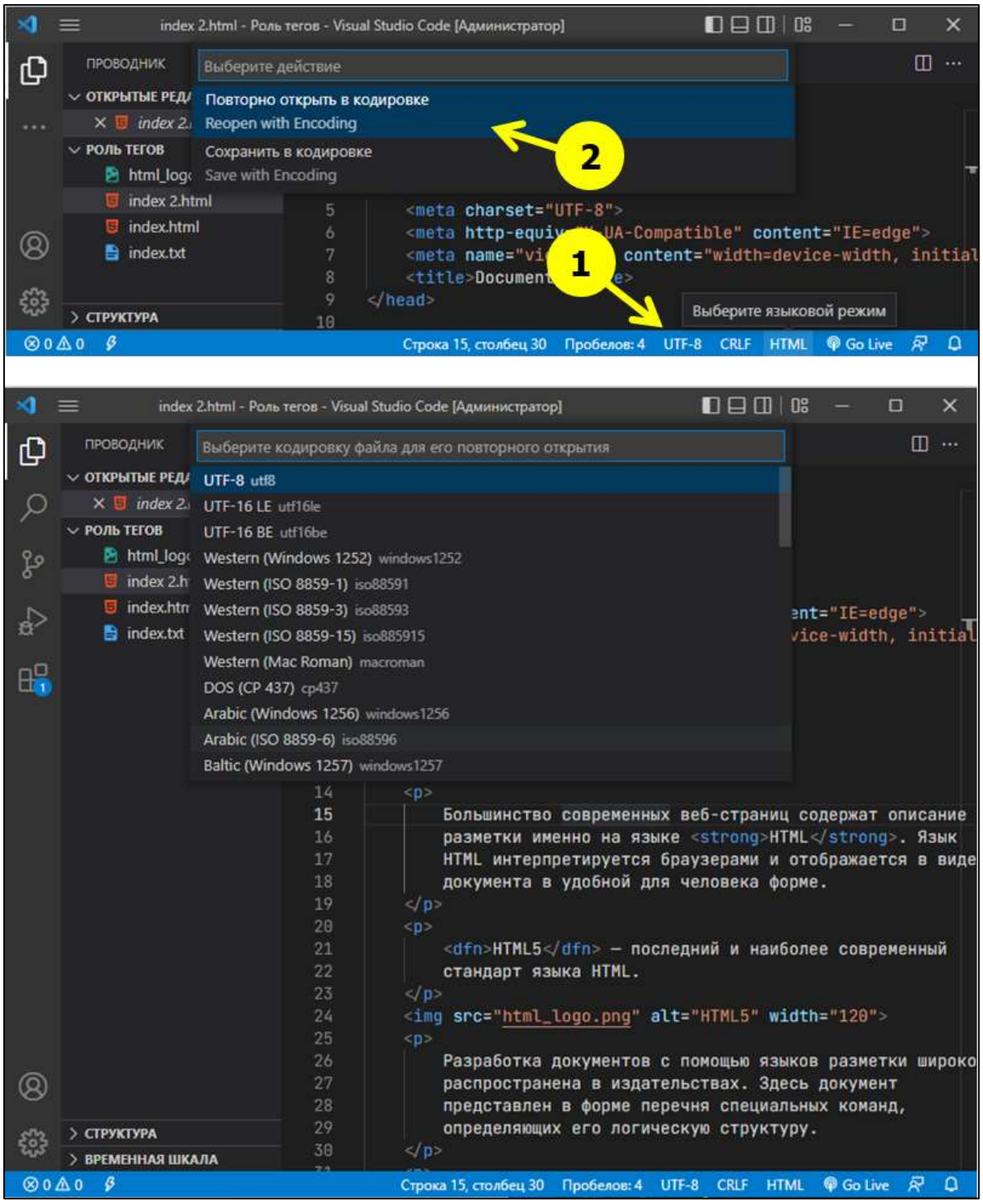


Рис. 2.80. Смена кодировки в Visual Studio Code.

Редактор также поддерживает большое число как распространенных, так и не часто используемых кодировок

Это полезно знать!

Visual Studio Code изначально создает файлы под кодировкой UTF-8. Поэтому менять кодировки в процессе не требуется.

Вопросы для самопроверки

1. В чем состоит значимость семантической разметки и как она организуется?
2. Какие новые возможности предоставляет платформа HTML5?
3. Почему атрибуты тегов, определяющие внешний вид элемента, не следует использовать?
4. Какие виды тегов выделяются и как их корректно описывать?
5. Укажите правила описания атрибутов тега.
6. Для каких целей служат глобальные атрибуты?
7. Как правильно осуществить преобразование кодировки файла?

Практикум

Задание 1

1. Изучите код разметки из раздела 2.3.2.
2. В любом онлайн-справочнике найдите описание тегов внутри `<body></body>` (для чего они нужны).

Задание 2

1. Наберите или скопируйте код предыдущего задания.
2. С помощью Visual Studio Code попробуйте изменить кодировку файла на другую.
3. Сохраните файл в качестве отдельной копии.
4. Проверьте отображение страницы в браузере. Появились ли «кракозябры»?
5. Изучите самостоятельно, для чего указывается кодировка в теге `<meta>` и как она связана с кодировкой файла.

2.4. Структура HTML-документа

2.4.1. Простой стартовый шаблон

Рекомендуемая структура

Базовая структура разметки веб-страницы в HTML5 может быть разбита на три части:

- *объявление* HTML-документа (сообщает о том, что разметка относится к HTML);
- *преамбула* (содержит метаданные документа и подключает внешние файлы);
- *тело* документа (разметка содержимоuо, которое непосредственно отображается в окне браузера).

Глава 2\Базовая структура\base.html

```
<!DOCTYPE html>
<html>
    <head>
        <title>Веб-страница</title>
        <meta charset="utf-8">
    </head>
    <body>

    </body>
</html>
```

Это важно знать!

Браузер распознает теги в любом регистре, например `<strong>`, `<STRONG>` *или даже* `<sTRoNg>` *интерпритируются одинаково.*
Однако консорциум W3C рекомендует набирать теги строчными буквами. Например: `<html>`, `<img>`, `<p>` *и т.д. Этому правилу оформления следуют и текстовые редакторы.*

Упрощенная разметка в HTML5

В HTML5 было включено множество новых семантических тегов и внесены усовершенствования в синтаксис языка разметки. Язык стал более упорядоченным и лояльным к возможным недочетам в разметках страниц.

Спецификация HTML5 также допускает более короткую реализацию основы разметки: блочные теги `<head></head>` и `<body></body>` можно опустить.

Глава 2\Базовая структура\simple.html

```
<!DOCTYPE html>

<title>Веб-страница</title>
<meta charset="utf-8">

<!-- Здесь разметка тела документа -->
```

Это полезно знать!

В целом подобная структура считается вполне валидной для стандарта HTML5. Однако в дальнейшем мы будет использовать полную форму записи структуры с явным указанием важнейших элементов. Этого же правила придерживаются и веб-разработчики.

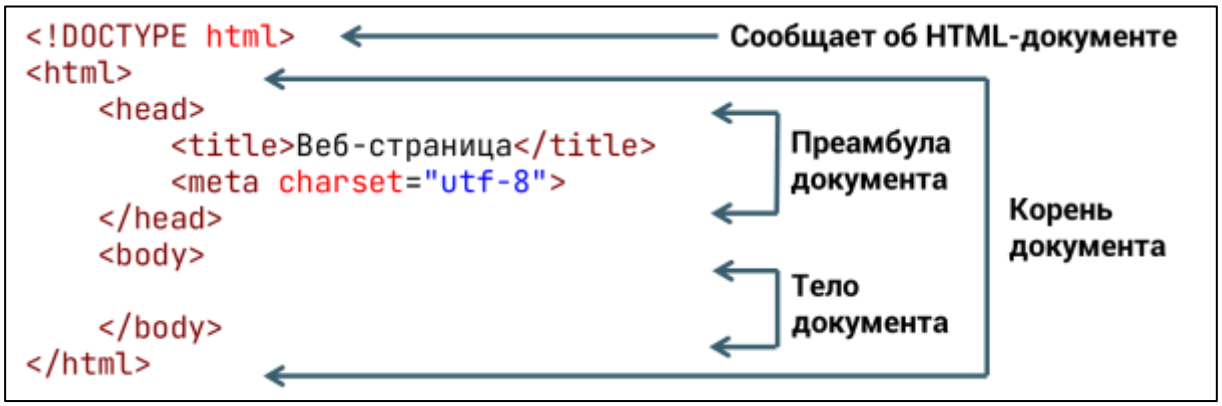


Рис. 2.81. Структура основы HTML-документа: содержит важнейшие теги, размечающие метаданные и тело документа

2.4.2. Основные теги структуры

Раздел DOCTYPE

DOCTYPE-объявление должно быть расположено в начале HTML-разметки веб-страницы. Оно не относится к каким-либо тегам и не отображается на странице. Главная задача DOCTYPE – указать стандарт HTML, поддерживаемый искомым документом.

Для спецификации HTML5 объявление очень простое: достаточно указать лишь `html`.

```
<!DOCTYPE html>
<html>
    <head>
        <title>Веб-страница</title>
        <meta charset="utf-8">
    </head>
    <body>

    </body>
</html>
```

Корневой элемент <html>

Парный тег `<html></html>` – это *корневой элемент* документа, который содержит все остальные теги. Контейнер сообщает браузеру, где начинается разметка веб-страницы

```
<!DOCTYPE html>
<html>
    <head>
        <title>Веб-страница</title>
        <meta charset="utf-8">
    </head>
    <body>

    </body>
</html>
```

Преамбула <head>

Преамбула документа

Парный тег `<head></head>` определяет *преамбулу* документа. Здесь описывается служебная информация о веб-странице: метаданные, данные о стилях, исполняемые скрипты и другое.

Данные из преамбулы документа необходимы и поисковым системам. На основе правил SEO-оптимизации осуществляется грамотная настройка содержимого этого раздела.

При этом информация внутри `<head>` не отображается на странице.

```
<!DOCTYPE html>
<html>
    <head>
        <title>Веб-страница</title>
        <meta charset="utf-8">
    </head>
    <body>

    </body>
</html>
```

Раздел <title>

Парный тег `<title></title>` является обязательным и содержит *заголовок* электронного документа. Указанный здесь текст отображается на вкладке.

Раздел включают в преамбулу `<head></head>`.

```
<!DOCTYPE html>
<html>
    <head>
        <title>Веб-страница</title>
        <meta charset="utf-8">
    </head>
    <body>

    </body>
</html>
```

Метаданные <meta>

Унарный тег `<meta>` предоставляет необходимую информацию для браузеров и поисковых систем. Тегу доступны многочисленные атрибуты, задающие различные метаданные. Среди них важным является атрибут `charset`, который указывает на используемую кодировку веб-страницы.

Раздел включают в преамбулу `<head></head>`. Тег `<meta>` допускает многократное применение.

```
<!DOCTYPE html>
<html>
    <head>
        <title>Веб-страница</title>
        <meta charset="utf-8">
    </head>
    <body>

    </body>
</html>
```

Важное замечание!

Атрибут `charset` указывает кодировку в предположении, что файл сохранен именно в этой кодировке.
Если символы в браузере отображаются кракозябрами, следует сверить кодировки.

Тело <body>

Парный тег `<body><body>` размечает «*тело*» документа. Именно в этом контейнере и размещается код разметки, который формирует визуальный интерфейс веб-страницы.

В тело документа также можно выносить описание или подключение внешних CSS-стилей и JavaScript-скриптов.

```
<!DOCTYPE html>
<html>
    <head>
        <title>Веб-страница</title>
        <meta charset="utf-8">
    </head>
    <body>

    </body>
</html>
```

2.4.3. Стартовый шаблон для современных браузеров

Расширенный шаблон

Приведенный ранее пример оформления базовой структуры HTML-страницы обычно расширяется некоторой дополнительной информацией. В частности, редактор Visual Studio Code предлагает вставку стандартного шаблона для начала разметки документа.

Глава 2\Базовая структура\standart.html

```
<!DOCTYPE html>
<html lang="ru">

<head>
    <meta charset="UTF-8">
    <meta http-equiv="X-UA-Compatible" content="IE=edge">
    <meta name="viewport" content="width=device-width,
initial-scale=1.0">
    <title>Веб-страница</title>
</head>

<body>
    <!-- Тело веб-страницы (отображается в браузере) -->
</body>

</html>
```

Приемы работы с Visual Studio Code

Для быстрой вставки представленного выше стартового шаблона используйте сниппет ***html:5*** *или* ***!****.*

Дополнительные элементы шаблона

Ведущий язык документа

```
<html lang="ru">
```

Глобальный атрибут `lang` для тега `<html>` определяет основной язык, на котором написан текст содержимого веб-страницы. Это поз-

воляет учитывать некоторые особенности при отображении символов на этом языке.

Совместимость с Internet Explorer

```
<meta http-equiv="X-UA-Compatible" content="IE=edge">
```

Метатег `<meta>` с атрибутом `http-equiv`, установленным со значением `X-UA-Compatible` и атрибутом `content` гарантирует, что браузер Internet Explorer (IE) использует новейший доступный механизм рендеринга веб-страницы, помогая обеспечить совместимость с современными веб-стандартами и функциями.

Когда IE обнаруживает этот метатег, он отображает веб-страницу, используя современный механизм визуализации, доступный на ПК пользователя. Это гарантирует, что страница будет отображаться с использованием механизма рендера, который соответствует современным стандартам.

Необходимо отметить, что указанный метатег актуален только для браузеров IE, другие современные браузеры его игнорируют.

Отображение на разных устройствах

```
<meta name="viewport" content="width=device-width,
initial-scale=1.0">
```

Метатег `viewport` отвечает за правило масштабирования при отображении страницы на разных устройствах.

Необходимость использования этого метатега связана с появлением различных типов устройств. Проблема была связана с тем, что область просмотра у разных устройств отличается. В частности, для мобильных устройств она будет меньше, чем для экранов настольных компьютеров.

Viewport сообщает браузеру, что ширину экрана следует считать относительно «полной ширины» страницы. Благодаря этому на мобильных устройствах содержимое страницы не будет отображаться слишком мелко и не потребует ручного масштабирования и горизонтальной прокрутки. Коэффициент 1.0 указывает начальный уровень масштабирования веб-страницы.

Таким образом веб-сайт будет соответствовать ширине устройства, которое используется для просмотра.

В целом возможности и задачи метатега существенно шире и его использование важно для дальнейшего обеспечения адаптивного дизайна страницы.

2.4.4. Оформление кода разметки

Отступы как показатель иерархии элементов

В приведенных ранее примерах при оформлении листингов HTML-кода использовались отступы, которые позволяли визуально отслеживать вложение элементов, т.е. зависимости между родительскими элементами и дочерними.

Однако для браузеров запись кода непринципиальна: его можно набрать даже в одну строку. Например, следующие две разметки эквивалентны с точки зрения описания метаданных и содержимого веб-страницы (рис. 2.82).

Глава 2\Базовая структура\correct.html

```
<!DOCTYPE html>
<html lang="ru">

<head>
    <meta charset="UTF-8">
    <meta http-equiv="X-UA-Compatible" content="IE=edge">
    <meta name="viewport" content="width=device-width,
initial-scale=1.0">
    <title>Веб-страница</title>
</head>

<body>
    <h1>Стиль оформления кода</h1>
    <p>Отступы в HTML-коде позволяют визуально
    отслеживать структуру разметки и зависимость между
    элементами.</p>
</body>

</html>
```

Глава 2\Базовая структура\wrong.html

```
<!DOCTYPE html><html lang="en">
<head>
    <meta charset="UTF-8">
<meta http-equiv="X-UA-Compatible" content="IE=edge">
<meta name="viewport" content="width=device-width,
initial-scale=1.0">
    <title>Веб-страница</title>
        </head>

<body><h1>Стиль оформления кода</h1><p>Отступы в HTML-
коде позволяют визуально отслеживать структуру разметки и
зависимость между элементами.</p>
</body>

</html>
```

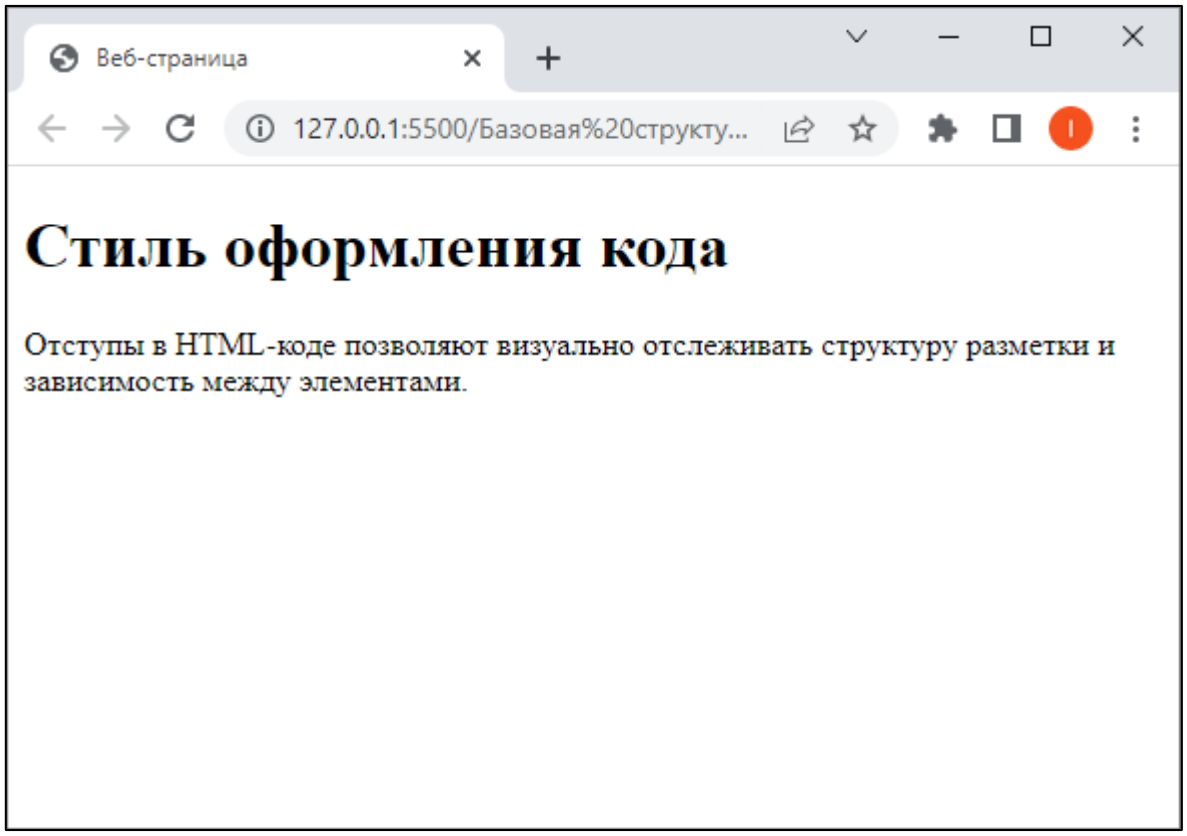


Рис. 2.82. Разметки в файлах correct.html и wrong.html дают эквивалентный результат

Несмотря на одинаковый результат, следует быть аккуратным в оформлении кода и придерживаться общепринятых правил.

Некоторые правила оформления HTML-кода

1. Теги и их атрибуты записываются в нижнем регистре.
2. Для парных тегов их внутренняя разметка смещается вправо на 4 или 2 пробела (на каждый вложенный уровень).
3. Между атрибутами и их значениями пробелы не ставятся.
4. Логически связанные блоки разметки допускается отделять пустой строкой (она не влияет на отображение).

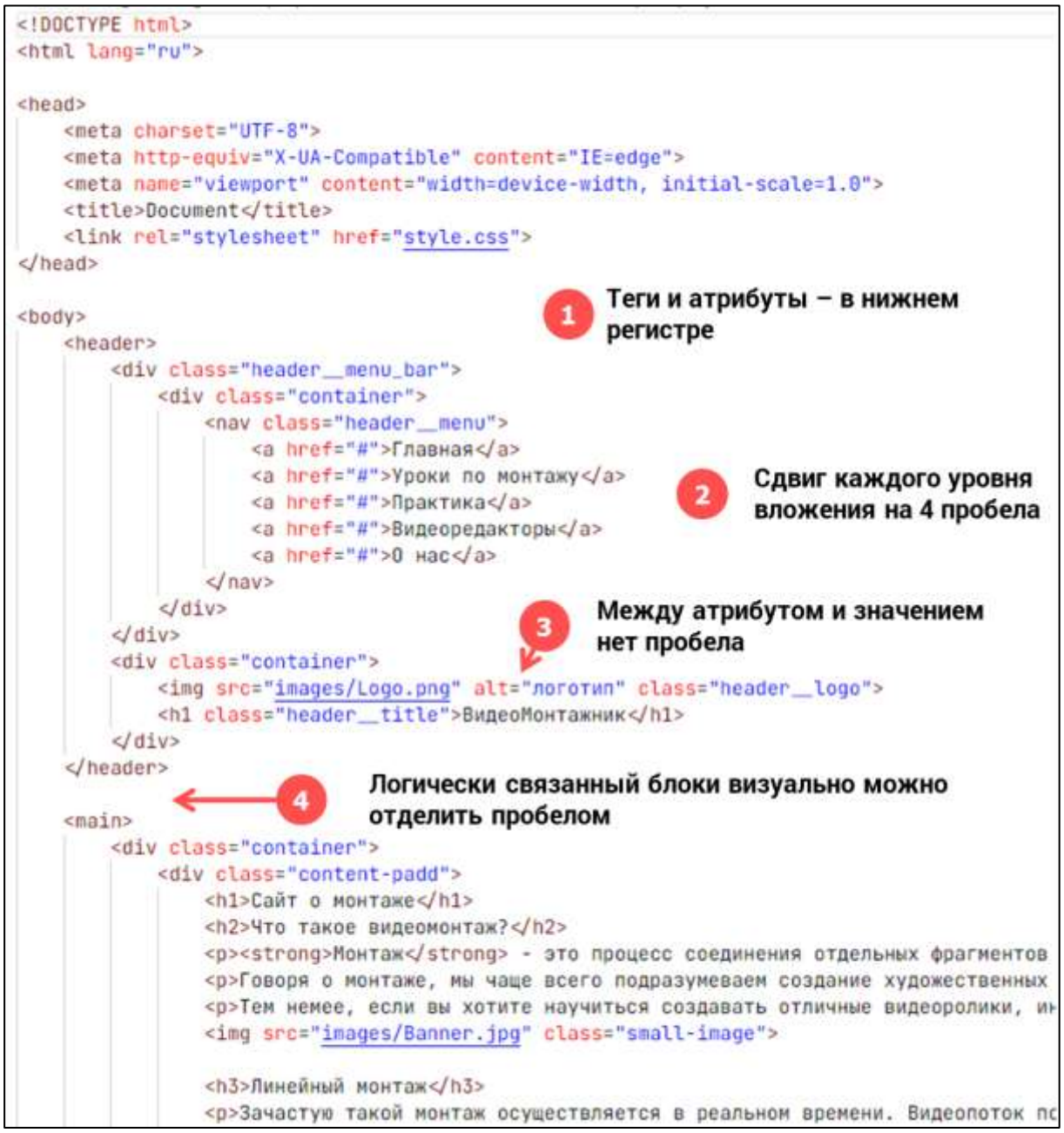


Рис. 2.83. Правила оформления кода на примере

Приемы работы с Visual Studio Code

Редактор позволяет форматировать HTML код согласно принятой стилистике.
*Для этого выделите весь документ (**CTRL + A**) и нажмите **ПКМ / Форматировать документ**.*
Важно заметить, что разметка не должна содержать ошибки в синтаксисе: лишь в этом случае гарантируется корректность форматирования.

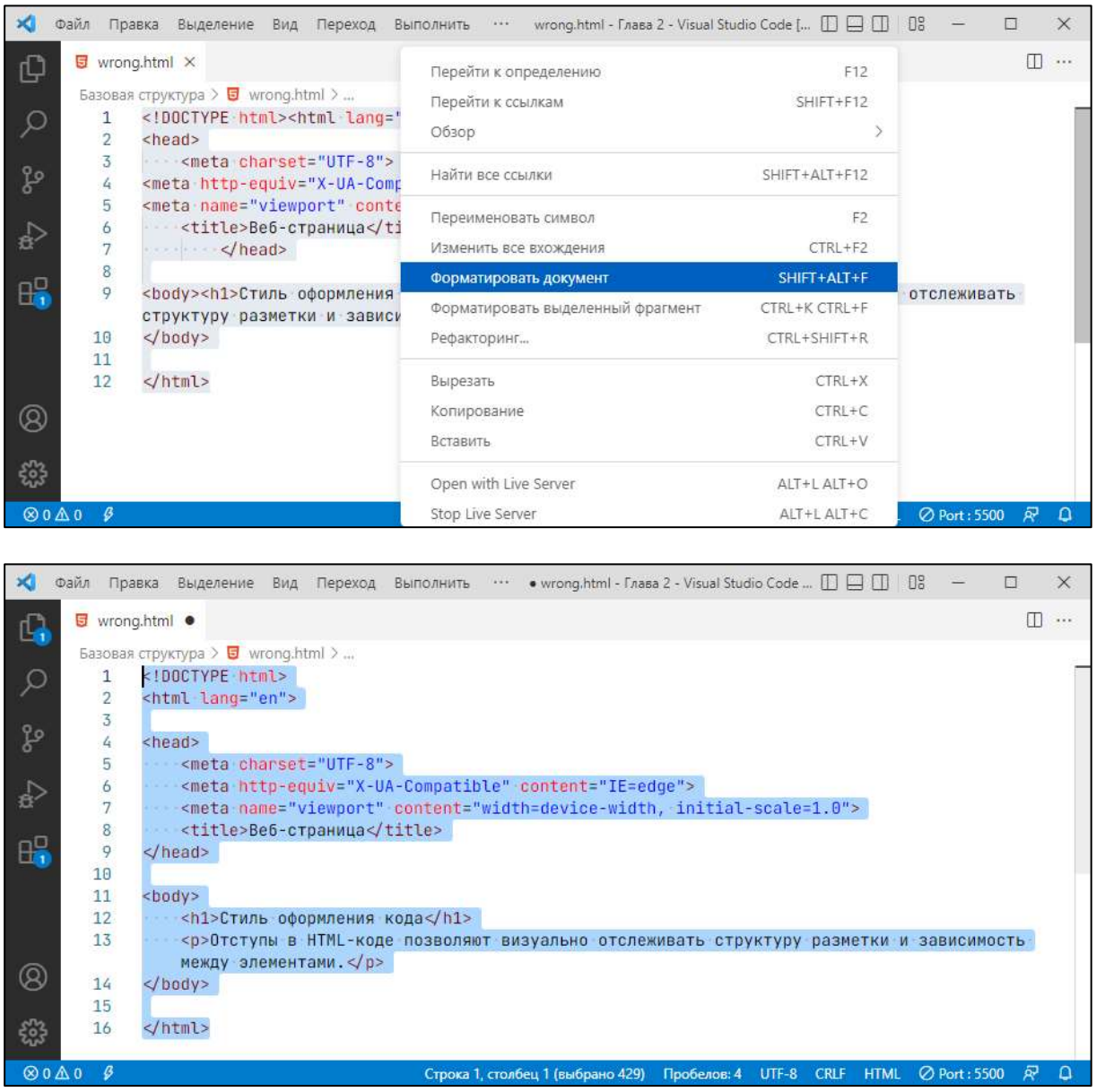


Рис. 2.84. Автоматическая корректировка оформления в редакторе

Вопросы для самопроверки

1. Опишите минимально возможную структуру HTML документа и как она может быть упрощена в спецификации HTML5.
2. Перечислите важнейшие теги структуры HTML-документа и опишите их назначение.
3. Для каких целей используются метаданные?
4. Что следует указывать при разметке шаблона современных веб страниц?
5. Какими правилами оформления кода разметки следует руководствоваться при верстке?

Практикум

Задание 1

1. Создайте каталог «Базовая разметка».
2. Создайте в нем отдельно файлы, в которых протестируйте запуск рассмотренных в текущем параграфе примеров.

Задание 2

1. Добавьте в созданный каталог пример из пункта 2.4.4.
2. Намерено нарушьте оформление кода и отформатируйте его средствами редактора.
3. Допустите в нарушенном коде ряд синтаксических ошибок. Определите, в каком случае Visual Studio Code не может корректно форматировать код с ошибками.

Задание 3

1. Изучите дополнительный материал по метатегу `<meta>`.
2. Пусть требуется создать веб-страницу, на которой должна быть размещена актуальная информация об электронных учебниках по объектно-ориентированному программированию. Опишите метатег(и), который будет включать в себя необходимое краткое описание и ключевые слова, упрощающие поисковым роботам поиск вашего сайта.
3. Реализовать задание в отдельном файле. Разметку тела страницы делать не требуется.

2.5. Разметка текстовых данных

2.5.1. Логическая и физическая разметка

Теги логической разметки

Теги *логической разметки* предназначены для разделения текста на смысловые разделы: заголовки, абзацы, цитаты, термины, устаревшую информацию и другое. Они определяют роль текстовой информации в структуре страницы.

Для HTML5 использование тегов логической разметки стало одним из важнейших принципов разметки, который позволяет рассматривать язык именно в роли языка семантической разметки.

Примеры:

```
<h1>Заголовок</h1>
<p>Текстовый абзац</p>
<p><dfn>Тег</dfn> - специальная команда в разметке
HTML.</p>
```

Рис. 2.85. Примеры тегов логической разметки

Теги физической разметки

Теги *физической разметки* отвечают за визуальное оформление текста. В отличие от тегов логической разметки, они не несут для поисковых систем какой-либо полезной информации и не задают роль тексту: их главная задача – оформление элемента.

Теги физической разметки использовались в предыдущих спецификациях HTML. В настоящее время в HTML5 от практики их использования отказались. Основная причина – теги физического фор-

матирования противоречат концепции HTML5 как языка семантической разметки и являются «пережиткам» предыдущих версий языка.

Заметим, что браузеры до сих пор поддерживают работу с тегами физической разметки. Ряд тегов оставлены в HTML5, но были наделены определенным смыслом.

Примеры:

```
<b>Полужирное начертание</b> и
<font face="Georgia" color="green">старый тег
форматирования</font>
```

Рис. 2.86. Теги меняют оформление элементов

Отказ от физической разметки

В HTML5 избавились от недостатков предыдущих версий и прекратили поддержку устаревших тегов физической разметки. Теги логического форматирования определяют *семантику* (смысл) текстовой информации. Это необходимо поисковым системам. Веб-страница с качественной логической разметкой и метаданными более приоритетна в выдаче поисковика по запросам пользователя.

Что касается оформления элемента, то за это отвечает технология CSS. Она позволяет описывать стили отдельно от элемента и не менять структура HTML-разметки.

Приемы работы с Visual Studio Code

Редактор помечает устаревшие теги и не выдает в сплывающей подсказке атрибуты тегов, отвечающие за их визуальное форматирование.

Рис. 2.87. Устаревшие теги помечаются другим цветом и не отображаются в подсказке редактора

2.5.2. Разметка текста

Заголовки

Определение

```
<h1></h1>
<h2></h2>
<h3></h3>
<h4></h4>
<h5></h5>
<h6></h6>
```

*Определяют шесть уровней **заголовков**.*
Уровень 1 – главный уровень, далее – по убыванию в иерархии.
Браузеры по умолчанию отображают их текст разным размером.

Уровни заголовков устанавливают значимость заголовка по отношению к его подчиненным подзаголовкам. Здесь используется тот же принцип, что и в книге:

- первым уровнем являются названия веб-страниц или крупных разделов (аналоги глав, разделов, частей);
- вторым уровнем отмечаются пункты внутри веб-страниц (аналоги параграфов внутри глав);
- третьим – подзаголовки внутри веб-страниц (аналоги подпунктов в параграфах) и т.д.

Рис. 2.88. Пример уровней заголовков в документе MS Word

На практике обычно достаточно первых трех или четырех уровней вложения заголовков. Если требуется создать больше уровней, возможно, более рациональным будет пересмотреть организацию структуры разметки и выделить более крупные блоки (статьи, новости, посты и др.).

Стоит отметить, что HTML непринципиально, если логическая последовательность уровней заголовков будет нарушена (такая разметка все равно будет валидной). За корректность следования уровней заголовков несет ответственность веб-разработчик.

Глава 2\Заголовки\index.html

```
<h1>Заголовок уровня 1</h1>
<h2>Заголовок уровня 2</h2>
<h3>Заголовок уровня 3</h3>
<h4>Заголовок уровня 4</h4>
<h5>Заголовок уровня 5</h5>
<h6>Заголовок уровня 6</h6>
```

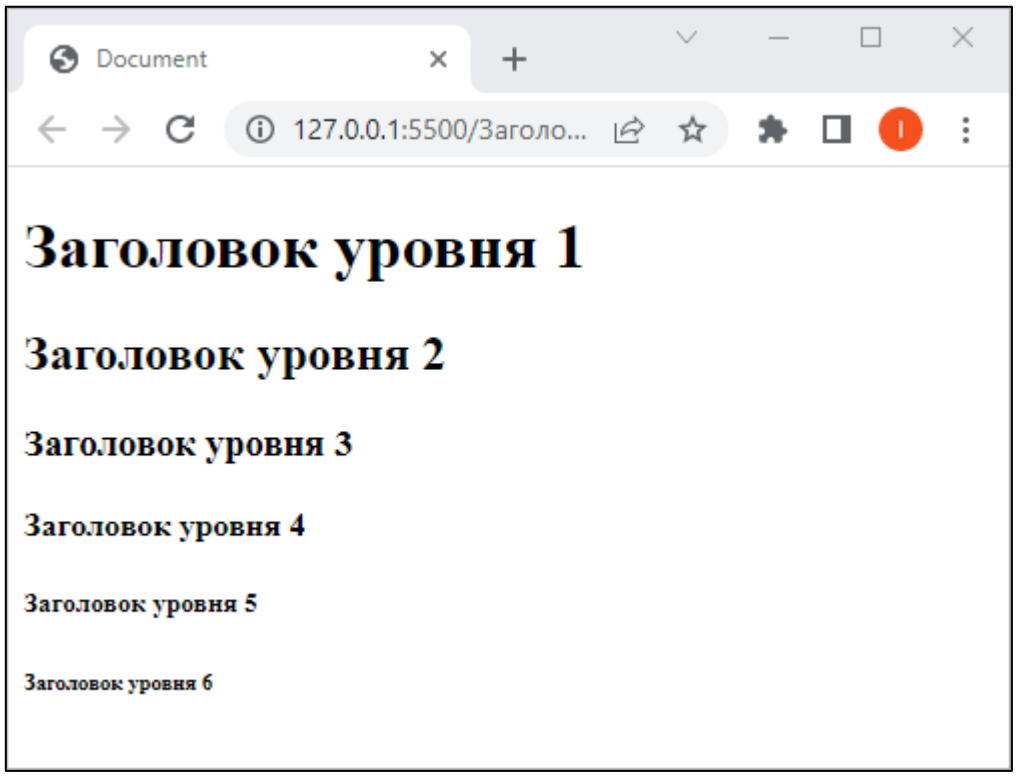


Рис. 2.89. Стандартное оформление заголовков разного уровня

Абзацы

Определение

`<p></p>`

*Тег размечает **текстовый абзац**.*
По умолчанию браузеры добавляют небольшой интервал (отбивку) сверху и снизу. Абзац является блоком, т.е. занимает всю ширину тега-родителя.

`<br>`

*Осуществляет принудительный **разрыв строки** в абзаце.*

Абзацы являются основой разметки текста и могут содержать в себе вложенные теги для разметки текста (термины, ключевые слова и др.).

Глава 2\Абзацы\index.html

```
<h1>Разметка текста</h1>
<h2>Текстовый абзац</h2>
<p><strong>Текстовый абзац</strong> является завершенным
по смыслу блоком текста из одного или нескольких
предложений. Между текстовыми абзацами браузеры по
умолчанию устанавливают небольшую отбивку.</p>
<p>Разрыв строки создает<br>
принудительный перенос</p>
```

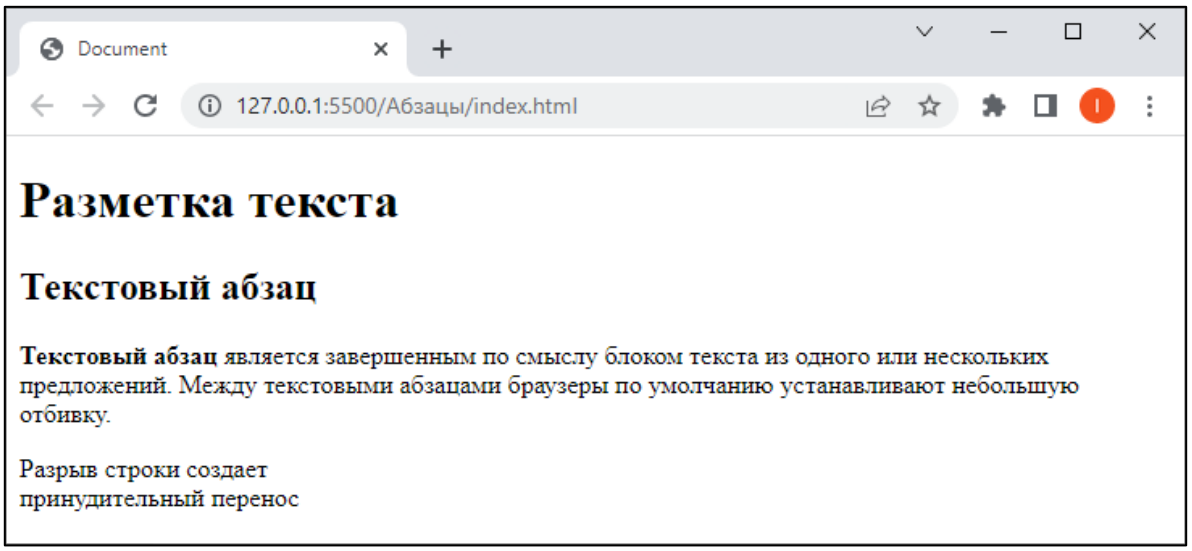


Рис. 2.90. Стандартное оформление абзацев

Акцентирование внимания

Определение

<dfn></dfn>

*Размечает новый **термин**. Обычно предполагается, что конкретное слово или словосочетание определяется этим тегом в документе только один раз.*

****

*Текст **высокой значимости**, на который необходимо сделать сильный акцент (предупреждение, замечание, термин).*

****

*Текст, на который стоит **акцентировать** внимание в предложении (абзаце).*

<del></del>

*Помечает текст как **удаленный** (например, потерявший актуальность). По умолчанию браузер его зачеркивает.*

<hr>

*Горизонтальная черта, обычно используемая в роли **тематического разделителя** абзацев.*

Обозначенные теги акцентируют внимание на важных деталях.

Глава 2\Абзацы\index.html

```
<h1>Разметка текста</h1>
<h2>Акцентирование внимания</h2>
<p><dfn>HTML5</dfn> - это новая платформа для разметки
веб-документов.</p>
<p><strong>Важно</strong>: уходя гасите свет!</p>
<p>При работе с тегами важна <em>аккуратность</em>.</p>
<p><del>Теги физической разметки позволяют форматировать
текст. Опишем их далее.</del></p>
<hr>
<p><em>Тематический разделитель</em> зачастую упрощает
восприятие информации.</p>
```

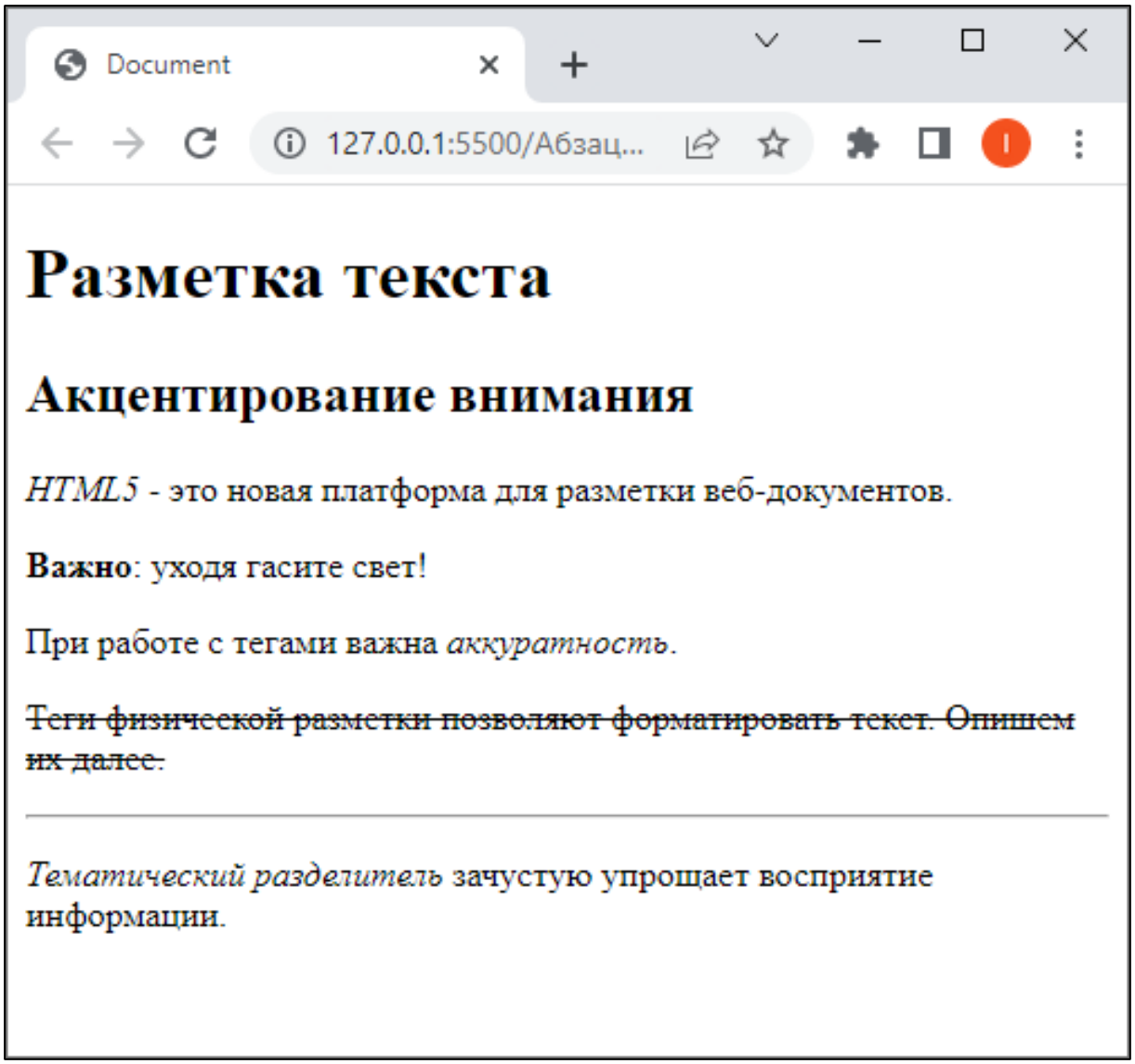


Рис. 2.91. Акцентирование внимание на тексте

Также отметим несколько тегов, которые в предыдущих версиях HTML использовались для форматирования текста, но не были удалены из HTML5 и теперь иногда задействуются для акцентирования внимания читателей.

При этом обозначенные ниже теги использовать необязательно и для поисковых систем они не несут полезной информации (только для пользователей).

Определение

Следующие теги до HTML5 использовались как теги физического форматирования.

****

Тег используется для привлечения внимания к тексту, однако важность или значение текста не играют роли. Браузеры по умолчанию отображают этот текст полужирным начертанием.

****

Тег используется для текста, который должен несколько отличаться от обычного (ключевые слова, термины, иностранные слова, названия и др.). Браузеры по умолчанию отображают этот текст курсивом.

<u></u>

Тег используется для разметки текста, который должен стилистически отличаться от обычного (помечать ошибки, имена собственные, иностранные слова.). Браузеры по умолчанию отображают этот текст подчеркнутым.

Глава 2\Абзацы\index.html

```
<h2>Акцент для читателя</h2>
<p><b>Полужирное начертание</b></p>
<p><i>Курсив</i></p>
<p><b><i>Полужирный курсив</i></b> или <i><b>аналогично
так</b></i></p>
<p>Акцентирование внимания <u>подчеркиванием</u></p>
```

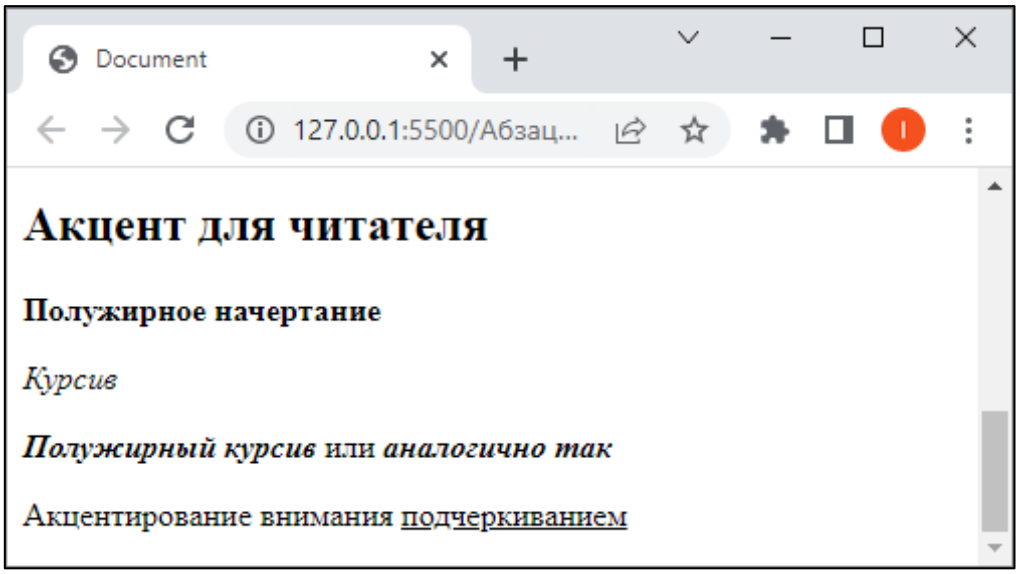


Рис. 2.92. Дополнительные приемы акцента внимания для читателя (с использованием тегов, выполнявших ранее роль элементов форматирования)

Это важно знать!

*Все перечисленные в этом пункте теги являются **строчными**, в отличие от **блочных**, к которыми относятся заголовки и абзацы. Блоки занимают доступную ширину тега родителя, а строчные элементы следуют подряд, как текст в строке.*

Приемы работы с Visual Studio Code

Обратите внимание, что редактор при наведении курсора на тег позволяет получить по ссылке справку на сайте MDN.

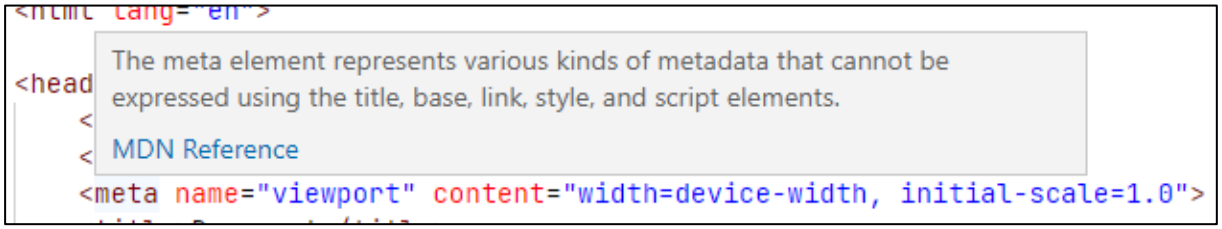


Рис. 2.93. Справка по тегу

Подтекст и контекст

Определение

`<abbr></abbr>`

*Тег определяет **аббревиатуру**, которую можно расшифровать в его атрибуте **`title`**. По наведению курсора на элемент отображается текст этой расшифровки.*

`<small></small>`

*Тег размечает текст в роли **подтекста**: сноски, пояснения, второстепенного комментария, уведомления об авторских правах и т.п.*

```
<sub></sub>
<sup></sup>
```

*Теги размечают текст относительно нижней и верхней линии строки, т.е. помечают его в роли **нижнего** и **верхнего индекса**. Браузеры по умолчанию делают их текст значительно меньше.*

До прихода спецификации HTML5 тег `<small></small>` имел антипод `<big></big>`, где оба служили для некоторого уменьшения / увеличения размера текста. В настоящее время `<big>` удален из языка как тег физического форматирования, а `<small>` продолжают использовать, однако уже в изложенном выше контексте.

Иногда `<small>` используется в конкретизации заголовков (в качестве подтемы).

Глава 2\Подтекст\index.html

```
<h1>Разметка текста</h1>
<h2>Текстовый абзац</h2>
<p><abbr title="Cascading Style Sheets">CSS</abbr> -
технология каскадных таблиц стилей.</p>

<h2>Прикладная информатика<br><small>Веб-
технологии</small></h2>

<p>Формула молекулы воды: H<sub>2</sub>O</p>
```

```
<p>В прямоугольном треугольнике сумма квадратов катетов
равна квадрату гипотенузы: a<sup>2</sup> + b<sup>2</sup>
= c<sup>2</sup>.</p>
```

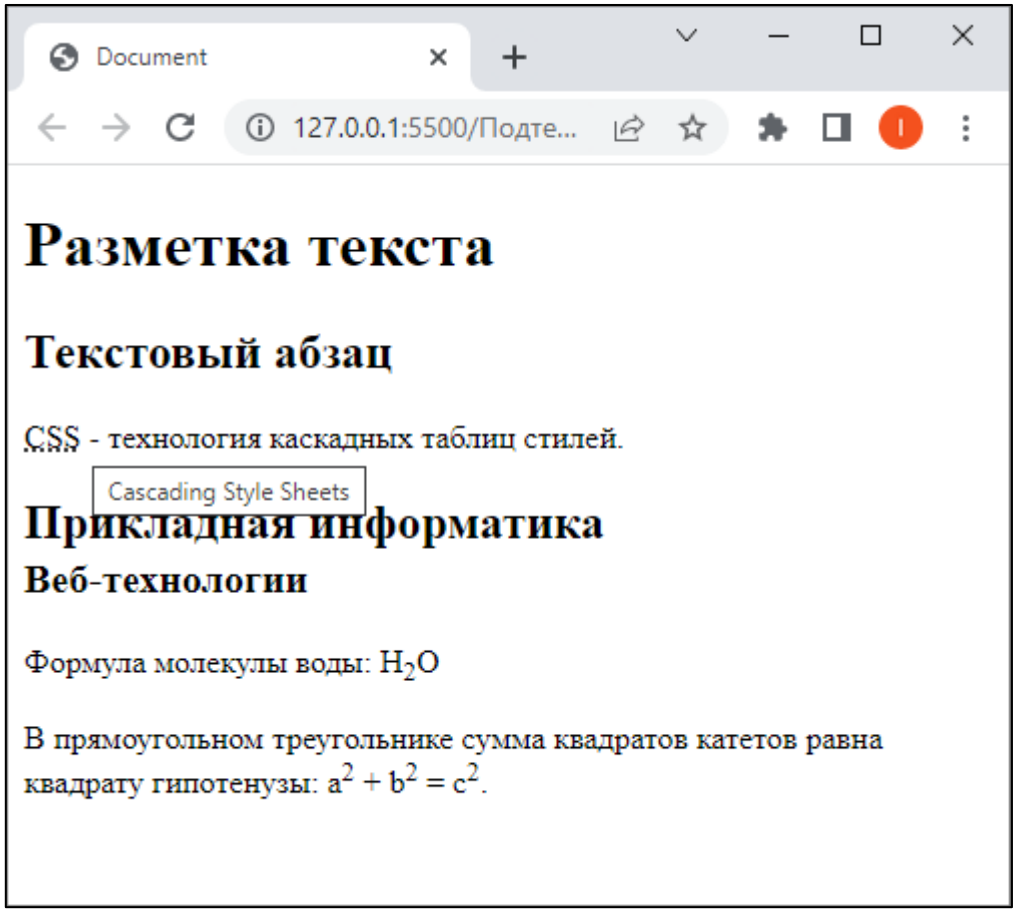


Рис. 2.94. Использование тегов для уточнения подтекста

Это важно знать!

В приведенных примерах обратите внимание, что:

- *текст переносится по строкам в зависимости от ширины окна браузера;*
- *любое количество пробелов между словами равносильно одному пробелу;*
- *пустые строки в разметке не создают вертикальный интервал.*

Цитирование

Определение

<blockquote></blockquote>

*Определяет **блок цитат** из другого источника. С помощью атрибута **cite** можно дополнительно сослаться на страницу первоисточника.*
По умолчанию браузеры отбивают блоку дополнительные отступы слева и справа, сверху и снизу. Также играет роль контейнера для следующих ниже тегов.

<q></q>

*Тег размечает **цитату в тексте**. По умолчанию окружается кавычками. В атрибуте **cite** также можно указать адрес источника цитаты.*

<cite></cite>

*Тег отмечает текст в качестве **цитаты** или **сноски** на другой источник. Браузеры по умолчанию отмечают их курсивом.*

Глава 2\Цитаты\index.html

```
<h1>Разметка текста</h1>
<h2>Цитаты</h2>
<p>Одна из 50 цитат о программировании всех времён.</p>
<blockquote cite="https://habr.com/ru/articles/275841/">
    <q>Не волнуйтесь, если что-то не работает.
    Если бы всё работало, вас бы уволили.</q>
    <br>
    <cite>Законы Мошера в разработке ПО</cite>
</blockquote>
```

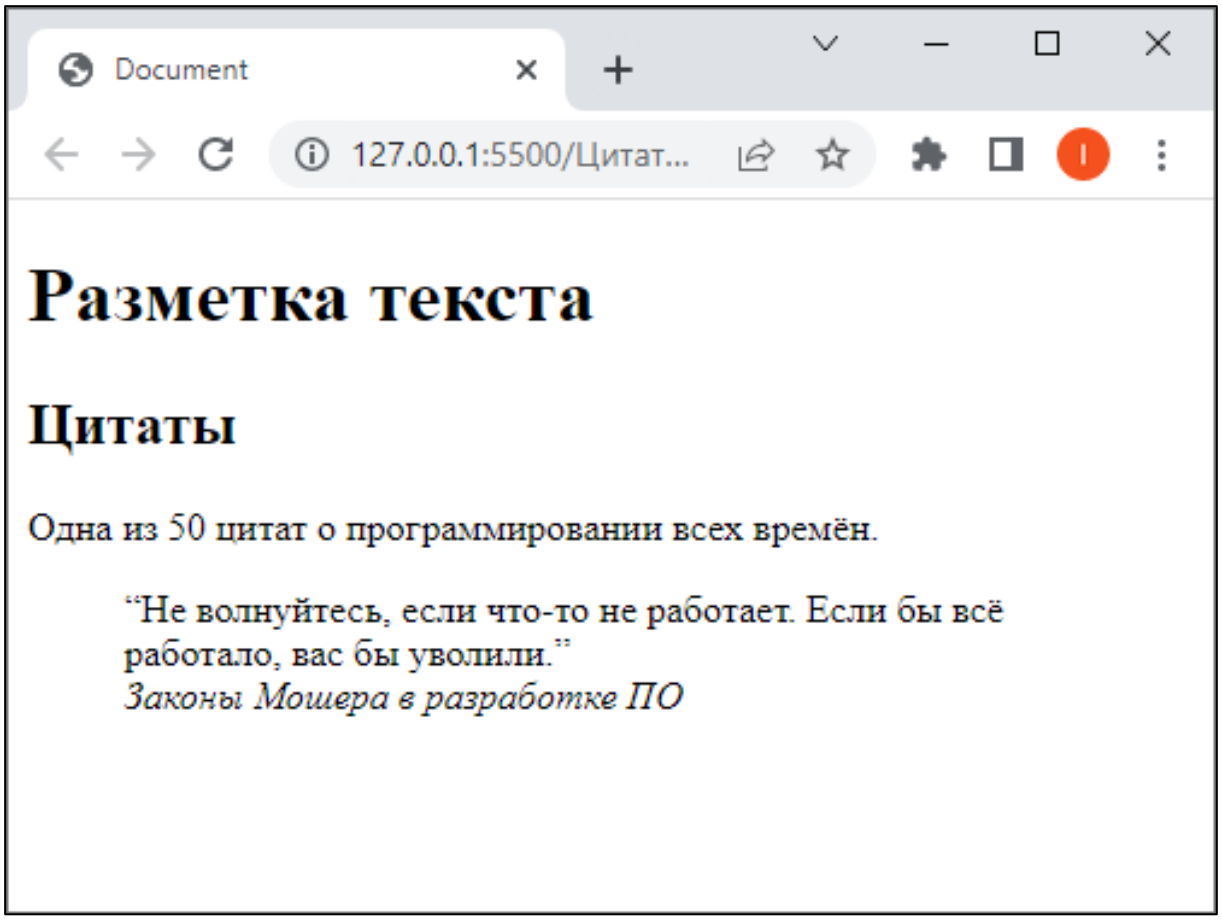


Рис. 2.95. Пример оформления цитат

Разметка листингов кода

Определение

```
<code></code>
```

*Контейнер помечает текст в роли **листинга программного кода**. Для сохранения отступов обычно используется в паре с `<pre>`. Браузеры обычно форматируют этот текст моноширинным шрифтом (Consolas, Courier или др.).*

```
<pre></pre>
```

*Тег размечает **предварительно отформатированный текст**. Учитывает все пробелы и пустые строки. Часто используется для оформления листингов программного кода.*

Разметка программного кода часто требуется для демонстрации примеров и фрагментов программ. В общем случае используются специальные CSS и JavaScript библиотеки, которые способны автоматически оформить код и подсвечивать синтаксис.

Глава 2\Листинги кода\index.html

```
<h1>Разметка текста</h1>
<h2>Оформление кода</h2>
<p>Поиск наибольшего из двух</p>
<p>Тег <strong>code</strong> не учитывает отступы.</p>
<code>
    double x = 3.4;
    double y = 5.3;

    if (x > y)
    {
        Console.Write("x > y");
    }
    else
    {
        Console.Write("x <= y");
    }
</code>
<p>Тег <strong>pre</strong> учитывает отступы.</p>
<pre>
    double x = 3.4;
    double y = 5.3;

    if (x > y)
    {
        Console.Write("x > y");
    }
    else
    {
        Console.Write("x <= y");
    }
</pre>
```

Рис. 2.96. Сравнение тегов <code> и <pre>

Другие теги

Определение

```
<!--  -->
```

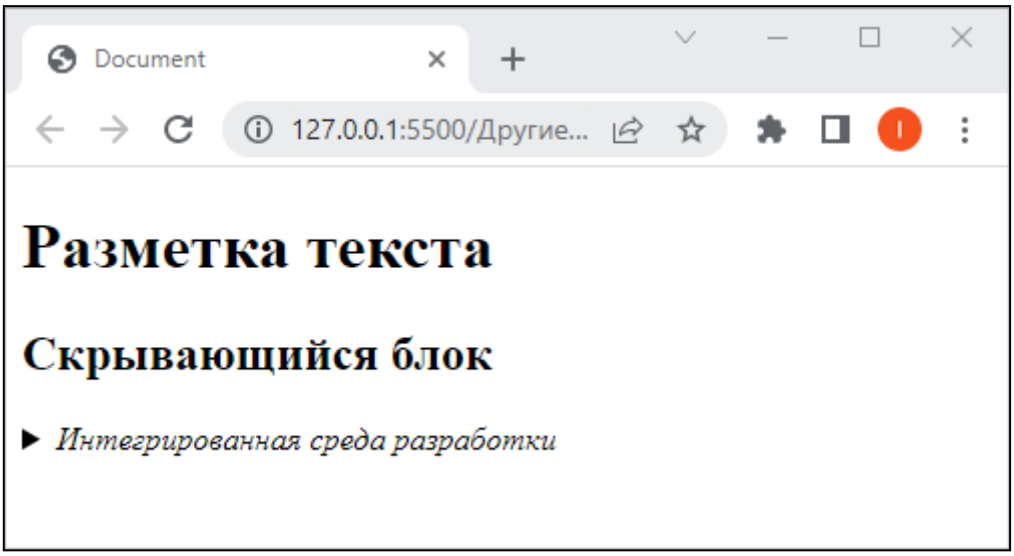
*Скобки **комментария** в разметочном коде.*
Текст внутри является комментарием, который не отображается в окне браузера и ни на что не влияет.
Можно использовать для «заморозки» или скрытия фрагмента разметки.

```
<details></details>
<summary></summary>
```

*Тег **<details>** создает блок, содержимое которого можно скрывать и раскрывать по нажатию. По умолчанию содержимое тега скрыто. Чтобы изменить статус, используется его атрибут **open**.*
*Тег **<summary>** содержит текст раскрывающегося блока, по которому необходимо нажимать для его активации.*

Глава 2\Любые теги\index.html

```
<h1>Разметка текста</h1>
<h2>Скрывающийся блок</h2>
<details>
    <summary><dfn>Интегрированная среда
    разработки</dfn></summary>
    <p><strong>Интегрированная среда разработки</strong>,
    IDE (от англ. Integrated Development Environment) –
    комплекс программных средств, предназначенный для
    разработки ПО.</p>
    <img src="IDE.png" alt="среда разработки"
width="100%">
</details>

<!--
<h2>Закомментированный код</h2>
<p>Этот фрагмент документа вы не увидите в браузере</p>
-->
```



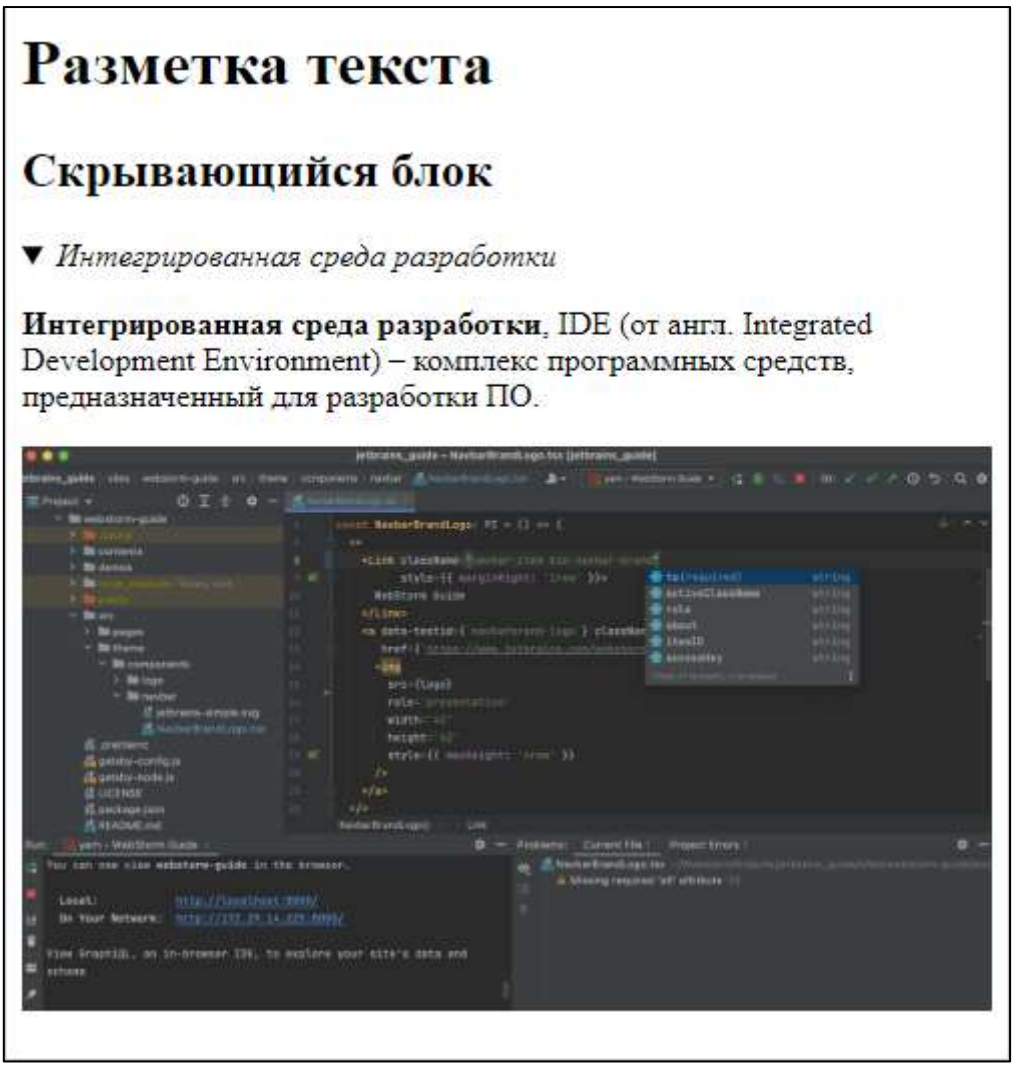


Рис. 2.97. В раскрывающийся блок можно размещать не только текст, но и разметку других элементов

Приемы работы с Visual Studio Code

Чтобы закомментировать выделенный фрагмент кода в Visual Studio Code, нажмите комбинацию **Ctrl** *+ /. Повторное нажатие удаляет символы комментирования.*
Комбинация работает для всех языков программирования и разметки (при этом для каждого выставляется свой тип комментария).

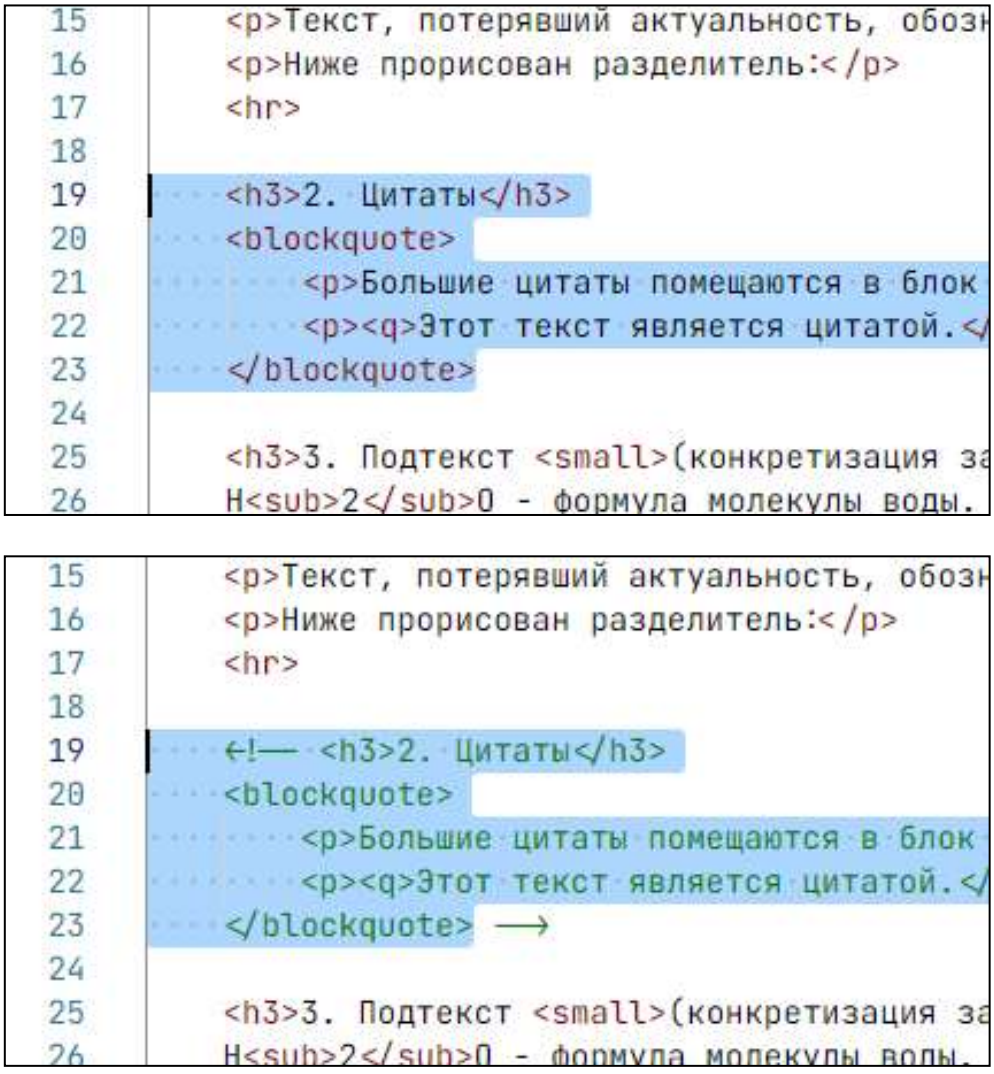


Рис. 2.98. Автоматическое комментирование фрагмента кода

Определение

`< >`

*Спецсимволы, которые являются **заменителями** угловых скобок < и >. Они необходимы, чтобы браузер смог отличить их от скобок тега, например, в случае вывода листинга HTML кода на странице. В этом случае говорят, что их **экранируют**.*

` `

*Кодовая инструкция, заменяющая один **пробел** (также **неразрывный пробел**).*

`&#код;`

Позволяет вставлять символ или иконку по HTML-коду.
Таблицы кодов символов можно найти в свободном доступе сети Интернет, например:

- *HTML-мнемоники: ссылка.*
- *Специальные символы: ссылка.*

Приемы работы с Visual Studio Code

*Выделенный текст можно перетаскивать зажатой **ЛКМ**.*

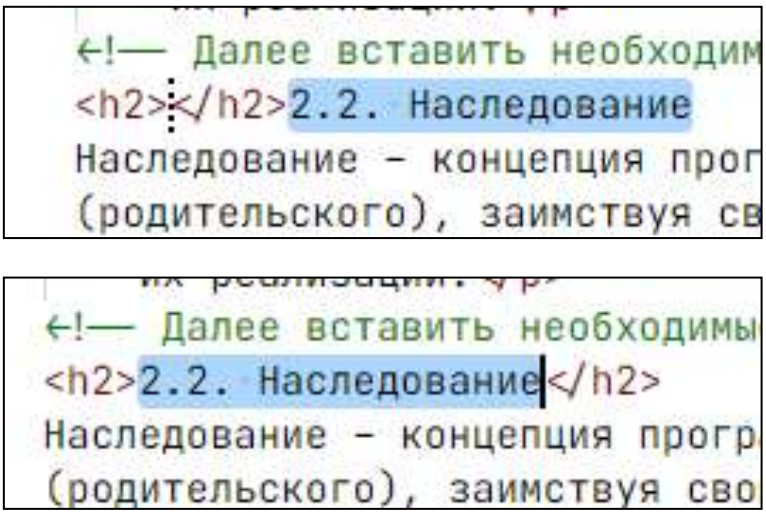


Рис. 2.99. Перетаскивание текста зажатием левой кнопки мыши

Примеры

Пример 1. Теги семантической разметки

В примере рассматриваются теги, позволяющие выделить структуру разметки текста. Здесь используются теги, которые поддерживаются спецификацией HTML5.

Результат изображен на рис. 2.100.

Глава 2\Семантическая разметка\index.html

```
<!DOCTYPE html>
<html lang="ru">

<head>
    <meta charset="UTF-8">
    <meta http-equiv="X-UA-Compatible" content="IE=edge">
    <meta name="viewport" content="width=device-width,
initial-scale=1.0">
    <title>Document</title>
</head>

<body>
    <h1>Разметка текста</h1>
    <h2>Логическое форматирование</h2>

    <h3>1. Общие теги</h3>
    <p>Данная категория тегов осуществляет
     <strong>семантическую разметку</strong> текста.</p>
    <p>Для акцента внимания на тексте поступим
    <em>следующим образом</em>.</p>
    <p>Текст, потерявший актуальность, обозначается
    <del>вот так</del>.</p>
    <p>Ниже прорисован разделитель:</p>
    <hr>
    <h3>2. Цитаты</h3>
    <blockquote>
        <p>Большие цитаты помещаются в блок
        &lt;blockquote&gt;</p>
        <p><q>Этот текст является цитатой.</q> (Подробнее
        у <cite>Якубовича Д.А. и
        Ероповой Е. С.</cite>)</p>
    </blockquote>

    <h3>3. Подтекст <small>(конкретизация
    заголовка)</small></h3>
```

```
    <p>
        H<sub>2</sub>O - формула молекулы воды.
        <br>
        a<sup>2</sup> + b<sup>2</sup> = c<sup>2</sup>
    </p>

    <h3>4. Оформление кода</h3>
    <p>Пример кода:</p>
    <code><pre>
    // фрагмент скрипта на языке JavaScript
    const box = document.getElementById('info-box');

    box.addEventListener('click', function(e) {
        box.innerHTML = "Hello!";
    });
    </pre></code>

    <!-- <h3>5. Комментирование</h3>
    <p>Эта часть разметки не отображается, поскольку
    закомментирована.</p> -->
</body>

</html>
```

Пример 2. Теги физической разметки

В этом примере используются теги физической разметки, которые отвечают за видоизменение текста. Многие из них удалены из HTML5, поэтому пример носит лишь ознакомительный характер.

Результат изображен на рис. 2.101.

Глава 2\Физическая разметка\index.html

```
<!DOCTYPE html>
<html lang="ru">

<head>
    <meta charset="UTF-8">
    <meta http-equiv="X-UA-Compatible" content="IE=edge">
    <meta name="viewport" content="width=device-width,
initial-scale=1.0">
    <title>Document</title>
</head>
```

```
<body>
    <h1>Разметка текста</h1>
    <h2>Физическое форматирование</h2>

    <h2>1. Начертание</h2>
    <b>жирный</b>
    <i>курсив</i>
    <s>зачеркнутый</s>
    <u>подчеркнутый</u>
    <tt>печатная машинка</tt>
    <font color="orange" size="6"
face="Tahoma">настраиваемый</font>

    <h2>2. Комбинирование</h2>
    <b><i>жирный курсив</i></b> или
    <i><b>курсивный жирный</b></i>
</body>

</html>
```

Это важно знать!

В стандарте HTML5 использование тегов физического форматирования не рекомендуется!
Они были внедрены в HTML ранее и продолжают распознаваться браузерами исключительно в целях совместимости старых сайтов.
В современной верстке веб-страниц оформление элементов настраивается CSS-стилями.

Кроме того, избегайте использования атрибутов тега, отвечающих за визуальное оформление (шрифт, цвет текста или фона и т.п.):

- *это нарушает саму концепцию HTML как языка логической разметки;*
- *многие из таких атрибутов удалены из HTML5 как устаревшие;*
- *для визуального оформления используются технология CSS (следующая глава).*

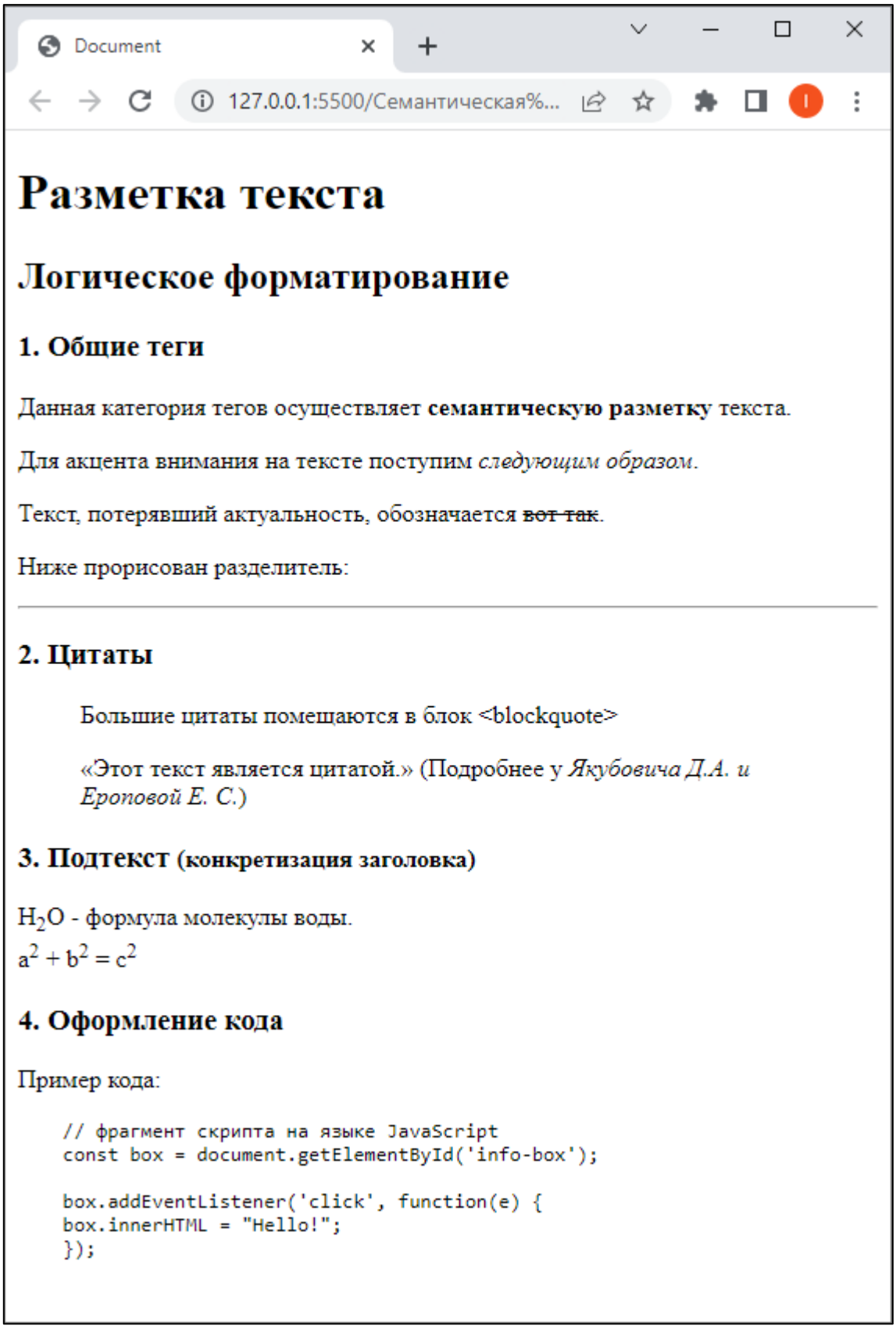


Рис. 2.100. Пример 1: семантическая разметка текста

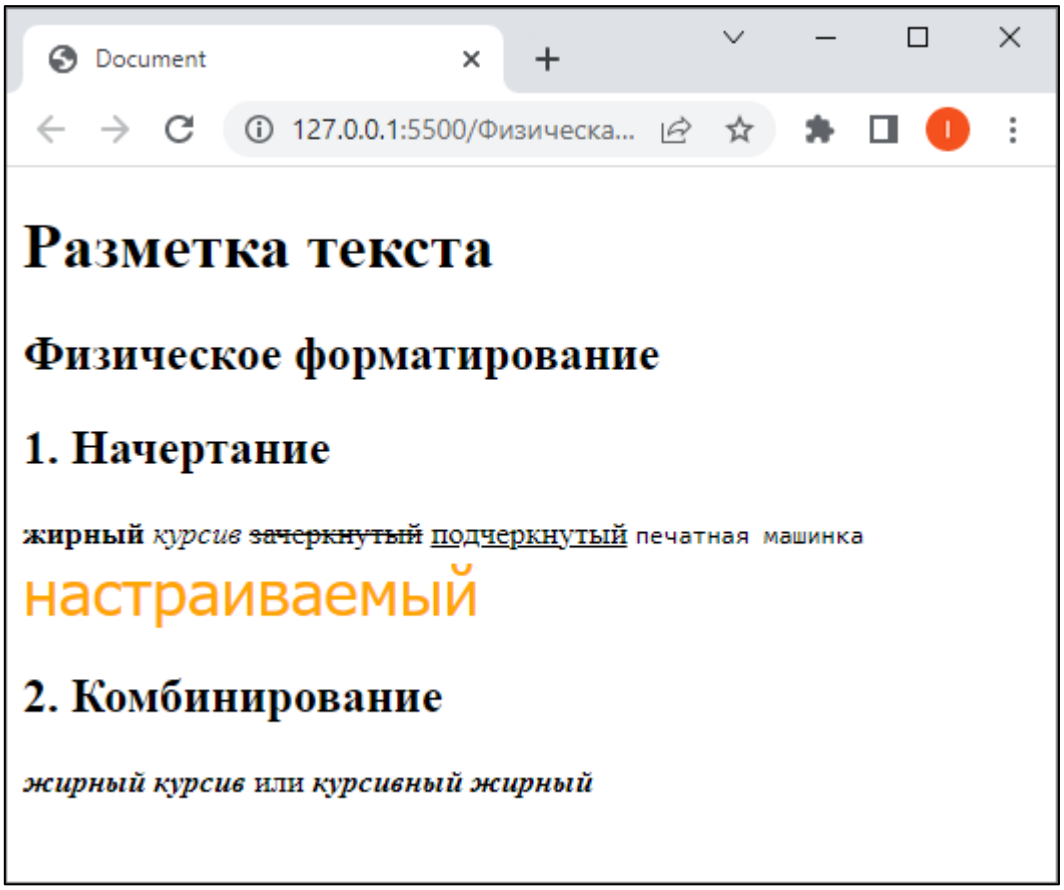


Рис. 2.101. Пример 2: физическая разметка текста

Приемы работы с Visual Studio Code

Для сдвига выделенного текста вправо / влево на размер табуляции можно использовать ***Tab*** */* ***Shift*** *+* ***Tab****.*

Рис. 2.102. Отступы влево и вправо с помощью *Tab*

Приемы работы с Visual Studio Code

Чтобы убрать горизонтальное прокручивание кода в VSC, нажмите комбинацию клавиш ***Alt*** *+* ***Z****. Теперь разметка будет переноситься согласно ширине окна редактора (рис. 2.103).*

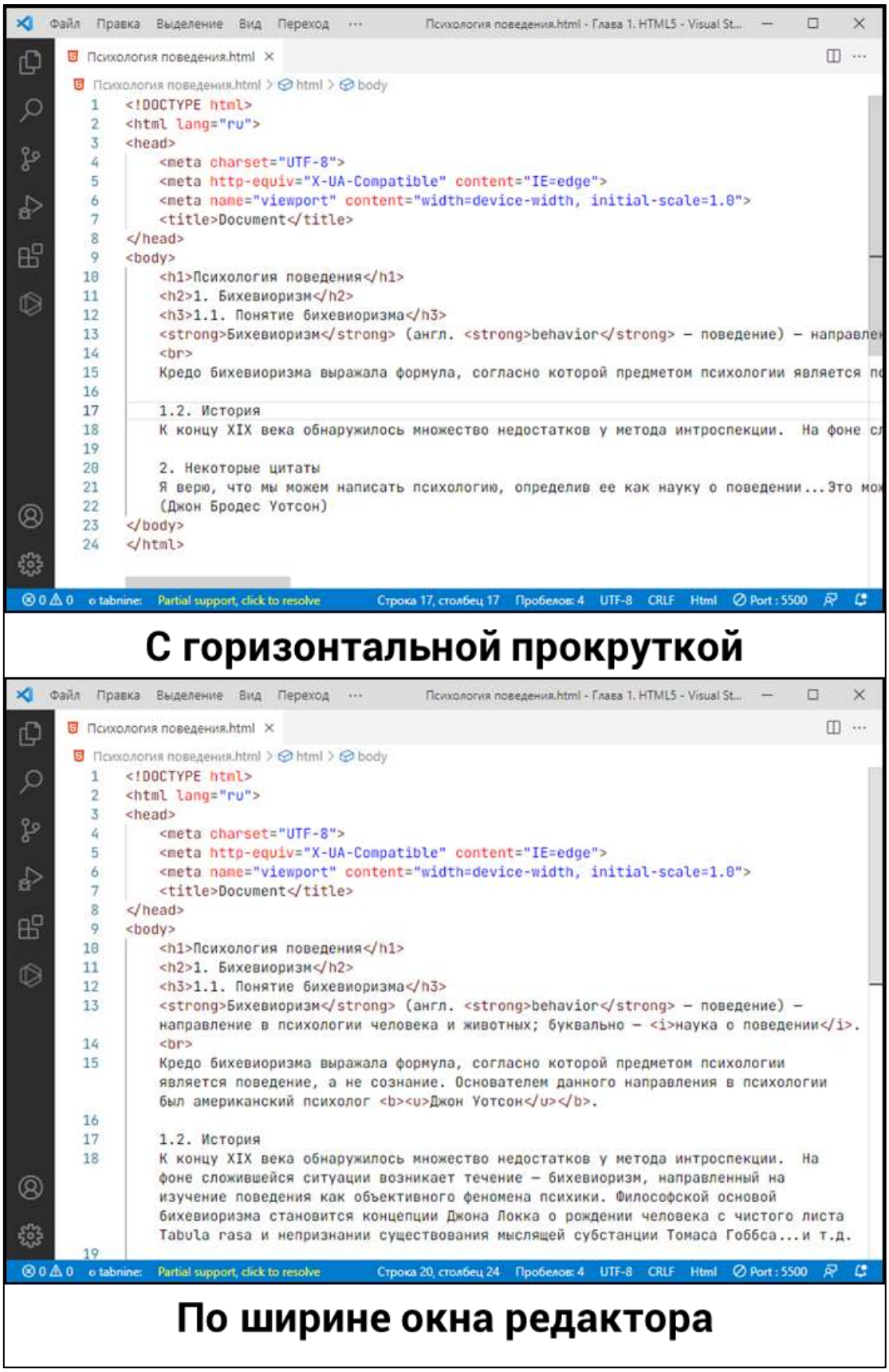


Рис. 2.103. Режим отображения разметки по горизонтали

Вопросы для самопроверки

1. В чем отличие между тегами физической и логической разметки документа?
2. Почему в спецификации HTML5 отказались от тегов физической разметки?
3. Перечислите теги, используемые в разметке заголовков.
4. Опишите особенности работы с абзацами.
5. Какие теги позволяют акцентировать внимание?
6. Приведите примеры ситуаций, когда могут быть использованы теги подтекста.
7. Как оформить листинги программного кода, сохранив отступы согласно стилистике языка программирования?

Практикум

Задание 1

1. Создайте каталог *ООП*.
2. Вставьте в него HTML-файл с типовой разметкой.
3. Используя изученные теги логического и при необходимости физического форматирования, завершите разметку следующей веб-страницы (по образцу, изображенному на рис. 2.104-рис. 2.105).

Задание 2

1. Сделайте копию каталога *ООП*. Переименуйте его в *ООП 2*.
2. Закомментируйте часть кода разметки, чтобы скрыть содержимое третьего и четвертого пункта (рис. 2.106).

Задание 3

1. Сделайте копию каталога *ООП 2*. Переименуйте его в *ООП 3*.
2. Используя возможности раскрывающихся блоков, сделайте описание каждого принципа ООП более компактным (рис. 2.107).

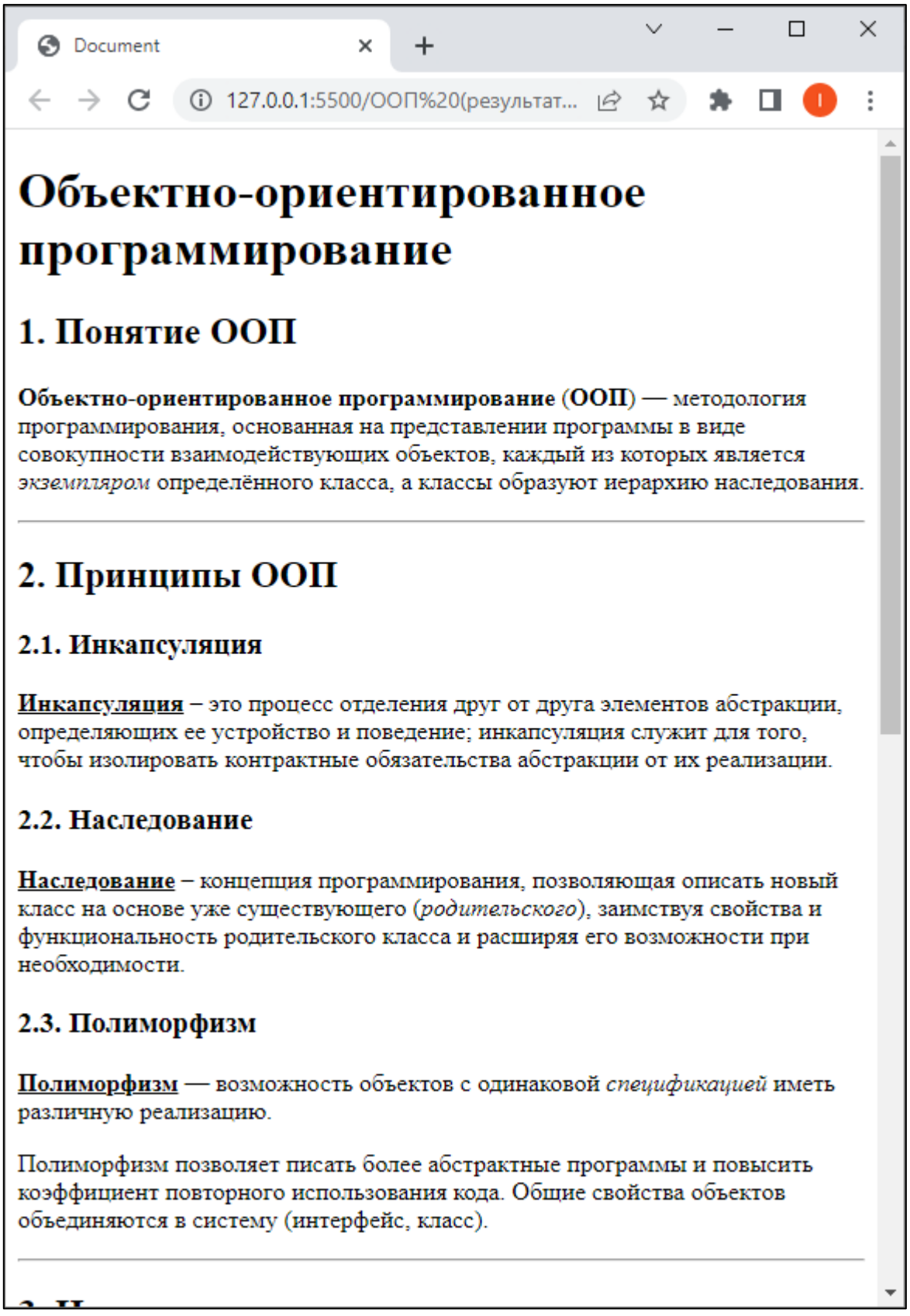


Рис. 2.104. Задание 1: первая часть веб-страницы

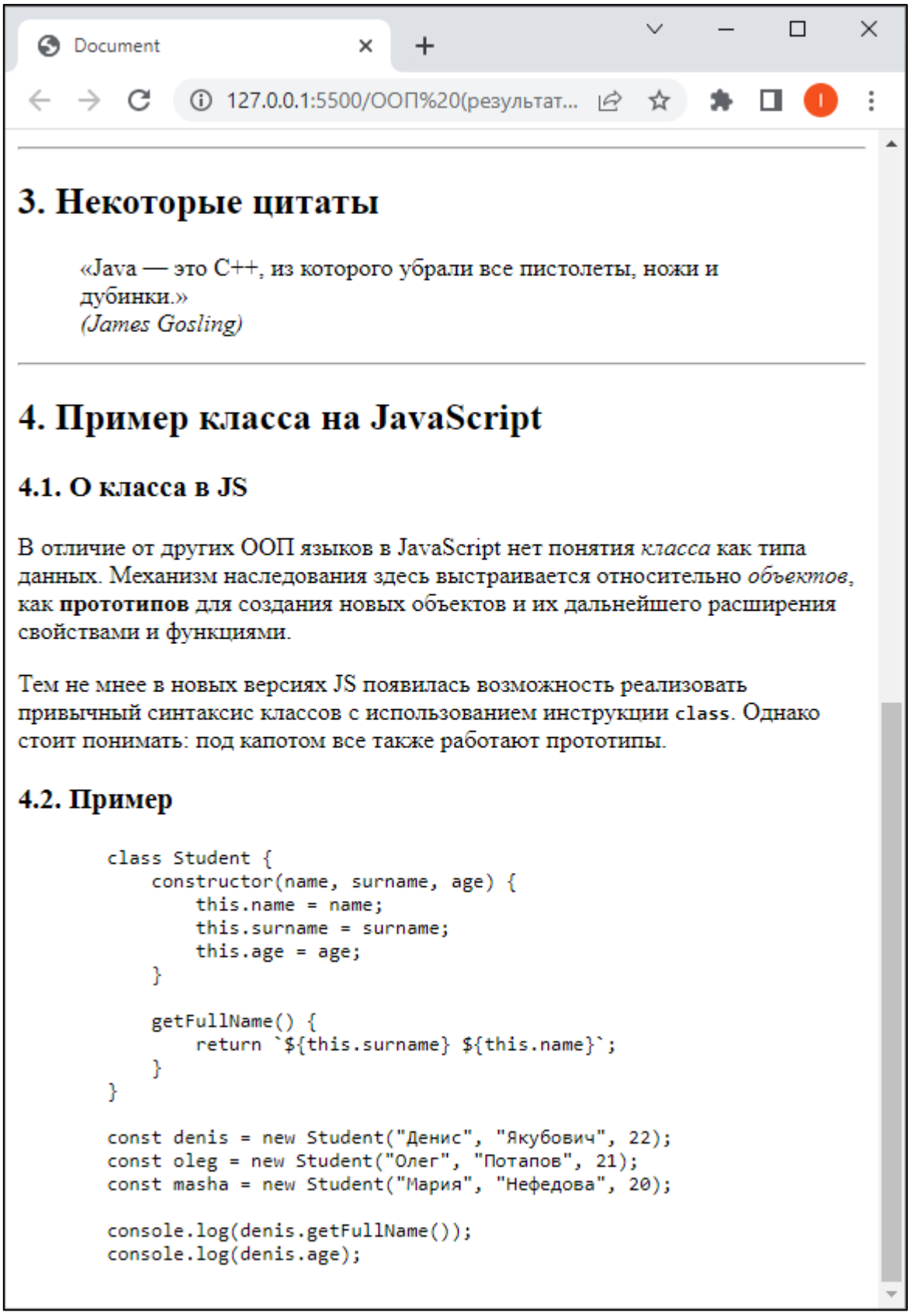
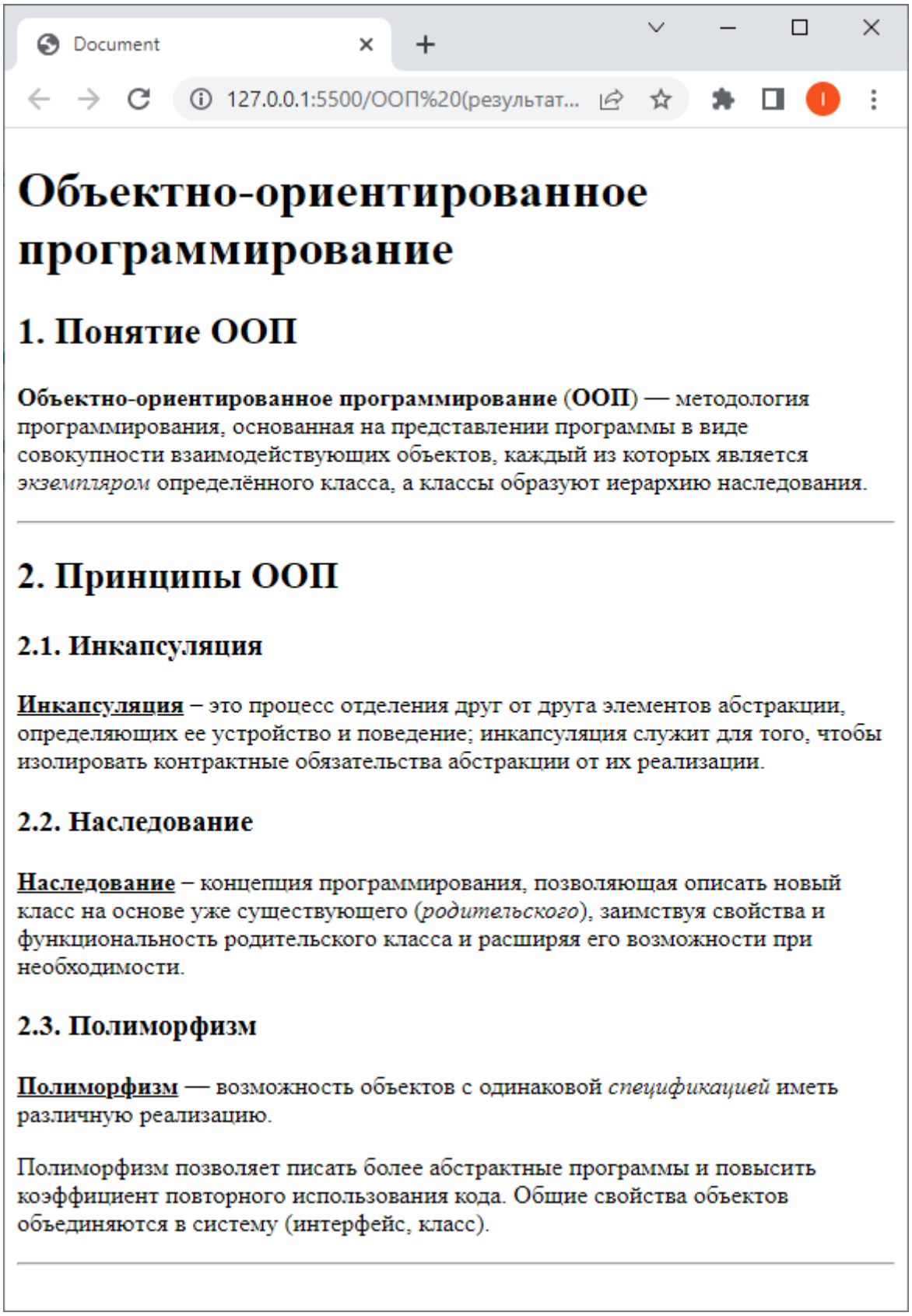


Рис. 2.105. Задание 1: вторая часть веб-страницы



Объектно-ориентированное программирование

1. Понятие ООП

Объектно-ориентированное программирование (**ООП**) — методология программирования, основанная на представлении программы в виде совокупности взаимодействующих объектов, каждый из которых является *экземпляром* определённого класса, а классы образуют иерархию наследования.

---

2. Принципы ООП

2.1. Инкапсуляция

Инкапсуляция – это процесс отделения друг от друга элементов абстракции, определяющих ее устройство и поведение; инкапсуляция служит для того, чтобы изолировать контрактные обязательства абстракции от их реализации.

2.2. Наследование

Наследование – концепция программирования, позволяющая описать новый класс на основе уже существующего (*родительского*), заимствуя свойства и функциональность родительского класса и расширяя его возможности при необходимости.

2.3. Полиморфизм

Полиморфизм — возможность объектов с одинаковой *спецификацией* иметь различную реализацию.

Полиморфизм позволяет писать более абстрактные программы и повысить коэффициент повторного использования кода. Общие свойства объектов объединяются в систему (интерфейс, класс).

Рис. 2.106. Задание 2: результат после комментирования

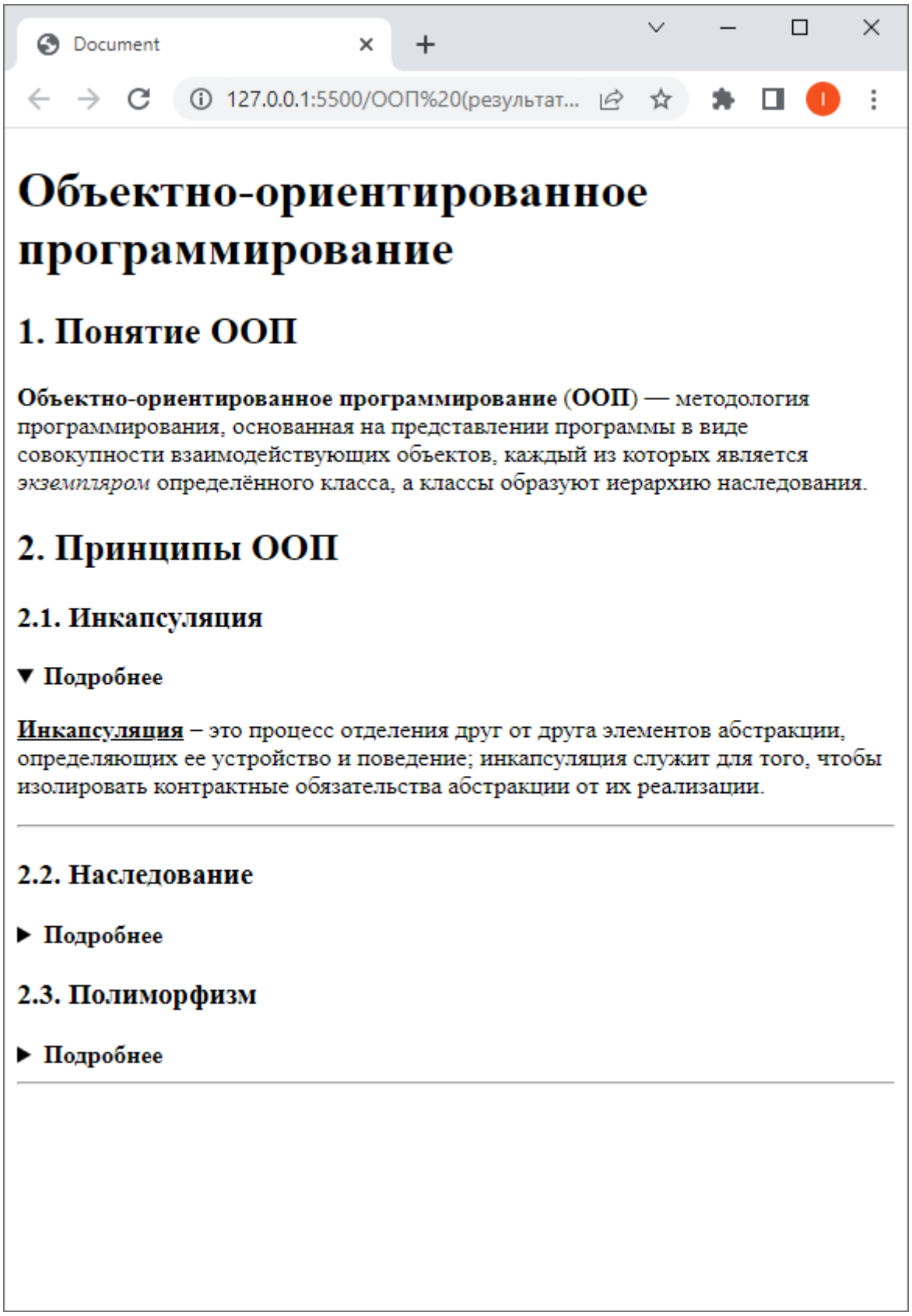


Рис. 2.107. Задание 3: добавление раскрывающихся блоков

2.6. Разметка списков

2.6.1. Виды списков в HTML

Списки в разметке текста

Определение

Список (перечисление) *– это способ описания данных, выделенных по общему признаку. Каждый пункт списка помечается специальным символом – маркером.*

Списки являются одной из форм представления текстовых данных, предполагающих упорядочивание или отражение зависимости элементов. Каждый пункт HTML-списка по существу является отдельным абзацем, которому может быть задан отступ первой строки, интервал между соседним пунктами, а также расстояние между маркером и первой строкой.

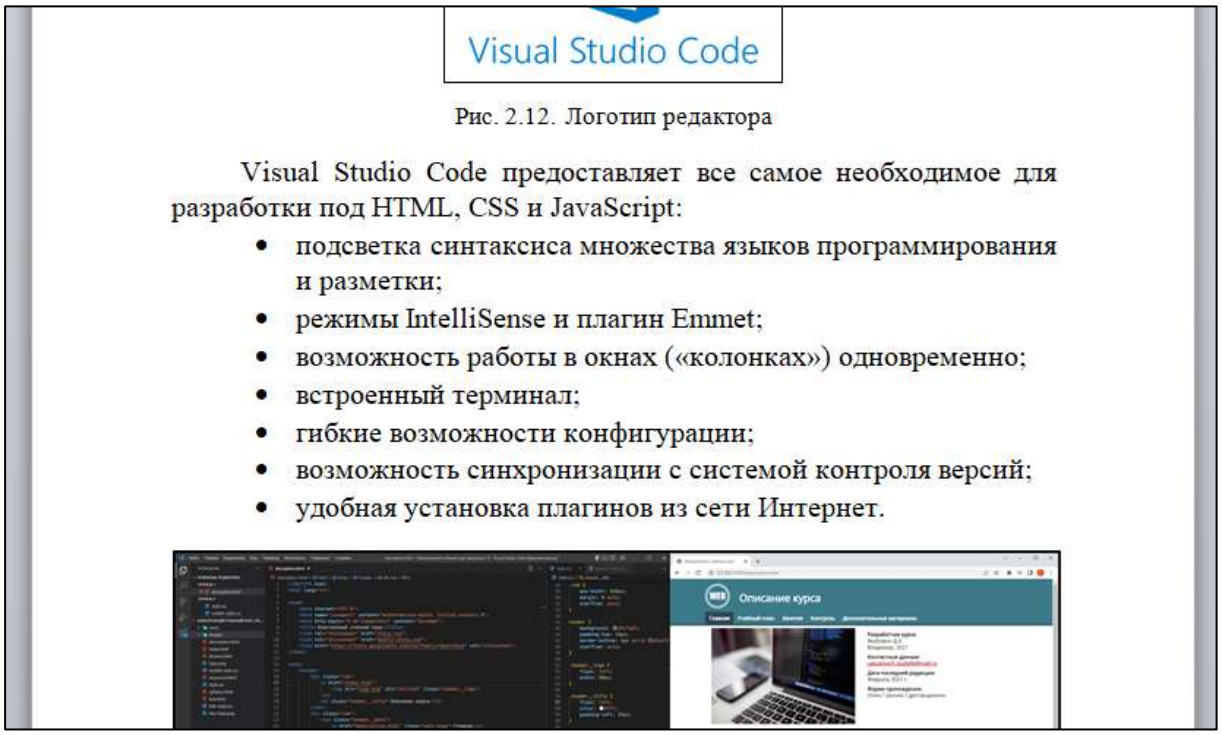


Рис. 2.108. Пример оформления списка в текущем документе

Виды списков

HTML поддерживает разметку трех видов списков:

1. нумерованного (упорядоченного);
2. маркированного (неупорядоченного);
3. списка определений.

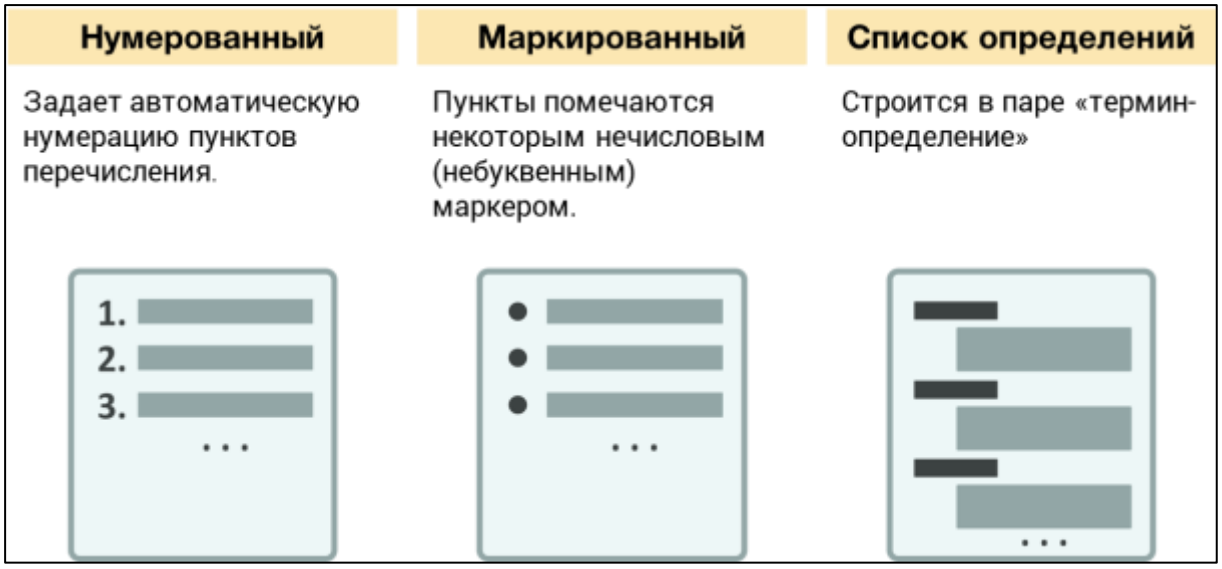
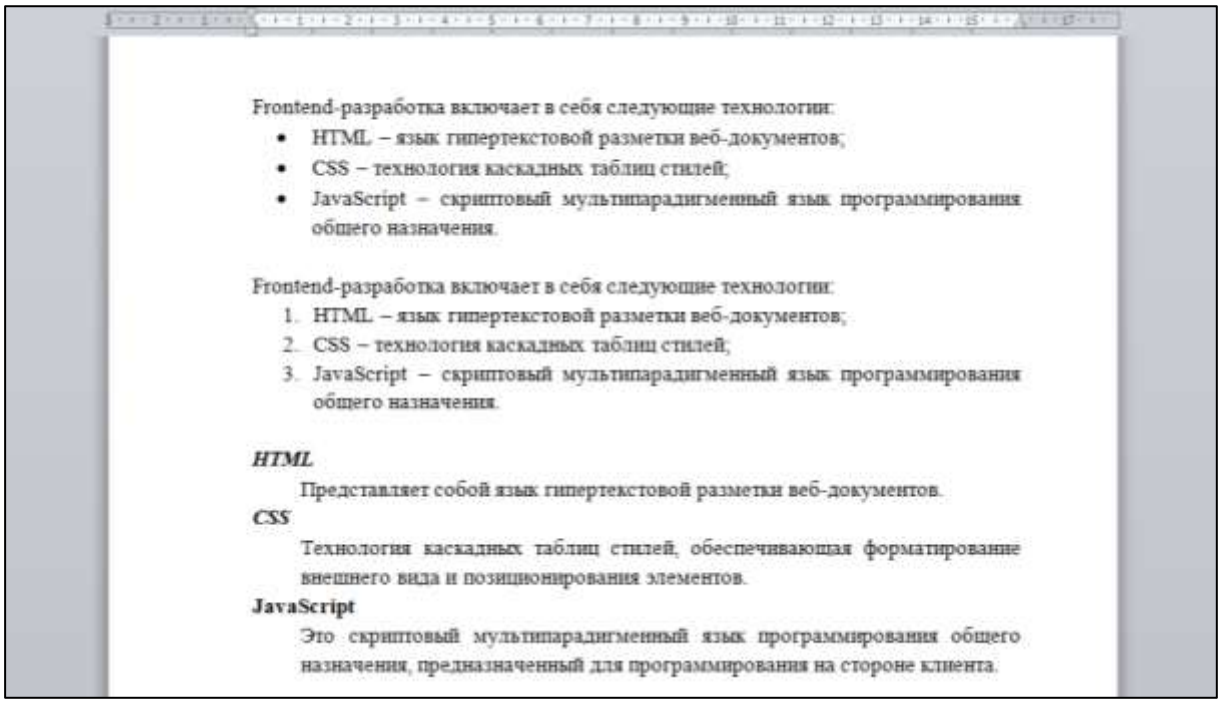


Рис. 2.109. Виды HTML-списков (кратко)



Frontend-разработка включает в себя следующие технологии:

- HTML – язык гипертекстовой разметки веб-документов;
- CSS – технология каскадных таблиц стилей;
- JavaScript – скриптовый мультипарадигменный язык программирования общего назначения.

Frontend-разработка включает в себя следующие технологии:

1. HTML – язык гипертекстовой разметки веб-документов;
2. CSS – технология каскадных таблиц стилей;
3. JavaScript – скриптовый мультипарадигменный язык программирования общего назначения.

HTML

Представляет собой язык гипертекстовой разметки веб-документов.

CSS

Технология каскадных таблиц стилей, обеспечивающая форматирование внешнего вида и позиционирования элементов.

JavaScript

Это скриптовый мультипарадигменный язык программирования общего назначения, предназначенный для программирования на стороне клиента.

Рис. 2.110. Виды списков на конкретном примере

2.6.2. Маркированный список

Понятие

Определение

Маркированный список *– это неупорядоченный перечень связанных по смыслу элементов.*

```
<ul></ul>
```

Тег является ***контейнером*** *для элементов маркированного списка.*

```
<li></li>
```

Тег размечает один ***элемент списка****. Должен быть расположен внутри контейнера* **`<ul></ul>`***. Чтобы изменить маркер по умолчанию, можно использовать атрибут* **`type`** *(однако рекомендуется использовать CSS).*

Разметка

Пример разметки маркированного списка (см. рис. 2.111):

Глава 2\Виды списков\index.html

```
<p>Frontend-разработка включает в себя следующие
технологии:</p>
<ul>
    <li>HTML – язык гипертекстовой разметки
    веб-документов;</li>
    <li>CSS – технология каскадных таблиц стилей;</li>
    <li>JavaScript – скриптовый мультипарадигменный язык
    программирования общего назначения.</li>
</ul>
```

С помощью атрибут **`type`** со значениями `disc`, `cirle` либо `square` можно изменить тип маркера:

```
<ul type="square"></ul>
```

Также атрибут `type` доступен и для каждого отдельного элемента `<li></li>` (однако разные маркеры в одном списке используются крайне редко).

2.6.3. Нумерованный список

Понятие

Определение

Нумерованный список *– это упорядоченный перечень связанных по смыслу элементов.*

```
<ol></ol>
```

Тег является ***контейнером*** *для элементов нумерованного списка.*

```
<li></li>
```

Тег размечает один ***элемент списка****. Должен быть расположен внутри контейнера* ***
****. Браузер автоматически нумерует элементы в порядке следования. Изменить тип номера поможет атрибут* ***type****, а инвертировать номера в порядке убывания –* ***reversed****.*

Разметка

Пример разметки нумерованного списка (см. рис. 2.112):

Глава 2\Виды списков\index.html

```
<p>Frontend-разработка включает в себя следующие
технологии:</p>
<ol>
    <li>HTML – язык гипертекстовой разметки веб-
документов;</li>
    <li>CSS – технология каскадных таблиц стилей;</li>
    <li>JavaScript – скриптовый мультипарадигменный язык
    программирования общего назначения.</li>
</ol>
```

Атрибут **type** позволит изменить вид нумерации:

```
<ol type="A" reversed>
```

Также в атрибуте **start** можно указать начальное значение нумерации:

```
<ol type="i" start="10"></ol>
```

2.6.4. Список определений

Понятие

Определение

Список определений *– это неупорядоченный перечень связанных по смыслу элементов, формируемых в виде ключевых значений и их описания.*

```
<dl></dl>
```

Тег является ***контейнером*** *для элементов списка определений.*

```
<dt></dt>
<dd></dd>
```

*Первый тег размечает «****термин****», а второй – формулирует его* ***описание****. Теги идут подряд и каждая пара считается единым элементом. По умолчанию браузер добавляет блоку описания небольшой отступ слева.*
Кроме того, одному термину можно дать несколько отдельных описаний.

Разметка

Пример разметки списка определений (см. рис. 2.113):

Глава 2\Виды списков\index.html

```
<dl>
    <dt><strong>HTML</strong></dt>
    <dd>Представляет собой язык гипертекстовой разметки
    веб-документов.</dd>
    <dt><strong>CSS</strong></dt>
    <dd>Технология каскадных таблиц стилей,
    обеспечивающая форматирование внешнего вида и
    позиционирования элементов.</dd>
    <dt><strong>JavaScript</strong></dt>
    <dd>Это скриптовый мультипарадигменный язык
    программирования общего назначения, предназначенный
    для программирования на стороне клиента.</dd>
</dl>
```

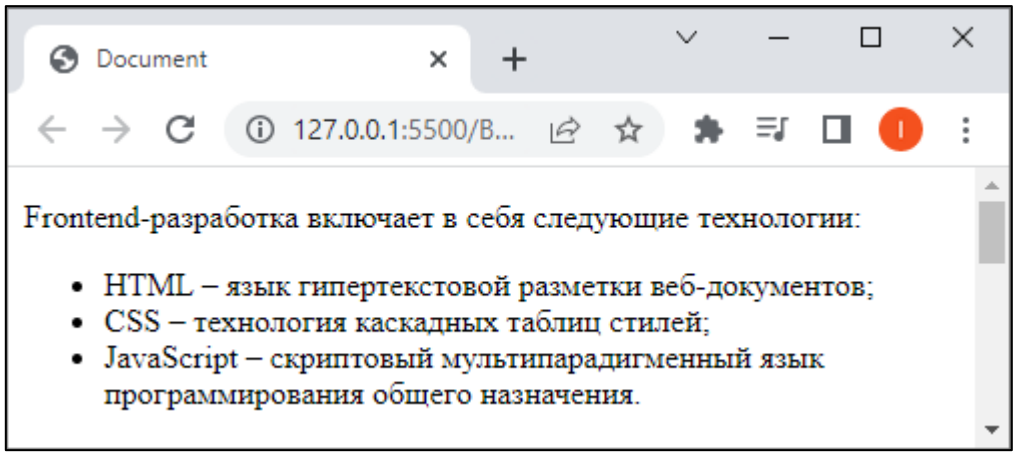


Рис. 2.111. Пример разметки маркированного списка

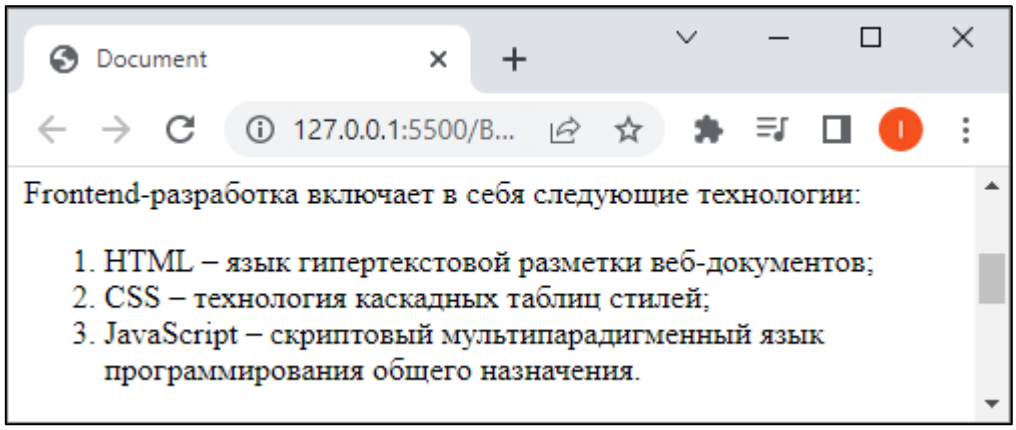


Рис. 2.112. Пример разметки нумерованного списка

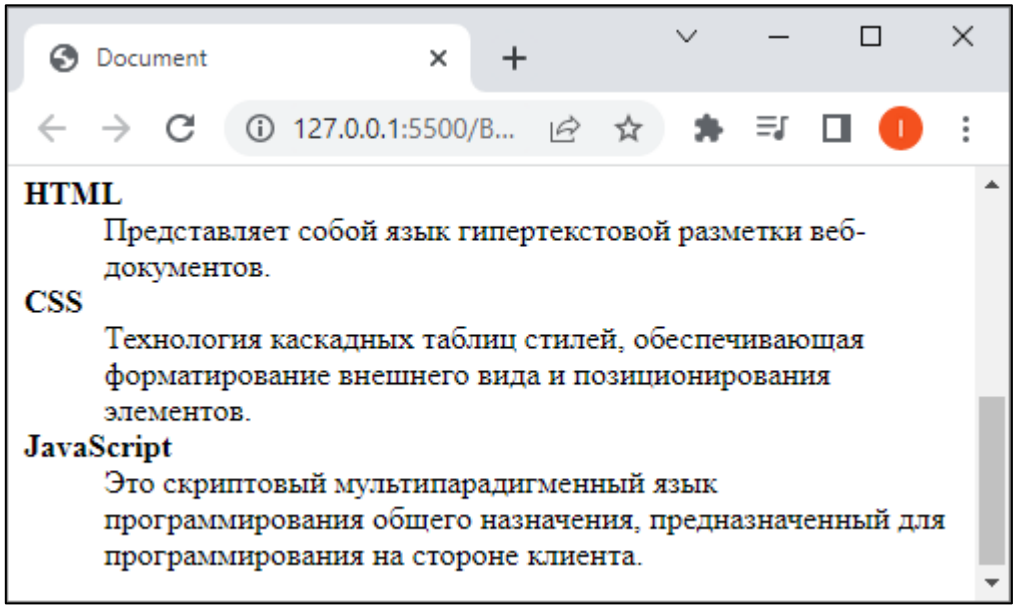


Рис. 2.113. Пример разметки списка определений

Приемы работы с Visual Studio Code

Visual Studio Code позволяет быстро сгенерировать шаблон списка для последующего заполнения. Для этого введите команду вида

```
ul>li*5
```

*и нажмите клавишу **TAB**. В результате получится следующий результат:*

```
<ul>
    <li></li>
    <li></li>
    <li></li>
    <li></li>
    <li></li>
</ul>
```

*Приведенный синтаксис является частью встроенного в редактор плагина **Emmet**, который существенно ускоряет набор HTML-кода за счет аббревиатур.*
*Так символ **>** означает вложение, а ***5** – продублировать элемент 5 раз.*
Кстати говоря, при вводе и автозавершении тегов вы его уже активно используете. Разумеется, это работает для всех тегов.
Emmet существенно ускоряет верстку страниц. К возможностям плагина мы будем обращаться еще не раз.

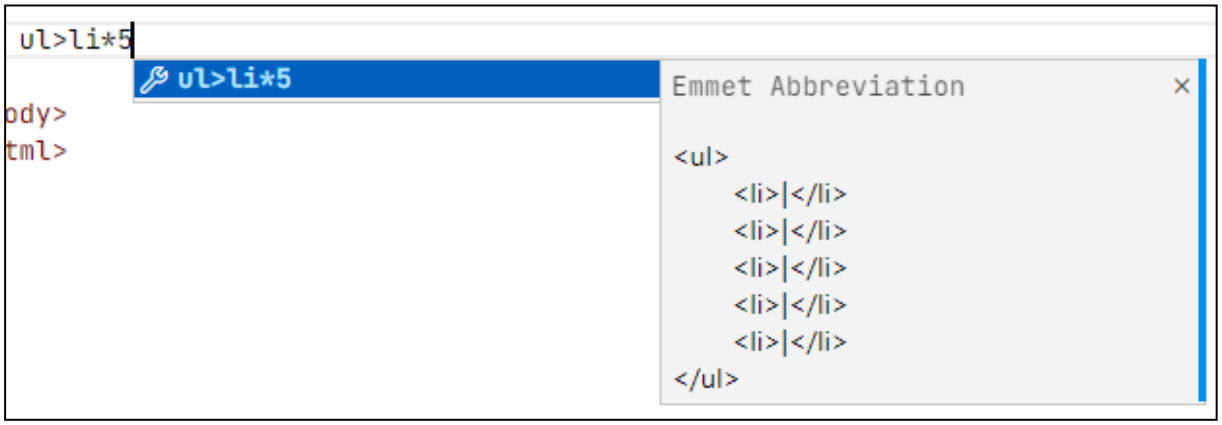


Рис. 2.114. Emmet в формировании структуры списка

2.6.5. Многоуровневые списки

Понятие

Определение

Многоуровневый список *– это список, допускающий создание списков для отдельных элементов.*
HTML не предусматривает отдельный синтаксис для многоуровневых списков. Вместо этого допускается вложение списков в отдельные элементы `<li></li>`.
Это позволяет комбинировать разные типы списков в единую многоуровневую структуру.

Разметка

При организации вложенных списков важно соблюдать следующие правила:

1. вложенный список должен располагаться непосредственно в том пункте `<li></li>`, к которому он относится;
2. вложенный список не должен находиться между соседними пунктами `<li></li>` или идти «вразрез» своего родителя (это структурная ошибка в разметке).

```
<ol>
    <li>Пункт 1</li>
    <li>Пункт 2
        <ul>
            <li>Пункт 2.1</li>
            <li>Пункт 2.2</li>
        </ul>
    </li>
    <li>Пункт 3</li>
</ol>
```

Вложенный список полностью во втором пункте

1. Пункт 1
2. Пункт 2
   - Пункт 2.1
   - Пункт 2.2
3. Пункт 3

Рис. 2.115. Порядок вложения: вложенный список находится целиком в пункте 2 основного списка

Приведем практический пример использования вложенных списков (см. рис. 2.116).

Глава 2\Многоуровневые списки\index.html

```
<h1>Дисциплины 6 семестра</h1>

<ul>
    <li>
        <strong>Экзамены</strong>
        <ol>
            <li>Объектно-ориентированное
                программирование</li>
            <li>Численные методы и исследование
                операций</li>
            <li>Теория вероятностей и математическая
                статистика</li>
        </ol>
    </li>

    <li>
        <strong>Зачеты</strong>
        <ol>
            <li>Методика обучения информатике</li>
            <li>Методика обучения математике</li>
            <li>Основы вожатской деятельности</li>
            <li>Основы научно-исследовательской
                деятельности</li>
        </ol>
    </li>

    <li>
        <strong>Дифференцированные зачеты</strong>
        <ol>
            <li>Педагогика</li>
            <li>Современные методы построения трехмерных
                фигур</li>
        </ol>
    </li>
</ul>
```

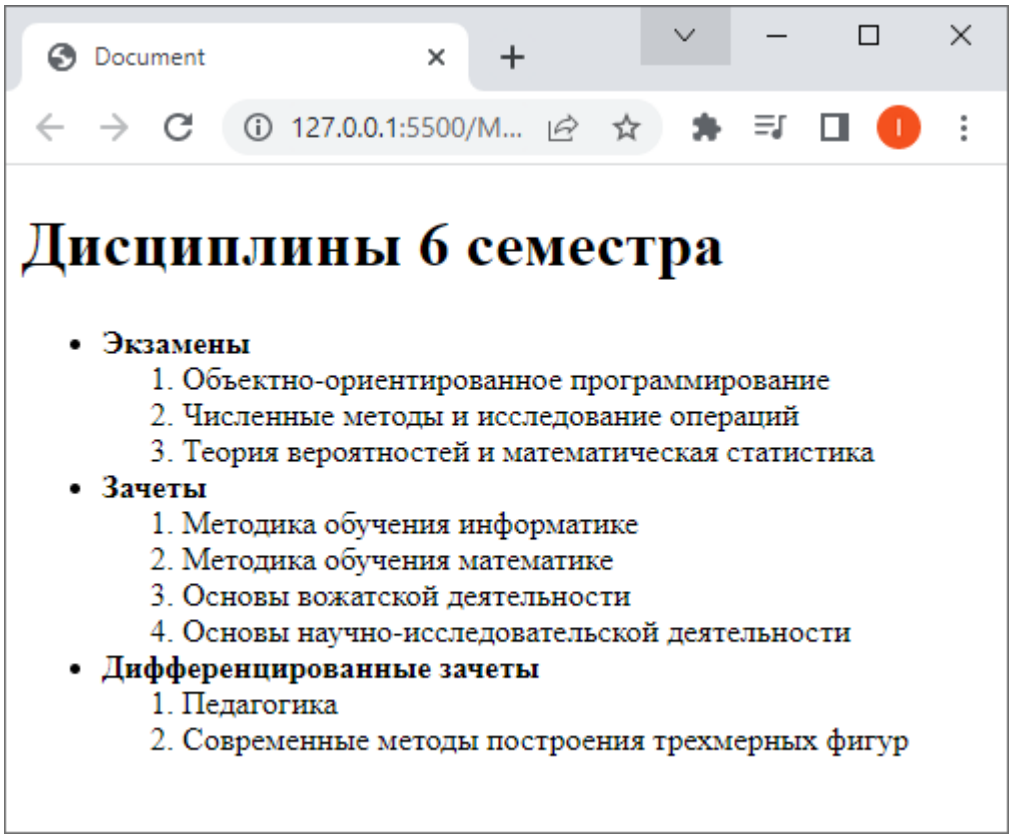


Рис. 2.116. Пример многоуровневого списка

Вопросы для самопроверки

1. Какие виды списков поддерживаются в HTML?
2. Опишите особенности разметки маркированных и нумерованных списков.
3. В каких случаях удобно использовать списки определений?
4. Возможно ли изменить маркер или номер списка? Если да, то каким образом?
5. Приведите пример разметки списка, в котором есть трехуровневое вложение.

Практикум

Задание 1

1. Создайте каталог *Работа со списками* и типовой HTML-файл со стандартной начальной разметкой.
2. Осуществите разметку маркированного списка из 5-7 известных цитат людей.

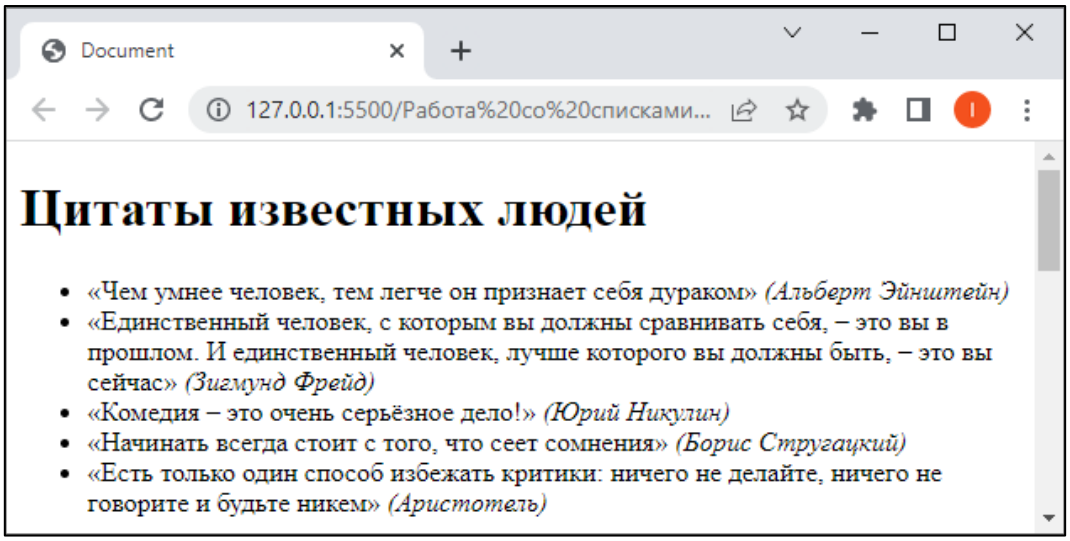


Рис. 2.117. Образец выполнения задания 1

Задание 2

1. Отделите предыдущий список тематическим разделителем.
2. Скопируйте созданный список и измените его на нумерованный.

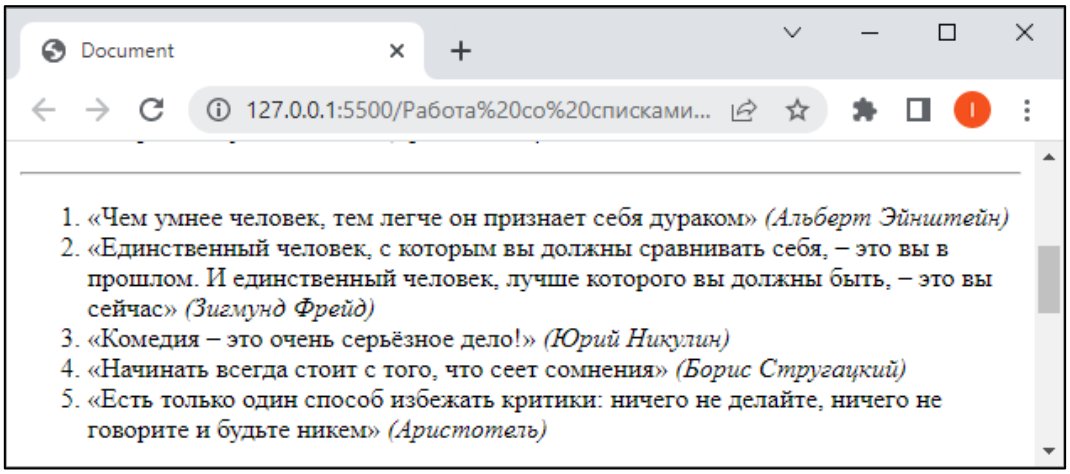


Рис. 2.118. Образец выполнения задания 2

Задание 3

1. Вновь отделите созданную разметку разделителем.
2. Скопируйте оба реализованных выше списка.
3. Поменяйте маркер маркированного списка на не закрашенный круг.
4. Для нумерованного списка сделайте нумерацию малыми латинскими буквами, а начало установите числом 6.

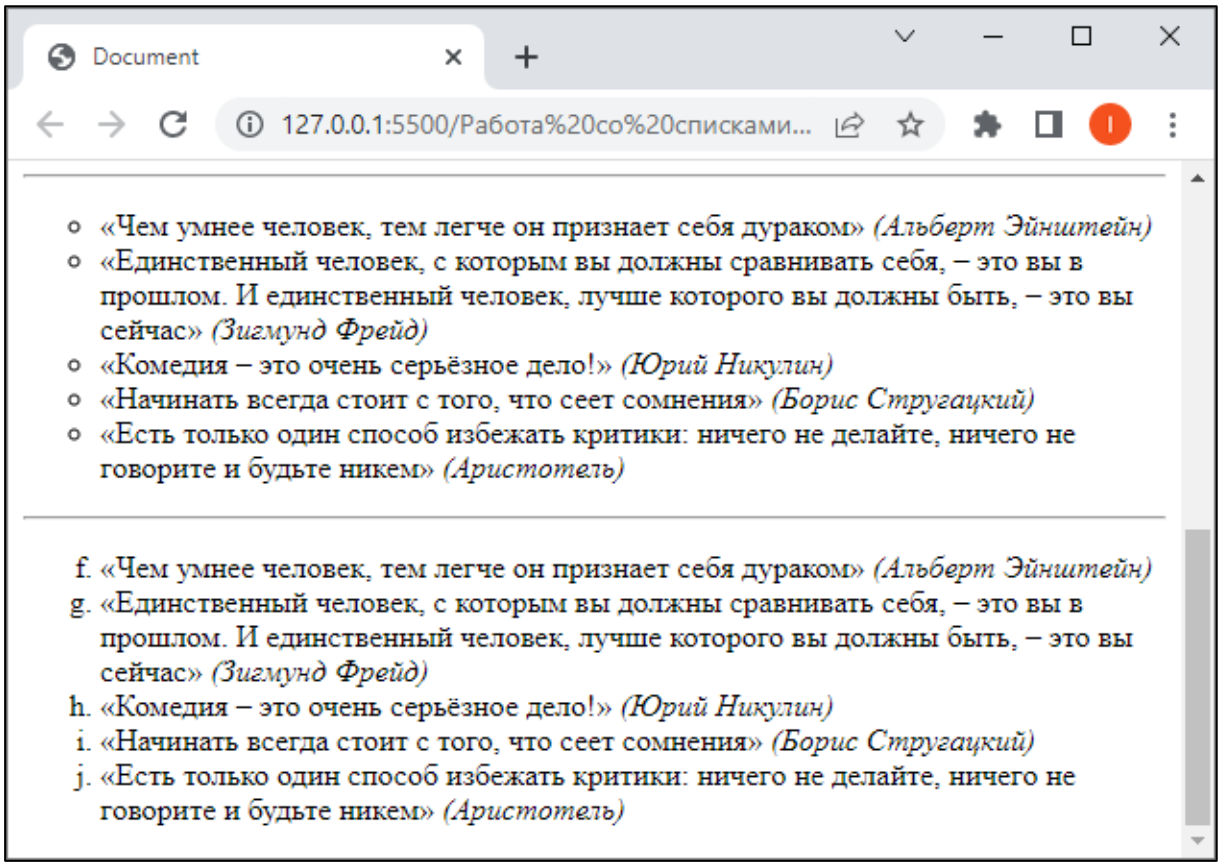


Рис. 2.119. Образец выполнения задания 3

Задание 4

1. Создайте каталог *Искусственный интеллект* и типовой HTML-файл со стандартной начальной разметкой.
2. Осуществите разметку многоуровневого списка, следуя образцу на рис. 2.120.

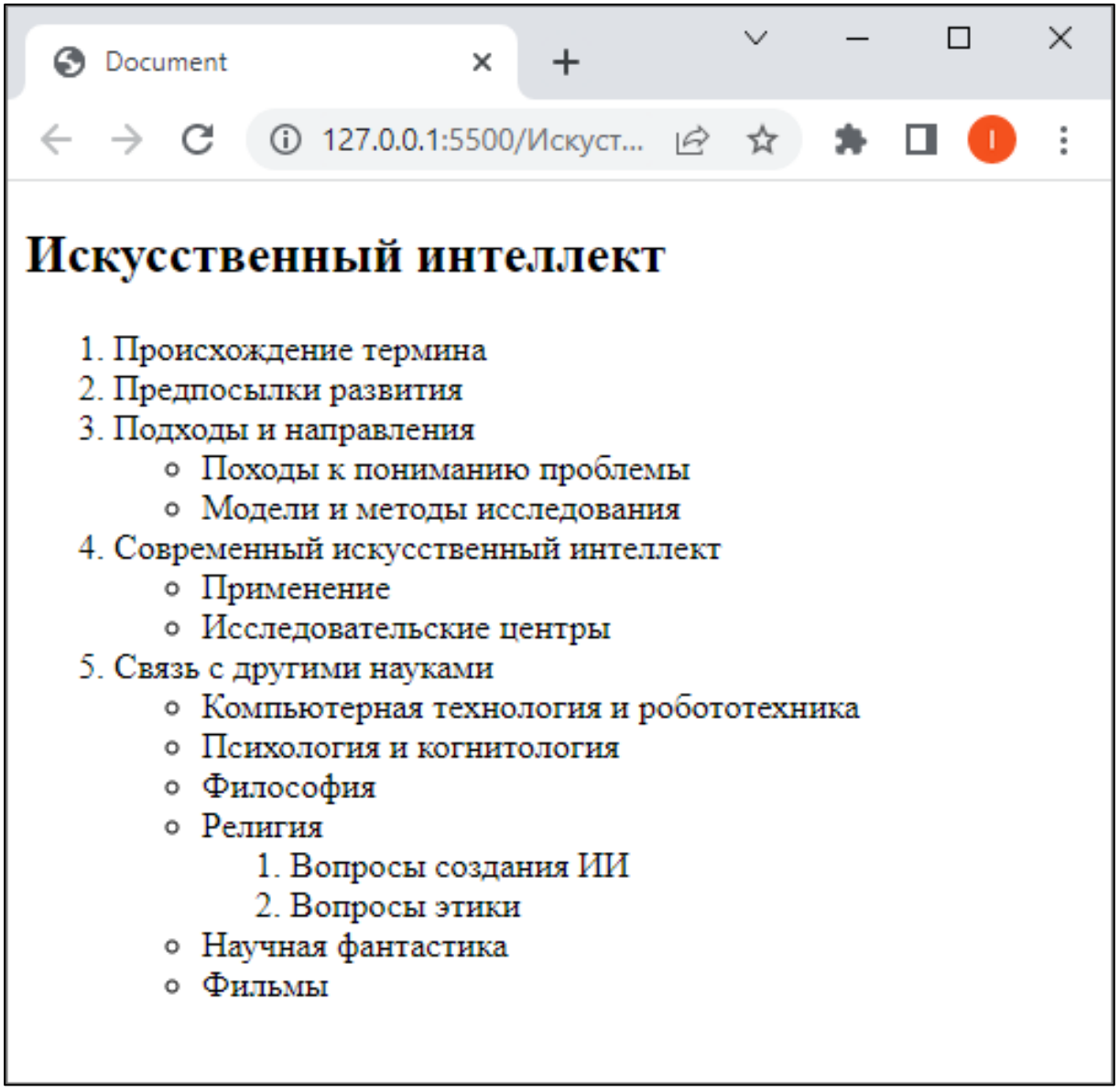


Рис. 2.120. Образец выполнения задания 4

2.7. Гиперссылки

2.7.1. Гиперссылки и их атрибуты

Синтаксис

Определение

Гиперссылка *– это указатель адреса в глобальной сети Интернет, по нажатию на который можно перейти из окна браузера на веб-ресурс.*

Гиперссылки являются одним из важнейших и базовых элементов, который позволяет реализовать интерактивное взаимодействие с веб-страницами. HTML поддерживает несколько тегов, которые способны создавать ссылочную связь: `<a>`, `<area>` и `<link>`.

Можно выделить две категории ссылок:

1. *ссылки на внешние ресурсы*, которые с помощью `<link>` подключают внешние файлы;
2. *гиперссылки*, позволяющие переходить на другие ресурсы сети Интернет, либо внутренние страницы веб-сайта.

Определение

```
<a></a>
```

Задает ***гиперссылку****.*
Допускается организация как внешних гиперссылок (на ресурсы сети Интернет, другие HTML-файлы в рамках сайта), так и локальных ссылок (по телу страницы).
Обязательный атрибут:

href *– задает URL-адрес, по которому следует перейти.*
Адрес может быть:

- *абсолютным (`href="http://vk.com"`);*
- *относительным (`href="Урок1.html"`).*

Приведем пример гиперссылки на сайт вуза:

```
<a href="https://www.vlsu.ru/">ВлГУ</a>
```

Структура адреса

Общий вид адреса

Гиперссылка состоит из двух частей:

1. *указатель* – это фрагмент текста, изображение или их комбинация, по нажатию на который осуществляется переход;
2. *адресная часть* – это URL-адрес ресурса или название файла, к которому необходимо перейти.

В общем случае адресная часть представляет собой URL-идентификатор следующего вида:

```
метод_доступа://имя_сервера:порт/путь
```

Метод доступа

Метод доступа определяет протокол обмена данными между узлами сети. Среди наиболее распространенных отметим file, http/https, ftp, mailto.

Имя сервера

Имя сервера указывает на полное название машины или ресурса в сети. Если оно не указано, то ссылка распознается в качестве локальной, т.е. относится к компьютеру, на котором она запущена.

Номер порта

Номер порта TCP представляет собой число, которое может запрашиваться при обращению к портам сервера. В случае отсутствия используется порт 80.

Абсолютный и относительный путь

Подходы к указанию пути

Если ссылка указывает только на имя файла, браузер ищет его в каталоге, где находится сама веб-страница. В реальности веб-сайты имеют разветвленную структуру из множества страниц и каталогов, аналогичную файловой системе. Поэтому возникает необходимость ссылаться на документы и в других каталогах.

Рассмотрим особенности ссылок на примере структуры веб-сайта, изображенного на рис. 2.121.

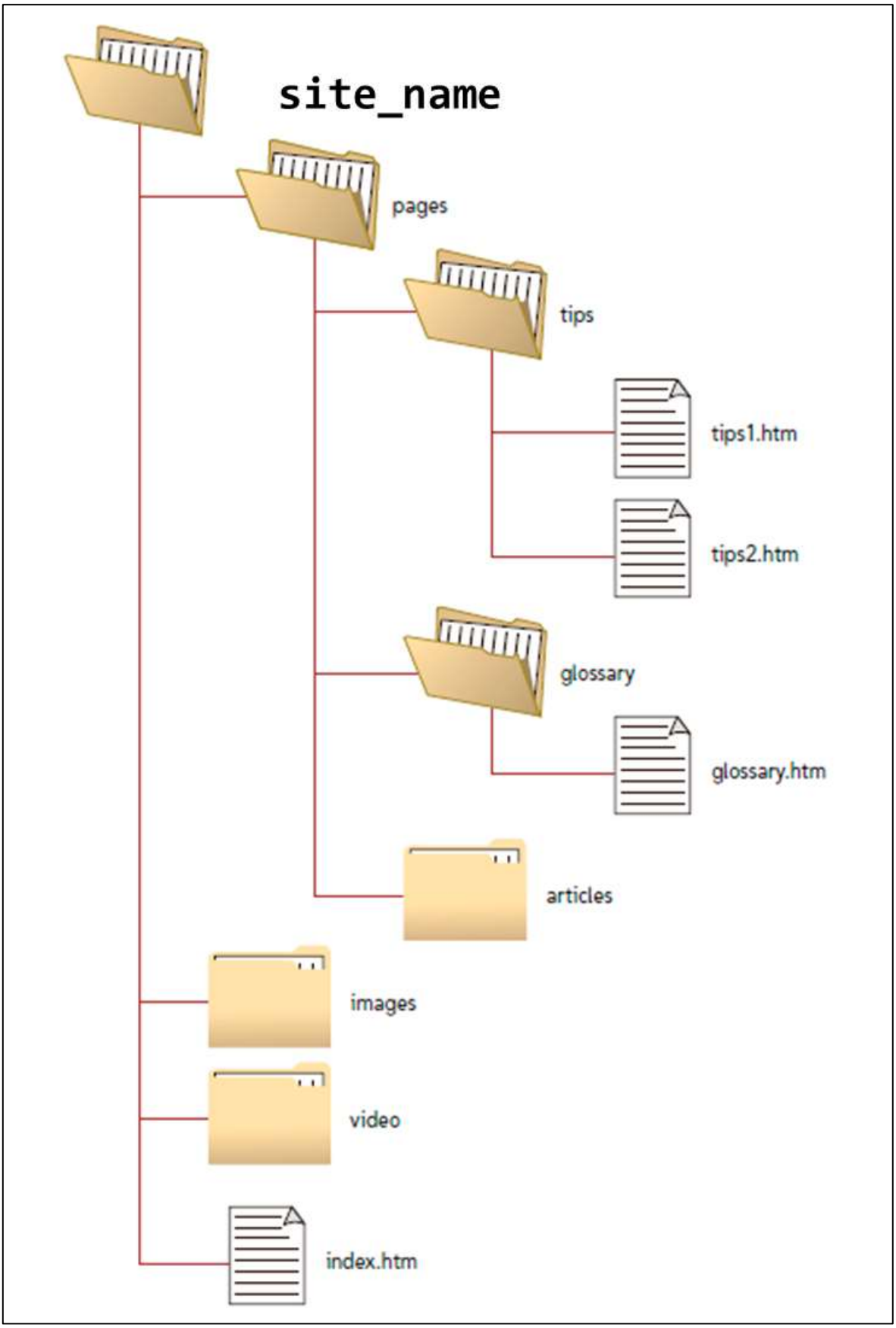


Рис. 2.121. Структура некоторого сайта

1. Абсолютный путь

Абсолютный путь ссылается на полный адрес файла в пределах всей структуры каталогов на ПК или сервере. Абсолютный путь позволяет обратиться к файлу из других ресурсов. Он включает в себя:

- протокол;
- домен (имя или IP-адрес);
- каталог (или последовательность вложения);
- название искомого файла.

Абсолютный путь может быть записан:

- с указанием протокола (полный путь):

```
http://site_name.ru/pages/glossary/tips1.html
```

- без указания протокола:

```
//site_name.ru/pages/glossary/tips1.html
```

При ссылке на файл из другого сайта абсолютный путь должен быть только полным.

Если файл расположен в корневом каталоге сайта, то путь к нему будет выглядеть следующим образом:

```
http://site_name.ru/index.html
```

Если имя файла не задано явно, то загружается веб-страница, установленная для загрузки по умолчанию в настройках веб-сервера (называемая *индексным файлом*):

```
http://site_name.ru/
```

Чаще всего индексному файлу задают название *index.html*.

- Слеш в конце ссылки означает, что обращение ведется к каталогу.
- В случае слеша отсутствия обращение ведется напрямую к файлу.

Это полезно знать!

Именно поэтому в примерах ранее и далее основной файл проекта мы называли и будем называть ***index.html****.*

2. Относительный путь

Относительный путь задает путь к документу относительно текущего. Он зависит от положения веб-страницы, на которой расположена ссылка.

Относительные гиперссылки необходимы для организации ссылок на другие документы сайта. Если браузер не находит в адресе ссылки протокол *http://*, то выполняется поиск документа на том же сервере.

Относительный путь состоит из следующих элементов:

- каталог (или последовательность вложения);
- название искомого файла.

Относительные ссылки имеют ряд специальных обозначений:

- `/` – корневая директория;
- `./` – указатель на текущий каталог;
- `../` – подъем на один каталог выше (переход в родительский каталог).

Таким образом, относительный путь, в отличие от абсолютного, не указывает название корневой директории и родительских каталогов. Поэтому адрес получается короче. Более того, при смене домена нет необходимости менять зафиксированные ранее абсолютные пути.

С другой стороны, сторонний ресурс может ссылаться, например, на изображения, который имеют относительные адреса. В этом случае они будут недоступны на другом сайте.

Атрибуты ссылок

href

Атрибут **`href`** задает URL-адрес документа, на который указывает ссылка. Является обязательным.

download

Атрибут **`download`** дополняет `href` и сообщает браузеру, что указанный ресурс должен быть загружен на ПК пользователя (вместо перехода или просмотра).

Например, загрузка изображения:

```
<a href="/pictures/back.jpg" download>
    <img src="/pictures/back.jpg" alt="фон">
</a>
```

Следующая команда скачает файл:

```
<a href="/docs/doc_4561.pdf" download="Отчет за
12.03.23.pdf">Скачать отчет от 12.03.23</a>
```

rel

Атрибут **rel** также дополняет href, указывая информацию об отношении между искомой веб-страницей и связанным по ссылке документом. Может принимать одно из нескольких фиксированных значений. Более подробная информация о rel.

Например, уточним в ссылке, что она относится к информации об авторе:

```
<a href="/about/author.html" rel="author">Об авторе</a>
```

target

Атрибут **target** конкретизирует, в каком окне браузера необходимо открывать документ. Возможные значения:

- **_self** – в текущем окне;
- **_blank** – в новом окне браузера;
- **_parent** – во фрейм-родителе;
- **_top** – в полное окно браузера.

Например, следующая ссылка открывает карты Google в отдельном окне браузера:

```
<a href="https://google.ru/maps" target="_blank">
    Открыть карту
</a>
```

type

Атрибут **type** уточняет *MIME-тип* файла ссылки (т.е. его формат). Он необходим, прежде всего, для справочной информации (например, для программной обработки).

Например, уточним, что скачиваемый по ссылке файл имеет формат PDF:

```
<a href="/docs/doc_4561.pdf" type="pdf" download>
    Скачать отчет от 12.03.23
</a>
```

2.7.2. Вариации гиперссылок

Ссылки на внешние ресурсы

Текстовые гиперссылки

Рассмотрим некоторые примеры ссылок на внешние ресурсы.
Гиперссылка на внешний ресурс:

```
<a href="http://www.google.com">Google</a>
```

Гиперссылка на локальную страницу сайта:

```
<a href="site_map.html">Карта сайта</a>
```

Открытие страницы на текущей вкладке:

```
<a href="http://www.google.com" target="_self">Google</a>
```

Открытие страницы в новой вкладке:

```
<a href="http://www.google.com"
target="_blank">Google</a>
```

Скачивание файла по указанной ссылке:

```
<a href="http://www.mylessons.ru/lesson.pdf"
download>скачать</a>
```

Гиперссылки на нетекстовые элементы

Часто гиперссылками делают изображения:

```
<a href="/pages/site-map.html">
    <img src="/images/site-map.jpg" alt="карта сайта">
</a>
```

В целом ссылка может содержать и комбинации элементов. Однако стоит учитывать, что в этом случае рационально настроить свойства подобных гиперссылок через стили CSS, в частности убрать подчеркивание.

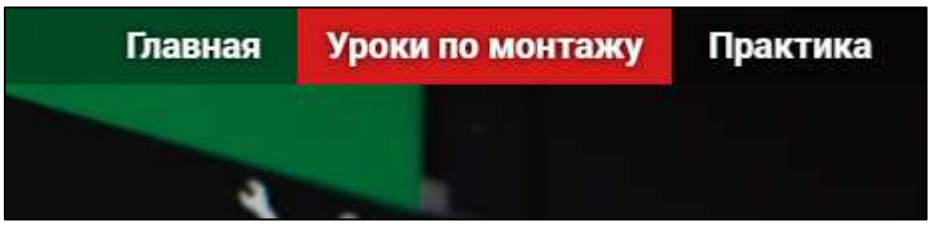


Рис. 2.122. Ссылки в меню сайта настроены с помощью CSS и больше похожи на анимированные кнопки

Гиперссылки на разные элементы

В HTML5 возможности гиперссылок расширены: теперь они позволяют одним кликом совершать звонки, отправлять сообщения или звонить по Skype.

Приведем некоторые примеры.

Ссылка на телефонный номер:

```
<a href="tel:+71231234567">+7 (123) 123-45-67</a>
```

Ссылка на e-mail (откроет почтовый сервис клиента):

```
<a href="mailto:username@mail.ru">username@mail.ru</a>
```

Ссылка на звонок по Skype:

```
<a href="skype:пользователь?call">Skype</a>
```

Ссылка на чат в Skype:

```
<a href="skype:пользователь?chat">Skype</a>
```

Ссылка на добавление контакта в Skype:

```
<a href="skype:пользователь?add">Skype</a>
```

Ссылка на отправку файла в Skype:

```
<a href="skype:пользователь?sendfile">Skype</a>
```

Приемы работы с Visual Studio Code

При вставке ссылок в редакторе обратите внимание, что в меню подсказки доступны не только теги, но и некоторые сниппеты популярных вариаций тегов с атрибутами.

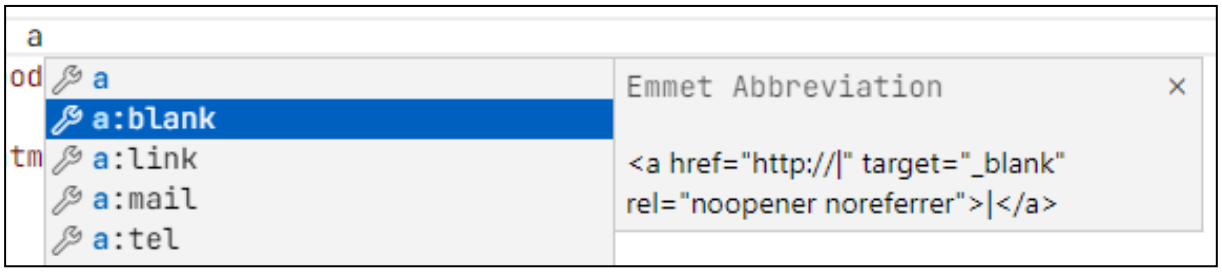


Рис. 2.123. Варианты вставки гиперссылок с разными атрибутами

2.7.3. Внутренние гиперссылки

Понятие якоря

Внутренние ссылки или *якоря* – специальный вид гиперссылок, которые осуществляют переходы создают в рамках текущей веб-страницы. Это может потребоваться, когда на веб-странице много материала и используется полоса прокрутки.

Для внутренней гиперссылки атрибут `href` содержит название указателя (якорь), а не URL-адрес. При этом перед названием якоря обязательно ставится **#**.

С другой стороны элемент, на который осуществляется переход, должен иметь глобальный атрибут **id** с таким же названием, но только уже без символа #.

Переходы внутри страницы

Следующий пример демонстрирует организацию внутренних переходов по ссылкам в меню (рис. 2.124). Полужирным начертанием выделены якоря ссылок:

Глава 2\Гиперссылки\index.html

```
<h1>Технологии frontend-разработки</h1>
<h2>Содержание</h2>
<ul>
    <li><a href="#html">Язык разметки HTML</a></li>
    <li><a href="#css">Каскадные таблицы стилей
        CSS</a></li>
    <li><a href="#js">Язык программирования
        JavaScrpt</a></li>
</ul>

<h2 id="html">1. Язык разметки HTML</h2>
<!-- Здесь много текста и изображений -->

<h2 id="css">2. Каскадные таблицы стилей CSS</h2>
<!-- Здесь много текста и изображений -->

<h2 id="js">3. Язык программирования JavaScrpt</h2>
<!-- Здесь много текста и изображений -->
```

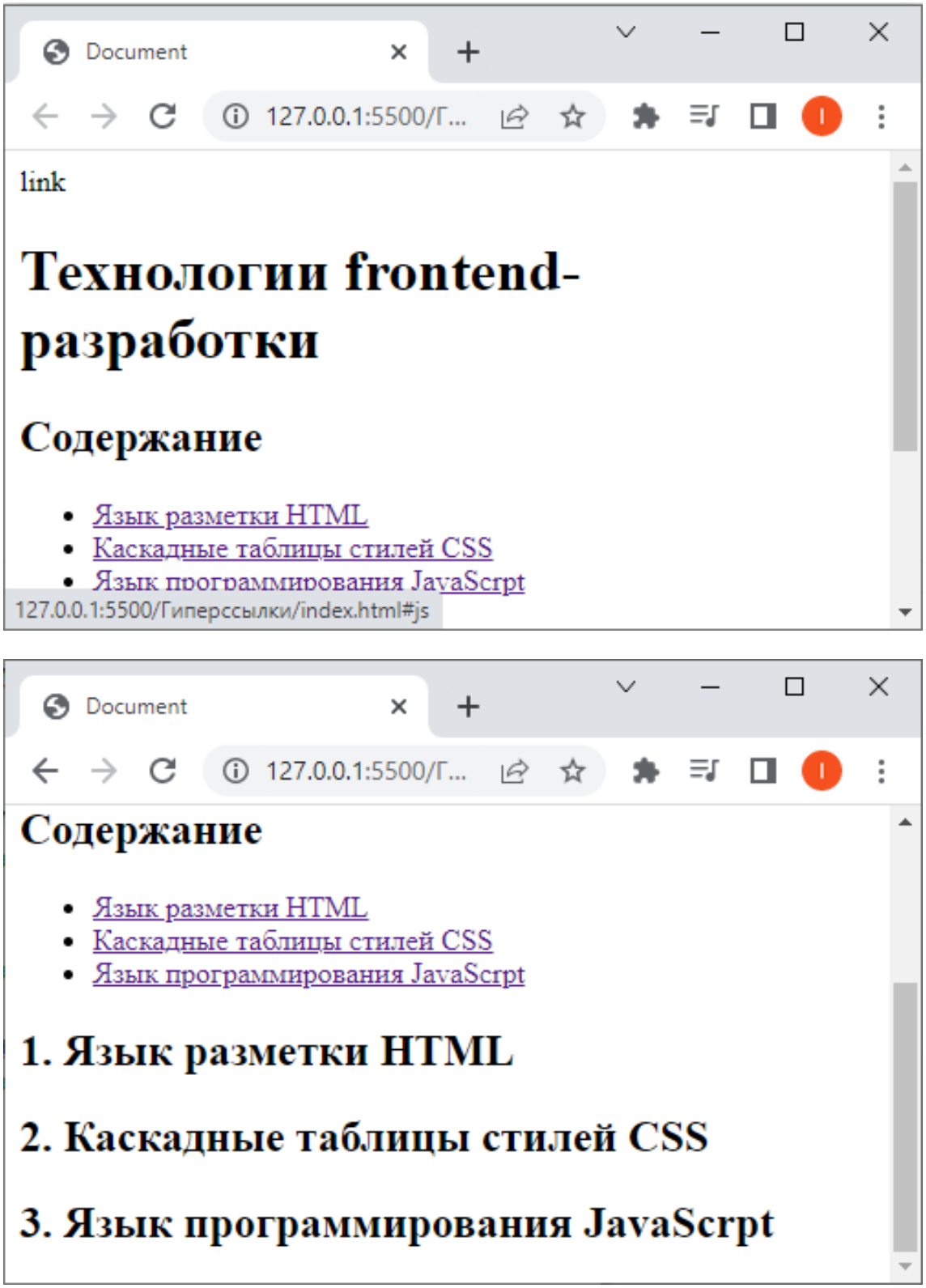


Рис. 2.124. При клике на третью ссылку меню осуществляется мгновенный переход к соответствующему заголовку

Вопросы для самопроверки

1. Каким тегом описываются гиперссылки и какие атрибуты для нее являются обязательными?
2. Опишите структурные компоненты адреса в гиперссылке.
3. В чем разница между абсолютной и относительной адресацией пути? Приведите достоинства и недостатки каждой.
4. Перечислите атрибуты тега гиперссылки и опишите их назначение.
5. Каким образом сделать гиперссылкой нетекстовые элементы и что стоит учитывать?
6. Опишите механизм создания внутренних гиперссылок.

Практикум

Задание 1

1. Скопируйте каталог *ООП 3* ранее выполненного задания 3 из параграфа 2.5. Переименуйте его в *Внешние гиперссылки*.
2. Сделайте термины ООП, инкапсуляции, наследования и полиморфизма ссылками на внешние веб-ресурсы, где дается их подробное описание (рис. 2.125).

Задание 2

1. Создайте каталог *Внутренние гиперссылки* и индексный файл *index.html* со стандартной начальной разметкой.
2. Изучите предложенную ниже частичную разметку документа. Дополните ее тегами и получите результат, как изображено на рис. 2.126-рис. 2.127. Вам потребуются теги структурного форматирования текста.
3. Сделайте внутренние гиперссылки из пунктов содержания на соответствующие подзаголовки.
4. В конце каждого пункта установите гиперссылки на соответствующие страницы в Wikipedia.

Глава 2\Внутренние гиперссылки\index.html

```
<!DOCTYPE html>
<html lang="ru">

<head>
    <meta charset="UTF-8">
    <meta http-equiv="X-UA-Compatible" content="IE=edge">
    <meta name="viewport" content="width=device-width,
initial-scale=1.0">
    <title>Document</title>
</head>


<body>
    <h1>Веб разработка</h1>

    <!--  Меню  -->
    <ul>
        <li><a href="#html">Технология HTML</a></li>
        <li><a href="#css">CSS</a></li>
        <li><a href="#js">Язык JavaScript</a></li>
    </ul>

    <h2>Введение</h2>
    <p>В основе современного сайта можно выделить
    следующие три технологии: HTML, CSS и
    JavaScript.</p>

    <!--  Про HTML  -->
    <h2>Технология HTML</h2>
    HTML - («язык гипертекстовой разметки») —
    стандартный язык разметки документов во Всемирной
    паутине.

    Большинство веб-страниц содержат описание разметки
    на языке HTML. Он интерпретируется браузерами и
    отображается в виде документа в удобной для
    восприятия форме.

    Подробнее про HTML.
```

```
    <!--  Про CSS  -->
    <h2>Технология CSS</h2>
    Стилем или CSS(«каскадные таблицы стилей»)
    называется набор параметров форматирования,
    который применяется к элементам документа
    для изменения их внешнего вида.

    Технология CSS создавалась с целью отделить
    логическую структуру веб-страницы (написанную,
    в частности, на HTML или другом языке разметки)
    от ее визуального оформления. Это существенно
    упростило процесс верстки веб-страниц, а также
    сделало его гибким в плане возможностей изменения
    стилевых настроек.

    Подробнее про CSS.

    <!--  Про JS  -->
    <h2>Язык JavaScript</h2>
    JavaScript – мультипарадигменный язык
    программирования. Он поддерживает
    объектно-ориентированный, императивный и
    функциональный подходы к программированию, что
    дает гибкий инструмент веб-разработчики.

    JavaScript преимущественно используется
    браузерами как как язык сценариев для придания
    интерактивности веб-страницам.

    Подробнее про JavaScript.
</body>

</html>
```

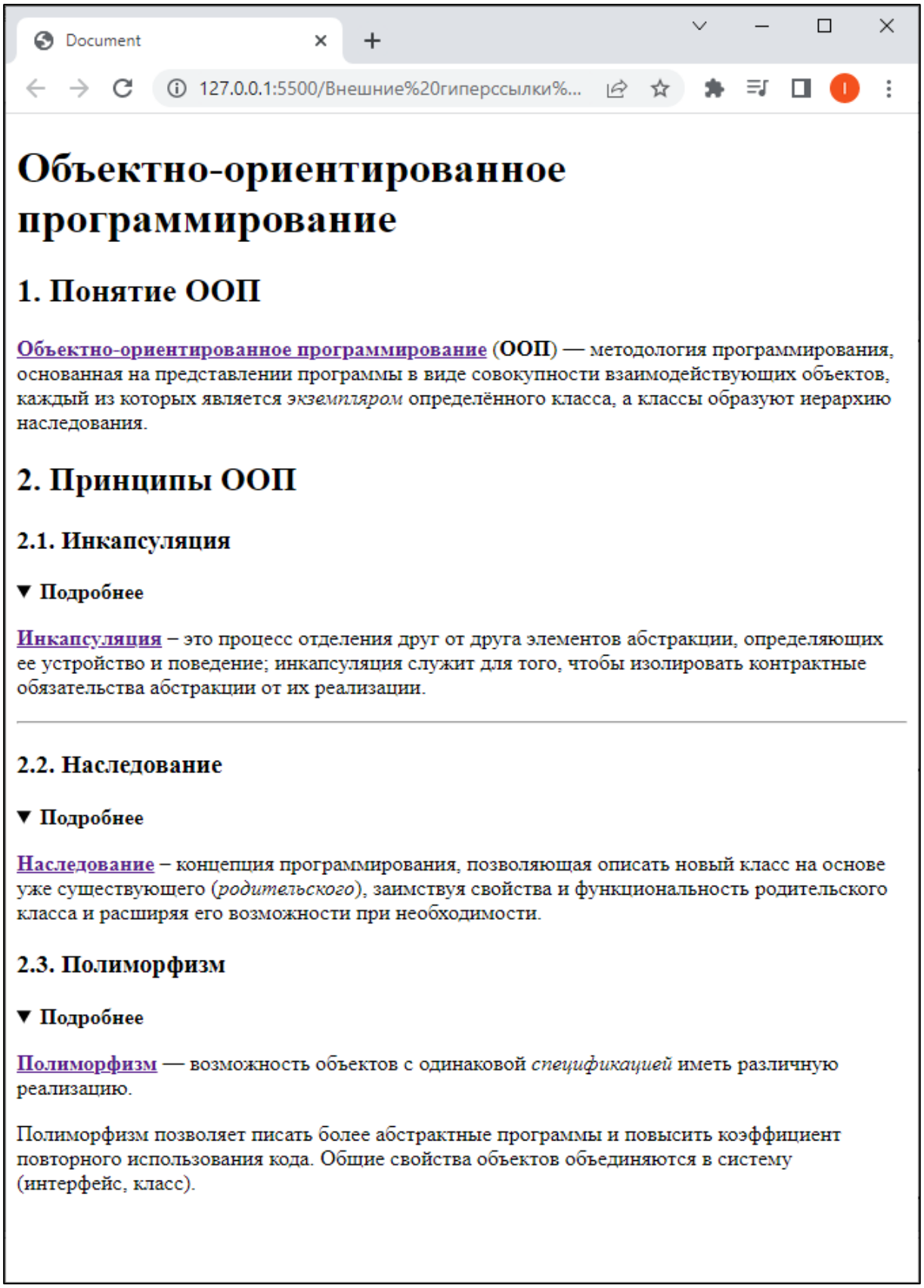


Рис. 2.125. Образец выполнения задания 1

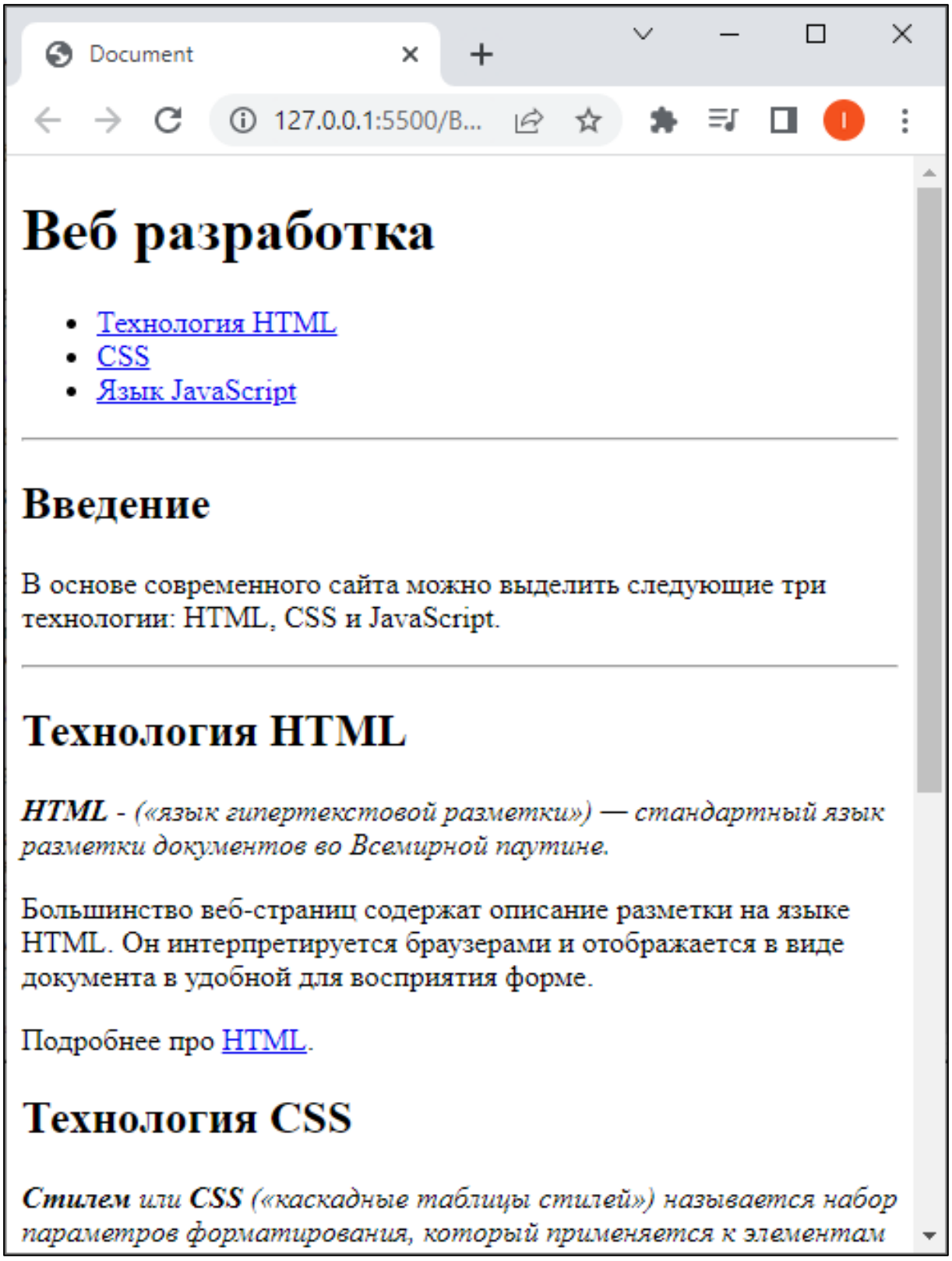


Рис. 2.126. Образец выполнения задания 2 (часть 1)

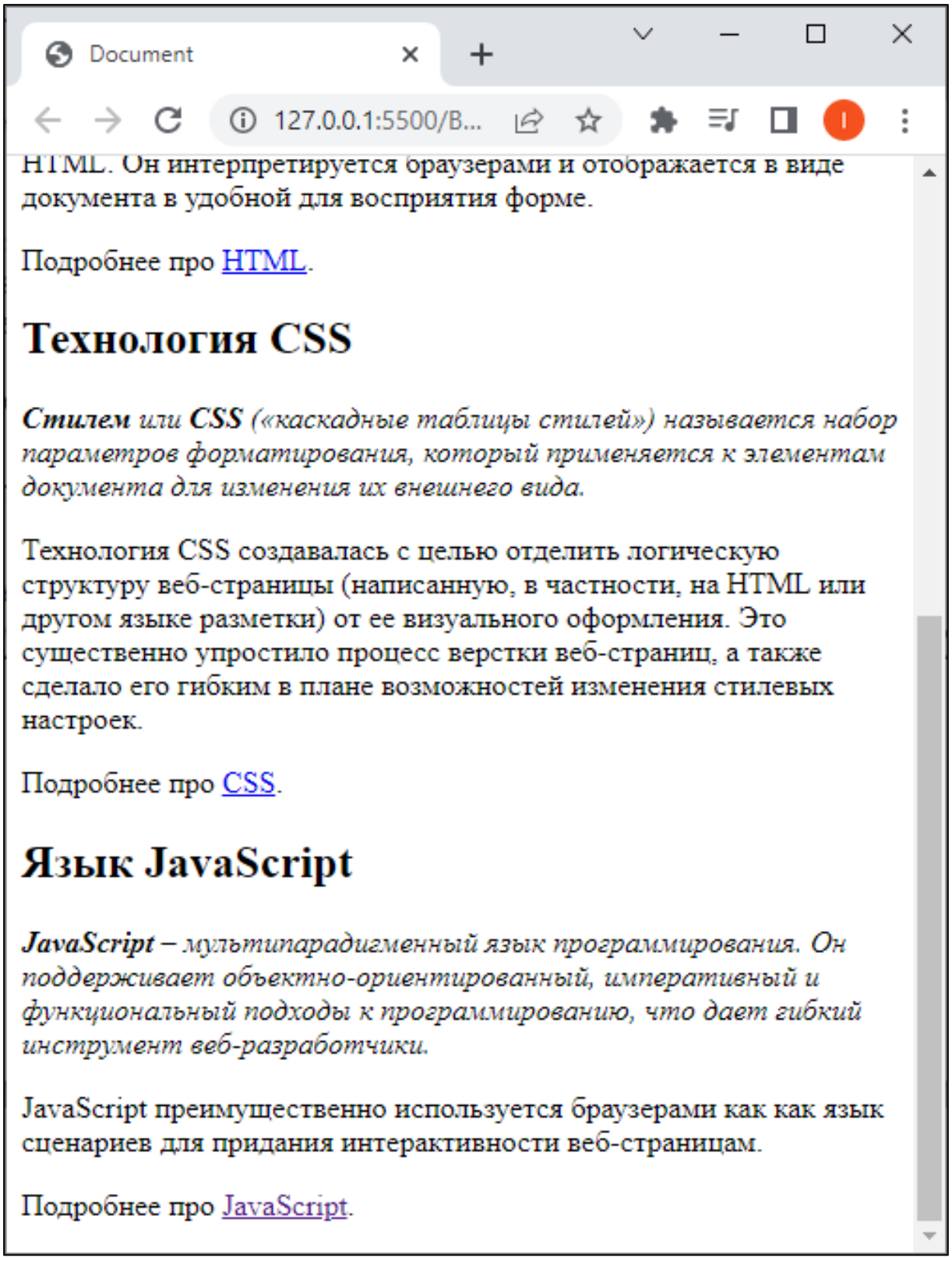


Рис. 2.127. Образец выполнения задания 2 (часть 2)

2.8. Разметка таблиц

2.8.1. Минимально валидная структура таблиц

Особенности разметки таблиц в HTML

Таблицы используются для разметки структурированных по одному или нескольким критериям данных. Таблицы в HTML включают в свою структуру *строки* и *столбцы*, которые образованы подряд идущими *ячейками* внутри строк.

Ячейки HTML-таблиц могут не только текстовые данные, но и другие элементы разметки: заголовки, списки, изображения, компоненты форм, а также вложенные таблицы. Дополнительно HTML предоставляет структурные теги для обозначения логической связи между группами строк и колонок.

Минимальный набор тегов для разметки таблиц

Основные теги

Определение

```
<table></table>
```

*Тег размечает **таблицу** и является контейнером для вложенных в нее элементов структуры.*

```
<tr></tr>
```

*Тег размечает **строку таблицы (ряд)** и является контейнером для вложенных в нее ячеек. Расположен внутри* `<table></table>`.

```
<td></td>
<th></th>
```

*Первый тег размечает **ячейку тела таблицы** и является контейнером для содержимого.*
*Второй тег размечает **ячейку заголовка таблицы** и является контейнером для содержимого в заголовочной строке.*
Оба контейнера расположены в строке `<tr></tr>`.

Простая таблица

Для описания таблицы, минимально проходящей валидацию, используется всего три парных тега:

- `<table></table>` – указывает, что задана таблица;
- `<tr></tr>` – задает строку таблицы;
- `<td></td>` – определяет ячейку в строке таблице.

Важное замечание!

Контейнеры для разметки таблиц важно вкладывать в строго перечисленном порядке!

Осуществим разметку простой таблицы из двух строк и трех колонок.

Глава 2\Таблицы\Простая таблица 1\index.html

```
<table>
    <tr>
        <td>Ячейка 1.1</td>
        <td>Ячейка 1.2</td>
        <td>Ячейка 1.3</td>
    </tr>
    <tr>
        <td>Ячейка 2.1</td>
        <td>Ячейка 2.2</td>
        <td>Ячейка 2.3</td>
    </tr>
</table>
```

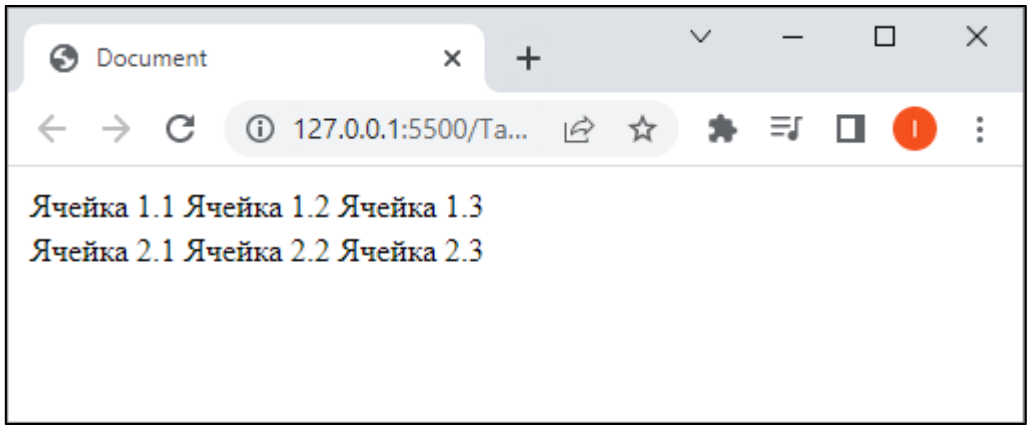


Рис. 2.128. Простая таблица без рамки

Таблица с рамками

По умолчанию браузеры не прорисовывают границы таблицы (их толщина равна нулю). Обычно они настраиваются CSS-стилями.

В более ранних версиях HTML для рисования рамки использовался атрибут **border**. Кроме того, таблицы, строки и ячейки содержали ряд атрибутов, управлявших оформлением таблиц.

В следующем примере прорисовывается граница толщиной 1 px, шириной 460 px, полями внутри ячеек 10 px и с выравниванием таблицы по центру.

Глава 2\Таблицы\Простая таблица 2\index.html

```
<table border="1" width="460" align="center"
cellpadding="10">
    <tr>
        <td>Ячейка 1.1</td>
        <td>Ячейка 1.2</td>
        <td>Ячейка 1.3</td>
    </tr>
    <tr>
        <td>Ячейка 2.1</td>
        <td>Ячейка 2.2</td>
        <td>Ячейка 2.3</td>
    </tr>
</table>
```

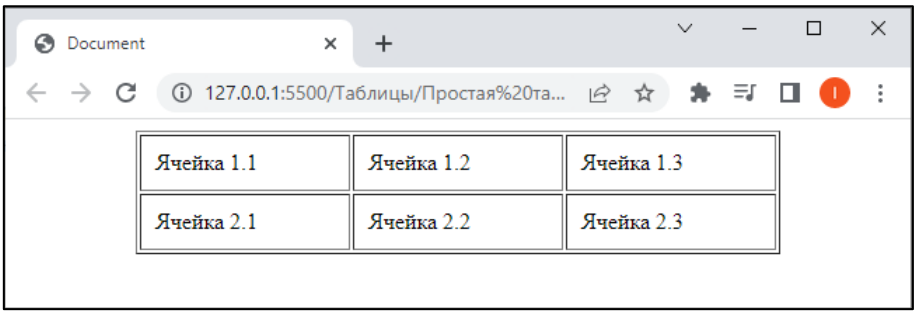


Рис. 2.129. Простая таблица с рамкой

Это полезно знать!

С приходом HTML5 от использования атрибутов таблиц практически отказались, поскольку с этим прекрасно справляется CSS. Текущий пример приводится исключительно для ознакомления.

2.8.2. Полная структура таблиц

Теги подразделов таблицы

Определение

Наиболее полная структура таблицы предполагает ее разбиение на следующие ниже подразделы.

```
<table></table>
```

Тег определяет ***область описания таблицы****.*

```
<caption></caption>
```

Содержит ***текстовую подпись*** *к таблице (заголовок или название), вставляется сразу после* **`<table>`**.

```
<thead></thead>
```

Определяет область ***строк заголовка таблицы****.*

```
<tbody></tbody>
```

Размечает ***основное содержимое строк таблицы*** *(может повторяться многократно).*

```
<tfoot></tfoot>
```

Предназначен для ***последних строк таблицы*** *(«подбивки», подытоживания).*

Дополнительно можно использовать теги **`<colgroup>`** *и* **`<col>`**, *которые* ***группируют*** *строки и столбцы* ***по смыслу****.*

```
<colgroup></colgroup>
```

Объединяет ***столбцы в группу****, выделяя однородные ячейки.*

```
<col>
```

Формирует ***группы столбцов*** *внутри* **`<colgroup></colgroup>`**. *Число объединяемых колонок в группе указывается в атрибуте* **`span`**.

Таким образом, полноценная структура таблицы имеет следующий каркас:

```
<table>
    <caption>Подпись к таблице</caption>
    <thead>
        Строки заголовочной части
    </thead>
    <tbody>
        Строки основной части
    </tbody>
    <tfoot>
        Строки подытоживания
    </tfoot>
</table>
```

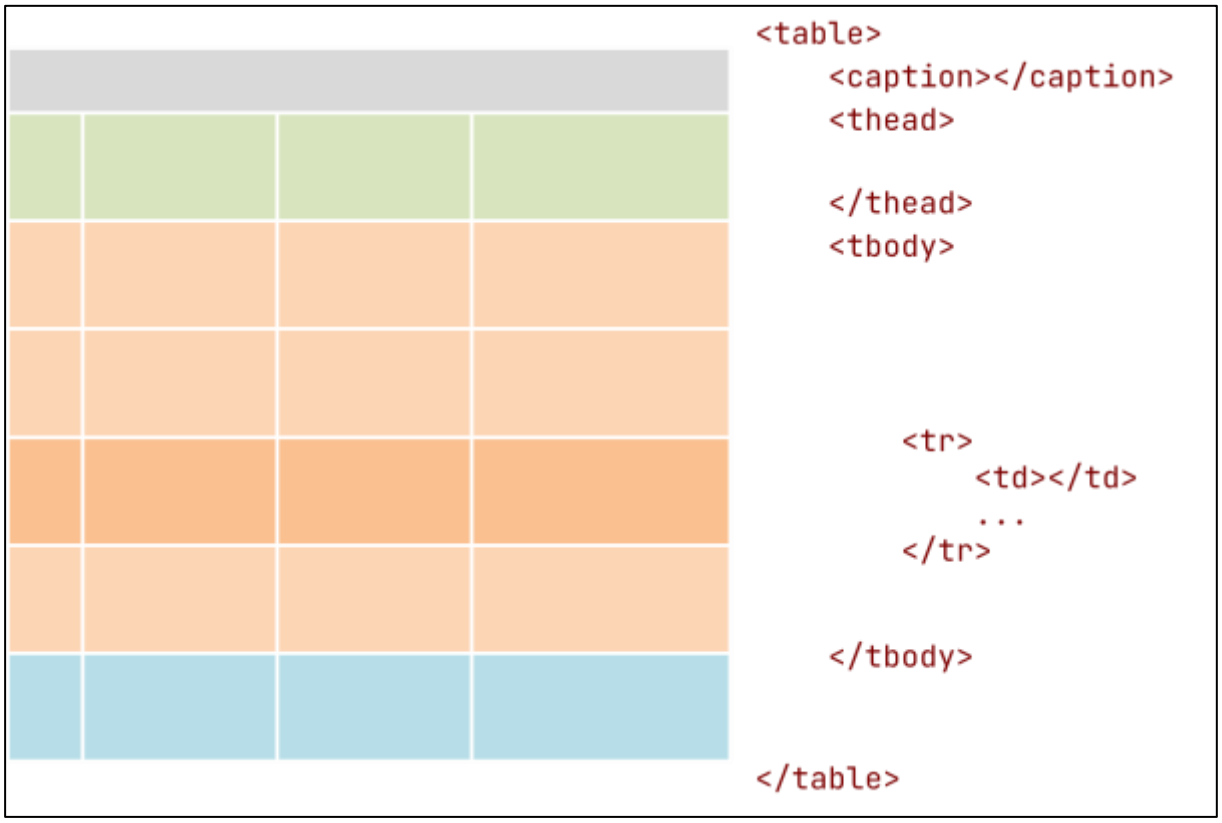


Рис. 2.130. Основные структурные разделы таблицы

Пример разметки полноценной таблицы

Приведем практический пример разметки таблицы.

Для лучшей визуализации подразделов таблицы задействуем CSS (детальному изучению это технологии посвящена следующая глава). Стили подключаются в отдельном файле с помощью тега `<link>`.

Если ширина таблицы (или колонок) не фиксируется, то ширина колонки подбирается согласно максимальной ширине содержимого в этой колонке.

В противном случае ширина колонок подбирается пропорционально ширине содержимого колонок.

Глава 2\Таблицы\Полноценная таблица\index.html

```
<!DOCTYPE html>
<html lang="en">

<head>
    <meta charset="UTF-8">
    <meta http-equiv="X-UA-Compatible" content="IE=edge">
    <meta name="viewport" content="width=device-width,
initial-scale=1.0">
    <title>Document</title>

    <!-- Здесь используются CSS-стили -->
    <link rel="stylesheet" href="style.css">
</head>

<body>
    <table class="demo-table">
        <caption>
            Процент студентов, сдавших экзамены
            без задолженностей.
        </caption>
        <thead>
            <tr>
                <th>Группа</th>
                <th>1 семестр (%)</th>
                <th>2 семестр (%)</th>
            </tr>
        </thead>
        <tbody>
            <tr>
                <td>МИ-119</td>
                <td>78</td>
                <td>74</td>
            </tr>
            <tr>
                <td>МИ-299</td>
                <td>72</td>
                <td>69</td>
```

```
            </tr>
            <tr>
                <td>ФМ-199</td>
                <td>82</td>
                <td>76</td>
            </tr>
        </tbody>
        <tfoot>
            <tr>
                <td>Среднее:</td>
                <td>77</td>
                <td>73</td>
            </tr>
        </tfoot>
    </table>
</body>

</html>
```

Глава 2\Таблицы\Полноценная таблица\style.css

```
.demo-table {
    margin: 0 auto;
}

table {
    border-collapse: collapse;
    font: 16px Tahoma, Arial;
    text-align: center;
}

th, td {
    border: 3px solid white;
    padding: 6px 10px;
}

thead th {
    background: #513a6e;
    color: #fff;
}

tbody td {
    background: #fff9df;
}

tfoot td {
```

```
        background: #e5ffd6;
    }

    caption {
        font: italic 14px Tahoma;
        color: dimgrey;
        margin: 10px 6px;
    }
```

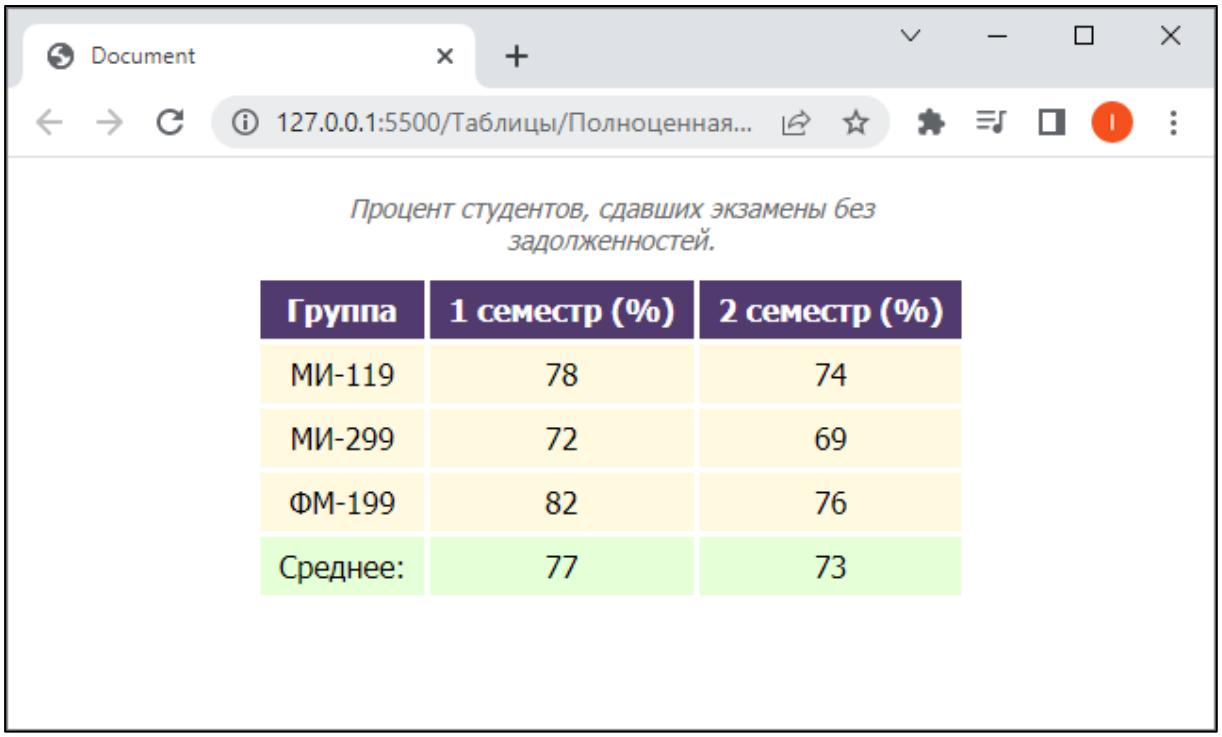


Рис. 2.131. Разметка полноценной таблицы

Приемы работы с Visual Studio Code

В Visual Studio Code быстро сгенерировать простую таблицу можно с помощью Emmet-аббревиатуры

```
table>tr*3>td*2
```

*и нажать **Tab** (т.е. таблица в 3 строки и 2 колонки). Здесь используется комбинирование вложения и дублирования элементов (строк и ячеек).*

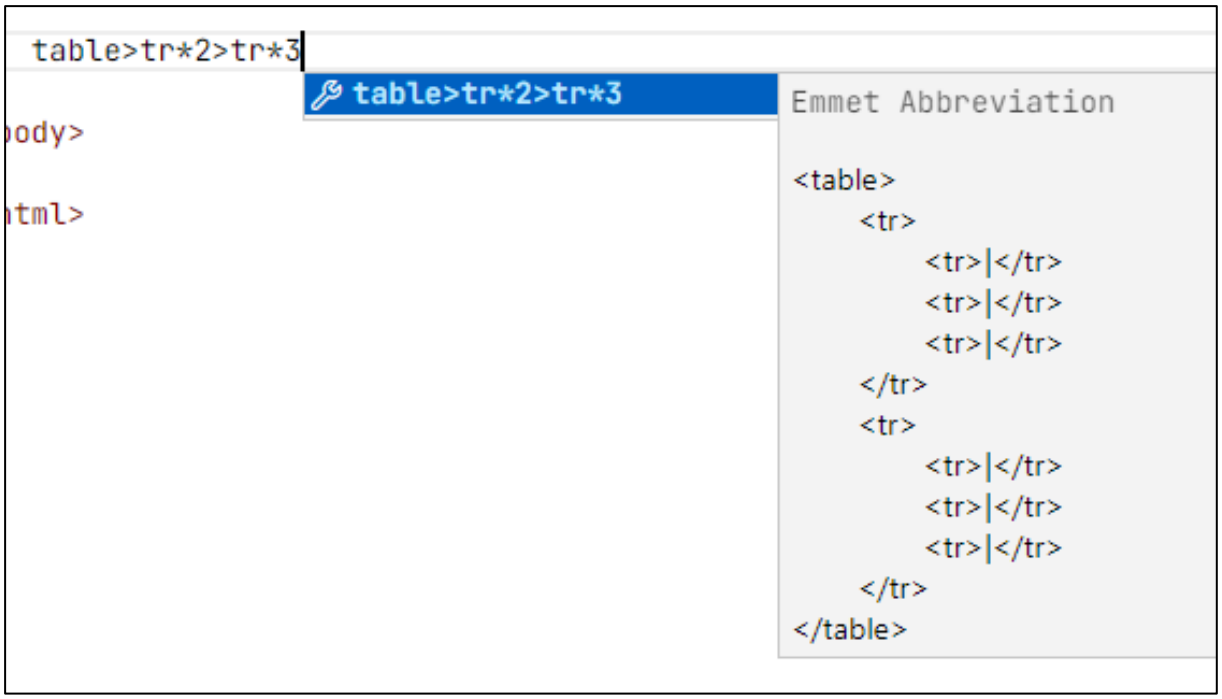


Рис. 2.132. Генерация простой таблицы

Приемы работы с Visual Studio Code

*Для генерации соседних элементов используйте Emmet-абривеатуру **+**. Например:*

```
table>caption+thead+tbody+tfoot
```

разметит основные разделы структуры таблицы.

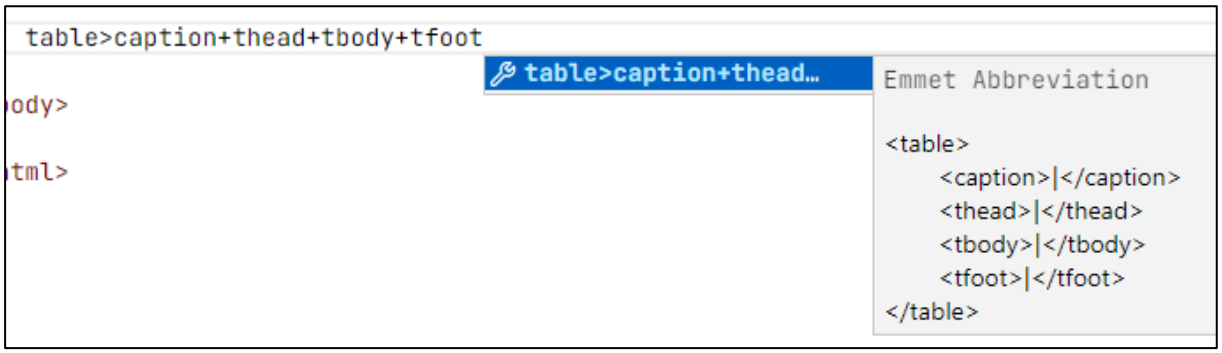


Рис. 2.133. Генерация основных разделов таблицы

2.8.3. Объединение ячеек в таблицах

Принцип объединения

Атрибуты тега ячейки `<td></td>` (или заголовочной `<th></th>`) `colspan` и `rowspan` используются при *объединении ячеек*.

- **`colspan`** показывает, на сколько ячеек по горизонтали следует расширить `<td></td>`;
- **`rowspan`** объединяет ячейки заданного столбца по строкам.

Все ячейки, входящие в объединение (кроме первой), должны быть удалены, чтобы не поломать структуру таблицы.

Практический пример

В следующем примере пошагово показано, как необходимо объединять ячейки, чтобы не допустить «поломки» структуры таблицы (см. рис. 2.134-рис. 2.136).

Глава 2\Таблицы\Объединение ячеек\index.html

```
<!-- Шаг 1. Исходная таблица -->
<table border="1" width="300" align="center">
    <tr>
        <td>1.1</td>
        <td>1.2</td>
        <td>1.3</td>
    </tr>
    <tr>
        <td>2.1</td>
        <td>2.2</td>
        <td>2.3</td>
    </tr>
    <tr>
        <td>3.1</td>
        <td>3.2</td>
        <td>3.3</td>
    </tr>
</table>

<hr>
```

```
<!-- Шаг 2. Объединение по строке -->
<table border="1" width="300" align="center">
    <tr>
        <td colspan="2">1.1</td>
        <!-- 1.2 уделена, т.к. вошла в объединение -->
        <td>1.3</td>
    </tr>
    <tr>
        <td>2.1</td>
        <td>2.2</td>
        <td>2.3</td>
    </tr>
    <tr>
        <td>3.1</td>
        <td>3.2</td>
        <td>3.3</td>
    </tr>
</table>

<hr>

<!-- Шаг 3. Объединение по колонке -->
<table border="1" width="300" align="center">
    <tr>
        <td colspan="2">1.1</td>
        <td rowspan="3">1.3</td>
    </tr>
    <tr>
        <td>2.1</td>
        <td>2.2</td>
        <!-- 2.3 уделена, т.к. вошла в объединение -->
    </tr>
    <tr>
        <td>3.1</td>
        <td>3.2</td>
        <!-- 3.3 уделена, т.к. вошла в объединение -->
    </tr>
</table>
```

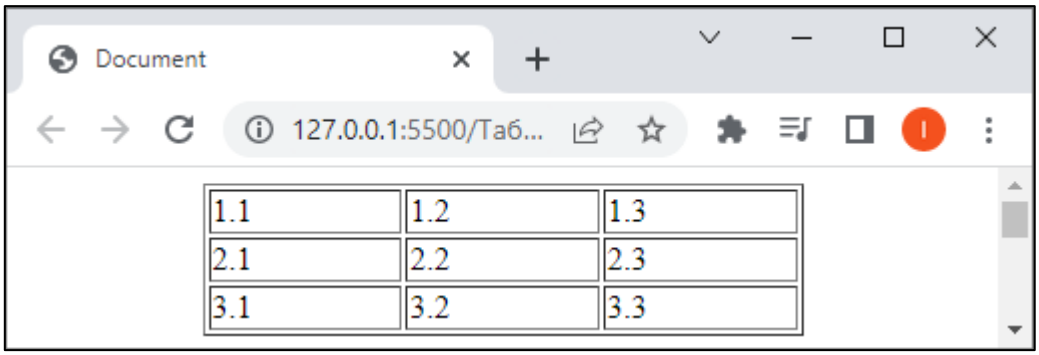


Рис. 2.134. Объединение таблицы: исходная таблица

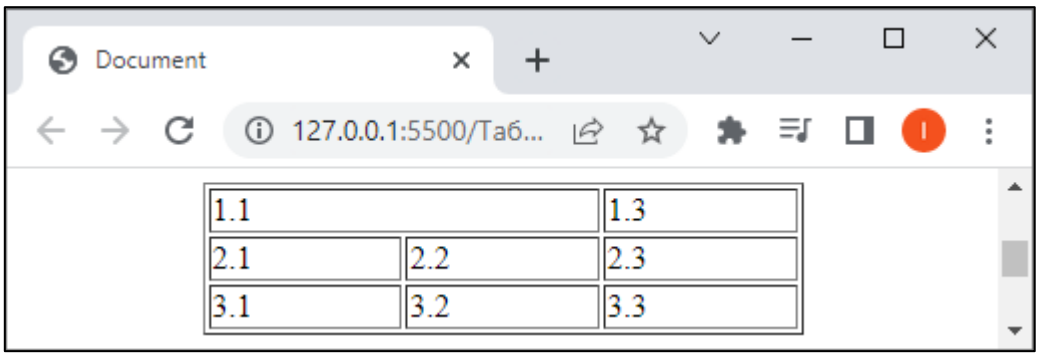


Рис. 2.135. Объединение таблицы: объединение по строке

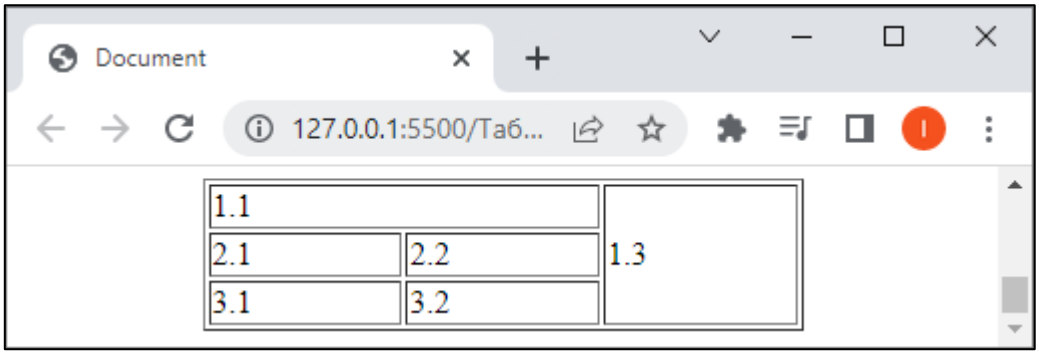


Рис. 2.136. Объединение таблицы: объединение по колонке

Использование конструктора таблиц

На самом деле размечать таблицы вручную далеко не всегда рационально. Зачастую для построения таблиц удобнее использовать разные визуальные конструкторы, которые способны сгенерировать код HTML-разметки таблицы и даже задать CSS-настройку.

Приведем некоторые примеры таких веб-сервисов:

- Генератор таблиц для сайта: iksweb.ru.
- Tables Generator: tablesgenerator.com.
- HTML Table Generator: codebeautify.org.

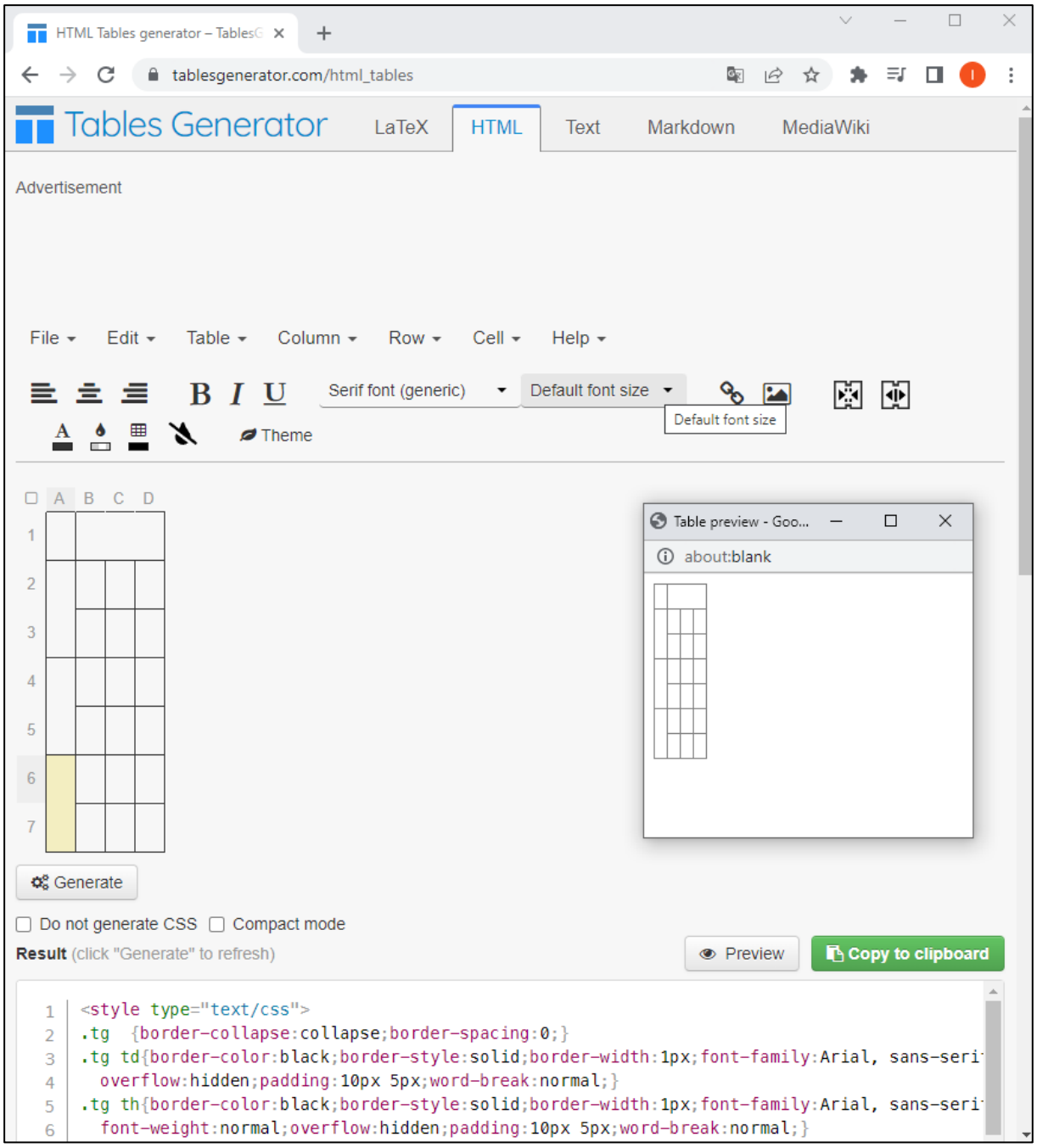


Рис. 2.137. Визуальный конструктор HTML-таблиц и возможностью CSS-стилизации

2.8.4. Устаревшая практика использования таблиц

Таблицы и WEB 1.0

Таблицы можно использовать в качестве каркаса веб-страницы. Обычно в этом случае толщину рамки берут равной нулю. Таким образом пользователь не увидит таблицу, однако элементы в ячейках будут располагаться требуемым образом.

Табличная верстка активно практиковалась раньше (другого подхода попросту еще не было). Вы можете убедиться в этом, поискав старые сайты и изучив их HTML-код.

Блочная верстка и WEB 2.0

Однако в современной верстке использовать таблицы в качестве инструмента разметки контента на странице не рекомендуется! Этот подход является устаревшим. Сейчас используется *блочная разметка* с применением технологии CSS, поддерживающей различные методы позиционирования блоков, элементов и их комбинаций.

Таблицы необходимо использовать по назначению – для разметки структурированных текстовых данных.

Следующий пример будет носить исключительно демонстрационных характер использования таблиц в качестве каркаса сайта. В следующей главе читатель познакомится и изучит основные приемы блочной верстки.

Пример табличной верстки

1. Подготовка каркаса

Создайте новый каталог проекта с названием *Frontend*. Добавьте HTML-файл с типовой начальной разметкой.

Вручную или с помощью генератора HTML-таблиц создайте таблицу вида (см. код ниже):

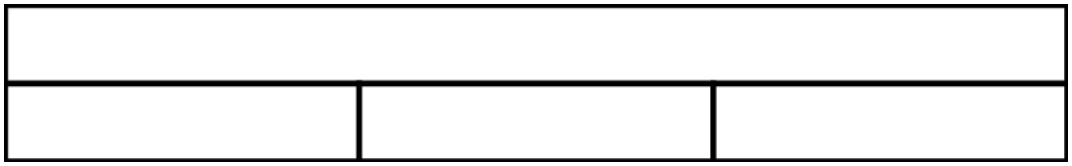


Рис. 2.138. Каркас таблицы

Установите фиксированную ширину таблицы и выравнивание по центру. Границу не отображать, т.к. таблица будет лишь невидимым каркасом для будущего контента:

Глава 2\Frontend\index.html

```
<table align="center" width="960">
    <tr>
        <td colspan="3">
            <!-- Заголовок, меню и введение -->
        </td>
    </tr>
    <tr>
        <td>
            <!-- Здесь будет разметка про HTML -->
        </td>
        <td>
            <!-- Здесь будет разметка про CSS -->
        </td>
        <td>
            <!-- Здесь будет разметка про JS -->
        </td>
    </tr>
</table>
```

2. *Наполнение ячеек каркаса содержимым*

В правом верхнем углу редактора нажмите на кнопку  . Редактор разделится на две колонки. Перетащите в новую колонку ранее созданный вами файл в проекте *Гиперссылки*. В таком режиме существенно удобнее копировать данные из одного файла в другой:

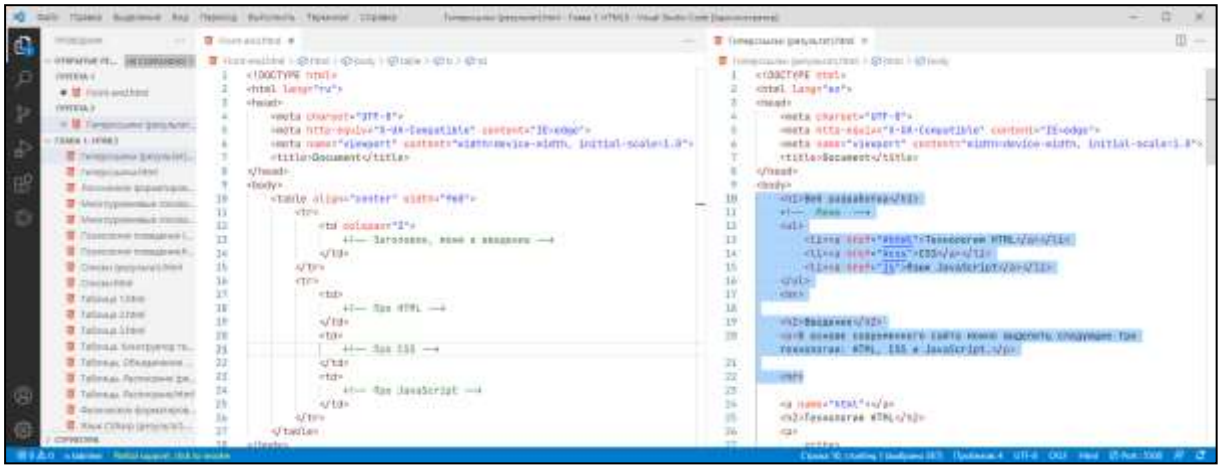


Рис. 2.139. Деление редактора на колонки

В соответствующие ячейки таблицы скопируйте необходимую разметку. Получится результат следующего вида:

Рис. 2.140. Предварительная разметка содержимого

По умолчанию HTML балансирует ширину колонок по содержимому, что в нашем случае смотрится негармонично. Кроме того, содержимое ячеек выравнивается по центру относительно вертикали.

a. В строке с тремя колонками установите равную ширину (по 320) последним (см. рис. 2.141).
b. Вертикальное выравнивание в трех колонках должно быть по верхнему краю (см. атрибуты).
c. Установите свойству **`cellpadding`** значение порядка 10-15, чтобы содержимое не примыкало близко к границам.

Окончательный результат изображен на рис. 2.142.

```
<table align="center" width="960" cellpadding="15">
    <tr>
        <td colspan="3">
            <!--  Меню  -->
        </td>
    </tr>

    <tr valign="top">
        <td width="320">
            <!--  Здесь будет разметка про HTML  -->
        </td>

        <td width="320">
            <!-- Здесь будет разметка про CSS  -->
        </td>

        <td width="320">
            <!-- Здесь будет разметка про JS  -->
        </td>
    </tr>
</table>
```

Рис. 2.141. Настройка выравнивания и ширины колонок

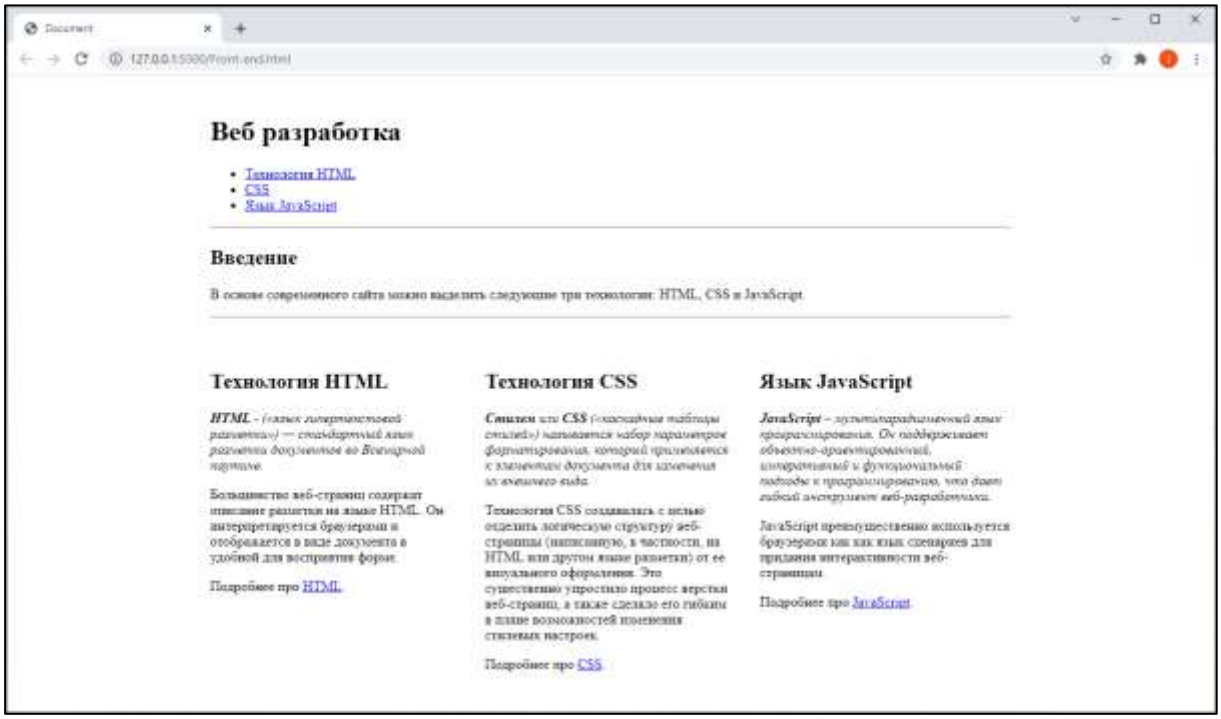


Рис. 2.142. Окончательный вид документа

Вопросы для самопроверки

1. Какие теги обязательно должны входить в структуру таблицы?
2. Опишите теги, которые необходимы для формирования полноценной структуры таблицы.
3. Опишите алгоритм объединения ячеек в таблице.
4. Чем могут помочь конструкторы таблиц? Перечислите их возможности и приведите примеры веб-сервисов.
5. Каким образом таблицы можно использовать в разметке макета веб-страниц и почему от этой практики в настоящее время отказались?

Практикум

Задание 1

1. Создайте каталог *Расписание* и стартовый HTML-файл.
2. Изучите начатую разметку таблицы.
3. Дополните таблицу недостающими строками с расписанием. Ориентируйтесь на образец рис. 2.143.

Глава 2\Расписание\index.html

```
<!DOCTYPE html>
<html lang="ru">

<head>
    <meta charset="UTF-8">
    <meta http-equiv="X-UA-Compatible" content="IE=edge">
    <meta name="viewport" content="width=device-width,
initial-scale=1.0">
    <title>Document</title>
</head>

<body>
    <h1>Понедельник</h1>
```

```
    <table border="2" cellpadding="8" align="center">
        <tr>
            <th>№</th>
            <th>Дисциплина</th>
            <th>Преподаватель</th>
            <th>Аудитория</th>
        </tr>
        <tr>
            <td>1</td>
            <td>Разработка веб-сайтов (лк.)</td>
            <td>Якубович Д.А.</td>
            <td>242-7</td>
        </tr>

        <!-- Добавьте недостающие строки -->

    </table>
</body>
```

Задание 2

1. Скопируйте код созданной в задании 1 таблицы ниже.
2. Расширьте до полноценной таблицы, используя ее структурные теги, описанные в пункте 2.8.2.

Задание 3

1. Создайте каталог *Погрешность замеров*, создайте стартовую разметку.
2. Перейдите на любой сайт-конструктор HTML-таблиц (см. ссылки, приведенные ранее).
3. В начале постройте сетку 6x4 и путем объединения ячеек получите и заполните таблицу, как на рис. 2.144.
4. Настройте атрибуты таблицы и ячеек следующим образом:
    a. ширина каждой колонки (атрибут **`width`**) составляет 80px;
    b. поля ячейки (атрибут **`cellpadding`**) – 10px.

Задание 4

1. Создайте каталог *Frontend*.
2. Реализуйте в нем документ по табличному макету, описанному в пункте 2.8.4.

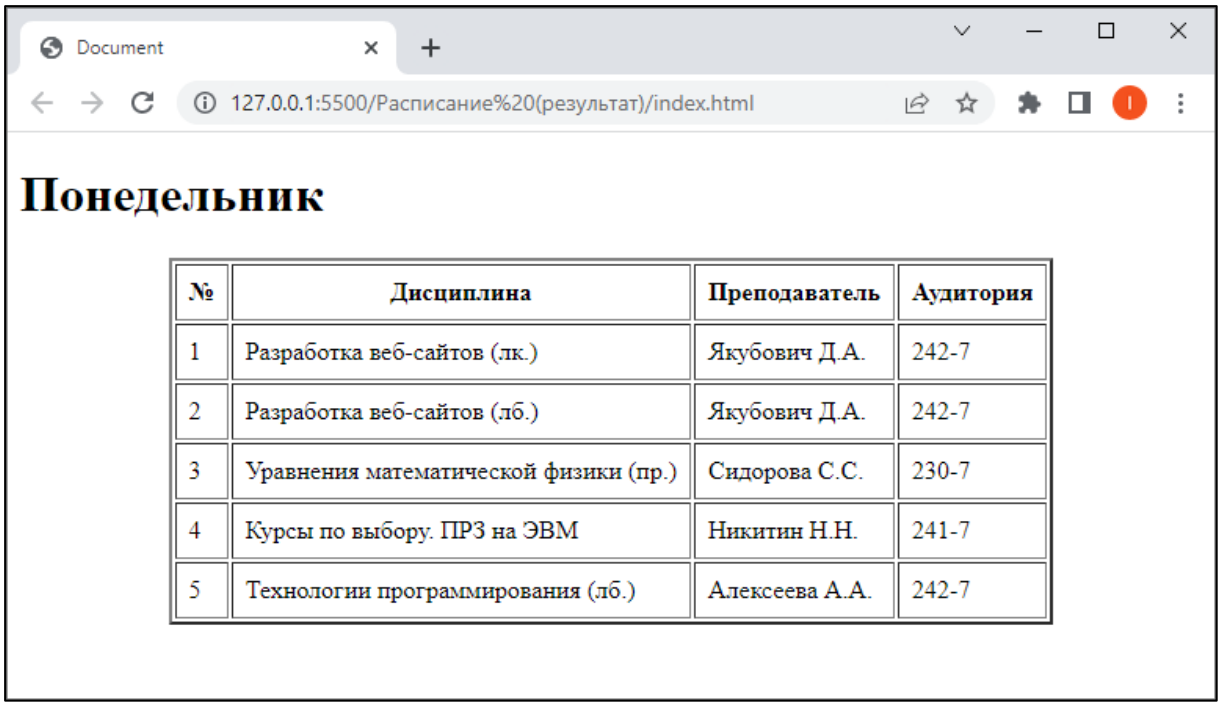


Рис. 2.143. Образец для задания 1-2

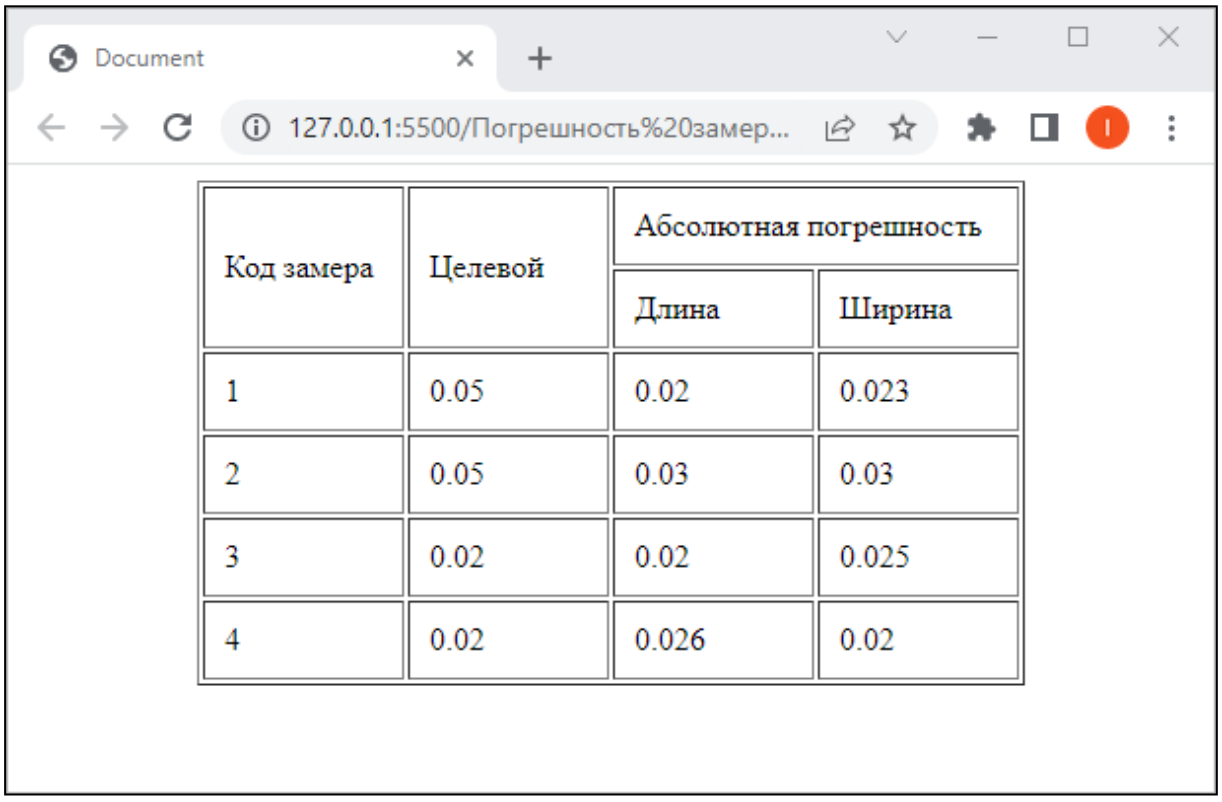


Рис. 2.144. Образец для задания 3

2.9. Работа с изображениями

2.9.1. Вставка изображения

Особенности стандартной обработки изображений

Изображения как плавающий объект

Для вставки изображений в HTML-документы используется унарный тег **`<img>`** (см. далее). Изображения относятся к *плавающим элементам*, т.е. другие объекты (в частности, текст) его могут обтекать. Однако подобное поведение изображений в современном вебдизайне используется редко и изображения чаще оформляют блоками, свойства которых задаются CSS-свойствами.

Поддерживаемые форматы

HTML5 поддерживает большинство важнейших форматов изображений (как растровых, так и векторных): JPEG, BMP, ICO, GIF, PNG, APNG, SVG.

Синтаксис

Определение

`<img>`

Унарный тег, вставляющий ***изображение****.*

Обладает двумя обязательными атрибутами.

- ***`src`*** *– определяет ссылку на изображение или его имя (в имени обязательно нужно указывать его формат);*
- ***`alt`*** *– альтернативный текст; отображается вместо изображения, если оно не найдено, не полностью загружено, либо возникли проблемы с распознанием формата.*

По умолчанию поведение изображения близко к строчным элементам: оно может быть прижато к левой или правой границе родительского блока.

Глава 2\Изображения\Стандартное\index.html

```
<h1>Фракталы</h1>
<p>Термин "фрактал" введен Бенуа Мандельбротом в 1975 г.
и получил широкую известность с выходом в 1977 году его
книги "Фрактальная геометрия природы".</p>

<img src="fractal.jpg" alt="фрактал" height="120" >

<p>С конца XIX века в математике появляются примеры
самоподобных фигур с неклассическими свойствами.
К ним можно отнести:</p>
<ul>
    <li>множество Кантора;</li>
    <li>треугольник Серпинского;</li>
    <li>кривую Пеано.</li>
</ul>
```

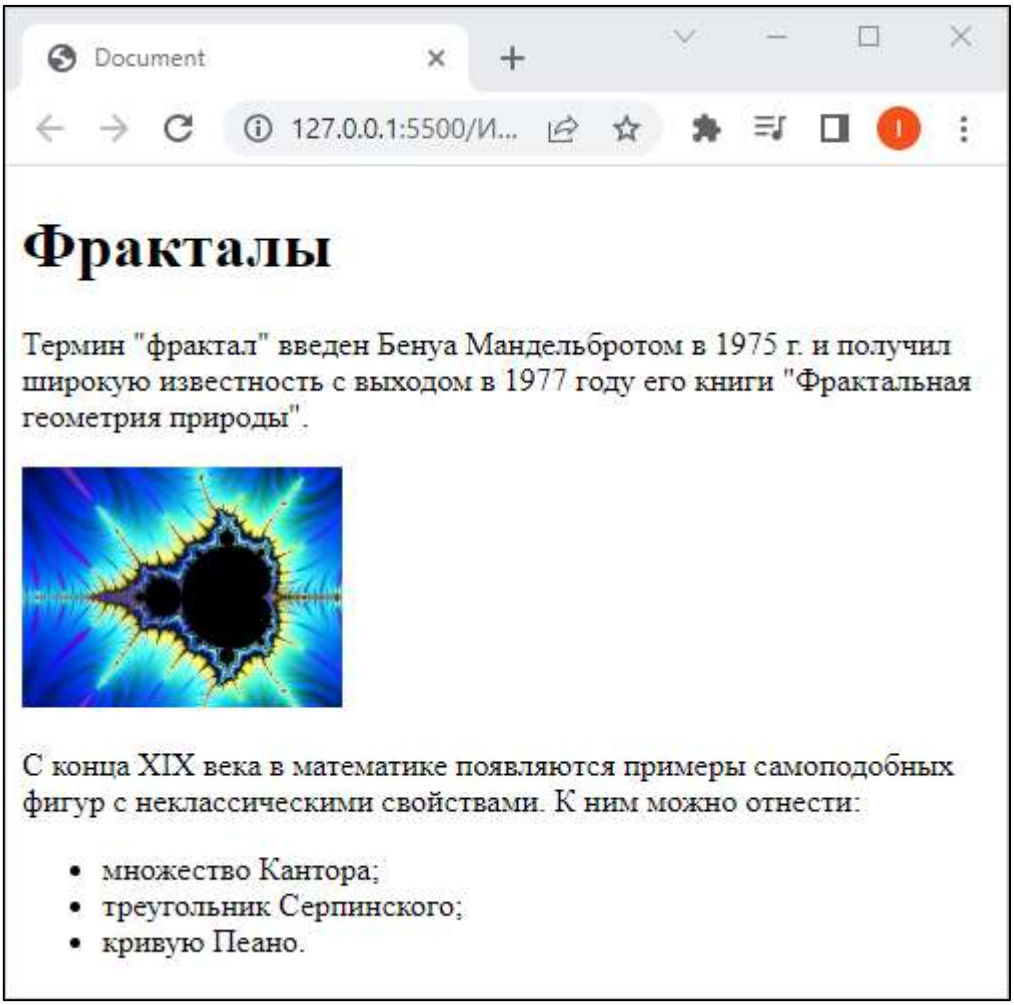


Рис. 2.145. Сверху и снизу изображения находятся блоки абзацев

Атрибуты

src

Обязательный атрибут **`src`**, который указывает гиперссылку или путь к изображению. Допускается абсолютная или относительная ссылка.

alt

Обязательный атрибут **`alt`** подставляет альтернативный текст вместо изображения, если оно не загружено по каким либо причинам. Это дает пользователю хотя-бы краткую информацию в случае неуспешной загрузки.

width и height

Атрибуты **`width`** и **`height`** задают ширину и высоту изображения в пикселях. Несмотря на то, что размеры изображений настраивают в CSS-стилях, указанные атрибуты позволят избежать «прыжков» содержимого при загрузке страницы с большими изображениями. Это связано с тем, что сначала браузер строит HTML-структуру документа и лишь затем применяет CSS.

При этом, если задать только один из атрибутов, то изображение сохраняет пропорции и вычисляет второй атрибут автоматически.

Другие атрибуты

Атрибуты **`srcset`** и **`sizes`** позволяют адаптировать изображения под ширину экрана:

- **`srcset`** выбирает один из нескольких вариантов изображения в зависимости от ширины экрана устройства;
- **`sizes`** меняет размер изображения в зависимости от ширины экрана устройства.

Атрибут **`loading`** может управлять процессом загрузки изображений: загружать некоторые из них сразу или дождаться прокрутки пользователя.

Атрибут **`decoding`** позволяет отсрочить отрисовку изображений до тех пор, пока не будет прогружена структура и оформление других элементов.

Подробнее информацию о работе с этими атрибутами и примеры использования можно найти на сайте doka.guide.

Путь к изображению

1. *Абсолютные и относительные пути*

Атрибут `src` может указывать как абсолютный, так и относительный путь к изображению.

Если указать только имя изображения, то браузер будет его искать в той же директории, где находится HTML-файл:

```
<img src="Котики.png">
```

Обычно изображения группируются по каталогам, чтобы избежать путаницы во множестве файлов. Тогда в относительном пути необходимо указать и каталог(и) относительно HTML-файла:

```
<img src="images/myphotos/Котики.png">
```

Полный абсолютный путь указывать не следует: при изменении директории проекта ссылки перестанут работать и их придется править заново:

```
<img src="D:/images/myphotos/Котики.png">
```

2. *Путь относительно домена сайта*

При переносе вашего сайта на хостинг важно помнить, что здесь работает еще один способ относительной адресации: путь относительно корня сайта. Причем это справедливо не только для изображений, но и для гиперссылок в целом.

Предположим, вы зарегистрировали свой домен:

```
yakubovich.studylib.ru
```

Полный путь к вашему сайту, например:

```
http://www.yakubovich.studylib.ru
```

Если изображения находятся в отдельных каталогах, то слеш в начале пути требует перейти в корень сайта и уже из него осуществить переход по каталогам:

```
<img src="/images/myphotos/Котики.png">
```

При этом глубина вложения относительно корня сайта не имеет значения.

Такая адресация еще удобна и потому, что при изменении доменного имени ссылки сохраняют работоспособность.

Приемы работы с Visual Studio Code

Visual Studio Code позволяет подсказывать и завершать названия файлов изображений.
Для этого поставьте курсор в атрибуте **`src`**, *нажмите* ***CTRL + Пробел***. *Далее выберете конкретное изображение из текущего каталога или вложенных.*

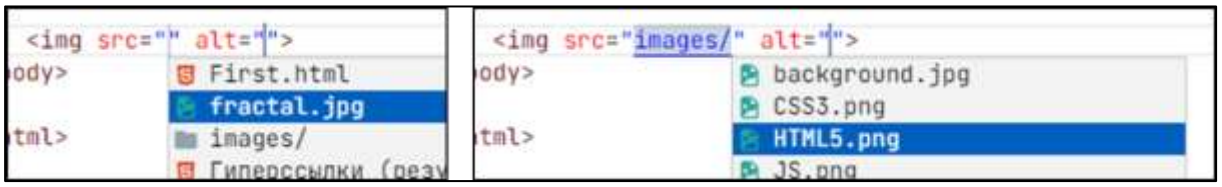


Рис. 2.146. Перемещение по каталогам и файлам в атрибуте `src`

2.9.2. Системный подход к разметке изображений

Верстка иллюстраций

Определение

`<figure></figure>`

Контейнер используется для разметки ***иллюстраций****, которое необходимо сопроводить подписью.*

`<figcaption></figcaption>`

Задает ***подпись*** *к изображению. Располагается внутри* ***`<figure></figure>`***.

Стандартно тег включает в себя изображение и подпись. Последняя должна быть либо в начале, либо в конце разметки. При необходимости подпись можно расширить и другими тегами разметки текста:

`Глава 2\Изображения\Figure\index.html`

```
<figure>
    <img src="fractal.jpg" alt="фрактал">
    <figcaption>Множество Мандельброта</figcaption>
</figure>
```

Обычно тег `<figure>` используют в разметке автономных блоков изображений, на которые необходимо ссылаться из содержимого документа. При этом предполагается, что `<figure>` можно перенести в другую часть документа, не нарушая при этом целостности и корректности отображения изображений.

Глава 2\Изображения\Figure\index.html

```
<!-- Изображение с подписью -->
<figure>
    <img src="/fractal.jpg" alt="фрактал">
    <figcaption>Множество Мандельброта</figcaption>
</figure>

<!-- Только изображение -->
<figure>
    <img src="/fractal.jpg" alt="фрактал">
</figure>
```

Адаптация изображений под разные устройства

Определение

`<picture></picture>`

*Тег **адаптирует изображения** под различные устройства и форматы. Обязательно должен содержать внутри **`<img>`** и опционально один или несколько **`<source>`**.*

`<source></source>`

*Определяет для **`<picture>`** изображение, которое будет вставлено в зависимости от текущей ширины экрана устройства.*

Контейнер `<picture>` позволяет обеспечить адаптивный подбор изображений на уровне HTML-кода, когда как обычно это прерогатива CSS и JavaScript.

Принцип действия <picture> следующий:

- Последовательно проверяются условия в атрибуте **media** для тега <source>. Если они справедливы, то выбирается изображение из атрибута **srcset**.
- Иначе осуществляется аналогичная проверка следующего <source> и т.д., пока условие ложно.
- В случае ложности всех условий в тегах <source> вставляется изображение тега <img>.

В следующем примере имеются три изображения: большое, среднее и малое (на практике это может быть одно и тоже изображение, но подготовленное в графическом редакторе под разные размеры). Если ширина экрана (или окна браузера) больше 960 px, то выбирается большое изображение, иначе если больше 680 px – выбирается среднее, в противном случае – малое.

Глава 2\Изображения\Picture\index.html

```
<picture>
    <source media="(min-width: 960px)"
            srcset="big-img.jpg">
    <source media="(min-width: 680px)"
            srcset="mid-img.jpg">
    <img src="small-img.jpg" alt="фрактал">
</picture>
```

Однако выбор изображения может зависеть не только от параметров его ширины. Атрибут media тега <sourse> может учитывать такие критерии, как:

- разворот изображения (портрет / альбомный);
- тема оформления (светлая / темная);
- формат графического файла и т.д.

Такая гибкость во многом связана с тем, что media копирует синтаксис *медиа-запросов* в CSS – это один из базовых приемов организации адаптивного веб-дизайна (вопрос подробнее рассматривается в главе 3).

Вопросы для самопроверки

1. Почему изображения в HTML часто называют «плавающими» объектами и каким образом эта особенность поведения может быть изменена?
2. Опишите синтаксис тега `<img>` и роль его основных атрибутов.
3. Чем отличаются атрибуты `width` и `height` от CSS-стилей?
4. В чем разница между абсолютной и относительной адресацией путей к изображениям?
5. Перечислите преимущества использования контейнера `<figure>` при верстке иллюстраций.
6. Какие возможности работы с изображением дает тег `<picture>`?

Практикум

Задание 1

1. Откройте ранее разработанный проект *Frontend* (из практикума параграфа 2.8):
2. Вставьте в нужное место изображение и задайте ему ширину 100%: в этом случае оно займет по ширине ровно столько, какова ширина ячейки таблицы (рис. 2.147).
3. Для изображений выделите отдельный каталог *images* (учитывайте это, когда будете ссылаться на изображения).
4. Добавьте третью строку в таблицу макета (сразу после строки с текстом), где будут размещаться изображения-логотипы. Вставьте следующую разметку:

Глава 2\Frontend\index.html

```
<table align="center" width="960" cellpadding="15">

    <!-- Выше две другие строки с контентом -->

    <tr valign="top" align="center">
        <td width="320">
            <img src="images/HTML5.png" width="30%">
        </td>
```

```
            <td width="320">
                <img src="images/CSS3.png" width="30%">
            </td>

            <td width="320">
                <img src="images/JS.png" width="30%">
            </td>
        </tr>
    </table>
```

5. Если все сделано корректно, то получится результат, как изображено на рис. 2.148.

Задание 2

1. Скорректируйте разметку изображения в блоке введения, используя тег `<figure>`.
2. Задайте подпись к изображению.

Задание 3

1. Приведите пример использования тега `<picture>` для разметки изображения, которое адаптируется под ширину окна браузера.
2. В качестве ограничений взять ширину 1280 px и 720 px.

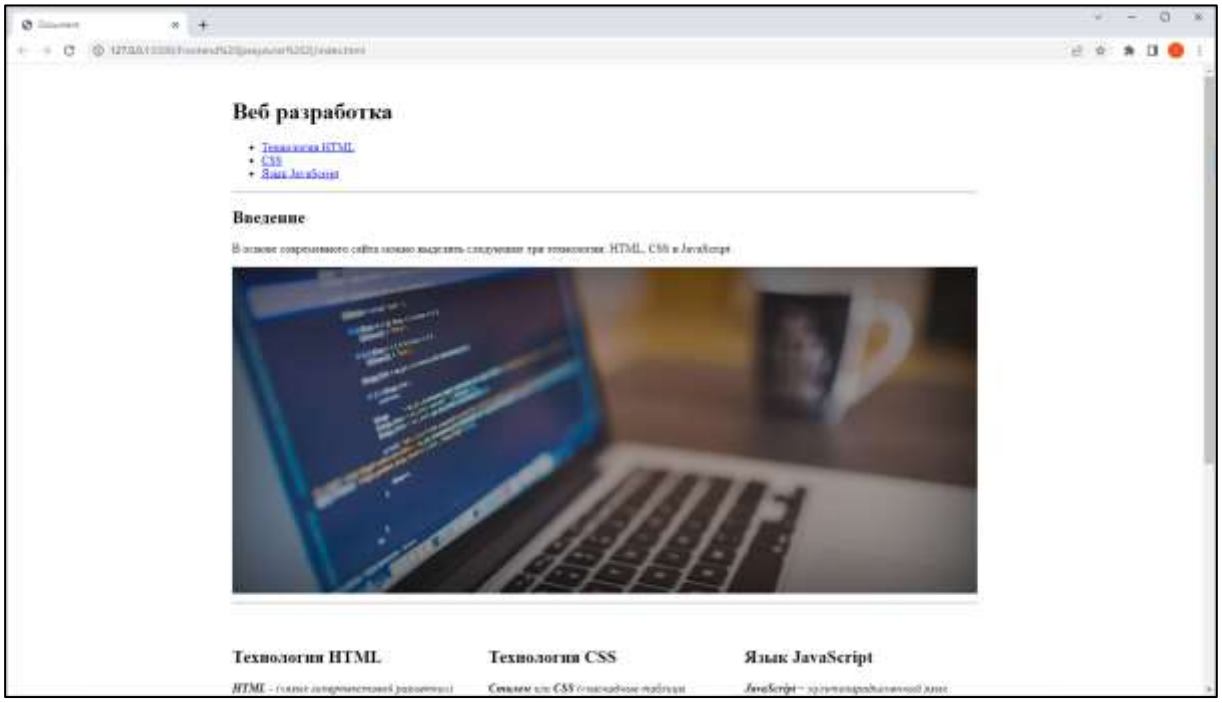


Рис. 2.147. Образец выполнения задания 1: вставка изображения

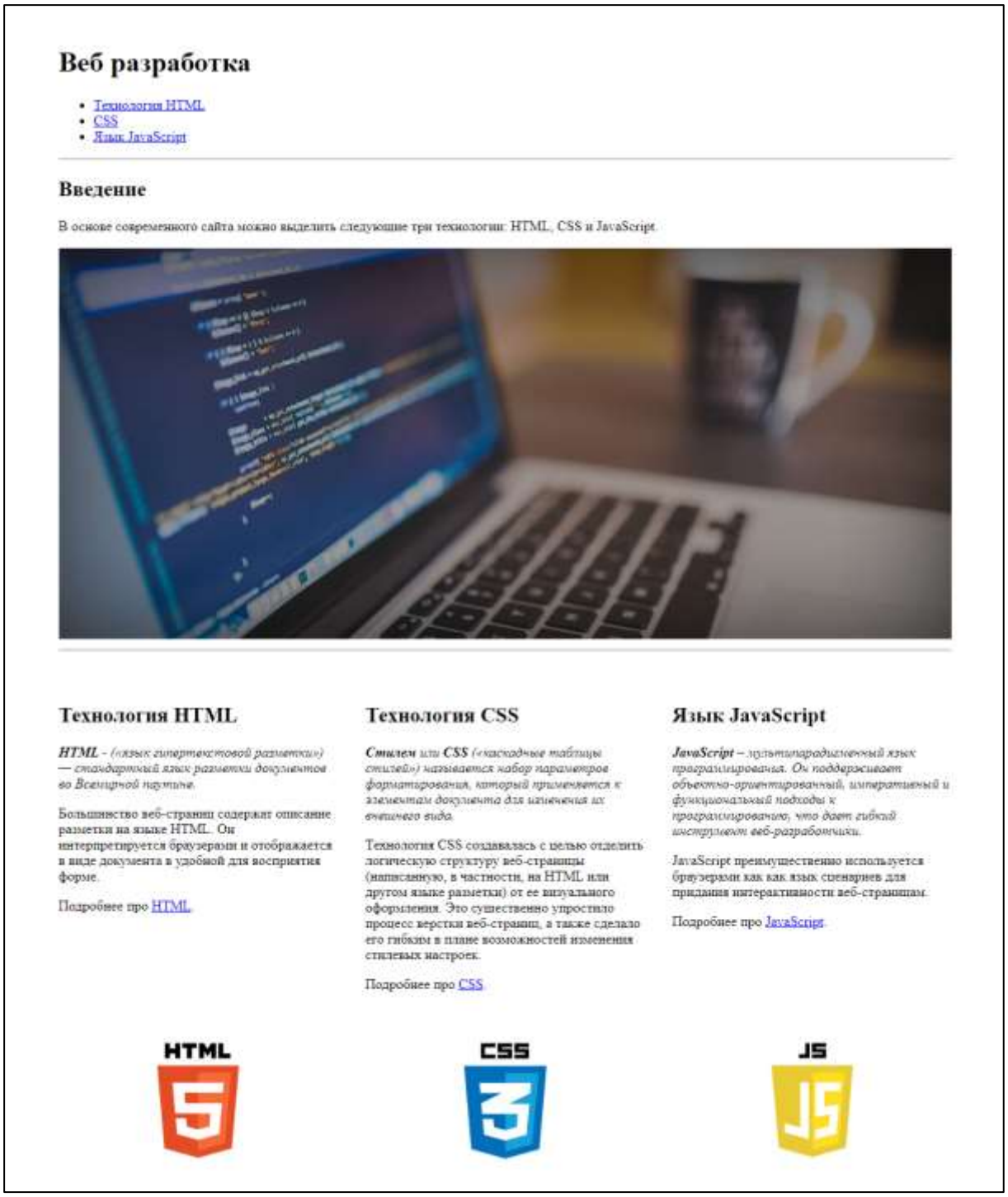


Рис. 2.148. Образец выполнения задания 1: вставка изображений в колонках

2.10. Новые теги HTML5

2.10.1. Теги структуры документа

Основная структура документа

Определение

```
<header></header>
```

*Задает **область верхнего информационного блока** («шапка» сайта). Обычно включает логотипы, основные разделы сайта, элементы управления, поиска, навигации, регистрации и т.п.*

```
<footer></footer>
```

*Задает **область нижнего информационного блока** («подвал» сайта). В «подвале» обычно размещается контактная информация и ссылки.*

```
<aside></aside>
```

*Задает **панель бокового меню** («сайдбар»). Контейнер удобно использовать для оформления блока меню, боковой панели, либо поясняющего раздела. Допускает многократное использование.*

```
<nav></nav>
```

*Задает **блок навигации** по сайту или краткой ее реализации. Обычно в этом блоке присутствуют самые важные ссылки на другие разделы сайта. В целом, может располагаться в любом блоке. Допускает многократное использование.*

```
<main></main>
```

*Задает **блок контента**. Определяет область для основных данных, которые не должны дублироваться. Он может служить основным для разметки контента, но используется только один раз.*

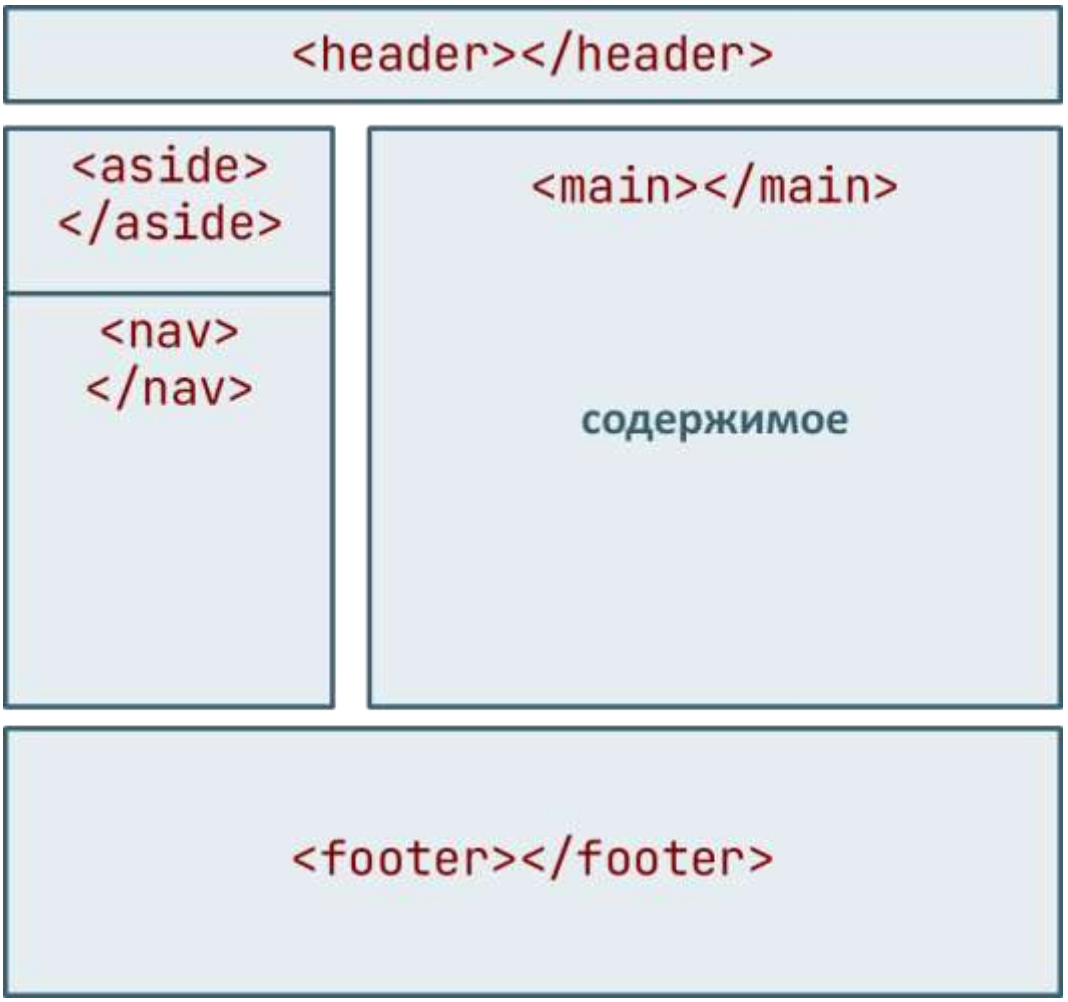


Рис. 2.149. Теги основной структуры документа

Теги секционирования

Определение

`<article></article>`

*Размечает **тематический блок**: новости, статьи, пост в блоге и другой автономный блок информации. Может включать в себя такие разделы, как* **`<header>`**, **`<footer>`** *и* **`<article>`**.

`<section></section>`

*Размечает **блок секционирования**. Используется для выделения определенных смысловых частей (пунктов, параграфов). Внутри первым тегом рекомендуется брать один из заголовочных (*`<h1>`, `<h2>` *и т.д.).*

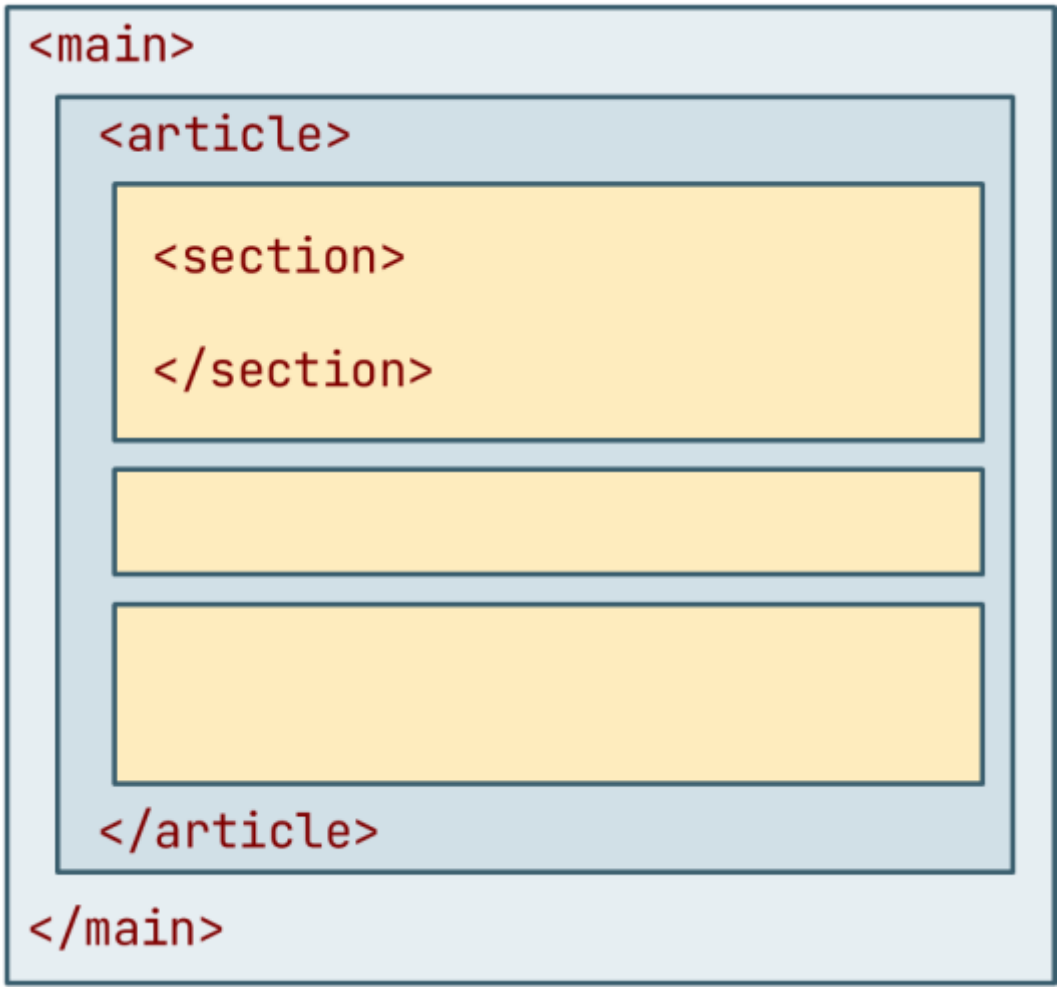


Рис. 2.150. Теги разбиения содержимого на секции

Таким образом, структура тела HTML5-документа подразумевает использование следующих семантических тегов:

```
<body>
    <header>
        <!-- Шапка сайта -->
    </header>
    <aside>
        <!-- Боковая панель (или несколько) -->
    </aside>
    <main>
        <!-- Блок контента -->
    </main>
    <footer>
        <!-- Подвал сайта -->
    </footer>
</body>
```

2.10.2. Информационные теги

Семантика текста

Определение

```
<details></details>
<summary></summary>
```

*Размечает область **долполнительной информации**, которую можно раскрывать или скрывать щелчком мыши. Содержит в себе контейнер **<summary></summary>**, который помечает текст, активирующий блок (см. пункт 2.5.2).*

```
<mark></mark>
```

*Размечает блок **выделенного текста** (аналог выделения важного текста маркером на бумаге).*

Разметка данных об авторе

Определение

```
<address></address>
```

*Размечает **информацию об авторе** проекта, статьи, блога и т.п.*

```
<time></time>
```

*Размечает **заметку о времени** или **дате** публикации статьи, последних изменений и т.д.*

Пример:

```
<address>
    <p>Разработчик:
        <a href="mailto:yakubovich.studylib@mail.ru">
            Якубович Д.А.
       </a>
    </p>
    <p>Дата последней редакции:
        <time>11.06.2022</time>.</p>
</address>
```

2.10.3. Объекты мультимедиа

Изображения и графика

Определение

`<figure></figure>`

*Вставляет **объект** (часто размечает графический элемент). Подходит для разметки области таких элементов как изображения, схемы, таблицы. Обычно содержит в себе тег **<figcaption></figcaption>**, приводящий текст пояснения к объекту.*

`<canvas></canvas>`

*Размечает контейнер для **рисования графических элементов** средствами JavaScript.*

Пример использования `<canvas>` (рис. 2.151):

Глава 2\Новые теги HTML5\Графика\index.html

```
<h1>Графические примитивы в canvas</h1>
<canvas id="draw-box" width="400" height="320"></canvas>

<script src="code.js"></script>
```

Глава 2\Новые теги HTML5\Графика\code.js

```
let canvas = document.getElementById("draw-box");
let ctx = canvas.getContext("2d");

ctx.fillStyle = "#FFEEEE";
ctx.fillRect(0, 0, 400, 320);

ctx.fillStyle = "#FF0023";
ctx.fillRect(40, 40, 180, 50);
```

Для визуализации работы с `<figure>` воспользуемся CSS (рис. 2.152):

Глава 2\Новые теги HTML5\Изображения\index.html

```
<h1>Вставка изображений в блоках</h1>
<figure>
    <img src="media/cat.jpg">
```

```
    <figcaption>Рис. 1. Котенок</figcaption>
</figure>
```

Глава 2\Новые теги HTML5\Изображения\style.css

```
* {
    box-sizing: border-box;
}

body {
    font: 16px Tahoma, Arial;
    background: #2d353f;
    margin: 0;
}

h1 {
    color: #fff;
    background: #44638a;
    margin: 0;
    padding: 17px 30px;
}

figure {
    display: block;
    max-width: 960px;
    margin: 20px 30px 10px;
    position: relative;
}

figure>img {
    display: block;
    width: 100%;
    border: 6px solid #cbd9e9;
    border-radius: 10px;
    box-shadow: 7px 8px 20px #000000ad;
}

figure>figcaption {
    display: block;
    color: #fff;
    margin-top: 20px;
}
```

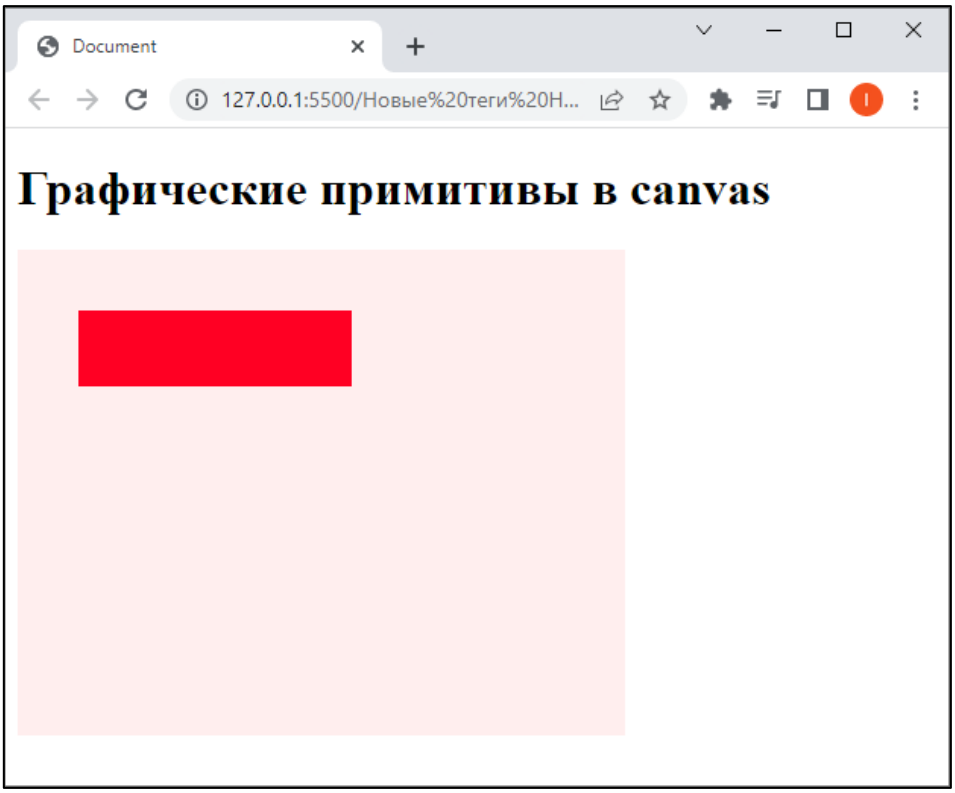


Рис. 2.151. Работа с графическими примитивами

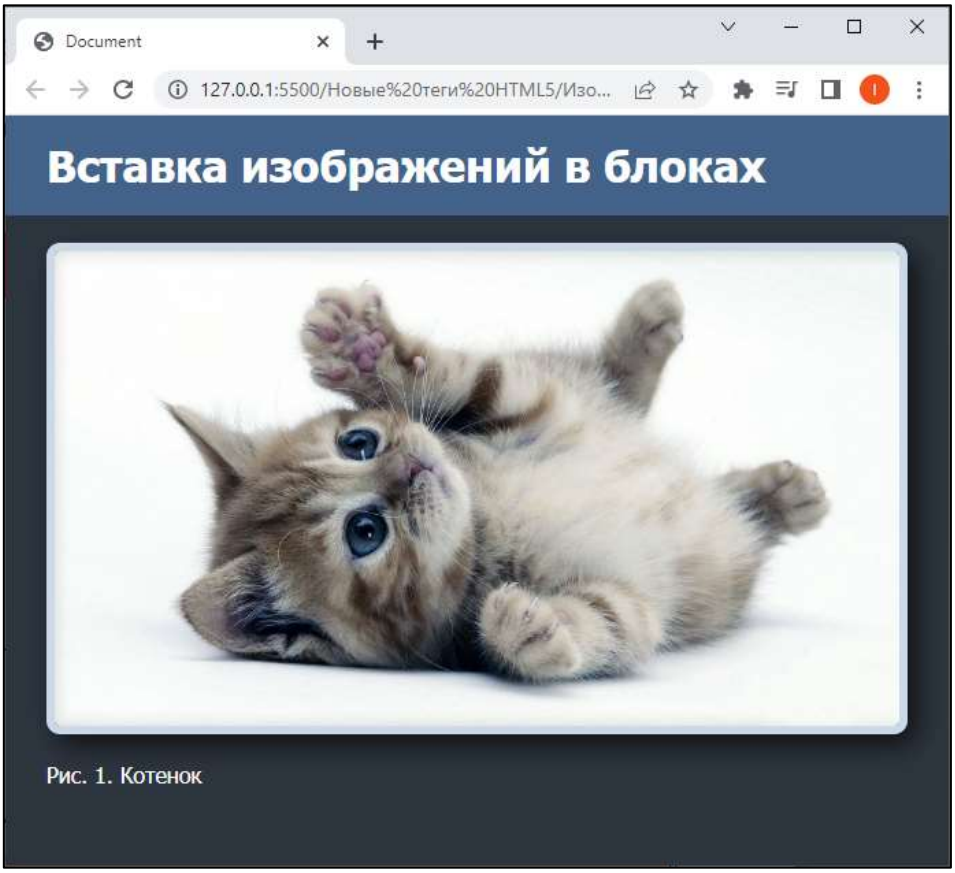


Рис. 2.152. Разметка и оформление иллюстраций

Аудио и видео

Определение

```
<audio></audio>
<video></video>
```

*Вставляет **мультимедиа элементы**. В HTML5 окончательно внедрена возможность вставки видео и звуковых проигрывателей. Тег **<sourse>** используется перечисленными тегами для указания источника для воспроизведения.*

```
<embed></embed>
```

*Осуществляет **встраивание внешних ресурсов** на веб-страницу. Предполагается, что содержимое элемента для корректной работы в браузере требует подключения внешних плагинов или специальных программ.*

Приведем примеры вставивания аудио и видео, а также вложений (рис. 2.153-рис. 2.154). Для подстраховки имеется по три копии файлов, но в разных форматах. Если браузер не сможет распознать хотя-бы один из них, то отобразится текст предупреждения:

Глава 2\Новые теги HTML5\Мультимедиа\index.html

```
<h1>Вставка аудио</h1>
<h2>Упрощенный подход</h2>
<audio src="media/трек.mp3" controls></audio>

<h2>Улучшенный подход</h2>
<audio controls>
    <source src="media/трек.mp3" type="audio/mpeg">
    <source src="media/трек.ogg" type="audio/ogg">
    <source src="media/трек.wav" type="audio/wav">
    <p>Ваш браузер не поддерживает проигрывание
        аудио.</p>
</audio>

<h1>Вставка видео</h1>
<h2>Упрощенный подход</h2>
<video src="media/Stars.mp4" controls
width="400"></video>
```

```
<h2>Улучшенный подход</h2>
<video controls poster="media/Preview.jpg" width="400">
    <source src="media/Stars.mp4" type="video/mp4">
    <source src="media/Stars.webm" type="video/webm">
    <source src="media/Stars.ogv" type="video/ogg">
</video>

<h1>Вложение сторонних компонент</h1>
<embed type="text/html" src="file.html" width="500"
height="200">
```

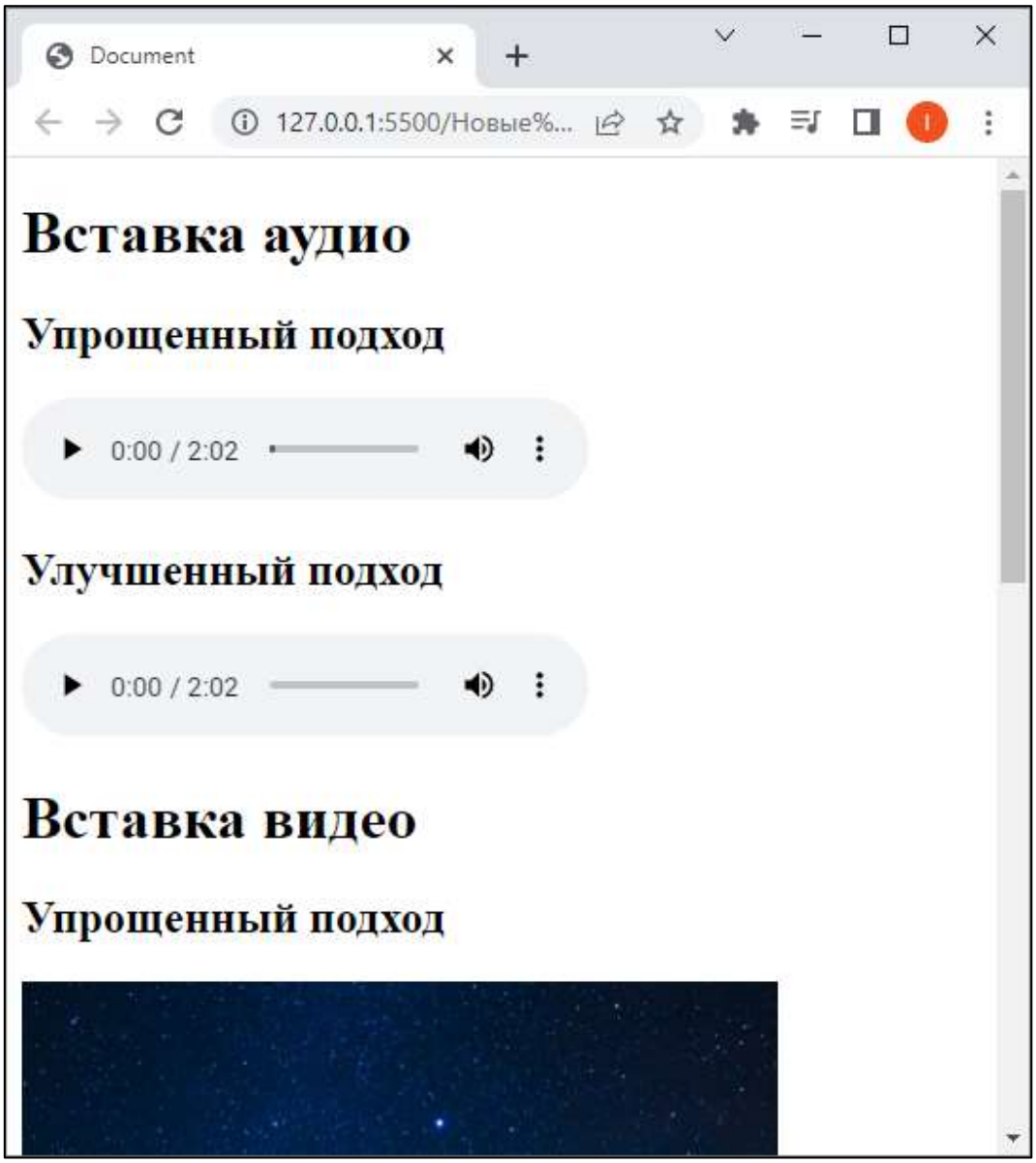


Рис. 2.153. Вставка мультимедийных элементов (часть 1)

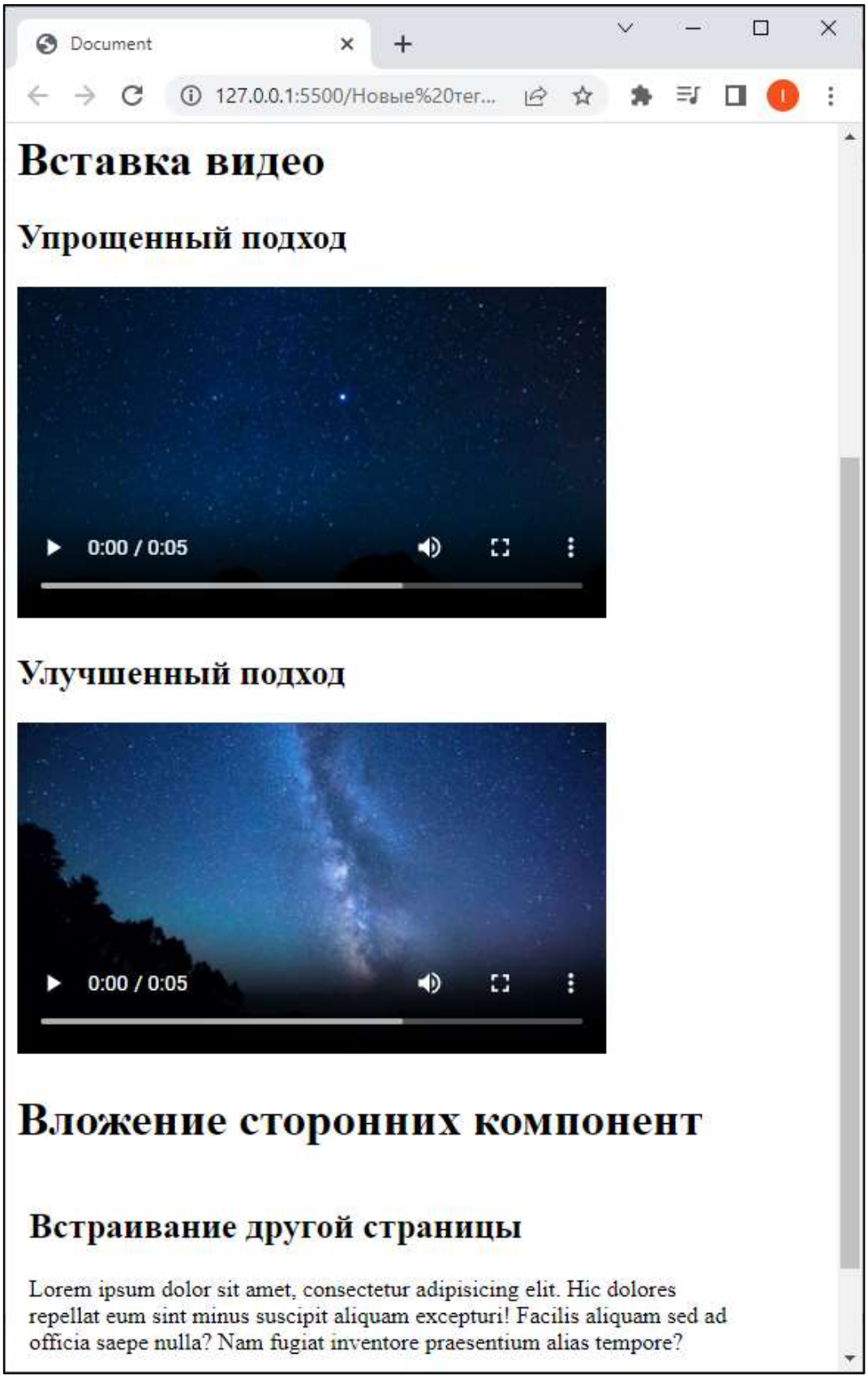


Рис. 2.154. Вставка мультимедийных элементов (часть 2)

Вопросы для самопроверки

1. Опишите назначение тегов структуры и секционирования HTML5-документа.
2. Какие новые теги для разметки текста появились в HTML5?
3. Какую полезную информацию можно извлечь об авторе?
4. Опишите возможности тегов для работы с изображениями и графикой.
5. Перечислите нововведения HTML5 в вопросе вставки аудио и видео контента.

Практикум

Задание 1

1. Создайте каталог *Структура документа*.
2. Осуществите разметку основных разделов веб-страницы (без уточнения содержимого), которая может быть использована для следующего макета:

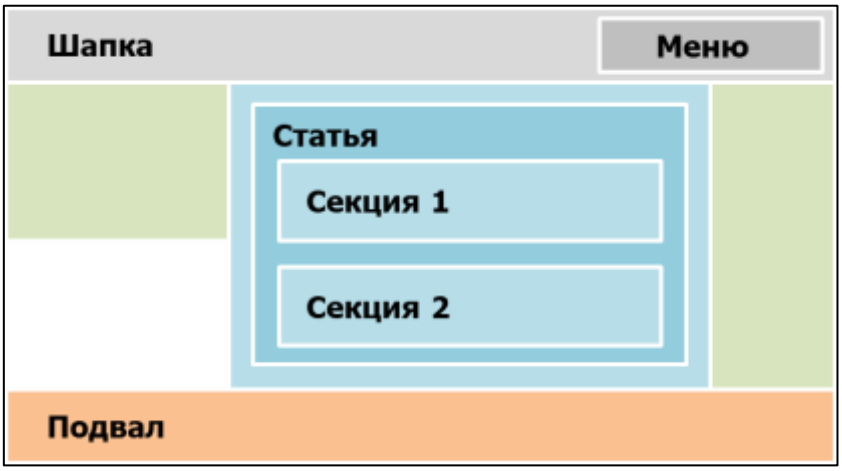


Рис. 2.155. Макет для задания

Задание 2

1. Создайте каталог *Работа с аудио и видео*.
2. Вставьте в разметку несколько скачанных аудио- и видеотреков.
3. Проверьте корректность их проигрывания страницы в разных браузерах.

ГЛАВА 3.
ТЕХНОЛОГИЯ CSS3

3.1. Понятие, назначение и возможности технологии CSS

3.1.1. Понятие CSS

CSS как технология оформления

Определение

CSS *или* ***стилем*** *(**C**ascading **S**tyle **S**heets,* ***каскадные таблицы стилей****) называется набор параметров форматирования, который применяется к элементам HTML-документа для изменения их внешнего вида.*

Основная задача HTML – задать структуру документа, в котором элементы имеют определенную роль. Такая семантическая разметка формирует структуру, но не формирует внешний вид документа. Любые атрибуты тегов, отвечающие за оформление, считаются в HTML5 устаревшими и не рекомендуются к использованию.

Каскадные таблицы стилей позволяют отделить оформление от разметки и обладают гораздо более широким спектром возможностей форматирования.

В настоящий момент используется спецификация CSS3.

Это полезно знать!

В дальнейшем для краткости ***стилем*** *будем называть форматирование отдельного элемента, а также в целом файл CSS.*

Преимущества CSS

- *Разграничение HTML-разметки и стиля оформления*. В идеале HTML должен определять только логическую структуру документа, а вид и настройки элементов задаются через CSS-стили.
- *Разное оформление и адаптивный дизайн для различных устройств*. С помощью стилей можно определить оформление веб-страницы для разных устройств вывода: монитора, принтера, смартфона, планшета и др. При этом HTML-разметка не меняется!
- *Расширенные по сравнению с HTML способы оформления элементов*. CSS предоставляет универсальный подход к стилизации разных элементов и имеет богатые возможности их настройки, в том числе – эффектов анимации.
- *Ускорение загрузки сайта*. Разделение HTML и CSS кода оптимизирует загрузку веб-страниц.
- *Гибкость использования*. Стили можно подключать как отдельные файлы, что позволяет оформлять множество страниц веб-сайта в едином стиле, а также оперативно менять оформление.

Спецификация CSS3

- Главная особенность спецификации CSS3 – это возможность создавать анимированные элементы без использования языка JavaScript, работа с градиентами, тенями, фильтрами, эффектами сглаживания и многое другое (чего не было в ранних версиях CSS).
- CSS3 внедряет работу с медиа-запросами, которые являются основной современного адаптивного веб-дизайна;
- Технология CSS3 – отдельная технология, напрямую не связанная с HTML5. Однако основное ее применение находится именно в сочетании с HTML5.

Практический пример использования CSS

Продемонстрируем использование CSS-стилей на простом примере, не вдаваясь в подробности особенностей синтаксиса (этому вопросу посвящены разделы ниже).

Для простоты оставим описание CSS-стилей в одном файле.

Глава 3\Введение в CSS\index.html

```
<!DOCTYPE html>
<html lang="ru">

<head>
    <meta charset="UTF-8">
    <meta http-equiv="X-UA-Compatible" content="IE=edge">
    <meta name="viewport" content="width=device-width,
initial-scale=1.0">
    <title>Document</title>
    <style>
        h1 {
            background: #E1FEAB;
            font: bold 200% Tahoma;
            color: #0F3244;
            text-align: center;
            padding: 10px;
            border-radius: 30px;
            box-shadow: 0 0 10px #0F3244;
        }
    </style>
</head>

<body>
    <h1>Заголовок 1</h1>
    . . .
    <h1>Заголовок 2</h1>
    . . .
</body>

</html>
```

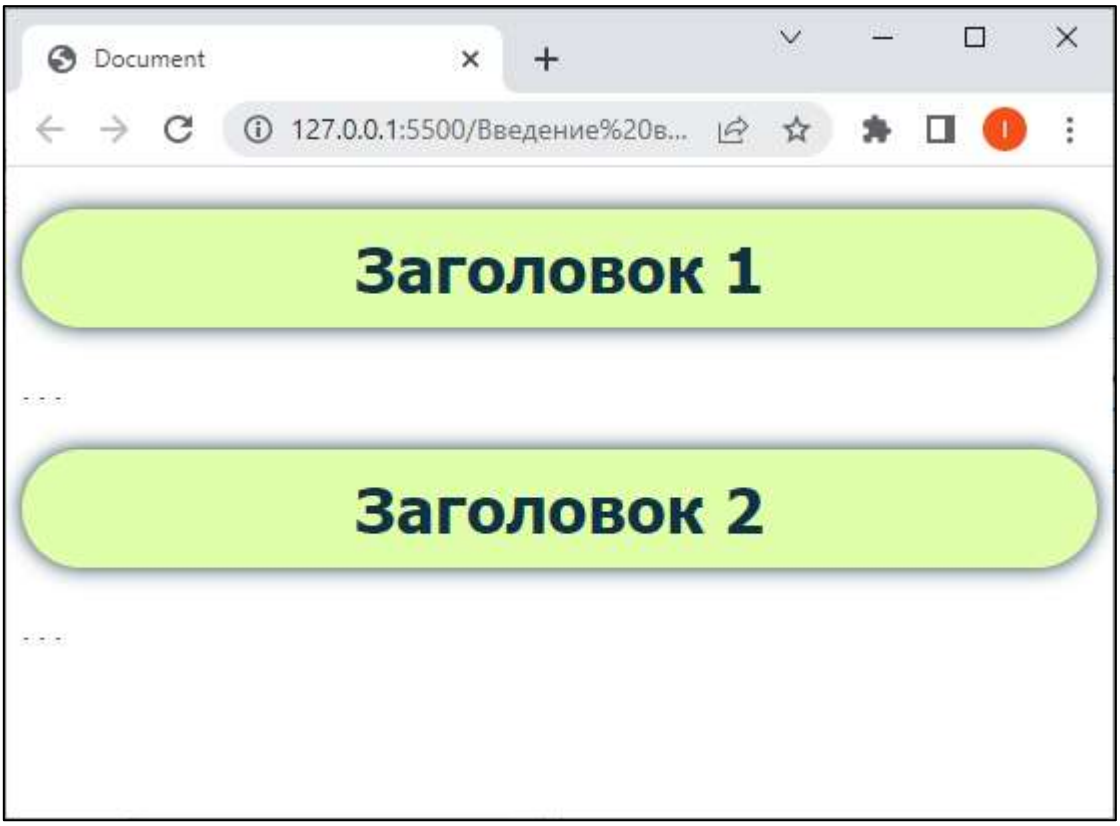


Рис. 3.1. Оформление заголовков с помощью CSS

В приведенном примере CSS используются для стилизации тега заголовка первого уровня `<h1></h1>`. Стиль этого элемента описывается с помощью специальных правил и отдельно от разметки. В данном примере – в теге-контейнере `<style></style>`, который расположен в преамбуле документа:

```
h1 {
    background: #E1FEAB;
    font: bold 200% Tahoma;
    color: #0F3244;
    text-align: center;
    padding: 10px;
    border-radius: 30px;
    box-shadow: 0 0 10px #0F3244;
}
```

Сам элемент в разметке не имеет каких-либо дополнительных атрибутов, нагромождающих HTML-код. Стиль привязывается к элементу в процессе отображения документа.

3.1.2. Селекторы

Селектор как основной элемент CSS

Определение

Селектор *– это некоторое имя CSS-стиля, для которого добавляются параметры форматирования. В качестве селектора могут выступают теги, атрибуты тегов, классы и идентификаторы.*

Далее в курсе рассматривается каждый тип селектора.

1. Теги-селекторы

Основной элемент, к которому можно привязать стиль – это *тег*. Стиль, заданный определенному тегу, распространяется на все теги в рамках документа.

Общий вид селектора тега:

```
тег { }
```

2. Классы и идентификаторы

На странице одинаковые элементы могут быть оформлены по-разному. Например, оформление гиперссылок в разных частях веб-страницы может быть различным.

Для разделения стилей элементов на отдельные компоненты используются *классы*.

Также каждый тег может иметь уникальный *идентификатор*, который способен принимать цеплять CSS-стили (хотя идентификаторы имеют более важную роль и обычно не связываются с CSS).

Общий вид селектора класса и идентификатора:

```
.класс { }
#идентификатор { }
```

3. Атрибуты в роли селекторов

Атрибуты тега тоже могут выступать в качестве селектора. Например, наличие атрибута или определенного его значения у тега может потребовать иного оформления элемента.

Общий вид селектора атрибута:

```
тег[атрибут] { }
```

Структура селектора и правила оформления кода

Если в HTML основным элементом является тег, то в CSS – это селектор. По существу, работа с CSS строится относительно правил описания селекторов и их взаимосвязи в одном документе.

Рис. 3.2. Структура селектора: основные компоненты

Рассмотрим компоненты селектора на примере селектора тега. Для остальных типов селекторов синтаксис аналогичен.

- *Селектор* тега описывается без угловых скобок и вне зависимости от того, является ли тег парным или унарным.
- Фигурные скобки отмечают начало и конец описания *тела селектора*. Внутри скобок задаются стили (рис. 3.3).
- *Свойства* описываются в формате «ключ: значение». В конце каждого обязательно ставится точка с запятой (рис. 3.4). Если в селекторе всего одно свойство, то часто его оформляют в одну строку (рис. 3.5).
- *Комментарии* в CSS пишутся внутри набора парных символов `/*  */` (это многострочный комментарий), рис. 3.6.

Фигурные скобки отмечают начало и конец описания тела селектора:

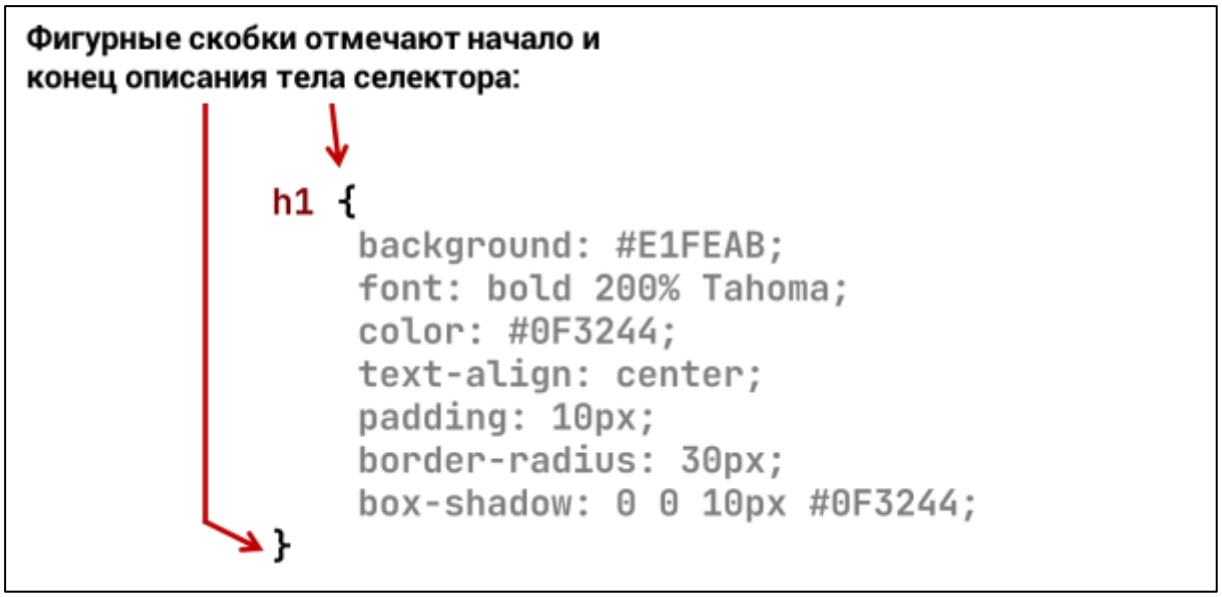


Рис. 3.3. Тело селектора

Свойства описываются в формате «ключ: значение».
В конце каждого обязательно ставится точка с запятой!

```
h1 {
    background: #E1FEAB;
    font: bold 200% Tahoma;
    color: #0F3244;
    text-align: center;
    padding: 10px;
    border-radius: 30px;
    box-shadow: 0 0 10px #0F3244;
}
```

Рис. 3.4. Свойства селектора

Стиль с одним свойством в общем виде:

```
h1 {
    font: bold 200% Tahoma;
}
```

Допускается оформление в одну строку:

```
h1 { font: bold 200% Tahoma; }
```

Рис. 3.5. Варианты оформления стилей с одним свойством

Комментарии в CSS пишутся внутри /* */ (это многострочный комментарий).

```
/*
   Описание стиля заголовков H1
   Автор: Якубович Д.А.
   Дата: 1.01.2019
*/
h1 {
    background: #E1FEAB;              /* цвет фона */
    font: bold 200% Tahoma;           /* шрифт */
    color: #0F3244;                   /* цвет текста */
    text-align: center;               /* выравнивание */
    padding: 10px;                    /* отступ */
    border-radius: 30px;              /* скругление углов */
    box-shadow: 0 0 10px #0F3244;     /* тень */
}
```

Рис. 3.6. Комментарии в CSS-коде

Это важно знать!

В отличие от HTML, CSS чувствителен к регистру букв. Все команды в CSS пишутся малыми буквами! На названия цветов это ограничение не распространяется.

Приемы работы с Visual Studio Code

Как и для HTML, Visual Studio Code выдает подсказки при описании CSS-стилей и позволяет их автоматически завершать.

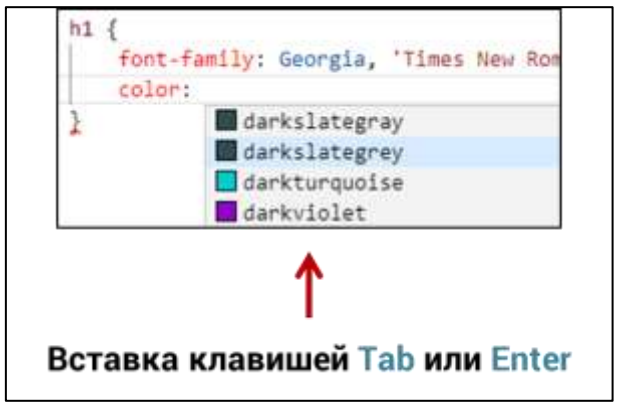


Рис. 3.7. Режим IntelliSense работает и для CSS

Приемы работы с Visual Studio Code

В редактор Visual Studio Code встроена палитра цветов. Для ее вызова наведите курсор на значение с цветом и в течении одной секунды палитра появится, как на рис. 3.8.
Палитра может отображать цвета в четырех моделях: RGB, HEX, HSL, HWB, а также поддерживает настройку прозрачной заливки (альфа-канал), рис. 3.9.

Рис. 3.8. Подсказка палитры цветов

Рис. 3.9. Смена цветовой модели и настройка прозрачности

Важное замечание!

При изучении примеров обращайтесь к справочнику CSS, если видите пока еще незнакомое свойство.
Кроме того, редактор Visual Studio Code при наведении курсора на свойство позволяет получить о нем справку на сайте MDN (Mozilla Developer Network).

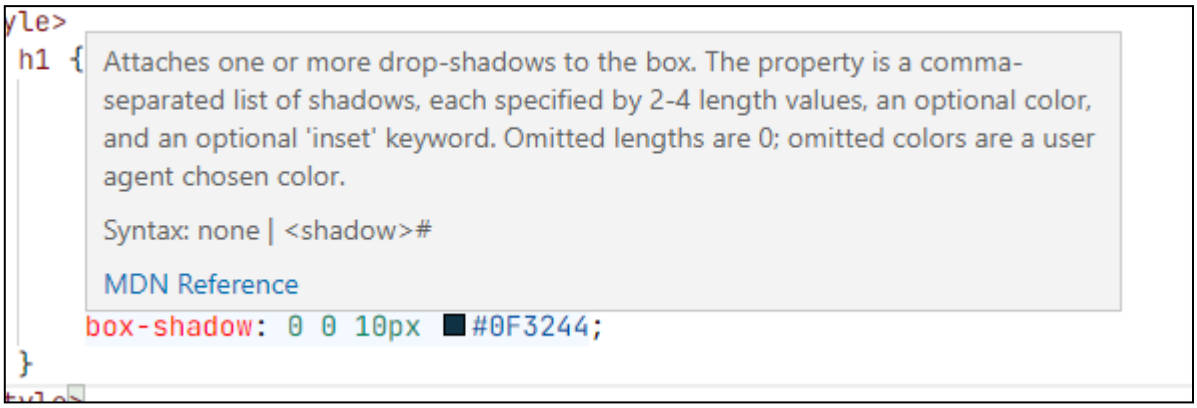


Рис. 3.10. Всплывающая подсказка по назначению и синтаксису CSS-свойств

Приемы работы с Visual Studio Code

Селекторы, не имеющие описания, редактор подчеркивает и помечает как неиспользуемые (рис. 3.11).

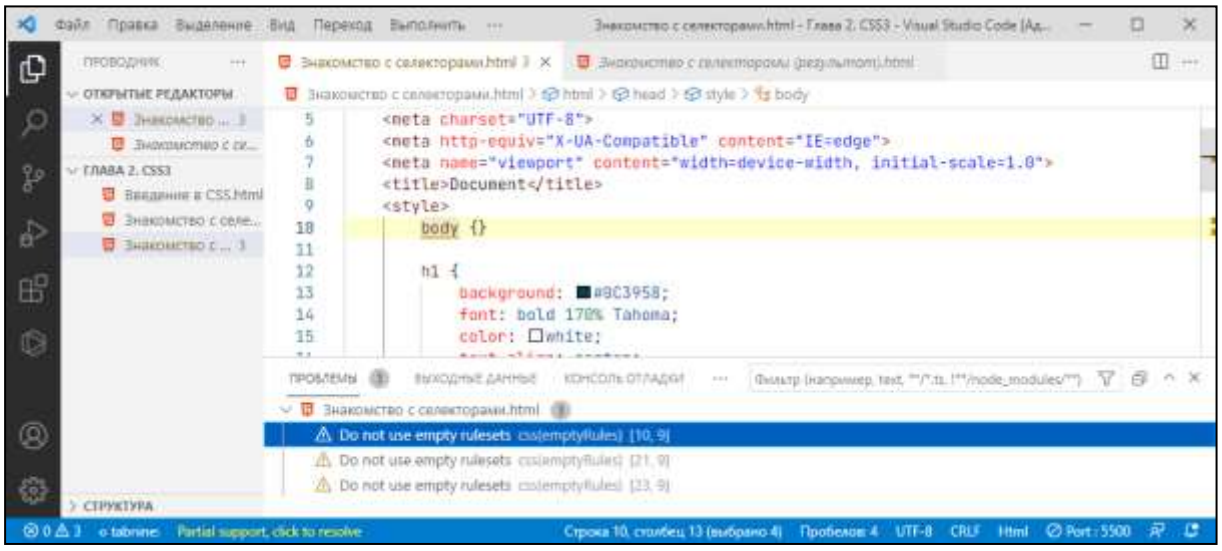


Рис. 3.11. Селекторы без свойств отмечаются редактором в списке предупреждений

3.1.3. Виды стилей

Способы подключения стилей

Работа с CSS-стилями может осуществляться несколькими путями, которые определяются способами подключения CSS к разметке или документу.

1. Встроенные стили

Встроенный стиль (также *inline-стиль*) подключается непосредственно к тегу и действует только на него. Такой стиль отличается малой гибкостью: если теги требуют одинакового оформления в разных частях разметки, то приходится многократного копировать и их стиль. А при необходимости редактирования таких стилей потребуется большой объем работы.

Для подключения внутреннего стиля к тегу используется глобальный атрибут **`style`**. В нем через «;» перечисляются требуемые свойства; фигурные скобки селектора не указываются.

Глава 3\Подключение стилей\1.Встроенные стили\index.html

```
<p>Обычный абзац</p>
<p style="font-size: 12pt; color: #339966">Стиль 1</p>
<p style="font: bold 14pt Verdana">Стиль 2</p>
<p style="font: bold 14pt Cambria; border: 2px dotted
#950000">Стиль 3</p>
```

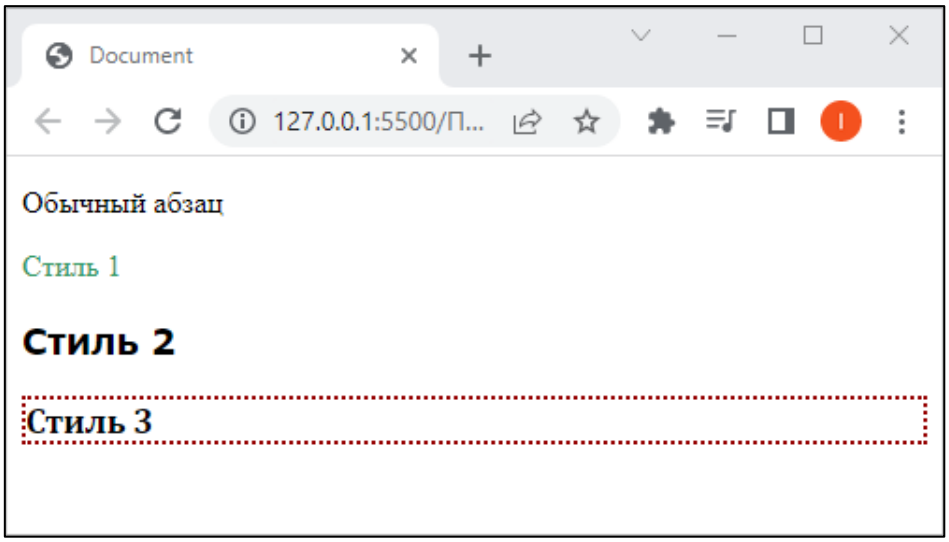


Рис. 3.12. Встроенные стили

Важное замечание!

Настоятельно рекомендуется отказаться от практики использования встроенных стилей! Они лишь приведены в ознакомительных целях.

2. Внутренние стили

При использовании внутренних стилей CSS-свойства описываются в разделе **<style></style>**. Этот контейнер может находиться в разных местах веб-документа и даже неоднократно повторяться. Однако его рекомендуется описывать в разделе <head></head>.

Внутренние стили, в отличие от встроенных, не загромождают код разметки и правятся в одном месте. Однако при необходимости распространить их на другие документы потребуется копировать блок <style></style> в каждый (аналогично и с их правкой).

Глава 3\Подключение стилей\2.Внутренние стили\index.html

```
<head>
    <!-- Другие элементы -->
    <style>
        body {
           background: #FFFFD9;
           font: 14px Tahoma;
           color: #1F1F43;
           margin: 10px 35px;
        }

        p {
           text-align: justify;
        }
   </style>
</head>

<body>
    <p>При использовании внутренних стилей свойства CSS
    описываются в самом документе и располагаются в
    заголовке веб-страницы.</p>
    <p>Для этого в раздел head помещается тег style со
    всеми стилями.</p>
</body>
```

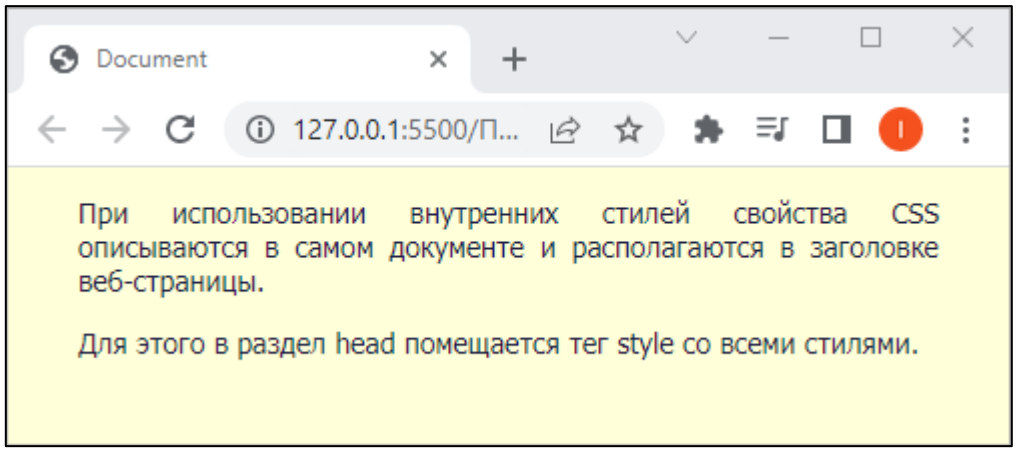


Рис. 3.13. Внутренние стили

3. Связанные стили

При использовании *связанных стилей* (*внешних стилей*) описание селекторов располагается в отдельном файле с расширением *.css* (его называют *таблицей стилей*, откуда и название CSS).

Для связывания документа с этим файлом применяется унарный тег **<link>**, который помещается в преамбулу `<head></head>`. Тегу `<link>` обязательно необходимо задать атрибут **href**, указывающий имя (путь) подключаемого стилевого файла. Кроме того, важно указать атрибуты **rel** (тип отношения – стилевой лист) и **type** (тип файла – CSS).

Например, следующий тег подключает стилевой файл *my_style.css*:

Рис. 3.14. Подключение связанного стиля

Заметим, что для HTML5 атрибут `type` уже необязателен (браузер без проблем распознает тип этого файла), достаточно следующей команды:

```
<link rel="stylesheet" href="my_style.css">
```

Приемы работы с Visual Studio Code

Связанные стили очень удобно использовать в таких редакторах, как Visual Studio Code. Здесь HTML-разметка и CSS-стили редактируются параллельно в отдельных колонках редактора.

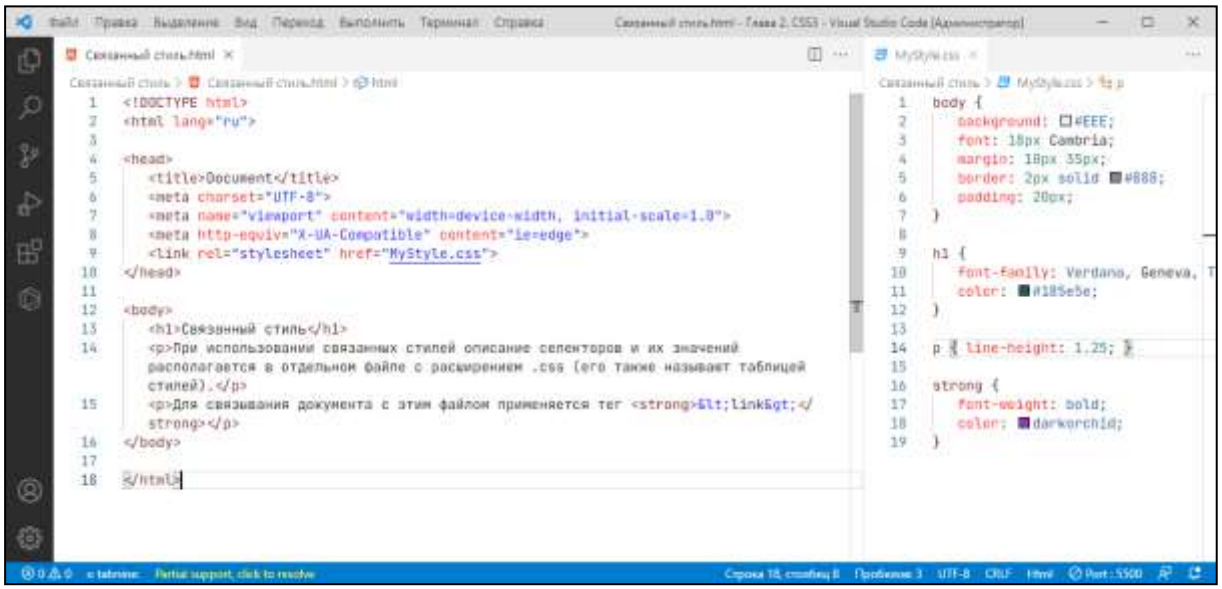


Рис. 3.15. Раздельное редактирование HTML и CSS

Покажем, как разбивать разметку и стили на отдельные файлы.

Шаг 1

В начале опишите основной HTML-документ, где тегом `<link>` подключите стилевой файл. В Visual Studio Code для быстрой вставки тега выбираем сниппет *link:css* и указываем название подключаемого файла с будущими CSS-стилями:

Рис. 3.16. Быстрая вставка тега подключения связанного стиля

Глава 3\Подключение стилей\3.Связанные стили\index.html

```
<!DOCTYPE html>
<html lang="ru">

<head>
    <meta charset="UTF-8">
    <meta http-equiv="X-UA-Compatible" content="IE=edge">
    <meta name="viewport" content="width=device-width,
initial-scale=1.0">
    <title>Document</title>
    <link rel="stylesheet" href="style.css">
</head>

<body>
    <h1>Связанный стиль</h1>
    <p>При использовании связанных стилей описание
    селекторов и их значений располагается в отдельном
    файле с расширением .css (его также называют
    таблицей стилей).</p>
    <p>Для связывания документа с этим файлом применяется
    тег <strong>&lt;link&gt;</strong></p>
</body>

</html>
```

Шаг 2

Нажмите в правом верхнем углу на кнопку  для разделения редактора на колонки. Их ширину можно менять.

- Создайте в правой колонке новый файл (*Файл / Новый файл*). Поменяйте подсветку синтаксиса на CSS (рис. 3.17), сохраните файл.

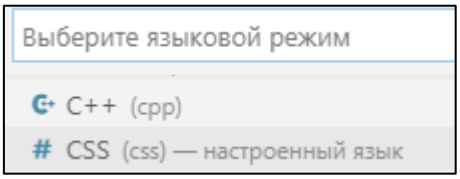


Рис. 3.17. Выбор синтаксиса

- Либо щелкните *ЛКМ* по названию файла *my_style.css* в теге `<link>` и редактор сам предложит создать несуществующий файл (рис. 3.18-рис. 3.19).

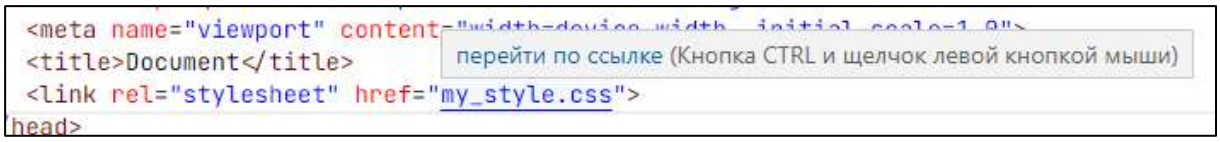


Рис. 3.18. Создание нового файла автоматически по названию и типу

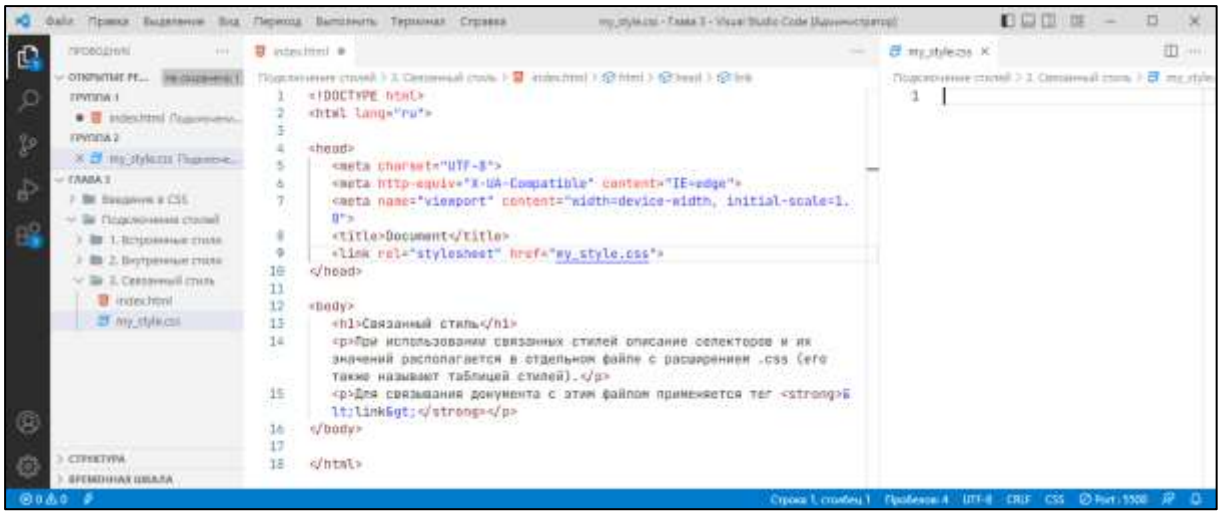


Рис. 3.19. Разделение редактора на колонки

Шаг 3

Введите в CSS-файл описание стилей. Никаких посторонних символов (тегов) здесь быть не должно:

Глава 3\Подключение стилей\3.Связанные стили\my_style.html

```
body {
    background: #EEE;
    font: 18px Cambria;
    margin: 10px 35px;
    border: 2px solid #888;
    padding: 20px;
}

h1 {
    font-family: Verdana, Geneva, Tahoma, sans-serif;
    color: #185e5e;
}

p {
    line-height: 1.25;
}
```

```
strong {
    font-weight: bold;
    color: darkorchid;
}
```

Шаг 4

Сохраните файл под названием *my_style.css* (если создавали его самостоятельно).

Не забудьте также сохранить и HTML-файл.

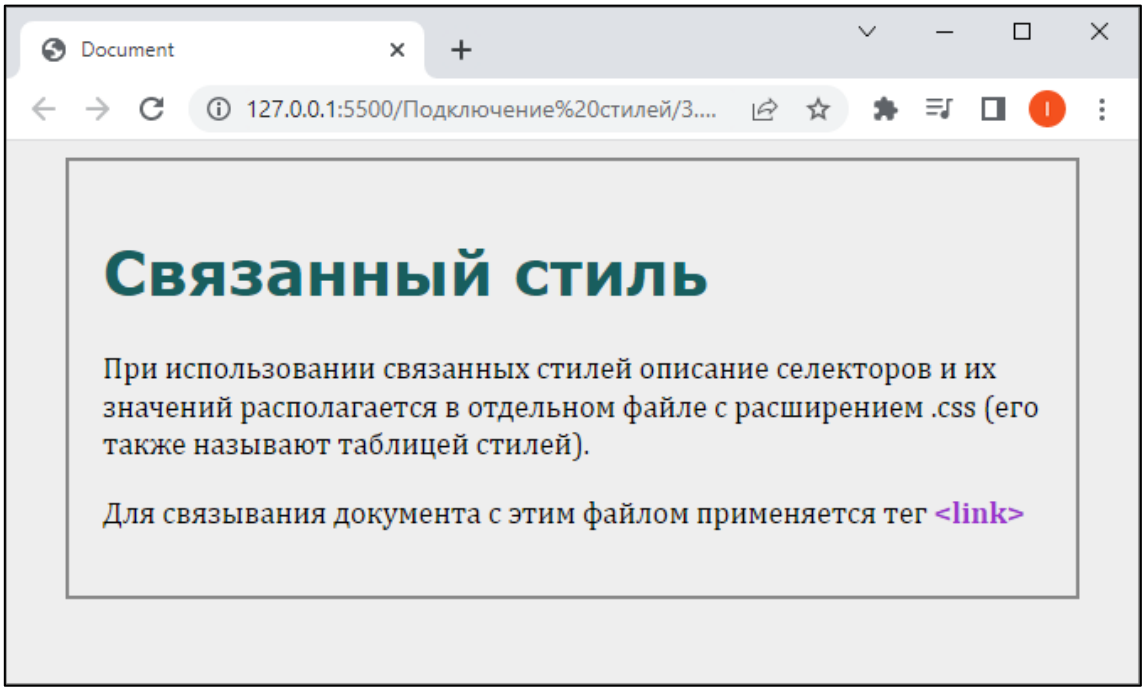


Рис. 3.20. Связанный стиль

На рис. 3.21 кратко обозначена схема способов подключения CSS-стилей.

Это важно знать!

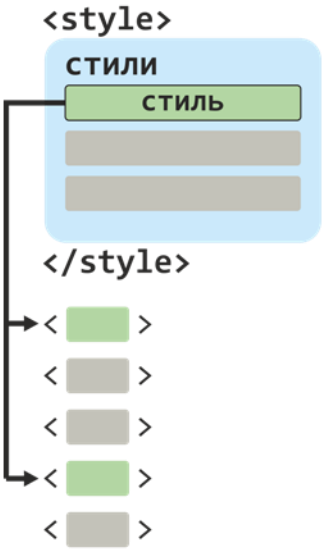
Если после подключения связанного стиля разметка не стилизуются, проверьте корректность названия файла, а также относительный путь к нему в атрибуте ***href*** *(если он расположен во внутренних подкаталогах проекта).*

1. Встроенный

Подключается непосредственно к тегу с помощью глобального атрибута `style`.

2. Внутренний

Описывается внутри парного тега `<style></style>`.



3. Связанный

Описание стиля выносится в отельный файл и подключается в основном документе тегом `<link>`.

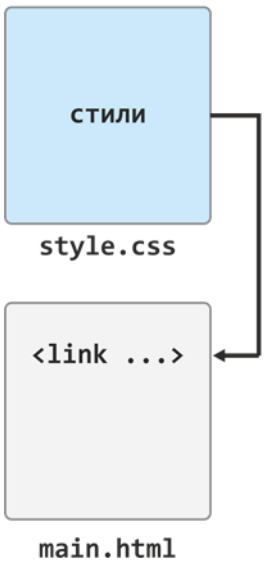


Рис. 3.21. Способы подключения стилей (кратко)

Важное замечание!

Используйте связанные стили.

1. *С ними удобно работать в редакторе.*
2. *Вы можете подключить к документу несколько стилевых файлов одновременно!*
3. *С другой стороны, разные HTML-файлы могут раздельно использовать один стилевой.*

Вопросы для самопроверки

1. Для каких целей используются CSS и почему они имеют больше возможностей в оформлении элементов, чем HTML?
2. Перечислите основные достоинства использования каскадных таблиц стилей.
3. Что нового было добавлено в спецификацию CSS3?
4. Какие типы селекторов поддерживает CSS?
5. Опишите структуру селектора.
6. Перечислите основные способы подключения CSS-стилей и их характерные особенности.
7. Почему встроенные стили использовать не рекомендуется?
8. Чем внутренние стили эффективнее встроенных?
9. Почему связанные стили считаются наиболее универсальными и в чем достоинства их использования?

Практикум

Задание 1

1. Создайте каталог *Знакомство с селекторами*. Вставьте в HTML-файл следующий ниже этого задания код. Изучите его разметку и описанные селекторы.
2. В справочнике найдите описание и примеры использования следующих CSS-свойств: `background-color`, `color`, `text-align`, `border-bottom`, `font`, `font-size`.

3. Установите описанные в комментариях свойства селекторам h1, h2, p, a, strong.
4. Поменяйте некоторые значения свойств для селектора h1 (произвольно).
5. Примерный результат изображен на рис. 3.23.

Глава 3\Знакомство с селекторами\index.html

```
<!DOCTYPE html>
<html lang="ru">

<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width,
initial-scale=1.0">
    <meta http-equiv="X-UA-Compatible" content="ie=edge">
    <title>Document</title>
    <style>
        body {
            font-family: Georgia, 'Times New Roman',
                Times, serif;
            /*
                С помощью background-color задать цвет
                фона.
                С помощью font-size размер шрифта в 18px
            */
        }

        h1 {
            background: #188870;
            font: bold 170% Tahoma;
            color: white;
            text-align: center;
            padding: 15px;
            border-radius: 10px;
        }

        h2 {
            /*
                С помощью font задать свойства шрифта:
                жирный, размер 130%, шрифт - Arial
                С помощью color и border-bottom задать
                цвет текста и границы
            */
        }
```

```
        p {
            /*
                С помощью text-align выровнять по ширине
            */
        }

        a {
            /*
                Задать алый цвет текста
            */
            color: #c50e64;
        }

        strong {
            /*
                Задать синий цвет текста
            */
        }
    </style>
</head>

<body>
    <h1>Технология CSS</h1>
    <h2>Понятие</h2>
    <p><strong>Стилем</strong> или <strong>CSS</strong>
    называется набор параметров форматирования, который
    применяется к элементам HTML-документа для изменения
    их внешнего вида.</p>

    <h2>Возможности</h2>
    <ul>
        <li>Разграничение кода и оформления.</li>
        <li>Разное оформление для разных устройств.</li>
        <li>Расширенные по сравнению с HTML способы
            оформления элементов.</li>
        <li>Ускорение загрузки сайта.</li>
        <li>Гибкость использования</li>
    </ul>
    <p>Подробнее о свойствах можно почитать
    <a href="https://html5book.ru/css-
css3/">здесь</a>.</p>
</body>

</html>
```

Задание 2

1. Создайте каталог с названием *Про JavaScript* и новый HTML-файл с разметкой, указанной ниже. Также поместите в этот каталог изображение с логотипом HTML5.
2. Переместите описание стилей из тега `<style>` в отдельный CSS-файл и подключите его тегом `<link>`.
3. С помощью CSS задайте стили для тегов `<body>`, `<h1>`, `<p>`, `<a>`, чтобы получить в итоге оформление, как на рис. 3.24. Используйте подсказки, указанные в комментариях к селекторам.
4. В результате в каталоге будет 3 файла:

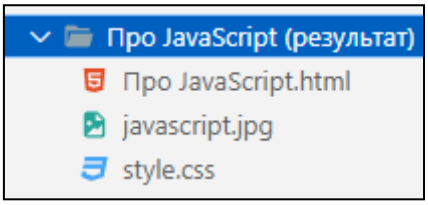


Рис. 3.22. Разделение разметки и стилей в задании

Глава 3\Про JavaScript\index.html

```
<!DOCTYPE html>
<html lang="ru">

<head>
    <meta charset="UTF-8">
    <meta http-equiv="X-UA-Compatible" content="IE=edge">
    <meta name="viewport" content="width=device-width,
initial-scale=1.0">
    <title>Document</title>
    <style>
        body {
            /* Шрифт Tahoma 16px, цвет - темно-серый,
               фон бледно-оранжевый */
        }

        h1 {
            /* Темно-фиолетовый фон, цвет текста - белый,
               выравнивание по центру, отступ внутри
               (см. свойство padding) - 10px */
        }

        p {
```

```
            /* Междустрочный интервал - 1.4
            (см. свойство line-height) */
        }

        a {
            /* Жирное начертание, цвет - зеленый */
        }

/* Изображение делаем как блок (т.е. по всей ширине),
выравниваем по центру с помощью авто отступа слева и
справа, ширина относительно ширины окна = 25%, но не
менее 100px и не более 240px,сплошная рамка толщиной 3px
*/
        img {
            display: block;
            margin: 10px auto;
            width: 25%;
            min-width: 100px;
            max-width: 240px;
            border: 3px solid #444444;
        }
    </style>
</head>

<body>
    <h1>JavaScript</h1>
    <p><strong>JavaScript</strong> - прототипно-
    ориентированный сценарный язык программирования.
    Является реализацией языка ECMAScript.</p>
    <p>Наиболее широкое применение язык находит в
    браузерах как язык сценариев для придания
    интерактивности веб-страницам.</p>
    <img src="javascript.jpg" alt="картинка">
    <p>Узнать подробнее можно здесь:
        <a href="https://learn.javascript.ru/">
            learn.javascript.ru
        </a>.
    </p>
</body>

</html>
```

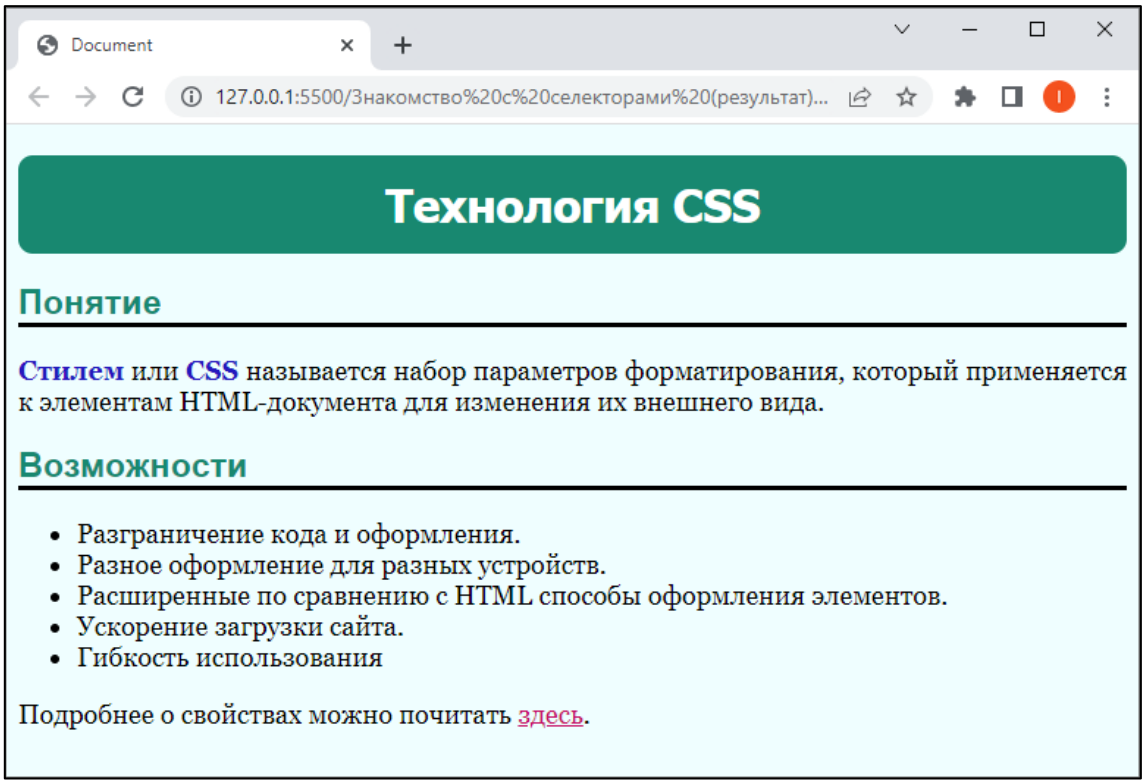


Рис. 3.23. Задание 1: образец выполнения

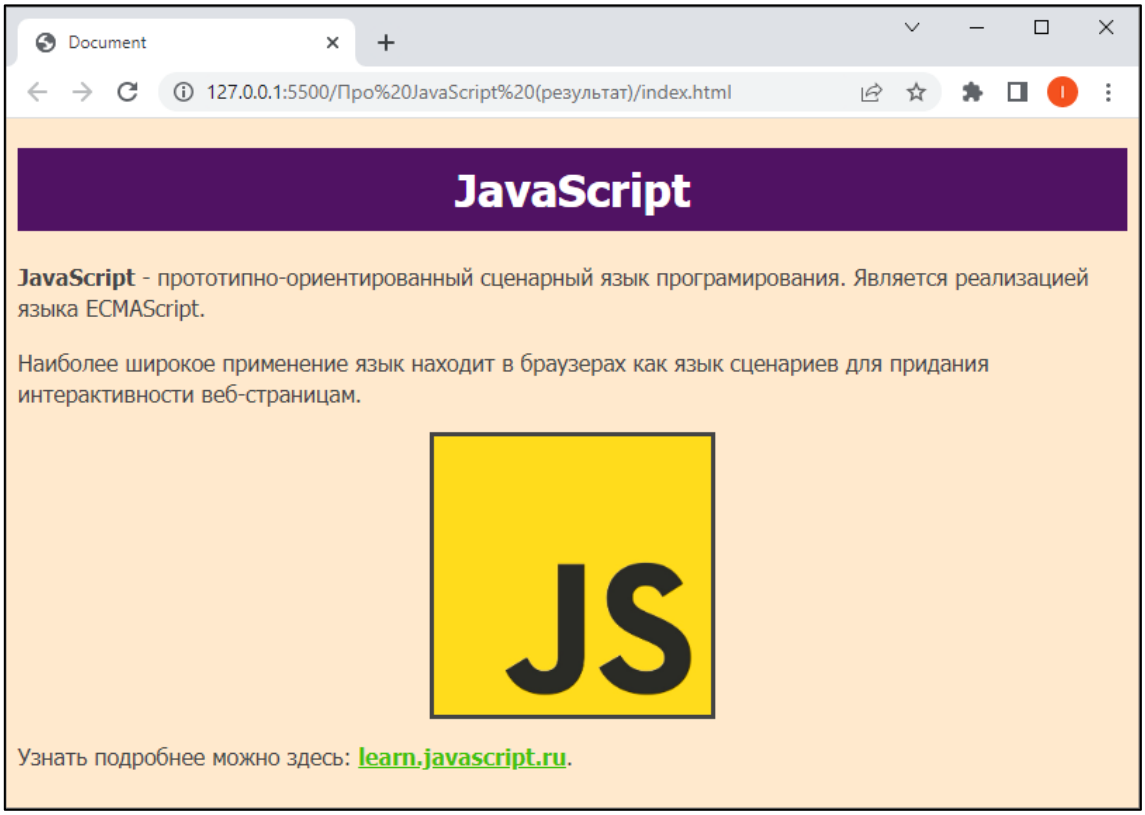


Рис. 3.24. Задание 2: образец выполнения

3.2. Классы и идентификаторы

3.2.1. Разграничение стилей элементов

Необходимость разделения стилей

Вне зависимости от способа подключения CSS-стилей селекторы тегов ограничивают возможность форматирования элементов, которые эти теги размечают. Так из примеров и задач параграфа 3.1 следует, что каждый селектор оформляет содержимое своего тега, поэтому элементы принимают одинаковое форматирование.

Однако на практике структура даже простых веб-страниц требует более сложного интерфейса. При этом одни и те же элементы в разных местах страницы должны быть оформлены по-разному.

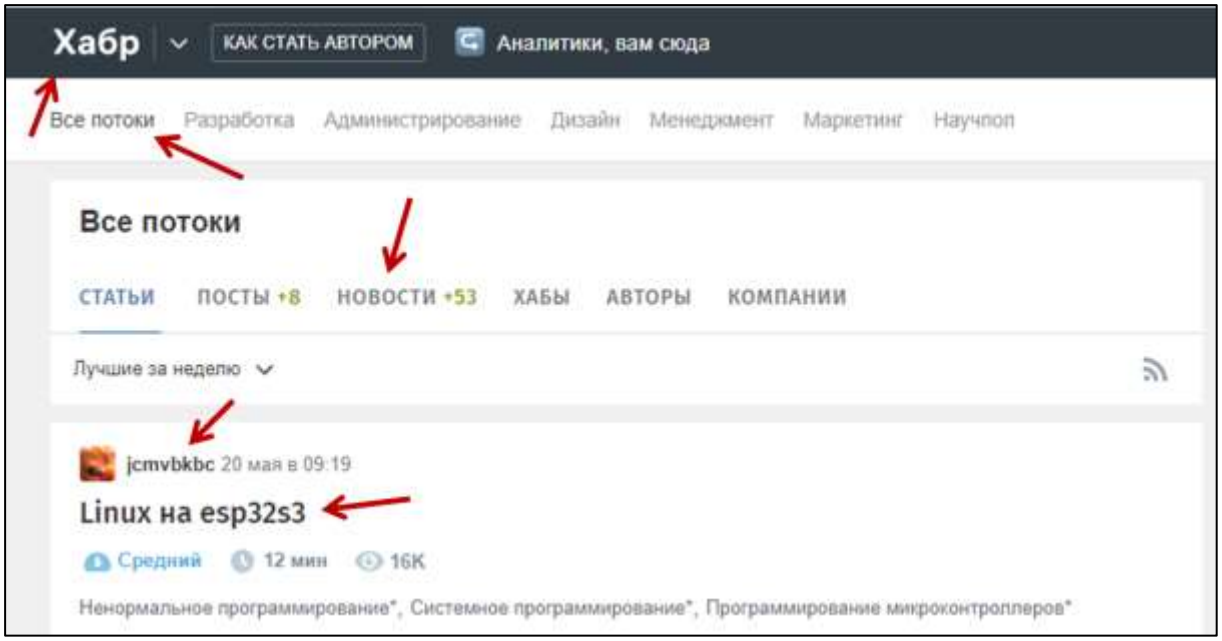


Рис. 3.25. Гиперссылки расположены в разный блоках веб-страницы и имеют разное форматирование

Для решения подобного вопроса необходимо обеспечить возможность одинаковым тегам иметь собственный стиль оформления.

Классовый подход

Чтобы обеспечить гибкий подход к оформлению элементов и получить возможность оформлять один и тот же тег по-разному любое число раз, в CSS используются *селекторы классов*. Каждый тег может подключать атрибут `class`, который реализует собственный стиль форматирования.

Классы являются ключевым компонентом в формировании разветвленной модели стилизации элементов. Именно классам задаются все необходимые свойства элементов.

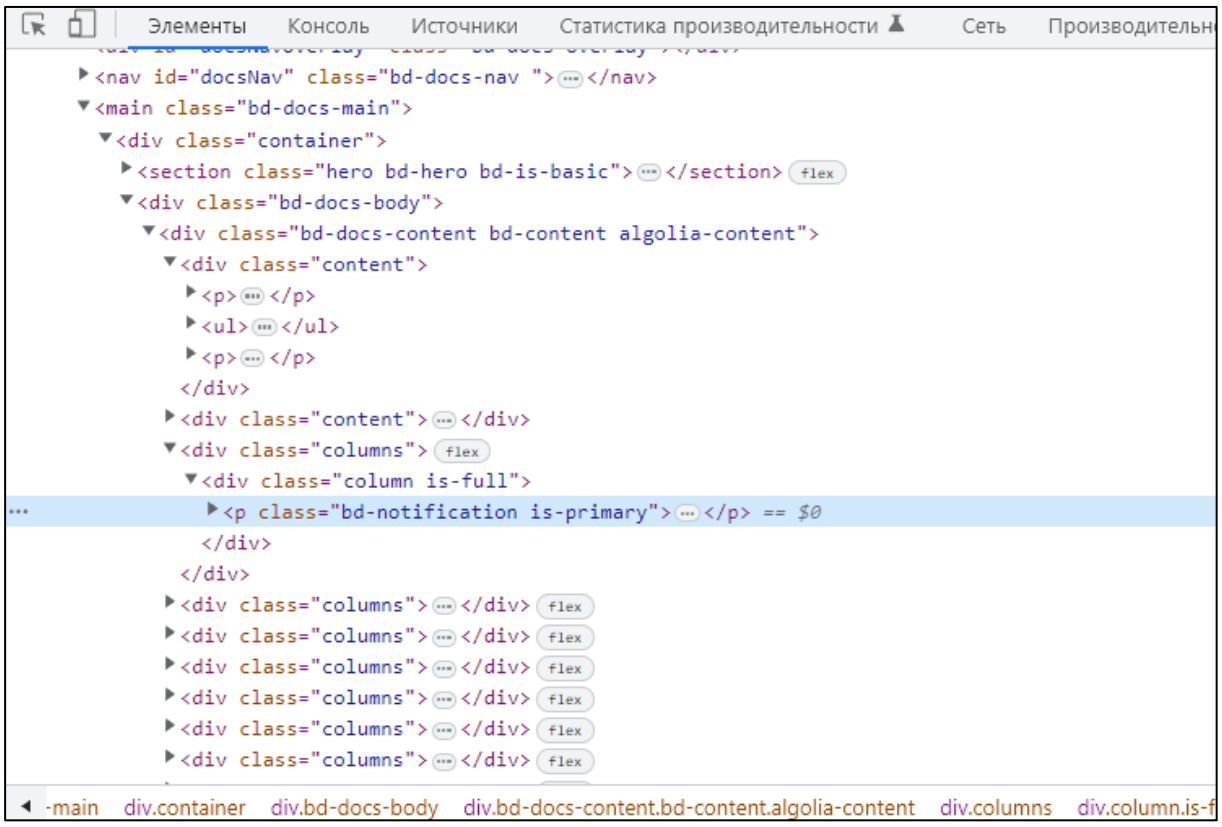


Рис. 3.26. Фрагмент разметки веб-сайта: почти все теги подключают классы, которые форматируют элементы

С другой стороны, селекторам тегов следует задавать минимально необходимый набор базовых свойств форматирования. Это позволяет классам получить полный контроль над оформлением элементов, а также избежать их нежелательной стилизации при наследовании свойств. Разработчику необязательно «сбрасывать» стили селекторов тегов самостоятельно, для этого можно воспользоваться файлами нормализации, например – Normalize.css.

3.2.2. Селекторы классов

Классы

Определение

Класс *– стиль(и) оформления, который может быть привязан к любому тегу, если это имеет смысл.*

Важные особенности классов:

- основное предназначение классов состоит в том, чтобы задать разное оформление для одинаковых тегов;
- с другой стороны, один и тот же класс можно подключать к разным тегам;
- классы могут использоваться многократно.

Определение

Селектор класса задается следующим образом:

```
.имя_класса {
    /* свойства стиля */
}
```

Для подключения класса к тегу доступен глобальный атрибут ***`class`****, в котором указывается имя подключаемого класса.*

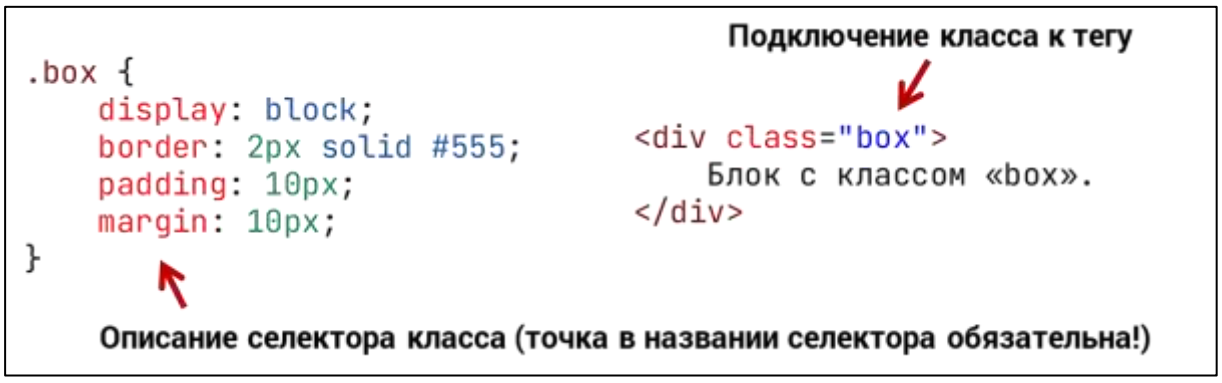


Рис. 3.27. Описание и подключение класса

Рассмотрим следующий пример (рис. 3.27). Опишем CSS-класс *box*, который делает элемент блоком, задает границу, а также небольшие отступы полей и внутри блока. Этот класс с помощью глобального атрибута `class` цепляем к тегу-контейнеру `<div>`.

Несмотря на то, что сам тег `<div>` является абстрактным контейнером и не имеет каких либо визуальных настроек, класс *box* наделяет блок свойствами форматирования.

При этом класс *box* можно подключать и к другим тегам разметки: они примут указанные свойства.

Виды классов

Практический пример

Классы можно подключать как *автономные*, т.е. описанные без привязки к какому-либо тегу, а также *связанными* только с конкретными тегами.

Рассмотрим оба подвида на следующем примере.

Глава 3\Классы\Подключение классов\index.html

```
<h1>Обычный заголовок</h1>
<p>Обычный абзац</p>
<p class="cite">Абзац с классом cite</p>
<p class="box">Абзац с классом box</p>

<h1 class="marker">Заголовок с классом marker</h1>
<p class="marker">Абзац с классом marker</p>
```

Глава 3\Классы\Подключение классов\style.css

```
h1 {
    text-align: center;
    color: #832E0E;
}

p { color: #337; }

p.cite { font-style: italic; }

p.box {
    font: bold 1.3em Arial;
    color: #6e113d;
    padding: 10px;
```

```
    border: 2px solid black;
}

.marker {
    background: yellow;
    border: 1px dotted #FF8CC6;
}
```

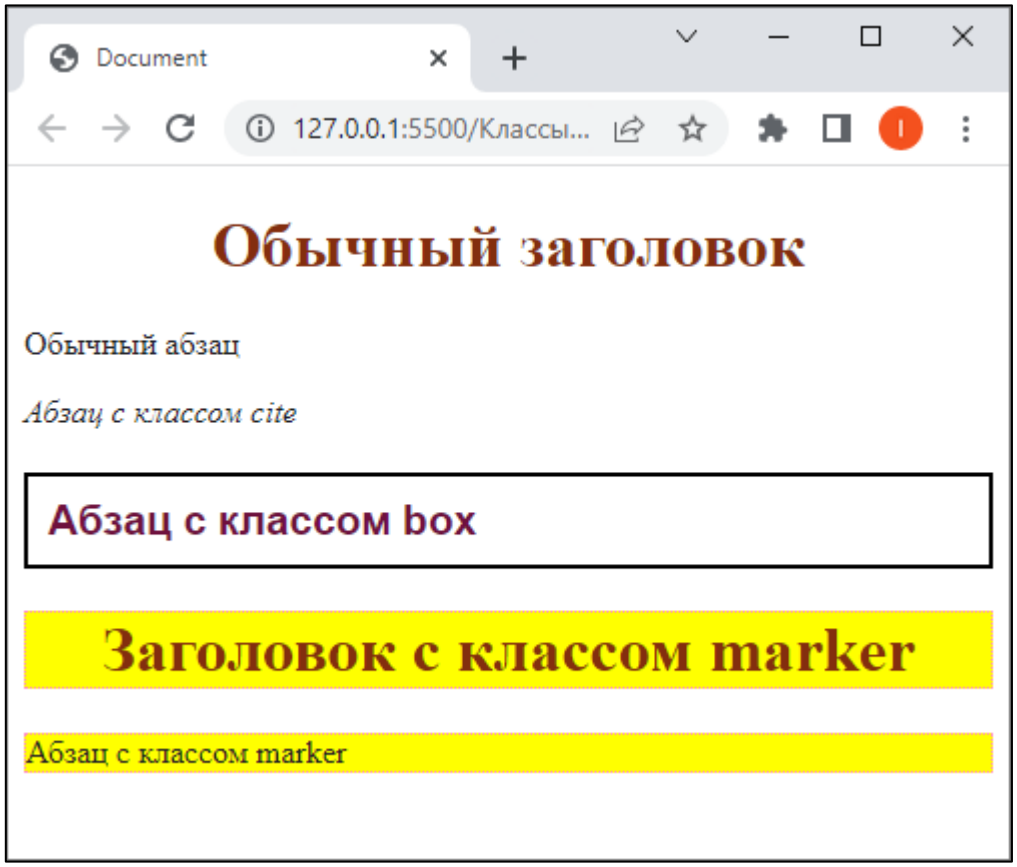


Рис. 3.28. Пример подключения классов

1. Селекторы классов с привязкой к тегу

В примере заданы классы *cite* и *box*, которые привязаны к тегу абзаца <p>:

```
p.cite { font-style: italic; }

p.box {
    font: bold 1.3em Arial;
    color: #6e113d;
    padding: 10px;
    border: 2px solid black;
}
```

Подобная запись селектора означает, что в случае подключения к тегу этих классов сработают описанные стили. При этом классы *cite* и *box* будут доступны только тегу абзаца <p>. Т.е. в текущей разметке только второй и третий абзацы примут указанное оформление:

```
<p class="cite">Абзац с классом cite</p>
<p class="box">Абзац с классом box</p>
```

Однако классы с таким же названием можно привязать и к другим тегам: их свойства не будут взаимосвязаны! (Впрочем такая практика дублирования имен классов нежелательна)

Важное замечание!

Важно заметить, что классы тега <p> наследуют свойства самого тега. Если же в классе есть свойство, которое присутствует и в описании тега, то его значение перезаписывается классом!

Приемы работы с Visual Studio Code

При наведении курсора на селектор класса можно увидеть, к какому тегу он привязан.

Рис. 3.29. Редактор подсказывает, как выглядит обращение к селектору класса

Это полезно знать!

Классы, привязанные к тегу, следует описывать рядом и последовательно.

2. *Селекторы автономных классов*

Кроме того, в примере описан автономный класс `marker`, который не связан с каким-либо тегом:

```
.marker {
    background: yellow;
    border: 1px dotted #FF8CC6;
}
```

Поскольку этому селектору класса не задан тег, то его можно подключить к любому тегу, если это имеет смысл и не нарушит уже подключенные стили элемента.

Как и в случае с привязанными к тегу стилями, подключение автономного класса осуществляется с помощью атрибута `class`:

```
<h1 class="marker">Заголовок с классом marker</h1>
<p class="marker">Абзац с классом marker</p>
```

Здесь второй заголовок `<h1>` и четвертый абзац `<p>` принимают надлежащее форматирование, определенное в классе `marker`. Кроме того, из рис. 3.28 видно, что базовые параметры абзаца и заголовка также наследуются.

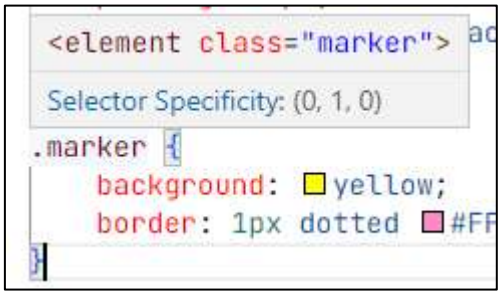


Рис. 3.30. Редактор подсказывает, что класс может использовать любой элемент в качестве тега

Это важно знать!

Важно отметить, что у связанного с тегом селектора класса приоритет стилей выше, чем у автономного селектора класса. Это связано с определенными правилами ранжирования селекторов CSS, которые мы рассмотрим в следующих темах.

Это полезно знать!

Обычно селекторы класса не следует привязывать к определенному тегу. Предполагается, что все классы имеют уникальные (разные) имена.
С другой стороны любой класс создается для форматирования определенных элементов, поэтому он должен описывать необходимые для этого свойства.
В примерах далее мы будем описывать автономные селекторы классов. Это нисколько не ограничивает возможностей форматирования и также частично упрощает их описание.

Несколько классов

К тегу можно подключать несколько классов. Для этого в атрибуте `class` имена классов перечисляются через пробел (рис. 3.31).

Глава 3\Классы\Несколько классов\index.html

```
<cite>Обычная цитата</cite>
<cite class="nice">Цитата с классом nice</cite>
<cite class="box">Цитата с классом box</cite>
<cite class="nice box">Цитата с классом nice и box</cite>
```

Глава 3\Классы\Несколько классов\style.css

```
/* Растягивает цитаты по всей ширине родителя */
cite { display: block; }

.nice {
    font: italic bold 120% Georgia;
    color: #400040;
}

.box {
    background: #FCE38F;
    padding: 15px 20px;
    border-left: 10px solid #E8743C;
    margin: 10px 0;
}
```

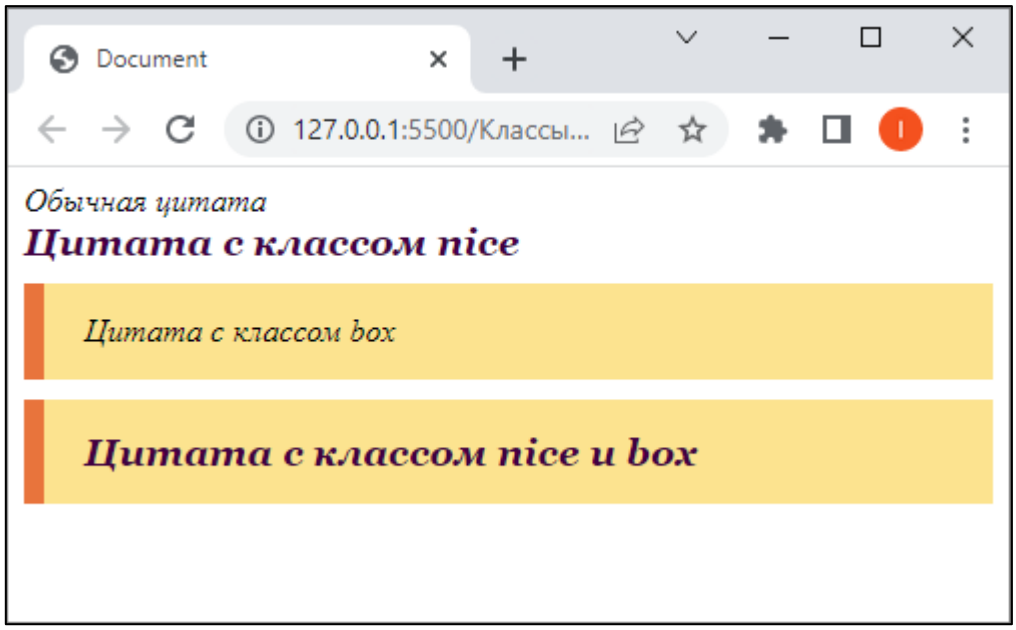


Рис. 3.31. Подключение нескольких классов

В приведенном примере описаны два селектора класса. Поскольку оба автономны, то их можно привязать к любым тегам разметки текста. В данном случае – к последней цитате `<cite>`:

```
<cite class="nice box">Цитата с классом nice и box</cite>
```

Нетрудно также заметить, что здесь наследуются как базовые настройки блока цитат, так и каждого из классов.

Это полезно знать!

Порядок перечисления классов в атрибуте `class` непринципиален. Обычно здесь используется соглашение, что первостепенно указывают классы, влияющие на важнейшие свойства элемента, а далее – отвечающие за его оформление.

3.2.3. Селекторы идентификаторов

Идентификаторы элемента

Определение

Идентификатор *– уникальное название HTML-элемента. Селектор идентификатора задается следующим образом:*

```
#имя_идентификатора {
    /* свойства стиля */
}
```

Для подключения идентификатора к тегу доступен глобальный атрибут **`id`***, в котором указывается имя подключаемого идентификатора.*

К селектору идентификатора (или далее также ID-селектору), как и класса, также можно привязать CSS-стили. Однако идентификаторы обычно используются JavaScript для обращения к элементу разметки с целью последующего манипулирования его содержимым или оформлением.

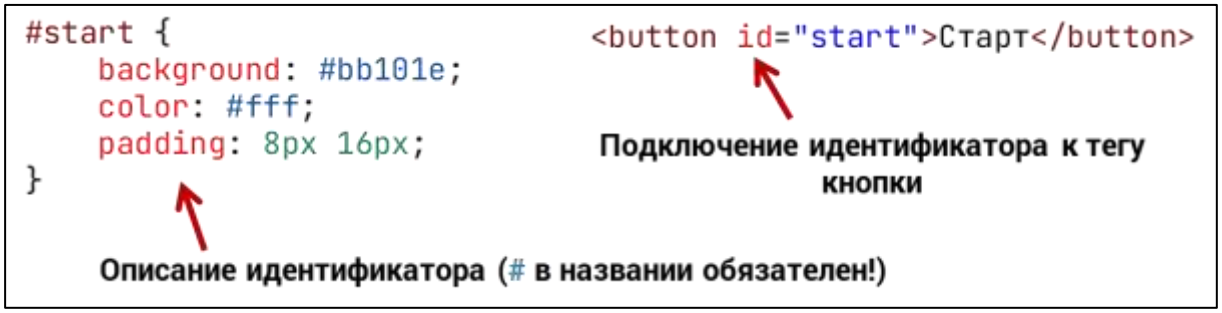


Рис. 3.32. Описание CSS-свойств идентификатора и его подключение к тегу

От классов идентификатор отличает следующее:

- идентификатор может быть подключен только один раз и не может дублироваться в разметке;
- идентификаторы имеют более высокий приоритет стиля (*специфичность*).

Приведем пример использования идентификатора для форматирования блока с текстом.

Глава 3\Идентификаторы\index.html

```
<h1>Обычный заголовок</h1>
<div id="content">
    <h1>Идентификаторы</h1>
    <p>Идентификатор также может быть привязан к тегу,
    либо быть автономным. Чтобы применить идентификатор,
    тегам доступен глобальный атрибут id.</p>
</div>
```

Глава 3\Идентификаторы\style.css

```
body {
    background: #D0D0D0;
    font: 14px/1.4 Georgia, serif;
}

#content {
    background: #E0E0E0;
    width: 400px;
    /* прием с полями, выравнивает блок по центру */
    margin: 0 auto;
    padding: 20px;
}
```

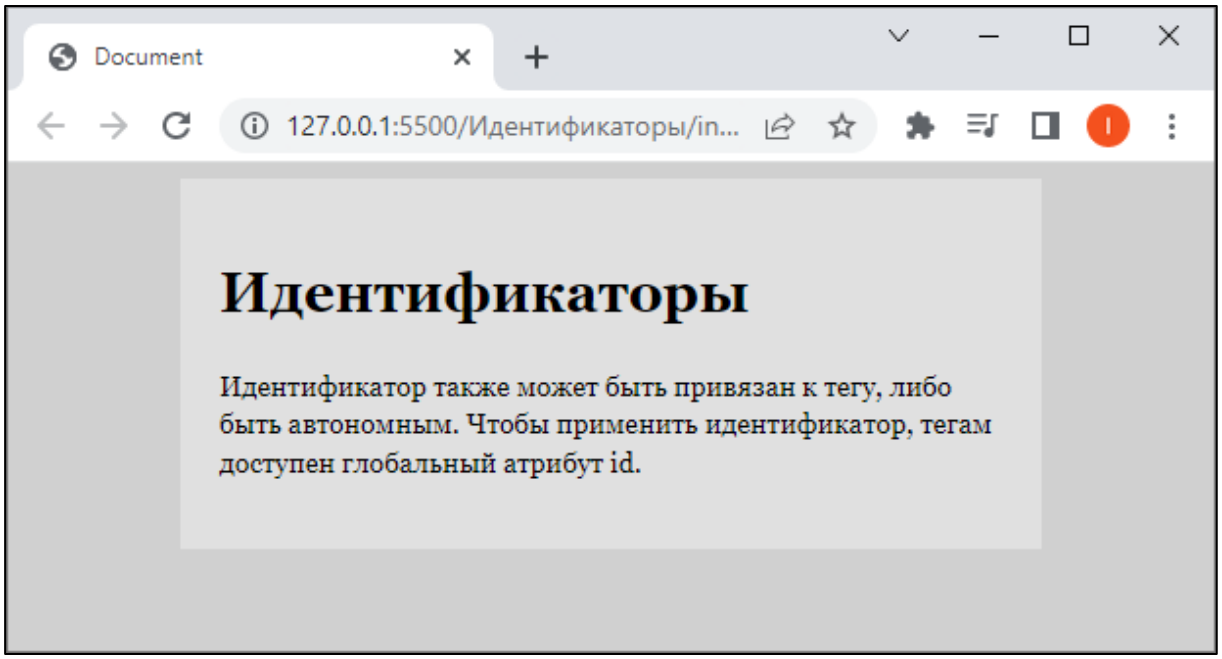


Рис. 3.33. Пример использования идентификатора для оформления блока

Описанный в примере селектор идентификатора `content` подключается к блоку `<div>` и задает необходимую стилистику.

Впрочем, более рациональным будет использовать для оформления именно класс. Тогда, если важно сохранить идентификатор, блок можно задать, например, следующим образом:

```
<div id="content" class="content-box">
    ...
</div>
```

Теперь описание стилей привязывается к селектору класса `content-box`, а идентификатор остается для управления JavaScript.

Это полезно знать!

На самом деле и идентификатору, и классу тега можно задать одинаковое имя:

```
<div id="content" class="content">
    ...
</div>
```

Однако важно помнить, что класс может использоваться многократно, а идентификатор – только один раз.
Хорошей практикой является не дублировать названия классов и идентификаторов.

Роль идентификаторов

Как было отмечено выше, формально к идентификатору тоже допускается цеплять CSS-стиль оформления. Однако в современной веб-разработке от привязки стилей к ID-селекторам практически отказались.

Идентификатор чаще всего задается тегу, чтобы к нему можно было обратиться в скриптах JavaScript. Даже если стиль используется в единственном экземпляре, то его также следует оформить в форме класса.

Следует руководствоваться простым правилом:

- классы – для оформления элементов;
- идентификаторы – для обращения в JavaScript к элементу.

Приемы работы с Visual Studio Code

В Visual Studio Code для быстрого набора тега с классом используйте возможности плагина Emmet.
Например, аббревиатура

```
main.box
```

после нажатия ***Tab*** *преобразуется в*

```
<main class="box"></main>
```

По аналогии для идентификатора:

```
h1#title
```

преобразуется в

```
<h1 id="title"></h1>
```

Наконец, можно комбинировать оба варианта (рис. 3.34).

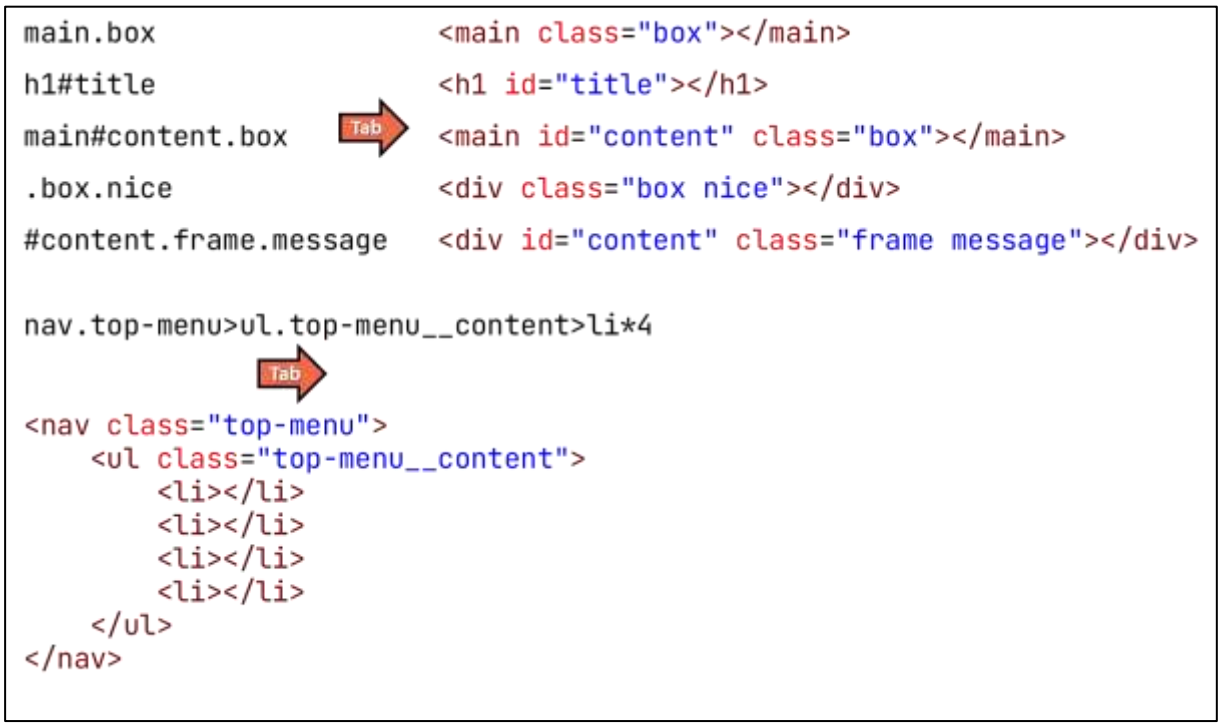


Рис. 3.34. Аббревиатуры Emmet существенно упрощают верстку больших блоков HTML-кода

3.2.4. Некоторые замечания по работе с классами

Оптимизация разметки и стилей

Концепция использования классов предполагает стремление к тому, что описание стилей должно быть сосредоточено именно в селекторах классов. Каждый элемент разметки тела документа получает один или несколько классов, которые вносят свою группу стилей в оформление элемента.

С другой стороны, верстальщику веб-страниц важно не нагромождать CSS-код многочисленными селекторами классов. В идеале необходимо обеспечить такую разметку, которая может быть в дальнейшем расширена без существенной переработки структуры HTML. А при необходимости масштабирования проекта расширяются списки классов оформления.

При этом селекторы тегов имеют как можно меньший набор стандартных стилей, которые без каких-либо конфликтов переопределяются подключаемыми пользовательскими классами.

Решение обозначенной задачи оптимизации кода не всегда тривиально и не имеет универсальный алгоритм действий. Во многом это зависит от выбранной методологии верстки (например, БЭМ, см. пункт 1.8.2) или корпоративной этики, выработанной в команде.

Пример

Рассмотрим следующий пример. Предположим есть простая разметка основных блоков шапки веб-страницы, с логотипом, заголовком и блоком навигации:

```
<header>
    <img src="" alt="">
    <h1></h1>
    <nav class="header__nav"></nav>
</header>
```

Чтобы для такой разметки обеспечить корректную стилизацию элементов в блоке `<header>`, потребуется учитывать приоритеты стилей, усиливая их вложенным контекстом селекторов:

```
header { }
header > img { }
```

```
header > h1 { }
.header__nav { }
```

Подобная разметка плохо структурирована:

- элементы с классом можно стилизовать напрямую;
- а вот элементы без класса стилизуются через контекст вложения в блок `<header>`, чтобы усилить приоритет стиля (иначе он будет таким же, как и обычных заголовков `<h1>` и изображений `<img>` по всему документу).

Назначим классы всем элементам:

```
<header class="header">
    <img src="" alt="" class="header__logo">
    <h1 class="header__title"></h1>
    <nav class="header__nav"></nav>
</header>
```

Теперь описание селекторов классов не требует вложенного контекста. Более того, на в названиях классов отражается смысловая привязка каждого элемента к блоку `<header>`:

```
.header { }
.header__logo { }
.header__title { }
.header__nav { }
```

Вопросы для самопроверки

1. Почему селекторов тегов недостаточно для полноценной настройки стилей элементов?
2. В чем состоит концепция разбиения разметки на отдельные классы элементов?
3. Опишите синтаксис селекторов классов и правило их подключение к тегам.
4. Чем отличаются селекторы с привязкой к тегу от автономных селекторов? Какие из них более универсальны и почему?
5. Для каких целей используют идентификаторы?
6. Почему важно описывать классы для тегов разметки документа и каким образом их грамотно использовать в оптимизации кода.

Практикум

Задание 1

1. Создайте каталог *Классы и идентификаторы*.
2. Изучите приведенный ниже код разметки и стилей.
3. Следуя описанию, дополните HTML-код разметкой таблицы и описанием свойств классов, которых не хватает для требуемого результата (рис. 3.35).

Глава 3\Классы и идентификаторы\index.html

```
<h1 class="title">Классы CSS</h1>
<p>Для выполнения этого задания читайте текст
разметки.</p>
<p class="example">К этому абзацу прикреплен класс
<strong>example</strong>. В его описании установить серый
цвет текста и шрифт <strong>Georgia</strong>.</p>
<p class="rule">Этот абзац описан с классом
<strong>rule</strong>. Он уже частично оформлен, однако
еще требуется установить размер шрифта в 1.25em, жирное
начертание и залить фон.</p>
<p>Ниже постройте таблицу в 6 строк и 2 колонки, шириной
400 пикселей. Ширина первой колонки - 10%. Далее к каждой
нечетной строке подцепите класс <strong>nechet</strong>,
а к четной - <strong>chet</strong>. Благо, классы автор
уже описал.</p>
```

Глава 3\Классы и идентификаторы\style.css

```
body {
    font: 1.0em Arial;
}

p {
    line-height: 1.35;
}

strong {
    color: #cf1a1a;
    font-style: italic;
}

.title {
    background: #960000;
    color: #fffafa;
```

```
    font: bold 2.5em Calibri;
    margin: 0;
    padding: 10px;
    text-align: center;
}

.example {
    /* Реализовать */
}

.rule {
    margin: 10px;
    padding: 10px 20px;
    border: 2px solid #6DA09C;
    /* Дописать */
}

td {
    padding: 5px;
}

tr.chet {
    /* Реализовать */
}

tr.nechet {
    /* Реализовать */
}
```

Задание 2

1. Создайте каталог *Корректировка разметки*.
2. Внимательно изучите предложенный ниже код (здесь используются элементы форм). Создайте три отдельный файла: с разметкой, стилями и скриптом. Скопируйте в них код.
3. Скорректируйте код разметки и стилей:
    a. избавьтесь от описания ID-селекторов в пользу селекторов классов (идентификаторы в тегах не удаляйте), допишите недостающие классы;
    b. замените контекстное вложение селекторов на описание классов к тем элементам, которым они не заданы и там, где это необходимо (см. пример пункта 3.2.4).
4. Результат должен быть аналогичен образцу на рис. 3.36.

Глава 3\Корректировка разметки и стилей\index.html

```
<!DOCTYPE html>
<html lang="ru">

<head>
    <meta charset="UTF-8">
    <meta http-equiv="X-UA-Compatible" content="IE=edge">
    <meta name="viewport" content="width=device-width,
initial-scale=1.0">
    <title>Document</title>
    <link rel="stylesheet" href="style.css">
</head>

<body class="globals">
    <header class="header">
        <div class="container">
            <h1>Каклькулятор квадратного
                уравнения</h1>
        </div>
    </header>

    <main id="content">
        <div class="container">
            <img src="images/formula.png"
                 alt="#"
                 class="equation-pic">
            <h2>Введите коэффициенты</h2>
            <form action="">
                <input type="text" value="1" id="coeffA">
                <label for="coeffA">x<sup>2</sup> +
                    </label>
                <input type="text" value="0" id="coeffB">
                <label for="coeffB">x + </label>
                <input type="text" value="0" id="coeffC">
                = 0
            </form>
        </div>
    </main>

    <script src="code.js"></script>
</body>

</html>
```

Глава 3\Корректировка разметки и стилей\style.css

```
body {
    font: 16px Tahoma;
    margin: 0;
}

/* Глобальная заливка страницы */
.globals {
    background: #cacaca;
}

/* Блок ограничивает ширину содержимого */
.container {
    min-width: 480px;
    max-width: 720px;
    margin: 0 auto;
    padding: 8px;
}

/* Шапка веб-страницы */
.header {
    background: #7e7b7b;
}

/* Заголовок в блоке шапки веб-страницы
   Здесь требуется контекстное вложение,
   Чтобы переопределить оформление заголовка
   в этом месте */
.header > .container > h1 {
    color: #ffffff;
    margin: 0;
    padding: 10px 0;
}

/* Блок контента */
#content {
    background: #dddddd;
    padding-top: 10px;
    padding-bottom: 15px;
}
```

```css
/* Изображение с уравнением */
.equation-pic {
    display: block;
    width: 70%;
    border: 1px solid #7e7b7b;
    margin: 0 auto;
}

/* Дополнительное оформление блока формы,
   которая содержит управляющие элементы.
   Также требуется контекстное вложение,
   Чтобы стили сработали */
#content > .container > form {
    background: #ececec;
    border: 2px solid white;
    border-radius: 5px;
    padding: 10px;
}

/* Текстовые поля для ввода коэффициентов.
   Заметим, что в случае описания классов
   достаточно будет только одного класса,
   т.к. поля имеют одинаковые свойства именно
   для этой формы.
   Этот класс и можно будет подключить ко всем
   трем тегам полей ввода */
#coeffA, #coeffB, #coeffC {
    width: 60px;
    font-size: 16px;
    border-radius: 6px;
}
```

Глава 3\Корректировка разметки и стилей\code.js

```js
// Здесь будет реализация JS-кода
// (В рамках этого задания делать не требуется!)

const coeffA = document.getElementById("coeffA");
const coeffB = document.getElementById("coeffB");
const coeffC = document.getElementById("coeffC");

// и т.д...
```

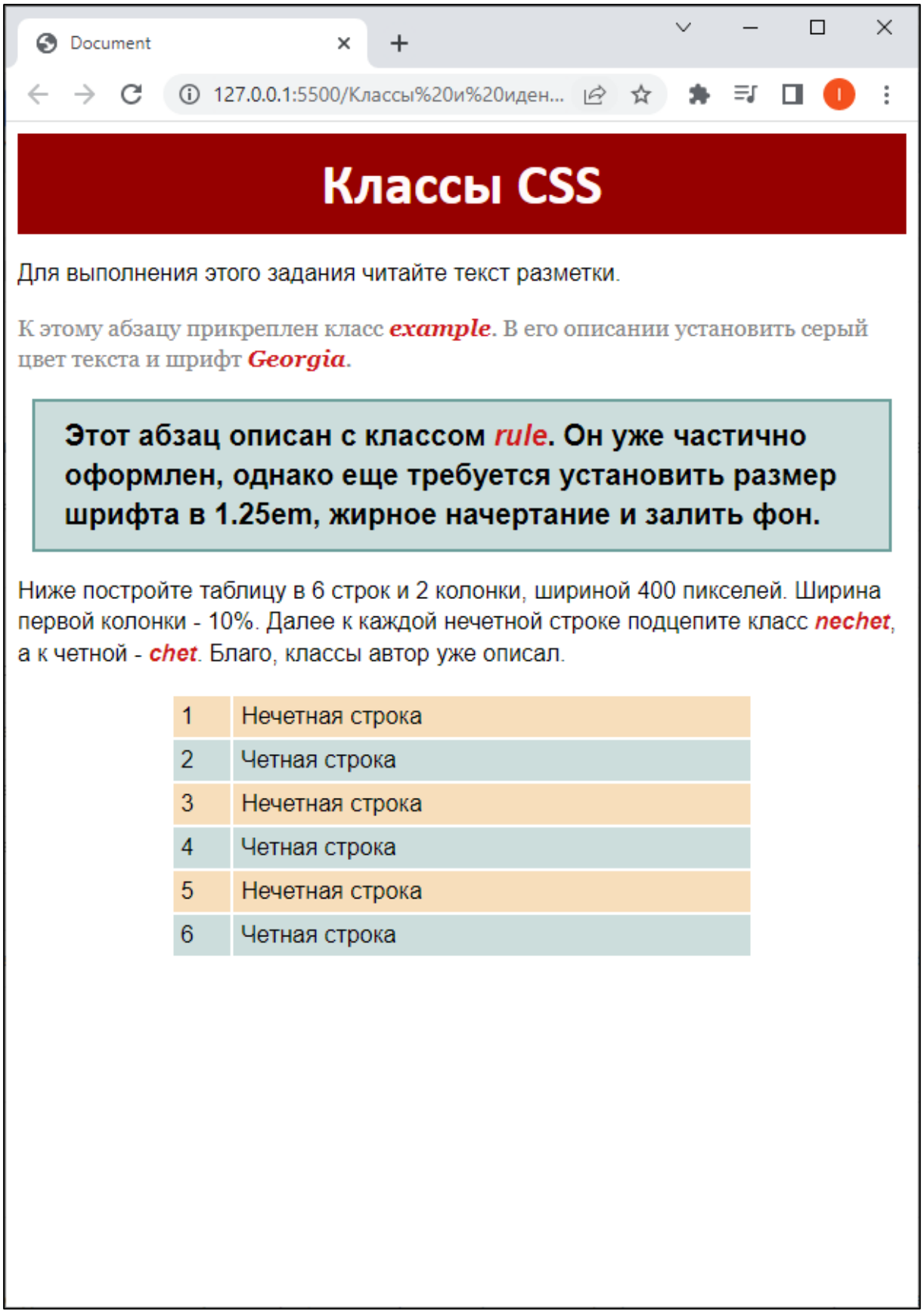
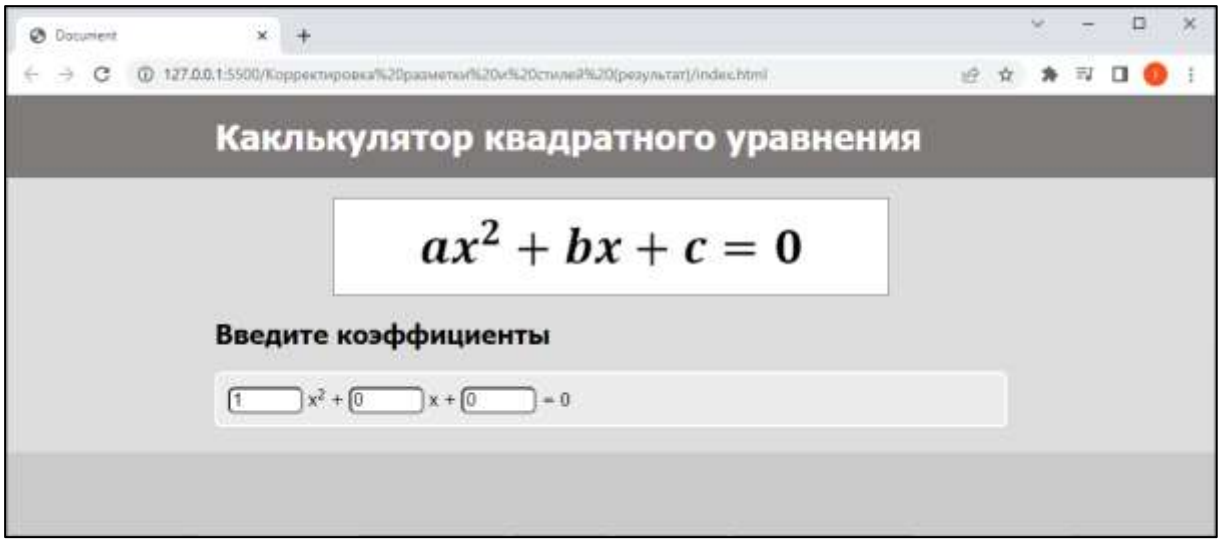


Рис. 3.35. Образец выполнения задания 1



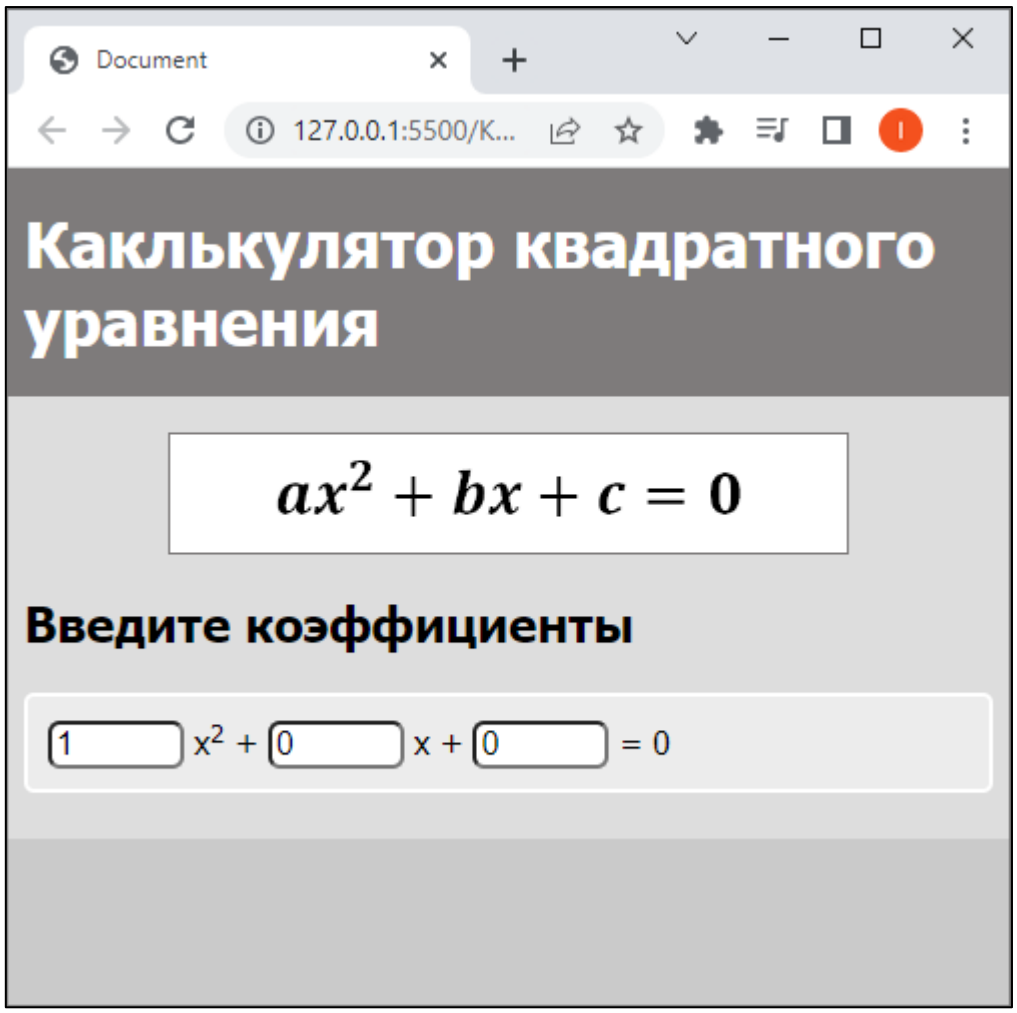


Рис. 3.36. Образец выполнения задания 2

3.3. Блочные и строчные элементы

3.3.1. Единицы измерения в CSS

Категории единиц измерения величин

В CSS поддерживаются многочисленные единицы измерения числовых величин. Условно их можно разбить на две категории: *абсолютные* и *относительные*.

Единица измерений	Описание	Тип
`px`	Пиксели	Относительная
`pt`	Пункты	Абсолютная
`in`	Дюймы	Абсолютная
`cm`	Сантиметры	Абсолютная
`mm`	Миллиметры	Абсолютная
`pc`	Пика	Относительная
`em`	Em (Мутт)	Относительная
`rem`	Rem	Относительная
`ex`	Ex (x-высота)	Относительная
`%`	Проценты	Относительная

Рис. 3.37. Некоторые часто используемые единицы измерения

Кроме того, при оформлении эффектов и фильтров часто требуются значения угловой меры: **`deg`** (*градусы*), **`rad`** (*радианы*), **`grad`** (*градианы*, 1/100 от $\pi/2$), **`turn`** (*полный оборот*).

Помимо основных единиц измерения, иногда возникает необходимость работать с **`dpi`** (*плотность пикселей на дюйм*), **`dpcm`** (*плотность пикселей на сантиметр*), **`dppx`** и **`x`** (*плотность в точках на пиксель*).

Значения времени в эффектах трансформации можно задавать в **`ms`** (*миллисекундах*) и **`s`** (*секундах*).

Более подробная информация: doka.guide.

Абсолютные единицы измерения

Абсолютные величины зависят от устройства вывода. Это может сказываться на качестве отображения текста и объектов на разных типах устройств.

Дюймы (**in**), *сантиметры* (**cm**), *миллиметры* (**mm**) являются одни из самых распространенных единиц измерений. *Пункт* (**pt**) часто используется для обозначения размера шрифта (`1in = 72pt`).

Однако в веб-дизайне использование абсолютных значений рекомендуется избегать.

Относительные единицы измерения

Относительные единицы не зависят от устройств вывода и сохраняют пропорции, поэтому рекомендуются в веб-разработке.

Пиксель (**px**) – наиболее часто используется в веб-дизайне. Размер экранной точки определяется размером и разрешением экрана пользователя.

Пика (*pc*) – единица измерений, которая используется для печати (`1in = 6pc`).

Em, или *Мутт* (**em**) – это относительная единица измерения, размер которой определяется по текущему контексту. Так если в некотором блоке размер шрифта составлял 20px, а в его дочернем блоке размер установлен в 1.5em, то дочерний блок будет определять размер шрифта, равный 30px.

Ex, или «*x-высота*» (**ex**), определяет высоту текущего шрифта по высоте строчной буквы «x». Ex применяется для установки размера контента пропорционально размеру шрифта.

Rem (**rem**) – величина, задающая размер шрифта относительно корневого элемента документа `<html>`, т.е. позволяющая менять размеры относительно глобального размера шрифта. Контекст на нее никак не влияет.

% – это относительная величина, задающая размер пропорционально размеру родительского элемента. 100% соответствует искомому размеру.

3.3.2. Блоки

Блочная модель

Каждый элемент в HTML-разметке тела документа можно отождествить с *блоком*, которому задаются определенные параметры. CSS определяет *блочную модель* построения веб-страниц, в которой положение объекта зависит как от его места в HTML-разметке, так и от свойств позиционирования.

Далее читатель убедится, что блочная разметка существенно отличается от табличной верстки.

Физика контейнеров элемента

Каждый отображаемый в браузере объект HTML-разметки генерирует прямоугольный блок, называемый *контейнером* элемента. Контейнер характеризуют четыре величины:

- *размер содержимого* – высота (**`height`**) и ширина (**`width`**);
- *отступ* (**`padding`**) – внутреннее расстояние до рамки;
- *рамка* (**`border`**) – видимая граница блока;
- *поле* (**`margin`**) – внешнее расстояние до рамки.

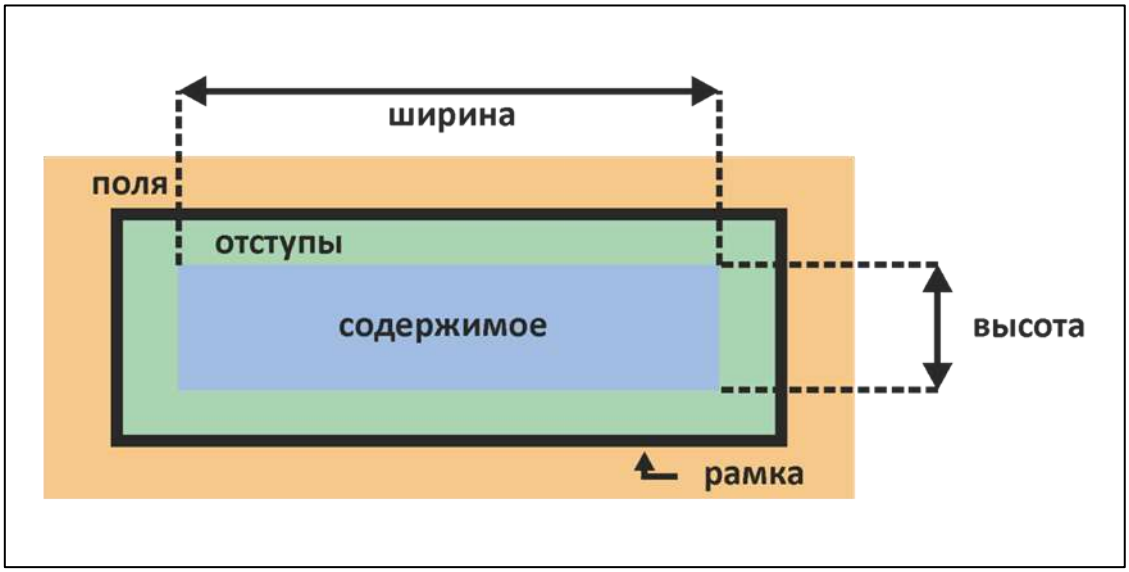


Рис. 3.38. Параметры контейнера для элемента

При этом допустимо, что одна или более характеристик блока могут быть равны нулю, в том числе все одновременно.

CSS-свойства `padding`, `margin` и `border` (и их вариации) могут задавать разные значения для любой из четырех сторон (далее распишем их подробнее).

Ширина и высота содержимого определяется свойствами `width` и `height` соответственно. Т.е. по умолчанию отступы, рамки и поля не входят в расчет ширины / высоты блока!

3.3.3. Блочные и строчные элементы

Типы элементов

Контейнеры элементов могут иметь одно из двух состояний, характеризующих их позиционирование.

Блочные элементы занимают всю доступную ширину окна браузера или родительского элемента, т.е. не позволяют другим элементам их обтекать. Высота определяется согласно содержимому, отступам, рамке и полям.

Примеры:

- `<div>` – абстрактный блочный элемент;
- `<p>` – текстовый абзац;
- `<h1-h6>` – все уровни заголовков;
- `<ul>`, `<ol>` – списки и др.

Строчные элементы позиционируют себя как часть текста в абзаце и обтекают другие элементы. Их фактическая ширина зависит от содержимого, отступов, рамок и полей; аналогично с высотой.

Примеры:

- `<span>` – абстрактный строковый элемент;
- `<b>`, `<i>`, `<strong>`, `<em>` – теги разметки фрагментов текста;
- `<a>` – гиперссылки;
- `<img>` – изображения (плавающий элемент) и др.

Блочные элементы

Главная особенность блочных элементов состоит в том, что они занимают всю доступную ширину блока-родителя, вытесняя другие соседние элементы (рис. 3.39).

Так, если в разметке присуствуют 4 подряд идущих блока, при этом некоторые имеют фиксированную ширину, меньше чем у родительского контейнера, то даже в этом случае каждый блок занимает свой «слой», не позволяя другим смежным блокам занять визуально «пустое» пространство.

Строчные элементы

Строчные элементы ведут себя иначе: их поведение похоже на размещение текста, который обтекает ближайшие к нему блоки изображений, таблиц или иных объектов.

CSS позволяет при необходимости делать блочные элементы строчными, что помогает управлять позицией элементов в разных частях веб-страницы (рис. 3.40).

Комбинирование блочных и строчных элементов

Наконец, обычно строчные и блочные элементы комбинируются в более сложную разметку, где смежные элементы взаимодействуют друг с другом.

- Как и ранее, смежные блоки располагаются отдельным «слоями» (уровнями).
- Смежные строковые элементы располагаются как текст в абзаце. Высота строки определяется по высоте самого высокого строчного элемента текущей строки.
- Блоки вытесняют строковые элементы. При этом даже зафиксированная ширина блока не позволит текстовому элементу подняться выше и занять свободное пространство (рис. 3.41).

Дополнительную информацию по блочным и строчным элементам можно найти на сайте html5book.ru.

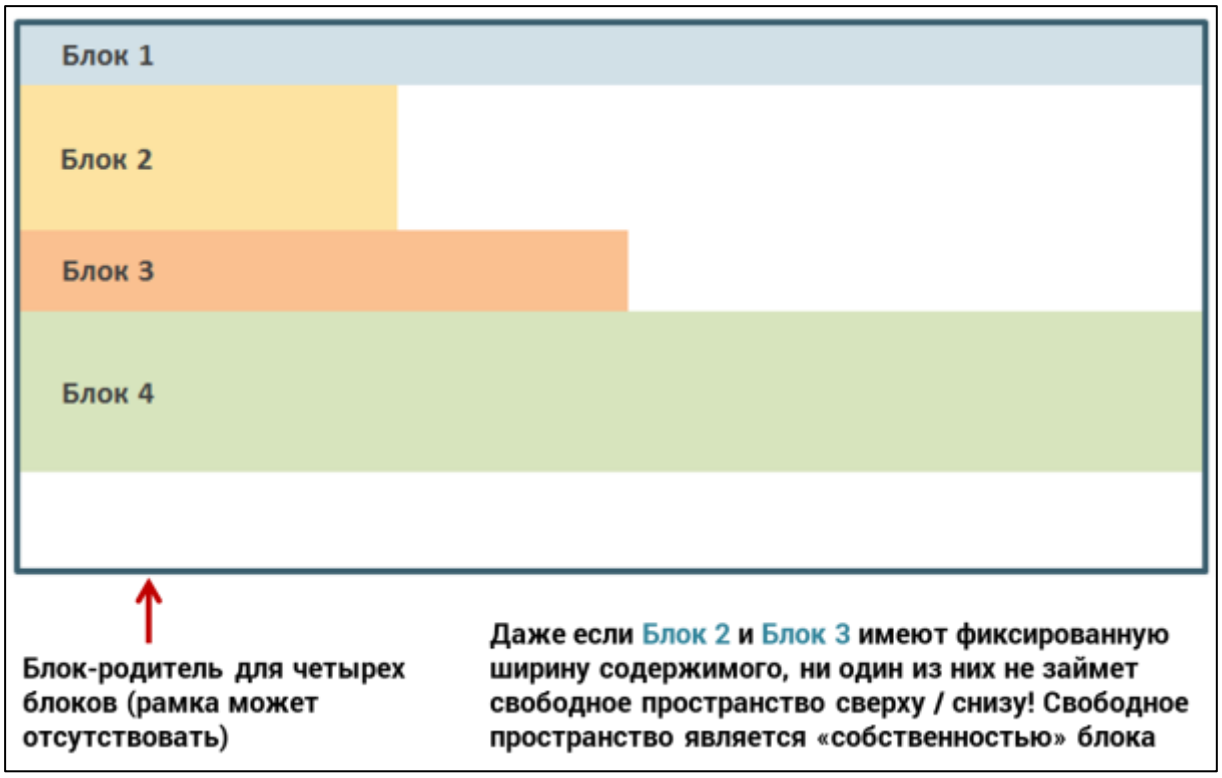


Рис. 3.39. Позиционирование смежных блоков

Рис. 3.40. Позиционирование смежных строчных элементов

Рис. 3.41. Комбинирование блочных и строчных элементов

Преобразование типа контейнера

CSS изначально задает тегам разметки свой тип, условно разделяя их на блочные и строчные (есть также понятие плавающих элементов, о них подробнее в других разделах).

При необходимости предустановленное поведение контейнера элемента можно изменить с помощью CSS-свойства **`display`**.

Определение

`display`

*CSS-свойство, задающее **тип контейнера элемента**.*
Чаще всего используется одно из трех его значений:

- ***`block`*** *(элемент отображается блоком и занимает доступную ширину родителя);*
- ***`inline`*** *(элемент становится строчным);*
- ***`inline-block`*** *(элемент становится блоком, но другие элементы, в частности текст, его уже будут обтекать).*

Например, сделаем с помощью класса блочные теги заголовка и абзаца *строчными блоками*. Чтобы не смешивать свойства типа контейнера и оформления элемента, разобьем их на отдельные классы:

- класс `text-box` переводит элемент в строчно-блочный;
- классы `sub-title` и `sub-par` отвечают за оформление заголовков и абзацев (которые их будут подключать).

Результат изображен на рис. 3.42.

Глава 3\Блочные и строчные элементы\Свойство display\index.html

```
<h1 class="tex-box sub-title">Свойство display</h1>
<p class="text-box sub-par">
    <dfn>display</dfn> - это CSS-свойство, задающее тип
    контейнера элемента. Чаще всего используется одно из
    трех его значений: <em>block</em> (элемент
    отображается блоком и занимает доступную ширину
    родителя), <em>inline</em> (элемент становится
    строчным), <em>inline-block</em> (элемент становится
    блоком, но другие элементы, в частности текст, его
    уже будут обтекать).
</p>
```

Глава 3\Блочные и строчные элементы\Свойство display\style.css

```
.text-box {
    display: inline-block;
}

.sub-title {
    background: #990606;
    color: white;
    padding: 10px;
}

.sub-par {
    width: 200px;
    background: #e7e7e7;
    padding: 20px;
}
```

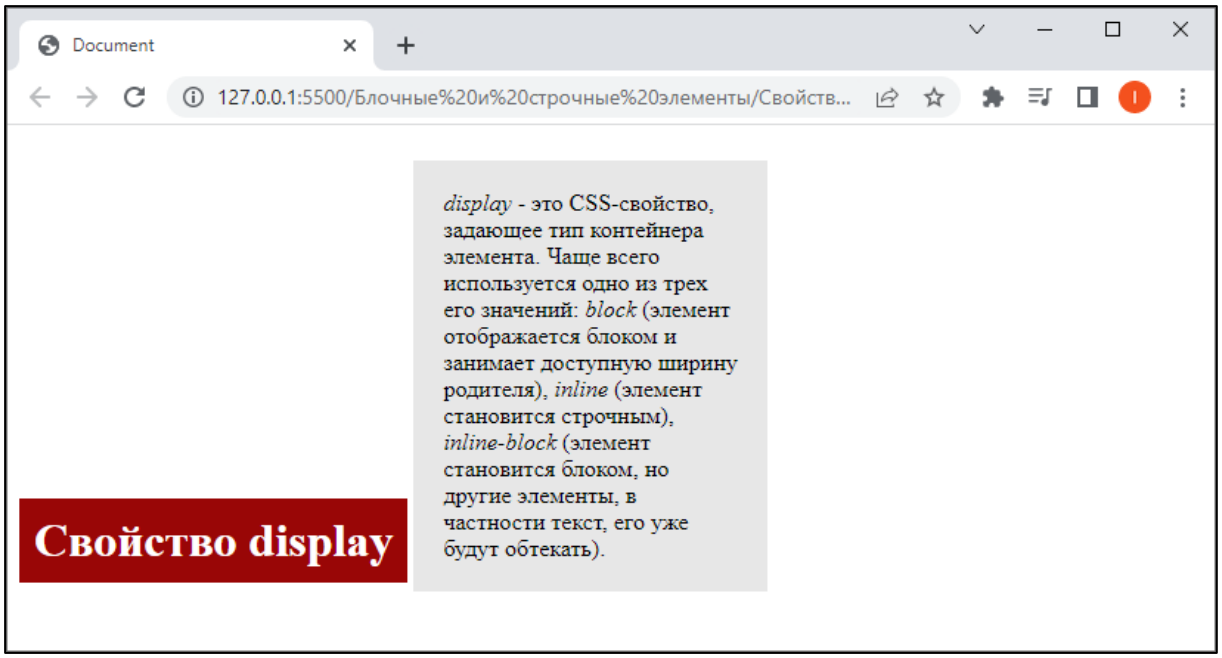


Рис. 3.42. Пример перевода блочных элементов в строчно-блочный

Здесь важно обратить внимание на ряд деталей:

- Для блока абзаца преднамеренно установлена фиксированная ширина содержимого. Это позволяет ему подняться на текстовую линию блока заголовка.
- Блок абзаца получается выше, поэтому визуально он «выталкивает линию текста вниз».
- Обратите внимание, что между блоками наблюдается тонкий пробел. Такая особенность характерна строковым элементам.

Если убрать класс `text-box` из обоих тегов, то получим стандартное поведение блоков:

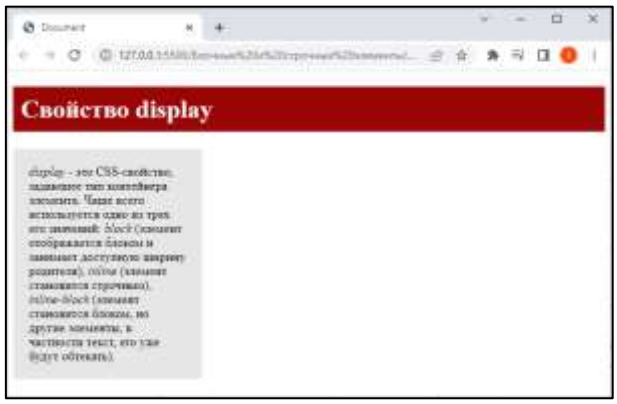


Рис. 3.43. Позиционирование смежных блоков по умолчанию

3.3.4. Абстрактные блочные и строчные элементы

Необходимость в абстрактных элементах

Количество тегов HTML ограничено. Часто создается ситуация, когда для разметки и оформления какого-либо элемента (блочного / строчного) просто нет подходящего по смыслу тега.

Для этих случаев в HTML предусмотрены два абстрактных тега-контейнера: `<div>` и `<span>`. Эти теги не вносят какой-либо смысл в разметку, однако они могут принять любое оформление за счет подключения классов.

Блочный тег

Определение

```
<div></div>
```

Контейнер представляет блочный элемент.
В него можно поместить как строчные, так и блочные элементы, в частности, другие блоки *** ****.*
По умолчанию тегу задано свойство

```
display: block;
```

С приходом CSS тег является наиболее важным и чаще всего используемым в разметке. По существу, он определил новый подход – *блочную вертску* (в отличие от табличной, которую в текущем курсе больше не используем).

Например, следующий код может использоваться для разметки блока определения, где класс `box` оформляет стилизованный блок, а вложенный контейнер с классом `definition` уточняет, что это будет разметка определения.

```
<div class="box">
    <div class="definition">
        ...
    </div>
</div>
```

Строчный тег

Определение

```
<span></span>
```

Контейнер представляет строчный элемент.
По умолчанию тегу задано свойство

```
display: inline;
```

Обычно отвечает за стилизацию части текста, реже других элементов, которые нужно сделать строчными.

Например, пусть в тексте ФИО будут выделяться отдельным стилем, который задается классом person:

```
<p>
    Пособие было разработано
    <span class="person">Якубовичем Д.А.</span>
    и
    <span class="person">Ероповой Е.С.</span>
</p>
```

Приемы работы с Visual Studio Code

Плагин Emmet при наборе автоматически цепляет «независимый» класс или идентификатор к тегу *** ****.*
Для ****** *редактор смотрит по контексту вложения. Например, когда осуществляется вложение внутри абзаца (т.к. внутри абзаца может быть расположен только строчный элемент).*

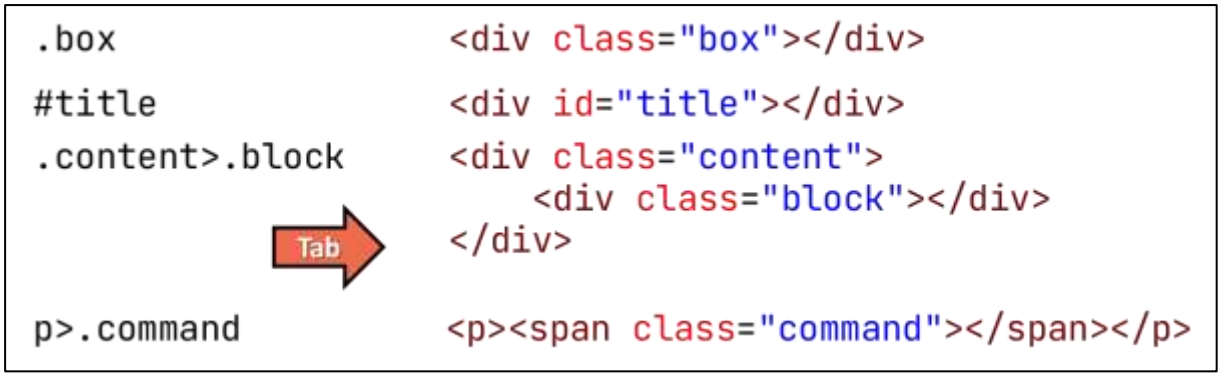


Рис. 3.44. Быстрая верстка блоков и

Приведем пример использования и в одном документе. Стоит заметить, что в данном случае лучше заменить на семантический тег <dfn> или : они как раз и предназначены для разметки терминов или важных слов:

**Глава 3\Блочные и строчные элементы\
Контейнеры div и span\index.html**

```
<div class="defin-box">
    <p>
        <span class="term">Тег</span> – правило, согласно
        которому информация отображается в браузере.
    </p>
</div>
```

**Глава 3\Блочные и строчные элементы\
Контейнеры div и span\style.css**

```
body {
    font: 18px Georgia, Cambria, 'Times New Roman';
}

.defin-box {
    background: #CFFCD7;
    font-weight: bold;
    border-left: 5px solid #2C8348;
    margin: 10px 20px;
    padding: 6px 15px;
}

.term {
    font: bold 120% Georgia, 'Times New Roman';
    color: #095c5f;
}
```

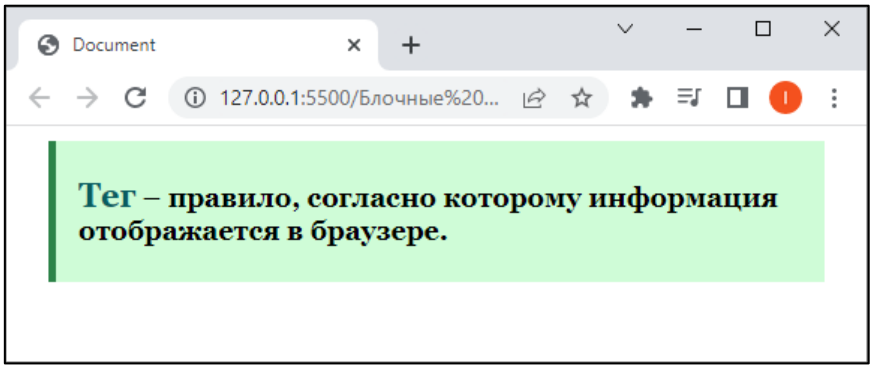


Рис. 3.45. Пример совместного использования и

3.3.5. Особенности вложения блоков

Особенности вложенных блоков

Более интересным является вопрос: каким образом ведут себя блоки в случае вложения и как могут влиять на это их свойства ширины и высоты?

Блочная верстка концептуально отличается от устаревшей табличной. И на первый взгляд кажется более сложной.

Однако на самом деле с помощью блоков можно реализовать сложную компоновку элементов на странице, которая недостижима для таблиц. Именно блочная верстка открывает возможности к современному адаптивному дизайну.

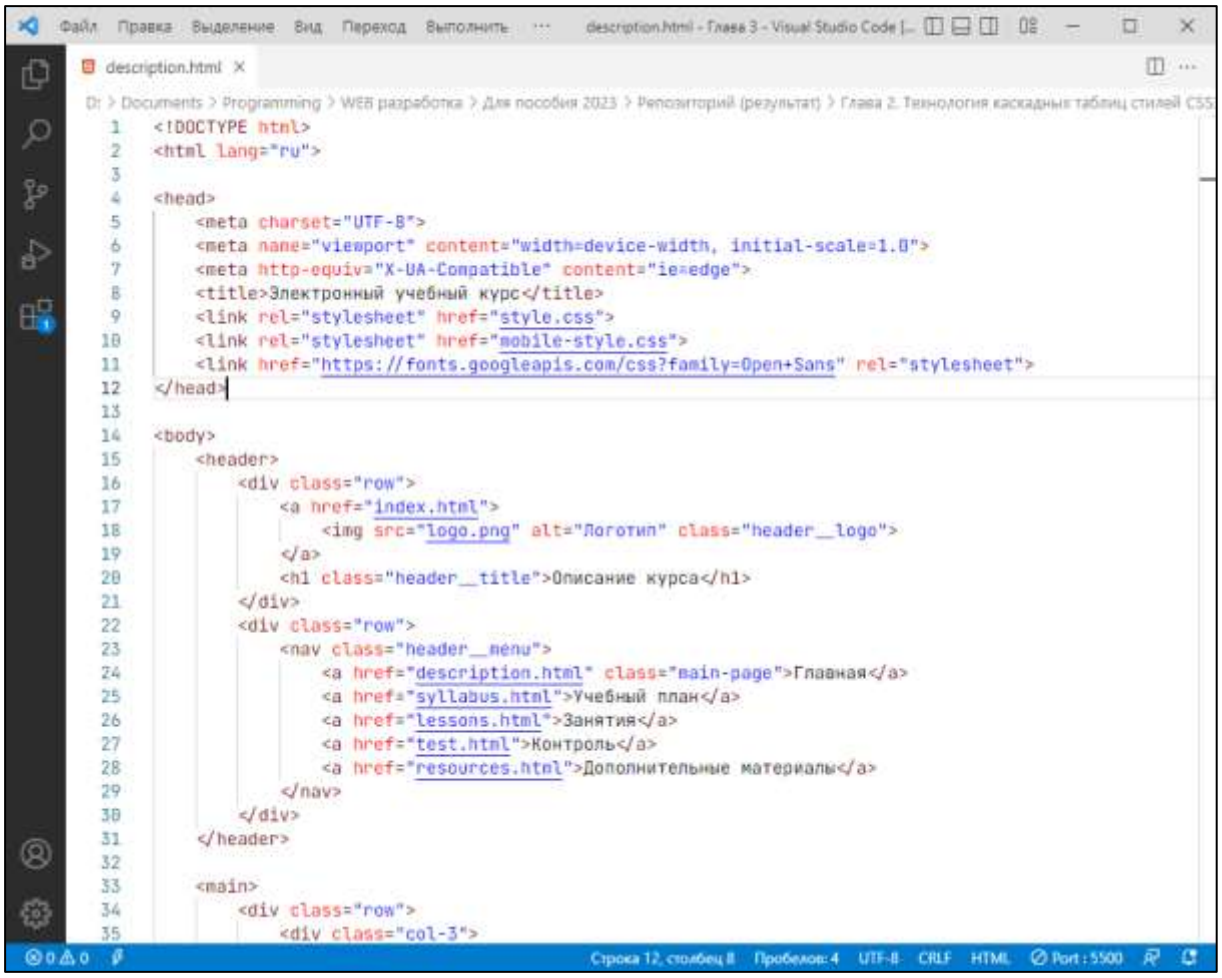


Рис. 3.46. Блочная верстка: тег `<div>` используется наряду с семантическими тегами структуры документа в HTML5

Рассмотрим некоторые важные случаи вложения блоков на ряде примеров.

Вложение блоков

1. Нормальный поток и неограниченная ширина

По умолчанию любой блочный элемент занимает всю доступную ширину тега-родителя. Говорят, что подряд идущие блоки располагаются в *нормальном потоке*.

**Глава 3\Блочные и строчные элементы\
Вложение блоков\Case 1\index.html**

```
<div class="blockA">
    Пример блока 1.
</div>
<div class="blockB">
    Пример блока 2.
</div>
```

**Глава 3\Блочные и строчные элементы\
Вложение блоков\Case 1\style.css**

```
/* Отступ внутри для двух блоков (группируем описание) */
.blockA,
.blockB {
    padding: 10px;
}

.blockA { background: #FF9882; }

.blockB { background: #BFF788; }
```

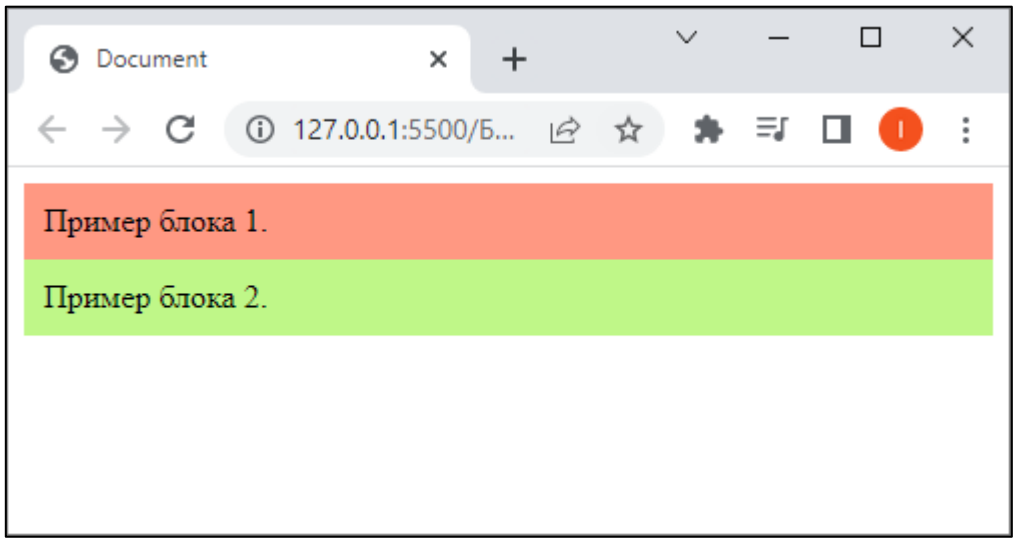


Рис. 3.47. Первый вариант вложения блоков

2. *Нормальный поток и фиксированная ширина*

Если зафиксировать ширину блока `blockA`, то на позицию блоков это никак не повлияет: блоки остаются в нормальном потоке. Свойство `width` отвечает за ширину контента блока и влияет на заливку. Аналогично будет, если зафиксировать ширину блока `blockB`.

Глава 3\Блочные и строчные элементы\Вложение блоков\Case 2\index.html

```
<div class="blockA">
    Пример блока 1.
</div>
<div class="blockB">
    Пример блока 2.
</div>
```

Глава 3\Блочные и строчные элементы\Вложение блоков\Case 2\style.css

```
.blockA,
.blockB {
    padding: 10px;
}

.blockA {
    background: #FF9882;
    width: 250px;
}

.blockB { background: #BFF788; }
```

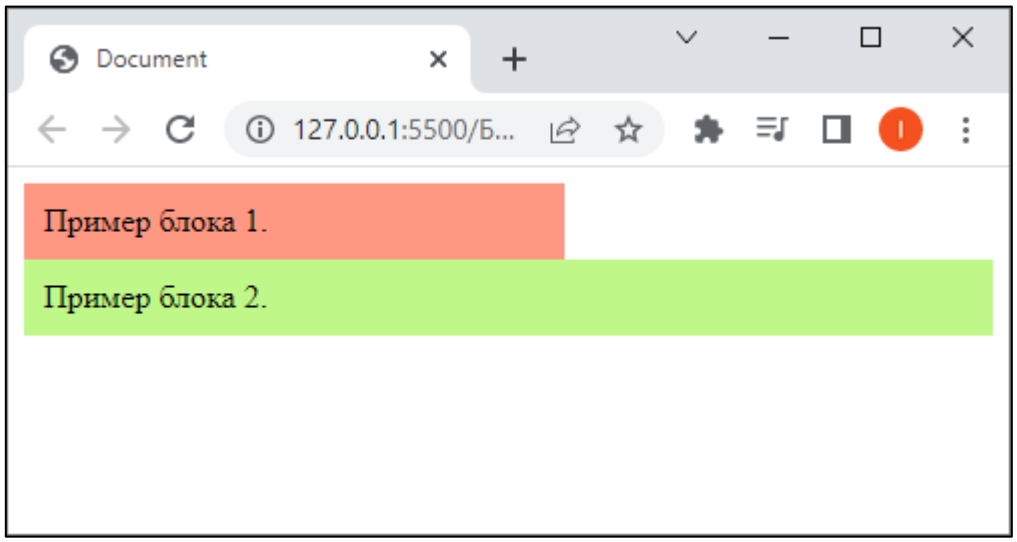


Рис. 3.48. Второй вариант вложения блоков

3. *Вложение с неограниченной шириной*

Вставим блок `blockA` внутрь `blockB`. Из рис. 3.49 видно, что правило расположения блочного и строчного элемента сохраняется, но уже внутри блока `blockB`. В частности, внутренний отступ (`padding`) блока `blockB` влияет как на текст этого блока, так и на вложенный блок `blockA`:

Глава 3\Блочные и строчные элементы\Вложение блоков\Case 3\index.html

```
<div class="blockB">
    <div class="blockA">
        Пример блока 1.
    </div>
    Пример блока 2.
</div>
```

Глава 3\Блочные и строчные элементы\Вложение блоков\Case 3\style.css

```
.blockA,
.blockB {
    padding: 10px;
}

.blockA { background: #FF9882; }

.blockB { background: #BFF788; }
```

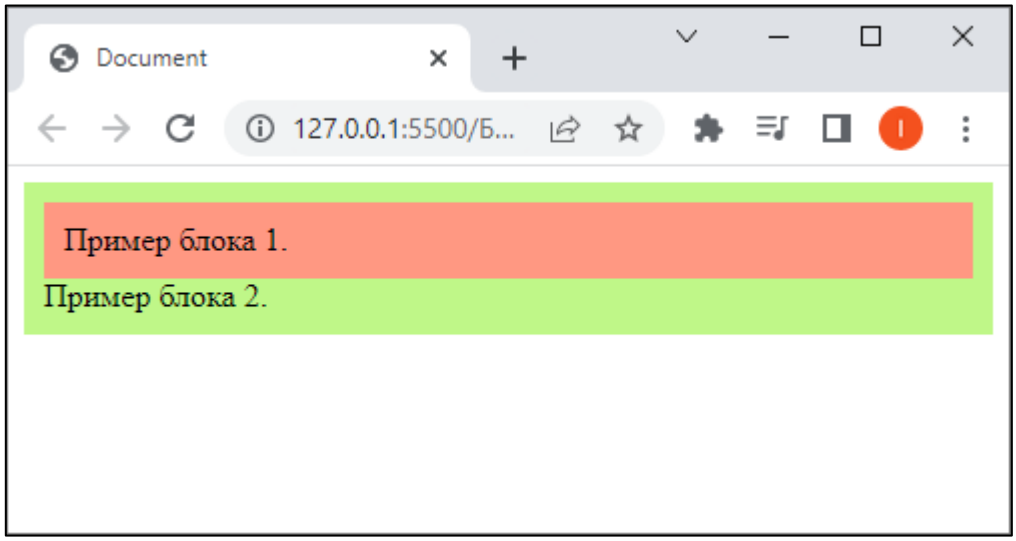


Рис. 3.49. Третий вариант вложения блоков

4. *Вложение с ограниченной шириной*

Ничего не изменится, если зафиксируем ширину контента вложенного блока `blockA`: он по-прежнему не позволит обтекание тексту блока `blockB`.

Глава 3\Блочные и строчные элементы\Вложение блоков\Case 4\index.html

```
<div class="blockB">
    <div class="blockA">
        Пример блока 1.
    </div>
    Пример блока 2.
</div>
```

Глава 3\Блочные и строчные элементы\Вложение блоков\Case 4\style.css

```
.blockA,
.blockB {
    padding: 10px;
}

.blockA {
    background: #FF9882;
    width: 250px;
}

.blockB { background: #BFF788; }
```

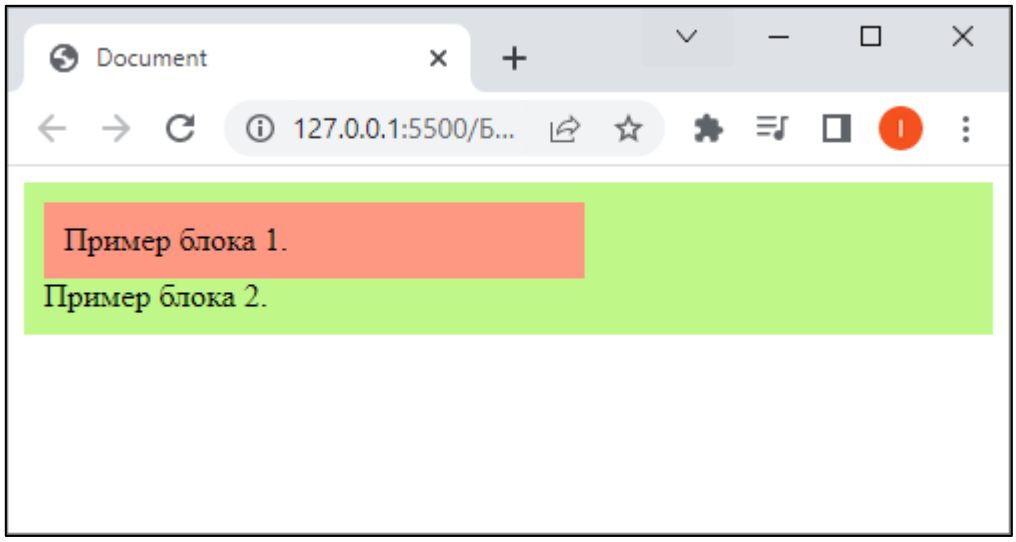


Рис. 3.50. Четвертый вариант вложения блоков

5. *Вложение с изменением высоты*

Зададим фиксированную высоту вложенному блоку `blockA`. Обратите внимание: блок `blockB`, как родительский блок, растягивается по высоте и сохраняет внутренний отступ в 10px.

Глава 3\Блочные и строчные элементы\Вложение блоков\Case 5\index.html

```
<div class="blockB">
    <div class="blockA">
        Пример блока 1.
    </div>
    Пример блока 2.
</div>
```

Глава 3\Блочные и строчные элементы\Вложение блоков\Case 5\style.css

```
.blockA,
.blockB {
    padding: 10px;
}

.blockA {
    background: #FF9882;
    width: 250px;
    height: 100px;
}

.blockB { background: #BFF788; }
```

Рис. 3.51. Пятый вариант вложения блоков

6. *Вложение с фиксированной высотой*

Однако, если также зафиксируем высоту родительскому блоку `blockB` и зададим ее меньше, чем высота блока `blockA`, то получится, что блок `blockA` «выталкивает» содержимое блока `blockB`:

Глава 3\Блочные и строчные элементы\Вложение блоков\Case 6\index.html

```
<div class="blockB">
    <div class="blockA">
        Пример блока 1.
    </div>
    Пример блока 2.
</div>
```

Глава 3\Блочные и строчные элементы\Вложение блоков\Case 6\style.css

```
.blockA, .blockB {
    padding: 10px;
}

.blockA {
    background: #FF9882;
    width: 250px;
    height: 150px;
}

.blockB {
    background: #BFF788;
    height: 100px;
}
```

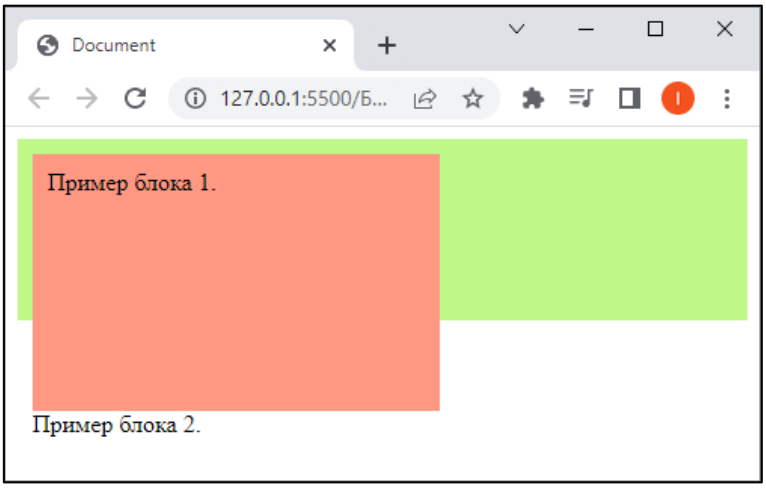


Рис. 3.52. Шестой вариант вложения блоков

7. Другие блоки в нормальном потоке

Наконец, добавим третий блок `blockC` после блока `blockB`. Третий блок встанет сразу после второго (согласно нормальному потоку следования), при этом «перекроет» частично выходящий за рамки вложенный блок `blockA`. Здесь демонстрируется еще одна особенность блоков: по умолчанию они выстраиваются последовательными слоями (их можно менять, управляя свойствами `position` и `z-index`, см. далее).

**Глава 3\Блочные и строчные элементы\
Вложение блоков\Case 7\index.html**

```
<div class="blockB">
    <div class="blockA">
        Пример блока 1.
    </div>
    Пример блока 2.
</div>
<div class="blockC">
    Пример блока 3.
</div>
```

**Глава 3\Блочные и строчные элементы\
Вложение блоков\Case 7\style.css**

```
.blockA, .blockB, .blockC {
    padding: 10px;
}

.blockA {
    background: #FF9882;
    width: 250px;
    height: 150px;
}

.blockB {
    background: #BFF788;
    height: 100px;
}

.blockC {
    background: #91eafa;
}
```

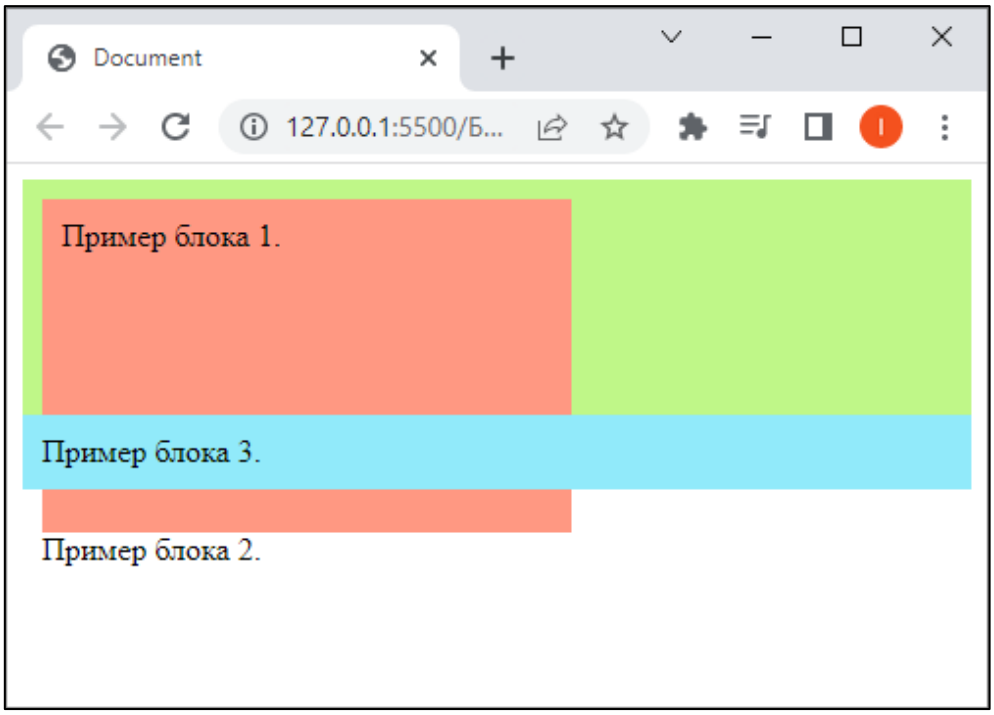


Рис. 3.53. Седьмой вариант вложения блоков

Это полезно знать!

При вложении блоков следует заранее определить, как содержимое может повлиять на поведение и оформление блоков при изменении ширины окна браузера.
Хорошей практикой является соблюдение следующих правил:

- *не фиксировать высоту родительского блока (она будет подбираться согласно высоте его контента, в т.ч. вложенных в него блоков);*
- *фиксировать высоту родительского блока при условии, что высота контента никак не повлияет на оформление при манипуляциях с шириной окна браузера.*

Важное замечание!

В примерах выше мы говорили о блоках, указывая названия классов. Это вполне естественное отождествление.
Далее любые классы, оформляющие элементы как блоки, для простоты будем называть ***блоками****.*

3.3.6. Поля, отступы и рамки элемента

Поля элемента

Определение

```
margin
margin-top
margin-bottom
margin-left
margin-right
```

Семейство CSS-свойств, задающих ***поля*** *элемента, т.е.* ***внешний отступ*** *от края объекта.*
Первое свойство является универсальным и позволяет задавать поля со всех сторон одновременно.
Остальные настраивают отступ с определенного края.

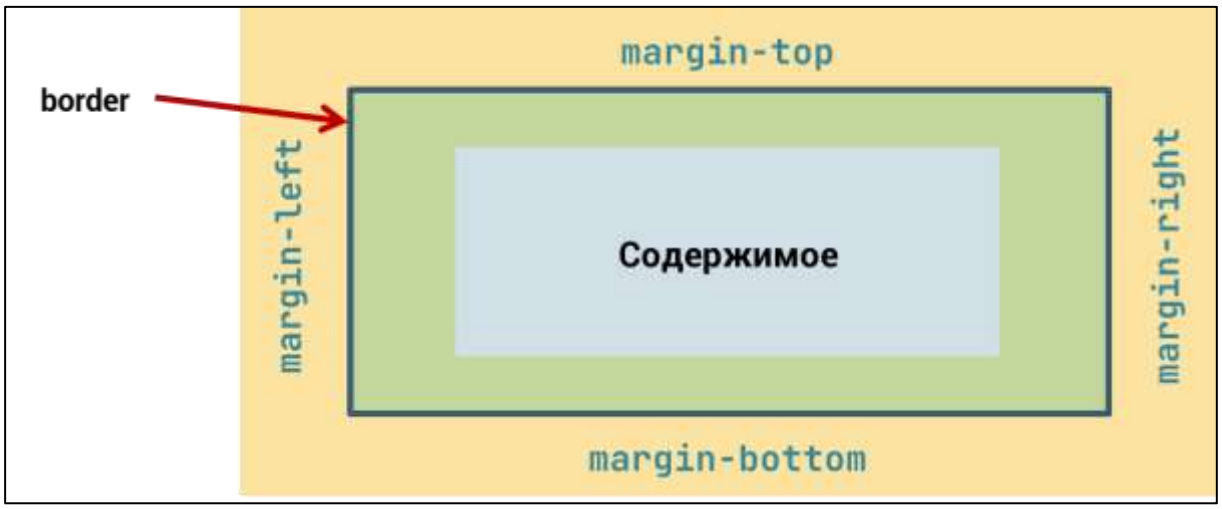


Рис. 3.54. Поля элемента

Свойство **margin** является универсальным и позволяет установить отступы для всех сторон одновременно (т.е. не менять поля по отдельности).

В зависимости от количества значений свойства margin поля устанавливаются следующим образом:

Таблица 3.1. Универсальное свойство `margin` и аналогичные преобразования в совокупности отдельных свойств

Универсальное свойство	**Отдельные свойства**
1 значение. Задает одинаковые поля со всех сторон	

`margin: 10px;`	`margin-top: 10px;`
	`margin-bottom: 10px;`
	`margin-right: 10px;`
	`margin-left: 10px;`

2 значения. Задает поля сверху / снизу и справа / слева.	

`margin: 10px 15px;`	`margin-top: 10px;`
	`margin-bottom: 10px;`
	`margin-right: 15px;`
	`margin-left: 15px;`

3 значения. Задает поле сверху, справа / слева и снизу.	

`margin: 15px 10px 30px;`	`margin-top: 15px;`
	`margin-right: 10px;`
	`margin-left: 10px;`
	`margin-bottom: 30px;`

4 значения. Задает поле сверху, справа, снизу и слева.	

`margin: 15px 10px 30px 20px;`	`margin-top: 15px;`
	`margin-right: 10px;`
	`margin-bottom: 30px;`
	`margin-left: 20px;`

Рис. 3.55. Схема распределения параметров в универсальном свойстве `margin` в противовес последовательному описанию полей с четырех сторон (обход по часовой стрелке). Каждый цвет обозначает отдельный параметр

Отступы элемента

Определение

padding
padding-top
padding-bottom
padding-left
padding-right

*Семейство CSS-свойств, задающих **отступы** элемента, т.е. **внутренний отступ** от края объекта до контента.*
Первое свойство является универсальным и позволяет задавать поля со всех сторон одновременно.
Остальные настраивают отступ с определенного края.

Рис. 3.56. Внутренние отступы для содержимого

Свойство **padding** является универсальным и позволяет установить отступы для всех сторон одновременно. Его синтаксис в точности аналогичен синтаксису свойства `margin` (см. предыдущий пункт).

Например, следующий код

```
padding: 20px 15px;
```

аналогичен коду

```
padding-top: 20px;
padding-bottom: 20px;
padding-left: 15px;
padding-right: 15px;
```

Рамки и углы элемента

Определение

`border`
`border-top`
`border-bottom`
`border-left`
`border-right`

*Семейство CSS-свойств, задающих **рамку** элемента. По умолчанию элемент не имеет рамки («нулевая» толщина).*
Первое свойство является универсальным и позволяет задавать рамку со всех сторон одновременно.
Остальные настраивают рамку для определенного края.
Каждому из перечисленных свойств указывается три значения: толщина, тип линии контура и цвет границы, например:

```
border: 3px solid #0F363C;
```

Дополнительно есть отдельные свойства, управляющие каждым из этих параметров в одиночку (для всех сторон или одной).

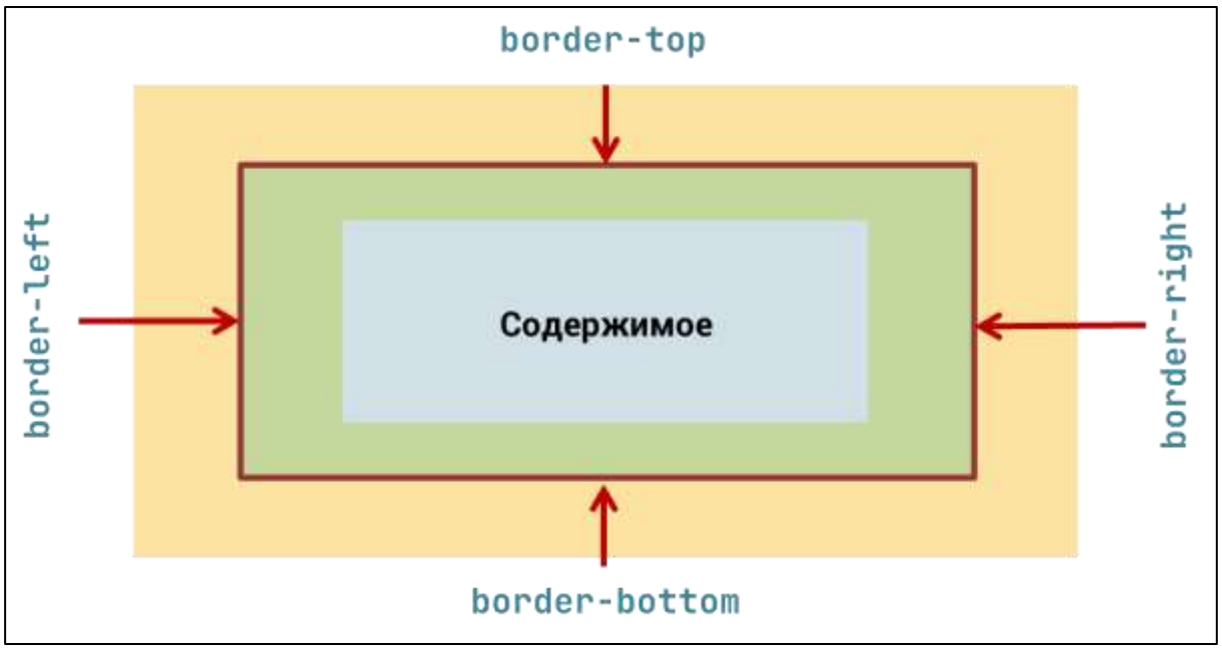


Рис. 3.57. Рамка элемента

В качестве примера зададим рамку толщиной в 5 пикселей, сплошным контуром и зеленым цветом:

```
border: 5px solid green;
```

Допускается менять отдельные параметры границы. Например, для нижней границы сменим тип контура на пунктир и скорректируем цвет:

```
border-bottom-style: dashed;
border-bottom-color: orange;
```

Определение

`border-radius`
`border-top-left-radius`
`border-top-right-radius`
`border-bottom-left-radius`
`border-bottom-right-radius`

*Семейство CSS-свойств, задающих **скругление углов** элемента.*
По умолчанию углы не скругляются.
Первое свойство является универсальным и позволяет скруглить все 4 угла.
Остальные скругляют соответствующие углы.

Каждому из свойств обычно задают одно значение – радиус скругления угла(ов). Например:

```
border-radius: 6px;
```

Если объект имеет форму квадрата, то значение в 50% трансформирует его в круг (для прямоугольной формы – в овал):

```
border-radius: 50%;
```

Также универсальному свойству **`border-radius`** допускается указывать от 1 до 4 параметров, задавая разные скругления углов (применяется редко):

```
border-radius: 12px 50px 14px 80px;
```

Приемы работы с Visual Studio Code

Плагин Emmet позволяет сократить набор CSS-свойств и их значений.
В частности – для отступов и полей, например рис. 3.58.

Рис. 3.58. Использование аббревиатур Emmet при генерации CSS-свойств

Вопросы для самопроверки

1. Какие единицы измерения величин используются в CSS и чем отличаются абсолютные единицы от относительных?
2. Опишите особенности блочной верстки.
3. Перечислите основные CSS-параметры, управляющие размерами контейнера элемента.
4. В чем принципиальное отличие поведения в разметке блочных и строчных элементов?
5. Каким образом можно преобразовать блочный элемент в строчный и обратно?
6. Охарактеризуйте назначение тегов `<div>` и `<span>`.
7. Какие проблемы могут возникать при вложении блоков и каких правил следует придерживаться при верстке и оформлении вложенных структур?
8. Какие группы свойств отвечают за поля, внутренние отступы и границы контейнера элемента?

Практикум

Задание 1

1. Создайте каталог *Разметка блоков* с типовым стартовым HTML-файлом.
2. Используя сокращенные аббревиатуры плагина Emmet, наберите одной командой и завершите следующую разметку (рис. 3.59):

Глава 3\Разметка блоков\index.html

```
<main class="content">
    <nav class="content__menu"></nav>
    <div class="container">
        <div class="row">
            <div class="column"></div>
            <div class="column"></div>
            <div class="column"></div>
        </div>
    </div>
</main>
```

Задание 2

1. Создайте каталог *Разметка блоков* с типовым стартовым HTML-файлом.
2. Изучите и скопируйте приведенный ниже код разметки и стилей. Найдите фоновое изображение большого размера и разместите его в каталоге *images* внутри проекта.
3. Требуется дописать свойства для созданных селекторов согласно подсказкам в комментариях. Образец результата изображен на рис. 3.60.

Глава 3\Социальная инженерия\index.html

```
<div class="main-container">
    <header class="header">
        <h1 class="header__title">Социальная
            инженерия</h1>
    </header>

    <main class="main-box">
        <div class="container">
            <strong>Социальная инженерия</strong> –
            это метод управления действиями человека
```

```
            без использования технических средств.
            Метод основан на использовании слабостей
            человеческого фактора и считается очень
            разрушительным.
        </div>

        <div class="container">
            <p>Техники СИ:</p>
            <ul>
                <li>фишинг;</li>
                <li>плечевой серфинг;</li>
                <li>квид про кво;</li>
                <li>троянский конь;</li>
                <li>сбор информации из открытых
                    источников;</li>
                <li>дорожное яблоко;</li>
                <li>обратная СИ.</li>
            </ul>
        </div>
    </main>
</div>
```

Глава 3\Социальная инженерия\index.html

```
body {
    /* Фоновое изображение */
    background: url('images/fon.jpg');
    font: 1.1em Cambria;
}

/* Основной блок */
.main-container {
    /* Белый прозрычный фон (60% непрозрачности) */
    background: rgba(255,255,255,0.6);
    width: 500px;
    /* Выравнивание содержимого блока по центру */
    margin: 0 auto;
    /* Тень сзади блока */
    box-shadow: 0 0 25px #000;
    /* Округляем углы блока на 5px */
    border-radius: 10px;
}
```

```
.header {
    border-radius: 10px 10px 0 0;
    /* Установить:
       - цвет фона - розовый прозрачный;
       - внутренний отступ со всех сторон - 15px */

}

.header__title {
    /* Установить:
       - шрифт Calibri;
       - белый цвет текста */

    /* Тень текста (для более отчетливых букв) */
    text-shadow: 0 0 5px #000;
}

.main-box {
    /* Установить:
       - внутренний отступ со всех сторон - по 20px */

}

.container {
    border: 2px solid #edfff4;
    /* Установить:
       - внутренние отступы - 10px сверху/снизу и
         15px слева/справа;
       - поле снизу блока - 15px;
       - радиус скругления - 5px;
       - прозрачный белый фон
    */

}
```

Задание 3

1. Самостоятельно изучите одну неочевидную особенность полей (свойства margin), называемую «схлопыванием».
2. Пример веб-ресурса для чтения: webref.ru.
3. Возьмите любой пример ранее выполненного задания с блочной разметкой и найдите места, где эта особенность проявляется. Возможно ли ее контролировать и как от нее избавиться?

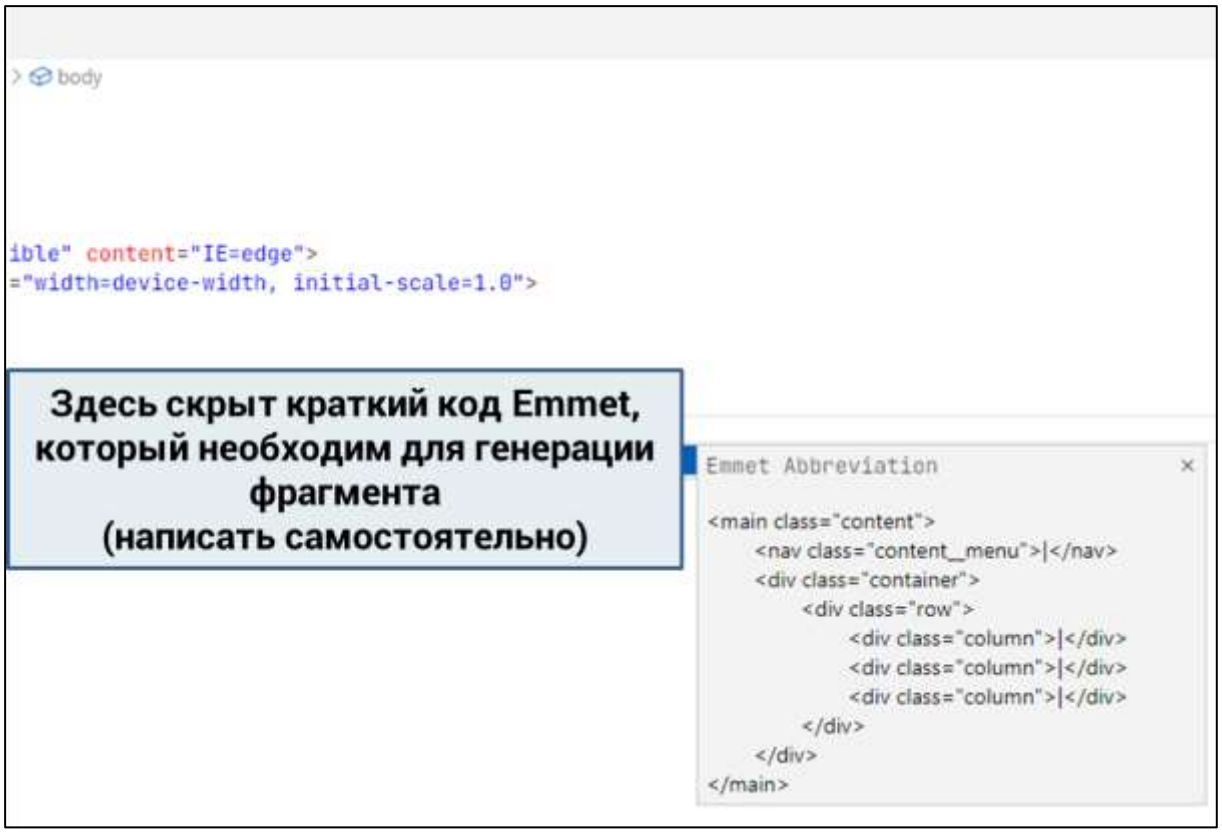


Рис. 3.59. Образец выполнения задания 1

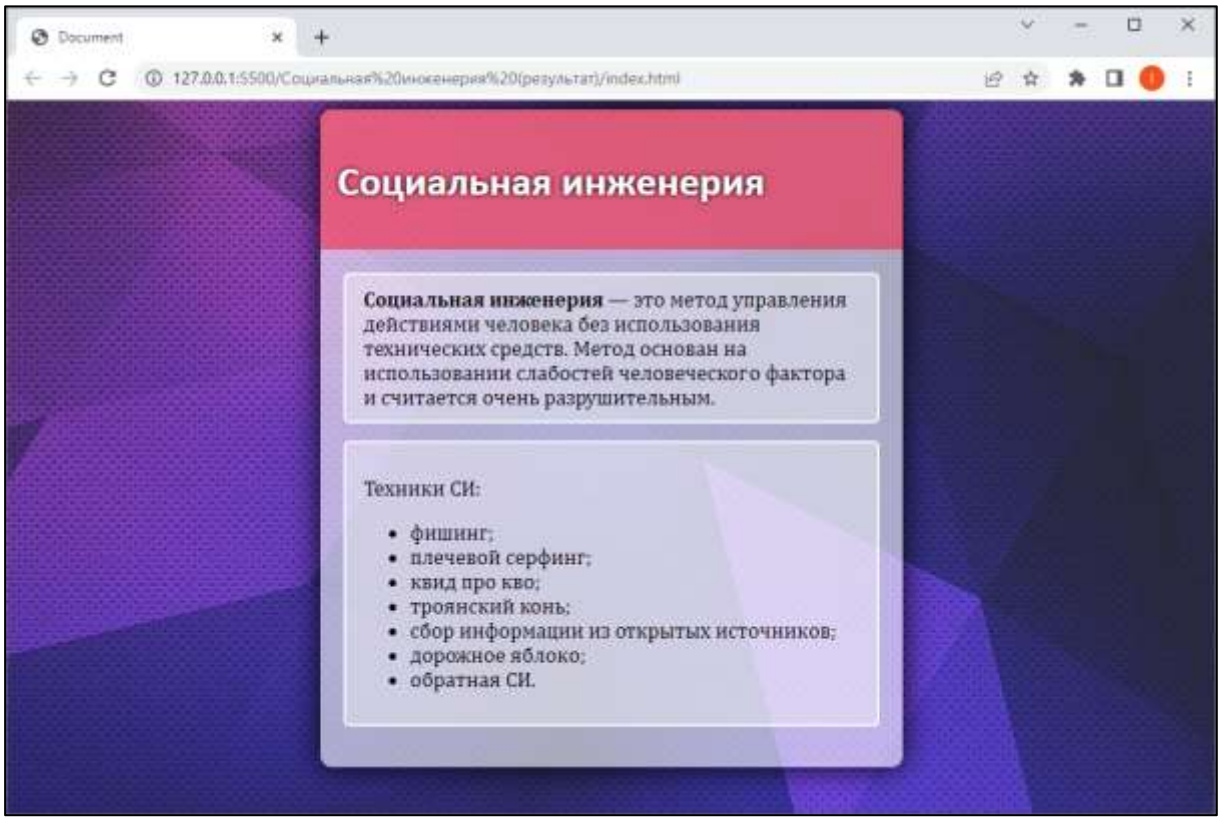


Рис. 3.60. Образец выполнения задания 2

3.4. Настройка типографики текста

3.4.1. Настройка параметров шрифта

Условные обозначения

Важное замечание!

CSS предоставляет достаточно обширную категорию свойств для настройки шрифта, фона и цвета. В рамках текущего курса мы не будем вдаваться в детали, подробно описывая каждое свойство. Развернутое описание свойств и примеры при необходимости читатель найдет в сети Интернет.

Однако, чтобы работать со справочником было проще, примем ряд соглашений по обозначению.

1. Символ **|** означает «ЛИБО» – допустимо только одно из перечисленных значений.
2. Символ **||** означает «ИЛИ» – допустимо одно или более из указанных значений, но в указанном порядке.
3. Необязательные параметры/значения указывают в **[**квадратных скобках**]**.
4. Значение **none** сбрасывает действие свойства.
5. Значение **inherit** означает наследование значения свойства от родительского элемента.
6. Для некоторых групп свойств есть *универсальное*. Например, код

```
body {
    font-family: Arial;
    font-size: 12pt;
    font-weight: bold;
}
```

эквивалентен коду (порядок перечисления значений может быть важен):

```
body { font: bold 12pt Arial; }
```

Свойства шрифта

Отдельные свойства

Определение

`font-family`

*Задает **гарнитуру** шрифта. В нем указывается название шрифта и/или его базовое семейство (рубленный, с засечками, моноширинный).*
Можно перечислить несколько шрифтов через запятую. Тогда, если указанный шрифт не будет найден, браузер ищет следующий. Если название шрифта состоит из нескольких слов, его заключают в кавычки.

`font-size`

*Задает **размер** шрифта. Обычно ему указывают числовое значение в **px**, **em**, **ex** или **rem**, иногда в **%**.*

`font-weight`

*Задает **насыщенность** шрифта («жирность»). Задается текстовым литералом или числом от 100 до 900 с шагом 100.*

`font-style`

*Задает **начертание** шрифта (обычное, курсивное или наклонное).*

`font-variant`

*Задает **модификацию(ии)** шрифта. Универсальное свойство, в CSS3 имеет множество дополнительных свойств-компонент вида **font-variant-**.*

Примеры:

```
font-family: Calibri;
font-family: Consolas, monospace;
font-family: 'Segoe UI', Tahoma, Verdana, sans-serif;
```

```
font-size: 16px;
font-size: 2.05rem;
font-size: 125%;

font-weight: bold;
font-weight: 200;

font-style: italic;
font-style: oblique;

font-variant: small-caps;
font-variant-ligatures: historical-ligatures;
font-variant-position: super;
```

Универсальное свойство

Определение

```
font: [стиль || вариант || насыщенность] размер
      [/ высота_строки] шрифт
```

Универсальное *свойство* ***настройки шрифта****. Высота строки определяет междустрочный интервал, но, вообще говоря, обычно его задают свойством* ***line-height****.*

Минимально свойство `font` должно содержать размер шрифта и его название.

Например:

```
font: 16px Consolas;
font: bold 18px Arial;
font: italic 1.2em Georgia, serif;
font: italic bold small-caps 20px/1.5 Tahoma, serif;
```

Пример использования

Глава 3\Типографика\Шрифты\index.html

```
<p class="cl-1">Работа с шрифтами.</p>
<p class="cl-2">Работа с шрифтами.</p>
<p class="cl-3">Работа с шрифтами.</p>
<p class="cl-4">Работа с шрифтами.</p>
```

Глава 3\Типографика\Шрифты\style.css

```
.cl-1 {
    font-family: Arial;
    font-size: 24px;
    font-style: oblique;
    font-variant: small-caps;
}

.cl-2 {
    font-family: Consolas, Courier, monospace;
}

.cl-3 {
    font-family: Roboto, Tahoma;
    font-size: 1.2em;
    font-weight: bold;
}

.cl-4 {
    font: bold 1.2em Roboto, Tahoma;
}
```

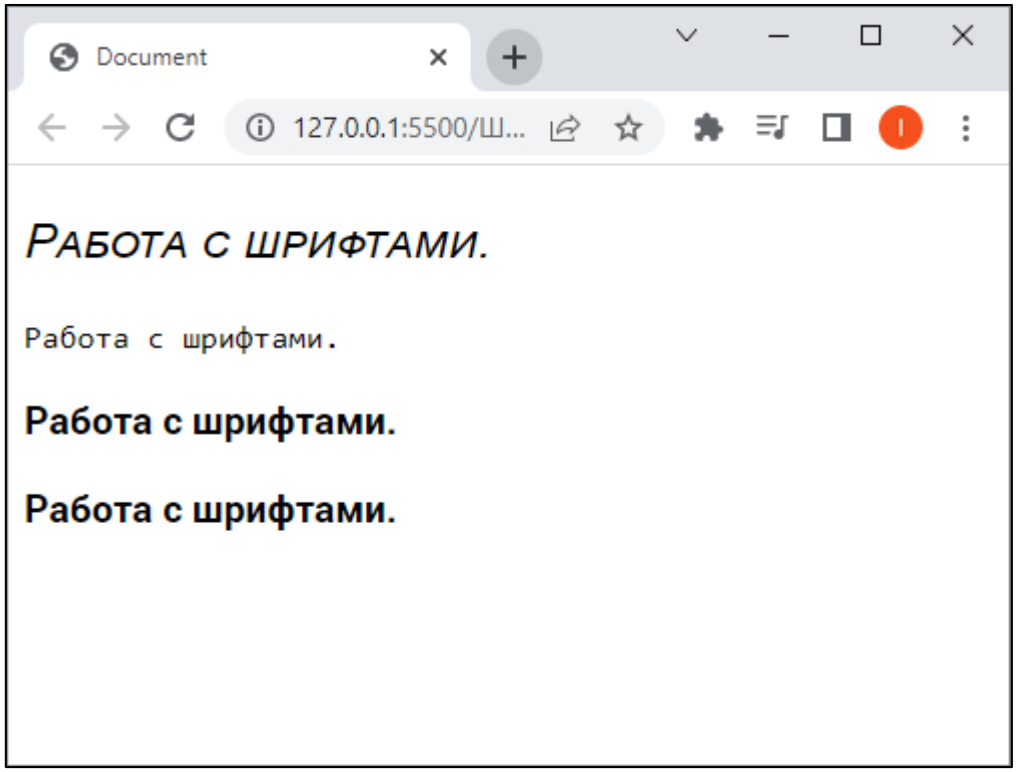


Рис. 3.61. Пример настройки шрифтов

3.4.2. Настройка параметров абзаца

Параметры текста

Основные свойства

Определение

`text-align`

*Определяет **режим выравнивания** текста. По умолчанию текст выравнивается по левому краю.*

`line-height`

*Устанавливает **междустрочный интервал**. Можно указать размерной величиной либо коэффициентом.*

`text-indent`

*Задает **отступ красной строки**. В верстке веб-страниц применяется редко.*

`text-decoration`

*Меняет **оформление** текста (**декорирование**). В частности, поможет убрать подчеркивание у гиперссылок, которые нужно оформить в виде кнопок (установите в нем значение **`none`**).*

`text-transform`

*Управляет **преобразованием** текста в заглавные или прописные символы.*

`text-overflow`

*Управляет **видимостью** текста в блоке, если он целиком не помещается в него. Может просто обрезается или обрываться троеточием. Работает в паре со свойством **`overflow`**, которое управляет показом содержимого блока в случае переполнения.*

Примеры:

```
text-align: center;

line-height: 1.25em;
line-height: 1.5;    /* полуторный интервал */

text-indent: 2.0rem;

text-decoration: double;
text-decoration: none;

text-transform: uppercase;
text-transform: capitalize;

text-overflow: ellipsis;
```

Это полезно знать!

Свойства этой группы не содержат краткой формы, как у ***font****.*

Пример использования

Глава 3\Типографика\Текст\index.html

```
<p class="cl-1">
    Выравнивание по левому краю и красная
    строка в 20px.
</p>
<p class="cl-2">
    Выравнивание по центру и междустрочный
    интервал 2.5.
</p>
<p class="cl-3">
    Выравнивание по правому краю,
    подчеркивание и заглавные буквы.
</p>
<a href="#" class="cl-4">
    Гиперссылка без подчеркивания
</a>
```

Глава 3\Типографика\Текст\style.css

```
.cl-1 {
    text-indent: 20px;
}

.cl-2 {
    text-align: center;
    line-height: 2.5;
}

.cl-3 {
    text-align: right;
    text-decoration: underline;
    text-transform: uppercase;
}

.cl-4 {
    text-decoration: none;
}
```

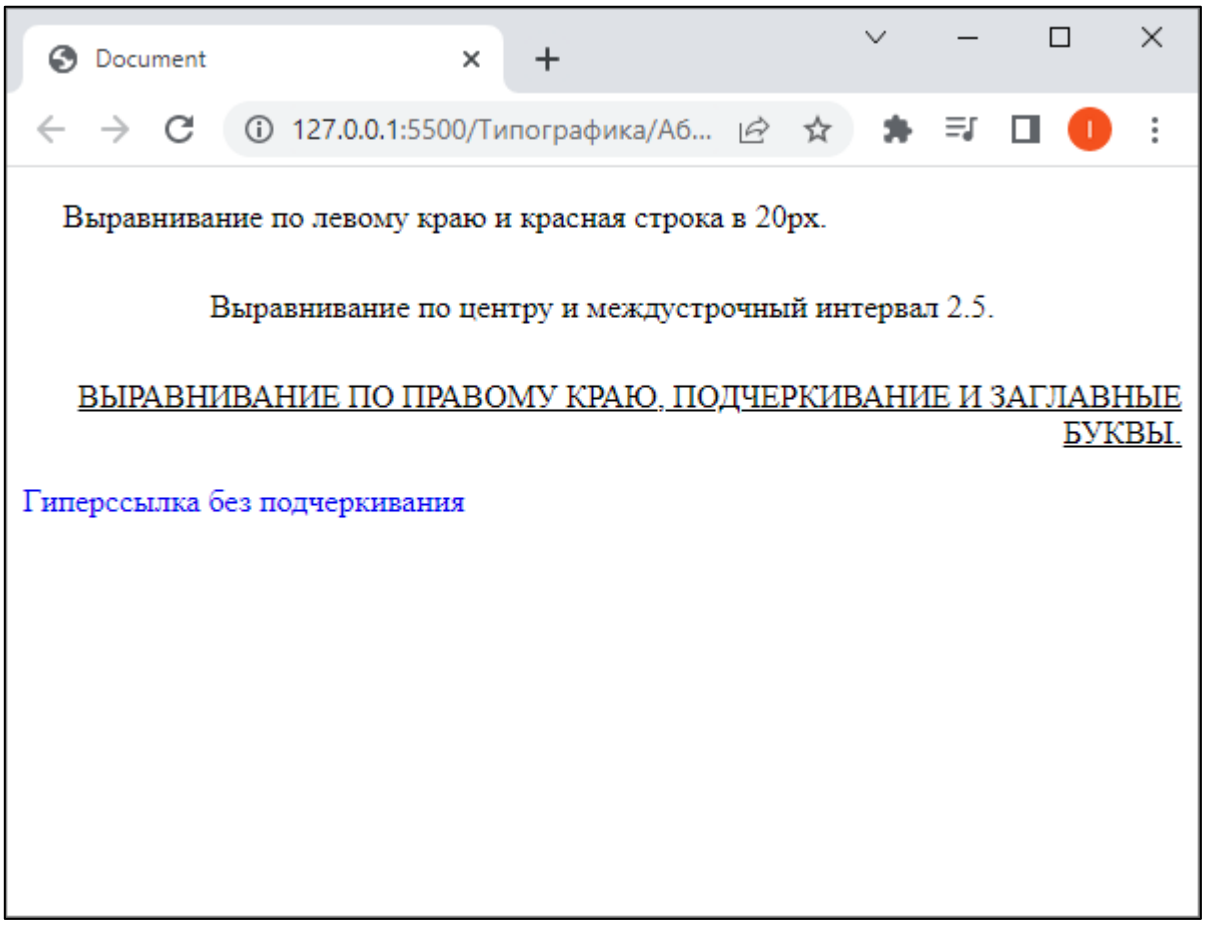


Рис. 3.62. Пример настройки текста

Настройка списков

Основные свойства

Определение

`list-style-type`

*Определяет **тип маркера** списка. При значении **`none`** маркер будет удален.*

`list-style-image`

*Вместо маркера подставляет **изображение** или **градиентную заливку**. Для вставки изображения используется функция **`url()`**.*

`list-style-position`

*Устанавливает **режим позиции маркера**: с отступом или выступом абзаца пунктов.*

Примеры:

```
list-style-type: circle;
list-style-type: upper-roman;

list-style-position: inside;

list-style-position: url('icons/funny.png');
```

Универсальное свойство

Все три свойства можно объединить в одном: **`list-style`**:

```
list-style: upper-roman
            linear-gradient(#abf051 0, #ffe868 20%)
            outside;
```

Пример использования

Глава 3\Типографика\Списки\index.html

```
<h1>Маркированный список</h1>
<ul class="new-ul-type">
    <li>Вы узнаете, что такое frontend и что входит в его
        основу.</li>
    <li>Познакомитесь с языком разметки HTML5.</li>
    <li>Изучите фундаментальные возможности и особенности
        стилизации веб-страниц с помощью CSS3.</li>
    <li>Научитесь верстать веб-страницы с использованием
        профессионального текстового редактора.</li>
    <li>Познакомитесь с современным и гибким языком
        программирования JavaScript.</li>
    <li>Отработаете свои навыки в программировании задач
        и реализации небольших проектов.</li>
</ul>

<h1>Нумерованный список</h1>
<ol class="new-ol-type">
    <li>Вы узнаете, что такое frontend и что входит в его
        основу.</li>
    <li>Познакомитесь с языком разметки HTML5.</li>
    <li>Изучите фундаментальные возможности и особенности
        стилизации веб-страниц с помощью CSS3.</li>
    <li>Научитесь верстать веб-страницы с использованием
        профессионального текстового редактора.</li>
    <li>Познакомитесь с современным и гибким языком
        программирования JavaScript.</li>
    <li>Отработаете свои навыки в программировании задач
        и реализации небольших проектов.</li>
</ol>
```

Глава 3\Типографика\Списки\style.css

```
body {
    font-size: 18px;
}

ul.new-ul-type {
    list-style-type: square;
    list-style-position: inside;
    list-style-image: linear-gradient(#ec691d 0, #ffba2f
50%);
}
```

```css
ol.new-ol-type {
    list-style: lower-greek outside;
    padding-left: 2em;
}

ul > li,
ol > li {
    margin-bottom: 10px;
    padding-left: 15px;
}
```

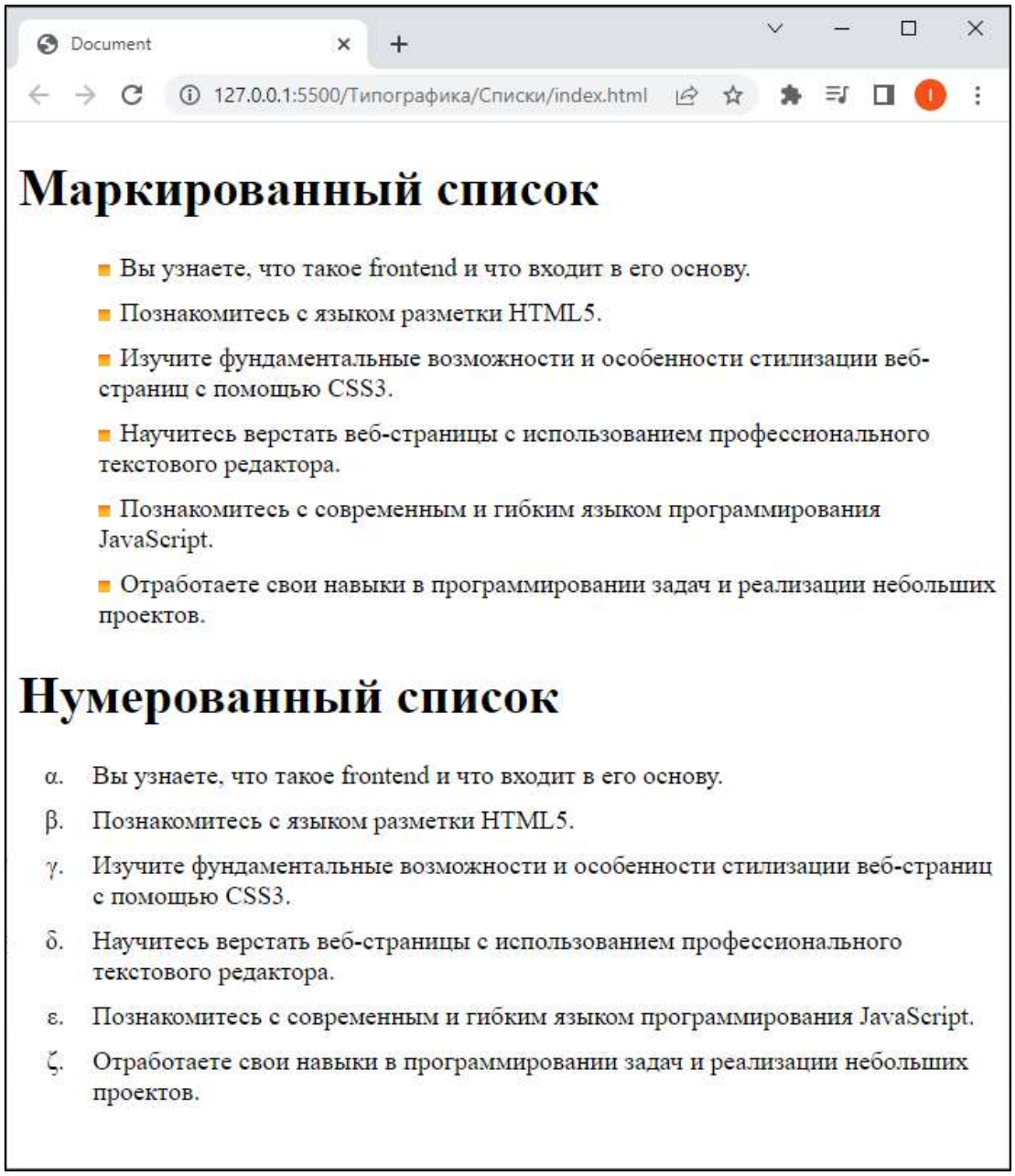


Рис. 3.63. Пример оформления списков с помощью CSS

3.4.3. Настройка цвета текста и фона

Модели цвета

В CSS используется два основных свойства, управляющих цветом:

- **`color`** отвечает за цвет текста;
- **`background-color`** (или краткое **`background`**) – за цвет фона блока.

CSS поддерживает разные модели представления цветов. Каждая из них имеет обычную форму и *альфа-канал*, т.е. поддерживает прозрачность цвета или заливки. Подробно остановимся на модели RGB и HEX; особенности остальных читатель при желании сможет разобрать самостоятельно.

Табличные

Задаются строковыми литералами. Точные названия цветов можно найти в онлайн справочниках, например: colorscheme.ru.

```
color: aqua;
background: darkgray;
```

RGB и RGBA

Классическая модель смешения красной, зеленой и голубой компоненты. Каждая из них может варьироваться от 0 до 255.

Для определения RGB-цвета используется функция **`rgb()`**:

```
rgb(42, 44, 128);
```

При перечислении параметров в CSS3 их допускается не отделять запятыми:

```
rgb(42 44 128);
```

Дополнительно можно добавить через **`/`** коэффициент непрозрачности в пределах от 0 до 1 (0 – прозрачный, 1 – непрозрачный):

```
rgb(127 20 131 / 0.4);
```

Наконец, для работы с прозрачностью можно использовать функцию **`rgba()`**. Четвертый параметр задает процент непрозрачности. Следующие варианты дают эквивалентные цвета:

```
color: rgb(127 20 131 / 0.4);
color: rgba(127 20 131 40%);
color: rgba(127, 20, 131, 40%);
```

HEX и HEXA

Представляет собой шестнадцатеричную форму RGB- и RGBA-цвета. Записывается единой константой с символом **#** в начале.

Для RGB-модели цвет задается в формате `#RRGGBB` или в случае равных групп более короткой `#RGB`.

Для RGBA-модели две последние шестнадцатеричные цифры определяют степень прозрачности, а запись задается в форме `#RRGGBBAA`.

Следующие записи попарно эквивалентные:

```
color: rgb(42, 44, 128);
color: #2a2c80;

color: rgba(42, 44, 128, 70%);
color: #2a2c80b3
```

HSL и HSLA

Модель HSL задает цвета по следующим трем критериям:

- Hue (оттенок);
- Saturation (насыщенность);
- Lightness (светлота).

Примеры записи:

```
color: hsl(106, 81%, 58%);
color: hsla(106, 81%, 58%, 0.45);
```

Рис. 3.64. Параметры модели HSL

HWB и HWBA

Также является трехкомпонентной моделью:

- Hue (оттенок);
- Whiteness (количество белого);
- Blackness (количество чёрного).

Примеры записи:

```
color: hwb(106 24% 8%);
color: hwb(106 24% 8% / 0.45);
```

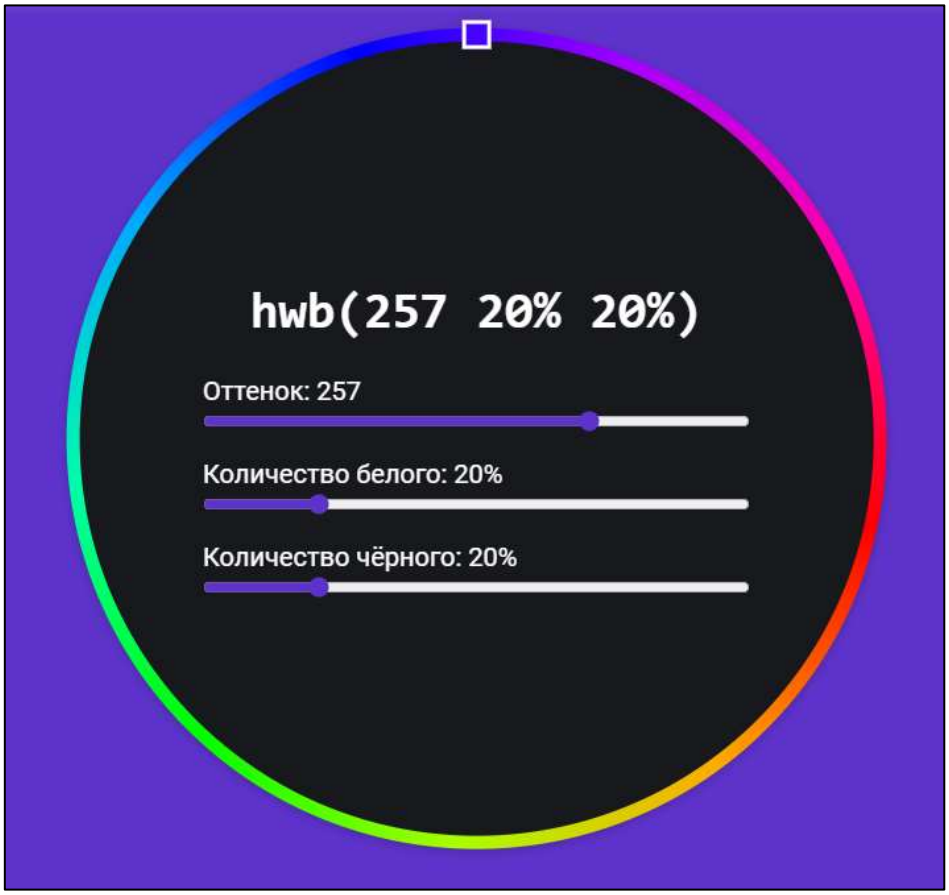


Рис. 3.65. Модель HWB

Дополнительная информация по моделям представления цветов в CSS3: doka.guide/css/web-colors/.

Цвет текста и фона

Основные свойства

Определение

`color`

*Задает **цвет текста**.*

Определение

`background-color`

*Задает **цвет фона**. Кроме сплошной заливки допускается и градиентная.*

`background-image`

*Устанавливает **фоновое изображение**. Использует функцию **`url()`**.*

`background-position`

*Управляет **смещением фонового изображения** относительно верхнего левого угла.*

`background-repeat`

*Настраивает **режим повтора изображения** плиткой (если изображение меньше блока).*

`background-attachment`

*Определяет, будет ли **прокручиваться** изображение. Чтобы оно стало неподвижным, используется значение **`fixed`**.*

`background-size`

*Позволяет **масштабировать** изображение.*

Примеры:

```
background-color: azure;
background-image: url('images/fon.jpg');
background-position: center center;
background-repeat: repeat;
background-attachment: fixed;
background-size: cover;
```

Универсальное свойство

Определение

```
background: цвет || изображение
            [повтор || скроллинг || положение];
```

Универсальное свойство, управляющее ***оформлением фона****.*

Минимально свойству **background** достаточно указать цвет или фоновое изображение.

Пример использования

Глава 3\Типографика\Фон и цвет\index.html

```
<div class="main">
    <h1>Цвет и фон</h1>
    <p>Всего существует два основных CSS свойства,
    отвечающих за установку цвета: свойство
    <strong>color</strong> отвечает за цвет текста, а
    <strong>background-color</strong> (или краткое
    background) – за цвет фона объекта.</p>
</div>
```

Глава 3\Типографика\Фон и цвет\style.css

```
body {
    background: url('images/fon.jpg') fixed;
    font: 18px Roboto, Arial;
    color: darkslategrey;
}
```

```css
.main {
    width: 580px;
    margin: 0 auto;
    background: rgba(234, 253, 255, 0.3);
    padding: 20px;
}

h1 {
    color: #fff;
    text-shadow: 2px 1px 3px #00000083;
}

strong {
    font-family: 'JetBrains Mono', Consolas, monospace;
    color: brown;
}
```

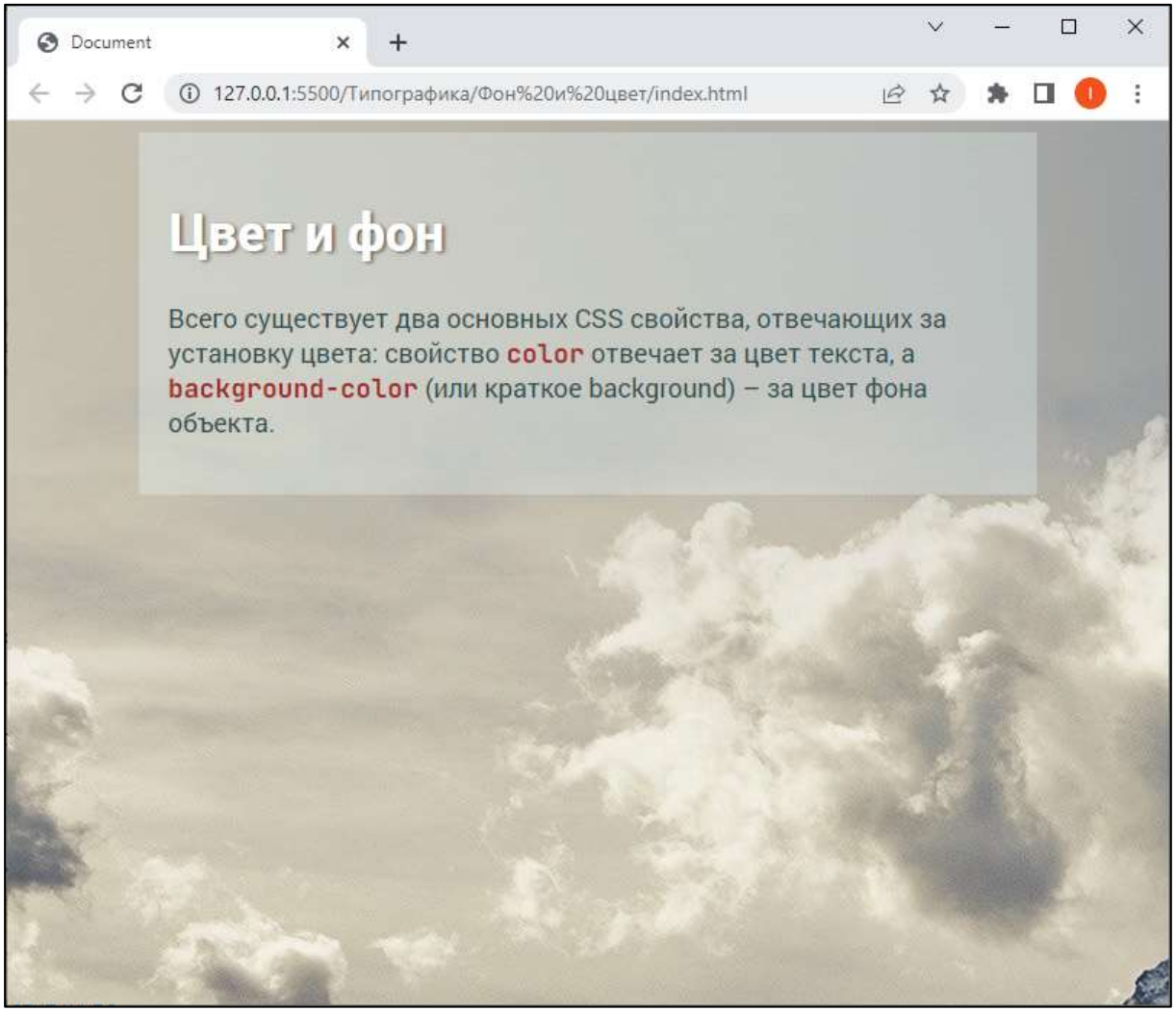


Рис. 3.66. Пример работы с цветом текста и фона

3.4.4. Инспектор кода в браузере

Инструменты разработчика в современных браузерах

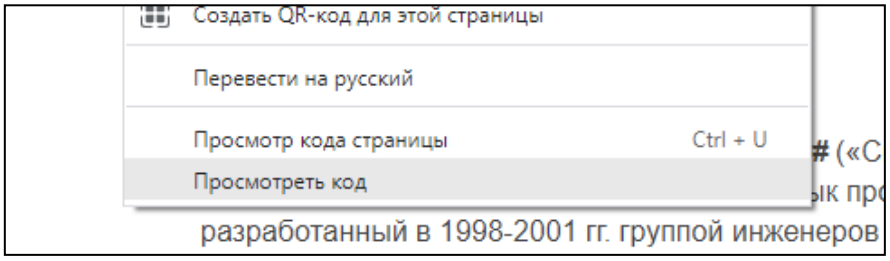
В каждый современный браузер встроены *инструменты разработчика*, позволяющие изучить код HTML-разметки, CSS-стилей, JavaScript-скриптов, отловить и исправить ошибки и многое другое.

Мощный инспектор кода поддерживают такие браузеры, как Google Chrome, Mozzila Firefox, Яндекс браузер, Opera.

Инструмент разработчика в Google Chrome

Одна из наиболее продвинутых консолей разработчика доступна в браузере Google Chrome.

Для вызова инспектора кода на веб-странице нажмите *ПКМ / Просмотреть код*:



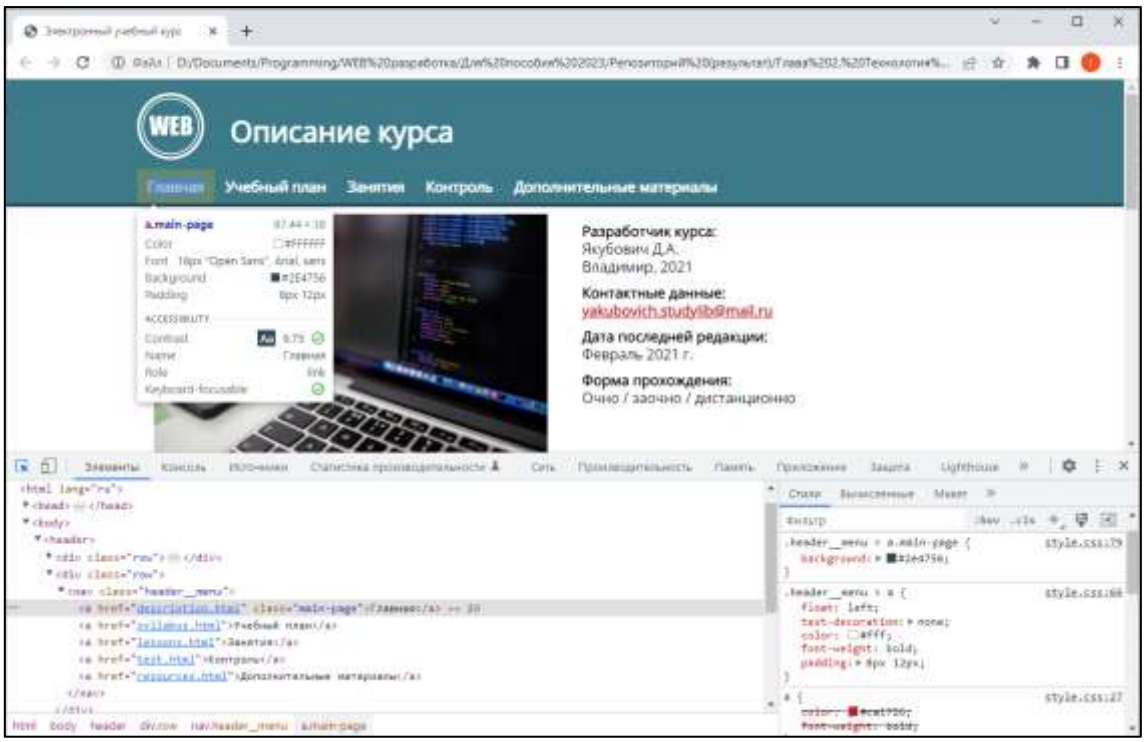


Рис. 3.67. Инспектор кода в Google Chrome

На вкладке *Элементы* отображается HTML и CSS код:

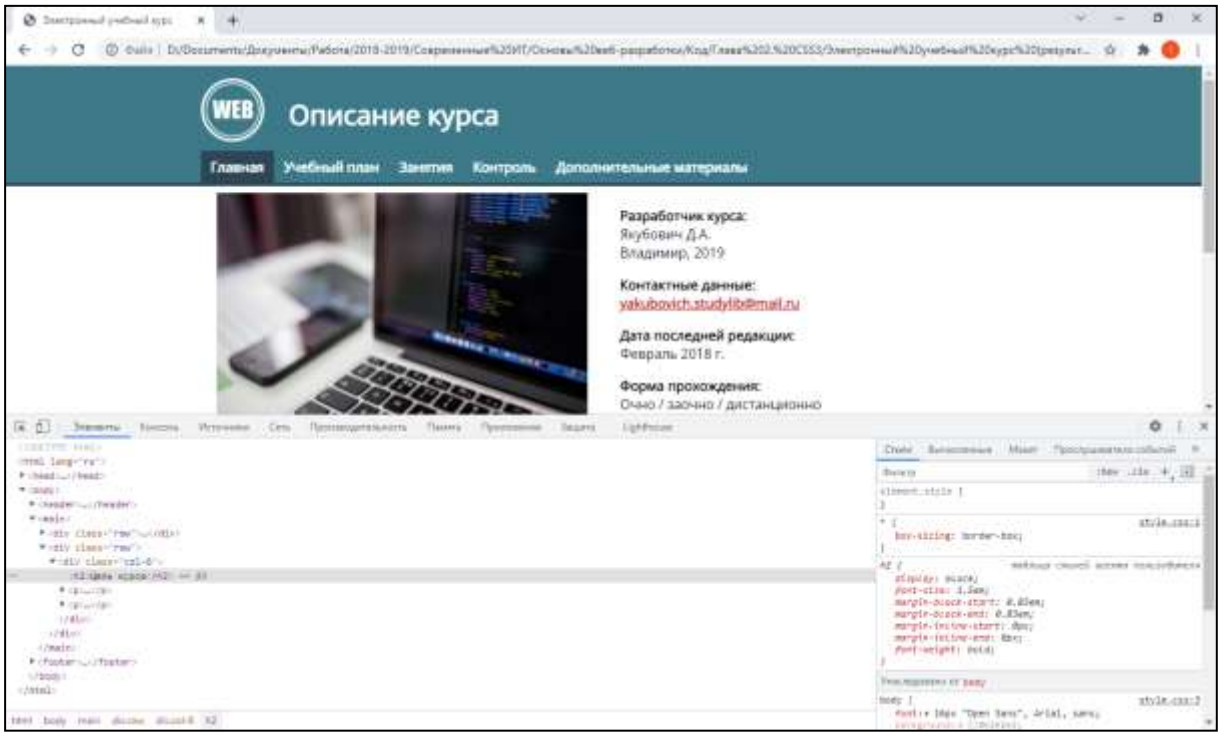


Рис. 3.68. Просмотр структуры HTML

Щелкая на элемент в HTML-разметке, открываются и связанные с ним CSS-стили:

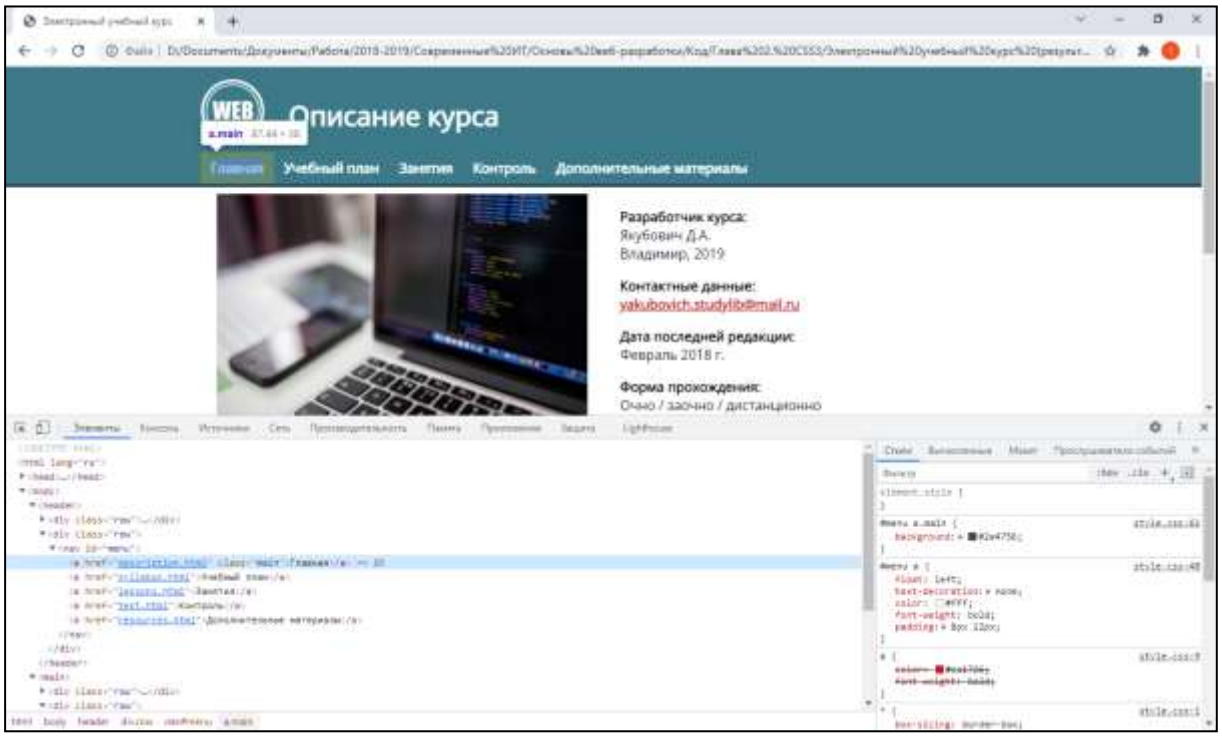


Рис. 3.69. Просмотр CSS-свойств

Щелкая на элемент *ПКМ / Редактировать как HTML* можно внести локальные изменения в разметку:

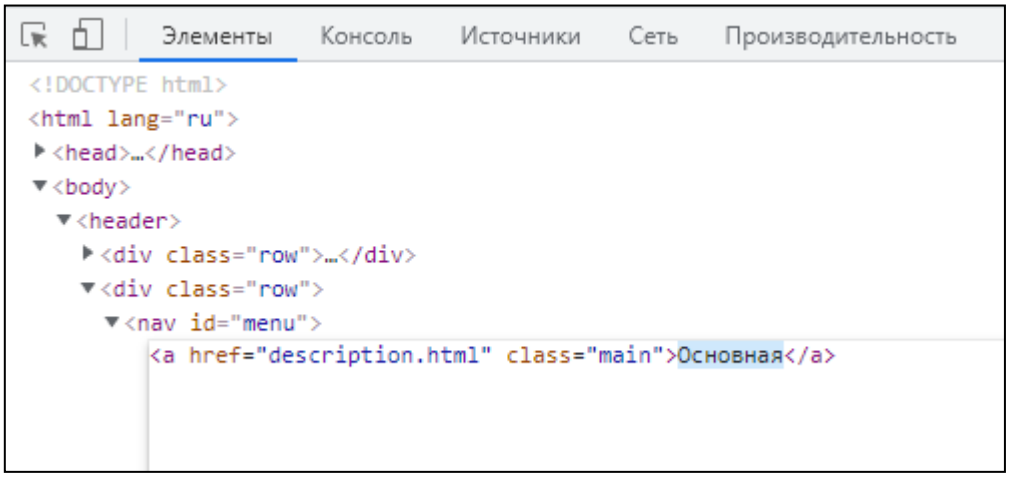


Рис. 3.70. Правка HTML-кода тега и его содержимого

Можно проследить за свойствами конкретного элемента, например, отступами и полями:

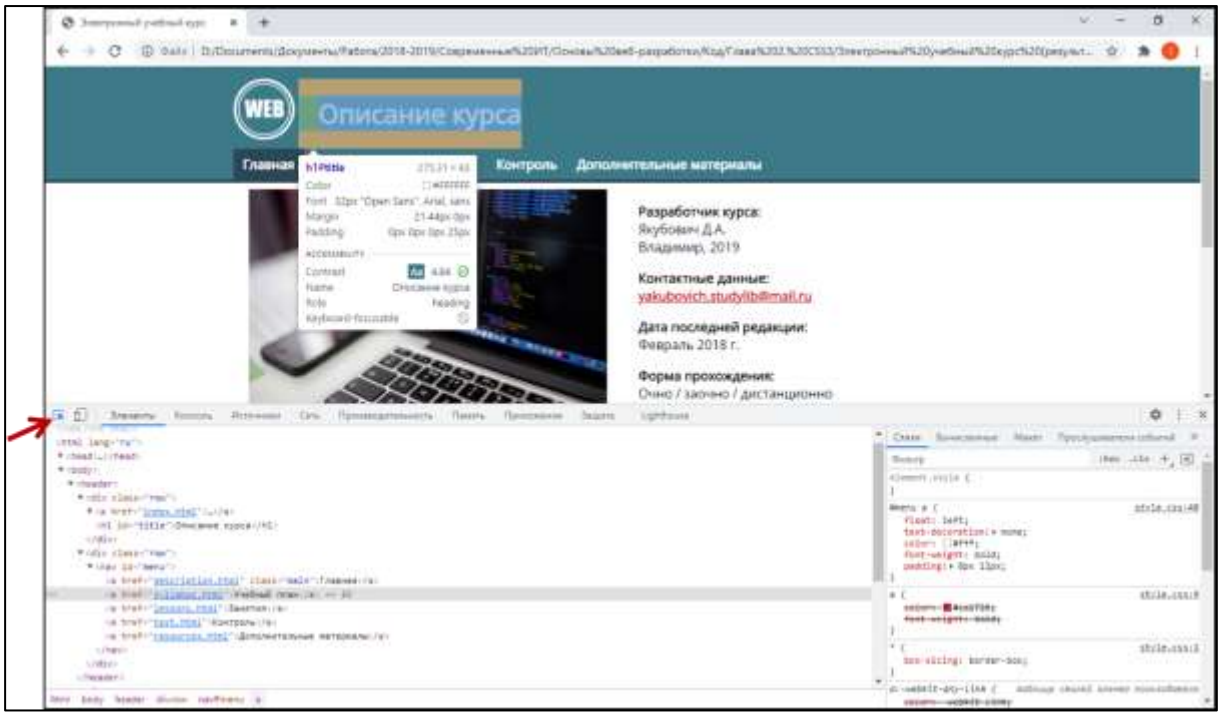


Рис. 3.71. Просмотр позиции элемента в разметке и его оформление

На вкладке *Стили* можно последовательно отключать и включать отображение стилей объекта. Это позволяет проследить за его видоизменением.

Зачеркивание стиля также обозначает, что в силу правил каскадирования свойства перезаписаны более приоритетными стилями:

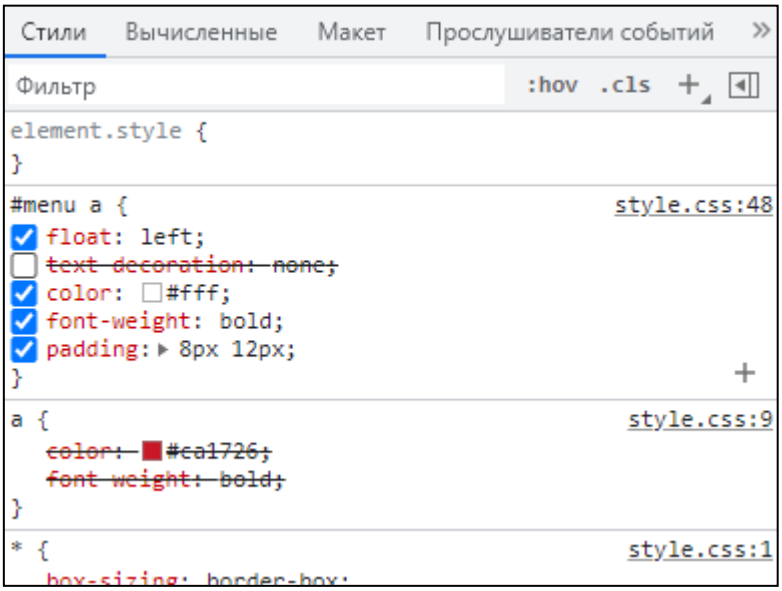


Рис. 3.72. Включение и отключение CSS-стилей

При необходимости значения свойств можно вручную исправлять. Либо добавлять новые свойства.

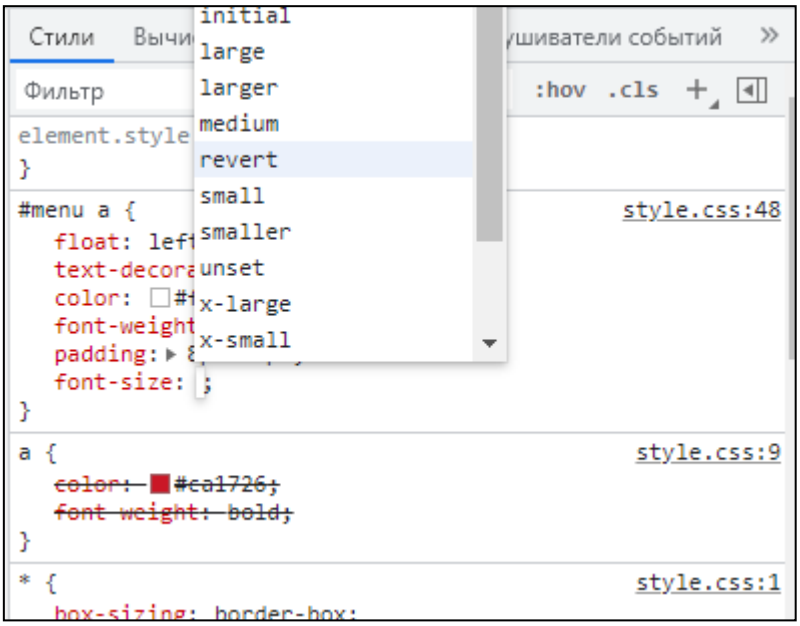


Рис. 3.73. Редактирование CSS-стилей

На вкладке *Вычисленные* визуально представлены размеры полей, границ, отступов и содержимого элемента. Можно выделить нужную часть и внести другие значения:

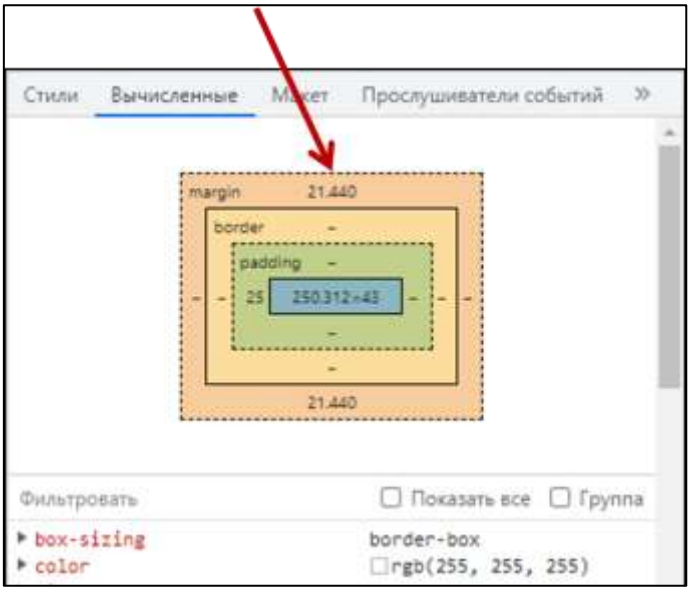


Рис. 3.74. Управление полями, отступами и рамкой

На вкладке *Источники* отображается проводник по файлам проекта: веб-страницы и приложения. Можно прямо на месте просмотреть, например, изображения:

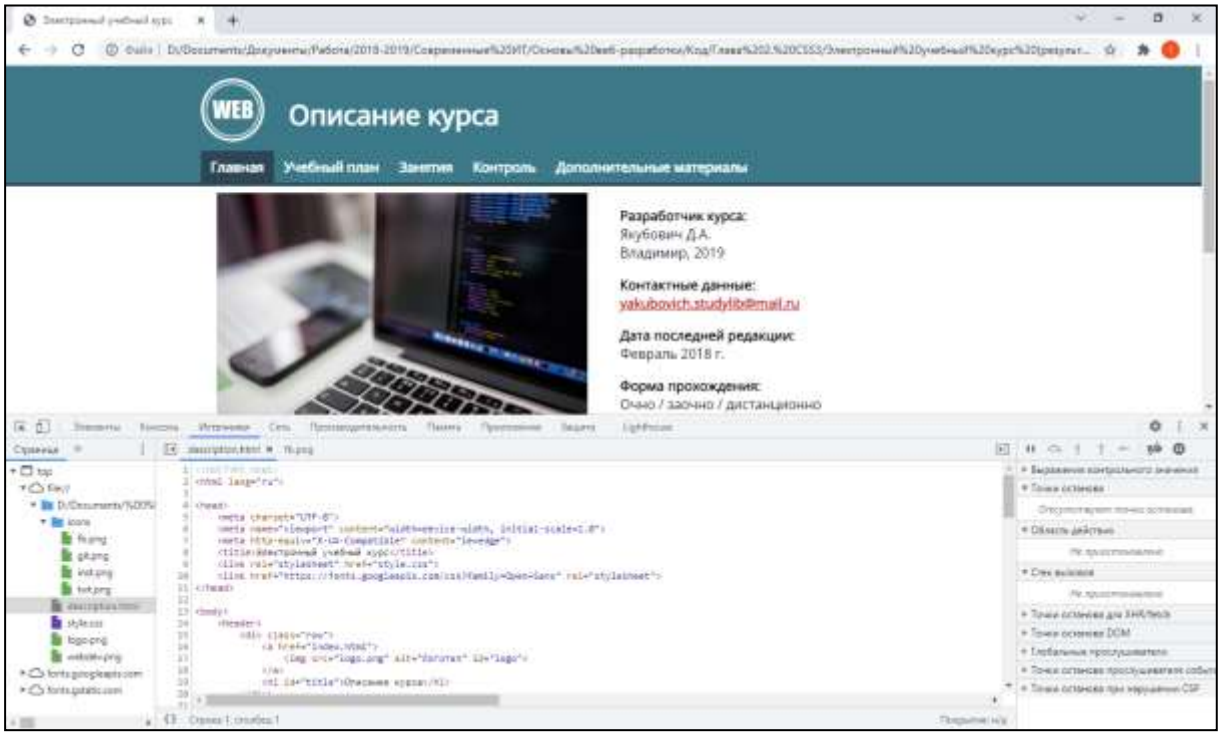


Рис. 3.75. Структура проекта

На вкладке *Консоль* доступна консоль отладки. Здесь можно проверить работу JavaScript-кода, а также отследить, допущены ли ошибки или недочеты в коде.

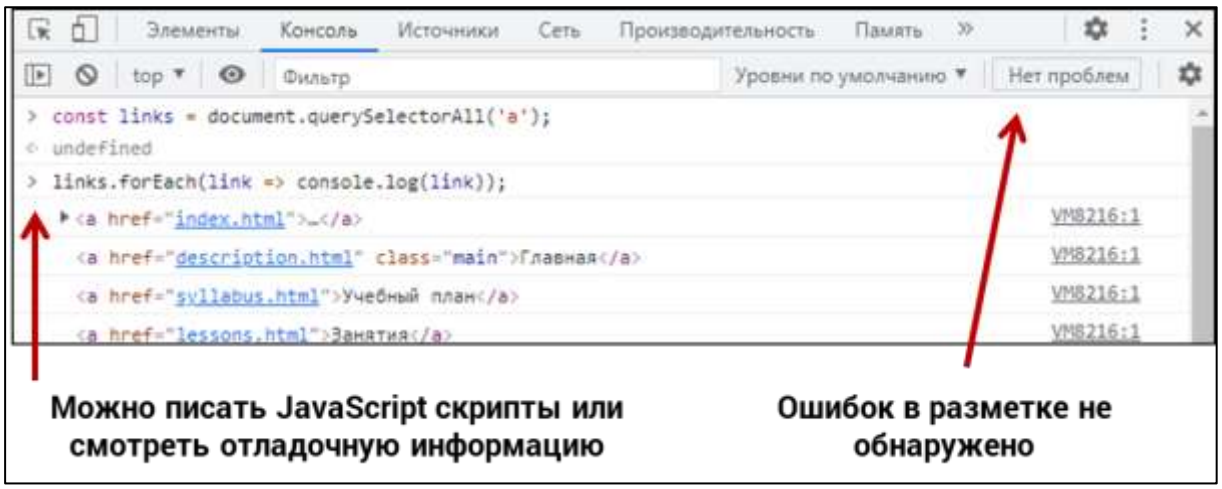


Рис. 3.76. Консоль для отладки JavaScript-кода

Это полезно знать!

Любые изменения в инспекторе кода не меняют искомые файлы! Обновив страницу, все внесенные изменения сбрасываются. При необходимости инспектор все-же позволяет перезаписать файл(ы), однако обычно изменения вносятся в редакторе.

Инспектор кода – отличный инструмент начинающего верстальщика сайтов. Изучайте с его помощью структуру различных сайтов и креативное оформление элементов, так вы быстрее освоите CSS и подчерпнете различные приемы верстки и дизайна.

Для продвинутого разработчика инспектор кода позволит быстро проанализировать ошибки на сайте.

Также инспектор кода поможет вам внедрить красочно оформленные HTML-письма в почтовых сервисах mail.ru, yandex.ru и др. Этот прием будет разобран далее.

3.4.5. Верстка электронных писем

HTML-письма

Технологии HTML и CSS можно использовать не только для верстки веб-сайтов и веб-документов. Они являются прекрасным инструментом для разметки *электронных писем* в почтовых онлайн сервисах, таких как Mail.ru, Yandex.ru др.

Умение верстать HTML-письма является крайне полезным навыком силу ряда причин:

- почтовые сервисы обычно обладают скромными возможностями редактирования и ограничиваются лишь простыми функциями разметки и оформления текста;
- HTML-письмо является отличным средством, если необходимо создать и разослать по адресатам качественно оформленное объявление, визитку, поздравление, рекламу и т.п.

Возможность создания HTML-писем в почтовых веб-сервисах связана с тем, что инспектор кода в браузере позволяет вставлять фрагмент разметки пользователя в блок письма и отправлять его уже как часть кода веб-страницы.

Некоторые проблемы при верстке писем

Однако верстка писем является специфической задачей, имеющей множество особенностей и ограничений, которые необходимо знать, чтобы избежать нежелательного форматирования письма или непредвиденных проблем. Это связано с тем, что внедряемый пользователем код может частично заимствовать CSS-стили самого веб-сервиса и смешивать их со стилями из письма пользователя.

Более того, в письме нет возможности использовать связанные стили (т.е. отделять HTML от CSS): браузер просто их вырезает, как и код скриптов, чтобы избежать потенциально небезопасных операций с данными. Единственный рабочий вариант – использование inline-стилей, описанных в атрибуте **`style`** каждого тега.

Немаловажной проблемой является верстка блоками. Поскольку код разметки пользователя вставляется как внешние данные, то возможны противоречия в оформлении. Чтобы минимизировать подобные недочеты, важно грамотно усиливать приоритетность пользовательских стилей.

Это полезно знать!

Как ни парадоксально, обычно для писем вместо современной блочной верстки используют устаревшую табличную. Таблицам можно задавать разные атрибуты размера и оформления прямо в теге и они не будут нарушать CSS. А вот для блочной разметки потребуется писать несколько избыточный код.

Практический пример

1. Описание задачи

В качестве примера создадим письмо-приглашение студентов на участие в Олимпиаде по программированию (рис. 3.87).

Прежде всего, начнем с базовой разметки:

Глава 3\HTML-письма\index.html

```
<!DOCTYPE html>
<html lang="ru">

<head>
    <meta charset="UTF-8">
    <meta http-equiv="X-UA-Compatible" content="IE=edge">
    <meta name="viewport" content="width=device-width,
initial-scale=1.0">
    <title>Document</title>
</head>

<body>

</body>

</html>
```

Как было отмечено ранее, при внедрении разметки в письмо:

- доступны только inline-стили (внутри тегов);
- необходимо усилить приоритеты собственных стилей, чтобы они не перекрывались стилями почтового сервиса.

2. *Разметка*

Для начала разметим страницу, используя внутреннее описание стилей классов. Далее воспользуемся веб-сервисом, который самостоятельно преобразует CSS-классы в inline-стили.

Первым шагом создадим контейнер, который будет содержать разметку нашего письма. Чтобы разметка была однообразной и можно было отличить наши стили от других, название всех классов будем начинать с префикса «email-» (хотя это необязательно).

При этом основной блок-контейнер оформим не классом, а через идентификатор. Это важно, поскольку селектор идентификатора имеет более высокий уровень приоритета, чем селекторы классов:

Глава 3\HTML-письма\index.html

```
<!DOCTYPE html>
<html lang="ru">

<head>
    <meta charset="UTF-8">
    <meta http-equiv="X-UA-Compatible" content="IE=edge">
    <meta name="viewport" content="width=device-width,
initial-scale=1.0">
    <title>Document</title>
    <style>
        #email-styles {

        }
    </style>
</head>

<body>
    <div id="email-styles">

    </div>
</body>

</html>
```

Реализуем разметку блока `email-styles` (см. рис. 3.87). В него войдет блок заголовка, два блока в виде рамочки и текст. Внизу разместим декоративно оформленные гиперссылки.

При описании каждого элемента используем класс, который будет его оформлять. Названия этих классов продолжаем задавать, используя префикс «email-».

Здесь также важно заметить, что необходимо полностью избегать семантических тегов основной структуры документа (таких, как `<header>`, `<main>`, `<footer>`, `<nav>` и прочее), поскольку они используются в разметке веб-интерфейса почтового сервиса, а это далее может привести к коллизиям стилей.

Всю разметку блоков выстраиваем с помощью контейнера `<div>` и подключаемых классов. Семантические теги разметки текста внутри допустимы, но каждый также желательно сопровождать классом.

Глава 3\HTML-письма\index.html

```
<div id="email-styles">
    <div class="email-header">
        <h1 class="email-title">
            Олимпиада по программированию
        </h1>
    </div>

    <div class="email-main">
        <div class="email-box">
            <h2 class="email-h2">
                Секция теоретической и прикладной
                Информатики
            </h2>
            <p class="email-par">Уважаемые студенты!</p>
            <p class="email-par">8 апреля пройдет
            Олимпиада по программированию среди студентов
            ФМОиИТ 1-3 курсов. Вы можете принять участие
            в ней. Победители получат сертификаты об
            участии и дополнительные баллы к
            дисциплине.</p>
            <p class="email-par">Тематика задач:</p>
            <ul class="email-ul-list">
                <li>циклические алгоритмы;</li>
                <li>поиск в одномерных и двумерных
                    массивах;</li>
                <li>анализ строк;</li>
                <li>задачи с элементами математических
                    преобразований;</li>
                <li>рекурсивные алгоритмы.</li>
```

```
            </ul>
            <img
src="https://c1.wallpaperflare.com/preview/595/1003/783/c
ode-coder-coding-computer.jpg" alt="" class="email-big-
img">
        </div>

        <div class="email-box">
            <h2 class="email-h2">Полезные ссылки</h2>
            <a href="#" class="email-ref-button">
                Регистрация
            </a>
            <a href="#" class="email-ref-button
                              email-button-red">
               Информация
            </a>
            <a href="#" class="email-ref-button
                              email-button-green">
               Сайт кафедры
            </a>
        </div>
    </div>
</div>
```

Кратко опишем назначение классов, которые будут отвечать за оформление блоков и элементов:

- `email-header` – оформляет блок заголовочной части (аналог `<header>`);
- `email-title` – задает основной заголовок;
- `email-main` – размечает блок контента (аналог `<main>`);
- `email-box` – декоративный блок, группирующий некоторую информацию и рисующий рамку с заливкой;
- `email-h2` – подзаголовок;
- `email-par` – текстовый абзац;
- `email-ul-list` – контейнер для разметки маркированного списка;
- `email-ref-button` – оформление ссылок в виде кнопок, дополнительные классы `email-button-red` и `email-button-green` перезаливают стандартный цвет ссылки;
- `email-big-img` – оформляет большое изображение.

3. Стилизация

Приступим к описанию стилей каждого класса. Для этого мы выделили раздел `<style>` в преамбуле документа:

Глава 3\HTML-письма\index.html

```
<head>
    <meta charset="UTF-8">
    <meta http-equiv="X-UA-Compatible" content="IE=edge">
    <meta name="viewport" content="width=device-width,
initial-scale=1.0">
    <title>Document</title>
    <style>
        #email-styles {

        }
    </style>
</head>
```

Начнем с описания основного блока-контейнера нашего письма. Он реализован идентификатором, чтобы задать более высокий приоритет стилям. Определим блоку минимальную и максимальную ширину, выровняем по центру и зададим глобально шрифт:

Глава 3\HTML-письма\index.html

```
<style>
    #email-styles {
        min-width: 320px;
        max-width: 720px;
        margin: 0 auto;
        font: 16px 'Segoe UI', Tahoma, Verdana,
            sans-serif;
    }
</style>
```

Далее задаем стиль, который поменяет модель расчета размеров в CSS (по умолчанию в ширину включены только ширина контента, а отступы контейнера элемента слева и справа игнорируются, что не очень удобно). Воспользуемся селектором `*`, который означает буквально любой тег (распространяем правило на все теги).

Кроме того, здесь и далее мы усиливаем стили за счет селектора идентификатора `email-styles`, показывая вложение в этот блок (**пробел**, в отличие от `>`, означает любое число промежуточных тегов-контейнеров):

Глава 3\HTML-письма\index.html

```
<style>
    <!- Выше предыдущие стили ->

    #email-styles * {
        box-sizing: border-box;
    }
</style>
```

Далее оформляем блок заголовка и сам заголовок внутри:

Глава 3\HTML-письма\index.html

```
<style>
    <!- Выше предыдущие стили ->

    #email-styles .email-header {
        background: #c9e9ff;
        border-radius: 8px;
    }

    #email-styles .email-title {
        color: #191986;
        margin: 0;
        padding: 10px;
    }
</style>
```

Переходим к оформлению стилей блока содержимого. Сам блок в целом не требует оформления, укажем этот селектор чисто формально:

Глава 3\HTML-письма\index.html

```
<style>
    <!- Выше предыдущие стили ->

    #email-styles .email-main {

    }
</style>
```

Блок с рамкой потребует светлой заливки и скругления углов, а также небольшого внутреннего отступа:

Глава 3\HTML-письма\index.html

```
<style>
```

```
        <!- Выше предыдущие стили ->

        #email-styles .email-box {
            background: #fafdff;
            padding: 10px;
            border: 2px solid #c9e9ff;
            margin-top: 15px;
            border-radius: 5px;
        }
    </style>
```

В разметке содержимого мы использовали заголовки второго уровня, обычные текстовые абзацы и список. Не будем задавать им какого-либо специфического форматирования, только подкорректируем цвет и междустрочный интервал:

Глава 3\HTML-письма\index.html

```
    <style>
        <!- Выше предыдущие стили ->

        #email-styles .email-h2 {
            color: #0909b6;
            border-bottom: #0a0ab4;
        }

        #email-styles .email-par {
            line-height: 1.25;
        }

        #email-styles .email-ul-list {
            line-height: 1.25;
        }
    </style>
```

Изображения будем растягивать по всей доступной ширине. Делаем его блоком, чтобы избежать любого возможного обтекания текстом, рисуем рамку:

Глава 3\HTML-письма\index.html

```
    <style>
        <!- Выше предыдущие стили ->

        #email-styles .email-big-img {
            display: block;
            width: 100%;
```

```
            border: 2px solid #444444;
        }
    </style>
```

Осталось оформить гиперссылки. Делаем их в форме строчных блоков, убираем подчеркивание, заливаем фон и увеличиваем размер с помощью внутренних отступов. Дополнительно скруглим углы и применим эффект тени для текста.

Также описываем два класса, которые будут определять разные цвета ссылок:

Глава 3\HTML-письма\index.html

```
<style>
    <!- Выше предыдущие стили ->

    #email-styles .email-ref-button {
        display: inline-block;
        text-decoration: none;
        background: #0909b6;
        color: aliceblue;
        font-weight: bold;
        padding: 8px 12px;
        border-radius: 6px;
        text-shadow: 0 0 3px black;
    }

    #email-styles .email-button-red {
        background: #b63109;
    }

    #email-styles .email-button-green {
        background: #23a10a;
    }
</style>
```

4. Преобразование в inline-стили

Перед тем, как код будет скопирован в письмо, внутреннее описание стилей в блоке `<style>` необходимо заменить на встроенные inline-стили, т.е. избавиться от классов взамен на атрибут **`style`** и описание в нем CSS-стилей у каждого тега. (Ранее отмечалось, что это нежелательный способ подключения стилей, но единственно рабочий в случае разметки HTML-писем)

Разумеется, вносить описание стилей внутрь тегов вручную нерационально. Воспользуемся одним из веб-сервисов, который это сделает автоматически.

Перейдем на сайт templates.mailchimp.com, скопируем в поле ввода весь код разметки нашей веб-страницы. После нажатия на кнопку *Convert* веб-сервис автоматически вставит стили в теги, согласно описанным классам:

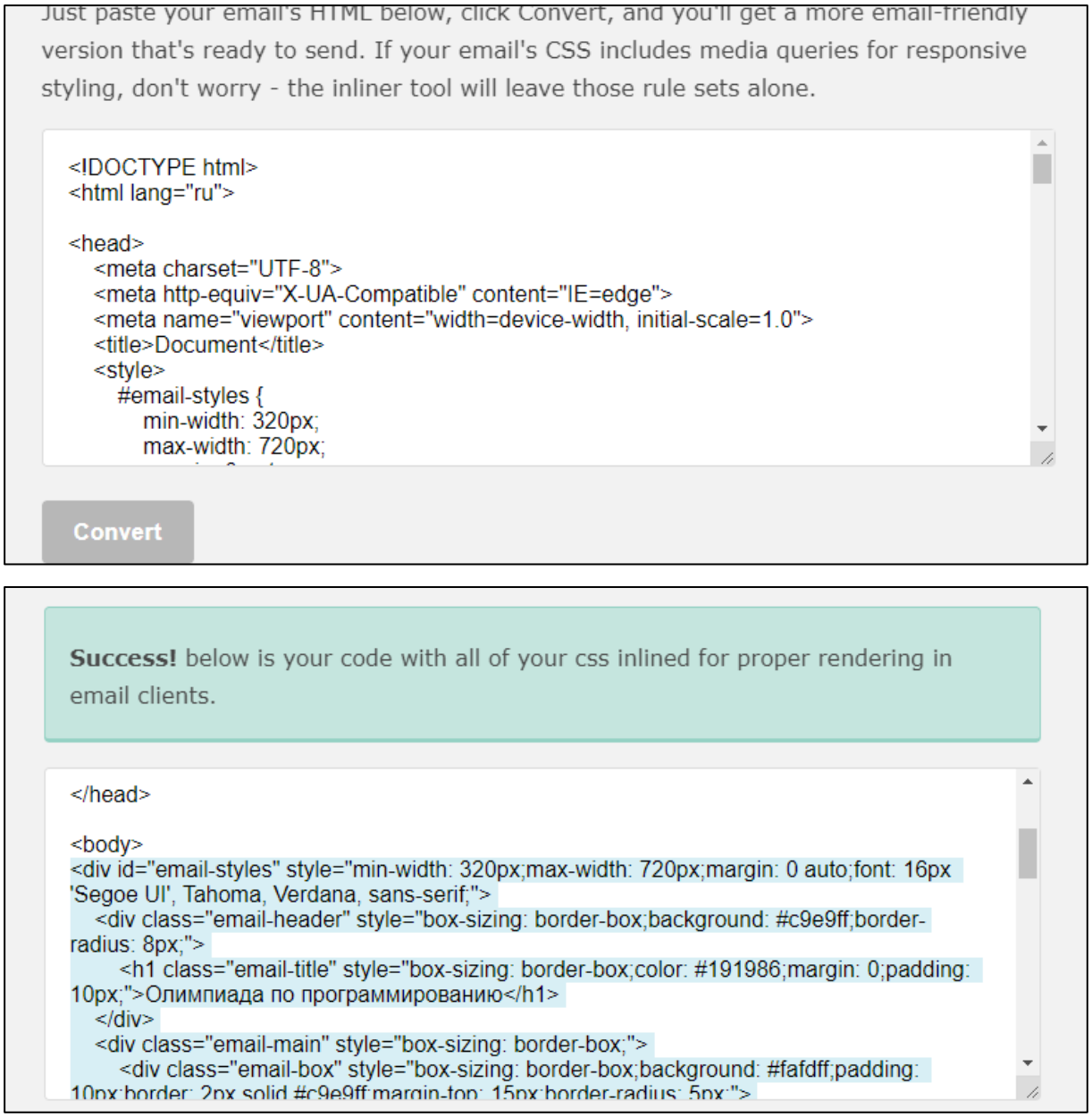


Рис. 3.77. Встраивание стилей в теги

Для письма необходимо скопировать только код нашего блока `email-styles`!

5. *Внедрение разметки в письмо*

Последний шаг – копирование полученной разметки в текст письма. Продемонстрируем эту процедуру на примера браузера Google Chrome и почтового сервиса от Mail.ru.

Создаем новое письмо. Удалим из него весь текст, вместе с блоком подписи.

Далее вызовем инспектор кода. С помощью кнопки *Выбрать элемент на странице* наводим курсор на блок, внутри которого размещается разметка письма (он не содержит каких-либо стилей):

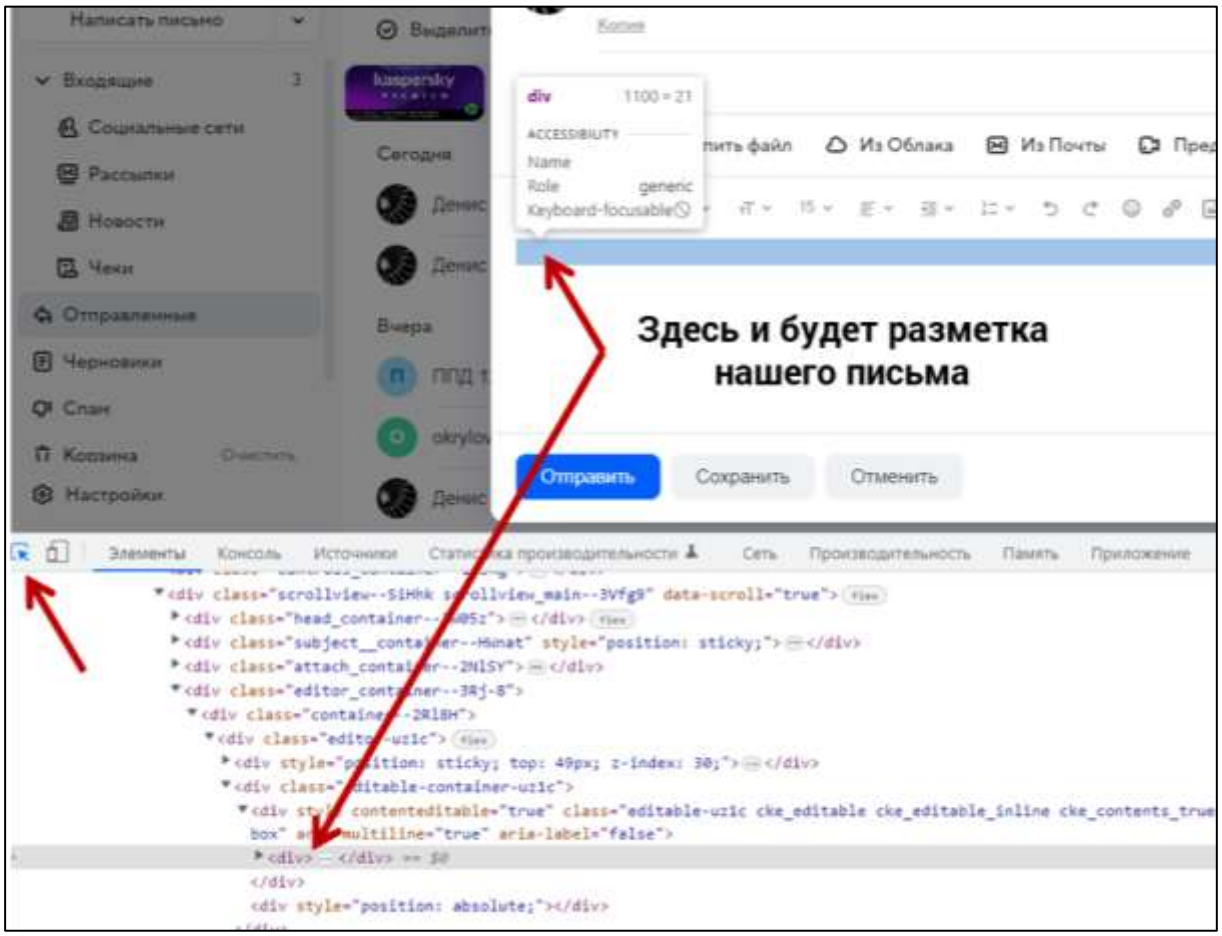


Рис. 3.78. Блок для вставки разметки письма

Далее в разметке щелкаем *ПКМ* по этому блоку и выбираем пункт *Редактировать как HTML*. В открывшемся поле удаляем код блока `<div>` и вставляем ранее скопированную разметку письма.

Чтобы письмо внедрилось, нажатием на крестик в правом углу закрываем панель разработчика (рис. 3.79). Письмо отобразится с сохранением форматирования, как и было задумано.

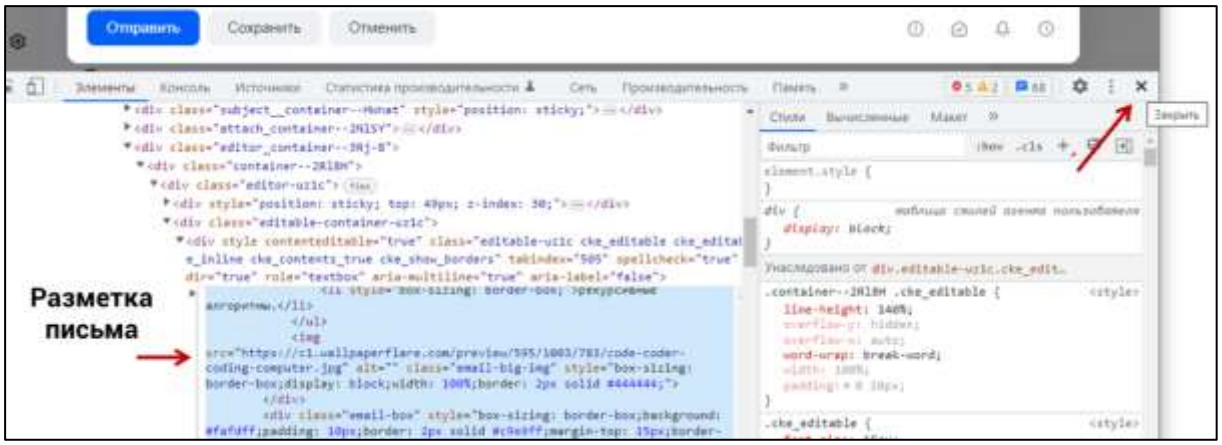


Рис. 3.79. Вставка разметки HTML-письма с помощью инспектора кода

Единственное, что может отличаться от искомого – отсутствие изображения. Это связано с тем, что для его загрузки потребуется обновление страницы (делать не нужно) или отправка письма адресату.

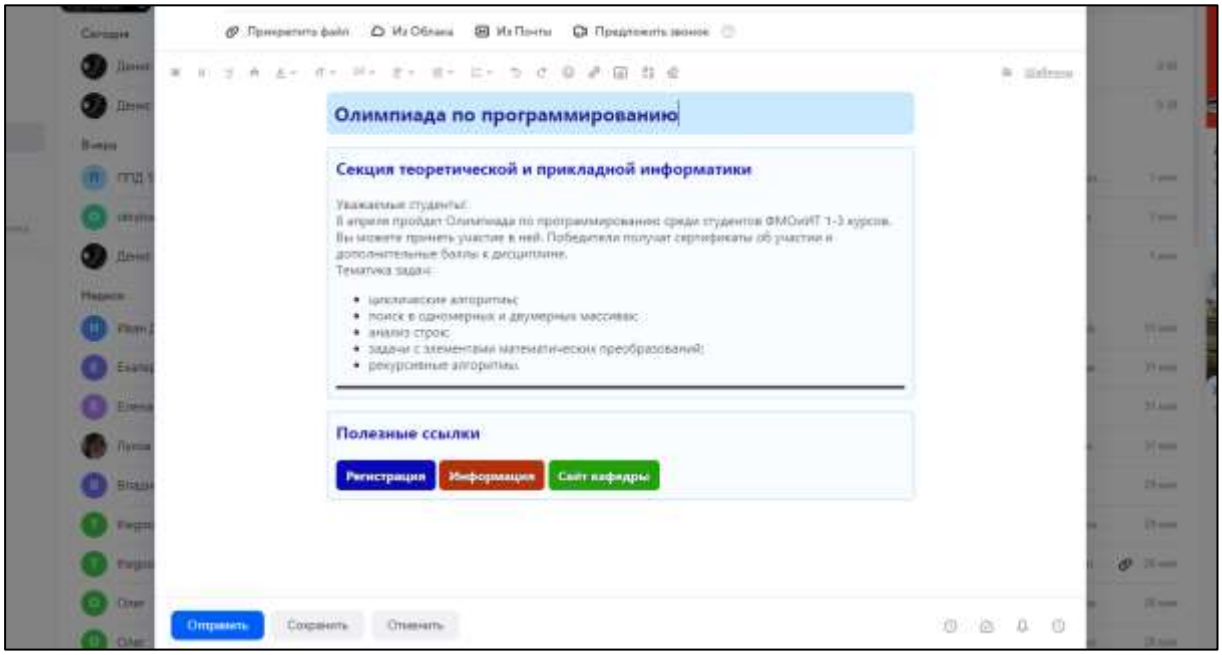


Рис. 3.80. Разметка внедрилась практически без изменений

Для тестирования отправим сообщение на свой почтовый ящик. Важно проверить, имеются ли существенные отличия в оформлении:

- исходящего сообщения;
- входящего сообщения;
- входящего сообщения на другом почтовом сервисе (для этого отправляем копию письма на другой почтовый ящик).

При необходимости текст сообщения можно поправить локально: каретка редактирования текста будет доступна, как и в обычном письме. Однако правку разметки в целом и оформления элементов необходимо делать в редакторе, повторяя процедуру встраивания CSS-стилей и внедрения письма с помощью инспектора кода.

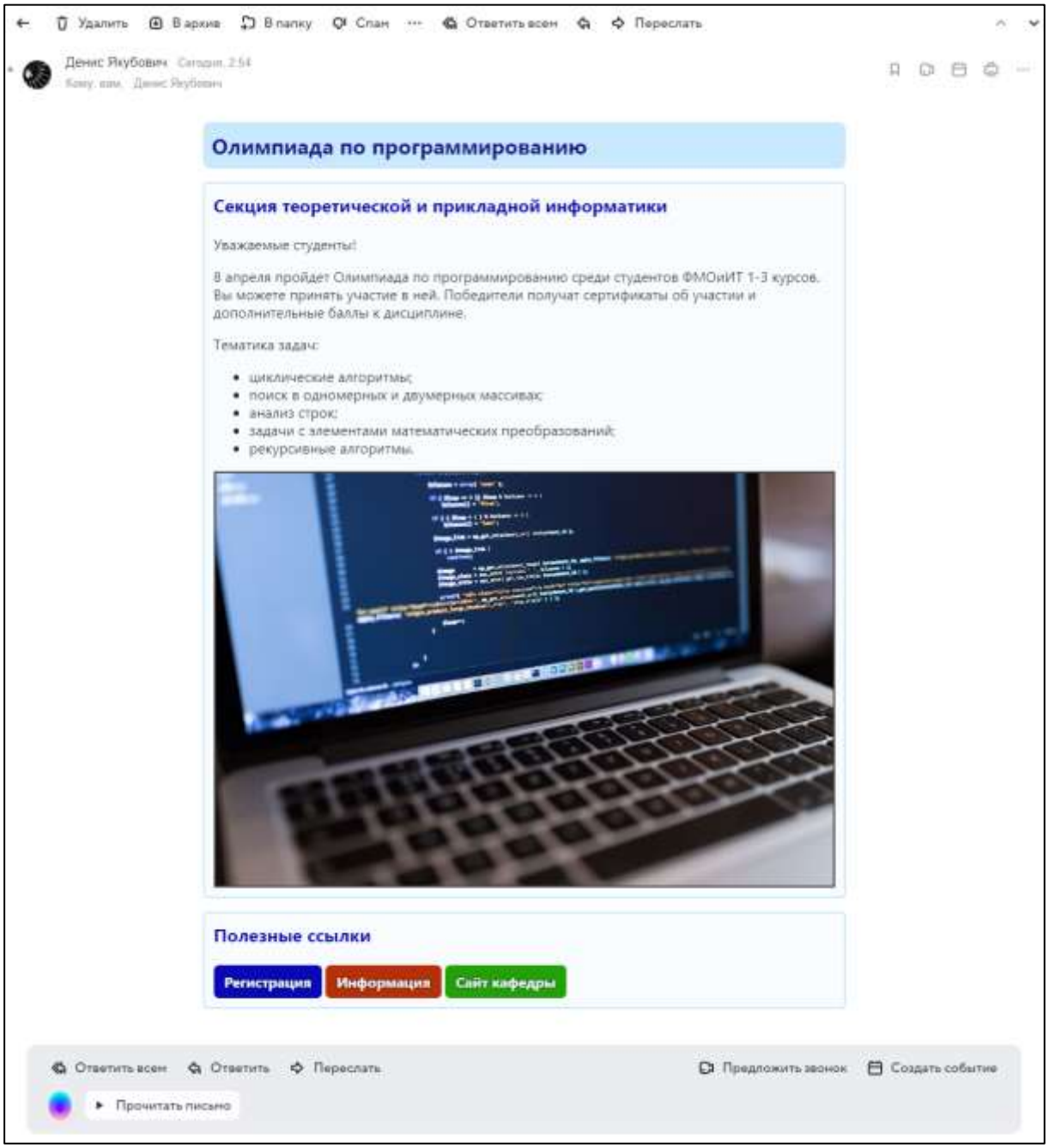


Рис. 3.81. Оформление входящего сообщения в Mail.ru

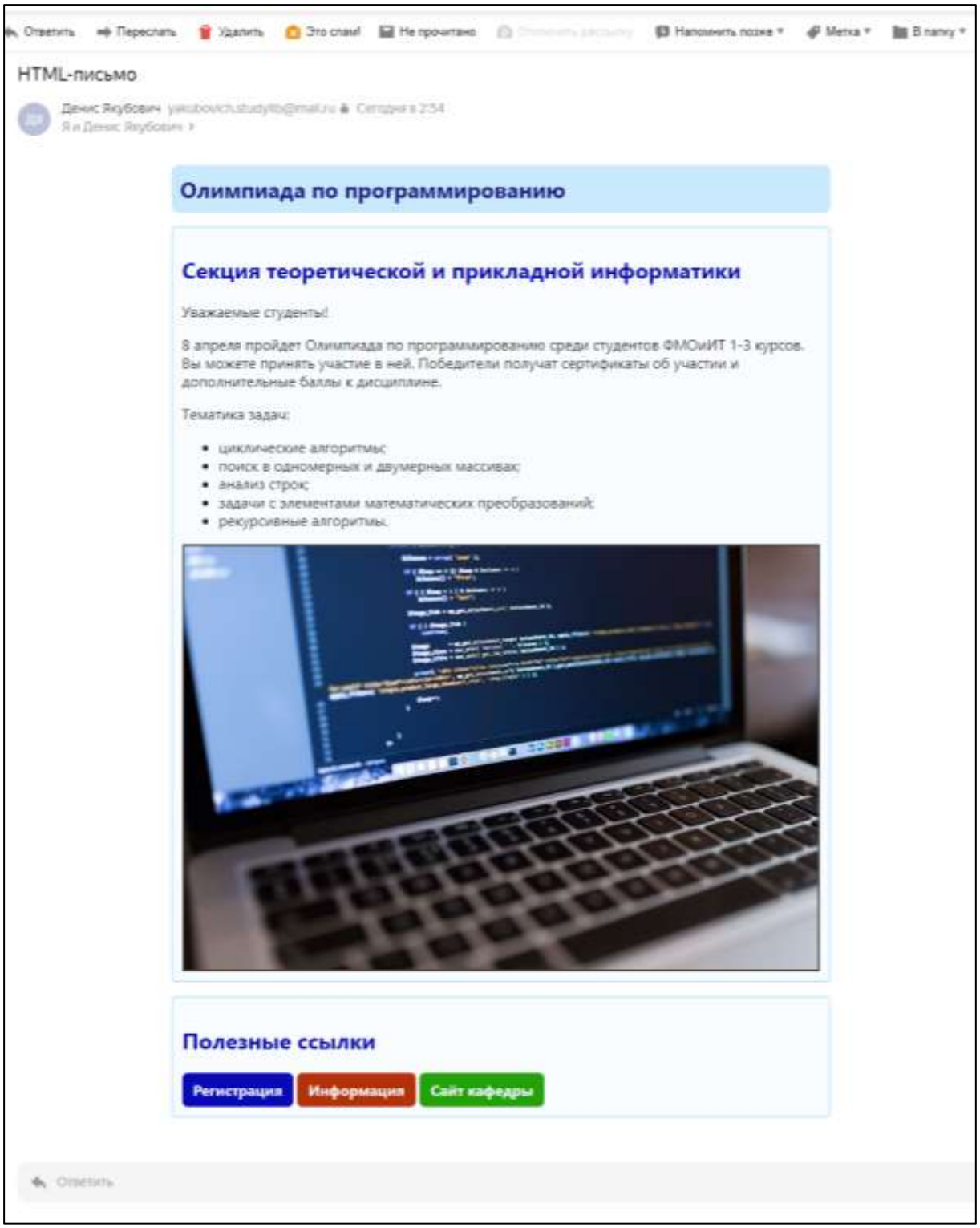


Рис. 3.82. Оформление входящего сообщения в Yandex.ru

Вопросы для самопроверки

1. Перечислите основные CSS-свойства для настройки шрифта.
2. Опишите синтаксис универсального свойства `font` и приведите примеры его использования.
3. Какие свойства можно настроить для оформления текста?
4. Перечислите назначение свойств для управления настройками списков.
5. Какие модели представления цвета поддерживаются в CSS3?
6. Чем отличается модель RGB от HEX?
7. Перечислите свойства. Которые необходимы для управления цветом текста и фоновой заливкой.
8. Какие возможности дает инспектор кода в браузере?
9. Опишите процедуру создания HTML-писем.
10. Какие аспекты важно учитывать при верстке HTML-писем?

Практикум

Задание 1

1. Создайте каталог *Свойства текста и фона* с типовой стартовой разметкой. Подключите таблицу стилей.
2. Скопируйте и изучите прикрепленную ниже разметку и созданные стили.
3. Требуется дописать свойства для созданных селекторов согласно указанному в комментариях описанию.
4. Образец результата изображен на рис. 3.83.

Глава 3\Свойства текста и фона\index.html

```
<div class="main">
    <h1>Белокаменное зодчество Владимиро-Суздальской
    земли</h1>

    <h2>Успенский собор</h2>
    <p class="def">Знаменитый памятник белокаменного
    зодчества домонгольской Руси. Успенский собор являлся
    кафедральным храмом Владимиро-Суздальской Руси. Собор
    был местом венчания на великое княжение князей
```

```
        владимирских и московских.</p>
        <p>Собор был построен в XII веке и стал образцом для
        целого ряда более поздних соборов, в числе которых
        Успенский собор Аристотеля Фиораванти.</p>
        <p>Историки датируют его 1158–1160 и 1186–1189 годами
        (перестроен после пожара). Отметим высокую
        сохранность структуры Собора.</p>
        <p>Интерьеры Собора XII хранят фрагменты фресок
        Андрея Рублёва и Даниила Чёрного 1408 года. В XIX
        веке внешний вид пятиглавого Собора изменился: был
        пристроен притвор и колокольня.</p>
        <p><a href="#">Подробнее</a></p>

        <h2>Дмитриевский собор</h2>
        <p class="def">Дмитриевский собор был построен в
        качестве придворного собора во времена правления
        Всеволода Большое Гнездо.</p>
        <p>Этот собор является каноническим образцом
        белокаменного храма владимиро-суздальской
        архитектурной школы. Особая ценность собора - хорошо
        сохранившаяся белокаменная резьба.</p>
        <p>Постройку собора датируют 1194–1197 годами.</p>
        <p>В 1840-х годов была проведена ошибочная
        реставрация, в результате которой Дмитриевский собор
        утерял галереи и лестничные башни начала XIII
        века.</p>
        <p><a href="#">Подробнее</a></p>

        <h2>Церковь Покрова на Нерли</h2>
        <p class="def">Одно из самых знаменитых творений
        владимиро-суздальской школы. Храм был построен в
        1165–1166 годах близ пригорода Владимира –
        Боголюбово, в месте, где река Нерль в впадает в
        Клязьму.</p>
        <p>Церковь Покрова на Нерли представляет собой
        белокаменный одноглавый четырёхстолпный храм с
        гармоничными пропорциями. Особо интересно его
        местоположение: широкой пойма Нерли и Клязьмы.</p>
        <p>До 1672 года к храму примыкала башня с ходом на
        хоры. К настоящему времени галереи утрачены.</p>
        <p><a href="#">Подробнее</a></p>
    </div>
```

Глава 3\Свойства текста и фона\style.css

```
body {
    /*  Установить:
        - шрифт Cambria, 1.05em, или любой с засечками;
        - цвет фона светло зеленый;
        - цвет текста - темно серый  */
}

.main {
    border: 2px solid #8d8d8d;
    border-radius: 3px;
    padding: 10px;
    margin: 10px;
    background: rgba(255, 255, 255, 0.75);
}

h1, h2 {
    /*  Установить:
        - темно-бирюзовый цвет заголовков;
        - шрифт - Calibri;
        - все буквы делаются заглавными  */
}

p {
    /*  Междустрочный интервал - 1.35  */
}

.def {
    /*  Оформить:
        - жирным начертанием, размер шрифта 110%;
        - междустрочный интервал 1.5;
        Реализовать через одно свойство - font!  */
}

a {
    /*  Реализовать:
        - убрать подчеркивание ссылки;
        - установить жирное и курсивное начертание;
        - изменить цвет (по вкусу)  */
}
```

Задание 2

1. Создайте каталог *Оформление ссылок* с типовой стартовой разметкой. Подключите таблицу стилей.
2. Скопируйте и изучите прикрепленную ниже разметку и созданные стили.
3. Требуется дописать свойства для созданных селекторов согласно указанному в комментариях описанию.
4. Образец результата изображен на рис. 3.84-рис. 3.85.

Глава 3\Свойства текста и фона\index.html

```
<main>
    <div class="light-theme">
        <h1 class="light-title">
            Кнопка на светлом фоне
        </h1>
        <a href="https://www.google.ru/"
           class="light-ref">
            Google
        </a>
    </div>

    <div class="dark-theme">
        <h1 class="dark-title">
            Кнопка на темном фоне
        </h1>
        <a href="https://yandex.ru/"
           class="dark-ref">
            Yandex
        </a>
    </div>
</main>
```

Глава 3\Свойства текста и фона\style.css

```
body { background: #c4c4c4; }

main { padding: 10px; }

h1 { font-family: 'Segoe UI', Tahoma, Verdana,
    sans-serif; }
```

```
/* Блоки для оформления фона (светлый/темный) */
.light-theme,
.dark-theme {
    padding: 25px 30px;
}

.light-theme { background: #fff; }
.dark-theme { background: #0a3ba5; }

/* Классы для оформления заголовков (светлый/темный) */
.light-title { color: #0a3ba5; }
.dark-title { color: #fff; }

/* Классы для декоративного оформления ссылок
   (на светлом/темном фоне) */
.light-ref,
.dark-ref {
    display: inline-block;
    /*
       убрать подчеркивание ссылки,
       шрифт Tahoma, размером 105%,
       полужирное начертание,
       отступ сверху/снизу - 15px, справа/слева - 22px,
       радиус скругления углов - 8px
    */
}

.light-ref {
    /* темный фон и светлый цвет текста */
}

.dark-ref {
    /*
       светлый фон и темный цвет текста,
       эффект свечения сделать с помощью свойства
       тени box-shadow
    */
}
```

Задание 3

1. Создайте проект с названием *HTML-письма*.
2. Реализуйте письмо, описанное в пункте 3.4.5.
3. Протестируйте его отправку на разные почтовые веб-сервисы, где вы зарегистрированы.

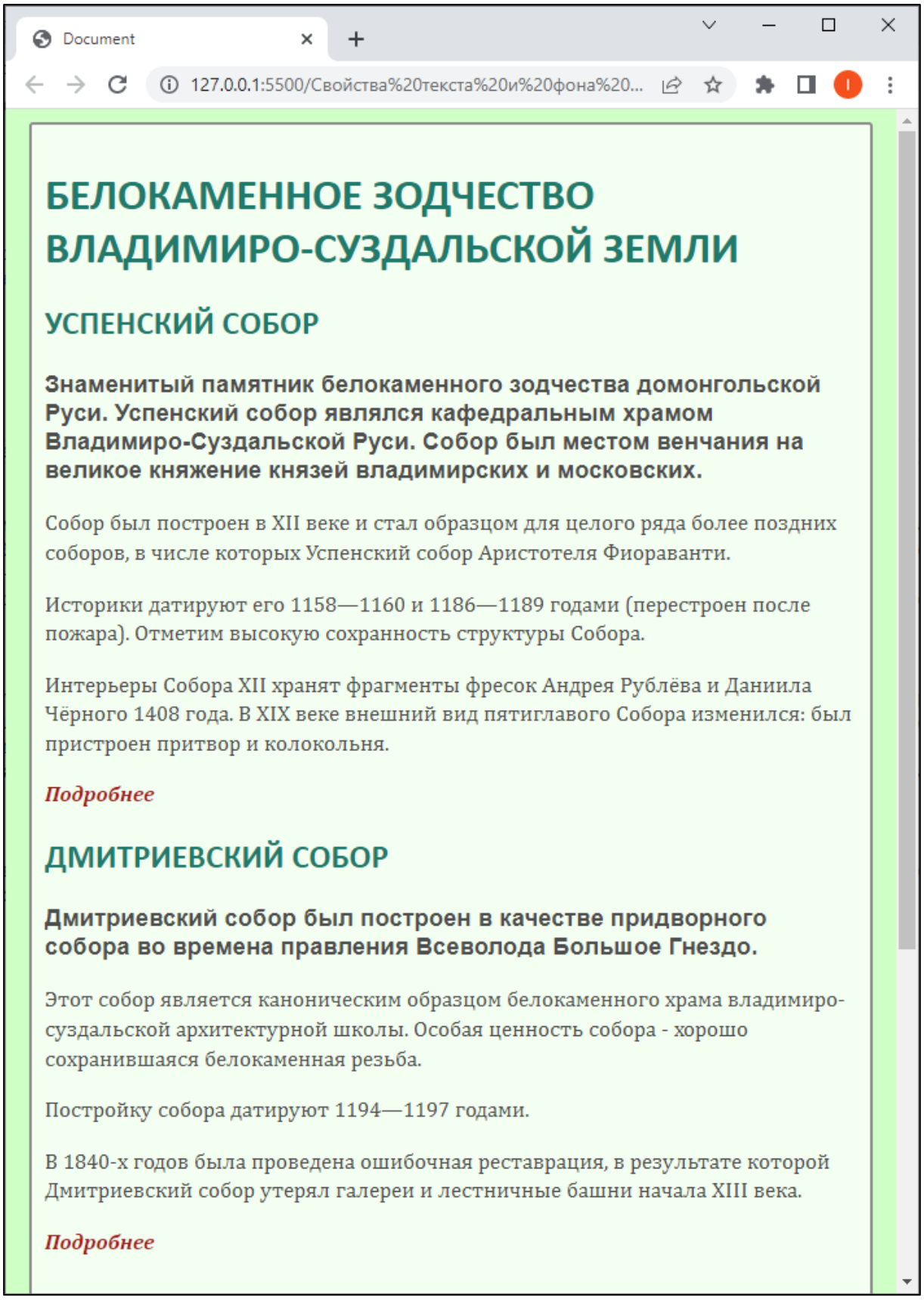


Рис. 3.83. Образец выполнения задания 1 (часть 1)

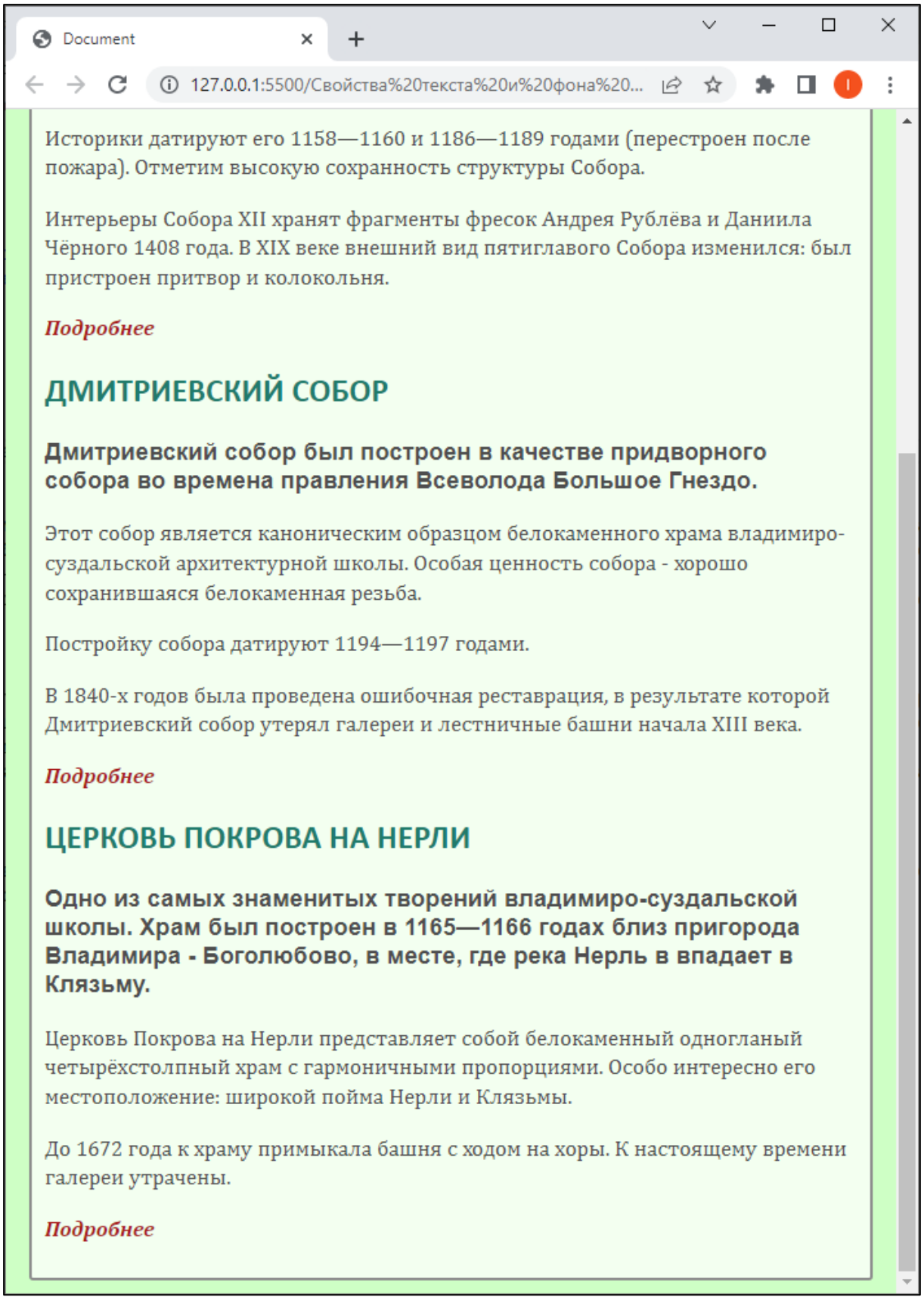


Рис. 3.84. Образец выполнения задания 1 (часть 2)

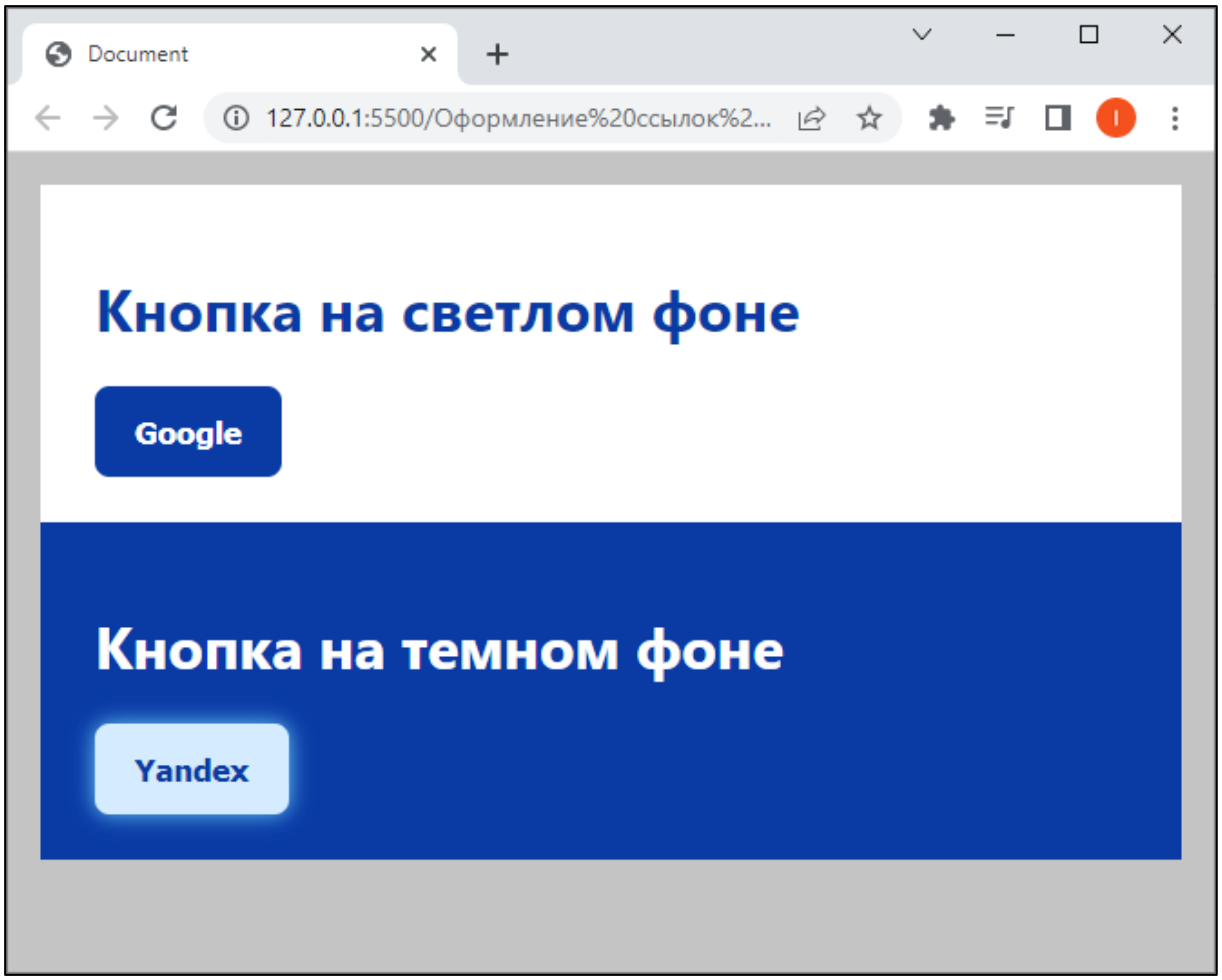


Рис. 3.85. Образец выполнения задания 1 (часть 2)

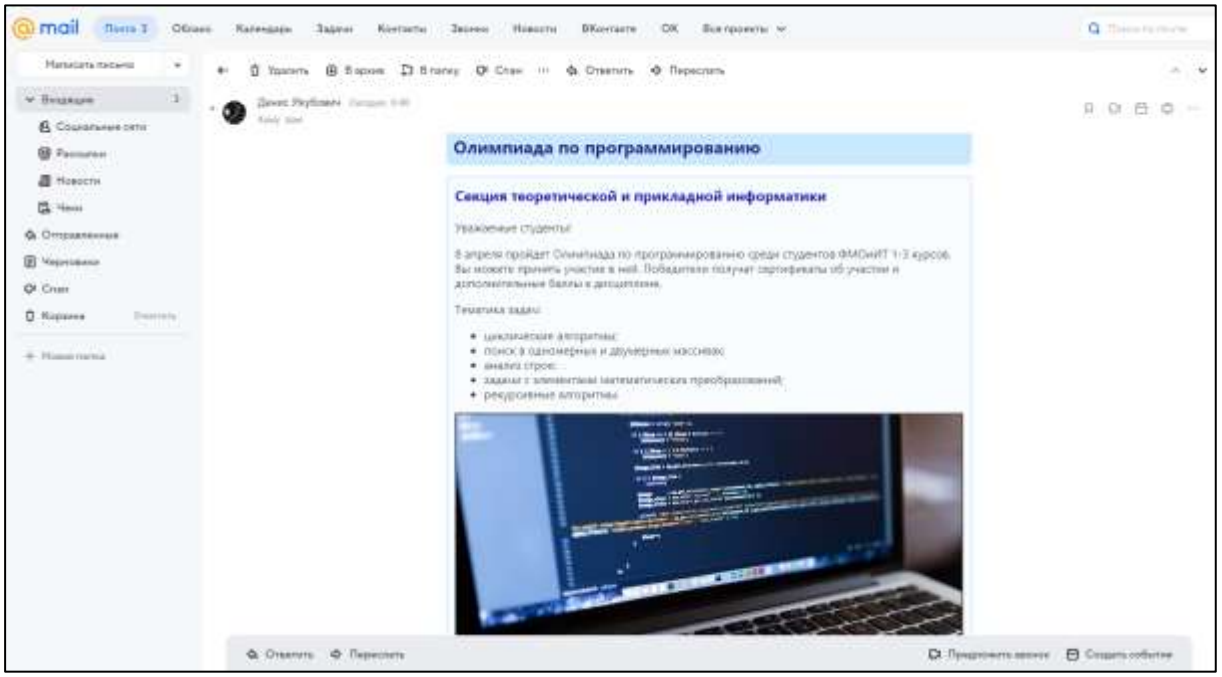


Рис. 3.86. HTML-письмо в почтовом ящике

Олимпиада по программированию

Секция теоретической и прикладной информатики

Уважаемые студенты!

8 апреля пройдет Олимпиада по программированию среди студентов ФМОиИТ 1-3 курсов. Вы можете принять участие в ней. Победители получат сертификаты об участии и дополнительные баллы к дисциплине.

Тематика задач:

- циклические алгоритмы;
- поиск в одномерных и двумерных массивах;
- анализ строк;
- задачи с элементами математических преобразований;
- рекурсивные алгоритмы.

Полезные ссылки

Регистрация | Информация | Сайт кафедры

Рис. 3.87. Образец выполнения задания 3

3.5. Каскадирование стилей

3.5.1. Правила наследования стилей

Назначение

Определение

Наследование стилей CSS *– это способность дочерних элементов копировать CSS-свойства родительских блоков.*

Наследование свойств является одним из основополагающих принципов работы CSS. Он позволяет автоматически учитывать при оформлении элемента формат его родителя(ей) и не описывать одни и те же группы свойств для настройки каждого селектора.

Многие теги разметки документа изначально имеют предустановленные CSS-свойства, которые отвечают за базовое форматирование HTML-тегов (их также называют *user agent-стилями* или стилями браузера). Например, для `<body>` предустановлены небольшие поля, гиперссылки `<a>` имеют синий цвет, заголовки `<h1-h6>` выделяются полужирным начертанием и имеют разный относительный размер и т.д.

Некоторые из CSS-свойств (например, цвет текста, размер шрифт, форма курсора и др.) по умолчанию распространяются для любых вложенных элементов. Подобные свойства изначально имеют значение **`inherit`** («наследуемое»), что и обеспечивает наследование. При необходимости свойство перезаписывают новым значением и наследование разрывается.

Общие правила

В общем случае правила наследования зависят от приоритета стиля селектора, который четко определяется правилами CSS.

1. Объединение стилей

Если один и тот же селектор описан в разных местах, то стили объединяются.

Например, если для селектора

```
strong {
    font-family: Roboto, Tahoma, sans-serif;
    color: brown;
}
```

ниже указать

```
strong {
    font-style: italic;
}
```

то такое обращение будет равносильно

```
strong {
    font-family: Roboto, Tahoma, sans-serif;
    color: brown;
    font-style: italic;
}
```

2. Переопределение стилей

В случае повторного описания одного и того же свойства селектору задается значение последнего из перечисленных.

Например, в силу повторного описания селектора класса `box-frame` свойство `border-radius` будет скорректирована на `8px`:

```
.box-frame {
    border: 1px solid grey;
    border-radius: 5px;
    margin: 10px;
}

.box-frame {
    border-radius: 8px;
}
```

3. Наследование свойств селектора тега

Теги с классами (идентификаторами) наследуют стили самого тега, но могут перезаписывать значения их свойств.

Например, селектор `p.attention` унаследует междустрочный интервал от селектора тега `p`:

```
p {
    line-height: 1.25;
}
```

```
p.attention {
    color: red;
}
```

То же самое касается и автономных классов. Они также наследуют свойства тега, к которому будут подключены:

```
.attention {
    color: red;
}
```

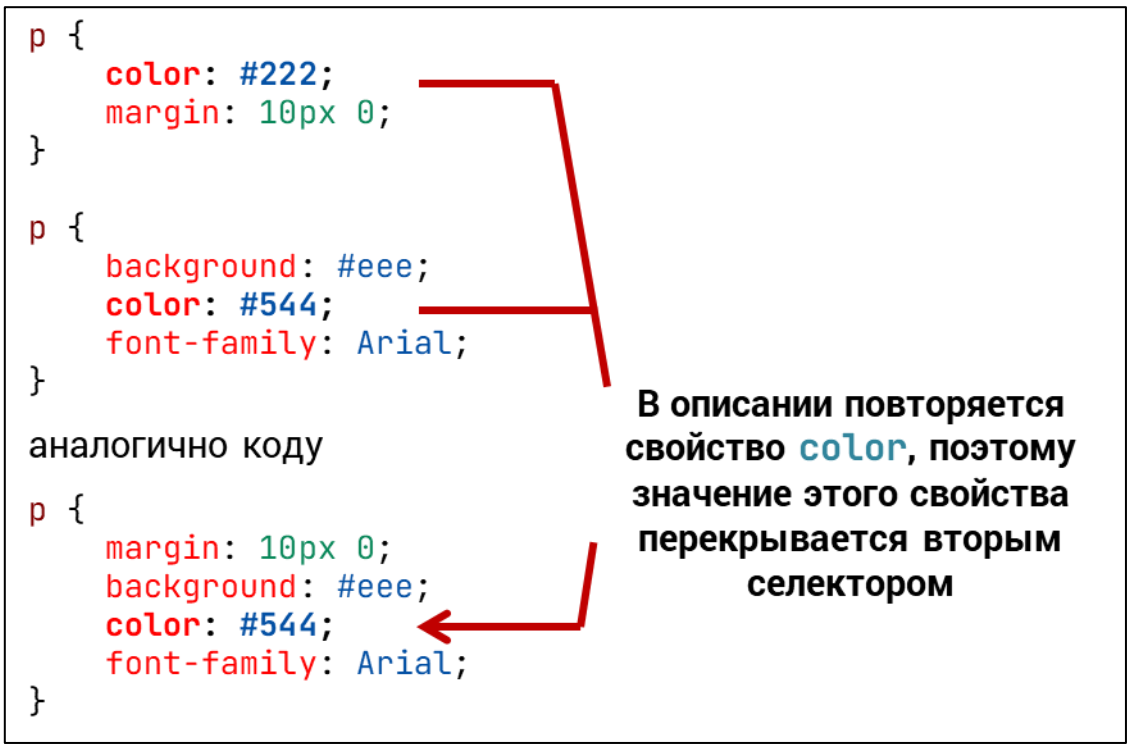


Рис. 3.88. Объединение и переопределение стилей селектора при наследовании

3.5.2. Правила группировки стилей

Назначение

Определение

Группировка стилей селекторов *– избавление от излишнего дублирования стилей в селекторах путем группировки свойств с одинаковыми значениями.*

Метод группировки позволяет избежать излишнего дублирования кода селекторов, имеющих в описании одно и более одинаковых свойств с одним и тем же значением.

Общие правила

1. *Группировка разных селекторов*

Для группировки стилей достаточно перечислить через запятую селекторы и описать для них общий стиль. А свойства, принимающие разные значения, указать ниже отдельно.

До группировки 😕

```
h1 {
    text-align: center;
    font-weight: bold;
    font-size: 200%;
}

h2 {
    text-align: center;
    font-weight: bold;
    font-size: 170%;
}

h3 {
    text-align: center;
    font-weight: bold;
    font-size: 145%;
    color: #AAA;
}
```

После группировки 🙂

```
h1, h2, h3 {
    text-align: center;
    font-weight: bold;
}

h1 { font-size: 200%; }

h2 { font-size: 180%; }

h3 {
    font-size: 145%;
    color: #AAA;
}
```

Рис. 3.89. Группировка стилей в отдельных селекторах

При этом группировать следует селекторы, которые связаны по смыслу, форматируют близкие по назначению элементы и дублируют несколько свойств. Искусственно группировать мало связанные элементы и всего лишь по одному свойству и его значению не следует.

2. *Группировка в одном селекторе*

Свойства селектора, описанного в разных местах, также группируются. А в случае дублирования свойства его значение перезаписывается.

Например, селектор описан в разных частях файла:

```
.form__text-input {
    background: lightgray;
    padding: 4px 2px;
    margin: 8px 0;
}

.form__text-input {
    width: 100%;
}

.form__text-input {
    background: #dbdbdb;
}
```

Вместо предложенного описания рационально сгруппировать код стилей в один блок (`background` перезапишется):

```
.form__text-input {
    width: 100%;
    background: #dbdbdb;
    padding: 4px 2px;
    margin: 8px 0;
}
```

Это полезно знать!

Следует сразу объединять все свойства селектора в одном блоке описания.

3.5.3. Контекстные и дочерние селекторы

Назначение

Определение

Контекстный селектор *– селектор, являющийся любым потомком указанного.*
Дочерний селектор *– селектор, являющийся прямым потомком указанного.*

Например, пусть задана разметка простого HTML-документа:

```
<!DOCTYPE html>
<html lang="ru">
<head>
    <meta charset="UTF-8">
    <title>Document</title>
</head>
<body>
    <h1>Название сайта</h1>
    <h2>Меню</h2>
    <ul id="menu">
        <li><a href="#">Пункт 1</a></li>
        <li><a href="#">Пункт 2</a></li>
        <li><a href="#">Пункт 3</a></li>
    </ul>
</body>
</html>
```

Для любого HTML-документа можно схематически изобразить структуру разметки. В случае приведенного примера она выглядит следующим образом:

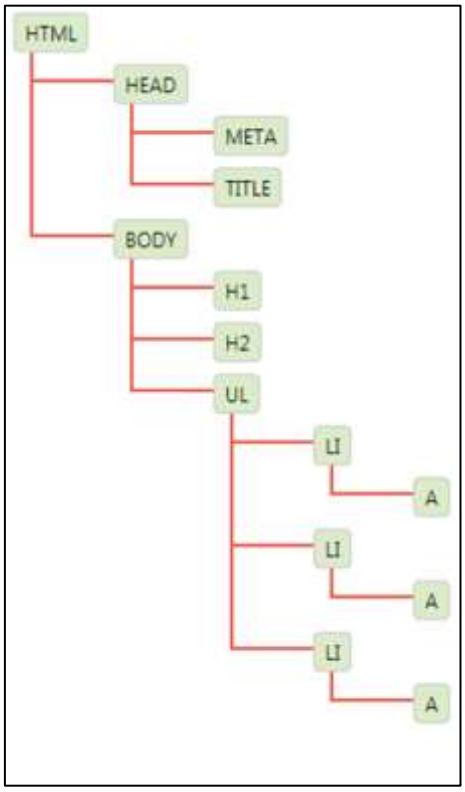


Рис. 3.90. Структура разметки: связь между родительскими и дочерними элементами

Если тег <body> рассматривать в качестве родителя, то:

- *контекстными* для него являются теги <h1>, <h2>, <ul>, <li> и <a>;
- *дочерними* – только <h1>, <h2> и <ul>.

Использование контекстных и дочерних селекторов помогает задать разное оформление одинаковым тегам в зависимости от их положения в HTML-дереве.

Кроме того, часто это позволяет избежать описания большого числа классов и их многократного применения для объектов с одинаковым оформлением, находящихся в одном родительском блоке.

CSS-синтаксис селекторов

1. *Контекстные селекторы*

Определение

Составной селектор

```
X Y { }
```

где X – родитель, Y – потомок, определяет ***контекстный*** *селектор. Запись означает, что стили описываются для любого селектора Y, находящегося внутри X и допускающего промежуточные вложения.*

Например, селектор

```
p a {
    text-decoration: none;
    color: #0080C0;
}
```

установит стиль тем ссылкам <a>, которые находятся внутри тега абзаца <p>. При этом тег ссылки может находиться внутри других парных тегов, которые, в свою очередь, находятся внутри тега абзаца:

```
<p>
    . . .
    <a href="#">ссылка</a>
    . . .
</p>
```

Другой, более «сложный» пример. Селектора вида

```
div.menu__panel .menu a {
    font: bold 120% Calibri;
    padding: 4px;
}
```

задает оформление только для тех ссылок `<a>`, которые находятся внутри любого тега с классом `menu`, который, в свою очередь, расположен в контейнере `<div>`, описанном со связанным классом `menu__panel`, т.е. свойства заработают, например, для ссылок следующей разметки:

```
<div class="menu__panel">
    . . .
    <nav class="menu">
        . . .
        <a href="#">Ссылка 1</a>
        <a href="#">Ссылка 2</a>
        . . .
    </nav>
    . . .
</div>
```

2. *Дочерние селекторы*

Определение

Составной селектор

```
X > Y { }
```

где X – родитель, Y – прямой потомок, определяет ***дочерний*** *селектор. Запись означает, что стили описываются для любого селектора Y, являющегося прямым потомком X, т.е. промежуточные вложения не допускаются.*

Например, селектор

```
p > strong { color: #F00; }
```

задает цвет только тому тексту внутри `<strong>`, который непосредственно является дочерним для тега `<p>`.

Так, в случае разметки

```
<p>
    <strong>Дочерние селекторы</strong>.
    <br>
    <b><strong>Дочерние селекторы</strong></b>
</p>
```

стиль сработает только для первого тега `<strong>`, т.к. он является прямым потомком `<p>`. Если бы правило описали через контекстный селектор

```
p strong { color: #F00; }
```

то стиль бы сработал и для второго вложения.

Дочерние селекторы удобно применять для элементов, которые обладают четкой иерархической структурой, не имеющей промежуточных тегов (списки, таблицы, меню).

Это полезно знать!

Не трудно заметить, что дочерний селектор является частным случаем контекстного, который ограничивается только прямыми потомками тега.

Приемы работы с Visual Studio Code

При наведении курсора на селектор отображается схема того, к какому элементу будет вестись обращение (рис. 3.91). Видно, что ***троеточие*** *обозначает любое число промежуточных тегов-контейнеров, а символ* **>** *- прямого потомка тега. Ключевое слово* **element** *подразумевает любой тег, поскольку не указано явно, к какому тегу цепляется класс* **content** *(для идентификатора будет аналогично).*

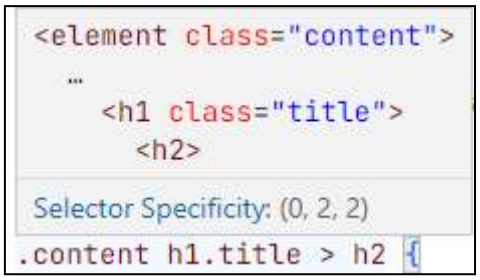


Рис. 3.91. Подсказка редактора по селектору и соответствующему тегу, к которому ведется контекстное обращение

Рассмотрим пример использования контекстного обращения к селекторам.

Глава 3\Каскадирование\index.html

```
<nav class="panel">
    <div class="center-box">
        <a href="files/index.html">Главная</a>
        <a href="files/lections.html">Лекции</a>
        <a href="files/practice.html">Практикум</a>
        <a href="files/exam.html">Проверь себя</a>
    </div>
</nav>
```

Глава 3\Каскадирование\style.css

```
/* Ограничить "плавающую" ширину страницы */
body { min-width: 420px; }

/* Область навигации */
.panel {
    background: #FEEE94;
    padding: 18px 0;
}

/* Класс выравнивает по центру блок любой ширины */
.center-box {
    display: table;      /* ведет себя как таблица */
    margin: 0 auto;      /* центрует ячейку по ширине */
}

/* Оформление ссылки внутри главной панели ссылок */
.panel a {
    background: #ad3823;
    text-decoration: none;
    border-radius: 4px;
```

```
        font: bold 110% Calibri;
        color: #fff;
        padding: 6px 18px;
        margin: 0 4px;
    }
```

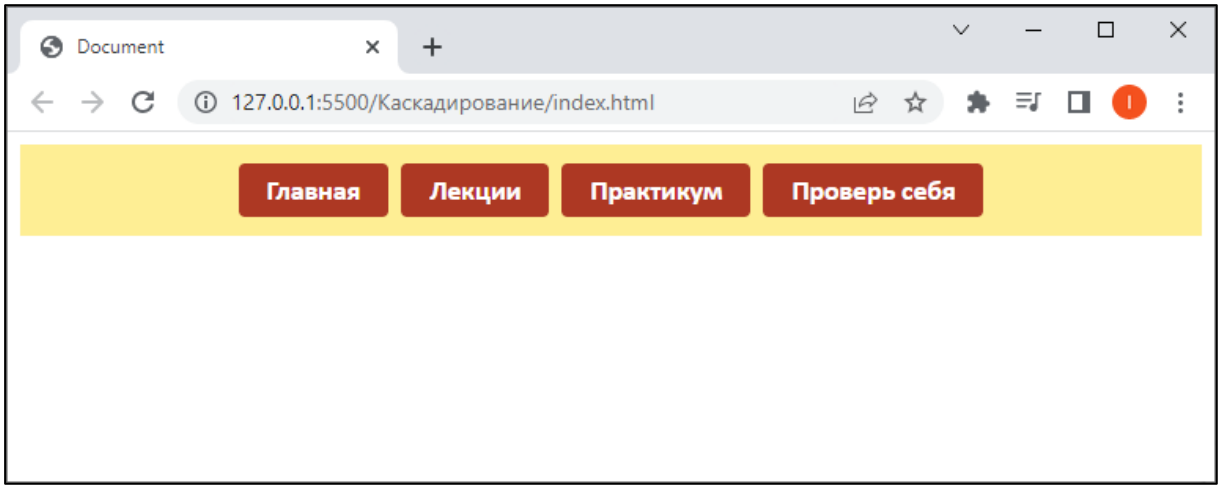


Рис. 3.92. Оформление блока гиперссылок

Из приведенного примера отметим, что

- селектор `.panel` также можно было описать как `.nav.panel`;
- селектор `.panel a` можно было описать как `.nav.panel a`.

Однако тег-селектор `nav` необязательно указывать, поскольку в документе больше нет тегов с классом `panel`: здесь браузер определяет его однозначно.

Преимущество контекстного вложения

Может возникнуть вопрос: что дает использование дочерних и контекстных селекторов, если подобный результат можно достичь и с помощью классов?

С одной стороны, контекстное обращение позволяет усиливать приоритетность стиля и корректно оформлять элементы в определенной области HTML-разметки (например, ссылки в меню и ссылки в тексте документа могут иметь разное оформление).

С другой стороны – оптимизация HTML-кода. Так, если имеется блок, внутри которого каждое изображение должно быть декоративно оформлено, то к каждому тегу `<img>` потребуется цеплять один и тот же класс:

```
<div>
    <img src="kotik.png" class="picture">
    <img src="hohma.jpg" class="picture">
    <img src="demotivator.jpg" class="picture">
</div>
```

С помощью контекстных (или дочерних) селекторов достаточно привязать класс только к блоку с изображениями:

```
<div class="pictures">
    <img src="kotik.png">
    <img src="hohma.jpg">
    <img src="demotivator.jpg">
</div>
```

и описать как стиль оформления самого блока, так и изображений внутри него:

```
.pictures {
    /* Оформляем сам блок с изображениями */
}

.pictures img {
    /* Оформляем изображения внутри блока с классом
       pictures */
}
```

В данном примере также можно использовать и вложение прямого потомка, поскольку нет промежуточных контейнеров:

```
.pictures > img {  }
```

Недостатки контекстного вложения

Однако преимущества контекстного вложения часто являются и его недостатками. Особенно остро проблема начинает проявляться при многочисленных вложениях тегов, когда одни и те же элементы требуют разного оформления в разных частях документа. Чтобы усилить приоритет стиля, в составной селектор приходится добавлять дополнительные промежуточные селекторы.

В подобных случаях вместо усложнения структуры селекторов может быть более рациональным пожертвовать лаконичностью HTML-разметки и использовать классы. Тем более что в настоящее время выработаны разные методологии верстки, систематизирующие подходы к структуризации кода.

3.5.4. Каскадирование и специфичность

Проблема неоднозначности стиля

Прежде чем дать определение понятию «каскадирование», приведем пример проблемной ситуации, в которой выбор стиля окажется неоднозначным.

Пусть имеется фрагмент разметки (троеточие обозначает наличие других тегов того же уровня):

```
<div id="content">
    . . .
    <article>
        . . .
        <div class="frame">
            . . .
            <div class="text">
                . . .
                <h1></h1>
                . . .
            </div>
            . . .
        </div>
        . . .
    </article>
    . . .
</div>
```

В описании стилей заданы следующие селекторы:

```
article div.frame div.text h1 {

}

#content h1 {

}
```

Оба селектора контекстно обращаются к заголовку `<h1>`. Какой из этих стилей оформит тег, если формально подходят оба?

Чтобы ответить на этот вопрос, необходимо руководствоваться правилами CSS, которые выбирают стиль.

Специфичность селектора и каскадирование стилей

Понятие

Определение

Каскадирование *– это процесс выбора стиля по способу подключения, правилам наследования и его специфичности.*

Правила каскадирования позволяют разрешать неоднозначные ситуации выбора стиля (как в примере выше), когда одному элементу задано сразу несколько значений одного свойства. Для этого в CSS определен целый ряд правил, по которым рассчитывается приоритет стиля.

1. Приоритет типа стиля

В начале приоритет определяется по *типу стиля* (в порядке убывания важности):

1. стили переходов (свойство `transition`);
2. стили браузера с пометкой `!important`;
3. пользовательские стили с пометкой `!important`;
4. авторские стили, отмеченные `!important`;
5. стили анимации;
6. обыкновенные авторские стили;
7. обыкновенные пользовательские стили;
8. стили браузера по умолчанию.

Обычно стили с пометкой **`!important`** лучше избегать: чаще всего она ставится свойству, чтобы обеспечить его срабатывание «наверняка» или в нарушение правил каскадирования. По существу использование `!important` является крайней мерой влияния на приоритет стиля, например:

```
color: red !important;
```

Для грамотной структуры разметки и стилей, как правило, необходимости в `!important` не возникает.

2. *Порядок в коде и наследование*

Для стилей, которые имеют одинаковый приоритет (*специфичность*, далее подробнее), дополнительно проверяются *правила группировки* (см. пункт 3.5.1). Стиль, описанный ниже в документе, имеет более высокий приоритет, чем определенный выше (точнее, нижний стиль перекрывает верхний):

```
.rule {
    /* Начальное значение */
    background: red;
}

.rule {
    /* Переопределяем значение */
    background: blue;
}
```

3. *Специфичность и правила ее вычисления*

Определение

Специфичность *– это приоритет стиля селектора.*

Специфичность можно отождествить с некоторым числом, которое отражает приоритет какого-либо правила (селектора). Она вычисляется по суммарному показателю специфичности каждого селектора, который входит в составной селектор.

Сопоставим каждому селектору четыре числа:

`(x, y, z, w)`

Считаем, что каждое число определяет свой уровень приоритета, который растет справа налево, т.е. при любых `x`, `y`, `z`, `w` справедливы соотношения (■ – любое число):

`(x,■,■,■) > (0,y,■,■) > (0,0,z,■) > (0,0,0,w).`

Наименьший приоритет равен `(0,0,0,0)`. Его имеет универсальный селектор `*`, свойства которого автоматически наследуются всеми тегами.

Определение

Алгоритм вычисления специфичности селектора:

1. ***Встроенный стиль** тега (inline-стиль) добавляет (1,0,0,0) к специфичности.*
2. *Каждый **`#идентификатор`** в селекторе добавляет (0,1,0,0) к специфичности.*
3. *Каждый **`.класс`**, **`:псевдокласс`** и **`[атрибут]`** селектора добавляет по (0,0,1,0) к специфичности.*
4. *Каждый **тег** в селекторе добавляет (0,0,0,1) к специфичности.*

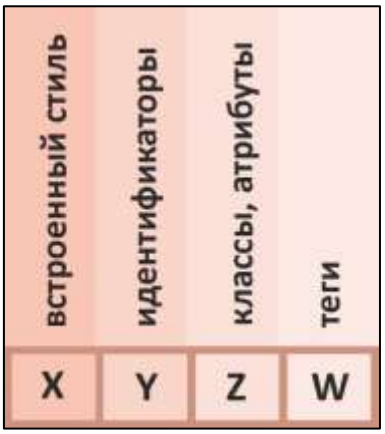


Рис. 3.93. Рост уровня приоритета селектора (справа налево)

Вернемся к примеру выше и вычислим суммарную специфичность каждого селектора.

Селектор `article div.frame div.text h1` имеет специфичность (0,0,2,4):

- содержит 2 класса (`.frame`, `.text`);
- содержит 4 тега (`article`, `div`, `div`, `h1`).

Селектор `#content h1` имеет специфичность (0,1,0,1):

- содержит 1 идентификатор (`#content`);
- содержит 1 тег (`h1`).

Таким образом тег `<h1>` из приведенного фрагмента разметки будет оформлен стилями, которые описаны вторым селектором. Это связано с тем, что он обладает идентификатором, поэтому уровень его специфичности превосходит любое число классов и тегов (рис. 3.94).

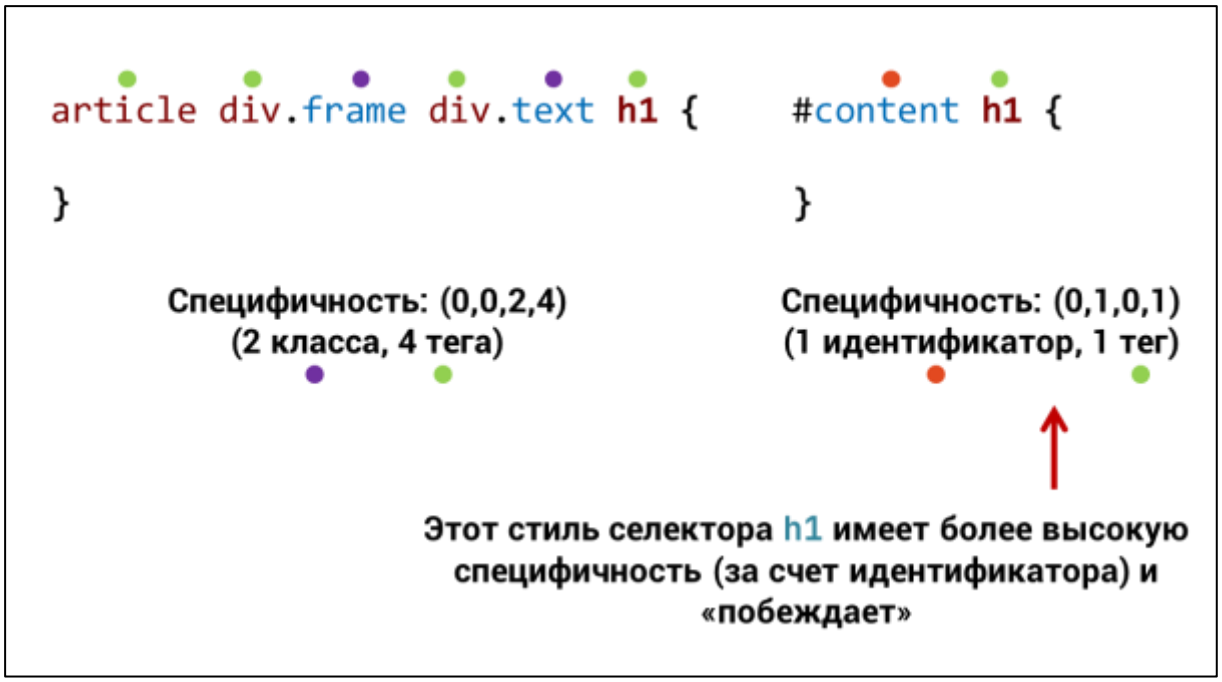


Рис. 3.94. Подсчет специфичности селекторов

Это важно знать!

Если требуемый селектор не принимает заданные свойства, проверьте его специфичность – она должна быть наибольшей среди всех, которые указывают на искомый селектор.

Приемы работы с Visual Studio Code

Считать специфичность «вручную» не нужно! В редакторе Visual Studio Code при наведении курсора на селектор специфичность отображается:

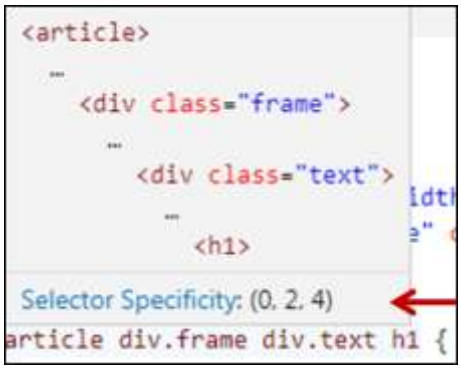


Рис. 3.95. Значение специфичности для селектора

Методология БЭМ как альтернатива контекстному вложению

Несмотря на то, что работа с контекстными селекторами позволяет по-разному оформлять элементы при вложении и зачастую сокращать объем описываемых классов, в промышленных проектах обычно используется иной подход. А именно – упор делается на описании классов каждому элементу.

Среди существующих подходов популярной является методология *БЭМ* (*Блок-Элемент-Модификатор*) – набор определенных соглашений веб-разработчиков, предполагающий принцип разделения интерфейса на независимые блоки. Он позволяет легко и быстро разрабатывать интерфейсы любой сложности и повторно использовать существующий код, избегая многократного дублирования.

БЭМ устраняет некоторые недостатки контекстного обращения к селекторам, проблемы конфликта стилей, упрощает структуру CSS-файлов и делает ее более однородной и хорошо систематизированной.

Основные компоненты БЭМ были достаточно подробно описаны в пунктах 1.8.2 и 3.2.4 (рекомендуем повторить этот материал).

Рис. 3.96. БЭМ – одна из методологий системной верстки больших проектов

Важное замечание!

Особенности каскадирования – крайне важная тема, которая определяет основные принципы выборки стилей по заданным селекторам и их расположению. Следует внимательно изучить этот вопрос, чтобы в дальнейшем не возникало затруднений и было проще находить причины некорректной работы стилей.

3.5.5. Селекторы атрибутов

Синтаксис

Рассмотрим еще один тип селекторов, который может активировать CSS-стили по наличию определенных *атрибутов тега* или установленных в них *значениях*.

Определение

```
[атрибут] { }
```

*Устанавливает стиль **любому тегу**, имеющему указанный **атрибут**.*

```
тег[атрибут] { }
```

*Устанавливает стиль **определенному тегу**, имеющему указанный **атрибут**.*

```
[атрибут="значение"] { }
```

*Устанавливает стиль элементу с заданным **атрибутом** и **конкретным значением**.*

```
[атрибут^="значение"] { }
[атрибут$="значение"] { }
[атрибут*="значение"] { }
```

*Позволяет **отбирать** элементы с **атрибутом**, **значение** которого*

- ***начинается**,*
- ***завершается**,*
- *либо **содержит** указанный в значении текстом.*

Селекторы атрибутов часто используются для выборки стилей родственных элементов (например, управляющих компонент форм), а также при выборе названия классов, имеющих единый префикс (такие используются, например, при оформлении адаптивных колонок).

Пример использования

Ограничимся простым примером использования селекторов атрибута. Пусть требуется применить некоторый стиль форматирования для текстовых полей внутри формы:

Глава 3\Селекторы атрибутов\index.html

```
<h1>Селекторы атрибутов</h1>
<form>
    <label for="age">Ваш возраст:</label>
    <input type="text" name="age">
    <br>
    <label for="fullname">ФИО:</label>
    <input type="text" name="fullname">
    <br>
    <input type="checkbox" name="rules_agree">
    <label for="rules_agree">Согласие на обработку
        персональных данных</label>
</form>
```

Глава 3\Селекторы атрибутов\style.css

```
input[type="text"] {
    margin: 4px 0;
}
```

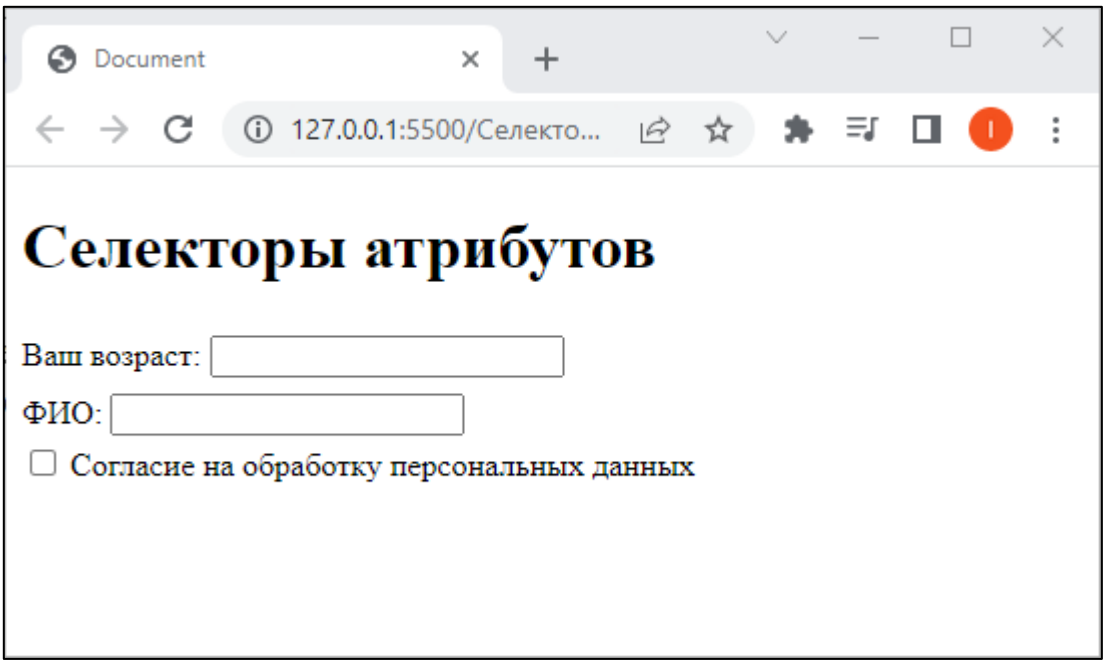


Рис. 3.97. Использование селекторов-атрибутов

Более полное описание и практические примеры использования селекторов атрибутов можно найти по указанным ссылкам:

- селекторы атрибутов: webref.ru;
- селектор по атрибуту: doka.guide.

Вопросы для самопроверки

1. Опишите основные правила наследования стилей.
2. Для каких целей используется группировка стилей?
3. Чем отличается контекстное обращение к селектору от дочернего? Привести пример, когда оба варианта будут давать одинаковый результат.
4. В чем заключаются достоинства и недостатки контекстного обращения к селекторам?
5. С какой проблемой можно столкнуться при определении стиля элемента?
6. Что представляет собой каскадирование и на базе каких правил оно функционирует?
7. Опишите алгоритм расчета специфичности и продемонстрируйте ее на конкретном примере.
8. Приведите примеры ситуаций, когда селекторы атрибутов удобны в использовании.

Практикум

Задание 1

1. Создайте каталог *Правила каскадирования* с типовым начальным HTML-файлом.
2. Скопируйте предложенную ниже разметку.
3. Используя полученные знания по правилам каскадирования, опишите составные селекторы (оформлять элементы не требуется), которые позволяют оформить:
   a. все ссылки внутри блока `main-box`.
   b. все ссылки любого блока `nav` внутри раздела `header`;
   c. только ссылки из раздела меню и только, если они являются элементами списка;

d. обратиться ко всем изображениям, находящимся в блоке с классом `gallery`;
e. укажите хотя-бы один способ обращения только на ссылку «О ресурсе», не затронув при этом оформление других ссылок (если необходимо, дополните HTML-код).

Глава 3\Правила каскадирования\index.html

```
<div class="main-box">
    <header>
        <img class="logo" src="#">
        <h1>Текстовые редакторы</h1>
        <nav class="menu">
            <h2>Меню</h2>
            <ul>
                <li><a href="#">Notepad++</a></li>
                <li><a href="#">Vim</a></li>
                <li><a href="#">Emacs</a></li>
                <li><a href="#">Sublime Text</a></li>
                <li><a href="#">Visual Studio
                        Code</a></li>
            </ul>
        </nav>
        <nav class="about">
            <a href="#">О ресурсе</a>
            <a href="#">Об авторе</a>
        </nav>
    </header>
    <main>
        <div class="gallery">
            <img src="#">
            <img src="#">
            <img src="#">
        </div>
    </main>
    <footer>
    </footer>
</div>
```

Задание 2

1. Создайте каталог *Базы данных* с типовым начальным HTML-файлом и CSS-стилями.
2. Внимательно изучите предложенную HTML-разметку страницы.
3. Также изучите CSS-свойства, особенно, как используется контекстное вложение элементов:
   - Найдите в документе прием по подключению шрифта с ресурса Google Fonts; измените его на шрифт Lato.
   - Как удается достичь разного оформления ссылок в различных частях разметки?
   - Что делает класс `box`?
   - Что делает класс `pictures`?
4. Дополните HTML-разметку кодом по классификации баз данных. Текст возьмите по ссылке из «подвала» сайта. CSS-стилизацию расширять не требуется. Должен получиться результат, как изображено на рис. 3.98-рис. 3.99.

Глава 3\Базы данных\index.html

```
<div class="main-box">
    <header>
        <h1>Базы данных</h1>
        <nav class="menu">
            <a href="#">MS Access</a>
            <a href="#">MS SQL Server</a>
            <a href="#">MySQL</a>
            <a href="#">PHPMyAdmin</a>
            <a href="#">MySQL Workbench</a>
        </nav>
    </header>

    <main>
        <div class="box">
            <h2>Введение</h2>
            <p><strong>База данных</strong> –
            представленная в объективной форме
            совокупность самостоятельных материалов
            (статей, расчётов, нормативных актов,
            судебных решений и иных подобных материалов),
            систематизированных таким образом, чтобы эти
            материалы могли быть найдены и обработаны с
```

```
                помощью электронной вычислительной
                машины.</p>
                <p>Многие специалисты указывают на
                распространённую ошибку, состоящую в
                некорректном использовании термина «база
                данных» вместо термина «система управления
                базами данных», и указывают на необходимость
                различения этих понятий.</p>
                <div class="pictures">
                    <img src="images/db_1.png">
                    <img src="images/db_2.svg">
                    <img src="images/db_3.png">
                </div>
            </div>
        </main>
        <footer>
            <p>Информация предоставлена ресурсом
                <a href="https://ru.wikipedia.org/">
                    wikipedia.org
                </a>
            </p>
        </footer>
    </div>
```

Глава 3\Базы данных\style.css

```
/* Общие свойства тела страницы */
body {
    background: #C4E3DC;
    font: 18px 'Arimo', serif;
    color: #395442;
}

/* Основной блок */
.main-box {
    min-width: 540px;
    max-width: 760px;
    margin: 0 auto;  /* выравнивание по центру */
    background: #E1FAF4;
}

/* Шапка */
header {
    background: #4A656B;
    padding: 10px 20px;
}
```

```css
/* Стили заголовков */
h1, h2, h3 { font-family: Calibri; }

h1 { color: white; }
h2 { color: #4E2D26; }
h3 { color: #8C5244; }

/* Гиперссылки */
a {
    color: rgb(8, 77, 122);
    text-decoration: none;
    font-weight: bold;
}

/* Абзацы */
p { line-height: 1.4; }

/* Элементы списков */
ul > li {
    line-height: 1.4;
    margin-bottom: 7px;
}

/* Блок фона для оформления текста */
.box {
    background: #E9FBF7;
    margin: 20px;
    padding: 15px 25px;
    border-radius: 5px;
    border: 1px solid #C4E3DC;
}

/* Гиперссылки в области меню */
.menu a {
    display: inline-block;
    font: bold 1.0em Calibri;
    color: #D7F9F1;
    padding: 4px 10px;
    border-right: 2px solid #7BA097;
}

.menu a:hover { color: #FFF; }

/* Блок для оформления изображений */
```

```
.pictures {
    display: table;
    margin: 0 auto;
}

/* Изображения внутри блока */
.pictures > img {
    height: 150px;
    margin: 5px;
}

/* Подвал */
footer {
    background: #2F4537;
    color: #E9FBF7;
    padding: 20px;
}

/* Ссылки в подвале */
footer a {
    color: #B2C897;
}
```

Задание 3

1. Создайте копию каталога *Базы данных*, выполненного в предыдущем задании.
2. Скорректируйте разметку, добавив классы тегам основной структуры (там, где отсутствуют).
3. Также исправьте стили, заменив селекторы тегов на селекторы классов.

Задание 4

1. Создайте каталог *Каскадирование* и копируйте в него реализацию примера, разобранного в одноименном примере из пункта 3.5.3.
2. Оформите ссылку «Главная» другим цветом: используйте полученные знания о контекстных и дочерних селекторах, каскадировании (рис. 3.100).
3. При необходимости можно внести правку в HTML.
4. Какие варианты возможны? Предложите несколько альтернативных способов оформления этой гиперссылки.

Задание 5

1. Создайте каталог *Контекстные и дочерние селекторы* с типовым начальным HTML-файлом и CSS-стилями.
2. Скопируйте предложенный в конце задания код разметки и стилей.
3. Допишите свойства для соответствующих селекторов, чтобы получить следующий результат, как на рис. 3.101 (подсказки в комментариях).

Глава 3\Контекстные и дочерние селекторы\index.html

```
<main id="box">
    <div class="container">
        <h1>Каскад</h1>
        <ul id="menu">
            <li><a href="#contextual">Контекстные</a>
                селекторы</li>
            <li><a href="#subsidiaries">Дочерние</a>
                селекторы</li>
        </ul>
    </div>

    <div class="container">
        <a name="contextual"></a>
        <h2>Контекстные селекторы</h2>
        <p>Мощный инструмент CSS, которым следует
        овладеть – контекстные (вложенные) селекторы. Он
        позволяет задать указанные свойства только тем
        селекторам, которые вложены в указанный
        (родительский). В качестве селекторов могут
        выступать теги, классы, идентификаторы и их
        комбинации.</p>
        <p>Чтобы показать зависимость, дочерний элемент
        ставится через пробел.</p>
    </div>

    <div class="container">
        <a name="subsidiaries"></a>
        <h2>Дочерние селекторы</h2>
        <p>Дочерние селекторы работают по принципу,
        похожему на работу контекстных селекторов. Но
        есть одно важное отличие: свойства
        распространяются только на прямых потомков
        (детей); более глубокие вложения уже
```

```
        «неподвластны» изменениям.</p>
        <p>Для указания зависимости используется знак
        «>».</p>
    </div>
</main>
```

Глава 3\Базы данных\style.css

```
body {
    background: #888;
    font: 16px Calibri, Arial;
}

/*  Основной блок  */
#box {
    background: #bbb;
    width: 560px;
    padding: 10px 20px;
    margin: 0 auto;
}

/*  Блоки разделов  */
.container {
    /*  серый фон, внутренние отступы,
        внешний отступ сверху 10px */
}

/*  Список в разделе меню  */
#menu { /* размер шрифта 120% */ }

/*  Ссылки в разделе меню  */
#menu > li > a {
    /* жирное начертание,
       вишневый цвет */
}

/*  Основные элементы разметки текста  */
h1, h2 { /* выравниваются по центру */ }

h2 { /* темно бирюзовй цвет */ }

p { /* междустрочный интервал - 1.2 */ }
```

Рис. 3.98. Образец выполнения задания 2 (часть 1)

средой постоянного хранения является периферийная энергонезависимая память (вторичная память) — как правило жёсткий диск.

- В оперативную память СУБД помещает лишь кэш и данные для текущей обработки.
- В оперативной памяти (англ. in-memory database, memory-resident database, main memory database): все данные на стадии исполнения находятся в оперативной памяти.
- В третичной памяти (англ. tertiary database): средой постоянного хранения является отсоединяемое от сервера устройство массового хранения (третичная память), как правило на основе магнитных лент или оптических дисков.
- Во вторичной памяти сервера хранится лишь каталог данных третичной памяти, файловый кэш и данные для текущей обработки; загрузка же самих данных требует специальной процедуры.

Классификация по содержимому

- Географическая
- Историческая
- Научная
- Мультимедийная
- Клиентская.

Классификация по степени распределённости

- Централизованная, или сосредоточенная (англ. centralized database): БД, полностью поддерживаемая на одном компьютере.
- Распределённая БД (англ. distributed database) — составные части которой размещаются в различных узлах компьютерной сети в соответствии с каким-либо критерием.
- Неоднородная (англ. heterogeneous distributed database): фрагменты распределённой БД в разных узлах сети поддерживаются средствами более одной СУБД.
- Однородная (англ. homogeneous distributed database): фрагменты распределённой БД в разных узлах сети поддерживаются средствами одной и той же СУБД.
- Фрагментированная, или секционированная (англ. partitioned database): методом распределения данных является фрагментирование (партиционирование, секционирование), вертикальное или горизонтальное.
- Тиражированная (англ. replicated database): методом распределения данных является тиражирование (репликация).

Информация предотавлена ресурсом wikipedia.org

Рис. 3.99. Образец выполнения задания 2 (часть 2)

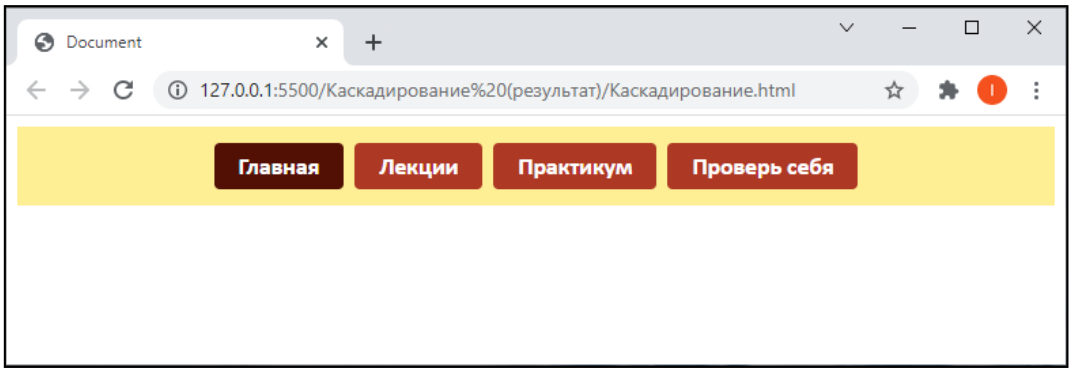


Рис. 3.100. Образец выполнения задания 4

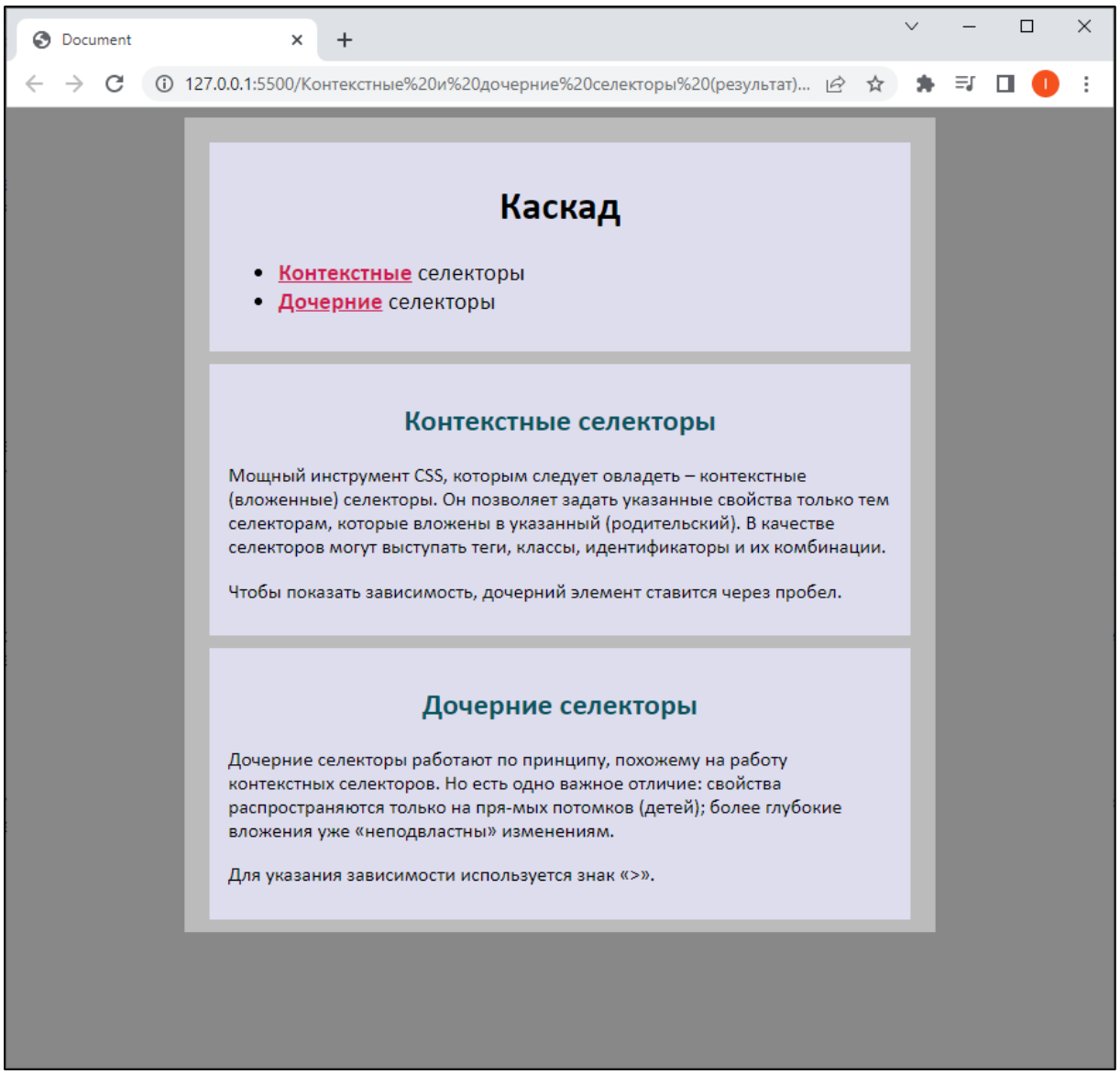


Рис. 3.101. Образец выполнения задания 5

3.6. Плавающие блоки

3.6.1. Плавающие блоки в разметке

Понятие float-блока

Определение

Плавающий блок (float-box) *– это блок, способный «прилипать» к левой либо правой стороне блока-родителя и допускающий обтекание.*

```
float: left | right | none | inherit
```

Делает ***плавающий блок****. Элемент выходит из нормального потока и прижимается к левой или правой границе родителя. При этом текст и встроенные элементы родительского блока будут его обтекать.*

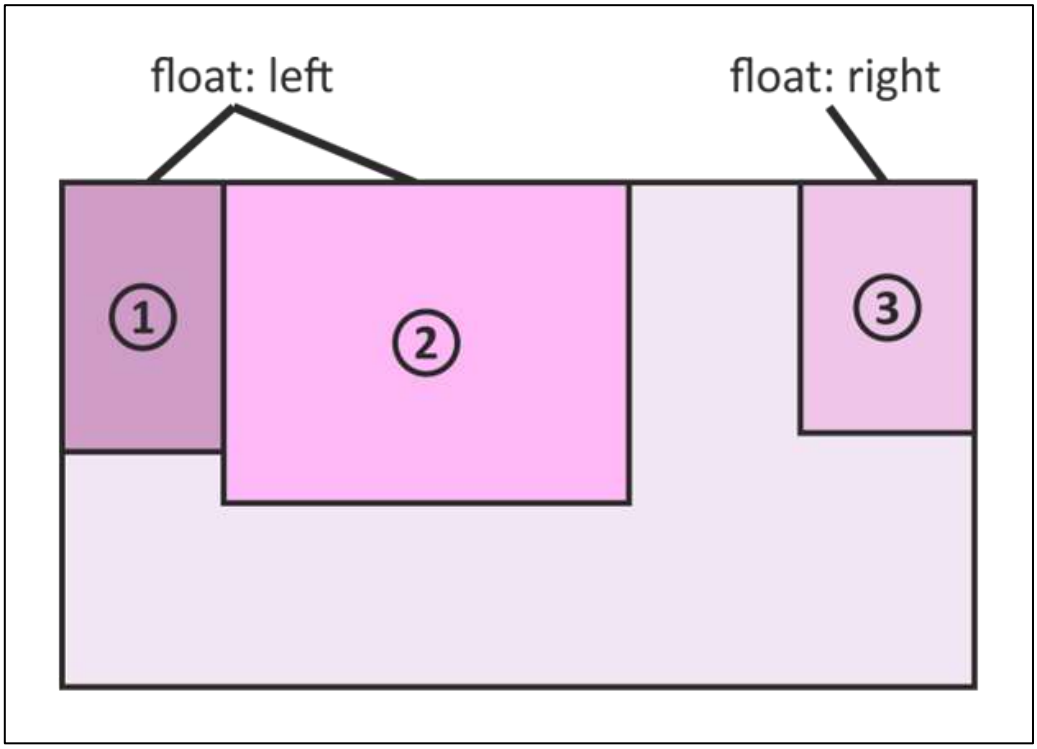


Рис. 3.102. Поведение float-блоков

Особенности плавающих блоков

1. Любой элемент со свойством `float` автоматически становится блочным.
2. Если ширина float-блока не задана, то она определяется шириной содержимого этого блока или шириной его родителя.
3. Float-блок стремится занять максимально левую / правую верхнюю область (если позволяет ширина).
4. Float-блок исключается из *нормального потока*. Это означает, что:
   a. другие блоки его «не видят» (в частности – его родительский);
   b. inline-элементы (строчные) его обтекают, в т.ч. внутри блока родителя.
5. Соседние float-блоки плотно «прилипают» друг к другу согласно порядку их следования.
6. Плавающий блок может быть родителем для любых других элементов, в т.ч. плавающих.

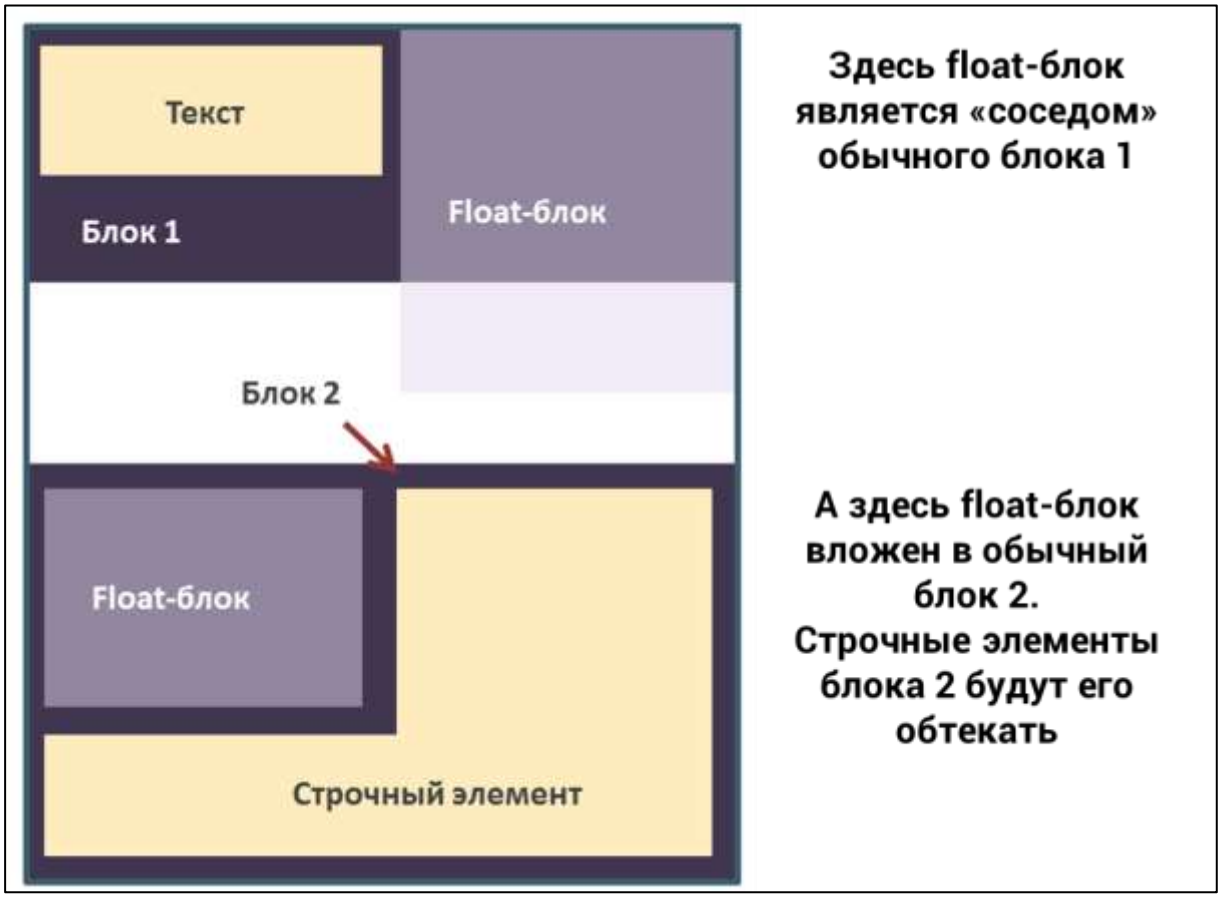


Рис. 3.103. Особенности поведения float-блока

Обтекание родительским блоком

Ранее было отмечено, что float-блок выходит из нормального потока и становится «невидимым» для обычных блоков, а его высота не влияет на высоту родителя. Поэтому он может выйти за границу своего родителя.

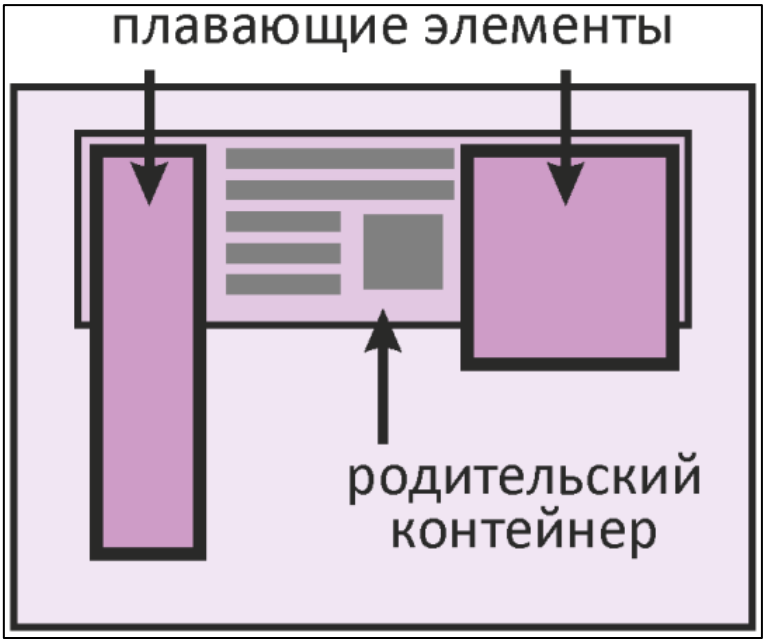


Рис. 3.104. Особенность плавающего блока: выход за границу контейнера родительского блока

Зачастую такой эффект нежелателен. Существует несколько приемов, позволяющих включить обтекание родителем вложенных в него float-блоков:

- сделать родитель также float-блоком;
- добавить ниже пустой блок `<div>` со свойством `clear: both`;
- описать специальный класс `clearfix`;
- воспользоваться свойством `overflow`.

Ограничимся последним вариантом.

Определение

```
oveflow: visible | hidden | scroll | auto
```

*Свойство **управляет содержимым** блочного элемента и указывает, что требуется сделать, когда его размер превышает доступную длину или ширину (т.е. они зафиксированы).*
***Переполненное** содержимое может выходить за границы блока (**visible**), обрезаться (**hidden**), добавлять полосу прокрутки в любом случае (**scroll**) либо только при переполнении (**auto**).*

Для включения обтекания можно использовать свойство `overflow` со значением `auto` или `hidden`. Оно указывается родительскому блоку.

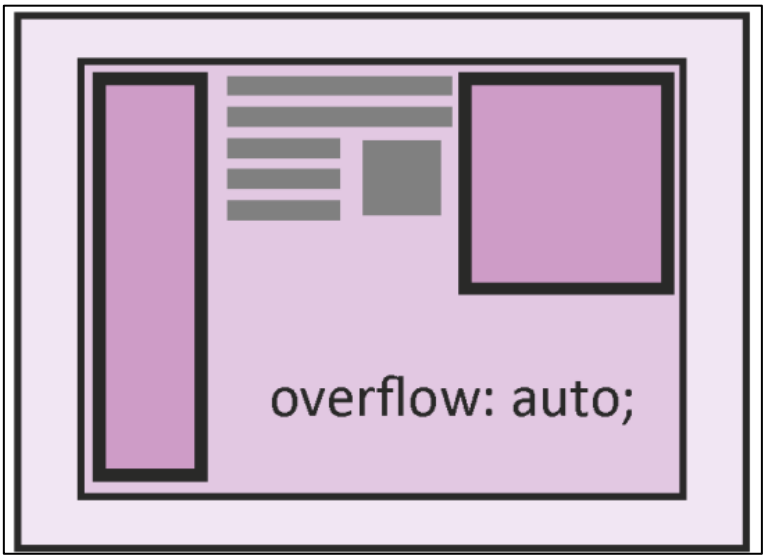


Рис. 3.105. Обтекание родительским блоком для плавающих блоков

Запрет обтекания в нормальном потоке

Содержимое блоков, находящихся рядом с плавающим блоком, вынуждено его обтекать. Если подобное требуется исключить, используйте свойство `clear`.

Блок с этим свойством «видит» плавающий блок и не позволяет ему обтекание.

Определение

```
clear: left | right | both
```

Запрещает обтекание *слева, справа или с обоих сторон. Указывается объекту, который может быть подвергнут обтеканию плавающим блоком.*

Рис. 3.106. Запрет обтекания для float-блока с любой стороны

Управление моделью расчета размера

Ранее было отмечено, что по умолчанию свойства `width` и `height` определяют ширину и высоту объекта в разметке только по содержимому, не включая внутренние отступы и границы. Такое правило определяется стандартом CSS.

Однако на практике такой подход далеко не всегда удобен.

Например, для блока

```
.container {
    width: 300px;
    padding: 10px;
}
```

фактическая ширина составит 320px, поскольку левый и правый отступы также влияют на размер занимаемой области.

Рис. 3.107. Фактическая ширина и высота блока

Чтобы не тратить лишнее время на подгонку всех этих параметров, можно воспользоваться свойством **box-sizing**. Его задача изменить алгоритм вычисления размеров контейнера, в котором размещается элемент.

Определение

```
box-sizing: content-box | border-box | padding-box |
            inherit
```

*Свойство **меняет** стандартный **алгоритм подсчета ширины** и **высоты блока**.*
В зависимости от выбранного значения:

- ***content-box*** *– размер контента определяется только свойствами* `width` *и* `height`*;*
- ***border-box*** *– в размер дополнительно включаются внутренние отступы и границы, но не внешние поля;*
- ***padding-box*** *– в размер дополнительно включаются внутренние отступы, но не границы и внешние поля;*
- ***inherit*** *– наследует значение у родителя.*

Как правило, `box-sizing` применяют для универсального селектора *, отвечающего за стили всех элементов. Этот селектор имеет наименьшую специфичность стилей (0,0,0,0), поэтому распространяется на все теги.

Чтобы учесть в ширине/высоте контейнера объекта внутренние отступы и толщину границ, установите в самом начале файла с CSS-стилями следующее правило:

```
* {
    box-sizing: border-box;
}
```

Пример использования float-блоков

Разметим плавающими блоками контейнеры, которые можно использовать далее для наполнения другим содержимым. Для простоты реализуем общий класс `container` и отдельные классы, устанавливающие свою заливку каждому блоку.

Глава 3\Плавающие блоки\index.html

```
<main>
    <div class="container box-1">
        <p>Lorem ipsum ...</p>
    </div>
    <div class="container box-2">
        <p>Lorem ipsum ...</p>
    </div>
    <div class="container box-3">
        <p>Lorem ipsum ...</p>
    </div>
    <div class="container box-4">
        <p>Lorem ipsum ...</p>
    </div>
</main>
```

Глава 3\Плавающие блоки\style.css

```
/* Включаем в алгоритм вычисления ширины и высоты отступы
   и толщину рамки */
* { box-sizing: border-box; }

/* Делаем "резиновой" ширину основного блока */
main {
    min-width: 300px;
    max-width: 900px;
    background: #DDD;
    overflow: auto;
}

/* Общее оформление плавающих контейнеров */
.container {
    float: left;
    width: 300px;
    padding: 20px;
    border: 1px solid black;
}

/* Разные цвета контейнеров */
.box-1 { background: #e4f2ef; }
.box-2 { background: #cee8e2; }
.box-3 { background: #b3cfc8; }
.box-4 { background: #8da8a2; }
```

Пока ширина окна браузера не менее 900px, в основной блок `main` вмещается по ширине три вложенных, а четвертый вытесняется вниз:

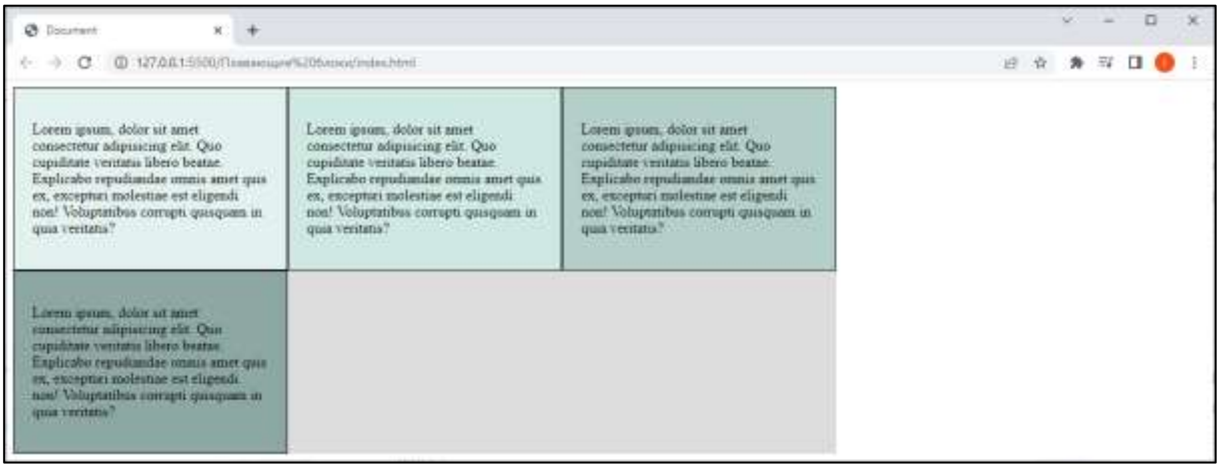


Рис. 3.108. Блоки заняли доступную ширину родителя

При уменьшении ширины окна основной блок начнет уменьшаться и вытеснит третий блок вниз:

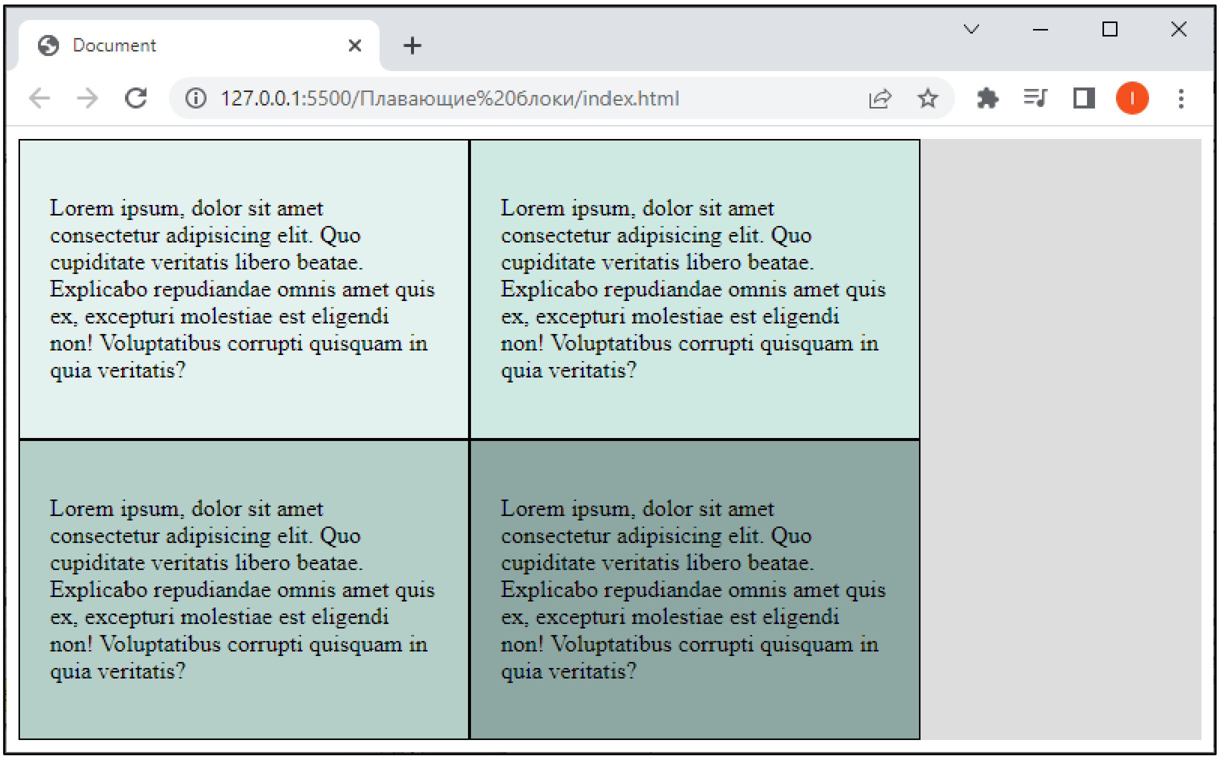


Рис. 3.109. Ширина родительского блока вытесняет третий блок вниз

Наконец, если ширина окна будет менее 600px, то все блоки выстроятся в одну колонку:

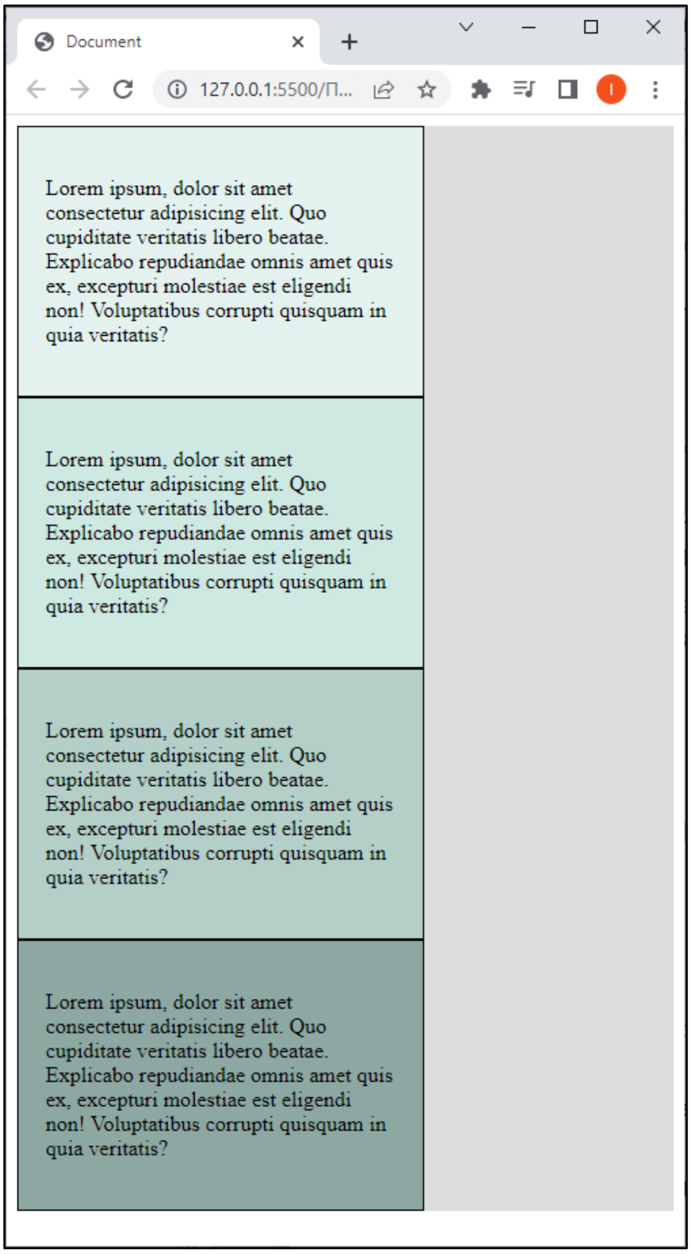


Рис. 3.110. Ширина основного блока позволяет разместить не более одного блока в «ряду»

Это полезно знать!

В приведенном примере прослеживается простейшая идея, которая позволяет использовать float-блоки в реализации адаптивного дизайна: «расслоение» блоков контента на отдельные полосы для малых экранов. Далее будет показано ее практическое развитие.

3.6.2. Использование в разметке колонок

Создание колонок

Предыдущий пример показывает, что float-блоки прекрасно подойдут для реализации контейнеров колонок (смежных блоков). В определенном смысле они дают больше возможностей, чем работа с блочно-строчными элементами.

Чтобы реализовать разметку в несколько колонок, необходимо:

1. блоку родителю задать свойство `overflow: auto`;
2. блокам-колонкам определить одинаковую ширину (разумеется, это необязательно);
3. для гибкой верстки ширину колонок удобно задавать в процентах.

Если суммарная ширина колонок равна ширине родителя, то колонки плотно заполняют ширину блока родителя и вытесняют другие блоки вниз.

Однако обычно в этом случае высота колонок из-за контента получается разной, что может привести к такой плитке:

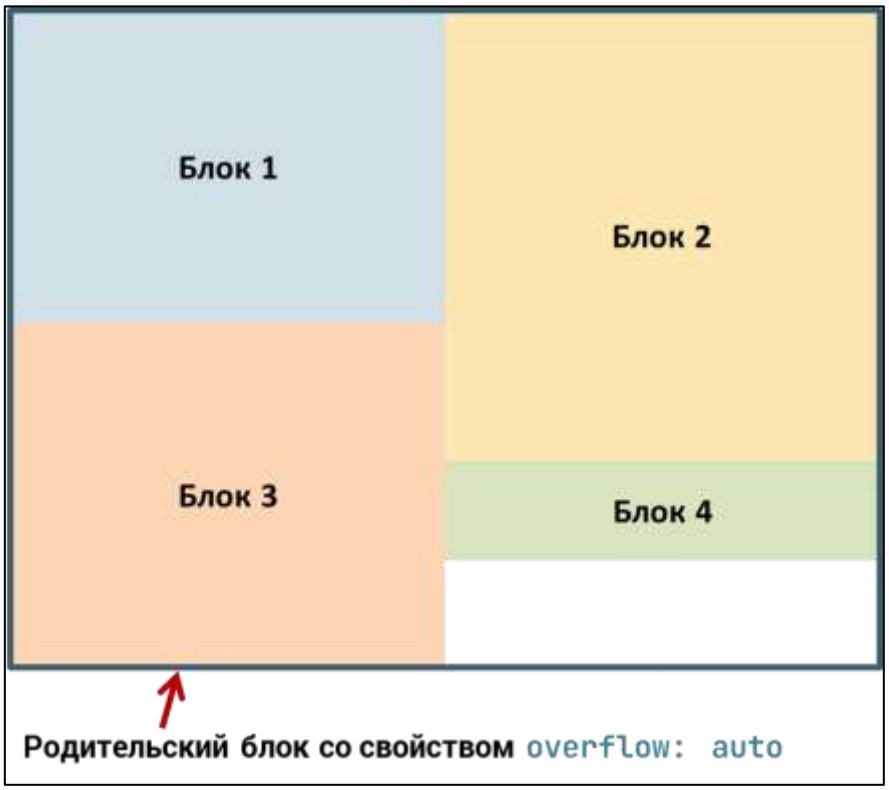


Рис. 3.111. Неравномерное заполнение блоков

Деление на слои

Если необходимо отделить колонки по «уровням», то достаточно ввести промежуточный -блок родитель, содержащий одно свойство: overflow: auto:

```
.container {
    overflow: auto;
}

<div class="content">
    <div class="container">
        <div class="block-1">Блок 1</div>
        <div class="block-2">Блок 2</div>
    </div>
    <div class="container">
        <div class="block-3">Блок 3</div>
        <div class="block-4">Блок 4</div>
    </div>
</div>
```

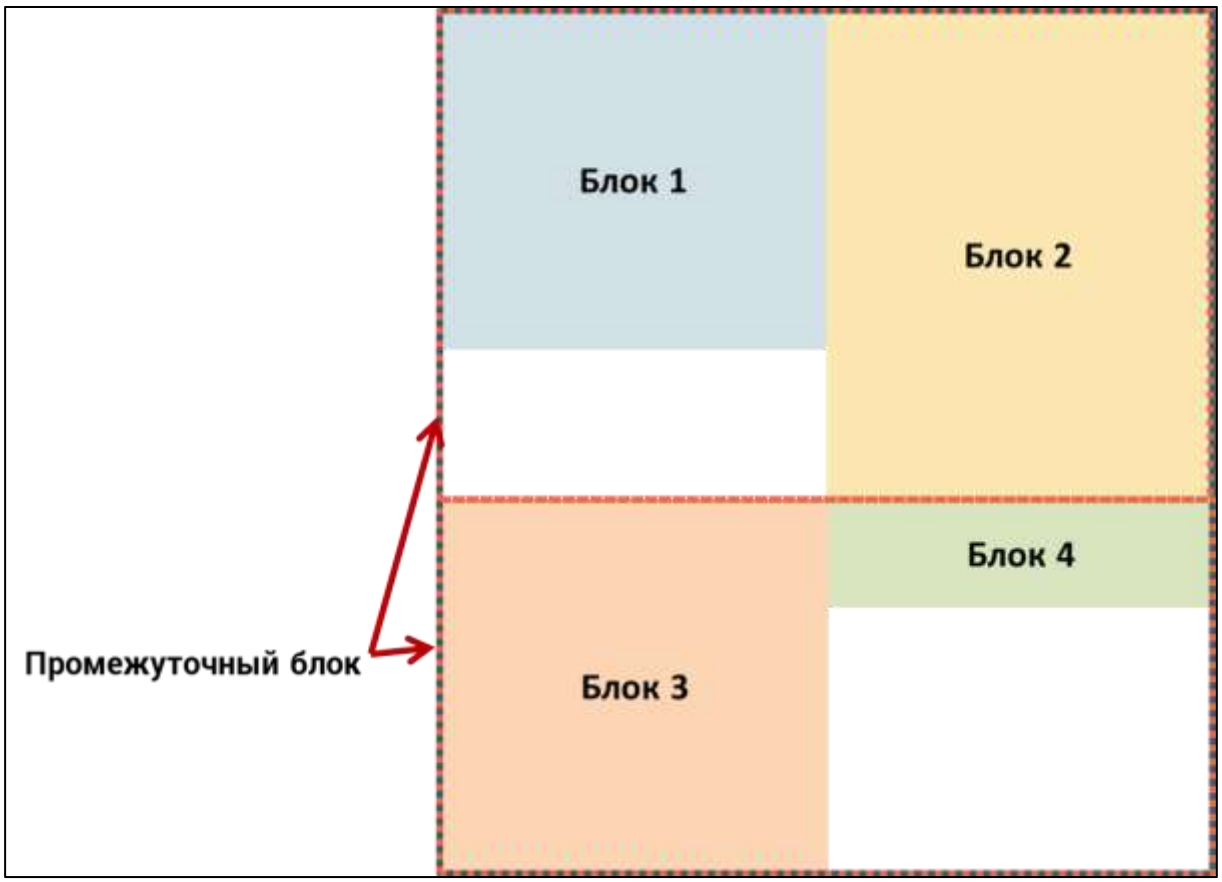


Рис. 3.112. Разделение float-колонок на отдельные уровни

Блоки-обертки

Классы на подобии `container` называют *оберткой* (*wrapper*). Такой класс можно применить где угодно: его действие будет одинаковым для всех плавающих или обычных блоков.

Например, в одном из внутренних float-блоков `container` можно вновь использовать для деления разметки на несколько колонок:

```
<div class="content">
    <div class="container">
        <div class="block-1">Блок 1</div>
        <div class="block-2">
            <div class="container">
                <div class="block-2.1">Блок 2.1</div>
                <div class="block-2.2">Блок 2.2</div>
            </div>
            <div class="block-2.3">Блок 2.3</div>
        </div>
    </div>

    <div class="container">
        <div class="block-3">Блок 3</div>
        <div class="block-4">Блок 4</div>
    </div>
</div>
```

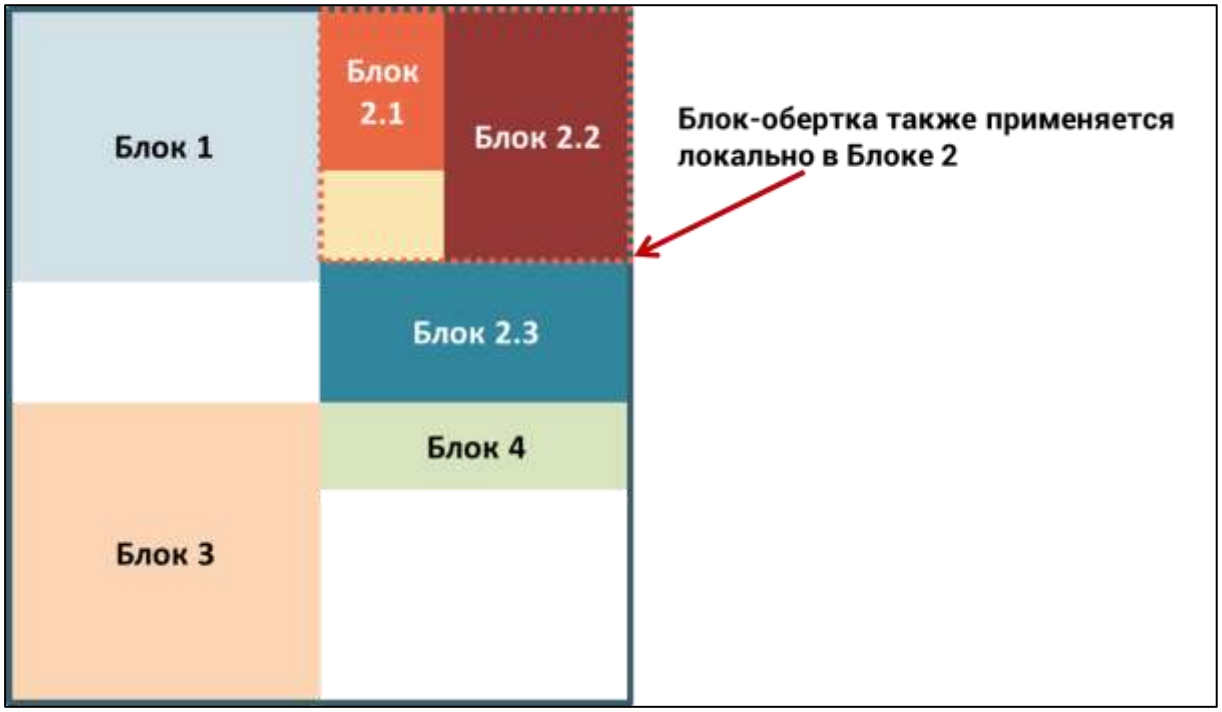


Рис. 3.113. Использование классов-оберток

Пример разметки веб-страницы

Техническое задание

Пусть требуется реализовать веб-страницу по простому макету разметки с фиксированной шириной основного контейнера:

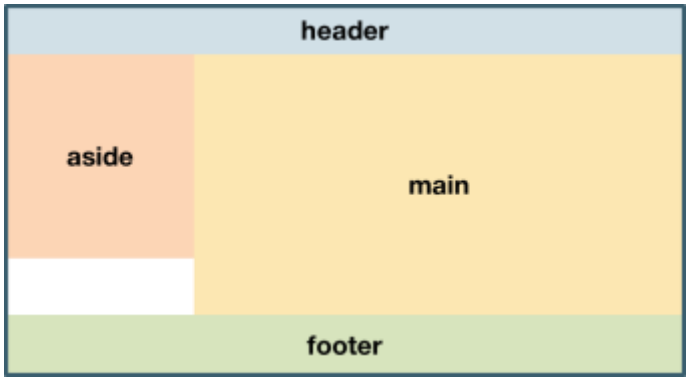


Рис. 3.114. Целевой макет

Представленный макет позволит разметить, например, следующую простую страницу:

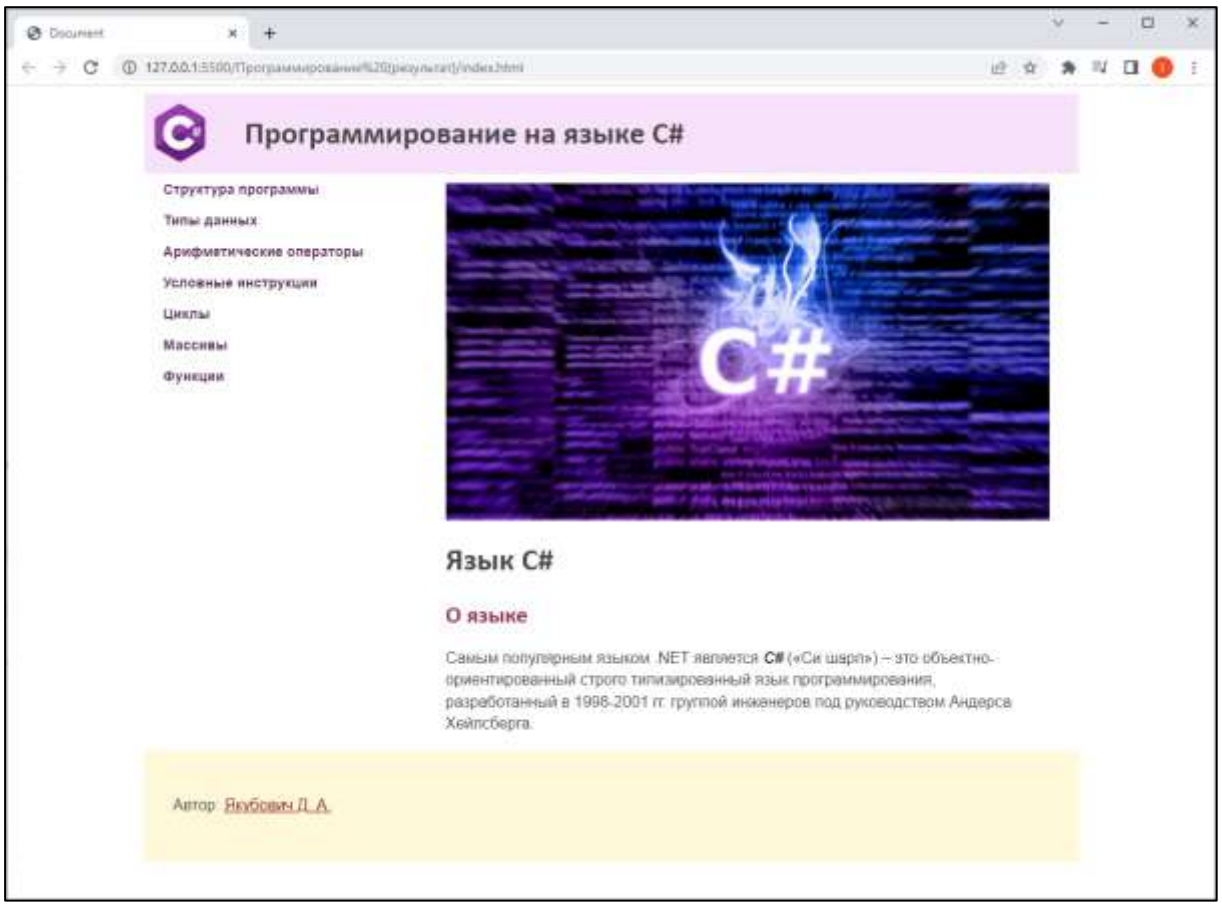


Рис. 3.115. Итоговое оформление страницы

1. *Подготовка файлов проекта*

Осуществим последовательную разметку веб-страницы, начиная от ее HTML-«скелета» и заканчивая поэтапной детализацией CSS-стилей отдельных элементов.

В начале создаем отдельный каталог *Программирование*. Размещаем в нем:

- новый HTML файл *index.html* с шаблонной начальной разметкой;
- файл CSS *style.css*;
- подключаем таблицу стилей к основному файлу;
- добавляем каталог *images*, внутри которого разместим логотип и необходимые изображения.

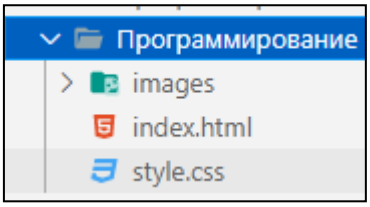


Рис. 3.116. Структура проекта

2. *Разметка каркаса*

Задаем разметку основных блоков (шапка, боковое меню, содержимое, подвал):

Глава 3\Программирование\index.html

```
<div class="main-box">
    <header>

    </header>
    <aside>

    </aside>
    <main>

    </main>
    <footer>

    </footer>
</div>
```

Описываем свойства блоков (боковая панель и контент суммарно равны ширине 960px, т.е. ширине основного блока `main-box`):

Глава 3\Программирование\style.css

```
* { box-sizing: border-box; }

.main-box, header, aside, main, footer {
    overflow: auto;
}

.main-box {
    width: 960px;
}

aside, main {
    float: left;
}

aside { width: 280px; }

main { width: 680px; }

header, footer {
    clear: both;
}
```

Здесь для удобства мы изменили алгоритм вычисления ширины и высоты контейнеров, учитывая внутренние отступы и толщину границ. Всем основным блокам включаем обтекание внутренних плавающих блоков, которые могут быть далее вложены в указанные.

Задаем фиксированную ширину основному блоку `main-box`. Блоки бокового меню `<aside>` и контента `<main>` делаем плавающими: они размечаются «колонками». Задаем им фиксированную ширину из расчета 280 + 680 = 960 (т.е. блоки плотно заполняют доступное пространство).

А вот блоки шапки и подвала занимают всю ширину. Чтобы float-блоки ниже/выше их не обтекали, включаем свойство `clear`.

3. Детализация блоков

Уточняем содержимое шапки сайта (она включает логотип и заголовок). Оформление элементов далее свяжем с подключаемыми классами:

Глава 3\Программирование\index.html

```
<header>
    <img src="images/logo.png"
         alt="Логотип"
         class="header__logo">
    <h1 class="header__title">
        Программирование на языке C#
    </h1>
</header>
```

Реализуем код для боковой панели меню: в блоке навигации перечислим ссылки на разделы сайта (для краткости адреса указывать не будем):

Глава 3\Программирование\index.html

```
<aside>
    <nav class="aside__menu">
        <a href="#">Структура программы</a>
        <a href="#">Типы данных</a>
        <a href="#">Арифметические операторы</a>
        <a href="#">Условные инструкции</a>
        <a href="#">Циклы</a>
        <a href="#">Массивы</a>
        <a href="#">Функции</a>
    </nav>
</aside>
```

Уточняем разметку области контента:

Глава 3\Программирование\index.html

```
<main>
    <img src="images/csharp.png" alt="Изображение"
         class="big-picture">
    <h1>Язык C#</h1>
    <h2>О языке</h2>
    <p>Самым популярным языком .NET является
    <strong>C#</strong> («Си шарп») – это объектно-
    ориентированный строго типизированный язык
    программирования, разработанный в 1998-2001 гг.
    группой инженеров под руководством Андерса
    Хейлсберга.</p>

    <!-- Далее код разметки текста -->
</main>
```

Вставляем информацию в «подвал»:

Глава 3\Программирование\index.html

```
<footer>
     <p>Автор: <a href="#">Якубович Д. А.</a></p>
</footer>
```

Описываем базовые стили основных тегов:

Глава 3\Программирование\style.css

```
body {
    font: 1.0em Lato, Arial, sans-serif;
    color: #444;
}

h1, h2, h3 {
    font-family: Calibri;
}

h2, h3 { color: #912a4b; }

p { line-height: 1.4; }

a { color: #8b0808; }
strong {
    font-weight: bold;
    font-style: italic;
}
```

Выравниваем по центру основной блок (дополняем уже созданный класс `main-box`):

Глава 3\Программирование\style.css

```
.main-box {
    width: 960px;
    margin: 0 auto;
}
```

Делаем заливку шапки и подвала. Для подвала также добавим отступов контенту и зададим минимально допустимую высоту:

Глава 3\Программирование\style.css

```
header {
    background: #F8E1FA;
}
```

```
footer {
    background: #fff9da;
    min-height: 100px;
    padding: 30px;
}
```

Оформляем логотип и название сайта. Их контейнеры также сделаем плавающими, чтобы расположить рядом:

Глава 3\Программирование\style.css

```
.header__logo {
    float: left;
    height: 80px;
    padding: 10px;
}

.header__title {
    float: left;
    margin-left: 30px;
}
```

Оформляем ссылки внутри боковой панели. Каждая ссылка делается блоком. Воспользуемся контекстным обращением к селекторами. В данном случае указывать селектор `nav` явно было необязательно, поскольку это единственная область навигации в документе:

Глава 3\Программирование\style.css

```
nav.aside__menu > a {
    display: block;
    padding: 8px 20px;
    text-decoration: none;
    color: #5F3164;
    font-weight: bold;
    font-size: 0.9em;
    transition: 0.15s all;
}
```

Дополнительно продемонстрируем прием работы с *псевдоклассом* **`:hover`**, который сработает при наведении курсора на ссылку и поменяет некоторые свойства. Время анимации задается в основном селекторе в свойстве **`transition`**:

Глава 3\Программирование\style.css

```
nav.aside__menu > a:hover {
    background: #5F3164;
    color: #FFF;
    border-left: 6px solid #AE40BD;
}
```

Для блока контента ограничимся лишь небольшим отступом от границ:

Глава 3\Программирование\style.css

```
main {
    width: 680px;
    padding: 0 30px;
}
```

Наконец, опишем класс для оформления больших изображений:

Глава 3\Программирование\style.css

```
.big-picture {
    display: block;
    width: 100%;
    margin: 10px 0;
}
```

Если все выполнено корректно, получится разметка страницы, изображенной на рис. 3.115.

Вопросы для самопроверки

1. Что представляют собой плавающие блоки и какими особенностями они обладают?
2. Каким образом решается вопрос обтекания плавающего блока в родительском теге и в целом в разметке?
3. Опишите процедуру деления содержимого на колонки, используя возможности float-блоков.
4. Каким образом можно осуществить разбиение содержимого блока на три равные по ширине колонки, которые при этом будут пропорционально менять ширину при изменении ширины окна браузера?
5. Приведите примеры практического использования float-блоков.

Практикум

Задание 1

1. Создайте проект *Работа с плавающими блоками*.
2. Изучите пример использования float-блоков из пункта 3.6.1.
3. По аналогии реализуйте разметку, где в основном блоке шириной не менее 500px и не более 1000px могут размещаться 5 плавающих блоков шириной 250px каждый.
4. Сколько максимум и минимум блоков могут размещаться в одном слое?

Задание 2

1. Создайте каталог *Программирование*.
2. Согласно описанию практического примера из пункта 3.6.2, пошагово воспроизведите реализацию разметки предложенной страницы.

Задание 3

1. Создайте каталог *Разметка структуры*. Подготовьте типовую разметку и стилевой файл.
2. Требуется подготовить каркас разметки по следующей схеме рис. 3.117 и настроить базовые свойства блоков. Используйте полученные знания по работе свойства `float`. Для оформления блоков `<aside>` каждому задайте свой класс.
3. Чтобы показать корректность разметки, сделайте для каждого блока заливку.

Задание 4

1. Создайте каталог *ИТ в образовании*. Скопируйте предложенную разметку и стили.
2. Дополните разметку в `<main>` и стили свойствами (при необходимости можете добавить свои селекторы, классы к тегам и т.д.), ориентируясь на следующий результат, изображенный на рис. 3.118.
3. При наведении курсора на ссылку меню ее фон сделать светлее, шрифт – полужирным начертанием.

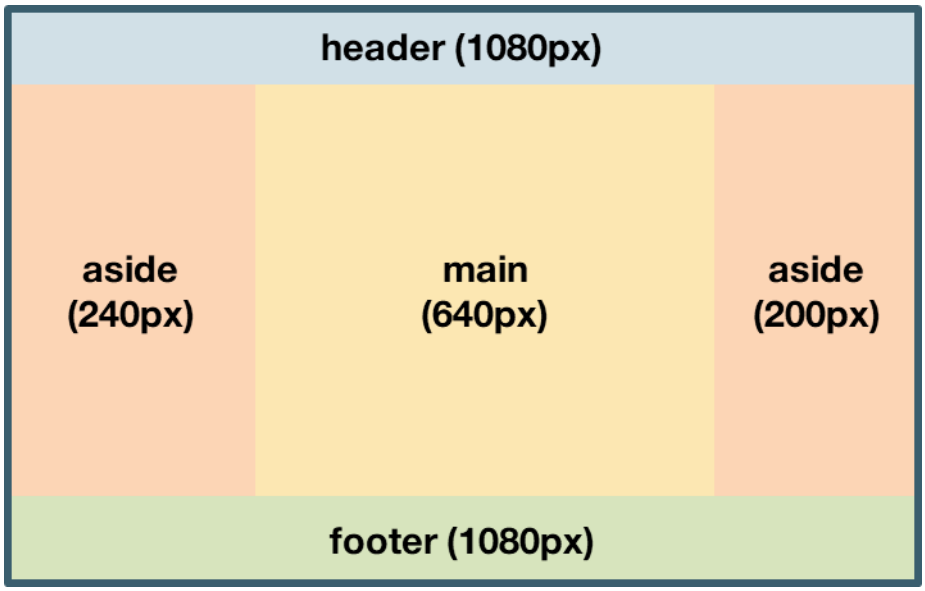
Глава 3\Программирование\index.html

```
<div class="main-box">
    <header class="header">
        <h1 class="header__title">ИТ в образовании</h1>
    </header>

    <aside class="aside__menu">
        <a href="#">ЭОР</a>
        <a href="#">MS Office</a>
        <a href="#">ДО</a>
        <a href="#">Работа с Internet</a>
        <a href="#">Методическая копилка</a>
    </aside>

    <main class="main">
        <!-- Дополнить разметку -->
        Процесс информатизации современного общества
        требует от сферы образования новых подходов и
        технологий в обучении. Современный специалист
        сферы образования должен не только владеть
        основами работы с ПК, но и профессионально
        использовать прикладное ПО.

        На этом ресурсе представленны учебные материалы,
        которые позволяют учителю лучше ориентироваться в
        возможностях современного ПО в области
        образования.
    </main>
</div>
```

Глава 3\Программирование\index.html

```
/* ========= Основные селекторы ========= */
* { box-sizing: border-box; }

body {
    background: #80642C url('images/фон.jpg');
    font: 16px Arial;
}

/* ========= Основной бокс ========= */
.main-box {
    width: 920px;
    background: rgba(251, 231, 179, 0.8);
    margin: 0 auto;
```

```
    overflow: auto;
}

/* ========= Шапка сайта ======== */
.header { /* дописать */ }

/* заголовок */
.header__title {
    margin: 0;
    /* дописать */
}

/* блок меню и контента идут по соседству, поэтому делаем
их плавающими */
.aside__menu, .main { float: left; }

/* ========= Сайдбар ======== */
.aside__menu { width: 260px; }

/* ссылки меню */
.aside__menu a {
    display: block;
    background: #65291B;
    /* дописать */
}

/* срабатывает при наведении курсора на ссылку */
.aside__menu a:hover { /* >>> */ }

/* ========= Контент ======== */
.main {
    width: 660px;
    padding: 20px;
}

/* ========= Разные элементы ======== */
/* блок оформления изображений и изображения внутри этого
   блока */
.puctures {
    display: table;
    margin: 0 auto;
}

.puctures img { /* дописать */ }
```



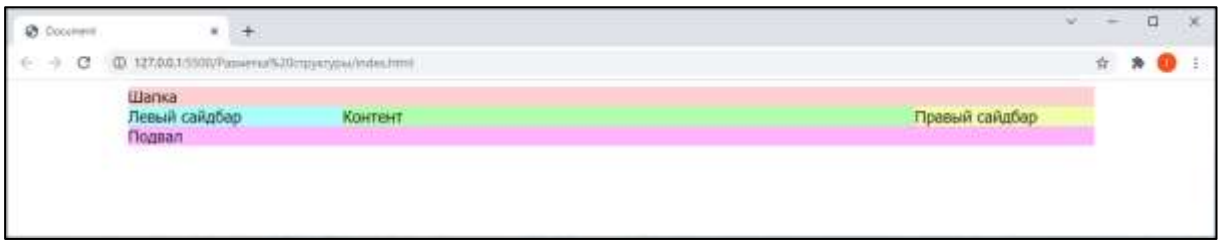


Рис. 3.117. Образец выполнения задания 3

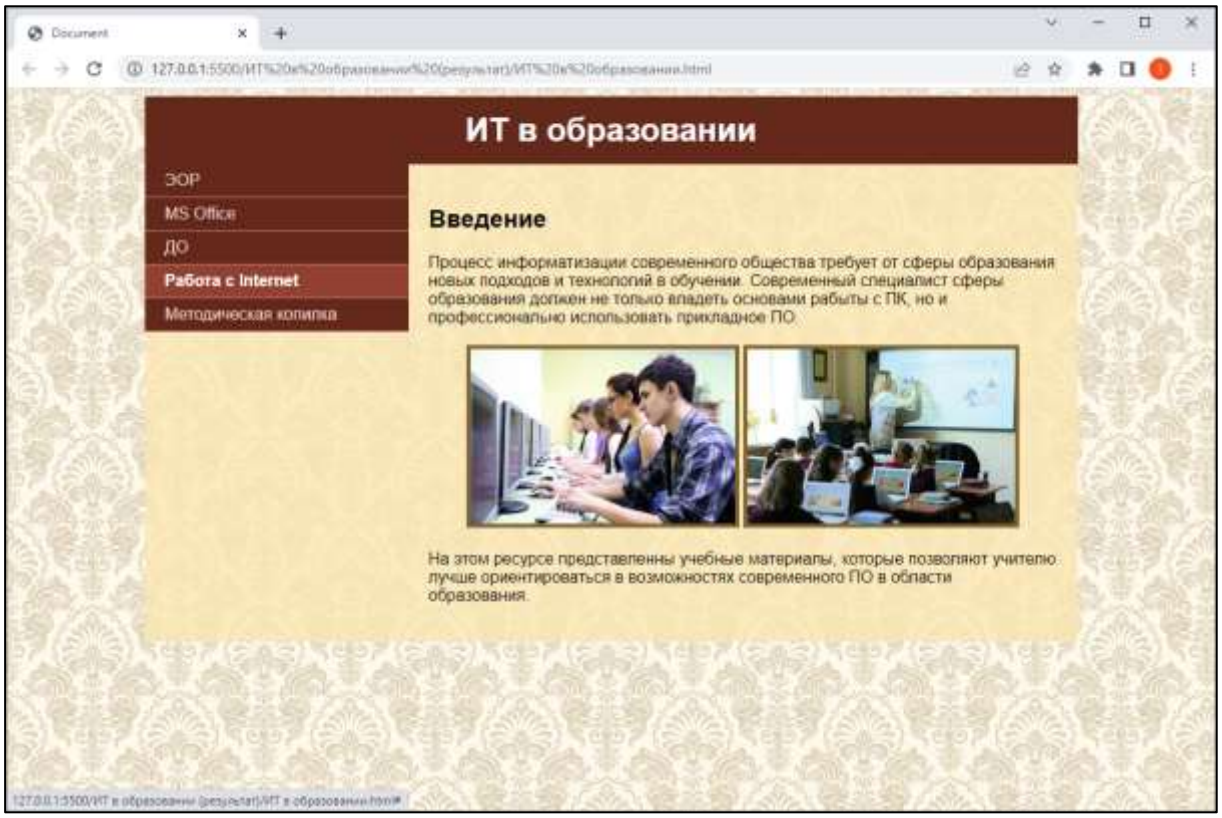


Рис. 3.118. Образец выполнения задания 4

3.7. Позиционирование блоков

3.7.1. Позиционирование

Особенности позиционирования элементов

HTML-документ представляет собой многоуровневую структуру, связывающую родительские и дочерние блоки и элементы. Одной из важнейших задач верстки является расположение элементов на странице требуемым образом. Для этого CSS реализует разные инструменты, которые могут использоваться как в одиночку, так и в комбинации друг с другом.

По умолчанию элементы на веб-странице располагаются в *нормальном* или *базовом потоке*. Это означает, что:

- порядок отображения элементов совпадает с порядком их разметки в HTML-коде;
- при вложении элементов работает система слоев (чем глубже вложен элемент, тем выше его визуальное положение);
- позиция элемента в потоке зависима от свойства `display`.

Однако этим позиционирование элементов в CSS не ограничивается. Дополнительные свойства расположения элемента могут быть настроены с помощью свойств **`float`** и **`position`**.

Позиция элемента в CSS3

Технология CSS3 предлагает схему позиционирования, которая зависит от следующих параметров:

- размера и типа контейнера элемента;
- схемы позиционирования (в нормальном потоке, обтекании или абсолютной позиции),
- соотношениями между элементами в HTML-дереве;
- внешних данных (размеры области просмотра, внутренние размеры изображений и др.).

Позиционирование

Определение

Позиционирование (блока) *– это настройка положения элемента относительно его родителя.*

```
position: absolute | fixed | relative | static | sticky
```

Определяет ***способ позиционирования элемента*** *относительно окна браузера или родительского элемента.*
Свойство обычно сопровождается следующей группой свойств: ***left***, ***right***, ***top***, ***bottom****, задающих отступ от соответствующей границы.*

Свойство `position` позволяет менять поведение элементов в потоке. Иными словами, оно может «вытащить» элемент из нормального потока разметки и разместить в другой позиции относительно окна браузера или родительского блока.

3.7.2. Режимы позиционирования

Пример для анализа

Разберем подробнее каждое из его возможных значений свойства `position` на едином примере и его модификациях.

Зададим блок-родитель `parent` и три подряд идущих блока `block_1`, `block_2`, `block_3`. Внутрь третьего блока вставим еще два: `block_3a` и `block_3b`.

Все блоки имеют одинаковое оформление (для простоты), которое определяет класс `block`. Поэтому подключаем по два класса к каждому блоку (рис. 3.119).

Глава 3\Позиционирование\Исходный\index.html

```
<div class="parent">
    <div class="block block_1">Блок 1</div>
    <div class="block block_2">Блок 2</div>
    <div class="block block_3">
        Блок 3
        <div class="block block_3a">Блок 3A</div>
        <div class="block block_3b">Блок 3B</div>
    </div>
</div>
```

Блоку (классу) `parent` зададим заливку, отступ внутри и поля снаружи.

Класс `block` определяет общие параметры блоков: занимает всю доступную ширину в родителе, немного меняет настройки шрифта и внутренний отступ. Отдельными классами каждому блоку зададим лишь свой цвет.

Глава 3\Позиционирование\Исходный\style.css

```
* { box-sizing: border-box; }

body { font: 16px Tahoma, Arial; }

.parent {
    background: #1b1320;
    padding: 20px;
    margin: 50px;
}

.block {
    width: 100%;
    color: white;
    font: bold 20px Tahoma;
    padding: 25px;
}

.block_1 { background: #49354d; }
.block_2 { background: #5C527F; }
.block_3 { background: #6E85B2; }
.block_3a { background: #9ab8f3; }
.block_3b { background: #b5c9f3; }
```

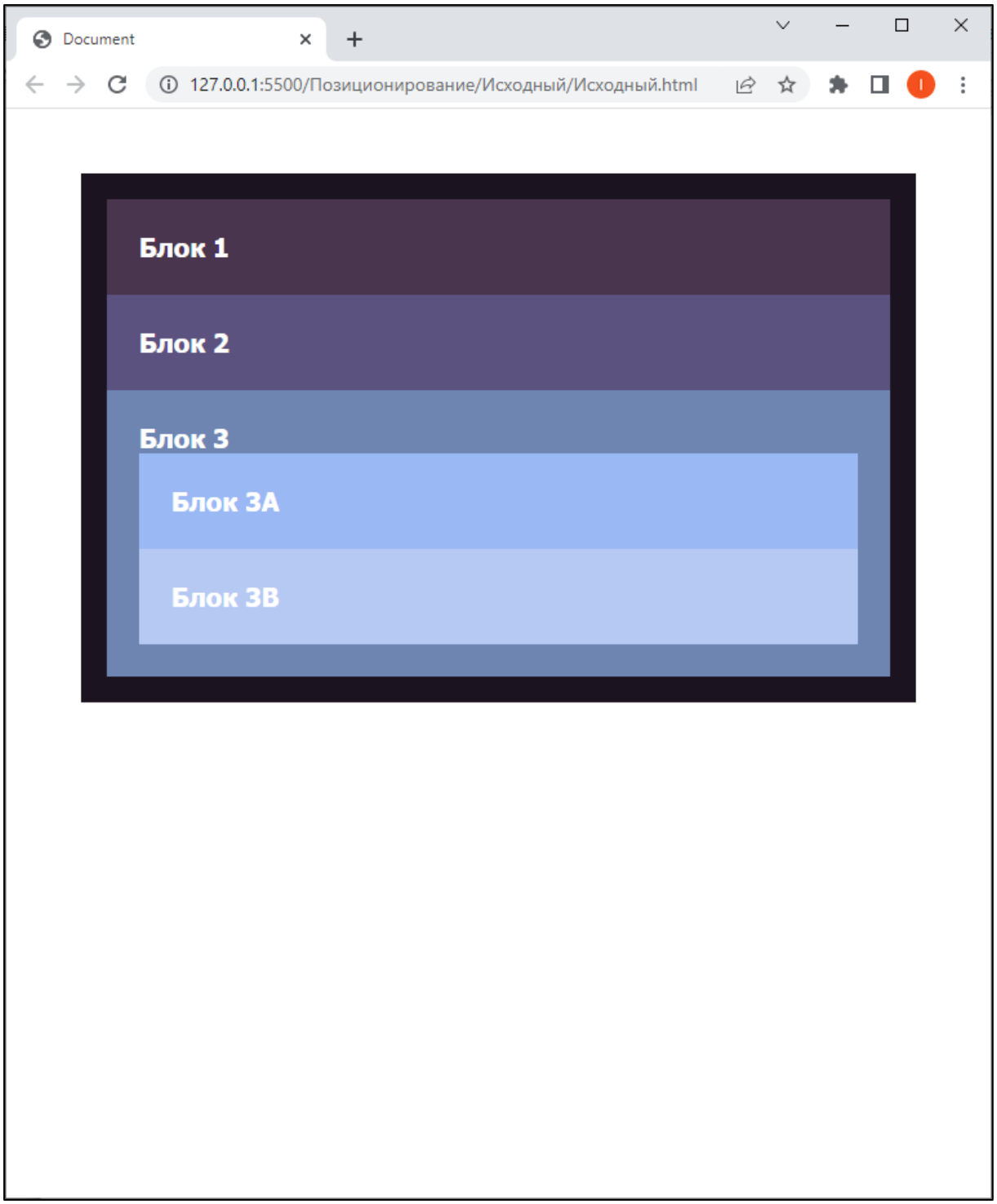


Рис. 3.119. Исходное позиционирование блоков

1. position: static

По умолчанию элементы располагаются *статически*, т.е. в том порядке, который определен HTML-разметкой. В таком случае говорят, что элементы следуют в *нормальном потоке* (как в исходном примере). Поэтому указывать значение **`static`** необязательно.

Единственный случай, когда может потребоваться явно указать значение `static` – это вернуть позицию блока в нормальный поток, если она была изменена, например, в результате наследования свойства позиционирования.

Кроме того, статические блоки не смещаются даже при явно указанных свойствах смещения.

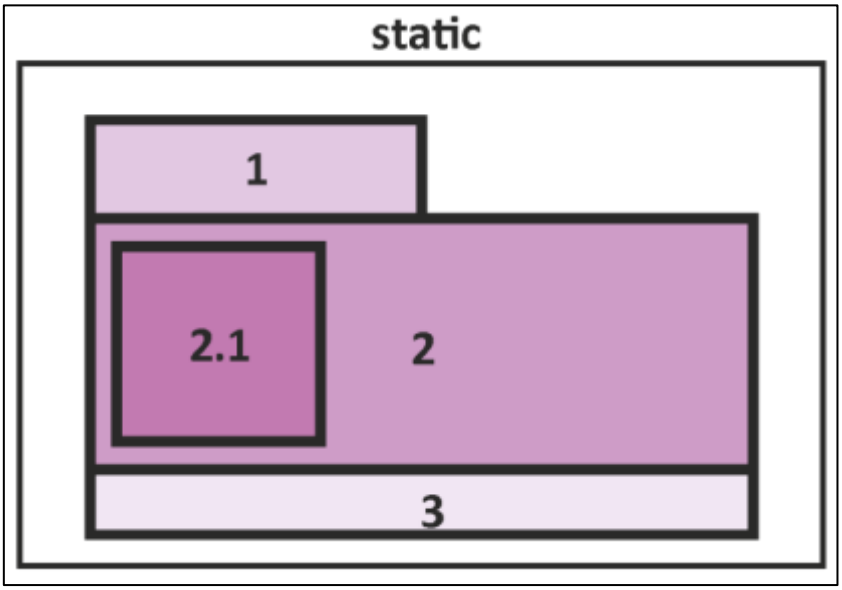


Рис. 3.120. Блоки 1, 2, 3 идут в нормальном потоке

Глава 3\Позиционирование\1. Position Static\style.css

```
...

.block_2 {
    background: #5C527F;
    position: static;
    left: 20px;
}

...
```

(См. рис. 3.126рис. 3.120)

2. position: relative

Относительное позиционирование с помощью свойств **left**, **right**, **top** и **bottom** смещает блок относительно его нормального положения в потоке.

При этом смещение носит визуальный характер. «Физически» блок остается на месте, резервируя свое исходной пространство в нормальном потоке. Другие блоки не занимают освободившееся пространство.

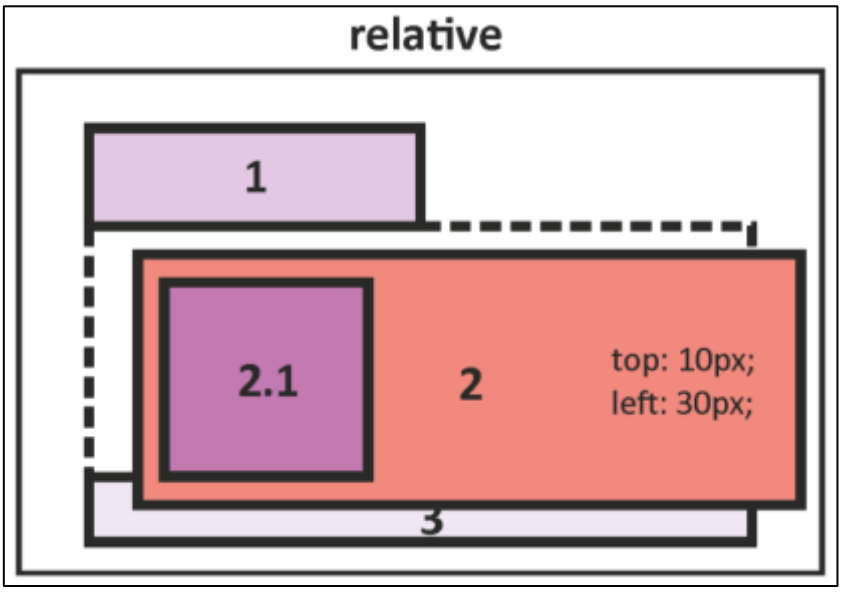


Рис. 3.121. Блок 2 занимает относительное положение. Свойства top и left смещают его положение относительно «нормального»

Глава 3\Позиционирование\2. Position Relative\style.css

```
...

.block_2 {
    background: #5C527F;
    position: relative;
    top: 15px;
    left: 40px;
}

...
```

(См. рис. 3.120рис. 3.127)

3. position: absolute

При *абсолютном* позиционировании элемент «выходит» из нормального потока, а другие блоки могут занять освободившееся пространство.

Самому элементу можно задать определенное положение относительно окна браузера (по умолчанию) или родительского элемента с помощью свойств `left`, `right`, `top` и `bottom`.

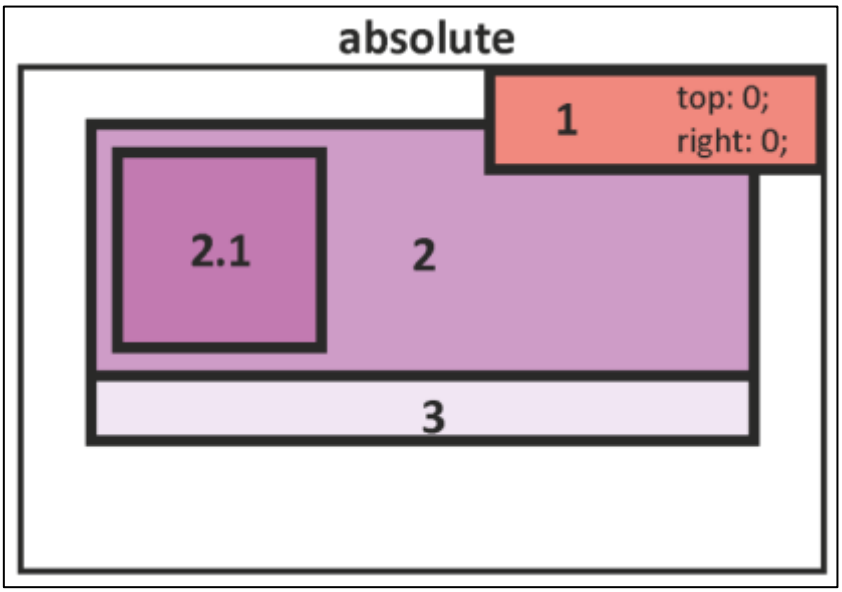


Рис. 3.122. Блок 1 позиционирован абсолютно. Поэтому блоки 2 и 3 поднимаются выше. А сам блок 1 прикреплен к правому верхнему углу

Рассмотрим два случая абсолютного позиционирования.

Глава 3\Позиционирование\3. Position Absolute\style.css

```
...

.block_2 {
    background: #5C527F;
    position: absolute;
    top: 15px;
    left: 130px;
}

...
```

(См. рис. 3.128-рис. 3.129)

Глава 3\Позиционирование\3. Position Absolute\style.css

```
...

.block_2 {
    background: #5C527F;
    position: absolute;
    top: 15px;
    left: 130px;
}

...
```

4. position: fixed

Фиксированное позиционирование аналогично абсолютному, однако элемент всегда зафиксирован в определенной позиции экрана при прокрутке.

Фиксированными, например, часто делают «шапку» и «подвал» сайта, меню быстрого доступа, кнопки возврата и другие элементы, которые должны быть всегда «на виду».

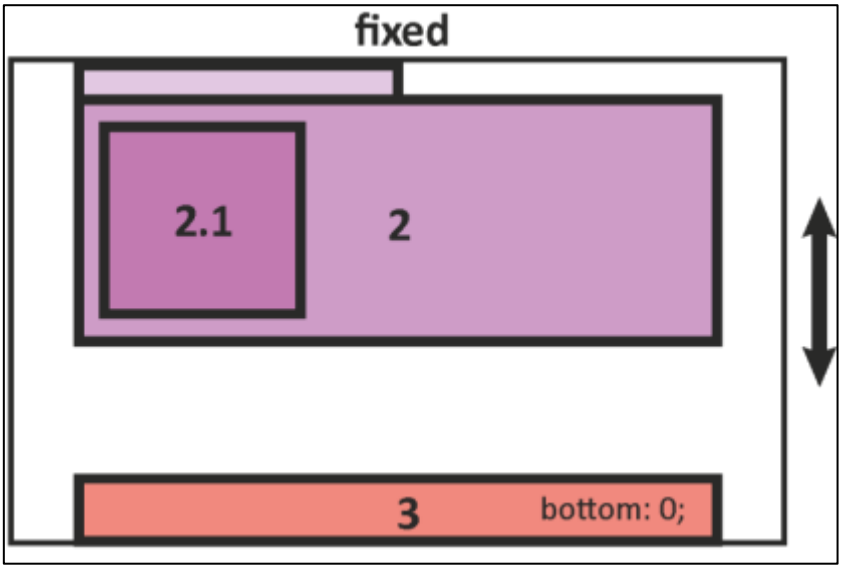


Рис. 3.123. Блок 3 зафиксирован и прижат к нижнему краю родителя. При прокрутке он сохраняет положение относительно экрана

Глава 3\Позиционирование\4. Position Fixed\style.css

```
...

.block_2 {
    background: #5C527F;
    position: fixed;
    bottom: 0;
    left: 0;
    right: 0;
}

...
```

Глава 3\Позиционирование\4. Position Fixed\style.css

```
...

.block_2 {
    background: #5C527F;
    width: 30%;
    position: fixed;
    bottom: 0;
    left: 0;
    right: 0;
}

...
```

(См. рис. 3.130-рис. 3.131)

5. position: sticky

Новое значение, сочетающее в себе значение `static` и `fixed`. В случае прокрутки блок *прилипает* к верхней/нижней границе блока родителя и далее остается на виду. При этом он остается именно внутри родительского блока.

Как только достигнута граница родителя, блок отлипает.

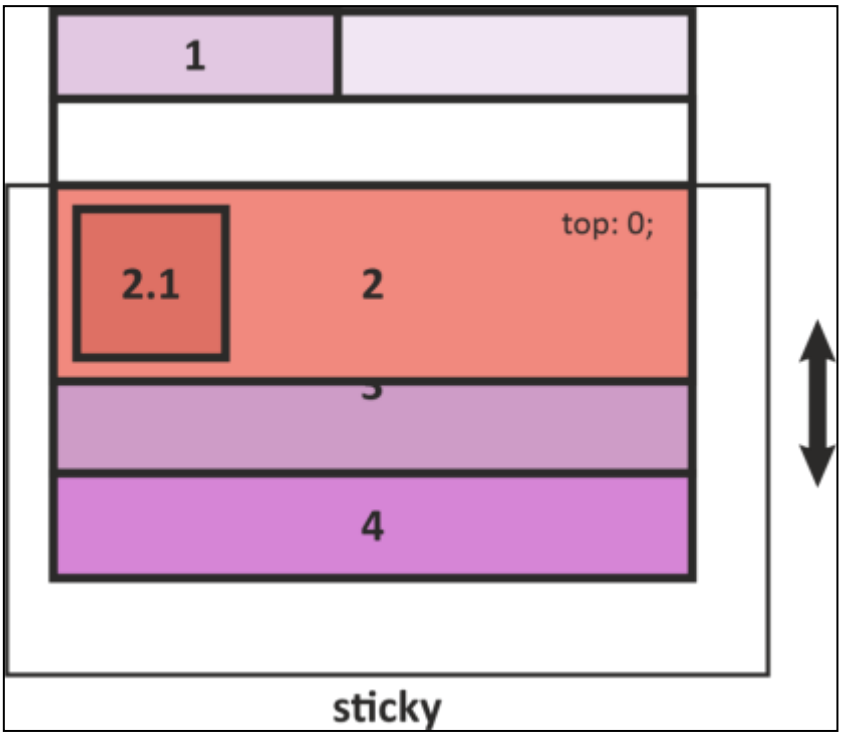


Рис. 3.124. Блоку 2 активировано прилипание к верхней границе. Как только прокрутка достигает его, он прилипает и двигается вместе с прокруткой. По достижению нижней границы блока 4 он прижимается к ней, пока не прокрутят обратно вверх

Глава 3\Позиционирование\5. Position Sticky\style.css

```
...

.block_2 {
    background: #5C527F;
    position: sticky;
    top: 0;
}

...
```

(См. рис. 3.132)

6. position: relative > position: absolute

Если родительский блок позиционирован как относительный, а его дочерний(ие) – абсолютно, то свойства `left`, `right`, `top` и `bottom` дочерних блоков будут сдвигать их уже относительно родителя, а не окна браузера.

Этот прием часто используется на практике.

При этом действие распространяется на ближайший в иерархии родительский блок со значением `relative`.

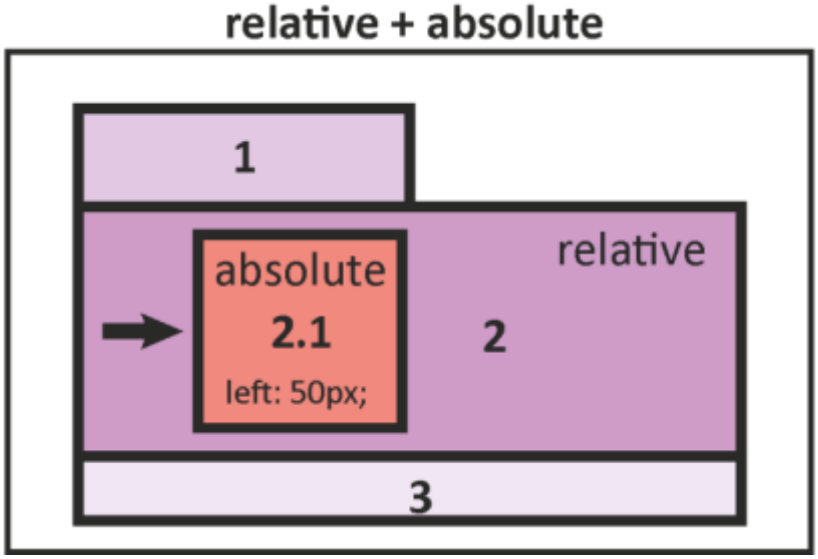


Рис. 3.125. В блоке 2 установлено относительное позиционирование. Его дочерний блок 2.1 позиционируется абсолютно. Свойство `left` смещает его относительно своего родителя

**Глава 3\Позиционирование\
6. Position Relative плюс Absolute\style.css**

```
...

.block_3 {
    background: #6E85B2;
    position: relative;
}

.block_3a {
    background: #9ab8f3;
    position: absolute;
    left: 50px;
    top: 10px;
}

...
```

**Глава 3\Позиционирование\
6. Position Relative плюс Absolute\style.css**

```
...

.block_3 {
```

```
    background: #6E85B2;
    position: relative;
}

.block_3a {
    background: #9ab8f3;
    width: 250px;
    position: absolute;
    left: 0;
}

.block_3b {
    background: #b5c9f3;
    width: 150px;
    position: absolute;
    right: 0;
}
...
```

(См. рис. 3.133-рис. 3.134)

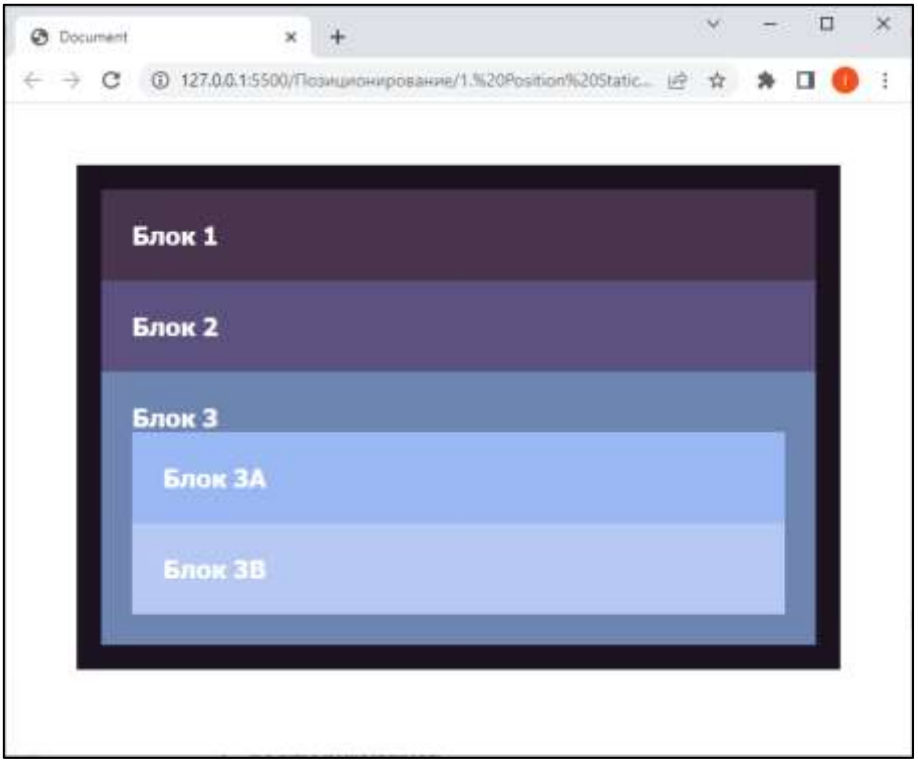


Рис. 3.126. Блок 2 никуда не смещается (смещение вообще не имеет смысл). Все блоки статичны по умолчанию

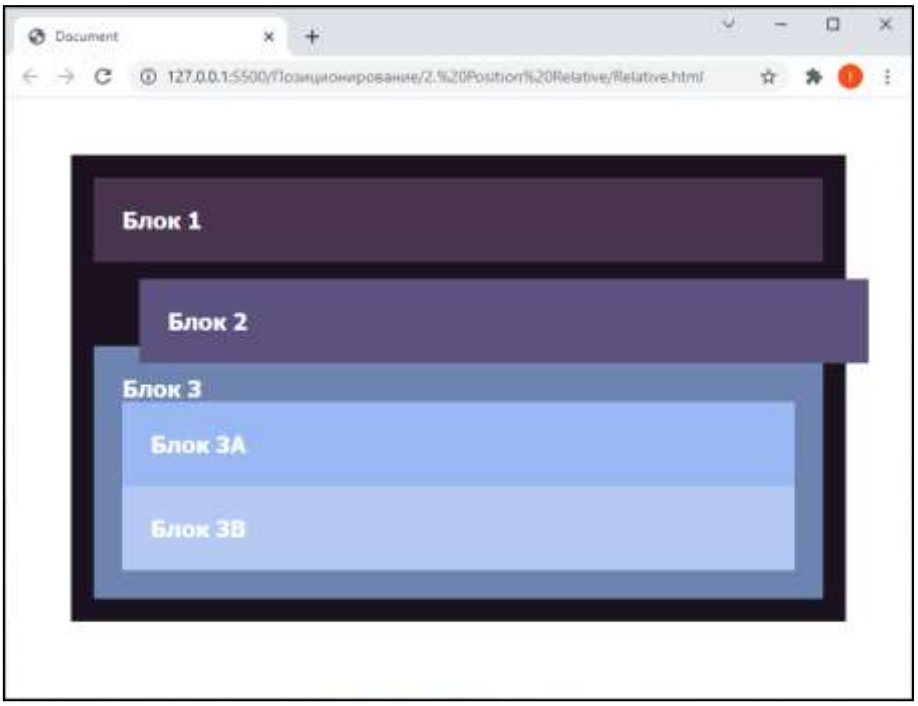


Рис. 3.127. Блок 2 визуально смещается относительно родительского сверху и слева. Однако «физически» он остается в нормальном потоке

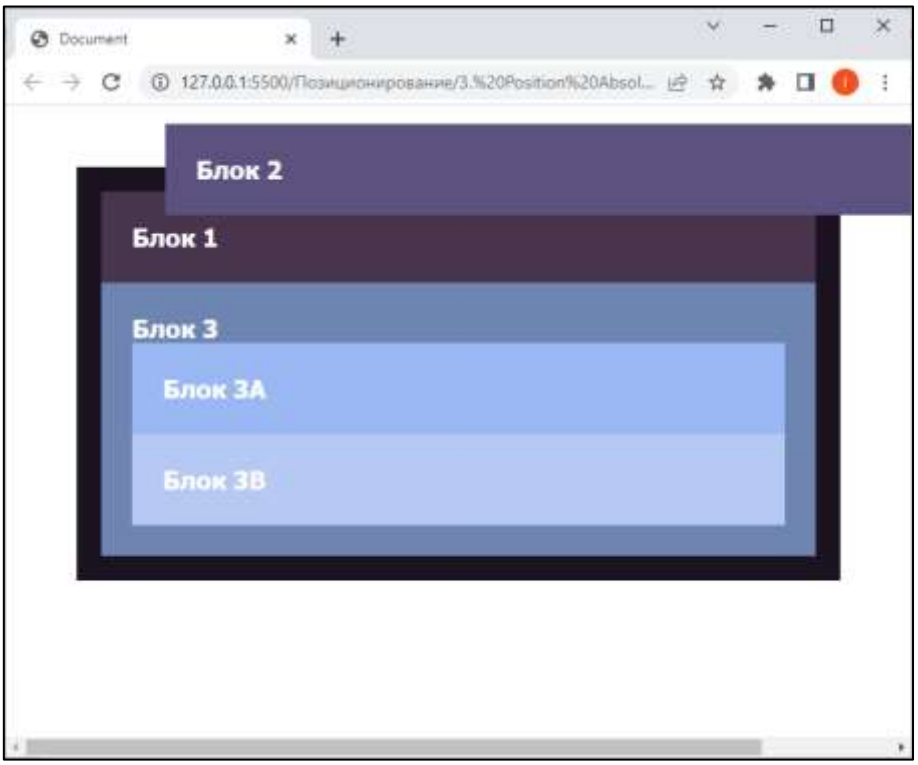


Рис. 3.128. Блок 2 выходит из нормального потока и занимает положение со смещением относительно окна браузера. Другие блоки занимают освободившееся пространство

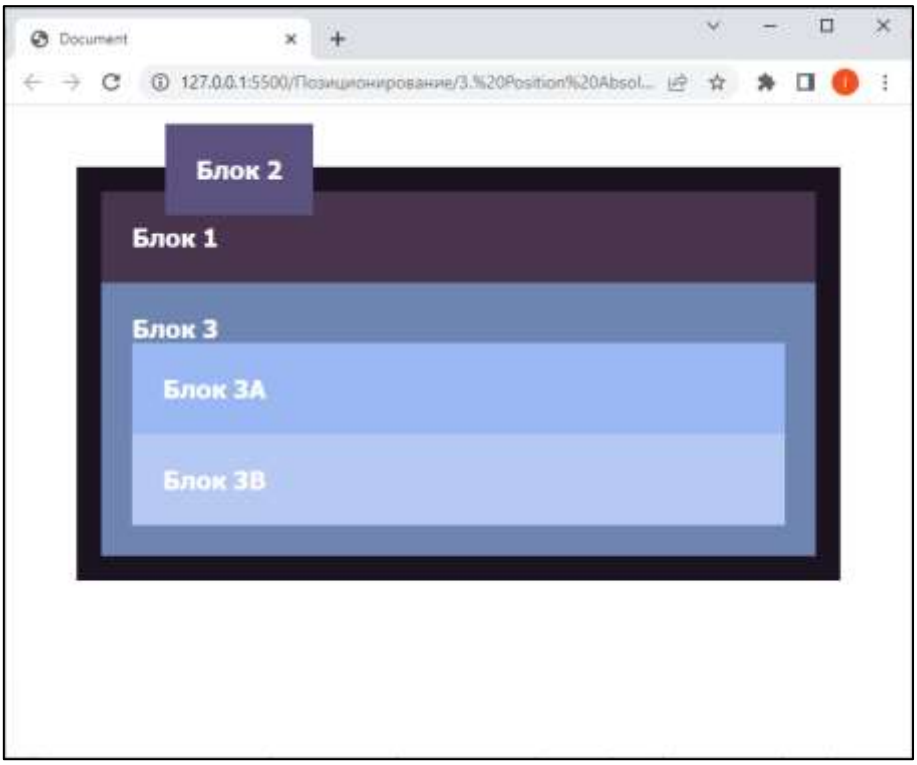


Рис. 3.129. Если при этом убрать/подавить 100% ширину блока, то он отобразится по ширине контента (или по фиксированной ширине)

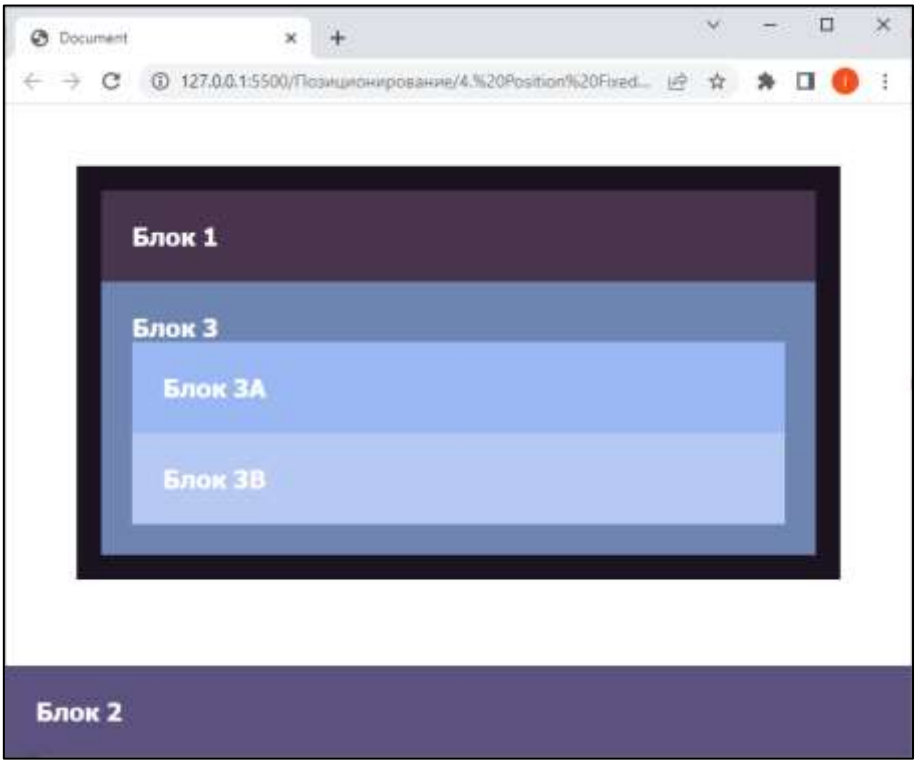


Рис. 3.130. Блок 2 выходит из нормального потока и занимает фиксированное положение: прижимаем его вниз и по краям. При прокрутке блок всегда занимает это положение

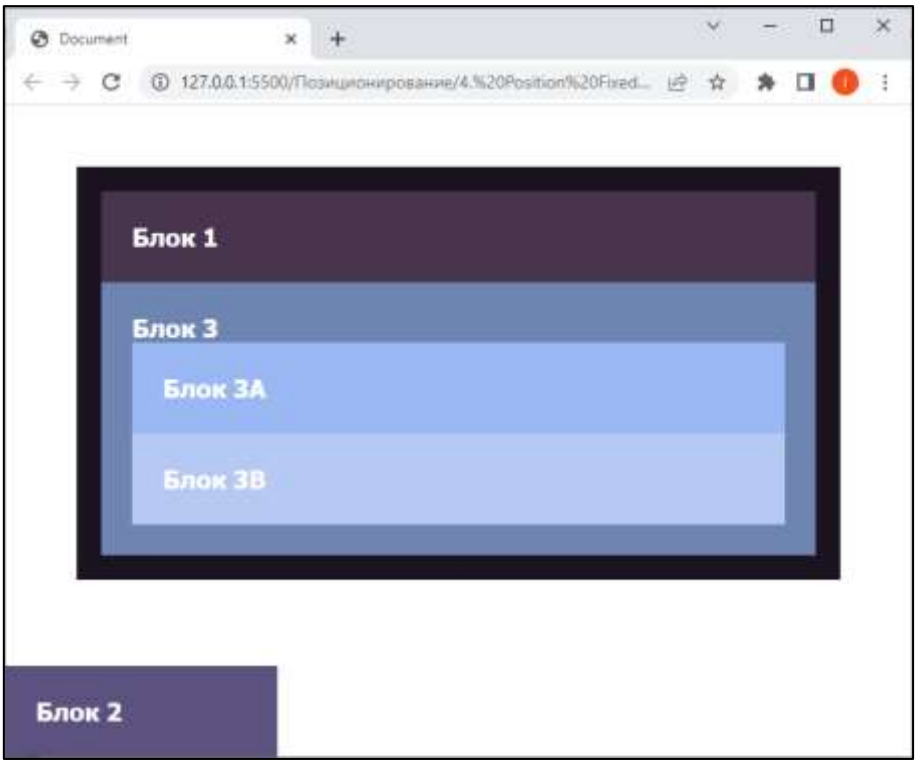


Рис. 3.131. Можно изменить ширину блока: это никак не повлияет на другие блоки из нормального потока

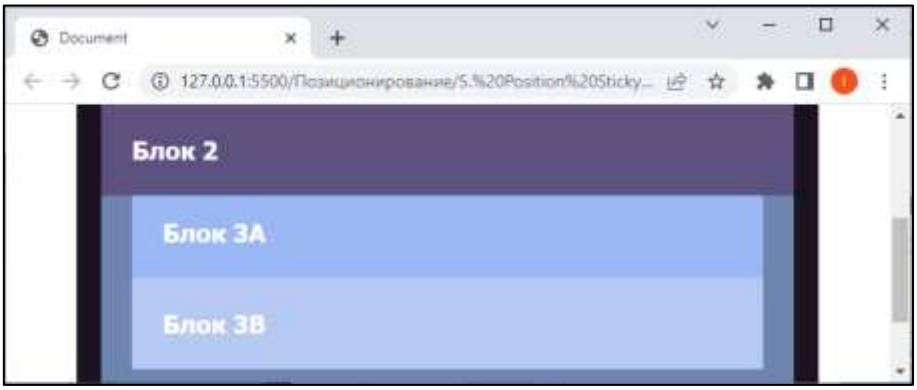


Рис. 3.132. Включено прилипание блока 2 сверху. Блок может двигаться от своей позиции вниз к границе родителя и обратно

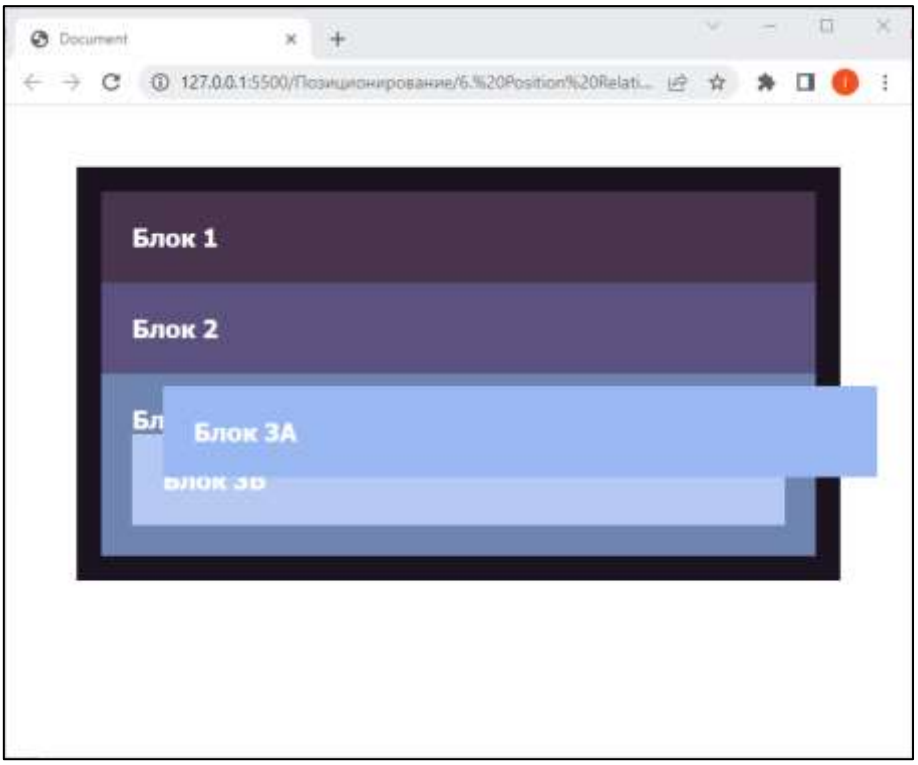


Рис. 3.133. Блок 3А занимает абсолютное положение уже относительно своего родителя Блока 3, а не окна браузера

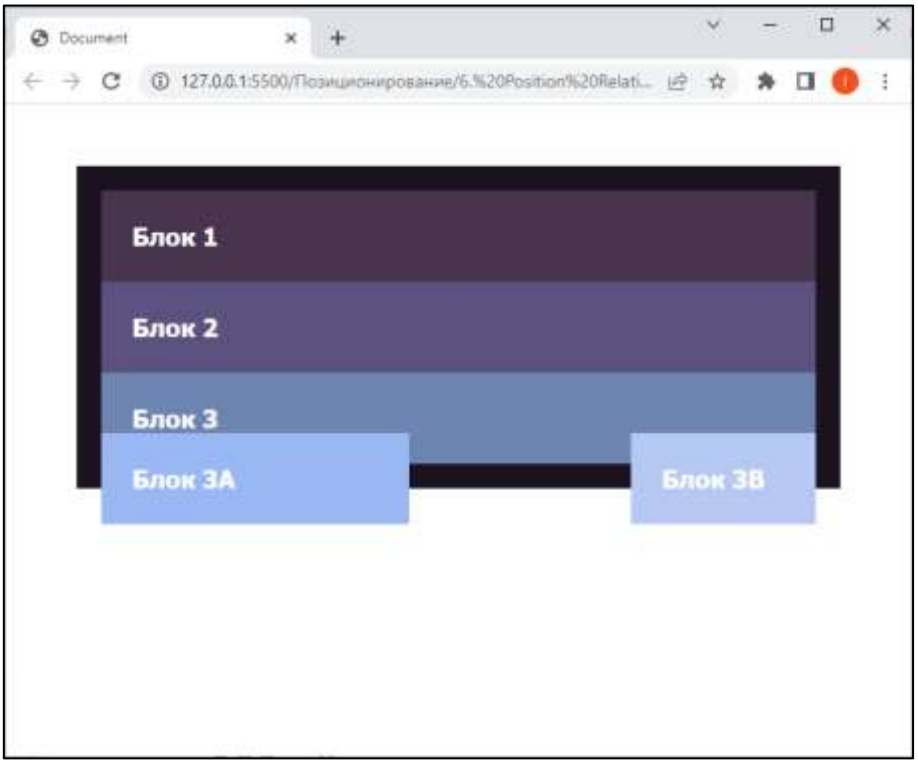


Рис. 3.134. Блоки 3А и 3В прижимаются по разным сторонам родителя (не забывайте, что позиционированные абсолютно элементы выходят из нормального потока)

Порядок слоев

Определение

`z-index: число`

*Определяет **порядок наложения слоя**. Чем больше число, тем выше элемент.*
Обычно используется в том случае, если блоки перекрывают друг друга и необходимо поменять порядок перекрытия.
*Использовать **z-index** имеет смысл, если для блоков активно свойство **position**.*

Глава 3\Слои\index.html

```
<div class="definition">
    <p>
        Свойство <strong>z-index</strong> определяет
        положение элемента и нижестоящих элементов по
        оси z. В случае перекрытия элементов, это
        значение определяет порядок наложения.
    </p>
    <div class="label">
        z-index = -2
    </div>
</div>
```

Глава 3\Слои\style.css

```
body {
    font: 18px Tahoma;
}

.definition {
    position: relative;
    background: #fff7ca80;
    margin: 17px 0;
    padding: 5px 20px;
    border: 2px solid rgba(255, 227, 149, 0.500);
}
```

```
.label {
    position: absolute;
    right: 10px;
    bottom: -18px;
    background: #e05119;
    color: white;
    padding: 7px 12px;
    border-radius: 14px;
    transform: rotate(-4deg);
    box-shadow: 3px 4px 0px #000000;
    z-index: 2;
}
```

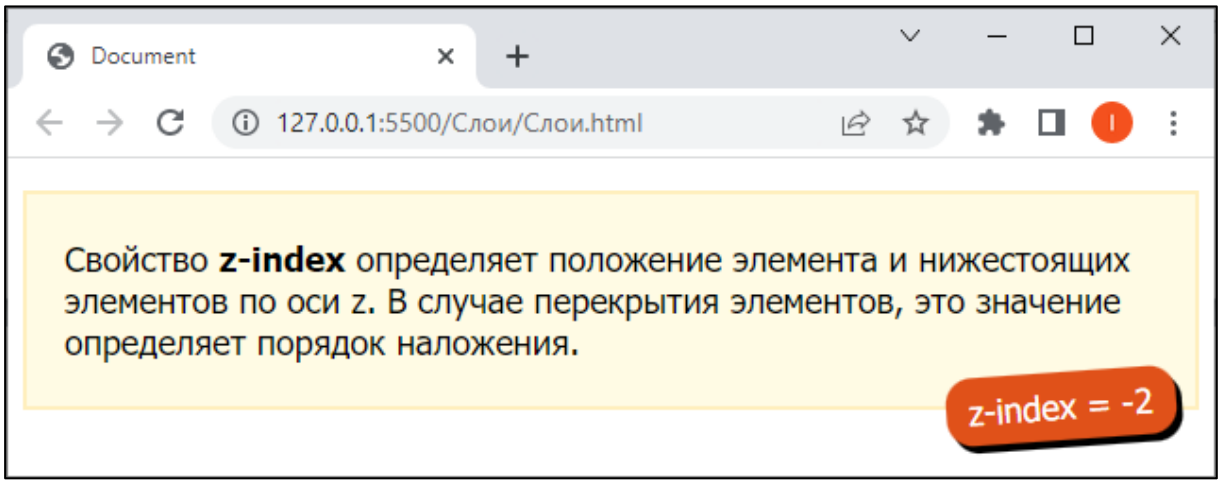


Рис. 3.135. Слой с меткой расположен выше блока (`z-index = 2`)

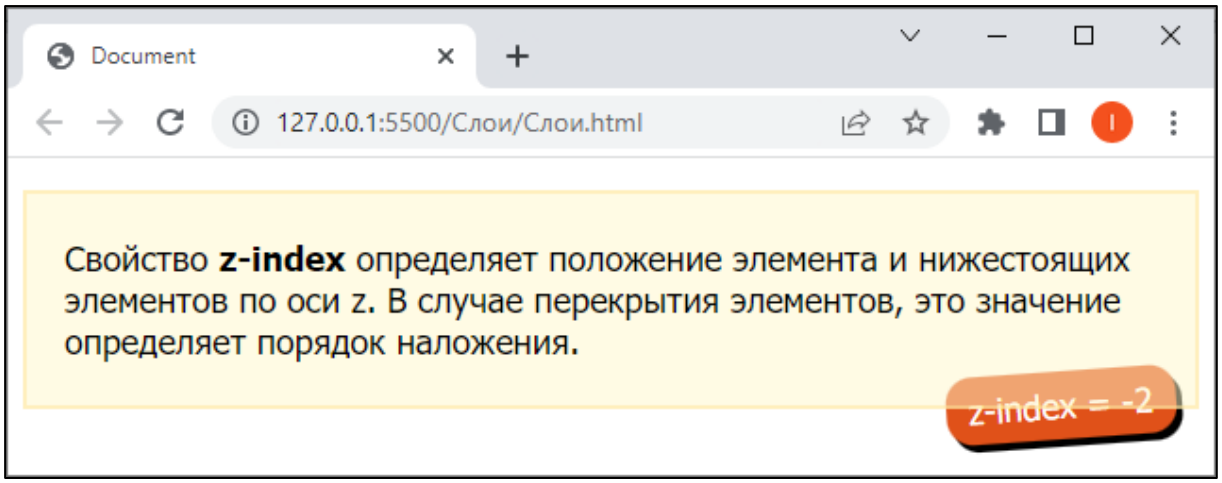


Рис. 3.136. Слой с меткой расположен выше блока (`z-index = -2`)

Это полезно знать!

На самом деле среди всех перечисленных типов позиционирования нет универсального. Каждый подход имеет свои достоинства и недостатки, возможности и ограничения. Поэтому при верстке страницы обычно комбинируются float-блоки, разные подходы к позиционированию.
Более того, в современном CSS используются новые модели управления положением блоков: ***Grid*** *и* ***Flex****. Но этот материал выходит за рамки нашего курса.*
Работа с блоками – это поэтапный процесс, в котором сначала с помощью блоков выстраивается каркас макета и далее идет детализация оформления каждого блока.

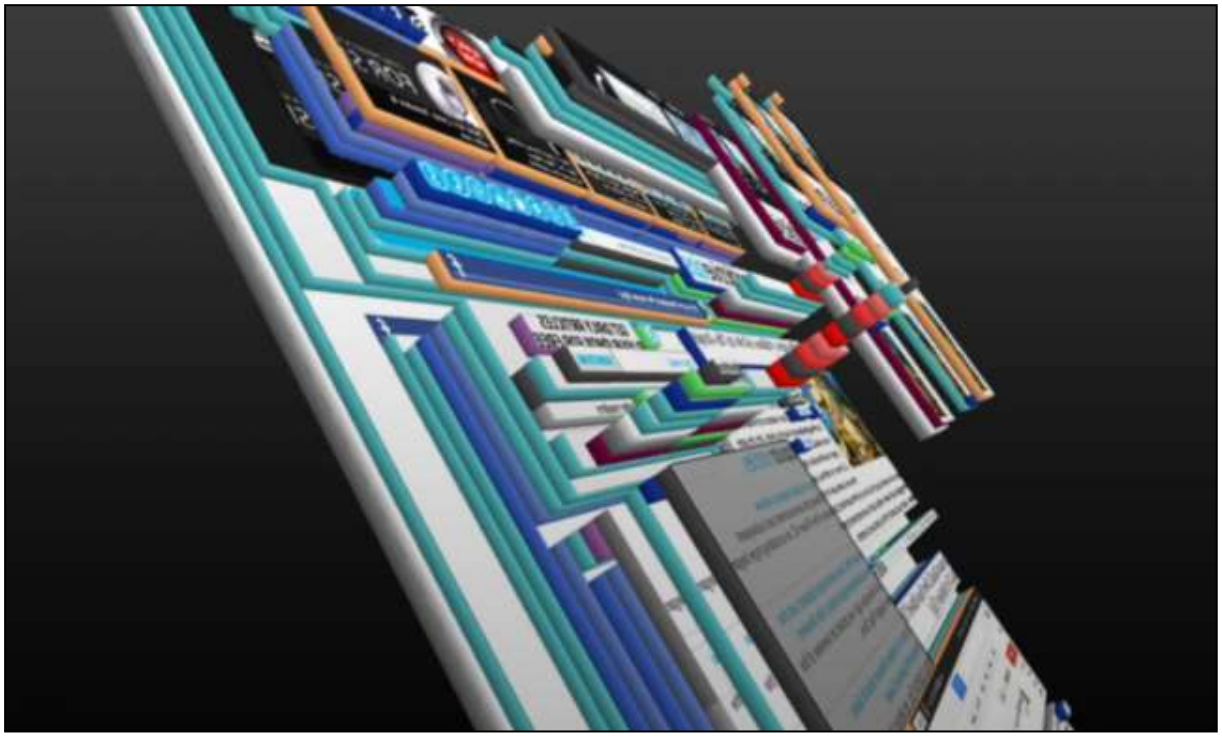


Рис. 3.137. Блочную верстку можно представить в форме сложной системы вложенных коробок. Каждая коробка – контейнер, внутри которого расположены другие контейнеры или элементы

3.7.3. Некоторые полезные приемы

Выравнивание содержимого блока по центру

1. Выравнивание одного блока с контентом

В CSS для выравнивания содержимого блока по центру часто используется настройка внешних полей блока, т.е. свойства `margin`. Для этого левое и правое поле должны иметь одинаковое значение, задающееся константой `auto`. Выравнивание происходит в том случае, если ширина этого блока меньше ширины окна браузера (или блока-родителя).

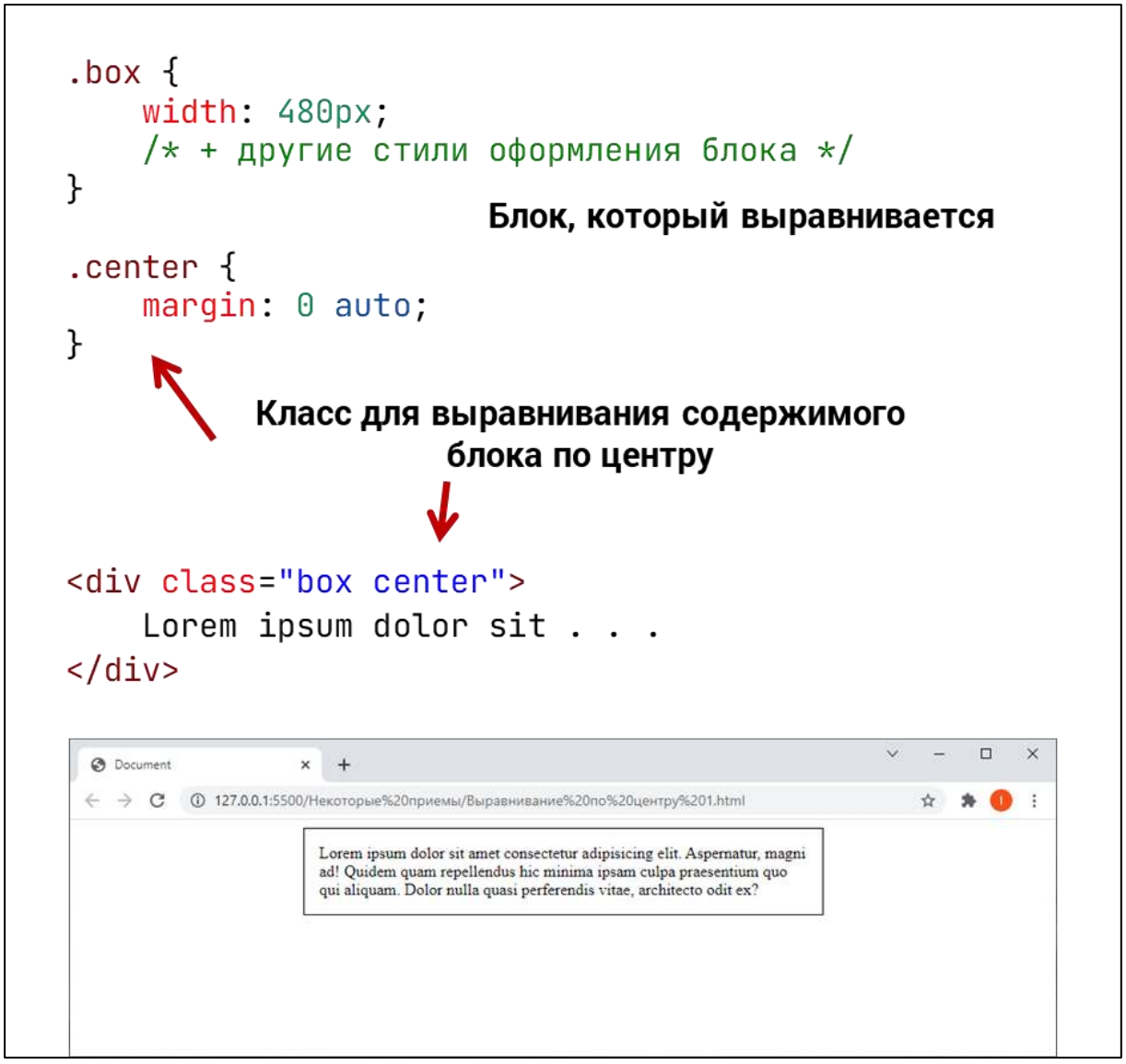


Рис. 3.138. Выравнивание по центру одного блока

2. *Выравнивание строчно-блочных элементов*

Однако подобный прием не сработает, если требуется выровнять по центру два или более блока, для которых установлено свойство

```
display: inline-block;
```

В этом случае блоку нужно задать отображение как таблице:

```
display: table;
```

Далее используется уже рассмотренный ранее прием выравнивания по центру.

```
.t-center {
    display: table;
    margin: 0 auto;
}


<div class="t-center">
    <a href="#" class="button">Кнопочка</a>
    <a href="#" class="button">Кнооооооопочка</a>
</div>
```

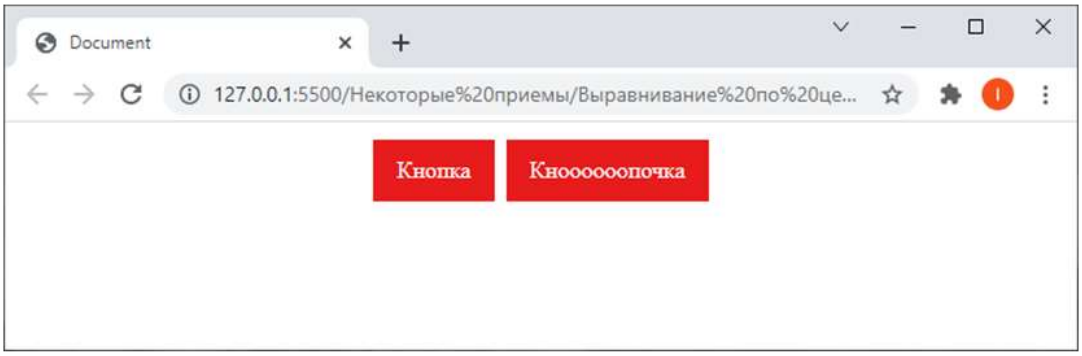


Рис. 3.139. Выравнивание по центру группы строчно-блочных элементов

3. Комплексное выравнивание

Наконец, если выравнивается сам блок с некоторым содержимым по центру, а затем содержимое его дочернего блока, то необходимо подключить классы к каждому из них.

```
.box {
    width: 480px;
    /* + другие стили оформления блока */
}

.center {
    margin: 0 auto;
}

.t-center {
    display: table;
    margin: 0 auto;
}


<div class="box center">
  <div class="t-center">
    <a href="#" class="button">Кнопочка</a>
    <a href="#" class="button">Кнооооооопочка</a>
  </div>
</div>
```

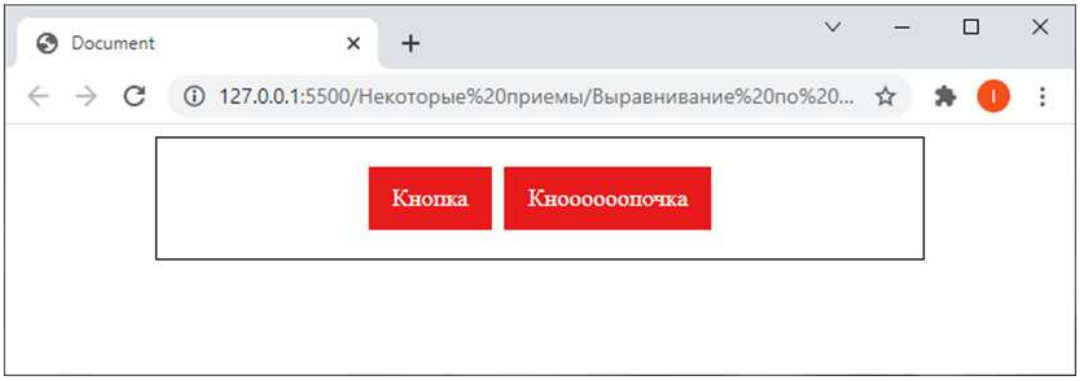


Рис. 3.140. Выравнивание по центру родительского блока и содержимого внутри него

Однако можно ограничиться и одним более универсальным классом.

```
.box {
    width: 480px;
    /* + другие стили оформления блока */
}

.center {
    display: table;
    margin: 0 auto;
}
```

Используем один и тот же класс, как более универсальный

```
<div class="box center">
  <div class="center">
    <a href="#" class="button">Кнопочка</a>
    <a href="#" class="button">Кнооооооопочка</a>
  </div>
</div>
```

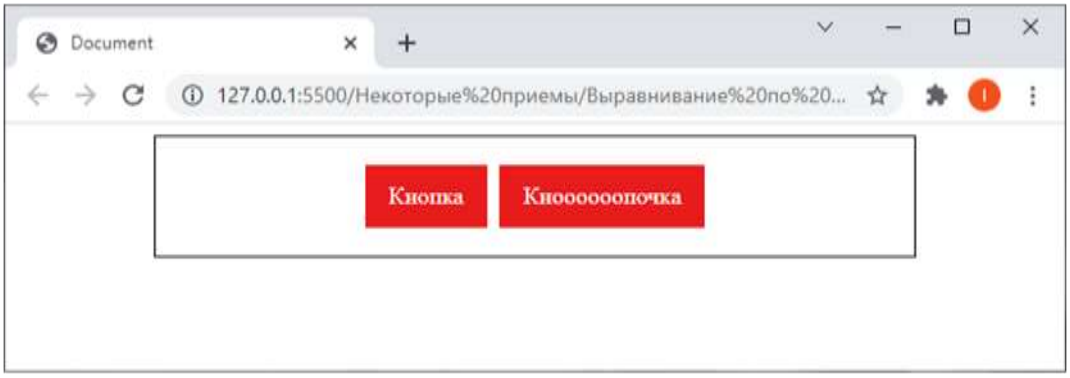


Рис. 3.141. Выравнивание по центру родительского блока и содержимого внутри него (оптимизация кода)

Создание колонок

Колонки можно организовать с помощью float-блоков. Если требуется разбить некоторую область на две равные по ширине колонки, то достаточно описать один класс и далее подключать его по мере необходимости.

```
* { box-sizing: border-box; }

.content, .column { overflow: hidden; }

.content {
    width: 720px;
    margin: 0 auto;
}

.column {
    float: left;
    width: 360px;
    padding: 10px 15px;
}

<div class="content">
    <div class="column">
        Lorem ipsum dolor sit amet ...
    </div>
    <div class="column">
        Lorem ipsum dolor sit amet ...
    </div>
</div>
```

Блок контента

Класс, оформляющий колонку

720 = 360 + 360

Рис. 3.142. Разбиение на две равные колонки

Если требуются колонки разной ширины, либо более двух в ряд, рационально создать отдельные классы на каждую колонку.

При этом ширину колонок удобно задавать в процентах (предполагается, что рядом можно поместить две или более колонок, так, чтобы их суммарная ширина была равна 100%).

Ширина в процентах удобна еще тем, что пропорции ширины колонок сохраняются, если вы реализуете «резиновый» макет для основного блока.

```
* { box-sizing: border-box; }

.row, .column-2, .column-3 { overflow: hidden; }

.row {
    min-width: 480px;
    max-width: 960px;
    margin: 0 auto;
}

.column-2, .column-3 {
    float: left;
    padding: 10px 15px;
}

.column-2 { width: 50%; }

.column-3 { width: 33.33% }
```

«Резиновый» блок: его ширина может меняться в указанных пределах

Классы для оформления колонок

Для двух равных

Для трех равных

```
<main>
   <div class="row">
      <div class="column-2">
         Lorem ipsum dolor ...
      </div>
      <div class="column-2">
         Lorem ipsum dolor ...
      </div>
   </div>

   <div class="row">
      <div class="column-3">
         Lorem ipsum dolor ...
      </div>
      <div class="column-3">
         Lorem ipsum dolor ...
      </div>
      <div class="column-3">
         Lorem ipsum dolor ...
      </div>
   </div>
</main>
```

Рис. 3.143. Разбиение на 3 и более колонок

Рис. 3.144. Основной блок развернут на максимальную ширину

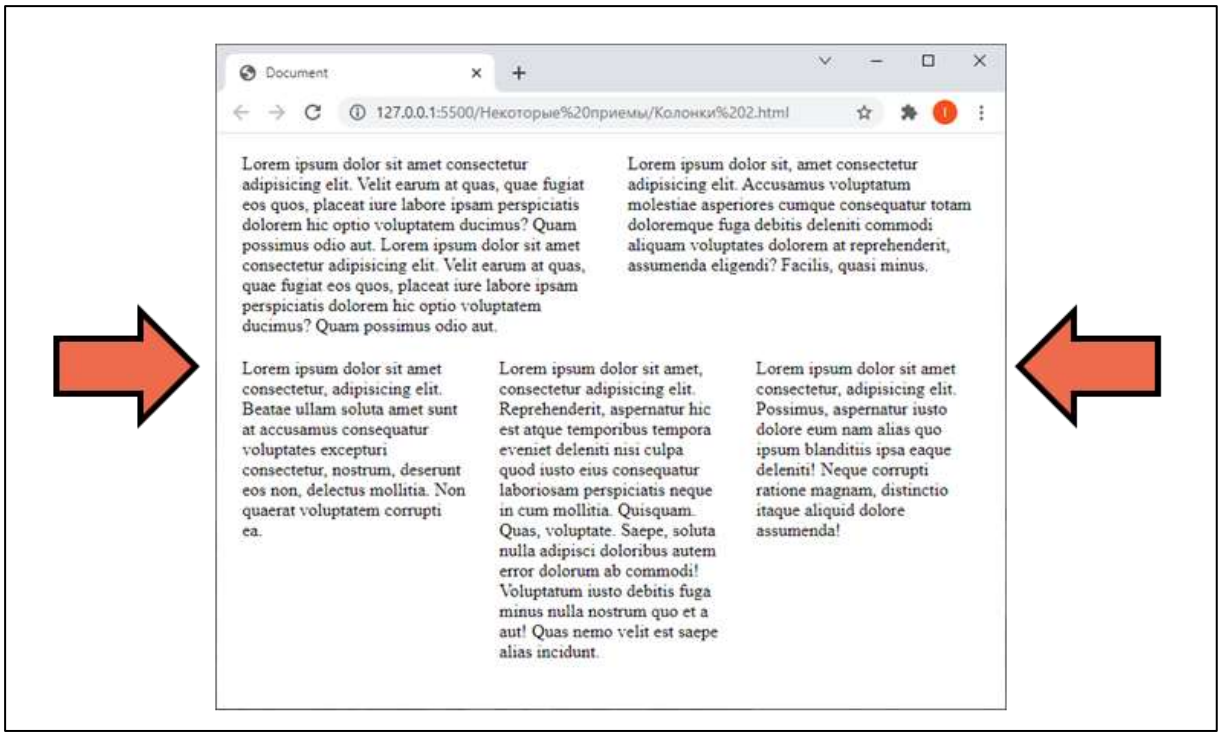


Рис. 3.145. После сужения окна браузера ширина основного блока и вложенных колонок меняется, но сохраняет пропорции относительной ширины

Вопросы для самопроверки

1. Какие критерии влияют на позицию элемента?
2. Что представляет собой нормальный поток следования элементов?
3. В чем состоит особенность относительного позиционирования элементов?
4. Опишите принципы абсолютного позиционирования блоков и их поведение.
5. В чем отличие фиксированного позиционирования от абсолютного? Что общего?
6. Для каких ситуаций рационально использовать sticky-позиционирование блоков?
7. Каким образом сместить позицию блока относительно его родительского контейнера?
8. Опишите процедуру управления порядком наложения слоев при позиционировании элементов.
9. Каким образом разбить разметку на несколько колонок?
10. Как выровнять по центру группу из нескольких блоков?

Практикум

Задание 1

1. Создайте проект *Позиционирование*.
2. Воспроизведите реализацию исходной разметки блоков и каждого варианта позиционирования в отдельном подкаталоге. Структура проекта изображена на рис. 3.146.
3. Еще раз внимательно изучите код, особенности разметки и стилизации.
4. Произвольно протестируйте работу со свойствами размеров блоков, свойствами `top`, `bottom`, `left` и `right`.

Рис. 3.146. Структура проекта для задания 1

Задание 2

1. Создайте проект *Видеомонтаж* с типовой начальной разметкой, скопируйте приведенную ниже разметку и стили.
2. Разберите структуру HTML-кода и CSS-стилей. Чтобы было проще ориентироваться, можете использовать инспектор кода в браузере. В работе показаны некоторые интересные приемы оформления внешнего вида и анимации объектов.
3. Также в примере описаны селекторы атрибутов тега.
4. Используя свойство `position`, сделайте блок с меню (ссылками) зафиксированным сверху при прокрутке.
5. В подвале сайта логотип и название выровняйте, используя абсолютное позиционирование в блоке вместо `float` (рис. 3.147-рис. 3.149).
6. Подсказка: для этого может потребоваться обернуть блок ссылки промежуточным `<div>`-блоком, которому задается свойство `position: relative` (рис. 3.150).

Глава 3\Видеомонтаж\index.html

```
<header>
    <div class="header__menu_bar">
        <div class="container">
            <nav class="header__menu">
                <a href="#">Главная</a>
                <a href="#">Уроки по монтажу</a>
                <a href="#">Практика</a>
                <a href="#">Видеоредакторы</a>
                <a href="#">О нас</a>
            </nav>
        </div>
    </div>
    <div class="container">
        <img src="images/Logo.png" alt="логотип"
             class="header__logo">
        <h1 class="header__title">ВидеоМонтажник</h1>
    </div>
</header>

<main>
    <div class="container">
        <div class="content-padd">
            <h1>Сайт о монтаже</h1>
            <h2>Что такое видеомонтаж?</h2>
            <p><strong>Монтаж</strong> - это процесс
            соединения отдельных фрагментов видео или
            аудио в единую композицию.</p>
            <p>Говоря о монтаже, мы чаще всего
            подразумеваем создание художественных
            фильмов, новостных сюжетов, рекламы и т.п. Но
            в современном мире видеомонтаж используется
            для создания различного видео контента.
            Видеомонтаж становится доступным широкому
            кругу пользователей, не обладающих
            специальными знаниями, но желающими создавать
            свой контент: для семьи, друзей, бизнеса.</p>
            <p>Тем не менее, если вы хотите научиться
            создавать отличные видеоролики, интересные
            пользователям, вам просто необходимо изучить
            разные тонкости видеомонтажа. Ведь главная
            задача режиссёра монтажа – создать такой
            видеоматериал, который бы максимально точно
            отражал суть сюжета, его идею, настроение,
```

```
эмоции и переживания персонажей</p>
<img src="images/Banner.jpg"
     class="small-image">

<h3>Линейный монтаж</h3>
<p>Зачастую такой монтаж осуществляется в
реальном времени. Видеопоток поступает либо
на записывающее устройство, либо на источник
прямой трансляции. При этом съёмка
параллельно ведётся с нескольких камер или
проигрывателей. Выбором источника
видеокартинки занимается режиссёр линейного
монтажа. Он смотрит, какой материал нужно
подать в эфир в конкретный момент времени.
Перед видеотрансляцией заранее готовятся
различные видеопереходы и эффекты, которые
запускаются в строго запланированном порядке.
Те, кто знаком с ведением прямых трансляций
на Youtube и на других ресурсах, используют
именно линейный монтаж. Помимо этого,
линейным монтажом также называют процесс
удаления материала, не нарушающего его
последовательность.</p>
<img src="images/LinearVE.jpg"
     alt="линейный монтаж"
     class="mid-image">

<h3>Нелинейный монтаж</h3>
<p>В отличие от линейного никогда не
совершается в реальном времени. Это всегда
тщательная и скрупулёзная работа с редактурой
отснятого видео. Записанное видео переносится
на компьютер, и уже на нём при помощи
специальных программ для видеомонтажа
объединяется в одно единое целое. Огромное
множество кадров может быть вырезано, а сам
фильм вполне может начинаться и с того
момента, который был отснят в самую последнюю
очередь. Большой плюс нелинейного монтажа —
это отсутствие потери качества при
многократном перезаписывании и перемещении
готового видео с одного источника на
другой.</p>
<p>В настоящее время нелинейный монтаж стал
наиболее популярным, почти полностью вытеснив
```

```
            собой монтаж линейный.</p>
            <img src="images/NonelinearVE.jpg"
                 alt="нелинейный монтаж"
                 class="mid-image">
        </div>
    </div>
</main>

<footer>
    <div class="container">
        <div class="content-padd">
            <a href="index.html">
                <img src="images/Logo.png"
                     alt="лого"
                     class="footer__logo">
                <div class="author">
                    ВидеоМонтажник, 2021
                </div>
            </a>
        </div>
    </div>
</footer>
```

Глава 3\Видеомонтаж\style.css

```
/* --- Основное --- */
* { box-sizing: border-box; }

body {
    font: 18px Roboto, Tahoma, Arial;
    color: #312e2e;
    margin: 0;
    background: #dbdbdb;
}

h1, h2, h3 { font-family: Tahoma, Arial; }

h2, h3 { color: #097c6d; }

p { line-height: 1.5; }

a { color: #d31b1b; }

strong { color: #49487a; }

/* --- Контейнеры --- */
```

```css
.container {
    max-width: 960px;
    margin: 0 auto;
}

.content-padd {
    padding: 8px 20px;
}

/* --- Шапка --- */
header {
    background: url(images/Banner.jpg) no-repeat;
    background-size: cover;
    min-height: 280px;
}

.header__menu_bar {
    background: rgba(0, 0, 0, 0.3);
}

.header__menu > a {
    display: inline-block;
    text-decoration: none;
    color: #fff;
    font-weight: bold;
    padding: 8px 14px;
    text-shadow: 0 0 4px #000000;
}

.header__menu > a:hover {
    background: #d31b1b;
}

.header__logo {
    display: block;
    width: 100px;
    margin: 0 auto;
    padding: 25px 0 10px;
}

.header__title {
    color: #fff;
    font: bold 220% Tahoma;
    text-align: center;
    text-shadow: 0 0 18px #000000;
```

```
    margin: 0;
}

/* --- Контент --- */
main {
    background: white;
    padding: 22px 0;
}

/* --- Подвал --- */
footer {
    padding: 20px 0;
    overflow: hidden;
}

.footer__logo {
    float: left;
    width: 30px;
    margin-right: 10px;
}

.author {
    float: left;
    font-size: 90%;
    font-weight: bold;
}

/* --- Другие элементы --- */
.big-image, .mid-image, .small-image {
    display: block;
    border: 1px solid #131314;
    margin: 0 auto;
}

.big-image { width: 100%; }
.mid-image { width: 60%; }
.small-image { width: 40%; }

img[class="mid-image"]:hover,
img[class="small-image"]:hover {
    transform: scale(1.05);
}
```

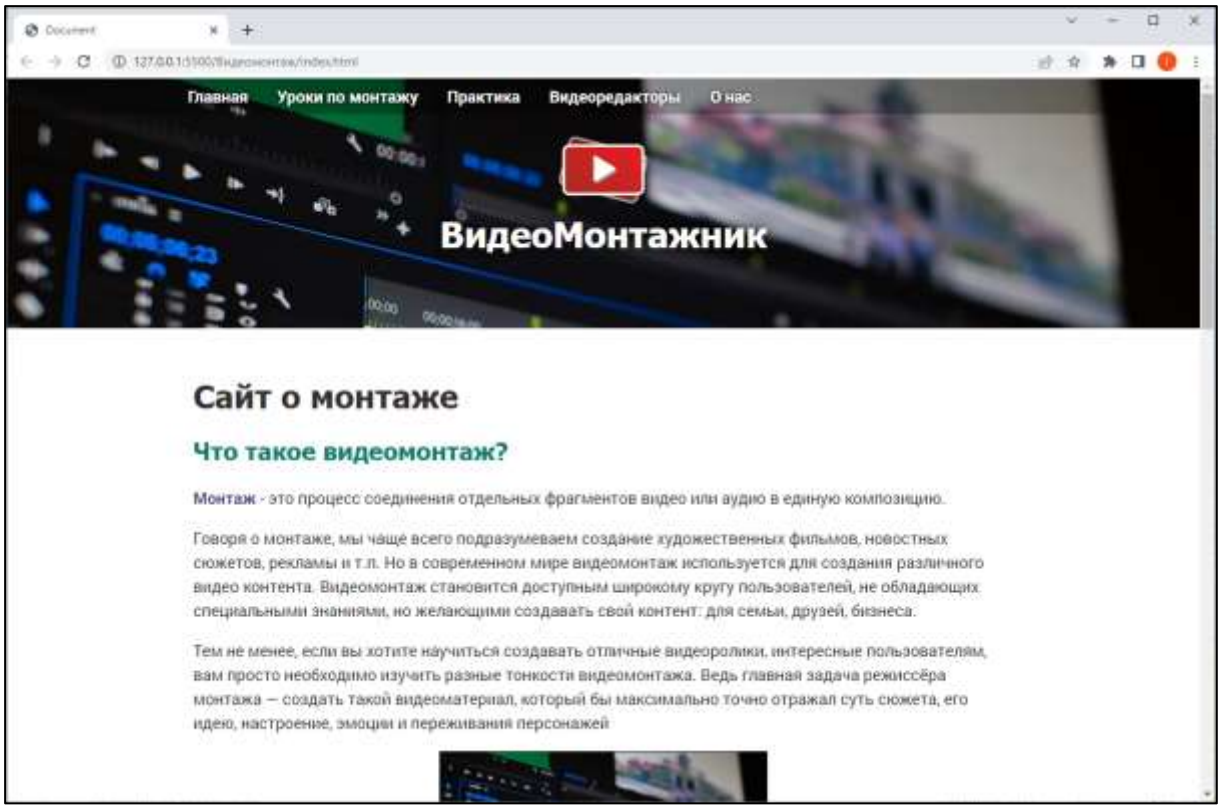


Рис. 3.147. Образец оформления задания 2 (часть 1)

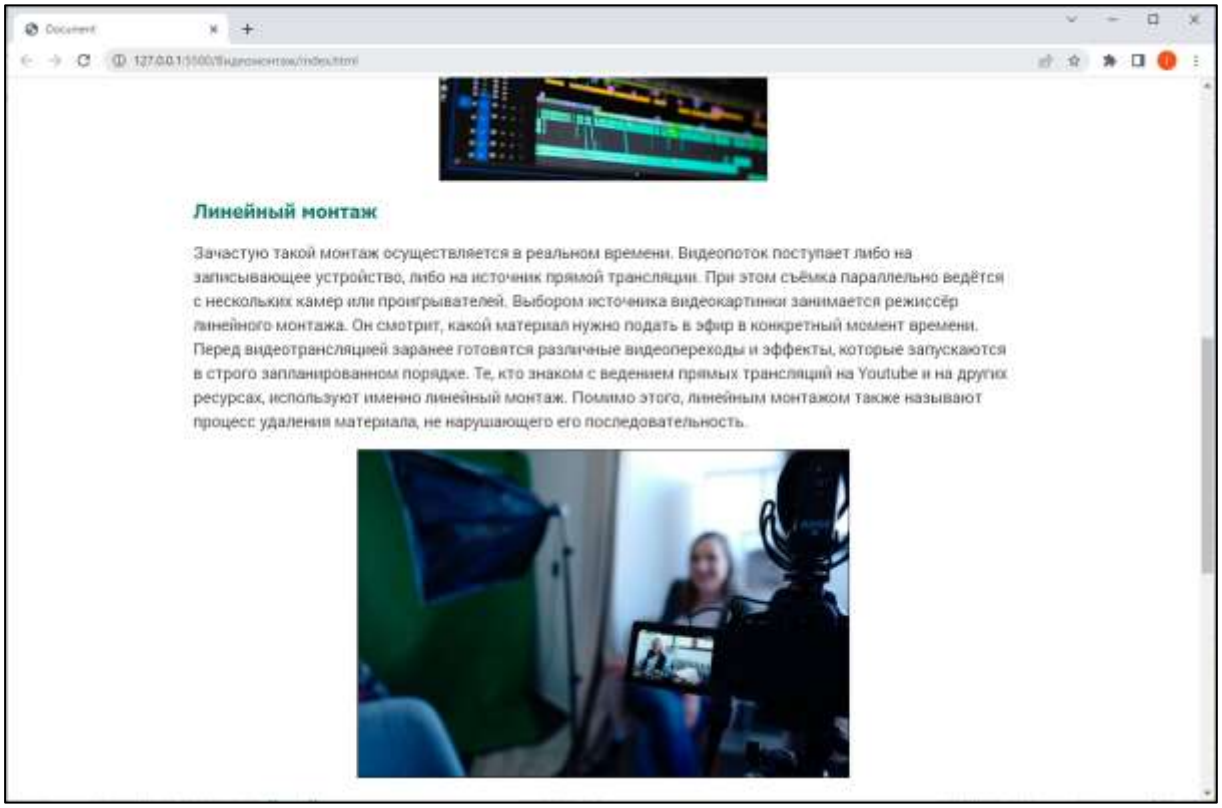


Рис. 3.148. Образец оформления задания 2 (часть 2)

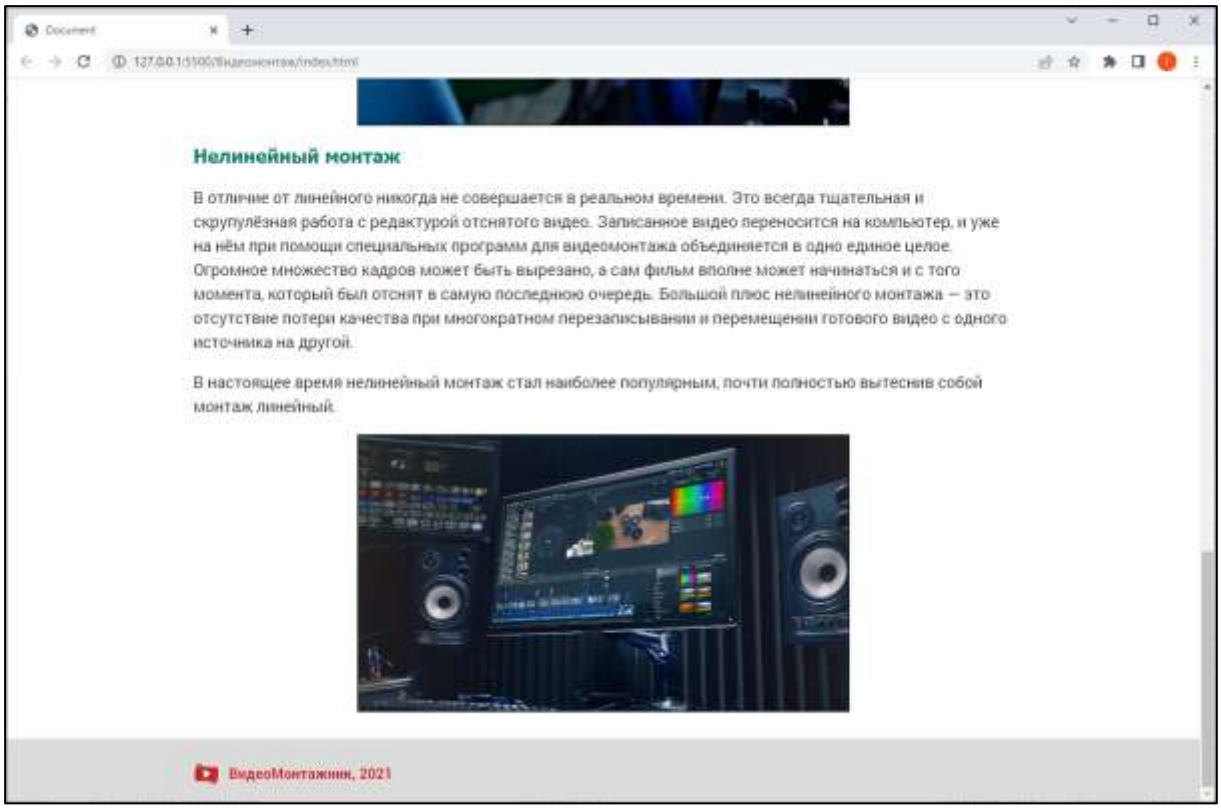


Рис. 3.149. Образец оформления задания 2 (часть 3)

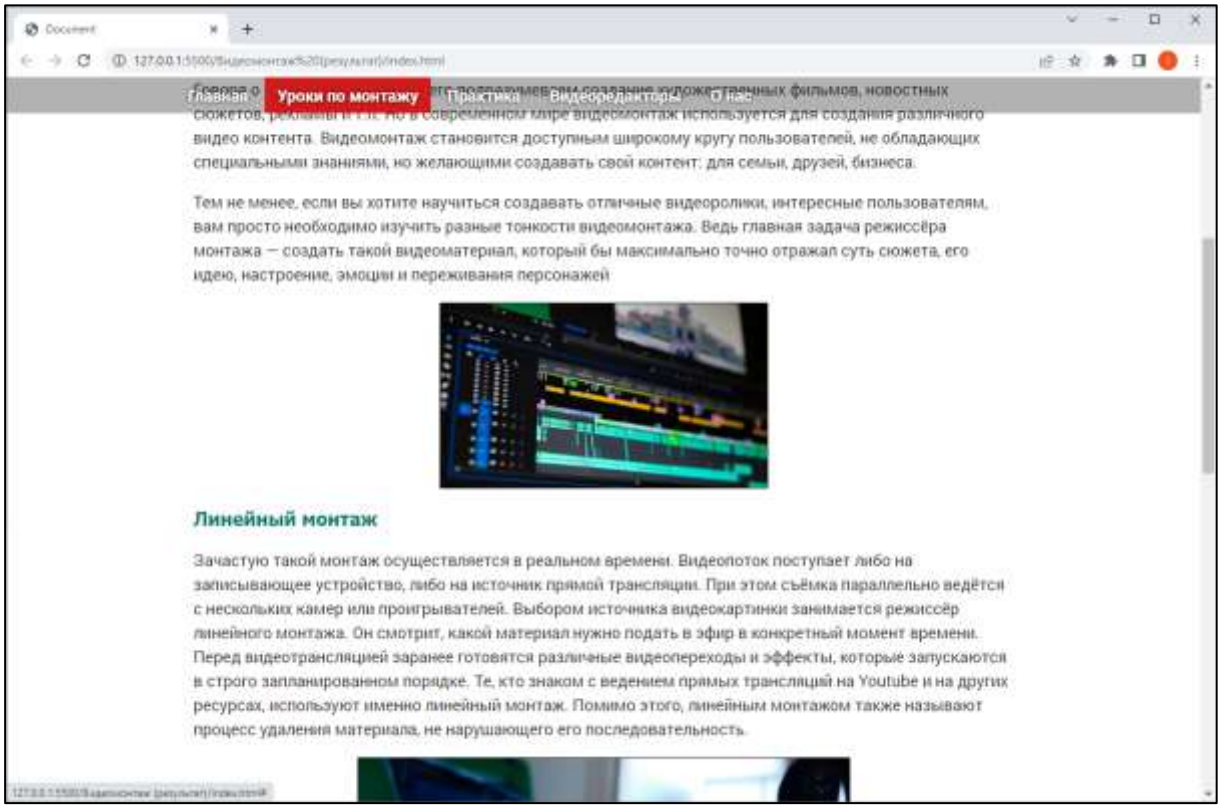


Рис. 3.150. Образец оформления задания 2 (зафиксированное меню)

3.8. Реализация проекта по верстке интерфейса электронного учебного курса

3.8.1. Постановка задачи

Описание

Продемонстрируем в целом изученные ранее возможности разметки и стилизации на практическом примере.

Пусть требуется сверстать веб-интерфейс для электронного учебного курса в форме веб-сайта. В простом случае его можно использовать в качестве веб-учебника, где каждая страница содержит описание учебного материала, ссылки на другие источники или страницы, видеоконтент и многое другое.

Электронные учебные курсы в обучении

Обычно *электронный учебный курс* (ЭУК) представляет собой тематически завершенный, качественно структурированный автором учебный материал, который посредством сети Интернет или на электронных носителях поставляется обучаемому.

В современных условиях ЭУК крайне востребованы, особенно в дистанционной форме обучения. Высокая популярность ЭУК связана с целым рядом преимуществ:

- предоставляют развитые иллюстративные возможности (изображения, аудио и видео, анимации, мультимедийные технологии);
- позволяют работать в интерактивном режиме;
- содержимое может меняться и адаптироваться под новые условия;
- поддерживают различные варианты контроля и оценки формируемых компетенций.

Создание ЭУК, как правило, требует значительных усилий, временных и даже финансовых затрат. Разрабатывать интерфейс курса с нуля необязательно: можно задействовать существующие платформы и системы управления ЭУК и электронным обучением, которые упрощают конструирование курсов, их дальнейшее администрирование и коммуникацию педагогического персонала с учащимися.

Примеры

В настоящее время в сети Интернет пользователям доступен обширный спектр ЭУК и образовательных платформ, где можно подобрать интересующую вас тематику и связанные с ней курсы. В подавляющем большинстве случаев прохождение дистанционных курсов требует регистрации на сайте, а также является платным.

Если рассматривать дистанционное обучение, то обычно в качестве инструмента конструирования и администрирования ЭУК для студентов используют систему Moodle. Она позволяет выстроить иерархию курсов внутри учебного заведения, подключать пользователей в роли администраторов, преподавателей и студентов, размещать цифровые образовательные материалы, вставлять различные интерактивные модули для коммуникации, управлять проверкой знаний и многое другое.

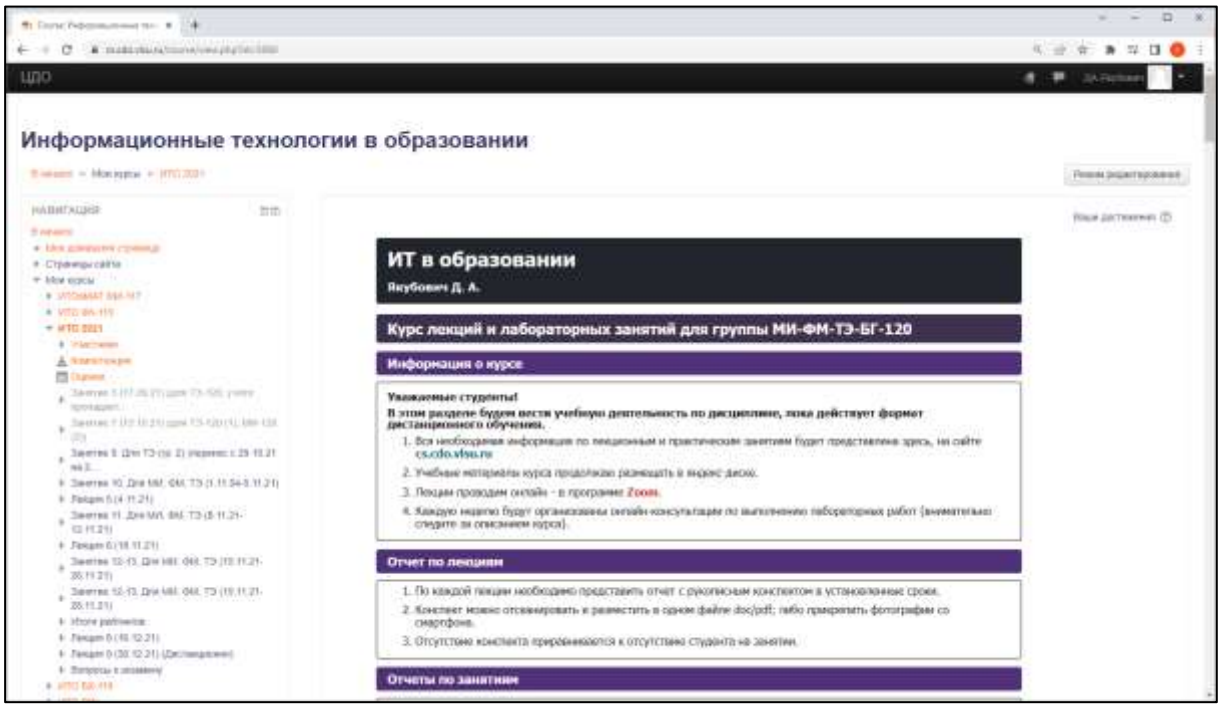


Рис. 3.151. СДО на базе LMS Moodle

В сети Интернет многие популярные учебные платформы реализуют собственный интерфейс ЭУК. Например, Coursera, Edx, iSpring LMS, Udemy, Stepik и др.

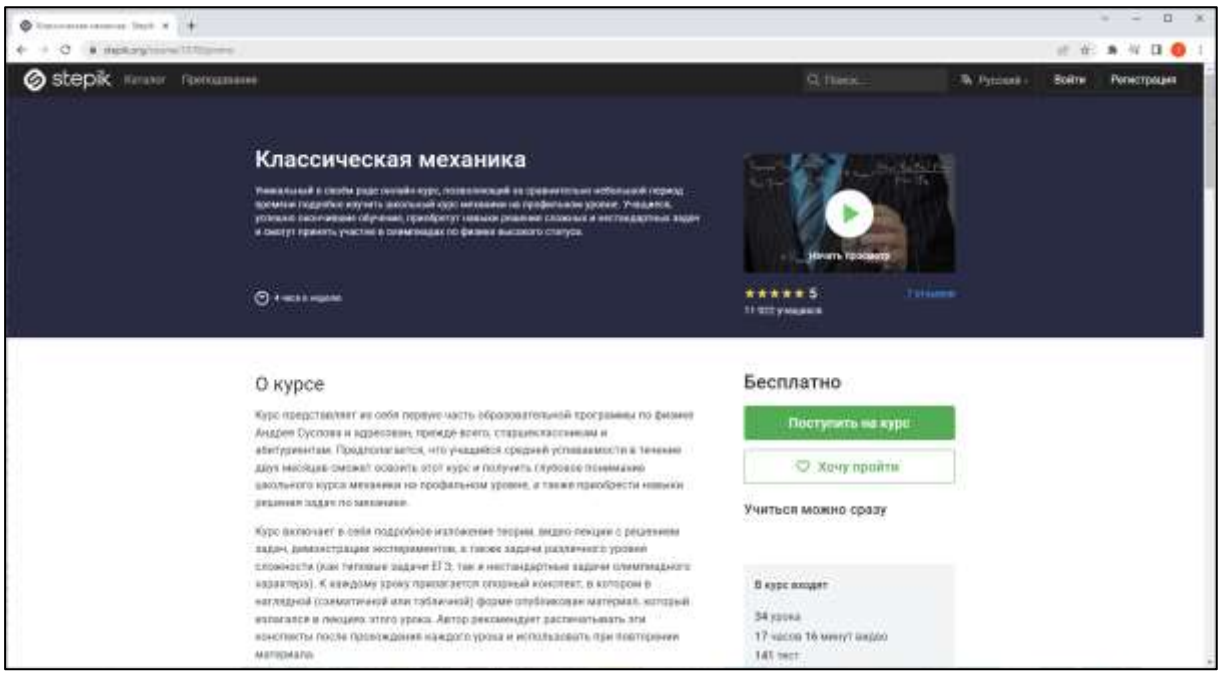


Рис. 3.152. Пример бесплатного ЭУК на Stepik.org

Крупные вузы в РФ и за рубежом также разрабатывают собственные веб-интерфейсы для электронного обучения.

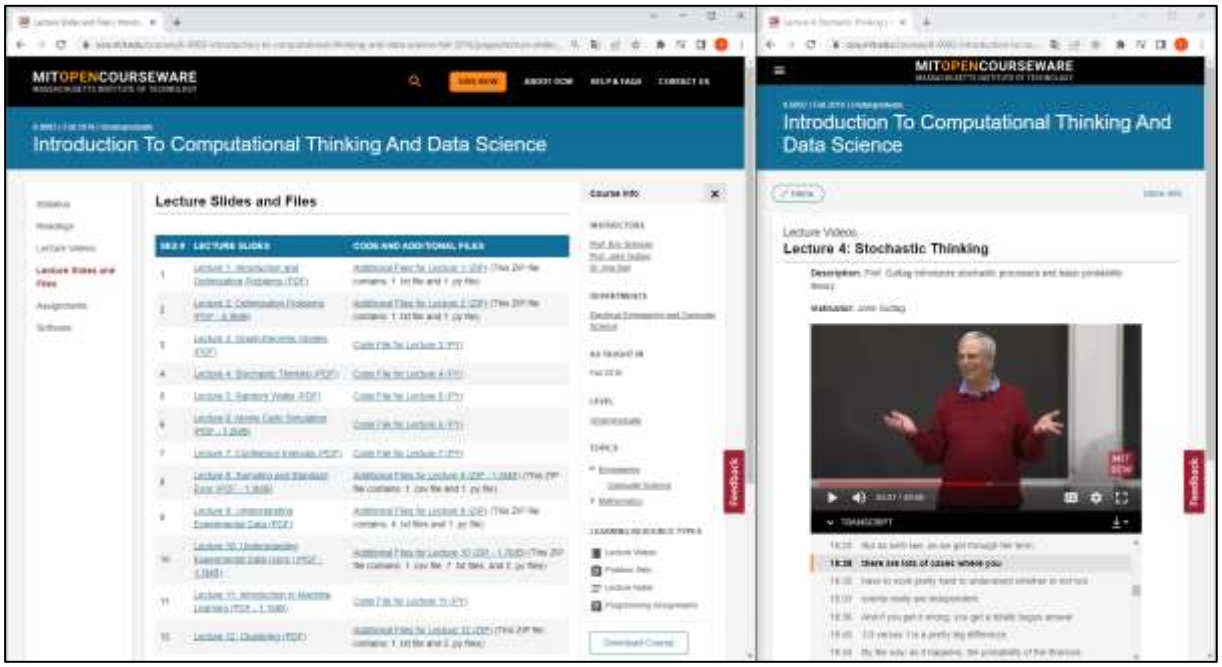


Рис. 3.153. Пример открытого курса MIT

Однако для создания курсов необязательно использовать CMS. Для простых интерфейсов достаточно и возможностей HTML, CSS и при необходимости обработки данных – JavaScript.

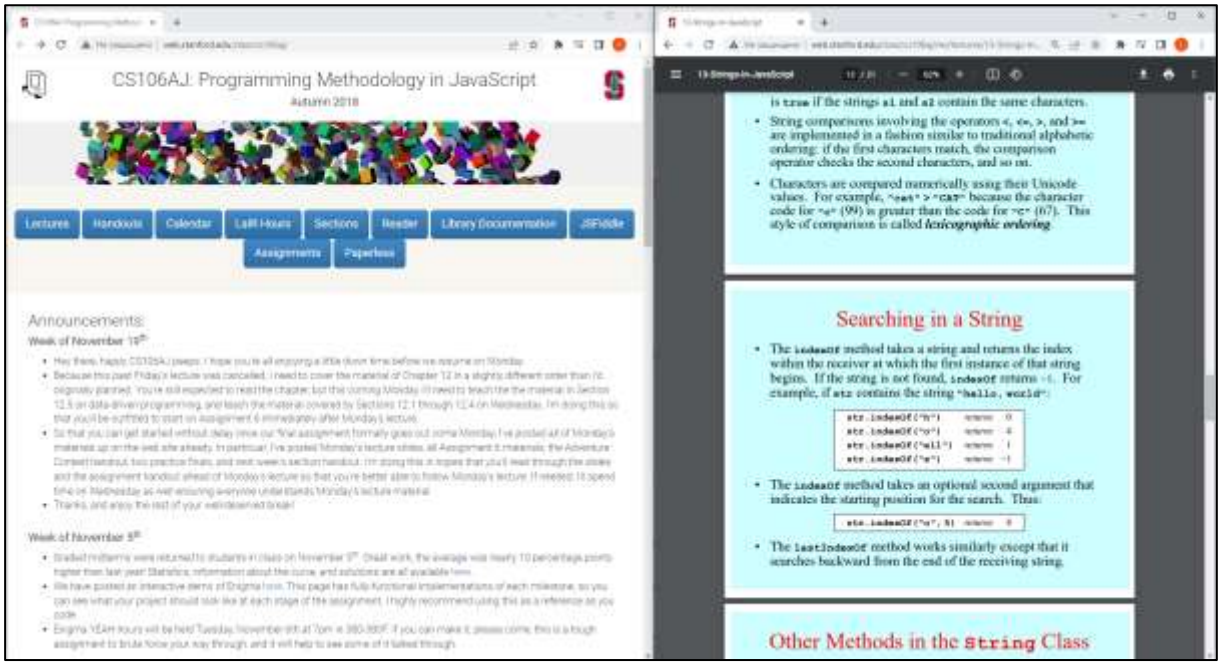


Рис. 3.154. Курс разработан преподавателем и размещен на сайте вуза

Требования к проекту

Для упрощения задачи опустим описание методической части, а также интерактивных элементов для авторизации пользователей, проведения тестирований и прочих возможностей.

В структуре ЭУК включим следующие разделы:

- титульная страница;
- главная страница, содержащая описание курса и инструктаж для учащихся;
- учебный план;
- материалы для занятий;
- КИМы;
- ссылки на дополнительные материалы.

Между разделами сайта курса требуется установить удобную навигацию.

Далее описана пошаговая реализация проекта. Ее необходимо воспроизвести!

3.8.2. Поэтапная реализация

1. Подготовка файлов

Цель работы – создать базовый интерфейс курса. Файл *index.html* открывает стартовую веб-страницу с данными о курсе и ссылкой на его основные разделы:

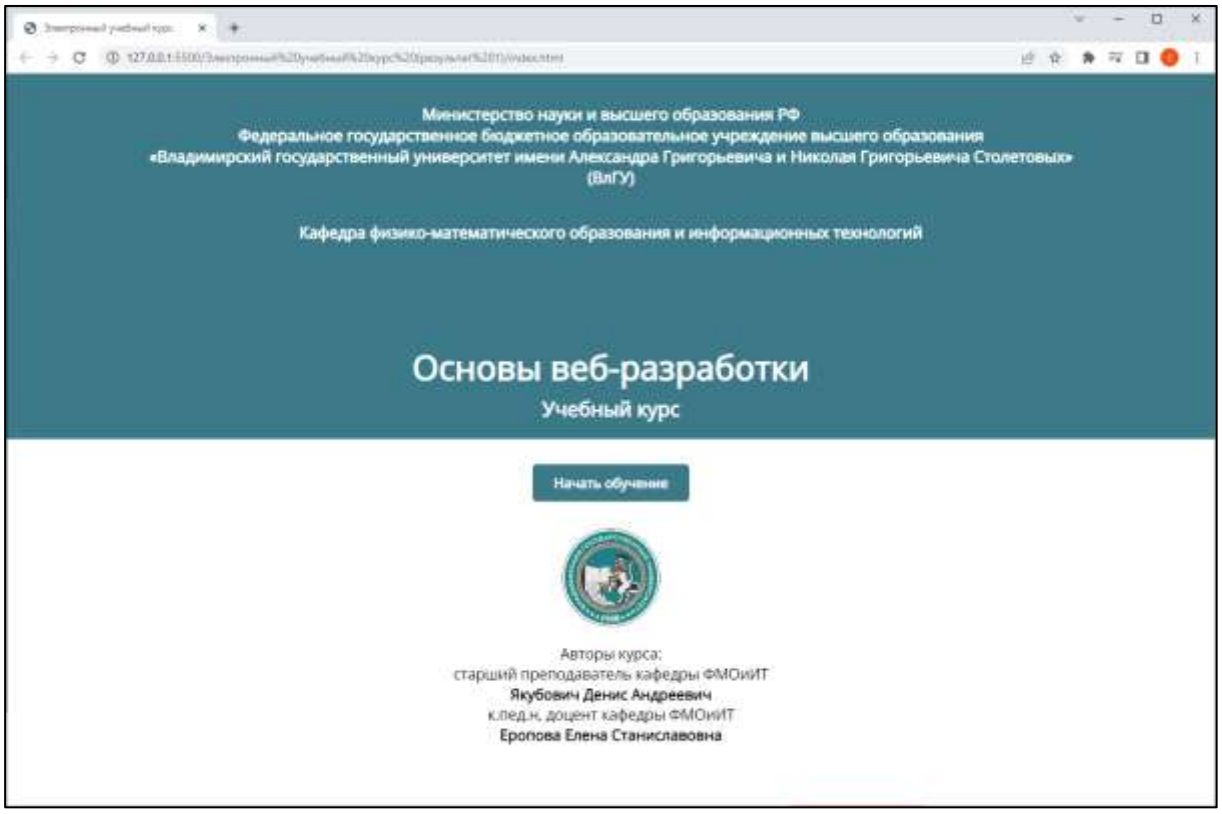


Рис. 3.155. Стартовая страница курса

По нажатию на ссылку осуществляется переход на страницу с описанием курса. С помощью гиперссылок меню можно будет переходить по основным подразделам курса: учебный план, занятия, контроль, дополнительные материалы. Будем придерживаться единого дизайна страниц разделов: это упростит верстку и сохранит стилистику оформления.

Для начала страницы и стили разместим в коренном каталоге. Изображения сгруппируем в каталоге *images*, а вспомогательные логотипы – *icons* (например, иконок для ссылки на социальные сети преподавателей).

Общая структура проекта будет выглядеть следующим образом:

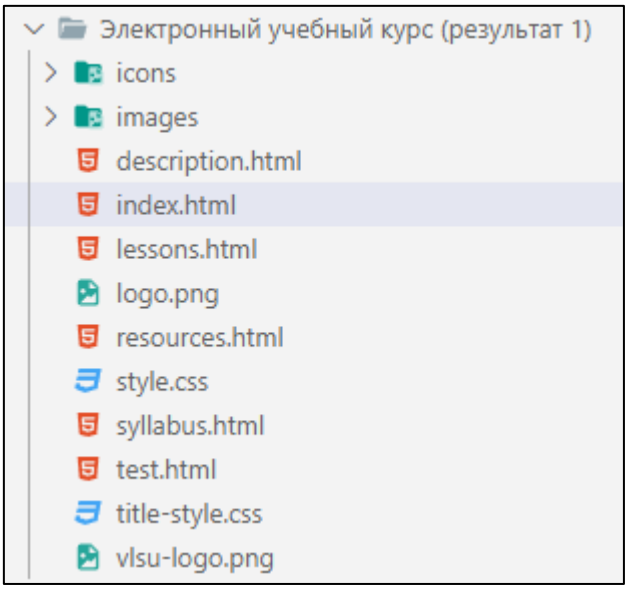


Рис. 3.156. Структура проекта (основные разделы, стили, каталоги)

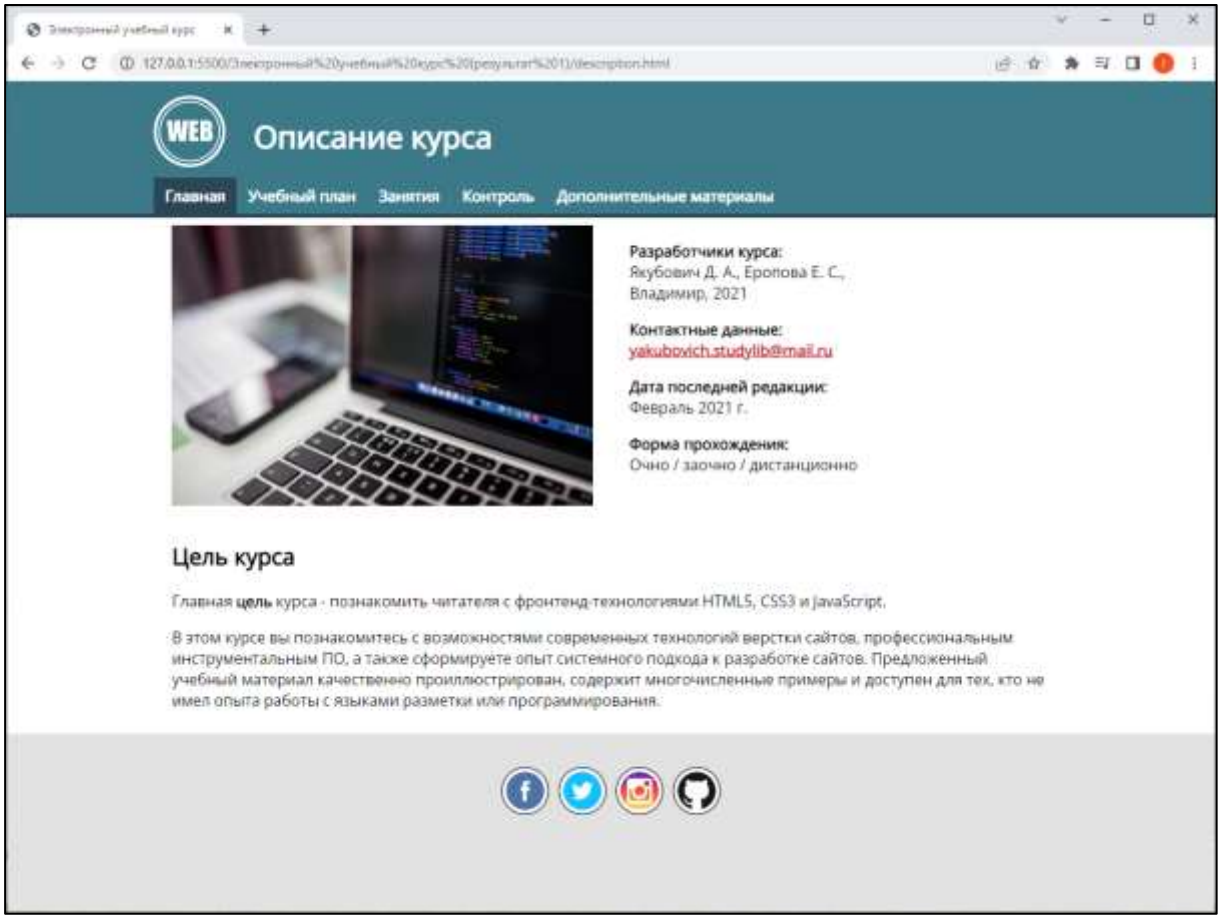


Рис. 3.157. Образец оформления страницы с описанием курса

2. Разметка и оформление титульной страницы

Создадим каталог с названием *Электронный учебный курс*. Добавим в него новый HTML-файл с *index.html* со стандартной начальной разметкой для титульной страницы курса (рис. 3.158).

Глава 3\Электронный учебный курс\index.html

```
<!DOCTYPE html>
<html lang="ru">

<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width,
initial-scale=1.0">
    <meta http-equiv="X-UA-Compatible" content="ie=edge">
    <title>Электронный учебный курс</title>
</head>

<body>

</body>

</html>
```

Рис. 3.158. Начальная страница курса

Создадим новый CSS-файл с названием *title-style.css* и подключим его в HTML-файле с помощью тега `<link>`. Также подключим новый шрифт Open Sans, который будет загружаться с ресурса Google Fonts: https://fonts.google.com/.

Глава 3\Электронный учебный курс\index.html

```
<!DOCTYPE html>
<html lang="ru">

<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width,
initial-scale=1.0">
    <meta http-equiv="X-UA-Compatible" content="ie=edge">
    <title>Электронный учебный курс</title>
    <link rel="stylesheet" href="title-style.css">
    <link
href="https://fonts.googleapis.com/css?family=Open+Sans"
rel="stylesheet">
</head>

<body>

</body>

</html>
```

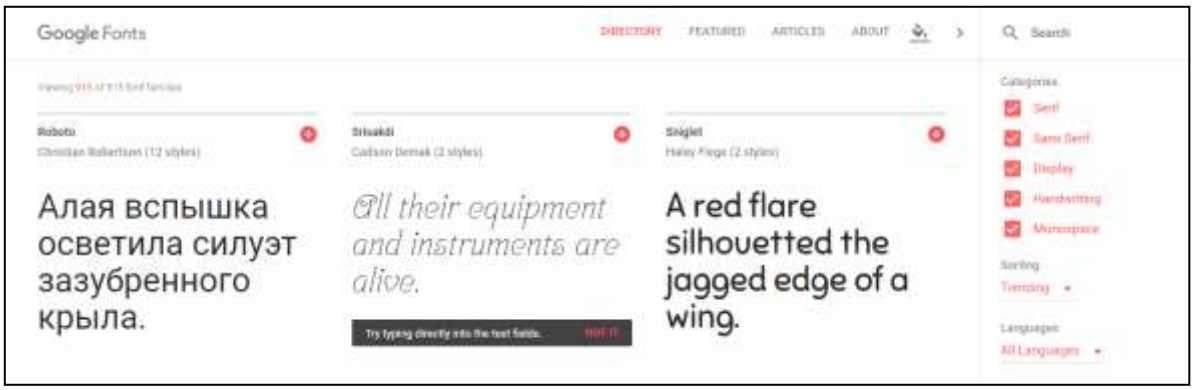


Рис. 3.159. Загрузка шрифта

Согласно планируемому дизайну, поделим разметку на два основных блока:

- в разделе <header> внесем информацию об университете, кафедре и названии курса;
- в разделе <main> разместим ссылку перехода к курсу, логотип вуза и информацию об авторе.

Как и ранее, оформление блокам и элементам будем привязывать с помощью классов.

Глава 3\Электронный учебный курс\index.html

```
<body>
    <header class="header">

    </header>
    <main class="main">

    </main>
</body>
```

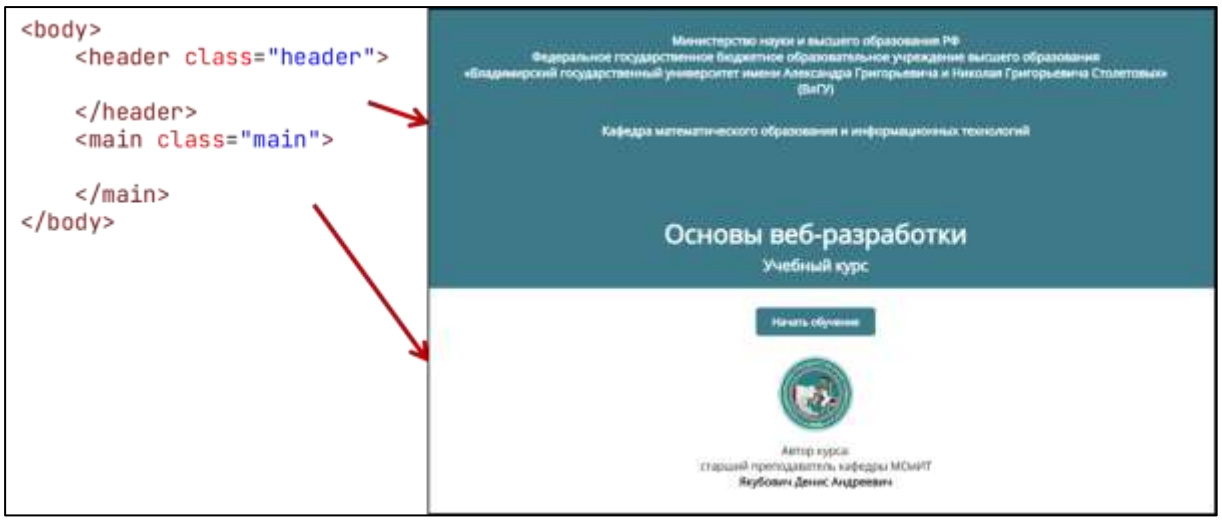


Рис. 3.160. Основные подразделы в разметке титульной веб-страницы

Зададим разметку внутри раздела <header>.

- Заголовок первого уровня отвечает за название курса.
- Заголовок второго уровня будем считать подзаголовком.
- В заголовках третьего уровня распишем информацию о вузе и кафедре.
- Для каждого заголовка укажем класс, чтобы далее им можно было настроить отдельные CSS-стили.

Глава 3\Электронный учебный курс\index.html

```
<header class="header">
    <h3 class="institute">
        Министерство науки и высшего образования РФ<br>
        Федеральное государственное бюджетное
        образовательное учреждение высшего
        образования<br>
        «Владимирский государственный университет имени
        Александра Григорьевича и Николая Григорьевича
        Столетовых»<br>
        (ВлГУ)
    </h3>
    <h3 class="department">Кафедра физико-математического
    образования и информационных технологий</h3>
    <h1 class="title">Основы веб-разработки</h1>
    <h2 class="subtitle">Учебный курс</h2>
</header>
```

В блоке контента `<main>` задаем разметку кнопке, логотипу вуза и информации об авторе.

Поскольку все объекты здесь необходимо выровнять по центру, то поместим их в контейнер с классом `center` (его опишем далее в стилях). Логотипу, как изображению, дополнительно требуется подключить этот класс, чтобы он выравнивал его уже внутри блока. Гиперссылку стилизуем под кнопку, которая ссылается на страницу с описанием курса (она же будет у нас главной).

Разумеется, каждому элементу зададим стиль по соответствующему классу.

Глава 3\Электронный учебный курс\index.html

```
<main class="main">
    <div class="center">
        <a href="description.html" class="start">
            Начать обучение
        </a>
        <img src="vlsu-logo.png"
             alt="Логотип"
             class="vlsu-logo center">
        <div class="author">
            <p>
                Авторы курса:<br>
                старший преподаватель кафедры ФМОиИТ<br>
                <strong>Якубович Денис
```

```
                    Андреевич</strong><br>
                    к.пед.н, доцент кафедры ФМОиИТ<br>
                    <strong>Еропова Елена
                    Станиславовна</strong>
                </p>
            </div>
        </div>
    </main>
```

Переходим к файлу со стилями. Для начала зададим глобальные настройки шрифта и уберем небольшие отступы полей (по умолчанию браузеры задает их тегу `<body>`, чтобы контент не располагался близко к границе):

Глава 3\Электронный учебный курс\title-style.css

```
body {
    font: 16px "Open Sans", Arial;
    margin: 0;
}
```

Далее опишем класс `center`, который выравнивает по центру содержимое блоков. Сделаем его весьма универсальным, чтобы можно было выравнивать как блоки, так и текст внутри них:

Глава 3\Электронный учебный курс\title-style.css

```
.center {
    display: table;
    margin: 0 auto;
    text-align: center;
}
```

Для блока `<header>` достаточно задать только заливку фона (через подколюченный одноименный класс):

Глава 3\Электронный учебный курс\title-style.css

```
.header {
    background: #3c7a89;
}
```

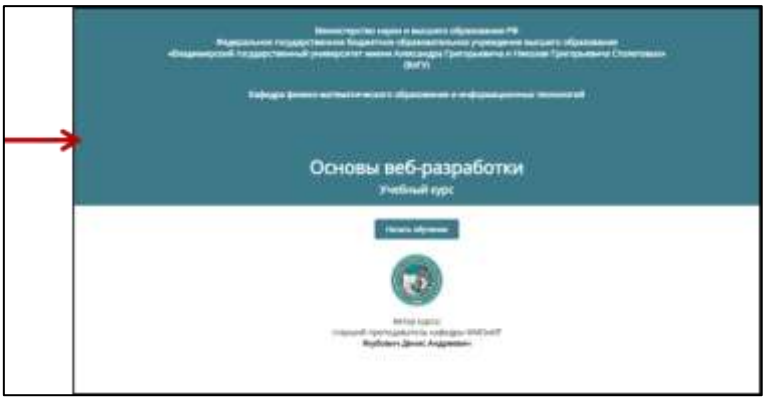


Рис. 3.161. Заливка блока заголовка

Все заголовки выравниваются по центру и должны быть оформлены белым цветом. Дополнительно уберем отступы полей (Рис. 3.162), а пространство между заголовками далее оформим через внутренние отступы. Используем возможность группировки селекторов:

Глава 3\Электронный учебный курс\title-style.css

```
.title,
.subtitle,
.institute,
.department {
    color: #fff;
    text-align: center;
    margin: 0;
}
```

Рис. 3.162. Сброс свойсва margin для заголовка, чтобы избежать эффекта «выталкивания» родительского блока

Ниже пропишем отдельные значения отступов заголовкам, чтобы визуально улучшить титул:

Глава 3\Электронный учебный курс\title-style.css

```
.title {
    font-size: 250%;
    padding: 115px 10px 5px;
}
```

```
.subtitle {
    padding-bottom: 20px;
}

.institute,
.department {
    padding: 35px 10px 5px;
}
```

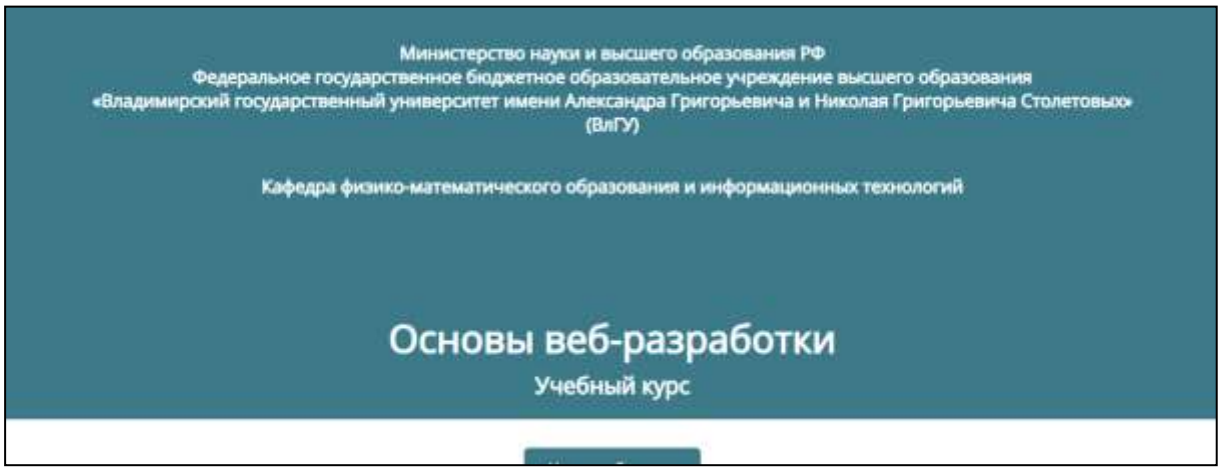


Рис. 3.163. Оформление верхнего блока завершено

Оформим ссылку перехода на главную страницу курса. Значительно переработаем ее свойства: сделаем блоком, уберем подчеркивание, изменим цвет фона и текста, установим жирное начертание, дополнительные поля и отступы. А с помощью псевдокласса `:hover` зададим свойства, которые она получит при наведении на нее курсора:

Глава 3\Электронный учебный курс\title-style.css

```
.start {
    display: inline-block;
    text-decoration: none;
    background: #3c7a89;
    color: #fff;
    font-weight: bold;
    padding: 10px 24px;
    margin: 30px 0;
    border: 1px solid #3c7a89;
    border-radius: 5px;
}
```

```
.start:hover {
    background: #fff;
    color: #3c7a89;
}
```

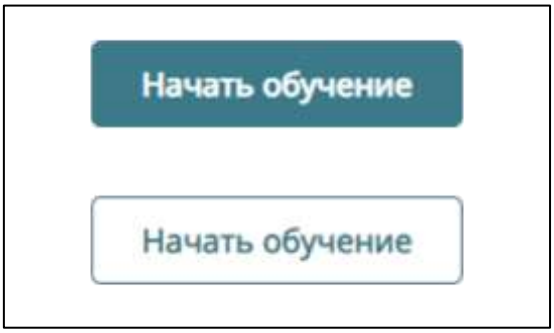


Рис. 3.164. Оформление ссылки в обычном режиме и при наведении курсора

Осталось оформить логотип и данные об авторах:

Глава 3\Электронный учебный курс\title-style.css

```
.vlsu-logo {
    display: block;
    width: 120px;
}

.author {
    font-size: 110%;
    margin-top: 15px;
}
```

Рис. 3.165. Оформление логотипа и реквизитов авторов курса

3. Разметка страниц основных разделов

Переходим к реализации главной страницы («Описание курса»). Чтобы упростить задачу, а также выдержать единый стиль оформления сайта, мы будем использовать одинаковое оформление для всех остальных страниц. Дизайн, к которому стремимся:

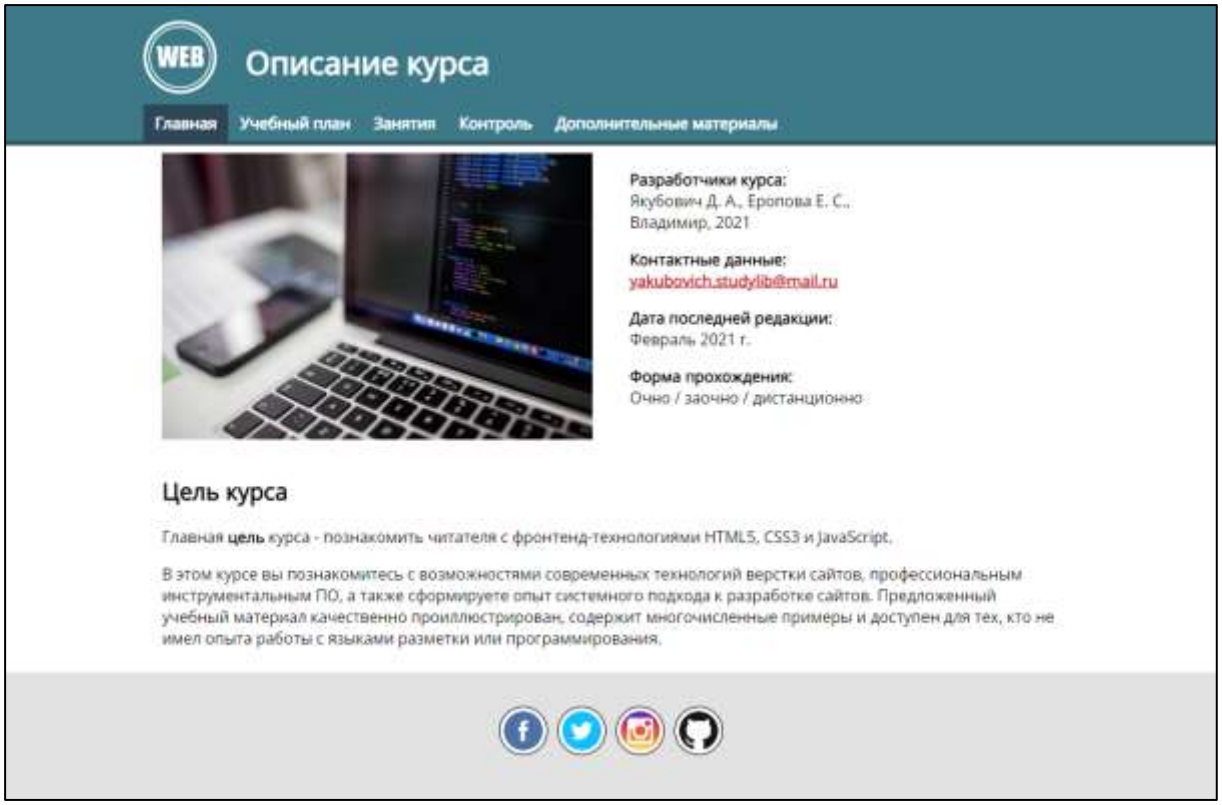


Рис. 3.166. Оформление главной страницы

Создаем страницу *description.html* и со стандартным стартовым шаблоном страницы. В ней подключаем уже новый стилевой файл *style.css* и Google-шрифт:

Глава 3\Электронный учебный курс\description.html

```
<!DOCTYPE html>
<html lang="ru">

<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width,
initial-scale=1.0">
    <meta http-equiv="X-UA-Compatible" content="ie=edge">
    <title>Электронный учебный курс</title>
    <link rel="stylesheet" href="style.css">
```

```
    <link
href="https://fonts.googleapis.com/css?family=Open+Sans"
rel="stylesheet">
</head>

<body>

</body>

</html>
```

Здесь макет разбивается уже на три основных области:

- раздел `<header>` содержит логотип, название и навигацию;
- раздел `<main>` размечает основное описание контента;
- раздел `<footer>` включает гиперссылки на аккаунты авторов (в виде иконок).

Глава 3\Электронный учебный курс\description.html

```
<body>
    <header class="header">

    </header>
    <main class="main">

    </main>
    <footer class="footer">

    </footer>
</body>
```

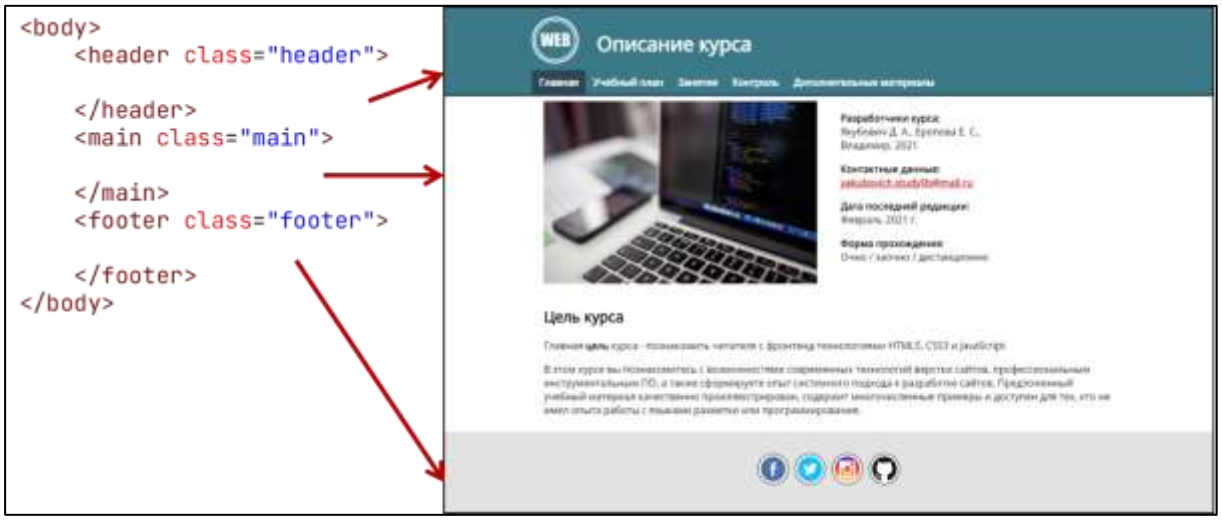


Рис. 3.167. Основные разделы разметки страницы

В стилевом файле *style.css* (его подключали к HTML-странице ранее) добавляем ряд начальных стилей. Из базового форматирования тегов поменяем только оформление гиперссылок:

Глава 3\Электронный учебный курс\style.css

```
* { box-sizing: border-box; }

body {
    font: 16px "Open Sans", Arial;
    background: #e2e2e2;
    margin: 0;
}

a {
    color: #ca1726;
    font-weight: bold;
}

.center {
    display: table;
    margin: 0 auto;
    text-align: center;
}
```

На страницах, посвященных основным разделам курса, разметка должна быть несколько сложнее, поскольку в целом содержит разные элементы, в отличие от титульной страницы.

Сделаем макет «резиновым». Но предельную ширину для контента ограничим, чтобы он не «расползался» на широкоформатных мониторах.

Обратите внимание: в нашем макете блоки шапки, контента и подвала имеют однородную заливку, но элементы располагаются в колонке с фиксированной максимальной шириной, которая выравнивается по центру. Мы не видим границ этого контейнера, однако он будет давать необходимое ограничение ширины и выравнивать содержимое (рис. 3.168).

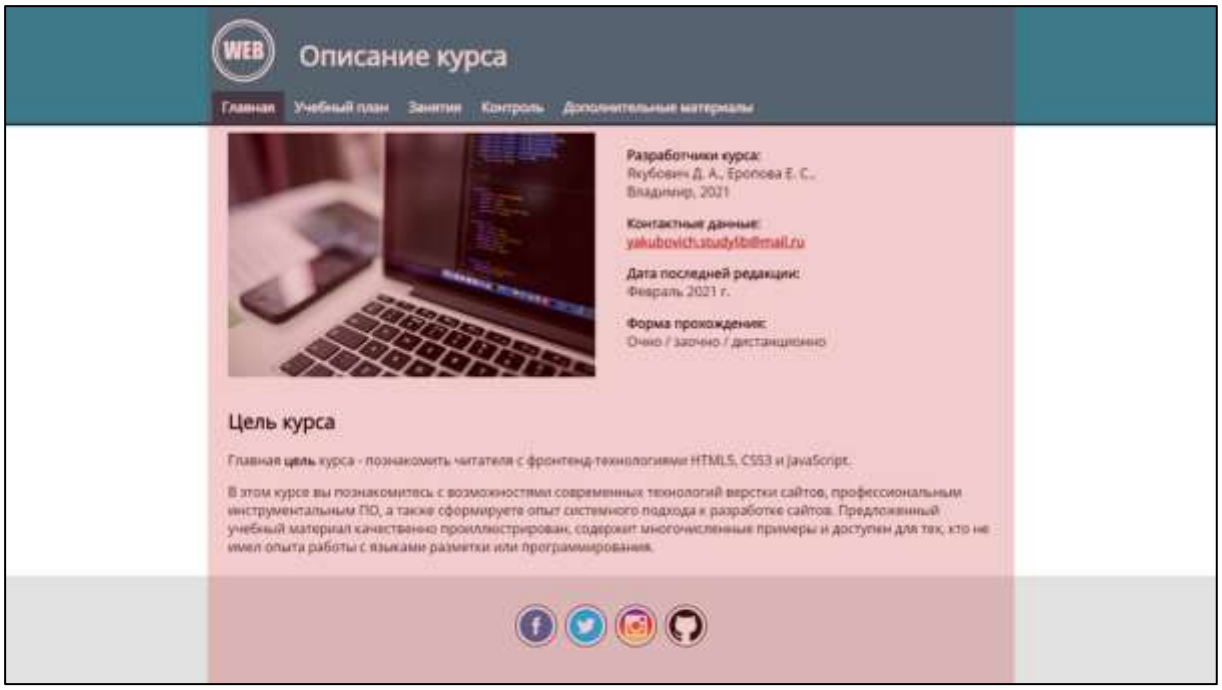


Рис. 3.168. Область, ограничивающая расползание контента

Чтобы добиться подобного эффекта, в каждый из блоков вставим промежуточный блок `<div>` с классом `row`, который выравнивает содержимое по центру и ограничивает его предельную ширину (см. далее в стилях).

Начнем с детализации разметки «шапки» страницы. Чтобы разместить логотип и заголовок рядом, их можно сделать float-блоками. А вот ссылки меню сделаем отдельным уровнем (рис. 3.169).

Таким образом, нам потребуется два блока, оформленных классом `row`.

Глава 3\Электронный учебный курс\description.html

```
<header class="header">
    <div class="row">
        <!-- Здесь будет логотип и название -->
    </div>

    <div class="row">
        <!-- Здесь будут ссылки меню -->
    </div>
</header>
```

Рис. 3.169. Невидимый контейнер row, который ограничивает предельную ширину содержимого в каждом «слое»

Опишем класс row в CSS-стилях. Содержимое этого блока будет растягиваться не более чем на 960px по ширине и выравниваться по центру, если ширина окна будет больше 960px.

Также учтем возможность обтекания float-блоков внутри:

Глава 3\Электронный учебный курс\style.css

```
.row {
    max-width: 960px;
    margin: 0 auto;
    overflow: auto;
}
```

Вернемся к разметке. В первый блок вставляем логотип. Причем делаем его ссылкой на титульную страницу. (Это обычная практика, когда логотип сайта ссылается на главную его страницу). Также вставляем название.

Во втором блоке делаем разметку меню. Как и ранее, подключаемые классы далее понадобятся для стилизации элементов.

Глава 3\Электронный учебный курс\description.html

```
<header class="header">
    <div class="row">
        <a href="index.html">
            <img src="logo.png" alt="Логотип"
                 class="header__logo">
        </a>
        <h1 class="header__title">Описание курса</h1>
    </div>
```

```
        <div class="row">
            <nav class="header__menu">
                <a href="description.html"
                   class="main-page">Главная</a>
                <a href="syllabus.html">Учебный план</a>
                <a href="lessons.html">Занятия</a>
                <a href="test.html">Контроль</a>
                <a href="resources.html">Дополнительные
                материалы</a>
            </nav>
        </div>
    </header>
```

Проработаем стили «шапки».

Для начала оформим сам блок: зададим ему заливку фона, небольшой отступ сверху (для контента), границу снизу, а также укажем обтекание для внутренних float-блоков (которыми будут логотип и заголовок):

Глава 3\Электронный учебный курс\style.css

```
.header {
    background: #3c7a89;
    padding-top: 15px;
    border-bottom: 4px solid #2e4756;
    overflow: auto;
}
```

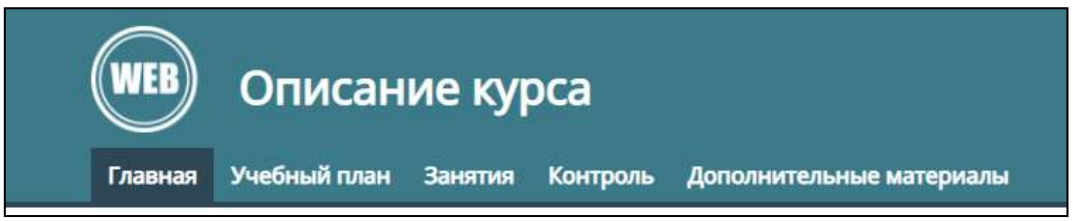


Рис. 3.170. Оформление блока «шапки»

Далее сделаем блоки логотипа и заголовка плавающими (свойством `float`): теперь они располагаются рядом. Для логотипа зафиксируем ширину (высота подберется пропорционально), а для заголовка поменяем цвет на белый и установим небольшой отступ слева, чтобы подвинуть его от логотипа (рис. 3.171):

Глава 3\Электронный учебный курс\style.css

```
.header__logo {
    float: left;
    width: 80px;
}

.header__title {
    float: left;
    color: #fff;
    padding-left: 25px;
}
```

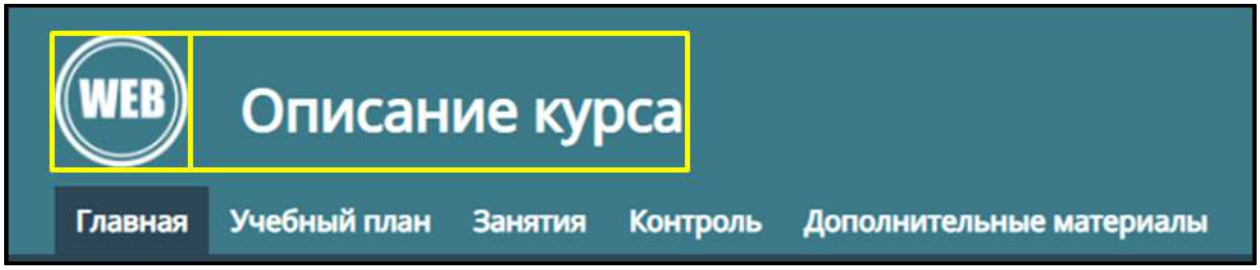


Рис. 3.171. Контейнеры логотипа и заголовка веб-страницы

Панель меню также оформим в форме идущих подряд гиперссылок. Стилизуем их под кнопки. Чтобы расположить их рядом и плотно друг к другу, вновь воспользуемся float-блоками. И добавим немного анимации при наведении курсора на ссылку:

Глава 3\Электронный учебный курс\style.css

```
.header__menu {
    overflow: auto;
}

.header__menu > a {
    float: left;
    text-decoration: none;
    color: #fff;
    font-weight: bold;
    padding: 8px 12px;
}

.header__menu > a:hover {
    background: #e8fbff;
    color: #000;
}
```

Рис. 3.172. Оформление панели с ссылками

Ссылку на главную страницу оформим немного иначе: выделим ее более темным фоном, а при наведении курсора зададим яркую подсветку (обратите внимание, что класс `main-page` мы подключали в разметке):

Глава 3\Электронный учебный курс\style.css

```
.header__menu > a.main-page {
    background: #2e4756;
}

.header__menu > a.main-page:hover {
    background: #e63946;
    color: #fff;
}
```

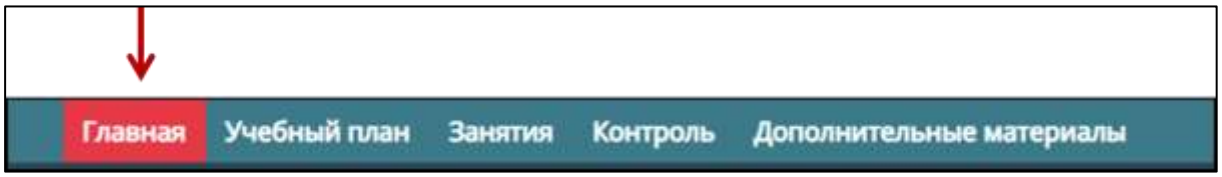


Рис. 3.173. Ссылка на главную страницу при наведении курсора

Блок «шапки» реализован.

Переходим к блоку контента `<main>`. Он не требует каких-либо существенных настроек: ограничимся лишь белым фоном, заданным через одноименный класс:

Глава 3\Электронный учебный курс\style.css

```
.main {
    background: #fff;
}
```

А вот содержимое `<main>` потребует более детальной проработки: это основной блок, в котором будут происходить изменения содержимого. Здесь у нас нет определенного ограничения на разметку и стили: их можно будет задавать как угодно.

Однако мы несколько упростим задачу, если будем использовать в разметке *колонки*. Этот подход используют в верстке печатных изданий (журналов, газет, постеров и др.). Профессиональные редак-

торы и издательские системы с помощью направляющих позволяют быстро разметить текст и достичь весьма привлекательного дизайна.

Далее мы создадим шесть классов, задающих колонки разной ширины. С помощью колонок удобно размещать содержимое, сохраняя при этом пропорции (сами колонки не отображаются).

Отталкиваемся от простого макета в шесть колонок. *Колонкой* будем называть плавающий блок с небольшим левым и правым отступом:

Рис. 3.174. Пример 6-ти колоночной разметки с помощью float-блоков

Для начала оформим классы. Минимальная по ширине колонка равна 1/6 ширины родителя, а максимальная – 6/6=1, т.е. занимает всю ширину родителя.

Все колонки сделаем float-блоками, что позволит размещать их несколько в ряд. Внутри колонок также можно размещать контент и даже сами колонки. У каждой колонки своя ширина относительно родителя (1/6, 2/6 и т.д. до 1)

Глава 3\Электронный учебный курс\style.css

```
.col-1,
.col-2,
.col-3,
.col-4,
.col-5,
.col-6 {
    float: left;
    padding: 0.5em 2%;
    overflow: hidden;
}
```

```
.col-1 { width: 16.66%; }
.col-2 { width: 33.33%; }
.col-3 { width: 50%; }
.col-4 { width: 66.66%; }
.col-5 { width: 83.33%; }
.col-6 { width: 100%; }
```

После реализации стилей колонок их можно использовать в разметке. Возвращаемся к блоку `<main>`. Колонки размещаем в блоке с классом `row`: он будет выполнять роль контейнера, отделяя колонки отдельными строками («уровнями», «полосами»):

Глава 3\Электронный учебный курс\description.html

```
<main class="main">
    <div class="row">
        <div class="col-3">
            <!-- Здесь будет изображение -->
        </div>
        <div class="col-3">
            <!-- Здесь будут данные об авторах -->
        </div>
    </div>

    <div class="row">
        <div class="col-6">
            <!-- Здесь будет описание курса -->
        </div>
    </div>
</main>
```

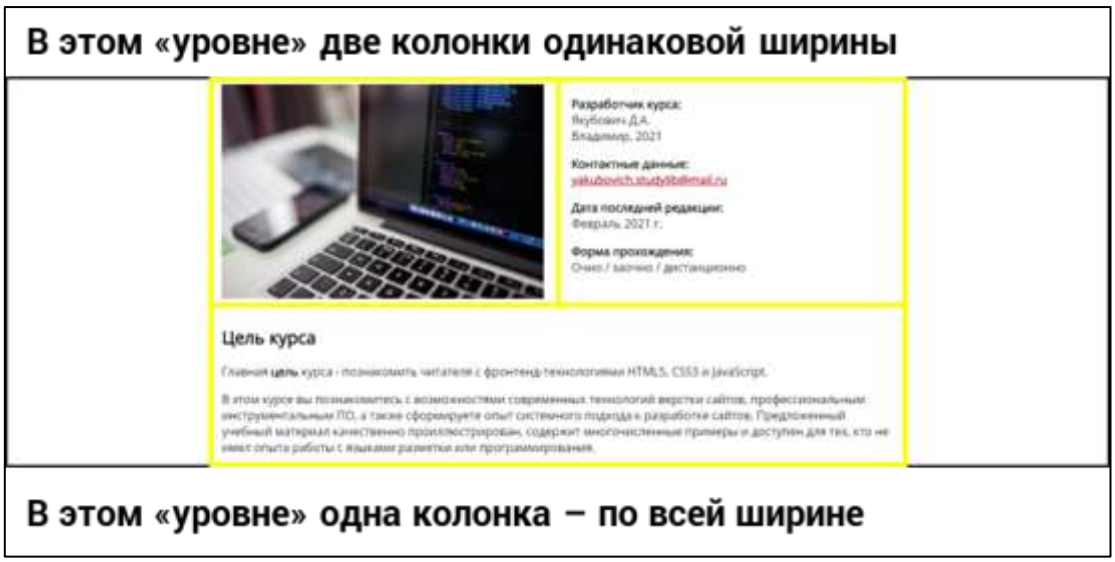


Рис. 3.175. Выделение полос для колонок

Главное, чтобы в сумме ширина колонок была равна 100%.

Также обратите внимание, что мы учли это в названии колонок: сумма цифр всех колонок «уровня» быть равной 6.

В первой полосе будет две равные колонки. Вносим в нее изображение и описываем его стиль `big-picture` (как класс для оформления больших изображений):

Глава 3\Электронный учебный курс\description.html

```
<main class="main">
    <div class="row">
        <div class="col-3">
            <img src="images/webdev.png"
                 alt="Изображение"
                 class="big-picture">
        </div>
        <div class="col-3">
            <!-- Здесь будут данные об авторах -->
        </div>
    </div>

    <div class="row">
        <div class="col-6">
            <!-- Здесь будет описание курса -->
        </div>
    </div>
</main>
```

Глава 3\Электронный учебный курс\style.css

```
.big-picture {
    display: block;
    width: 100%;
}
```

Во вторую колонку вносим данные об авторе:

Глава 3\Электронный учебный курс\description.html

```
. . .
<div class="col-3">
    <p>
        <strong>Разработчики курса:</strong><br>
        Якубович Д. А., Еропова Е. С.,<br>
        Владимир, 2021
    </p>
```

```
        <p>
            <strong>Контактные данные:</strong><br>
            <a href="mailto:yakubovich.studylib@mail.ru">
                yakubovich.studylib@mail.ru
            </a>
            <br>
            <a href="mailto:eropova13061962@mail.ru">
                eropova13061962@mail.ru
            </a>
        </p>
        <p>
            <strong>Дата последней редакции:</strong><br>
            Февраль 2021 г.
        </p>
        <p>
            <strong>Форма прохождения:</strong><br>
            Очно / заочно / дистанционно
        </p>
    </div>
    . . .
```

Вторая полоса содержит одну колонку на всю ширину родительского блока и размечает описание самого курса:

Глава 3\Электронный учебный курс\description.html

```
    . . .
    <div class="col-6">
        <h2>Цель курса</h2>
        <p>Главная <strong>цель</strong> курса - познакомить
        читателя с фронтенд-технологиями HTML5, CSS3 и
        JavaScript.</p>
        <p>В этом курсе вы познакомитесь с возможностями
        современных технологий верстки сайтов,
        профессиональным инструментальным ПО, а также
        сформируете опыт системного подхода к разработке
        сайтов. Предложенный учебный материал качественно
        проиллюстрирован, содержит многочисленные примеры и
        доступен для тех, кто не имел опыта работы с языками
        разметки или программирования.</p>
    </div>
    . . .
```

Осталось реализовать «подвал». В нем разместим гиперссылки на аккаунты авторов в социальных сетях и веб-сервисах. Содержимое выравниваем по центру, а оформление иконок определим через класс `icon` (далее его опишем).

Обратите внимание: иконки размещаются в каталоге *icons* и это важно учитывать при адресации в атрибуте `src`:

Глава 3\Электронный учебный курс\description.html

```
<footer class="footer">
    <div class="row">
        <div class="center">
            <a href="#">
                <img src="icons/fb.png"
                     alt="facebook"
                     class="icon">
            </a>
            <a href="#">
                <img src="icons/twt.png"
                     alt="twitter"
                     class="icon">
            </a>
            <a href="#">
                <img src="icons/inst.png"
                     alt="instagram"
                     class="icon">
            </a>
            <a href="#">
                <img src="icons/git.png"
                     alt="github"
                     class="icon">
            </a>
        </div>
    </div>
</footer>
```

Вместо #, разумеется, должны быть указаны ссылки на конкретные веб-ресурсы.

Оформляем «подвал» и стилизуем иконки. Поскольку логотипы предварительно обрезались в графическом редакторе до квадратов, то свойство `border-radius` со значением 50% оформит его в виде круга (для прямоугольных изображений получится овал).

Также продемонстрируем работу с функцией трансформации и фильтра изображений. При наведении курсора на иконку она будет увеличена в 1.1 раз (на 10%), а также усилена ее насыщенность:

Глава 3\Электронный учебный курс\style.css

```
.footer {
    padding: 30px 0 20px;
}

.icon {
    display: inline-block;
    height: 50px;
    background: #fff;
    padding: 4px;
    margin: 5px;
    border: 1px solid #000;
    border-radius: 50%;
    transition: 0.15s;
}

.icon:hover {
    transform: scale(1.1);
    filter: saturate(175%);
}
```

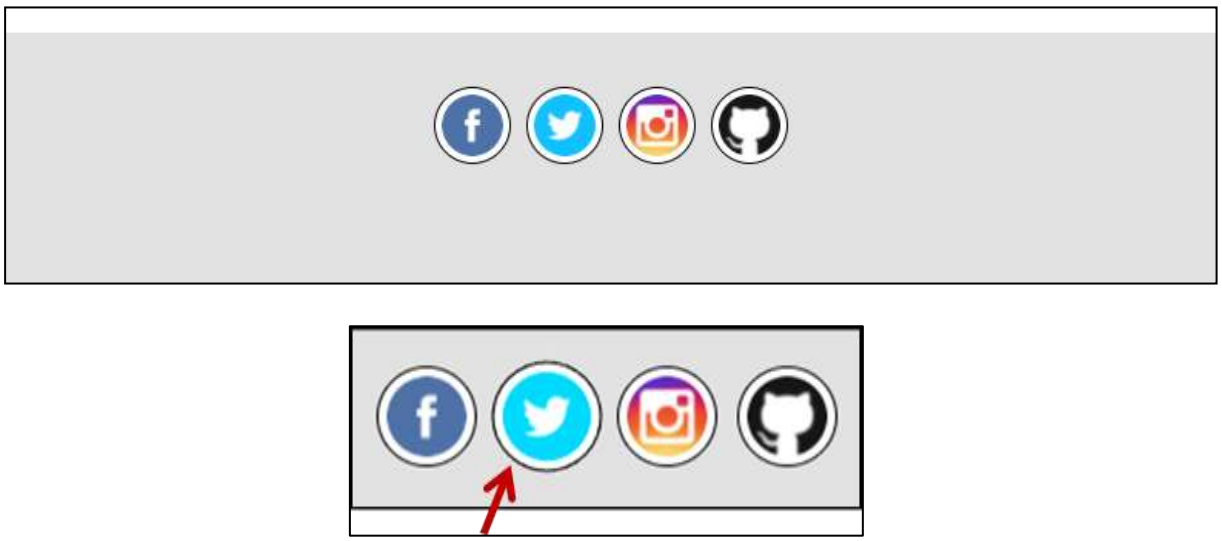


Рис. 3.176. Оформление ссылок в «подвале». При наведении курсора добавляются некоторые эффекты анимации

4. Работа над адаптивным дизайном

В целом на этом этапе можно завершить верстку основы страницы и по аналогии верстать остальные разделы курса из меню.

Однако реализованный дизайн будет хорошо согласован лишь для просмотра на стационарном ПК или ноутбуке. Если же ширина страницы в браузере достаточно маленькая, то содержимое колонок будет смотреться крайне сжато:

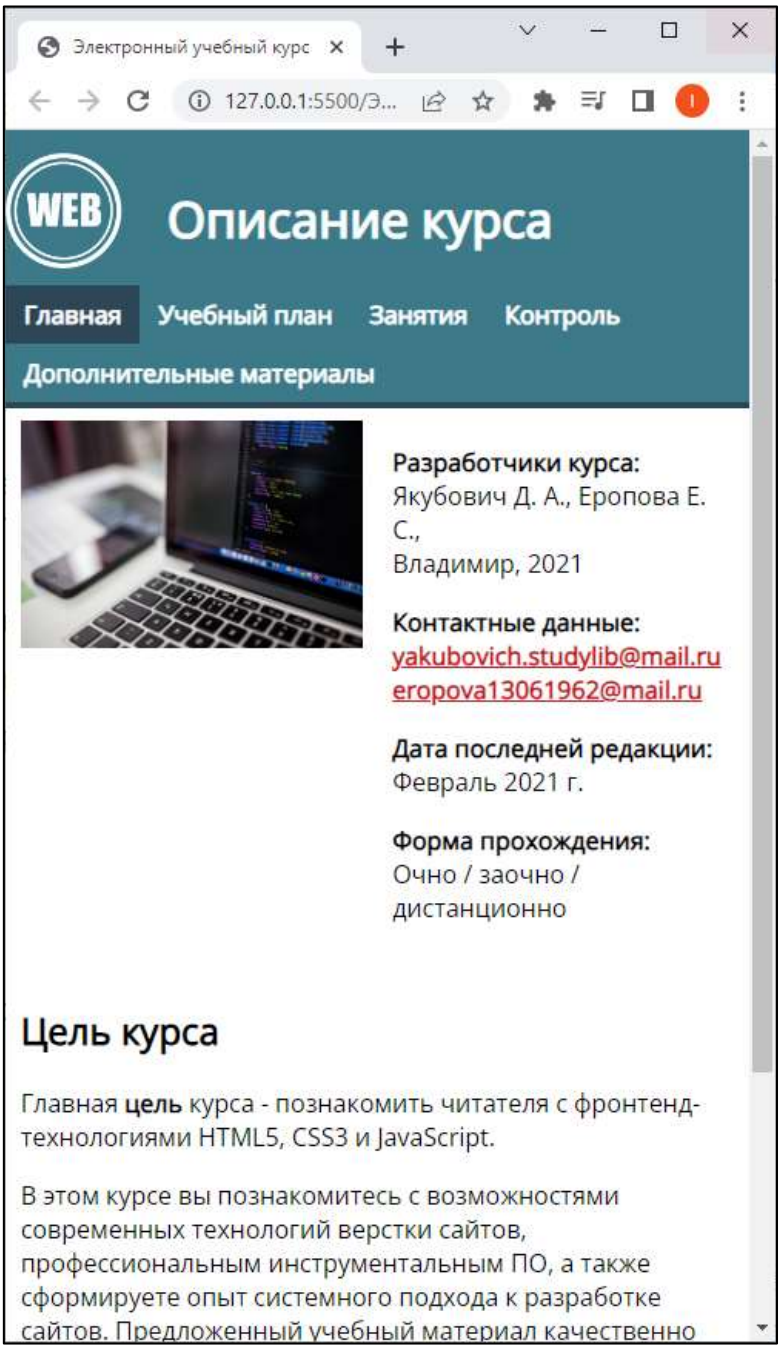


Рис. 3.177. Колонки получаются слишком узкими

В современных условиях при верстке веб-страниц отдельное внимание уделяют *адаптивному дизайну*, т.е. оформлению, которое способно изменяться, например, при разной ширине окна браузера или используемом устройстве.

Один из механизмов реализации адаптивного дизайна в CSS3 – *медиа-запросы*.

Определение

Медиа-запрос *– это CSS-стиль (или их набор), срабатывающий при определенных условиях.*

```
@media [условие срабатывания] {
    /* Здесь описаны селекторы и их стили */
}
```

Например, следующий медиа-запрос сработает для всех устройств (экрана, планшета, смартфона, проектора, принтера и т.д.) и в том случае, когда ширина окна браузера стала меньше чем 540px:

```
@media all and (max-width: 540px) {
    /* Здесь описаны селекторы */
}
```

Внутри медиа-запроса осуществляется описание новых или перезапись свойств стилей ранее заданных селекторов.

Чтобы решить проблему узких колонок, поступим следующим образом. Как только ширина окна уменьшится, скажем, до 540px, все колонки сделаем шириной в 100%. Тогда, если в полосе `row` было несколько колонок, они последовательно вытеснят друг друга вниз, отобразившись отдельными слоями (рис. 3.178).

Плавающие колонки в этом случае начинают сами играть роль полос, на которые расслаивается уровень из нескольких колонок. Медиазапросы автоматически контролируют соблюдение условий и активацию (деактивацию) селекторов.

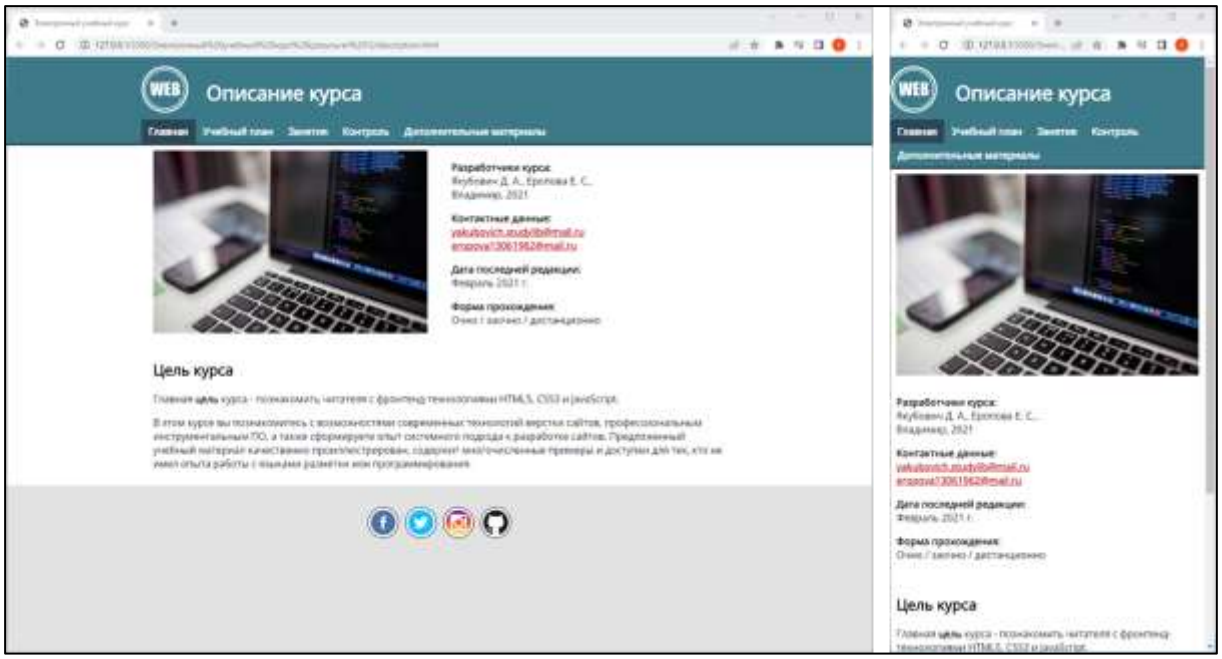


Рис. 3.178. При достижении порога ширины окна в 540px колонки расслаиваются отдельными полосами

Добавим в конце файла со стилями следующий медиа-запрос:

Глава 3\Электронный учебный курс\style.css

```
@media all and (max-width: 540px) {
    .col-1,
    .col-2,
    .col-3,
    .col-4,
    .col-5,
    .col-6 {
        width: 100%;
    }
}
```

Этот запрос сработает, если ширина окна браузера станет меньше 540px: все колонки будут растягиваться по всей ширине и вытеснять друг друга вниз.

Как только ширина превзойдет это значение, колонки вновь примут указанные ранее относительные пропорции ширины.

Теперь созданную страницу можно использовать в качестве образца для разметки основной структуры других страниц курса.

Вопросы для самопроверки

1. Чем удобны ЭУК и как их можно разрабатывать?
2. Что обычно входит в структуру ЭУК?
3. Приведите примеры платформ для конструирования и размещения ЭУК.
4. В чем преимущество использования колонок в разметке страниц и как их можно реализовать с помощью HTML и CSS?
5. Что представляют собой медиа-запросы?

Практикум

Задание 1

1. Создайте каталог *Электронный учебный курс*.
2. Подготовьте необходимые файлы изображений: логотипы, иконки и др.
3. Реализуйте последовательно описанные в этом параграфе шаги и получить разметку стартовой и начальной страницы курса (рис. 3.155, рис. 3.157).

Задание 2

1. По аналогии со страницей *description.html*, реализуйте основу для остальных страниц, согласно ссылкам меню (путем копирования основы).
2. Временно на каждую добавьте разметку тела, как изображено на рис. 3.179.
3. Для вставки «гифки» в более современном формате webm используйте тег `<video>`:

```
<video src="images/loveyoustepan.webm"
       class="center"
       autoplay
       loop
       muted>
</video>
```

Задание 3

1. Добавьте в подвал каждой страницы ссылку-иконку на аккаунт Вконтакте, сохранив форматирование.
2. Также скорректируйте все ссылки, чтобы они вели на соответствующие сайты (рис. 3.180).
3. Используя возможности созданных стилей для колонок, на главной странице добавьте три колонки и описание в каждой, как на рис. 3.181.
4. Изображениям задайте одинаковую высоту (создайте для их оформления отдельный класс).
5. Внесите по своему усмотрению некоторые стилевые изменения в оформление основных тегов: `<body>`, `<h1-h3>`, `<p>`, `<strong>`, `<a>` и др (рис. 3.182).

Задание 4

1. Добавьте в описание медиа-запроса стили.
2. Требуется, чтобы при срабатывании медиазапроса:
   a. растягивают каждую ссылку меню по всей ширине окна;
   b. скрывают изображения логотипов из колонок (для этого создайте класс `hidden` и используйте свойство `display` со значением `none`);
   c. дополнительно пропишите некоторым тегам свойства, которые будут изменены медиа-запросом (например, несколько уменьшены интервалы, отступы, шрифт или т.п.), рис. 3.183.

Задание 5

1. Создайте копию каталога с проектом.
2. Разбейте описание стилей и медиа-запроса на отдельные файлы. Для этого создайте новый файл *mobile-style.css* и перенесите в него описание медиа-запроса.
3. Далее на каждой странице подключите эти стили с помощью тега `<link>`. Следуйте порядку: в начале подключаются основные стили `style.css`, далее – `mobile-style.css`.
4. Ориентируйтесь на структуру, изображенную на рис. 3.184.

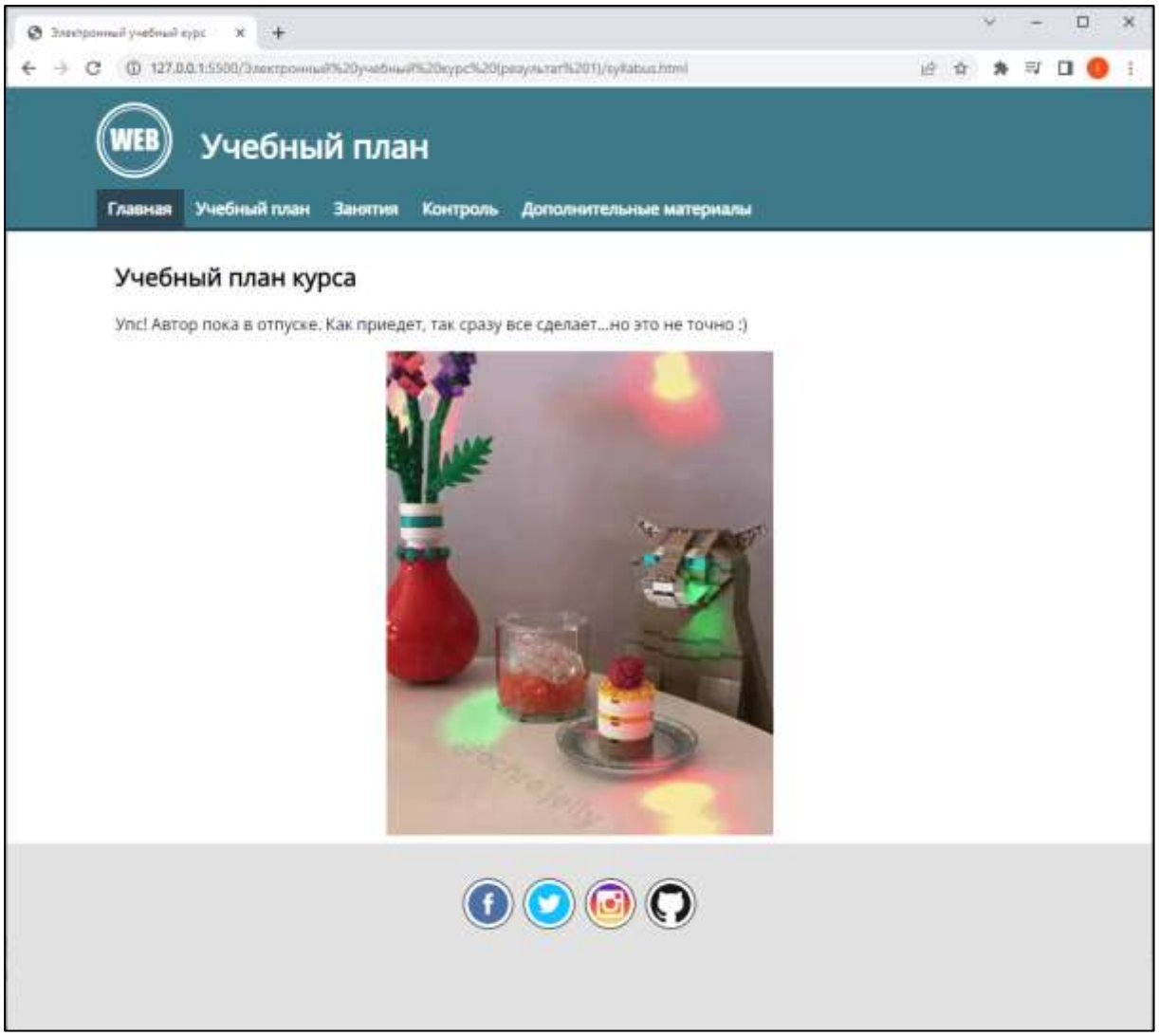


Рис. 3.179. Образец выполнения задания 2

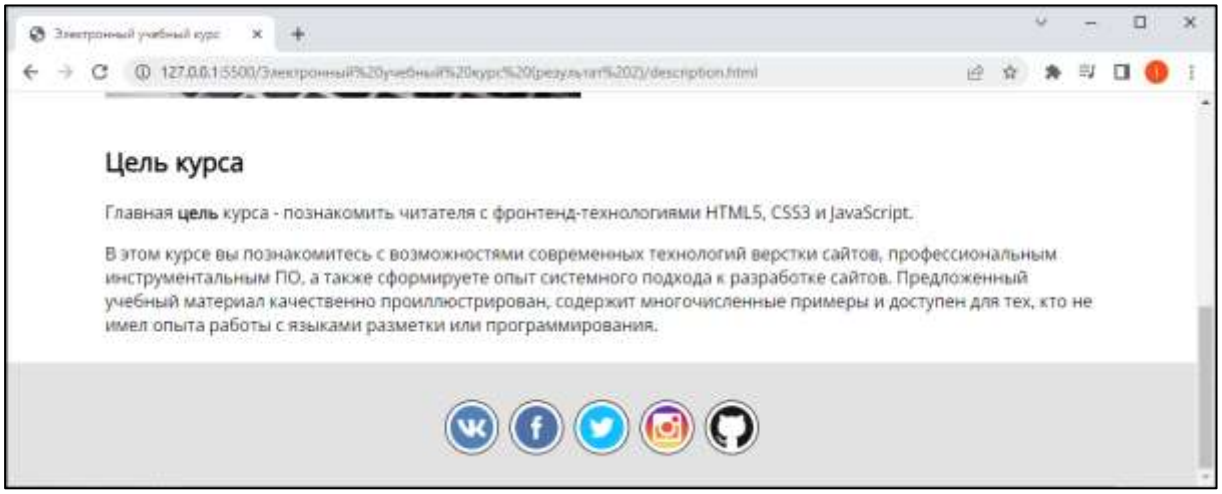


Рис. 3.180. Образец выполнения задания 3 (часть 1)

Рис. 3.181. Образец выполнения задания 3 (часть 2)

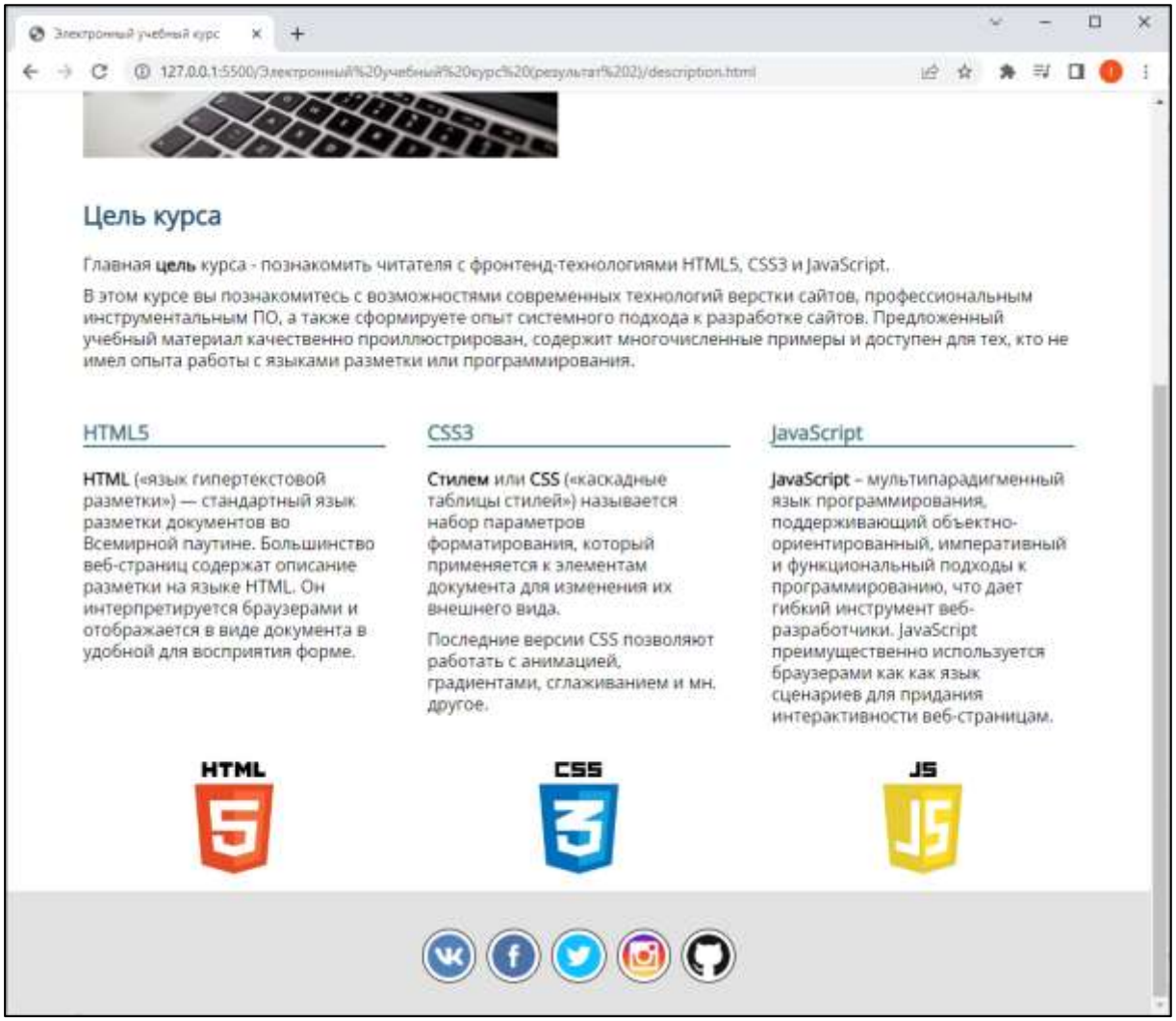


Рис. 3.182. Образец выполнения задания 3 (часть 3)

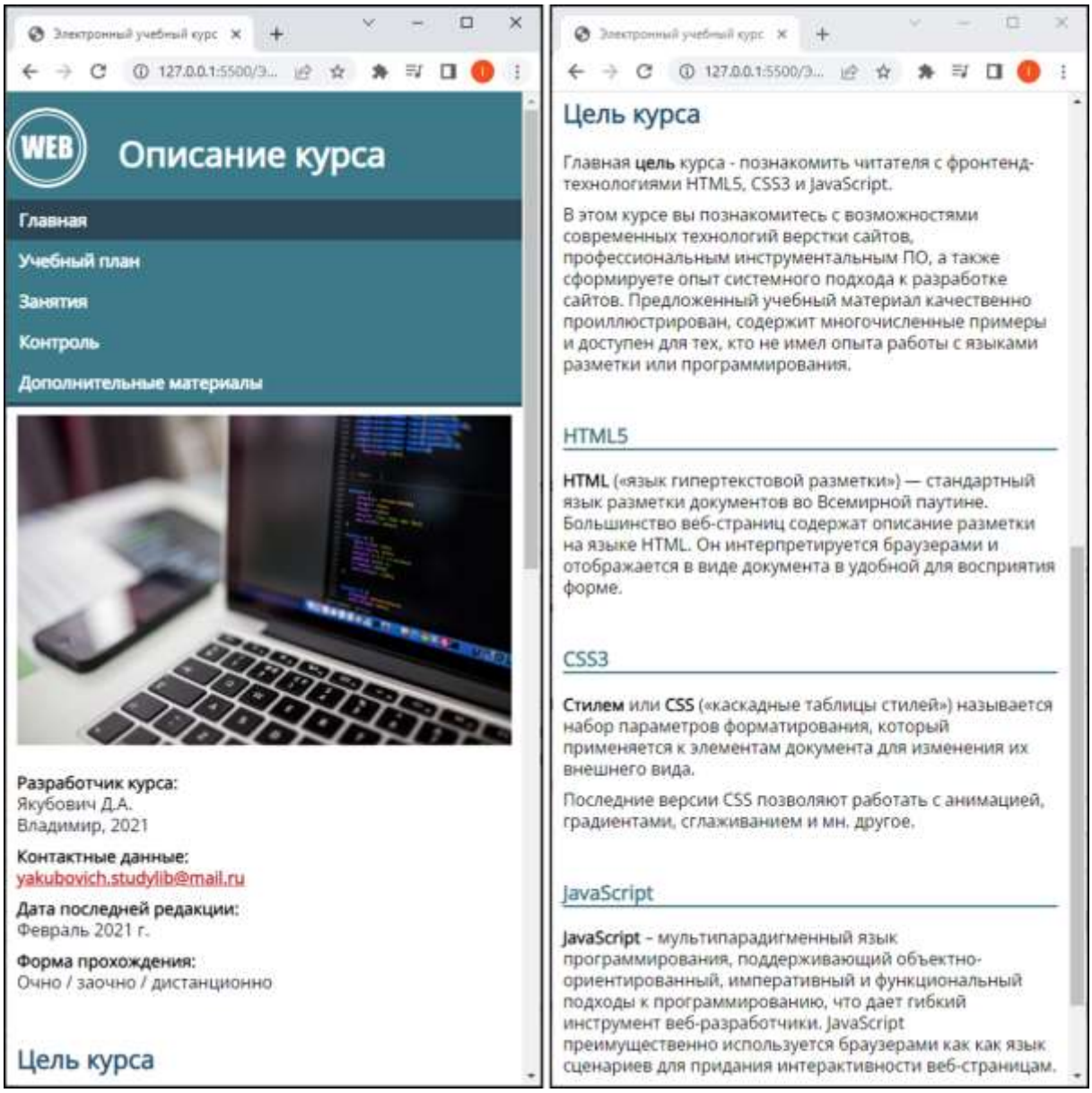
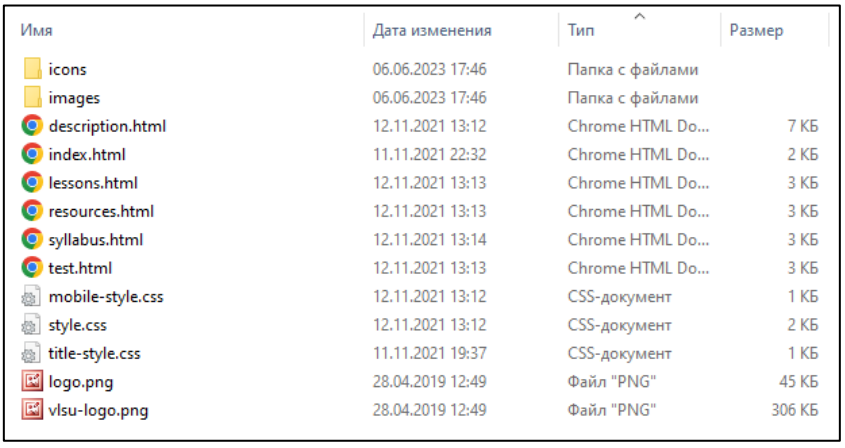


Рис. 3.183. Образец выполнения задания 4



Имя	Дата изменения	Тип	Размер
icons	06.06.2023 17:46	Папка с файлами	
images	06.06.2023 17:46	Папка с файлами	
description.html	12.11.2021 13:12	Chrome HTML Do...	7 КБ
index.html	11.11.2021 22:32	Chrome HTML Do...	2 КБ
lessons.html	12.11.2021 13:13	Chrome HTML Do...	3 КБ
resources.html	12.11.2021 13:13	Chrome HTML Do...	3 КБ
syllabus.html	12.11.2021 13:14	Chrome HTML Do...	3 КБ
test.html	12.11.2021 13:13	Chrome HTML Do...	3 КБ
mobile-style.css	12.11.2021 13:12	CSS-документ	1 КБ
style.css	12.11.2021 13:12	CSS-документ	2 КБ
title-style.css	11.11.2021 19:37	CSS-документ	1 КБ
logo.png	28.04.2019 12:49	Файл "PNG"	45 КБ
vlsu-logo.png	28.04.2019 12:49	Файл "PNG"	306 КБ

Рис. 3.184. Образец выполнения задания 5

ГЛАВА 4.
ЯЗЫК ПРОГРАММИРОВАНИЯ JAVASCRIPT

4.1. Особенности и возможности JavaScript

4.1.1. Понятие и краткая история JavaScript

Понятие JavaScript

Определение

JavaScript *– это скриптовый язык программирования, поддерживающий объектно-ориентированный, императивный и функциональный стили программирования.*

Язык JavaScript создавался качестве скриптового языка, который позволял управлять динамикой изменения внешнего вида и данных веб-страниц на стороне клиента (браузера). Скрипты на JavaScript встраиваются в HTML-файлы или подключаются в качестве внешних файлов (как CSS) и выполняются автоматически при загрузке веб-страницы.

Для выполнения скриптов на JavaScript не требуются каких-либо преобразований или компиляций. За обработку скриптов отвечает *движок* браузера. Он имеет весьма сложную реализацию. Упрощенный порядок работы движка следующий:

1. движок анализирует («парсит») код скрипта;
2. далее осуществляется транслирование скрипта в язык машинных инструкций;
3. машинный код запускается и выполняется.

На каждом этапе работы движка осуществляется оптимизация. Поэтому современные версии браузеров и движков позволяют выполняться скриптам достаточно быстро.

Предыстория

Язык JavaScript был разработан Бренданом Эйхом, Марком Андрессеном и Биллом Джойем в первой половине 90-х годов. Разработчики ставили цель создать «язык склеивания» для компонентов веб-ресурса: изображений, плагинов, Java-апплетов. Язык должен был стать простым и удобным инструментом для веб-дизайнеров и разработчиков.

Первоначально язык получил название Mocha («Мока»). Позже его переименовали в LiveScript, который был предназначен для программирования на стороне клиента и сервера.

Официальное название JavaScript получает 4.12.1995, его выход был приурочен к выходу браузера Netscape Navigator 2.

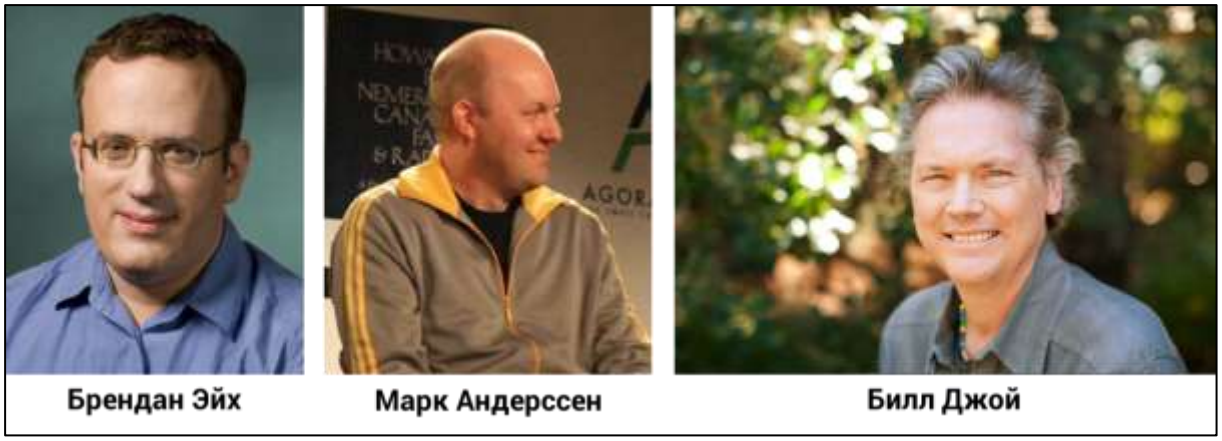


Рис. 4.1. Создатели JavaScript

Это полезно знать!

«Java» в названии языка JavaScript является данью популярному в то время (1994-1995 гг.) языка Java. Кроме того, базовый синтаксис JavaScript зачастую во многом аналогичен синтаксису Java, а также их предшественникам C / C++ и молодому C#.

Возможности языка

Современный JavaScript представляет собой гибкий и безопасный язык программирования. Возможности языка определяются окружением, в котором его используют. Так основной областью работы языка является браузер, где скрипты способны манипулировать структурой, данными и оформлением веб-страниц, а также управлять некоторыми параметрами браузера. Технология Node.JS расширяет возможности JavaScript за границы браузера и позволяет работать с файлами на жестком диске, выполнять сетевые запросы и т.д.

Среди основных возможностей JavaScript отметим:

- управление HTML-структурой и CSS-стилями;
- обработка действий пользователя;
- отправка сетевых запросов (использование технологий AJAX и COMET);
- управлять cookie браузера.

С другой стороны, возможности JavaScript ограничены политикой безопасности передачи данных. Поэтому язык не позволяет:

- читать или записывать файлы на жёсткий диск;
- запускать приложения и процессы операционной системы;
- работать с веб-камерой или микрофоном без явного разрешения пользователя;
- беспрепятственно передавать данные из разных окон или вкладок.

4.1.2. ECMAScript

ECMAScript как стандарт синтаксиса

Определение

ECMAScript *– это встраиваемый расширяемый язык программирования, не имеющий средств ввода/вывода данных и используемый в качестве базы для построения других скриптовых языков.*

ECMAScript – это стандарт, которому подчиняется JavaScript, а также ряд других скриптовых языков. Стандарт ECMAScript определяет общие правила для следующих компонент:

- синтаксиса;
- типов данных;
- инструкций;
- ключевых и зарезервированных слов;
- операторов;
- объектов.

Версии спецификаций

Говоря о JavaScript, следует подразумевать, что его развитие напрямую связано с развитием спецификации ECMAScript.

- ECMAScript определяет основу синтаксиса JavaScript.
- ECMAScript постоянно обновляется, что расширяет возможности синтаксиса JavaScript.

Таблица 4.1. Версии ECMAScript и даты их появления

Выпуск	**Название**	**Дата публикации**
ES1	ES1	Июнь 1997
ES2	ES2	Июнь 1998
ES3	ES3	Июнь 1999
ES4	ES4	Заброшен
ES5	ES5	Декабрь 2009
ES5.1	ES5.1	Июнь 2011
ES6	ES2015	Июнь 2015
ES7	ES2016	Июнь 2016
ES8	ES2017	Июнь 2017
ES9	ES2018	Июнь 2018
ES10	ES2019	Июнь 2019
ES11	ES2020	Июнь 2020
ES12	ES2021	Июнь 2021

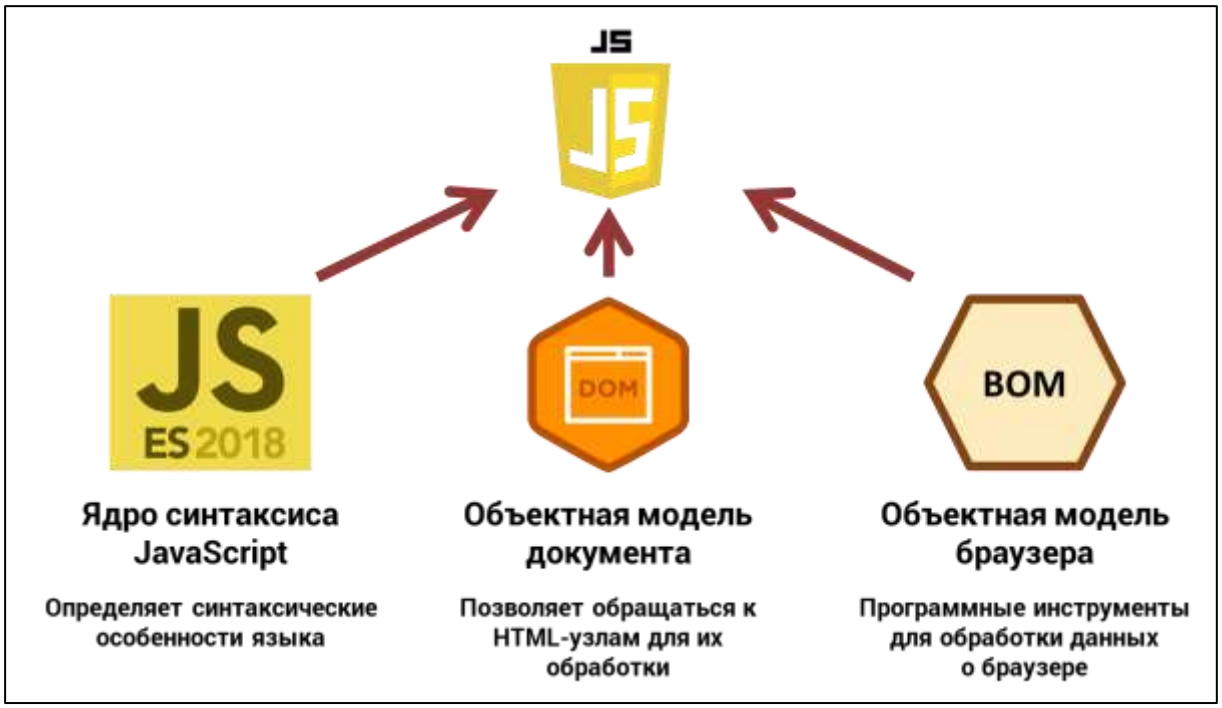


Рис. 4.2. JavaScript определяется не только стандартом ECMAScript, но и другими моделями

Важное замечание!

Дале ев курсе мы будем опираться на возможности синтаксиса ES6 и выше. Начиная с этой версии спецификации в синтаксис JavaScript начали вноситься возможности, свойственные ряду современных языков программирования.

Важное замечание!

В этой главе мы рассмотрим некоторые фундаментальные вопросы по синтаксису современного JavaScript. Возможности этого языка и связанные с ним модели DOM и BOM требуют немало времени на детальное изучение и могут быть изучены пользователем самостоятельно после завершения текущего курса.

4.1.3. Важные особенности JavaScript

1. Динамическая типизация

В JavaScript переменная связывается с типом в момент присваивания значения, а не в момент объявления переменной. Таким образом в различных участках программы одна и та же переменная может принимать значения разных типов.

```
let num = 2022;              // Сначала num - число
                             // (установлен тип Number)
num = "Привет, Денис!";      // Теперь num - строка
                             // (установлен тип String)
```

2. Слабая типизация

Компилятор движка JavaScript генерирует код, обеспечивающий преобразование типов, а логическая корректность такого преобразования контролируется программистом, т. е. фактически значение переменной одного типа можно присвоить значению переменной другого почти без ограничений.

```
let text = 2022 + " год";    // Приводится к строке
let num = 30 * "5";          // Приводится к числу
```

```
2022 год
150
```

Рис. 4.3. JavaScript неявно преобразует тип в зависимости от операндов

3. Автоматическое управление памятью

В JavaScript реализован специальный механизм управления оперативной памятью при выполнении скрипта – *сборщик мусора* (как, например, у платформы .NET и языка Java). Он позволяет периодически и автоматически очищать неиспользуемую ОЗУ от объектов.

4. Прототипное программирование

В JavaScript отсутствует понятие класса как типа данных (как в C++, C#, Java, Python и др.). Механизм наследования здесь реализован на основе клонирования существующего экземпляра объекта, называемого *прототипом*. Поэтому JavaScript вместо объектно-

ориентированного языка программирования часто называют *прототипно-ориентирваонным*.

Однако начиная с ES6 в JavaScript внедрили поддержку классов и их компонент в форме синтаксического сахара. При этом новый синтаксис просто скрывает работу на основе прототипов.

Это полезно знать!

При создании абстракций в JavaScript следует мыслить не классами, а объектами и связями между ними.

5. Функции как объекты первого класса

Важную роль в языке играют функции: их можно использовать без существенных ограничений.

1. Функции в ECMAScript являются объектами.
2. Функции могут храниться в переменных, объектах и массивах.
3. Функции можно передавать как аргументы в другие функции и возвращать как результат работы функции.
4. Функции могут иметь свойства.
5. Функции могут содержать описание функций.

Функции обычно определяют структуру JS-кода. В силу столь высокой значимости функций в скриптах и реализации их как объектов, функции также называют *объектами первого класса*.

4.1.4. Подключение скриптов

1. Внутренний скрипт

Для подключения *внутреннего* скрипта в HTML-документе используется парный тег `<script>`:

```
<script>
    // Здесь размещается код скрипта
</script>
```

При этом:

- контейнер можно подключать в разделе `<head>` и `<body>`.
- возможно многократное применение в разных местах.

**Глава 4\Подключение скриптов\
Внутренний скрипт 1\index.html**

```
<!DOCTYPE html>
<html lang="ru">

<head>
    <meta charset="UTF-8">
    <meta http-equiv="X-UA-Compatible" content="IE=edge">
    <meta name="viewport" content="width=device-width,
initial-scale=1.0">
    <title>Document</title>
    <script>
        alert("Тестирование скрипта")
    </script>
</head>

<body>

</body>

</html>
```

Одним из вариантов реализации внутреннего подключения является использование *атрибутов событий* HTML-элементов. Этот подход был популярен ранее, однако сейчас его следует избегать, поскольку он мало универсален и нагромождает разметку.

**Глава 4\Подключение скриптов\
Внутренний скрипт 2\index.html**

```
<!DOCTYPE html>
<html lang="ru">

<head>
    <meta charset="UTF-8">
    <meta http-equiv="X-UA-Compatible" content="IE=edge">
    <meta name="viewport" content="width=device-width,
initial-scale=1.0">
    <title>Document</title>
    <script>
        function getMessage() {
            alert("Тестирование скрипта")
        }
    </script>
</head>
```

```
<body>
    <p>Нажмите на кнопку для вывода сообщения.</p>
    <button onclick="getMessage()">Старт</button>
</body>

</html>
```

В данном примере кнопке `<button>` указан атрибут `onclick`, который при нажатии на кнопку вызовет функцию `getMessage()`.

2. Внешний скрипт

Внешнее подключение скриптов является наиболее универсальным и похоже на подключение внешних стилевых CSS-таблиц. Для этого код скрипта располагают в отдельном файле с расширением *.js*. В HTML-файле скрипт подключается инструкцией

```
<script src="code.js"></script>
```

или при вложении в каталоги:

```
<script src="путь/code.js"></script>
```

(название файла может быть любым другим).

Как и в случае внутреннего описания скриптов, для подключения внешних допускается:

- использовать их в разделе `<head>` и `<body>`;
- многократное подключение.

**Глава 4\Подключение скриптов\
Внешний скрипт\index.html**

```
<!DOCTYPE html>
<html lang="ru">

<head>
    <meta charset="UTF-8">
    <meta http-equiv="X-UA-Compatible" content="IE=edge">
    <meta name="viewport" content="width=device-width,
initial-scale=1.0">
    <title>Document</title>
</head>

<body>
    <!-- Здесь разметка документа -->
```

```
    <script src="code.js"></script>
</body>

</html>
```

**Глава 4\Подключение скриптов\
Внешний скрипт\code.js**

```
alert("Тестирование скрипта");
```

Это важно знать!

Обратите внимание, что в примере скрипт подключен внизу, после разметки документа. Это необходимо, чтобы скрипт дождался прорисовки элементов страницы и подключил ссылки к элементам, на которые будет ссылаться.

Это полезно знать!

Далее в примерах мы будем стараться использовать только внешнее подключение скриптов. Это позволяет отделить JS-код от HTML-разметки и сделать более эффективным управление кодом и его сопровождение.

4.1.5. Порядок выполнения скриптов

Стандартный порядок выполнения скриптов

Вне зависимости от способа подключения скрипта элемент `<script>` интерпретируются браузером в порядке следования в HTML-разметки. Такой порядок соответствует последовательному отображению содержимого тега при загрузке страницы.

Использование внешних скриптов более эффективно, поскольку браузер кеширует код скриптов и может его многократно использовать далее без необходимости полной перезагрузки страницы.

Синхронное и асинхронное выполнение

Все теги, находящиеся ниже `<script>`, загружаются в HTML-структуру только после полного выполнения скрипта. В этом случае говорят, что все скрипты работают в *синхронном режиме* по умолчанию.

Однако синхронное выполнение не всегда рационально. Например, скрипт может выполняться длительное время или в цикле, поэтому остальные скрипты будут заблокированы. В таком случае необходима отложенная обработка скриптов, или *асинхронный режим* выполнения скрипта.

В асинхронном режиме браузер не останавливает обработку страницы, а работает дальше. Когда скрипт будет загружен – он будет выполнен.

Для асинхронного выполнения используются атрибуты `async` и `defer`.

Атрибуты async и defer

При указании атрибута **`async`** скрипт выполняется полностью асинхронно: браузер не останавливает обработку страницы, а работает дальше. Когда скрипт будет загружен – он выполнится.

```
<script src="code.js" async></script>
```

При указании атрибута **`defer`** скрипт выполняется асинхронно, но, в отличие от `async`, браузер гарантирует, что относительный порядок скриптов с `defer` будет сохранён.

```
<script src="code.js" defer></script>
```

Оба атрибута поддерживается всеми браузерами, включая IE.

Важное замечание!

1. *Атрибуты **`async`** и **`defer`** работают только для внешних скриптов.*
2. *Атрибуты **`async`** и **`defer`** не имеет смысла указывать одновременно.*

4.1.6. Простой пример

Простое динамическое изменение разметки

Создадим каталог *Знакомство с JavaScript*. Внутри – новый HTML-файл со стандартным шаблоном и внутренним скриптом (для краткости приведем код только из раздела <body>):

Глава 4\Знакомство с JavaScript\index.html

```
<body>
    <h1>Знакомство с JavaScript</h1>
    <h2>Первый скрипт:</h2>

    <script>
        document.write("<p>Привет, JavaScript!</p>");
    </script>
</body>
```

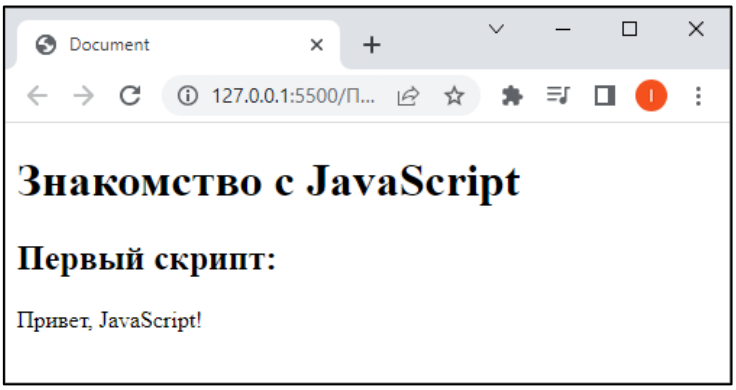


Рис. 4.4. Внедрение фрагмента разметки с помощью JavaScript

В документе может быть несколько скриптов. Добавим ниже еще один заголовок и внутренний скрипт:

Глава 4\Знакомство с JavaScript\index.html

```
<body>
    <h1>Знакомство с JavaScript</h1>
    <h2>Первый скрипт:</h2>

    <script>
        document.write("<p>Привет, JavaScript!</p>");
    </script>

    <h2>Второй скрипт:</h2>
```

```
    <script>
        let year = 2023;
        let author = "Денис";
        document.write("<p>Привет, " + author +
            "!<br>На дворе " + year + " год.</p>");
    </script>
</body>
```

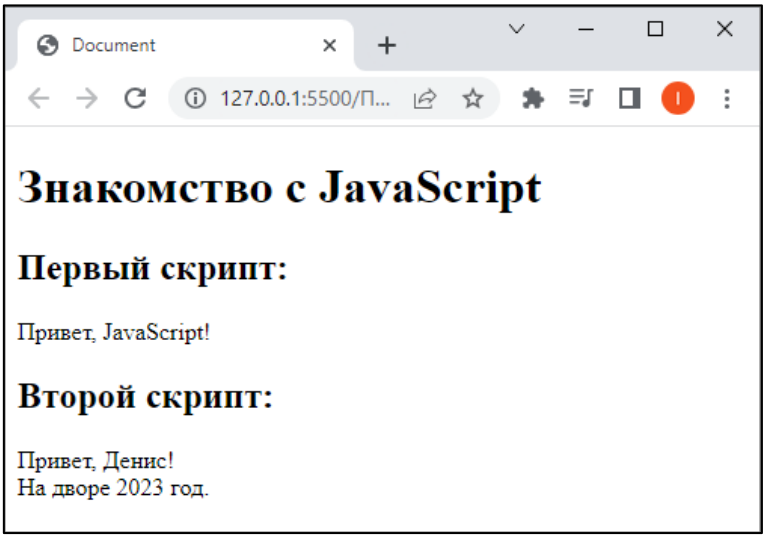


Рис. 4.5. Добавляем работу с переменными

Команда `write()` является функцией, описанной внутри объекта `document`. В скобках перечисляются параметры функции. Указанная функция динамически встраивает в текущее положение HTML-кода заданную разметку (в виде строки), распознавая среди нее и теги.

Параметр функции склеивается из разных фрагментов: значения переменных и фрагментов строковых литералов.

В конце каждой инструкции ставится «`;`». Это позволяет писать несколько команд в одну строку (но так лучше не делать).

Это полезно знать!

В современном JavaScript завершать инструкции оператором `;` больше необязательно. Однако мы продолжим его использовать, чтобы сохранить преемственность версий языка.

Разумеется, вызывать функцию `write()` можно многократно. Кроме того, мы вправе передать и другое строковое представление

разметки, которое преобразуется в HTML-структуру. Разобьем длинную запись на отдельные команды:

Глава 4\Знакомство с JavaScript\index.html

```
<body>
    <!-- Код выше -->

    <script>
        let year = 2023;
        let author = "Денис";
        document.write("<p>Привет, " + author + "!</p>");
        document.write("<p>На дворе " + year + "
            год.</p>");
    </script>
</body>
```

Важное замечание!

JavaScript, как и CSS, чувствителен к регистру букв!

Ожидание прогрузки разметки

Ранее отмечалось, что по умолчанию скрипты выполняются синхронно. Зачастую важно быть уверенным, что скрипты начнут работать лишь после того, как будут загружены элементы страницы. Иначе может возникнуть ситуация, когда скрипт пытается обратиться к элементу, который еще не «прорисован» и ничего не произойдет.

1. Способ

Наиболее популярный подход в этом случае (мы также его будем использовать как основной) – разместить скрипт(ы) в самом конце тела документа (рис. 4.6).

2. Способ

Другой подход заключается в обработке JS-событий. Один из приемов – использовать событие **`onload`** для глобального объекта **`window`**. Тело заданной функции будет стартовой точкой для остального кода.

Теперь скрипт можно подключать в любом месте. В частности – в преамбуле веб-документа (рис. 4.7).

Hello.html (внешний скрипт)

```
<!DOCTYPE html>
<html lang="ru">

<head>
    <meta charset="UTF-8">
    <meta http-equiv="X-UA-Compatible" content="IE=edge">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Document</title>
</head>

<body>
    <h1>Hello, JS</h1>

    <script src="code.js"></script>
</body>

</html>
```

code.js

```
document.write("Привет, JavaScript!");
```

Скрипт размещается в конце, чтобы гарантировать построение разметки выше

Рис. 4.6. Ожидание загрузки разметки: первый подход

Hello.html (внешний скрипт)

```
<!DOCTYPE html>
<html lang="ru">

<head>
    <meta charset="UTF-8">
    <meta http-equiv="X-UA-Compatible" content="IE=edge">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Document</title>
    <script src="code.js"></script>
</head>

<body>
    <h1>Hello, JS</h1>

</body>

</html>
```

code.js

```
window.onload = function() {
    document.write("Привет, JavaScript!");
}
```

Если скрипт размещен в преамбуле, то необходимо дождаться загрузки разметки страницы.
Один из приемов – использовать событие onload для глобального объекта window.

Рис. 4.7. Ожидание загрузки разметки: второй подход

Вопросы для самопроверки

1. Какими возможностями обладает JavaScript?
2. Что представляет собой ECMAScript и каким образом он связан с JavaScript?
3. Перечислите важнейшие особенности JavaScript.
4. Какими способами можно подключить JS-скрипт к HTML-разметке?
5. В чем заключаются преимущества внешних скриптов?
6. Что нужно учитывать в порядке выполнения скриптов и чем отличается синхронный режим выполнения кода от асинхронного?

Практикум

Задание 1

1. Создайте каталог *Первый проект по JS*.
2. Воспроизведите разобранный в 4.1.6 пример разметки и скрипта (рис. 4.9).
3. Измените реализацию рассмотренного ранее примера, подключив скрипты как внешние JS-файлы (т.е. каждый скрипт вынести в отдельный файл и подключить в порядке следования HTML-разметки документа).

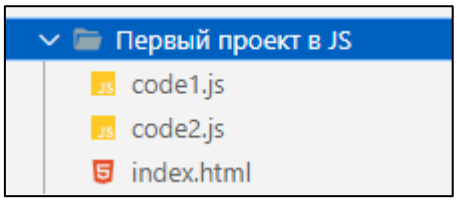


Рис. 4.8. Структура проекта

Задание 2

1. Создайте каталог *Функция write*, добавьте HTML-файл с типовой начальной разметкой.
2. Создайте CSS-файл с приведенным в конце задания кодом.
3. Подключите внешний скрипт.
4. С помощью команды `document.write()` сгенерируйте документ с заголовком и абзацами, как изображено на

рис. 4.10. При этом вашу имя и фамилию задать в отдельных переменных.

5. Замечание. В текстовой строке также можно указывать и атрибуты тегов, например – классы:

```
document.write("<p class='info'>Текст</p>");
```

Это позволит оформить вставляемый скриптом в разметку элемент.

Глава 4\Функция write\style.css

```
body { font: 16px Tahoma, Arial; }

h1 { color: #0f534f; }

.info {
    font-size: 125%;
    color: #292cbd;
}
```

Задание 3

1. Создайте каталог *Приветственное сообщение*, добавьте HTML-файл с типовой начальной разметкой
2. Подключите стилевой файл и внешний файл JS-скрипта.
3. С помощью `document.write()` сформировать страницу, как изображено на рис. 4.11.
4. Замечание: в разметке используется тег заголовка `<h1>`, тег абзаца `<p>` и тег акцентиования внимания `<strong>`. Оформление задается в CSS.
5. Функцию `write()` можно вызвать несколько раз, чтобы не работать с длинным параметром.

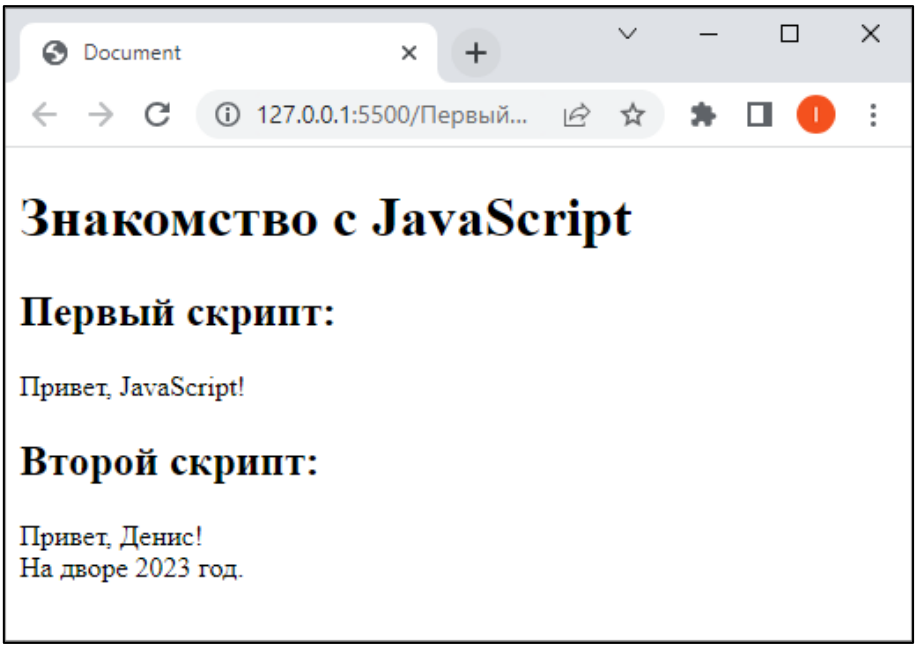


Рис. 4.9. Образец выполнения задания 1

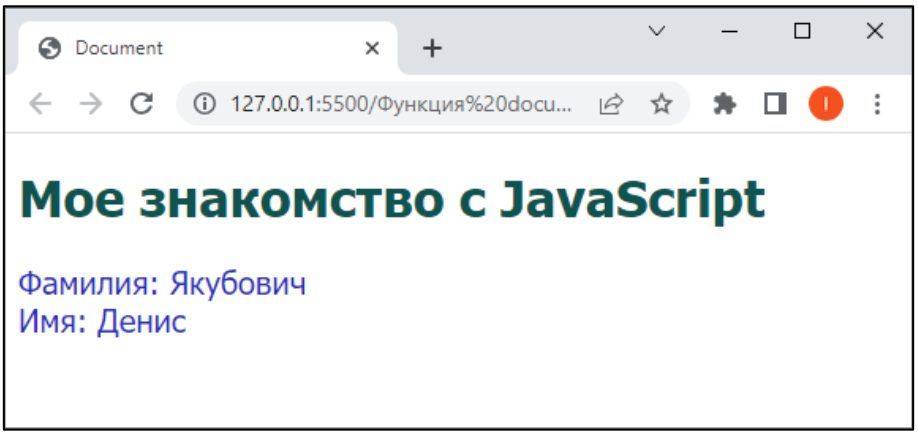


Рис. 4.10. Образец выполнения задания 2

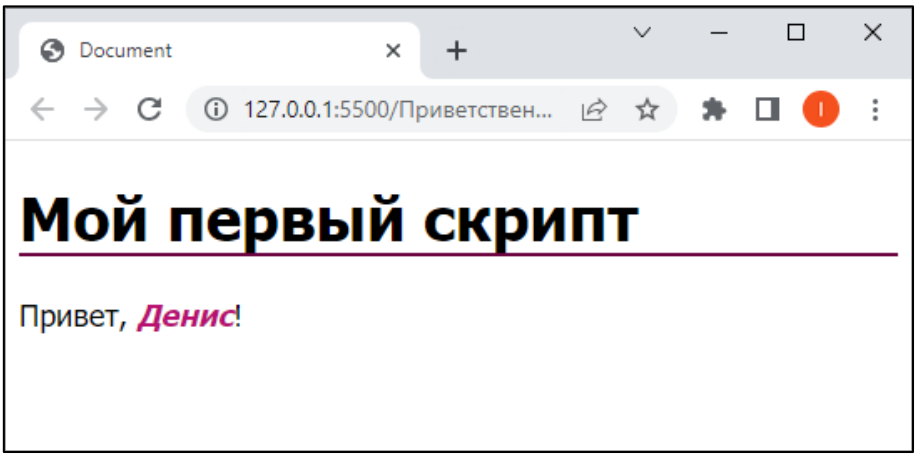


Рис. 4.11. Образец выполнения задания 3

4.2. Описание переменных и типизация данных в JavaScript

4.2.1. Комментарии

Комментарии в коде

Определение

Комментарии *– фрагменты кода, содержащие текст пояснения, который не влияет на работу скрипта и не отображается в качестве результата.*

JavaScript поддерживает два типа комментариев.

1. ***Однострочный***

Действует в рамках одной строки, начиная с инструкции **//**. Удобен для пояснения работы команд.

```
// Версия скрипта: v1.2
document.write(2022);    // Вывод числа
```

2. ***Многострочный***

Текст комментария задается внутри «скобок» **/* */** и может включать более чем одну строку. Удобен, если требуется закомментировать фрагмент кода скрипта (т.е. временно его «заморозить»).

```
/*
let a = 5;
let b = 6;
let sum = a + b;
*/
```

Приемы работы с Visual Studio Code

При комментировании фрагмента кода не забывайте про комбинацию клавиш ***Ctrl*** *+ /.*

Строгий режим

Определение

Строгий режим *– это режим работы скрипта в соответствии с современными стандартами ECMAScript, при котором генерируются ошибки для небезопасных действий.*

Для переключения браузера в строгий режим в начале скрипта указывается директива

```
"use strict";
```

Строгий режим позволяет писать более безопасный код. Возможные недочеты, которые в обычном режиме будут проигнорированы движком, в строгом режиме распознаются уже в качестве ошибок.

Например, в строгом режиме нельзя присвоить значение переменной, которая не описана:

```
"use strict";
num = 40.3;       // Ошибка! Переменная не описана
```

4.2.2. Переменные

Переменные

Определение

Переменная *– временно используемая часть ОЗУ, хранящая данные приложения. Для JavaScript переменная может хранить как значение (например, числовые), так и ссылку на него (например, на объект).*

Переменные – это основной компонент любой программы. Упрощенно переменную можно понимать, как коробку, в которую помещаются данные (число, текстовая строка, логическое значение или объект).

В дальнейшем значение переменной может быть изменено в ходе операций.

JavaScript – динамический слабо типизированный язык. Это позволяет записывать в переменную значения разного типа.

Определение

Для объявления переменной используется команда ***let****.*
Важно: работает, начиная со спецификации ES6.

```
let имя_переменной;      // Переменная только описана
```

Операция присваивания (инициализации) задается оператором «=».

```
// Переменная описана и инициализирована значением
let имя_переменной = значение;
```

Рассмотрим и прокомментируем некоторые примеры описания и инициализации переменных:

Глава 4\Переменные\Команда let\code.js

```
let counter;
counter = 20;

// let counter = 30;

counter = 30;

let num = 10.78;
let text = "Строка";
let fileIsOpen = true;

let student = {
    id: 4512,
    fullname: {
        firstName: "Денис",
        lastName: "Якубович",
    },
    age: 20,
    group: "МИ-555"
};
```

В начале описана переменная `counter`. Далее ей присвоено значение. Повторное объявление переменой `counter` с помощью `let` является ошибкой: к ней можно обращаться для чтения или записи нового значения уже по ее имени.

Следом описаны переменные `num`, `text` и `fileIsOpen`, которые инициализированы числовым, строковым и логическим значениями соответственно.

Наконец, создана переменная `student`, являющаяся *объектом*. Объекты сочетают в себе переменные разного типа как характеристики единой сущности. У объекта определены 4 *свойства*, среди которых `fullname` – составное (само является объектом из двух свойств). Инициализация свойств объекта осуществляется с помощью записи в формате `ключ:значение`.

Это важно знать!

Переменная, которая только описана, но не инициализирована значением, имеет специальное значение ***undefined*** *(«не определено»). Рекомендуется описывать переменные и сразу же их инициализировать.*

Константы

Определение

Константы задаются командой ***const****.*
Важно: работает, начиная со спецификации ES6.
Константа обязательно должна быть инициализирована сразу при описании.

```
const имя_константы = значение;
```

Константа не может менять значение.

В JavaScript понятие константы имеет некоторую специфику работы относительно объектов. Рассмотрим и прокомментируем далее следующий пример:

Глава 4\Переменные\Команда const\code.js

```
const PI = 3.14159;     // Постоянная Пи
PI = 3.14;

const avto = {
    model: "LADA Vesta Sport",
    color: "красный",
    year: 2021,
    cost: 1597600
}

avto = {
    model: "LADA XRAY Cross"
}

avto.color = "черный";
```

В начале определена константа `PI`. Далее была попытка поменять значение константы на другое, однако возникает ошибка: это невозможно по определению. Для переменных «простого» типа понятие константы справедливо в полном смысле этого слова.

Далее описан объект `avto` в качестве константы. Попытка заменить объект на другой приводит к ошибке.

Однако менять значения свойств const-объекта допускается. Это связано с тем, что переменные в JavaScript хранят не сами значения объектов, а *ссылки* на объекты, записанные в ОЗУ. Для const-объекта ссылка должна быть постоянной, а вот его компоненты уже могут меняться.

Это полезно знать!

Если переменная не будет меняться, предпочтительнее описывать ее константой.

Старый синтаксис

Определение

Для объявления переменной вместо ***let*** *можно использовать* ***var****. Устаревший способ, имеющий ряд недостатков в современном программировании.*

```
var имя_переменной;      // Переменная только описана

var имя_переменной = значение;  // Переменная описана и
                                // ей задано значение
```

Приведем несколько примеров использования var:

Глава 4\Переменные\Команда let\index.html

```
<h1>Описание и инициализация переменных</h1>
<script src="code.js"></script>
```

Глава 4\Переменные\Команда let\code.js

```
var currentYear = 2022;

var student = {
    fullName: {
        name: "Денис",
        surname: "Якубович",
        patronymic: "Андреевич"
    },
    age: 20,
    group: "МИ-555"
};

document.write("Фамилия: " + student.fullName.surname +
"<br>");
document.write("Группа: " + student.group);
```

Здесь также показано, каким образом идет обращение к свойствам объекта: для этого используется оператор «.», который согласно ООП-нотации отделяет объект от его атрибута:

```
student.group
```

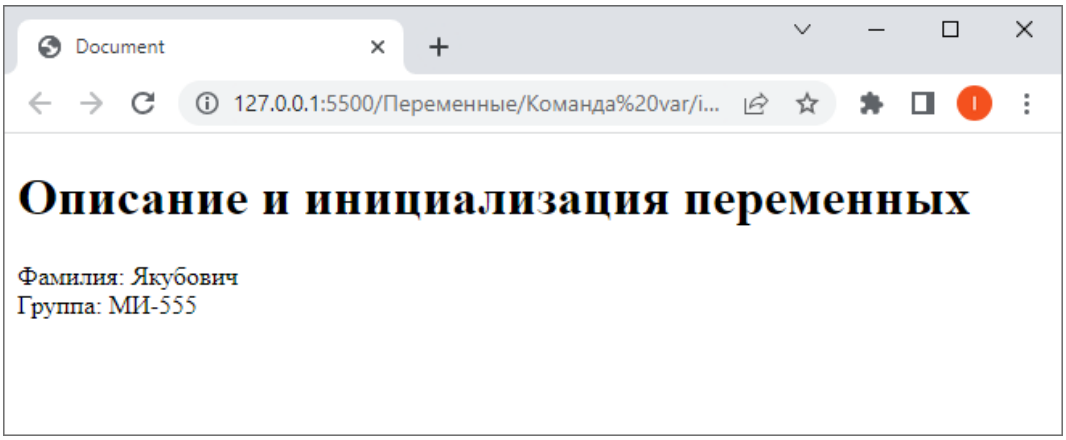


Рис. 4.12. Использование var в описании переменной

Между `let` и `var` существует весьма значимая разница, которая связана с областью видимости определяемой переменной. Если сформулировать упрощенно, то:

- для **`var`** область видимости переменной определяется телом функции;
- для **`let`** область видимости определяется областью блока, в котором она определена (т.е. внутри **`{ }`**).

Иными словами, `var` в некотором смысле задает более глобальную область видимости переменной, что зачастую может привести к нежелательным побочным эффектам.

В дальнейшем мы будем использовать `let` и `const`. Для команды `var` есть ряд ситуаций, когда ее аккуратное использование может быть полезно. Однако в настоящее время чаще всего ее присутствие можно найти в реализации скриптов старых сайтов.

Подробнее примеры работы с `let`, `const` и `var` можно найти на сайте skillbox.ru.

Это полезно знать!

В новых стандартах ECMAScript рекомендуется использовать команду ***`let`***.

Некоторые примеры описания переменных

Приведем еще ряд примеров описания переменных.
Хорошей практикой является описывать переменные отдельно:

```
let x = 10;
let y = 20;
let error = 0.001;
```

Допускается описывать нескольких переменных через запятую, в том числе их инициализировать:

```
let coeffA, coeffB, coeffC;
let x = 10, y = 20, error = 0.001;
```

Если переменные связаны по смыслу, то часто используют следующую форму группировки описания (так удобнее комментировать каждую при необходимости):

```
let x = 10,
    y = 20,
    error = 0.001;
```

Переменной в любой момент можно присвоить значение другого «типа», однако так делать не рекомендуется: лучше описать новую переменную. В следующем примере имя переменной после переприсваивания противоречит ее содержимому (хотя это и не ошибка):

```
// Динамическая типизация
let number = 23;

// Здесь какие-либо операции

number = "Число: " + number;        // "Число: 23"
```

JavaScript имеет аналоги массивов. При этом с т.з. языка массивы тоже являются объектами:

```
let testArray = [34, 4, -45, 0, 12, 4];
```

Опишем объект `user` с тремя свойствами (логин, пароль, статус):

```
let user = {
    login: "yakubovich.a.d",
    password: "123456",
    status: "admin"
}
```

4.2.3. Правила именования переменных

Используемые нотации именования

В JavaScript действуют ряд технических ограничений на имена переменных и констант:

- название переменной должно содержать только буквы, цифры или символы `$` и `_`;
- первый символ не должен быть цифрой;
- нельзя использовать зарезервированные слова (команды).

Это полезно знать!

Имя переменной должно коротко, полно и точно описывать сущность, которую она представляет.

В JavaScript разработчики часто используют одно из двух соглашений по стилю оформления имени переменных:

- *camelCase* («верблюжья» нотация);
- *snake_case* («змеиная» нотация).

Рис. 4.13. Нотации именования переменных

Рис. 4.14. Именование констант

Некоторые рекомендации по именованию переменных

Правило 1

Формулируйте суть переменной в словах.

```
// Что за значение хранится в переменной x?
const x = 5299.00;

// Теперь очевидно – это стоимость товара.
const goodsCost = 5299.00;
```

Правило 2

Оптимальным именем переменной часто оказывается само это высказывание.

```
// Нельзя однозначно предположить, для какой цели
// используется эта строка
const my_name = "xakker_2022";

// Теперь понятно – это логин!
const login = "xakker_2022";

// Тоже допустимо, хотя это может подразумевать,
// к примеру – название компьютера
const username = "xakker_2022";
```

Правило 3

Хорошее мнемоническое имя чаще всего описывает проблему, а не ее решение, т.е. выражает ЧТО, а не КАК.

```
// «Подсчет яблок» – это процесс.
// Но переменная должна характеризоваться в роли объекта
let to_count_apples = 0;
```

```
// «Счетчик яблок» – так более информативно.
let applesCounter = 0;
```

Правило 4

Имя переменной не должно быть слишком коротким, либо слишком длинным.

```
// Слишком короткое имя не информативно,
// а слишком длинное – ужасно!
const com = ["if", "switch", "for", "while", "do"] ;

const array_of_key_words_in_codding_listings = [
    "if", "switch", "for", "while", "do"
];

// Теперь понятно, что это массив ключевых слов (команд)
const commands = ["if", "switch", "for", "while", "do"] ;
```

Правило 5

Используйте в названии слова спецификаторы.

```
// Не понятно, о какой температуре речь
const Temperature = 22.3;

// init (от «initial») – начальная температура;
// max (от «maximum») – максимальная температура;
// fin (от «final») – конечная температура
const InitTemperature = 22.3;
const MaxTemperature = 37.8;
const FinTemperature = 24.1;
```

Правило 6

Используйте слова антонимы.

```
// Ответ «да» и «отрицательно» плохо согласуются
const answer = { Yes, Negative };

// Ответы «да» и «нет» взаимно противоположны
const answer = { Yes, No };
```

Правило 7

Называйте переменные прописными буквами, а константы – заглавными.

```
// Некорректно
let Radius = 2.5;
const pi = 3.14159;
```

```
// Корректно
let radius = 2.5;
const PI = 3.14159;
```

Правило 8

Записывайте «магические числа» в константы с понятным именем.

```
// Вероятно, только по названию переменной можно
// догадаться, что считается число секунд в сутках
const DaySeconds = 24 * 60 * 60;

// Ввели три вспомогательные константы, спрятав числа под
// текстовую «обертку». Теперь вычисление обретает
// очевидный смысл
const SECONDS = 60;
const MINUTES = 60;
const HOURS = 24;

const DaySeconds = HOURS * MINUTES * SECONDS;
```

Правило 9. Логические переменные

- Присваивайте логическим переменным имена, подразумевающие значение **`true`** или **`false`**.
- Используйте утвердительные имена логических переменных.

Примеры удачных имен:

- `done` – сделано, завершено;
- `error` – ошибка, некорректно;
- `found` – найдено;
- `success` / `ok` – успешно, хорошо.

Примеры неудачных имен:

- `status` – статус (какой: да / нет, хорошо / плохо или другое?);
- `notEmpty` – не пустой (истина на отрицании, можно будет запутаться, если взять отрицание `!notEmpty` (не верно, что не пустой, т.е. пустой…)).

Общие рекомендации по именованию переменных

1. Избегайте обманчивых имен или аббревиатур.
2. Избегайте имен, имеющих похожие значения.
3. Избегайте переменных, имеющих разную суть, но похожие имена.
4. Избегайте имен, включающих цифры.
5. Избегайте орфографических ошибок в названиях.
6. Избегайте слов, при написании которых люди часто допускают ошибки.
7. Избегайте смешения разных языков в названии.
8. Избегайте имен, содержащих символы, которые можно спутать с другими символами («i» с «j», «1» с «l» и т.п.).

4.2.4. Типы данных

Особенность типизации данных в JavaScript

Читатель наверняка уже заметил, что при описании переменной в JavaScript нигде не указывается ее тип. Каким же образом тогда JavaScript «понимает», какие операции допустимы для переменной?

На самом деле JavaScript четко различает восемь типов данных. В большинстве случаев движок сам распознает тип переменной и корректно в неявной форме преобразует его в разных операциях.

Основные типы данных

JavaScript содержит 8 основных типов данных:

- *Неопределенный* (`Undefined`) – значение не определено (не задано).
- *Пустой* (`Null`) – значение определено как пустое (пустая ссылка на объект).
- *Числовой* (`Number`) – значение является числом.
- *Большой числовой* (`BigInt`) – значение целого числа любого размера.
- *Логический* (`Boolean`) – значение имеет логический тип;
- *Строковый* (`String`) – значение является строкой.

- *Символьный* (**Symbol**) – значение уникального идентификатора в объектах.
- *Объект* (**Object**) – значение является объектом.

Определение

Чтобы определить тип переменной в операциях, можно использовать оператор ***typeof****. Он возвращает строковое название типа.*

Например:

```
let n = 10.34;
let text = "Этот текст";

document.write(typeof n);      // number
document.write(typeof text);   // string
```

Подробнее рассмотрим каждый тип данных. Отдельно подключим файлы со скриптами в одном HTML-документе:

Глава 4\Типы данных\index.html

```
<body>
    <script src="js/number.js"></script>
    <script src="js/bignumber.js"></script>
    <script src="js/boolean.js"></script>
    <script src="js/string.js"></script>
    <script src="js/undefined.js"></script>
    <script src="js/null.js"></script>
    <script src="js/object.js"></script>
    <script src="js/symbol.js"></script>
</body>
```

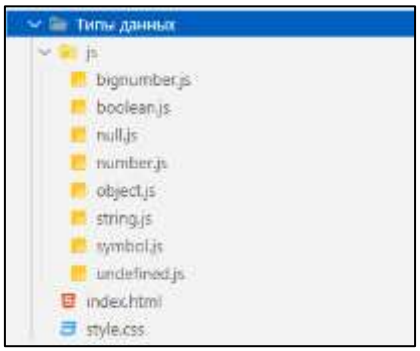


Рис. 4.15. Структура файлов в примере

Типы данных

1. Тип Number

Числовой тип данных способен описывать как целочисленные значения, так и числа с плавающей точкой.

Дополнительные возможности представлены в глобальном объекте **Number**.

Глава 4\Типы данных\js\number.js

```
// 1. Числовой тип
const coeff = 14;          // Десятичное число
const float = 46.01;       // Десятичное число
const hex = 0x0000ff;      // Шестнадцатеричное FF = 255
const oct = 0o0022;        // Восьмеричное 22 = 18
const exp1 = 1.45e4;       // 1.45 * 10^4 = 14500
const exp2 = 1.45e-4;      // 1.45 * 10^(-4) = 0.000145

// Объект Number, его свойства и функции
const num = 24.5;

// Предельные значения и точность
const MIN = Number.MIN_VALUE;
const MAX = Number.MAX_VALUE;
const EPS = Number.EPSILON;

// Бесконечность
const NEG_INF = Number.NEGATIVE_INFINITY;
const POS_INF = Number.POSITIVE_INFINITY;

document.write("<h1>Числовой тип</h1>");
document.write("<p>Тип: " + typeof(num) + "</p>");
document.write("<p>Min = " + MIN + "</p>");
document.write("<p>Max = " + MAX + "</p>");
document.write("<p>Epsilon = " + EPS + "</p>");
document.write("<p>-∞ = " + NEG_INF + "</p>");
document.write("<p>+∞ = " + POS_INF + "</p>");
document.write("<p>-∞ = " + -1 / 0 + "</p>");
document.write("<p>+∞ = " + 1 / 0 + "</p>");
document.write("<p>Не число: " + 0 / 0 + "</p>");

document.write("<p>Не число: " + Number.isNaN(num) +
"</p>");
document.write("<p>Конечное: " + Number.isFinite(num) +
"</p>");
```

```
document.write("<p>Целое: " + Number.isInteger(num) +
"</p>");
```

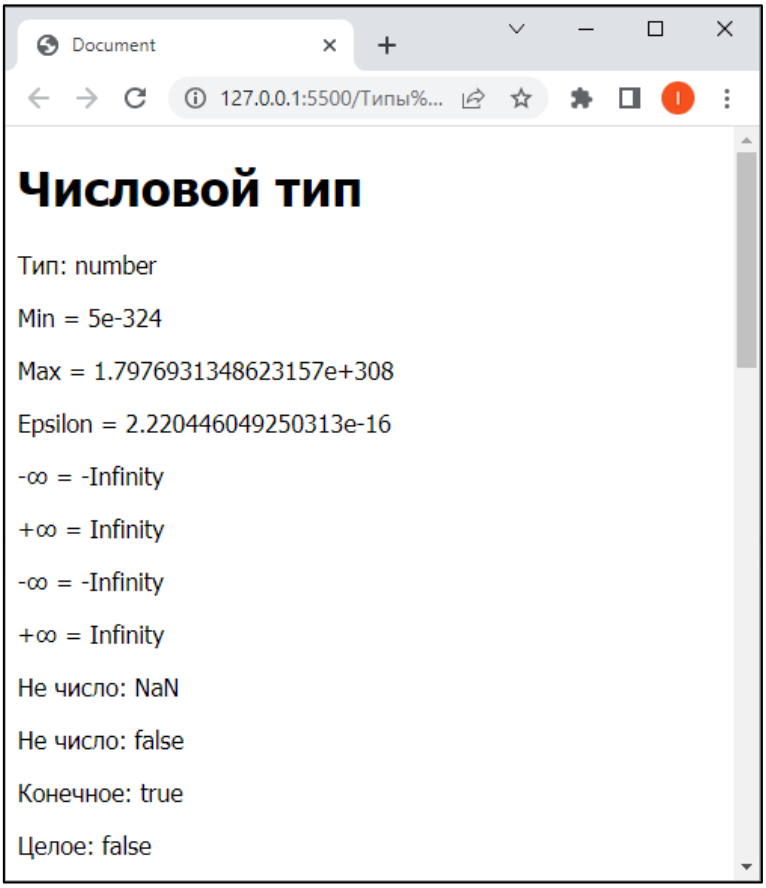


Рис. 4.16. Работа с числовыми переменными

2. Большой числовой

Предназначен для работы с большими целыми числами, которым недостаточно стандартного диапазона `Number`. В конце числа приписывается символ «n».

Дополнительные возможности представлены в глобальном объекте **`BigNumber`**.

`Глава 4\Типы данных\js\bignumber.js`

```
// 2. Целочисленный неограниченный тип
const bigNumber = 12345678912345678912345678 9n;

document.write("<h1>Большое целое</h1>");
document.write("<p>Тип: " + typeof(bigNumber) + "</p>");
document.write("<p>Большое целое: " + bigNumber +
"</p>");
```

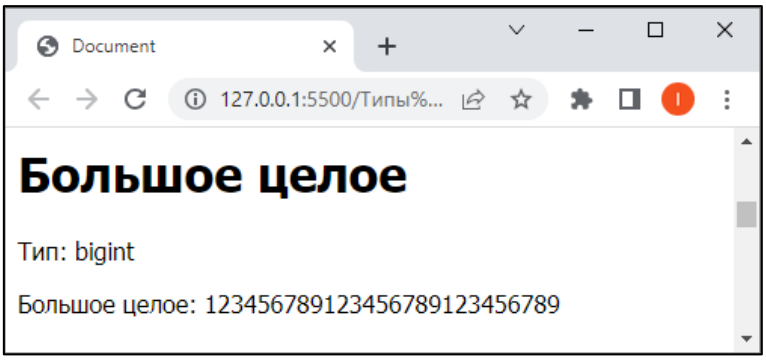


Рис. 4.17. Работа с большими целыми числами

3. *Тип Boolean*

Логически тип может принимать одно из двух значений: **`true`** (истина) и **`false`** (ложь).

Дополнительные возможности представлены в глобальном объекте **`Boolean`**.

Глава 4\Типы данных\js\boolean.js

```
// 3. Логический тип
const bool = true;

document.write("<h1>Логический тип</h1>");
document.write("<p>Тип: " + typeof(bool) + "</p>");
document.write("<p>Логическое значение: " + bool +
"</p>");
```

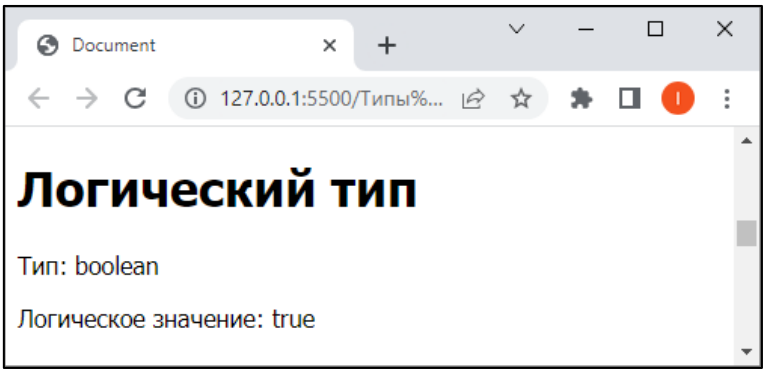


Рис. 4.18. Работа со значениями логического типа

4. *Тип String*

Строковый тип в JS кодирует символы по стандарту Unicode. Значения заключаются в кавычки (любые парные).

Дополнительные возможности представлены в глобальном объекте **`String`**.

Глава 4\Типы данных\js\string.js

```
// 4. Строковый тип
const text1 = "Текст в двойных кавычках";
const text2 = 'Текст в одинарных кавычках';
const text3 = `Текст в косых кавычках`;

// Свойства и функции строки
const text = "Пример текстовой строки";
const textLength = text.length;          // Длинна строки
// Извлечь массив слов, разделенных пробелом
const words = text.split(" ");
// Преобразовать в заглавные буквы
const bigLetters = text.toLocaleUpperCase();

document.write("<h1>Строковый тип</h1>");
document.write("<p>Тип: " + typeof(text) + "</p>");
document.write("<p>Значение = " + text + "</p>");
document.write("<p>Количество символов = " + textLength +
"</p>");
document.write("<p>Первое слово: " + words[0] + "</p>");
document.write("<p>Заглавные буквы: " + bigLetters +
"</p>");
```

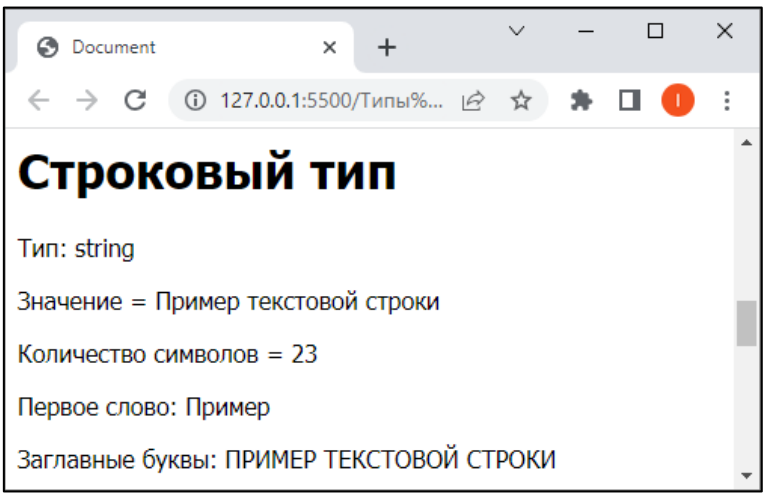


Рис. 4.19. Работа со строками

5. *Тип Undefined*

Специальное значение **undefined** означает, что «значение не было присвоено».

Обычно является показателем, что переменная не определена или не имеет значения.

Глава 4\Типы данных\js\undefined.js

```
// 5. Неопределенный тип
let undef1;

let undef2 = undefined;

document.write("<h1>Неопределенный тип</h1>");
document.write("<p>Тип: " + typeof(undef1) + "</p>");
document.write("<p>Значение = " + undef1 + "</p>");
document.write("<p>Значение = " + undef2 + "</p>");
```

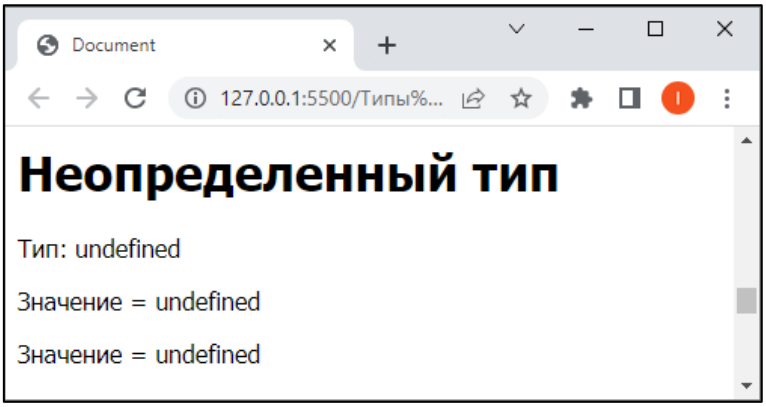


Рис. 4.20. Значение неопределенного типа

6. *Тип Null*

Специальное значение **null** означает, что значение неизвестно.

Обычно используется как маркер того, что объект пока еще пустой.

Глава 4\Типы данных\js\null.js

```
// 6. Пустой тип
const vector = null;

document.write("<h1>Пустой тип</h1>");
document.write("<p>Тип: " + typeof(vector) + "</p>");
document.write("<p>Значение = " + vector + "</p>");
```

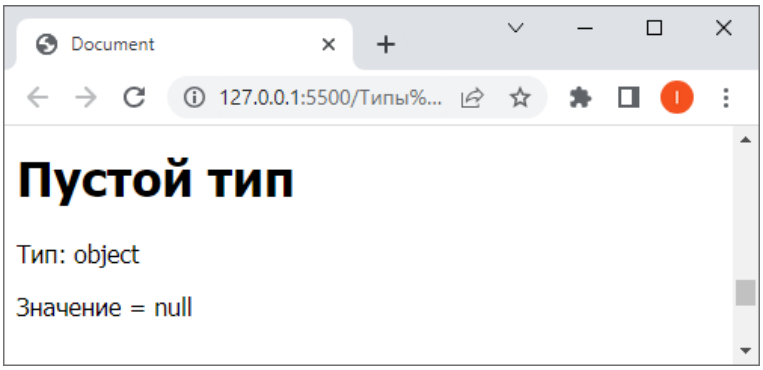


Рис. 4.21. Значение null

7. Тип Object

Тип **`Object`**, в отличие от перечисленных базовых (примитивных) типов, хранит коллекции данных или более сложные структуры. Любой создаваемый объект наследует объект `Object`.

Объект является центральным понятием JavaScript, относительно которого выстраиваются концепции ООП.

Глава 4\Типы данных\js\object.js

```
// 7. Объекты
const player = {
    name: "Александр",
    games: ["Герои 3", "WoW", "Lineage II"],
    totalScore: 5120090,
    rank: 3
};

document.write("<h1>Объекты</h1>");
document.write("<p>Тип: " + typeof(player) + "</p>");
document.write("<p>Значение = " + player + "</p>");
document.write("<p>Значение свойства name = " +
player.name + "</p>");
```

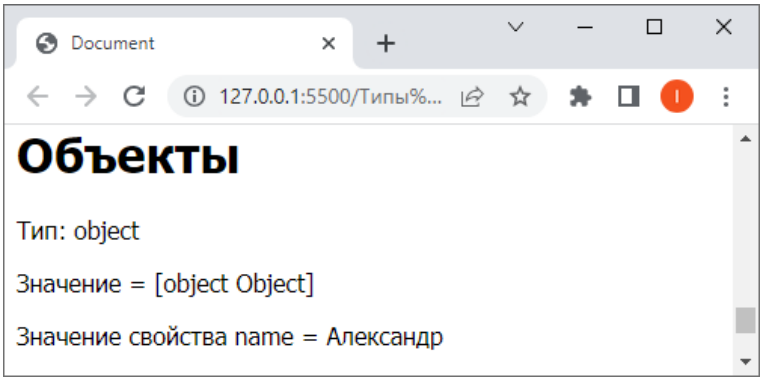


Рис. 4.22. Объектный тип данных

8. Тип Symbol

Тип **`Symbol`** позволяет определять уникальные идентификаторы, которые, в частности, можно использовать для организации более гибкого доступа свойствам объекта.

Например, название свойства объекта может быть сформировано в процессе работы скрипта.

Важно отметить, что поддержка этого типа была включена только в спецификации ES6.

`Глава 4\Типы данных\js\symbol.js`

```
// 8. Символьный тип
const special = Symbol("метаданные объекта");

const account = {
    login: "user12345",
    [special]: "[data]font: Arial[/data]"
};

document.write("<h1>Символьный тип</h1>");
document.write("<p>Тип: " + typeof(special) + "</p>");
document.write("<p>Значение свойства = " +
    account[special] + "</p>");
```

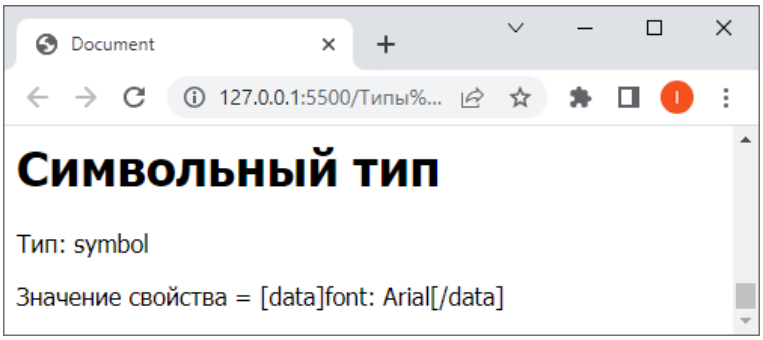


Рис. 4.23. Использование символьного типа

Здесь обращение к полю объекта осуществляется синтаксисом, похожим на работу с массивом. Через оператор-точку обратиться не получится, потому что название свойства содержит пробелы. А символьный тип решает эту проблему.

Объекты в JS еще называют *ассоциативными массивами*.

Вопросы для самопроверки

1. Какие типы комментариев поддерживаются в JavaScript.
2. Каким образом описать и инициализировать переменную или константу?
3. В чем разница между командами `let` и `var`?
4. Перечислите основные правила и рекомендации по именованию переменных.
5. Какие типы данных поддерживает JavaScript?
6. Опишите особенности каждого типа данных и приведите примеры описания переменных.

Практикум

Задание 1

1. Создайте каталог *Переменные и константы*.
2. Скопируйте предложенную ниже разметку, стили и фрагмент скрипта.
3. Следуя тексту из комментариев, последовательно опишите указанные переменные и константы, а также ниже выведите их на экран.
4. Ориентируйтесь на рис. 4.25.

Глава 4\Переменные и константы\index.html

```
<header class="header">
    <div class="row">
        <h1 class="header__title">
            Переменные и константы
        </h1>
    </div>
</header>

<main class="main">
    <div class="row">
        <div class="main__content">
            <h2>Вывод данных</h2>
            <script src="code.js"></script>
        </div>
    </div>
```

```
</main>
```

Глава 4\Переменные и константы\style.css

```
* { box-sizing: border-box; }

body {
    font: 16px Tahoma, Arial;
    margin: 0;
}

h1, h2 { color: #485a6d; }

h3 { color: #416c9b; }

.row {
    max-width: 480px;
    margin: 0 auto;
}

.header {
    background: #485a6d;
    min-height: 100px;
}

.header__title {
    color: #f3f7f6;
    margin: 0;
    padding-top: 30px;
}

.main__content {
    padding: 10px;
    margin: 10px 0;
    border: 2px solid #c0cecd;
}
```

Глава 4\Переменные и константы\code.js

```
// 1. Опишите числовую переменную year со значением 2022:
let year = 2022;

// 2. Опишите числовую переменную number со значением
34.06:
let number = 36.06;
```

```
// 3. Опишите отдельно константы PI и E с точностью до 4
знаков:

// 4. Опишите одной командой переменные
// dx со значением 0.02 и
// dy со значением 0.05:

// 5. Опишите строковую константу fullName, записав в нее
полностью ваши ФИО:

// 6. Опишите массив flags из трех логических значений:
true, true, false:

// 7. Опишите объект vector со свойствами x и y (значения
- любые)
// Выведите ниже значения этих свойств:

// 8. Опишите объект rectangle (прямоугольник) со
свойсвами height и width
// Выведите ниже значения этих свойств:

// 9. Опишите переменную emptyVariable, в которую ничего
не присваивайте

// 10. Опишите функцию sum(a,b), возвращающую сумму двух
заданных чисел a и b
function sum(a, b){
    return a + b
};

// Вывод переменных:
document.write("<h3>Числа</h3>");
document.write("<p>Целое число: " + year + "</p>");
document.write("<p>Дробное число: " + number + "</p>");

// Остальные переменные

document.write("<h3>Функция</h3>");
document.write("<p>Сумма двух чисел: " + sum(10,34) +
"</p>");
```

Задание 2

1. Создайте каталог *Именование переменных*, добавьте типовой файл разметки, подключите файл скрипта.
2. Следуя изученным рекомендациям по именованию переменных, а также при необходимости используя онлайн-переводчик, только опишите переменные (константы) с любыми значениями (без вывода):
   a. числовая «счетчик»;
   b. числовая «начальная скорость»;
   c. логическая «статус записи»;
   d. числовая «размер файла»;
   e. массив из 10 чисел «цифры»;
   f. объект «квартира» со свойствами «число комнат», «площадь», «этаж», «адрес».

Задание 3

1. Создайте каталог *Определение типа*.
2. Изучите приведенный ниже код.
3. Создайте соответствующие файлы разметки, подключите внешним образом файлы для CSS-стилей и JS-скрипт.
4. В скрипте описаны 6 переменных. С помощью инструкции `document.write()` выведите построчно тип каждой переменной.
5. Чтобы выделить название переменной, оберните его тегом `<strong>`. Образец выполнения – на рис. 4.24.

Глава 4\Определение типа\index.html

```
<h1>Типы данных</h1>
<script src="code.js"></script>
```

Глава 4\Определение типа\style.css

```
body {
    font-family: 'Gill Sans', 'Gill Sans MT', Calibri,
'Trebuchet MS', sans-serif;
    font-size: 18px;
}

strong {
    color: brown;
    font-weight: bold;
```

```
    font-family: Courier, monospace;
}
```

Глава 4\Определение типа\code.js

```
let n = 20;

let text = "Привет!";

let done = true;

let undf;

let vector = {
    x: 1.2,
    y: 7.9
};

let sum = function(a,b) {
    return a + b;
};

document.write("<p>Тип <strong>n</strong>: " + typeof(n)
+ "</p>");
```

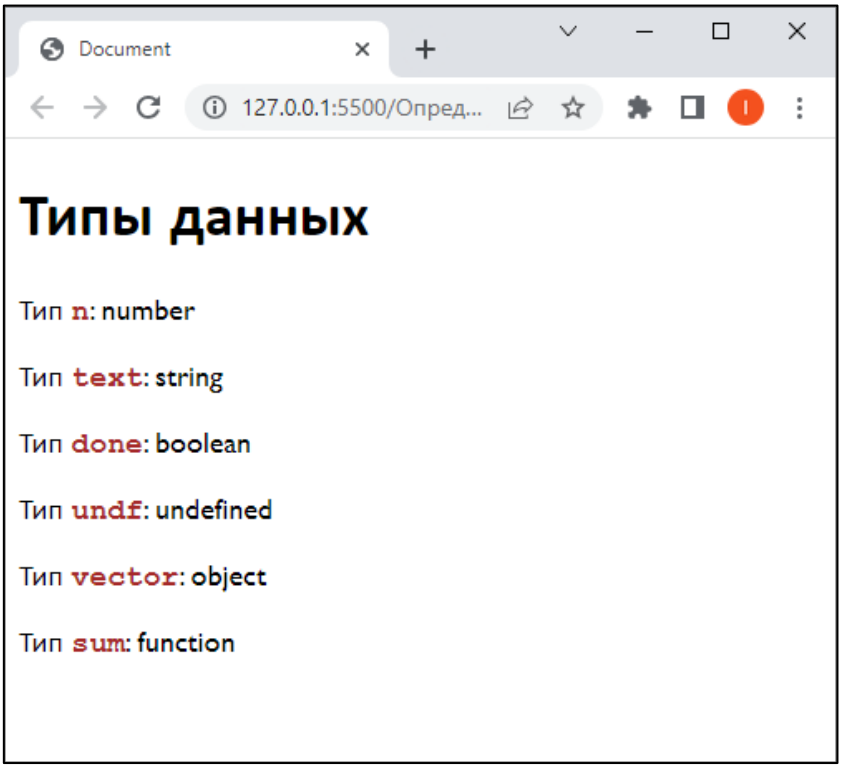


Рис. 4.24. Образец выполнения задания 3

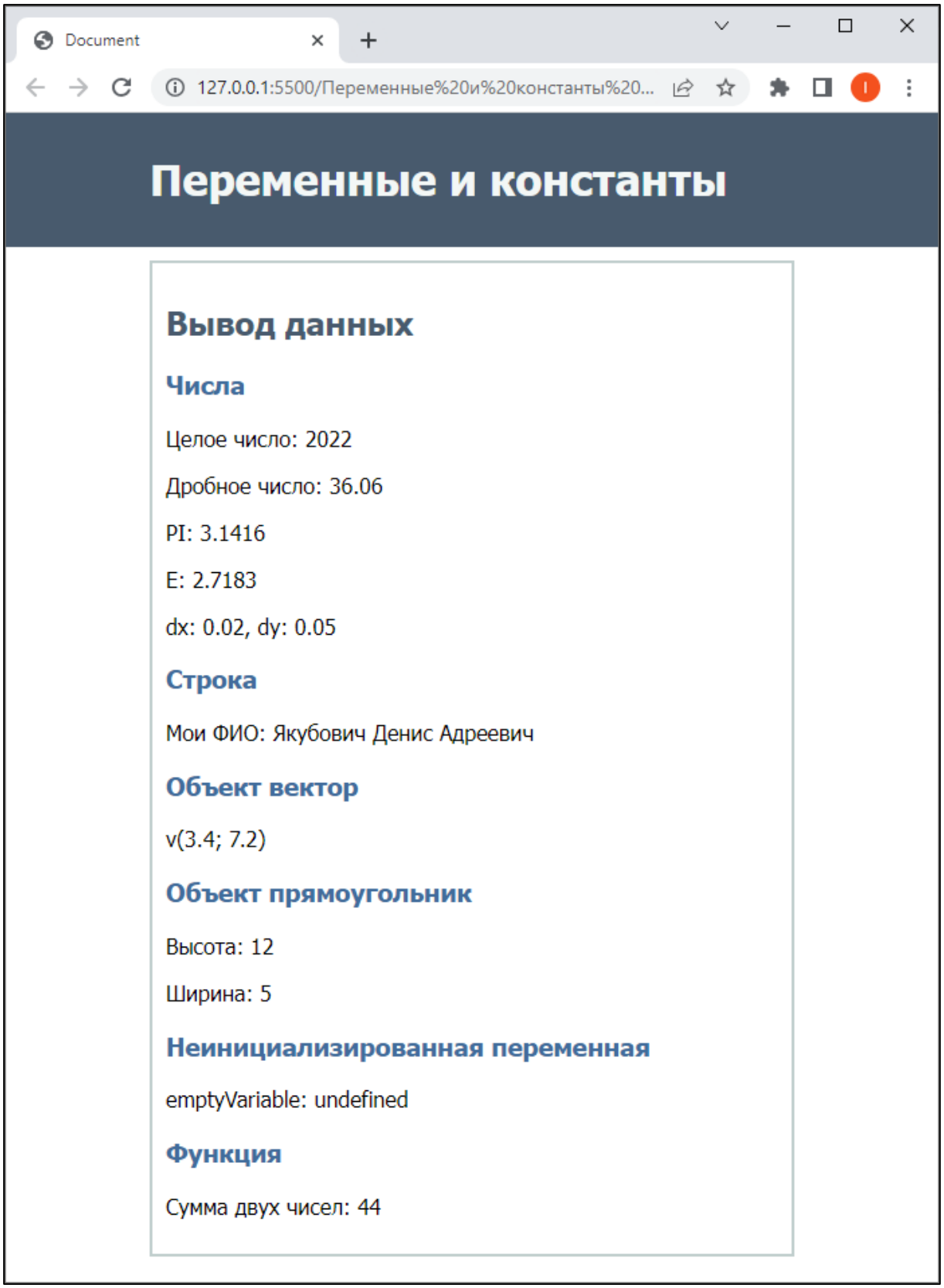


Рис. 4.25. Образец выполнения задания 1

4.3. Основы синтаксиса

4.3.1. Операторы

1. Арифметические операторы

Обычные

JavaScript поддерживает все основные *арифметические операторы* для математических расчетов, а также некоторые дополнительные. Как и в других языках, скобки позволяют управлять порядком выполнения операций.

Таблица 4.2. Арифметические операторы

Оператор	Действие
+	Сложение либо унарный плюс (знак числа).
-	Разность либо унарный минус (знак числа).
*	Умножение.
/	Деление.
%	Остаток от деления.
**	Возведение в степень.

Примеры:

```
let a = 10;
let k = 0.5;

let res = k * (a + 30);    // res <- 20
res = res % 6;             // res <- 2 (20 = 6*3 + 2)

let deg = res ** k;        // res <- 2^0.5
```

Расширенные

Кроме того, в JavaScript существует целая группа команд, которые расширяют синтаксис оператора присваивания. Их также можно отнести и к арифметическим операторам, которые позволяют в ряде случаев значительно сократить запись и убрать дублирование изменяемой переменной в правой части.

Таблица 4.3. Расширенные операторы

Оператор	Действие
`++`	Инкремент (увеличить на 1).
`--`	Декремент (уменьшить на 1).
`+=`	Добавить.
`-=`	Вычесть.
`*=`	Домножить.
`/=`	Поделить.
`%=`	Записать остаток от деления.

Примеры:

```
let counter = 1;
counter++;          // аналогично: counter = counter + 1
                    // (counter <- 2)

let sum = 10;
sum += 20;          // аналогично: sum = sum + 20
                    // (sum <- 30)
```

При этом для *инкремента* и *декремента* существует постфиксная и префиксная форма:

- *постфиксная* форма сначала вычисляет выражение, а затем меняет переменную;
- *префиксная* форма сначала меняет переменную, а затем вычисляет выражение с уже измененным значением этой переменной.

Например, в следующем случае префиксная и постфиксная форма никак не влияют на вычисления:

```
let counter = 0;
counter++;          // counter <- 1
++counter;          // counter <- 2
```

А в составных выражениях порядок ассоциирования операторов инкремента и декремента важен!

Например, для постфиксной формы `b` изменится после вычисления `x`:

```
let a = 0;
let b = 1;

let x = a + b++;    // Сначала x <- 0 + 1
                    // Затем b <- 2
```

Однако для префиксной формы сначала увеличится `b`, и только потом вычисляется `x`:

```
let a = 0;
let b = 1;

let x = a + ++b;    // Сначала b <- 2
                    // Затем x <- 0 + 2
```

Это полезно знать!

Не рекомендуется использовать операторы инкремента и декремента в составных выражениях: это повышает риск получить нежелательный результат из-за невнимательности.
Лучше всего их эти операторы использовать по одиночке.

2. Операторы сравнения

Перечень операторов

Операторы сравнения позволяют сравнивать выражения, допускающие упорядочивание. Значения можно проверить на равенство (неравенство) и превосходство.

Результатом сравнения является значение логического типа: **`true`** либо **`false`**.

Однако в JavaScript в силу слабой типизации сравнение имеет некоторую специфику при проверке операндов на равенство.

Таблица 4.4. Операторы сравнения

Оператор	Действие
`==`	Равенство (с приведением типа).
`!=`	Неравенство (с приведением типа).
`===`	Тождественное равенство.
`!==`	Тождественное неравенство.
`>`	Больше.
`<`	Меньше.
`>=`	Больше или равно.
`<=`	Меньше или равно.

Тождественное равенство

В JavaScript отдельно выделяют понятие *равенства* и *тождественного равенства*:

- В случае операторов `==` и `!=` проверяются только значения. Если JavaScript может привести сравниваемые выражения к одному значению, то они считаются равными / неравными.
- В случае операторов `===` и `!==` проверяются и значения, и тип данных. Если значения и типы сравниваемых значений совпадают, то они считаются тождественно равными / неравными.

Поясним на примере:

```
document.write(5 == "5");     // true

document.write(5 === "5");    // false
```

В обоих случаях сравниваются выражения разных типов.

Для первого JavaScript пытается преобразовать сравниваемые значения к единому типу. Поскольку это возможно и значения совпадают, то возвращается истина.

Для второго случая по значению операнды также совпадают, однако их типы разные, поэтому они уже не равны тождественно.

Необходимость в тождественном сравнении

Подобное разбиение операторов сравнения необходимо, чтобы поддерживать гибкость языка. Поскольку JavaScript слабо типизированный, то он допускает неявные преобразования переменных разного типа. В частности – при сравнении.

В зависимости от ситуации может потребоваться либо обычное, либо тождественное сравнение.

Например, ответ с сервера может прийти число в виде числовой константы, а может и виде текстовой. Если такое допустимо, то можно использовать обычное сравнение.

Однако если требуется жесткий контроль над значениями и их типами, то необходимо использовать уже операторы тождественного сравнения.

Это полезно знать!

В целом же чаще всего используется обычное сравнение `==` / `!=`. На начальном этапе изучения JavaScript рекомендуется просто избегать ситуаций с проверками, где сравниваются разные типы данных или возможны неоднозначные результаты.

3. Логические операторы

Операторы

Логические операторы выполняют роль логических союзов для выражений. Результатом сравнения является значение логического типа: **`true`** либо **`false`**.

Таблица 4.5. Логические операторы

Оператор	Действие
`!`	Логическое НЕ.
`&&`	Логическое И.
`||`	Логическое ИЛИ.

Например:

```
const a = 1.0;
const b = 5.6;
const x = 2.82;

if ((a <= x) && (x <= b))
    document.write("x принадлежит [a,b]");
else
    document.write("x не принадлежит [a,b]");

let error = !true;        // false
```

Оптимизация вычислений

Стоит отметить, что логические **&&** и **||** используют оптимальный алгоритм вычисления.

- Оператор && (союз И) проверяет составное выражение до первого ложного и в этом случае возвращает `false`, не продолжая проверять остальные выражения.
- Оператор || (союз ИЛИ) проверяет составное выражение до первого истинного и в этом случае возвращает `true`, не продолжая проверять остальные выражения.

4. Другие операторы

JavaScript также поддерживает и другие операторы. Например, *побитовые*, которые часто используются в различных тонких оптимизациях вычислений, где важно или возможно. Подробнее информацию о побитовых операциях можно найти здесь: learn.javascript.ru.

Кроме того, некоторые арифметические операторы в зависимости от контекста могут отвечать за преобразования чисел в строки (см. далее).

4.3.2. Математические функции

Объект Math

Объект **Math** предоставляет целый ряд функций и свойств, которые необходимы при математических расчетах. Каждая функция является частью объекта и вызывается относительно его названия. Такие функции также называют *методами объекта*.

Методы Math

Таблица 4.6. Некоторые методы объекта Math

Функция	Описание
`abs(x)`	Возвращает модуль числа x.
`pow(x,y)`	Возвращает x^y.
`sqrt(x)`	Возвращает квадратный корень числа x.
`sin(x), cos(x), tan(x)`	Возвращает синус, косинус и тангенс числа x.
`asin(x), acos(x), atan(x)`	Возвращает арксинус, арккосинус и арктангенс числа x.
`log(x)`	Возвращает натуральный логарифм $\ln x$.
`exp(x)`	Возвращает экспоненту e^x.
`min(), max()`	Возвращает наименьшее / наибольшее число из своих аргументов.
`round(x)`	Возвращает значение числа, округлённое до ближайшего целого.
`trunc(x)`	Возвращает целую часть числа, отсекая дробную часть.

Свойства Math

Таблица 4.7. Некоторые свойства объекта Math

Свойства	Описание
`E`	Возвращает число $e = 2{,}71\ldots$
`PI`	Возвращает число $\pi = 3{,}14\ldots$

Пример:

```
let n = Math.sqrt(225);     // n <- 15
const Pi = Math.PI;         // Pi <- 3.14159...

let res = 10 * Math.cos(Pi*n);
```

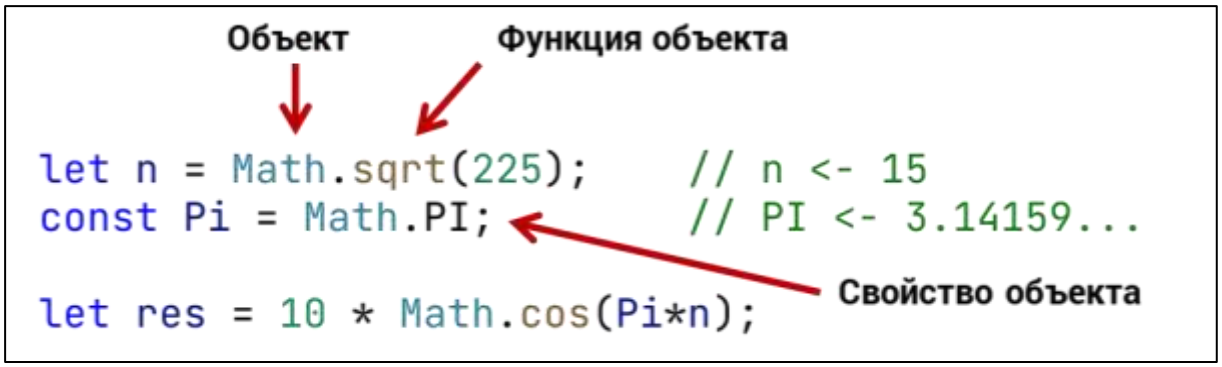


Рис. 4.26. Элементы объекта

Задача «Банковский вклад»

Условие

Пусть процентная ставка по вкладу «Выгодный» составляет 11,5% годовых. Пользователь открывает вклад размером S_0 на n полных лет.

Какой доход будет получен по истечению срока, если ставка не будет меняться?

Математическая модель

Для расчета прибыли по вкладам используется формула сложного процента. Ограничимся ее простейшим случаем. Получим эту формулу.

Пусть $percent$ – процент по вкладу, S_0 – начальный капитал. Для перевода процента в дробь берем $p = percent/100$.

Сумма, полученная к концу первого года, равна начальному капиталу плюс доход (процент):

$$S_1 = S_0 + pS_0 = S_0(1 + p).$$

В следующем году в качестве начальной суммы рассматривается уже S_1. Тогда к концу второго года сумма накоплений составит

$$S_2 = S_1 + pS_1 = S_1(1 + p) = S_0(1 + p)^2.$$

Очевидно, что к концу третьего года итоговая сумма составит

$$S_3 = S_2(1 + p) = S_0(1 + p)^3.$$

Таким образом, если процентная ставка остается неизменной, то следующая формула позволяет определить остаток по вкладу за определенный период времени:

$$\boldsymbol{S_n = S_0(1 + p)^n},$$

где n – число лет.

Чистый доход тогда составит $S_n - S_0$.

Программная реализация

Глава 4\Банковский вклад\index.html

```
<h1>Вклад</h1>
<script src="code.js"></script>
```

Глава 4\Банковский вклад\style.css

```
body {
    font: 16px Tahoma;
}

h1 {
    background: #086d8b;
    color: #fff;
    padding: 8px;
    margin: 0;
}
```

Глава 4\Банковский вклад\code.js

```
const s0 = 150000;
const n = 5;
const percent = 11.5;

const p = percent / 100;
let s = s0 * Math.pow(1 + p, n);

// Вывод на экран
document.write("<p>Начальный вклад: " + s0 + "</p>");
document.write("<p>Процентная ставка (годовых): " +
    percent + "%</p>");
document.write("<p>Срок вклада (лет): " + n + "</p>");
document.write("<hr>");
document.write("<p>ИТОГО: " + s.toFixed(2) + "p</p>");
document.write("<p>Доход: " + (s - s0).toFixed(2) +
    "p</p>");
```

Поскольку в общем случае числа получаются дробными, то используем метод **toFixed()**, который форматирует число до указанного количества цифр в дробной части. Метод вызывается относительно чисел, как объектов.

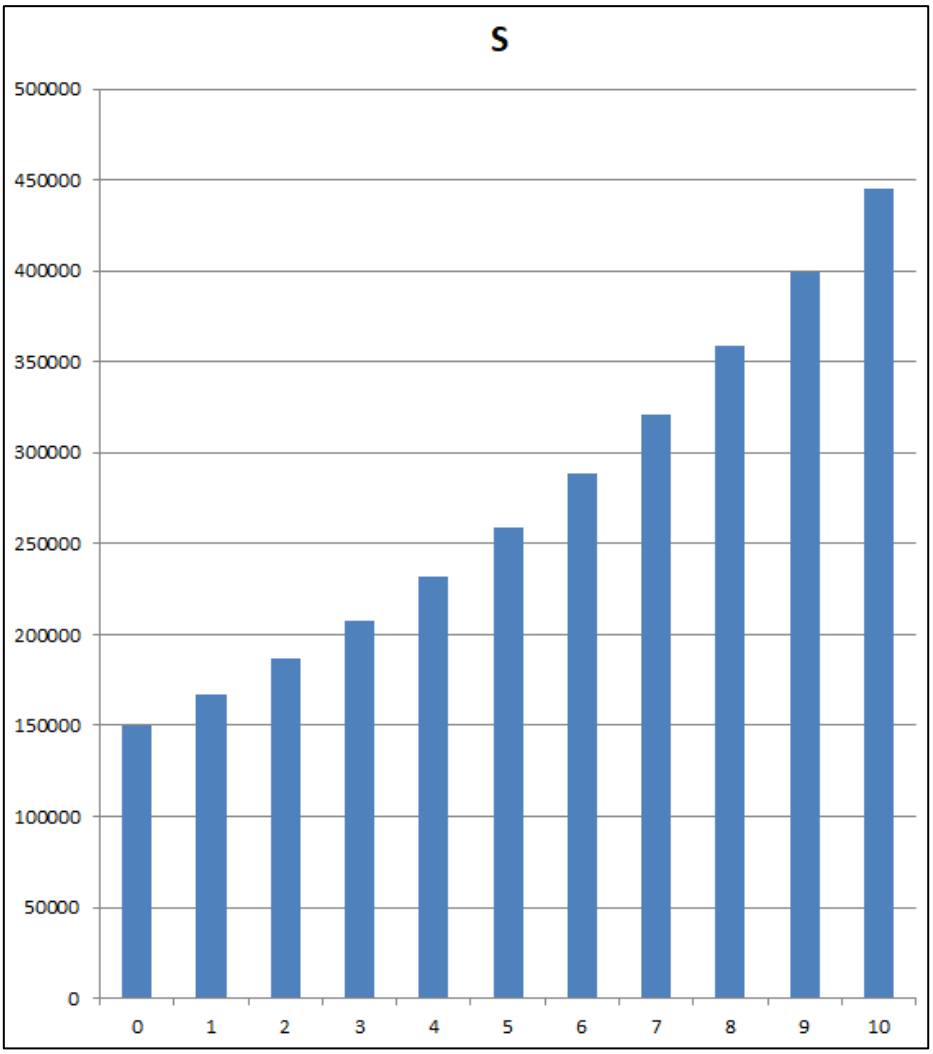


Рис. 4.27. График изменения суммы вклада в первые 10 лет

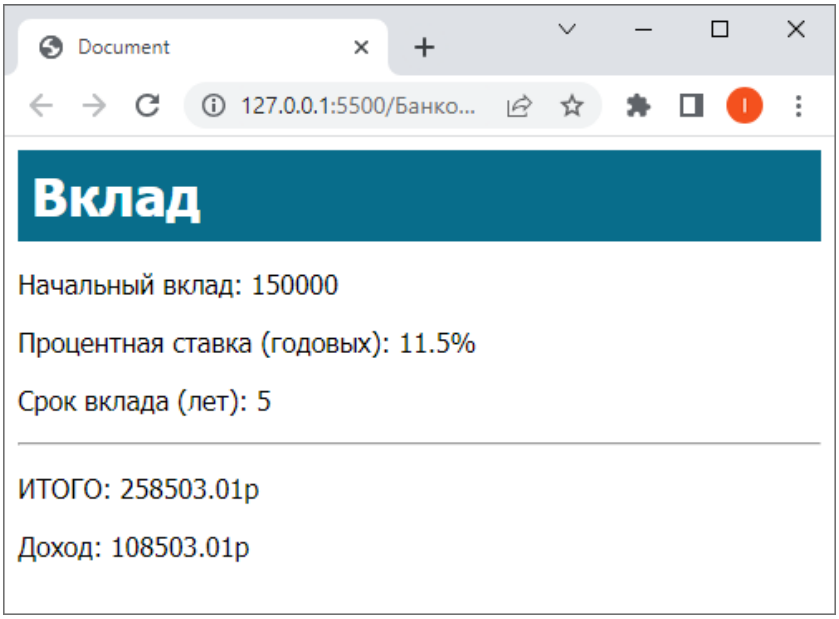


Рис. 4.28. Результат работы скрипта

4.3.3. Простейшие команды ввода / вывода

Метод alert

Определение

Метод ***alert()*** *выводит* ***модальное*** *окно с указанным сообщением. Синтаксис:*

```
alert(сообщение);
```

Особенность модального окна состоит в том, что оно блокирует работу страницы до тех пор, пока не закрыто. В современном веб-дизайне модальные окна не приветствуются, но могут использоваться в отладке.

```
let number = 10;
let text = "Денис";

alert("Привет");
alert("Якубович " + text);
alert("число: " + number);
```

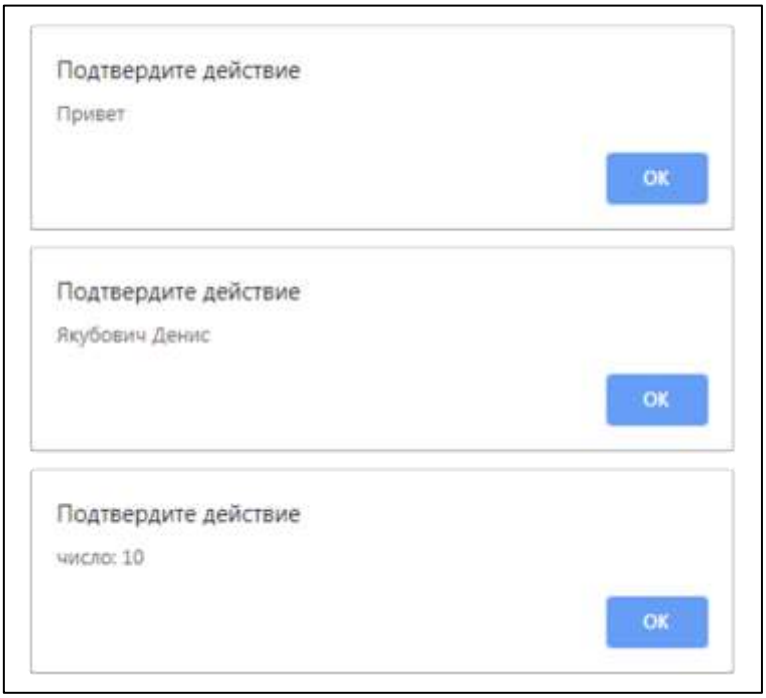


Рис. 4.29. Модальные окна в приведенном примере открываются последовательно

Метод confirm

Определение

Метод ***confirm()*** *выводит окно с вопросом и двумя кнопками: OK и CANCEL. Метод возвращает результат логического типа:* ***true*** *при нажатии OK и* ***false*** *при нажатии CANCEL. Синтаксис:*

```
let answer = confirm(вопрос);
```

Пример:

```
let isAdmin = confirm("Вы являетесь администратором?");

if (isAdmin)
    alert("Добро пожаловать!");
else
    alert("Выберете другую учетную запись.");
```

В условии можно писать полностью: `isAdmin == true`.

Но по умолчанию любое логическое выражение в условии сравнивается с `true`, поэтому явная проверка излишняя.

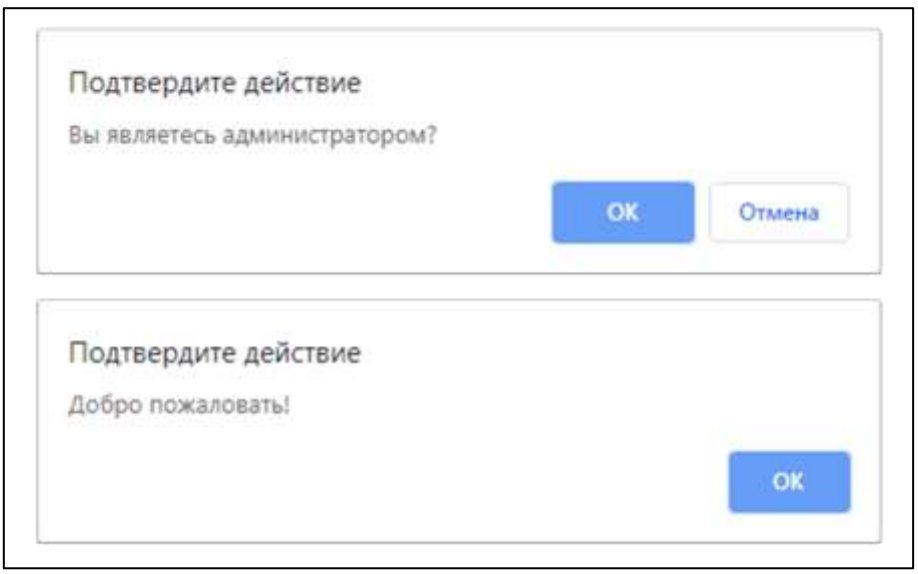


Рис. 4.30. Работа с `confirm`

Метод prompt

Определение

Функция ***prompt()*** *выводит модальное окно с заголовком «Текст» и полем для ввода текста, заполненным текстом «Значение по умолчанию» и кнопками OK/CANCEL. Возвращает введенный текст (т.е. значение типа «string»). Синтаксис:*

```
let tmp = prompt("Текст", "Значение по умолчанию");
```

Пример:

```
let num = prompt("Введите число:", "10");

let text = prompt("Введите текст:","");
```

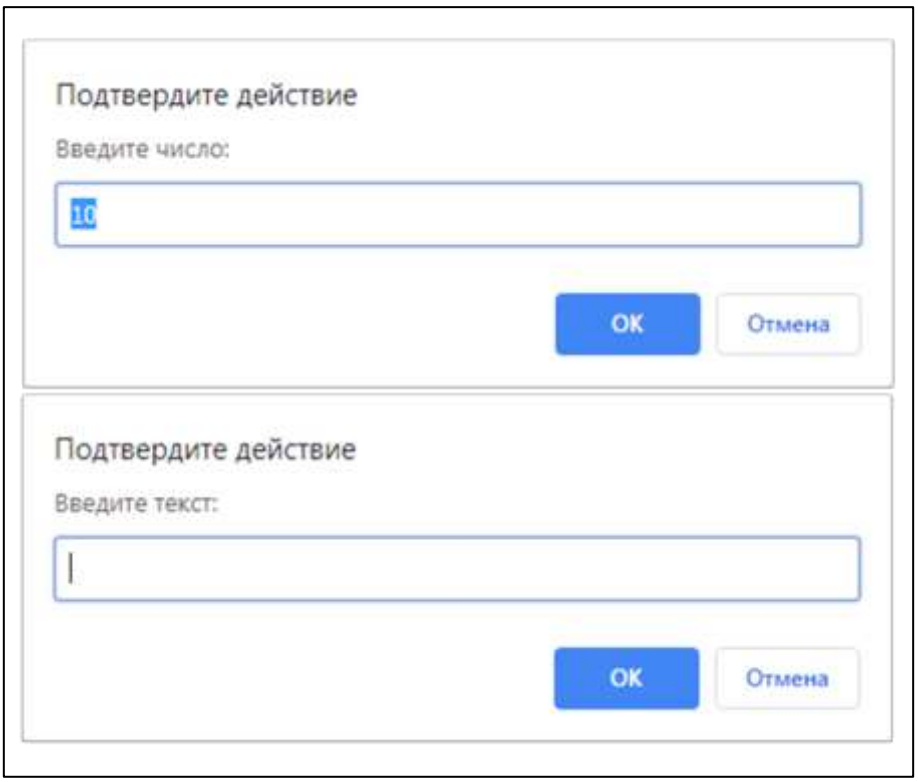


Рис. 4.31. Использование окна ввода

Метод write

Определение

Метод **`write()`** *объекта* **`document`** *динамически вставляет содержимое в структуру HTML. Это может быть обычный текст или теги вместе с атрибутами, классами и идентификаторами. Синтаксис:*

```
document.write(текст);
```

Метод распознает в текстовой строке теги и вставляет их в том месте HTML, где вызван скрипт. При этом, например, CSS-классы также влияют на оформление элементов.

```
document.write("<h1>Заголовок 1</h1>");
document.write("<h1 class='title'>Заголовок 2</h1>");

let text = "Привет, Денис!";
document.write("<h1 class='title'>" + text + "</h1>");
```

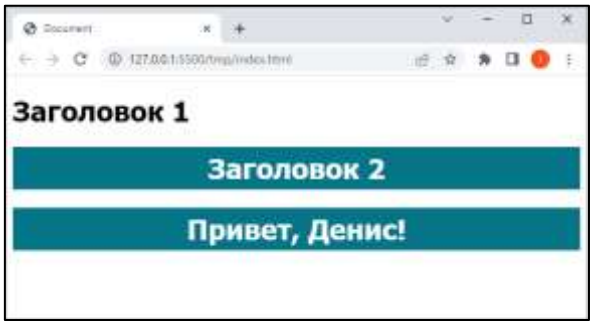


Рис. 4.32. Метод `write()` динамически меняет содержимое HTML-страницы

Это полезно знать!

Метод `write()` объекта `document` дает неплохие возможности по динамическому изменению структуры разметки. Однако это один из старейших методов JavaScript и он серьезно ограничивает гибкость работы с веб-страницами, к тому же небезопасен.
В современном JS активно используются методы DOM, которые позволяют обращаться к любому тегу с любыми классами и атрибутами, находящемуся в любом месте разметки.

4.3.4. Отладка кода в браузере

Консоль отладки в браузере

Для отладки скриптов и поиска ошибок удобно использовать консоль отладчика в браузере Google Chrome. Для этого необходимо нажать *ПКМ / Просмотреть код*. Отладочная информация выводится в разделе *Console*.

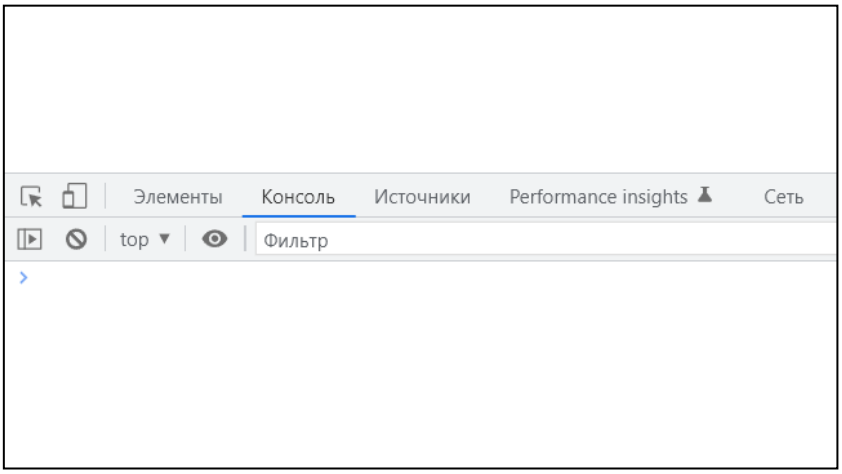


Рис. 4.33. Консоль отладки JavaScript-кода

Отладочная консоль доступна также, например – в браузере Mozilla Firefox, Opera, Yandex.

Метод log

Определение

Метод **`log()`** *объекта* **`console`** *выводит указанный текст в окно отладочной консоли браузера. Синтаксис:*

```
console.log(текст);
```

Метод `log()` крайне удобен в отладке. Он не блокирует выполнение скриптов и загрузку страницы, как модальные окна, а также не внедряет программный код в HTML-разметку, как `document.write()`.

Пример:

```
let a = 10;
let text = "Привет!";

console.log(a);
console.log(text);
console.log(document.body);
```

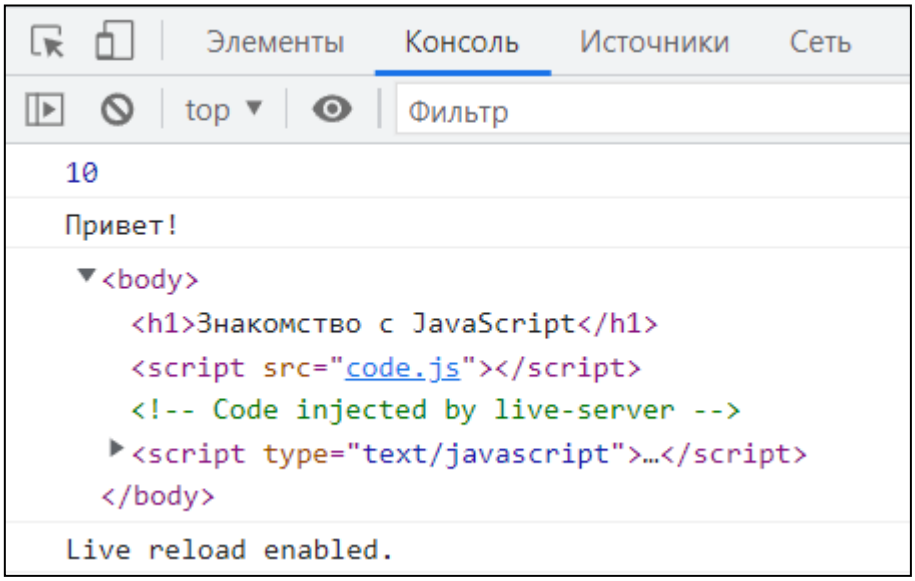


Рис. 4.34. Вывод информации в консоль отладки

Приемы работы с Visual Studio Code

*Набирать всю команду `console.log()` не обязательно: можно использовать сниппет **`log`**.*

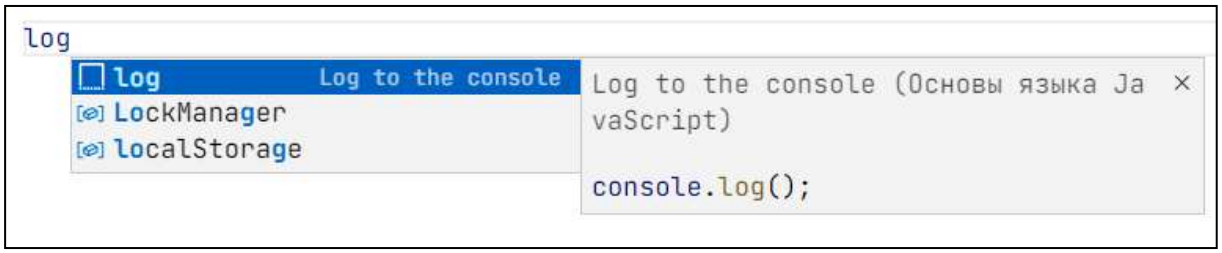


Рис. 4.35. Быстрая вставка метода `log()`

Метод table

Определение

Метод ***table()*** *объекта* ***console*** *позволяет структурировать вывод данных в окне отладочной консоли в форме таблицы:*

```
console.table(значение);
```

Метод `table()` особенно удобно использовать при выводе сводки по объектам.

Например, выведем информацию по массиву и объекту:

```
let array = [34, -45, 0, 23, 67, 23];

const student = {
    id: 12345,
    fullName: {
        name: "Денис",
        surname: "Якубович",
        patronymic: "Андреевич"
    },
    age: 20,
    group: "МИ-555"
};

console.table(array);
console.table(student);
```

Рис. 4.36. Метод `table()` в выводе сводки по объектам

Это полезно знать!

Используйте окно отладочной консоли при изучении JavaScript и дальнейшей работе. В нем можно проверить наличие ошибок, значение определенных переменных и сделать это существенно быстрее, чем используя какие-либо другие подходы.

4.3.5. Преобразование данных

Типы преобразований

JavaScript является гибким языком в плане преобразования типов, что зачастую избавляет разработчика от излишних операций приведения типов.

Однако такая гибкость полезна в том случае, если разработчик понимает детали преобразований и уверен, что исключены непредсказуемые операции с данными.

Можно выделить два типа преобразований.

1. Неявные преобразования типа

Осуществляются автоматически в операциях с разными типами данных. Подчинены определенному набору правил.

Например:

```
const result = 6 + "22";    // "622"
```

2. Явные преобразования типа

Указываются разработчиком с помощью вызова специальных команд и функций. Также называется *приведением типов*.

Например:

```
const result = 6 + Number("22");   // 28
```

Для явного преобразования в примитивные типы данных в JavaScript используются функции `Boolean()`, `Number()`, `String()`. Они играют роль конструкторов объектов заданного типа и пытаются преобразовать искомое выражение в него.

Также для явных преобразований используются операторы `+`, `-`, `++`, `--`, `!` (далее подробнее).

Преобразование чисел

Функция parseInt

Функция **parseInt()** возвращает целое число из строки. Если это невозможно, то возвращается значение **NaN** («Not a Number», «не число»).

```
console.log(parseInt("123text"));
console.log(parseInt("0xFF"));
console.log(parseInt(20.22));
```

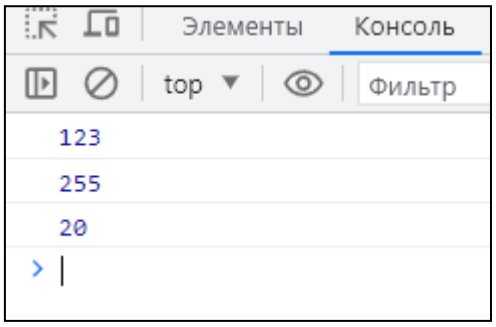


Рис. 4.37. Использование функции `parseInt()`

Функция parseFloat

Функция **parseFloat()** возвращает вещественное число из строки. Если это невозможно, то возвращается значение **NaN**.

```
console.log(parseFloat("20.22"));
console.log(parseFloat("2.1e-4"));
console.log(parseFloat("2.25text"));
console.log(parseFloat("d56.9"));
```

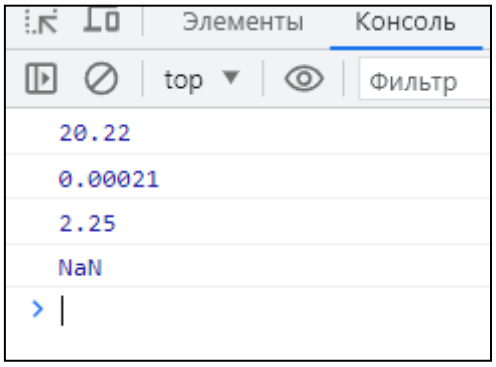


Рис. 4.38. Использование функции `parseFloat()`

Функция Number

Функция **`Number()`** осуществляет явное преобразование в число.

```
console.log(Number(undefined));
console.log(Number(null));
console.log(Number(true));
console.log(Number(false));
console.log(Number("20"));
console.log(Number("20.22"));
console.log(Number("20.22f"));
console.log(Number(2022));
```

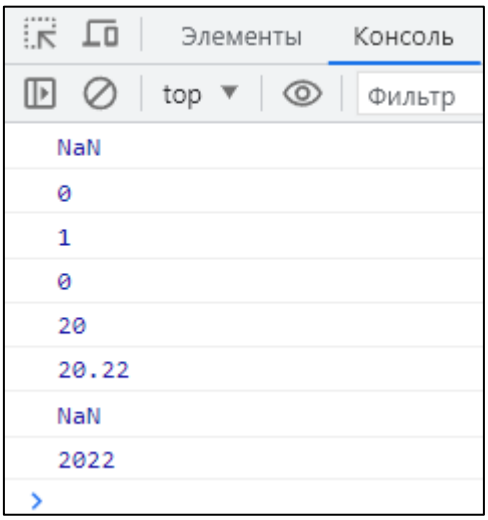


Рис. 4.39. `Number()` является конструктором объекта

Преобразование строк

Метод toString

Метод **`toString()`** возвращает текстовую запись об объекте, относительно которого он вызван. Обычно поведение метода переопределяют в объекте, который создает пользователь.

```
var num = 20.23;
var fl = true;

var obj = {
    id: 12345,
    pass: "Elena"
};

console.log(num.toString());
console.log(fl.toString());
console.log(obj.toString());
```

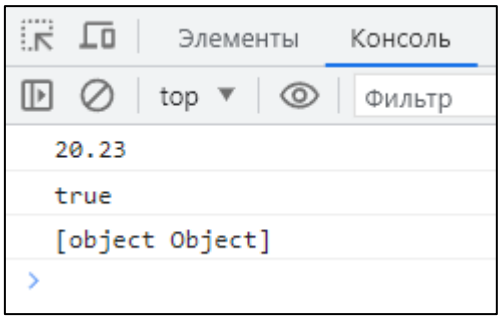


Рис. 4.40. По умолчанию `toString()` выводит `[object Object]` для всех объектов

Функция String

Функция **`String()`** осуществляет явное преобразование в строку.

```
console.log(String(null) === "null");
console.log(String(2022));
console.log(String(20.22));
console.log(String(undefined));
console.log(String(null));
console.log(String(true));
console.log(String(false));
console.log(String("2023"));
```

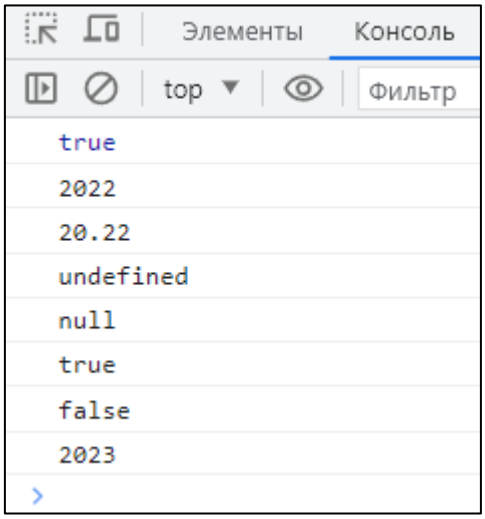


Рис. 4.41. `String()` является конструктором объекта

Логические преобразования

Функция **`Boolean()`** осуществляет явное преобразование в логическое значение.

```
console.log(Boolean(0));
console.log(Boolean(0));
console.log(Boolean(-0));
```

```
console.log(Boolean(NaN));
console.log(Boolean(undefined));
console.log(Boolean(null));
console.log(Boolean(""));
console.log(Boolean(false));

console.log(Boolean(45));
```

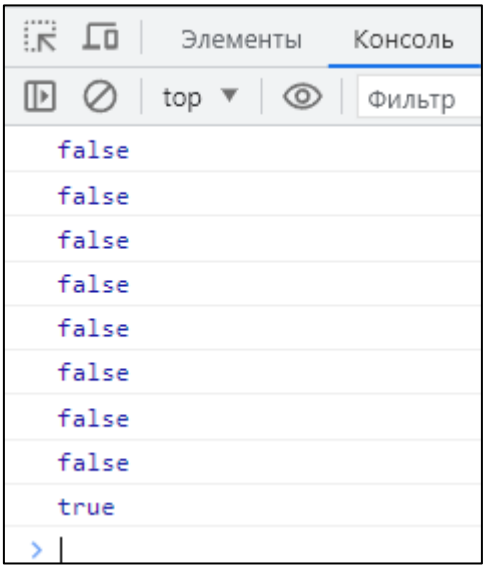


Рис. 4.42. `Boolean()` является конструктором объекта

Все значения, кроме последнего, при конвертировании дают `false`. Любое «ненулевое» выражение, от переменных примитивного типа до объектов, конвертируются в `true`.

Операторы в преобразованиях

***Операторы* +, -**

Унарные операторы **+** или **–** конвертируют строку в число, если это возможно.

```
// Унарные операторы "+" и "-"
const a = +"20";          // в число
const b = 5 + "20";       // в строку
const c = "5" + 20;       // в строку
const d = +"5" + 20;      // в число
const e = +"5" + -"20";   // в число

console.table([a, b, c, d, e]);
```

Рис. 4.43. Вариации преобразования в строки или числа

Унарный + или - перед строкой аналогичны действию функции `Number()`. При использовании + со строкой и числом получается строка (*конкатенация*).

Оператор !!

Оператор `!!` конвертирует строку в логическое выражение.

```
const f = !!"";
const g = !!" ";
const h = !!"0";
const i = !!"124";

console.table([f, g, h, i]);
```

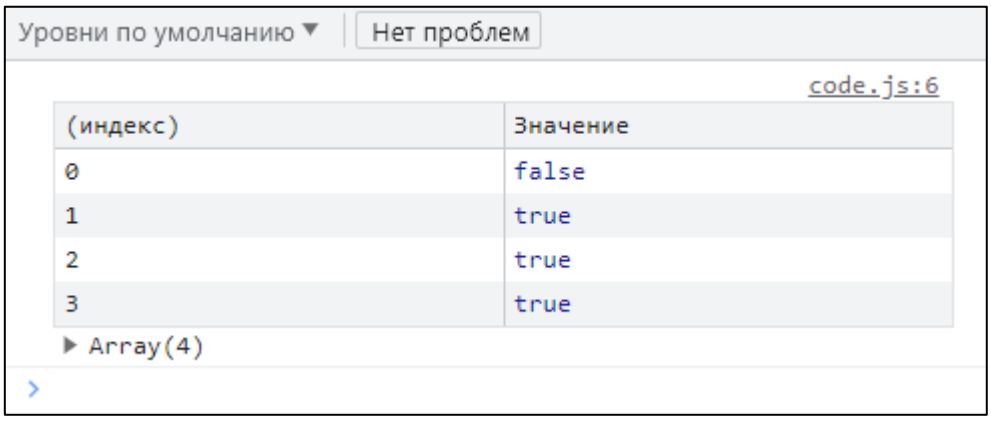


Рис. 4.44. Вариации преобразования в логический тип других типов данных

Преобразования типов – достаточно многогранная тема в JavaScript. Узнать больше можно, например, на сайте doka.guide.

Это полезно знать!

В начале изучения JavaScript старайтесь не делать упор на мало очевидные неявные преобразования. Как было отмечено ранее, в силу слабой типизации языка некоторые преобразования далеко не всегда могут приводить к ожидаемому результату.

Вопросы для самопроверки

1. Перечислите арифметические операторы, поддерживаемые в JavaScript.
2. В чем отличие между оператором обычного и тождественного сравнения?
3. Для каких целей можно использовать логические операторы?
4. Опишите возможности объекта `Math`.
5. Перечислите функции, которые создают модальные окна и опишите их назначение.
6. Какие возможности поддерживает `document.write()`?
7. Чем полезна отладочная консоль в браузере?
8. В чем отличие явных преобразований от неявных?
9. Опишите функции для преобразования чисел, строк и логических выражений.
10. Перечислите операторы, которые могут использоваться в преобразованиях чисел и строк.

Практикум

Задание 1

1. Создайте каталог *Окружность*.
2. Сгенерируйте стандартный начальный файл *index.html*.
3. Подключите таблицу стилей и скрипт. Скопируйте код в файлы, как указано в конце задания.
4. Вычислить и вывести на экран радиус, длину окружности и площадь круга (рис. 4.45).
5. Округлите выводимые значения с точностью до 3-х знаков.

Глава 4\Окружность\index.html

```
<div class="box">
    <h1 class="title">Окружность</h1>
    <script src="code.js"></script>
</div>
```

Глава 4\Окружность\style.css

```
body {
    background: url("wallpaper.jpg");
    font: 1.1em Arial;
    margin: 0;
}

.box {
    background: rgba(255, 255, 255, 0.6);
    padding: 15px 25px;
    width: 480px;
    margin: 0 auto;
    border-radius: 10px;
    margin-top: 40px;
}

.title {
    color: brown;
}

p {
    color: #3d0a6d;
}
```

Глава 4\Окружность\code.js

```
let radius = 5.0;
const PI = Math.PI;

// Опишите переменные circle_lenght и circle_area,
задайте в них формулы для вычисления искомых величин

// Далее последовательный вывод с помощью
document.write()
document.write("<p>Радиус = " + radius + "</p>")
```

Задание 2

1. Создайте каталог *Банковский вклад*.
2. Используя код разобранного в п. 4.3.2 примера, создайте необходимые файлы и проверьте корректность отображения страницы и работы скрипта.
3. Доработайте проект: организуйте ввод начального капитала, процента и срока через последовательный вызов функции `prompt()`.
4. Образец оформления изображен на рис. 4.46.

Задание 3

1. Создайте каталог *Математические вычисления*.
2. Сгенерируйте стандартный начальный файл *index.html*.
3. Подключите таблицу стилей и скрипт. Скопируйте код в файлы, как указано в конце задания. Изображение с формулами можно сделать скриншотом из текущего пособия.
4. Используя функции объекта `Math`, рассчитайте наиболее оптимальным способом значения следующих функций при заданных x и y:

$$f(x,y) = \sin\left(\ln\sqrt{x^2+y^2}\right)$$

$$g(x,y) = \cos\left(\ln\sqrt{x^2+y^2}\right)$$

$$h(x,y) = 2\cdot\sin\left(\ln\sqrt{x^2+y^2}\right)\cdot\cos\left(\ln\sqrt{x^2+y^2}\right)$$

5. Вывести результаты в разметку, используя изученные приемы работы с `document.write()`.
6. Ориентируйтесь на результат рис. 4.47.

Глава 4\Математические вычисления\index.html

```
<main class="main">
    <div class="container">
        <h1 class="main__title">
            Математические функции
        </h1>
        <div class="main__content">
            <script src="code.js"></script>
        </div>
    </div>
</main>
```

Глава 4\Математические вычисления\style.css

```
body {
    font: 18px Cambria;
    margin: 0;
}

.container {
    max-width: 720px;
    margin: 0 auto;
}

.main__title {
    background: #0a7052;
    color: #fafdfd;
    text-align: center;
    margin: 0;
    padding: 10px;
    border-radius: 6px;
    box-shadow: 0 0 15px #82fa32;
}

.main__content {
    padding: 10px 25px;
}
```

Глава 4\Математические вычисления\code.js

```
let x = 3.5;
let y = 5.0;

// Вычислить f, g, h и вывести их в разметку с
// форматированием
```

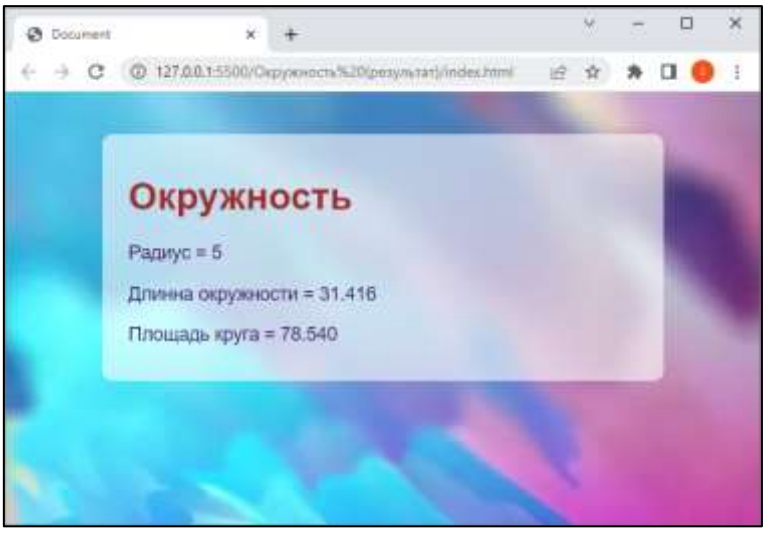


Рис. 4.45. Образец выполнения задания 1

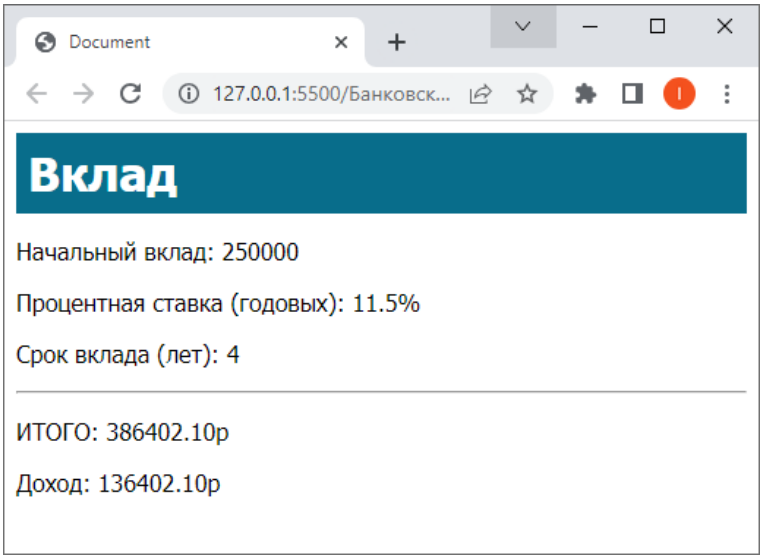


Рис. 4.46. Образец выполнения задания 2

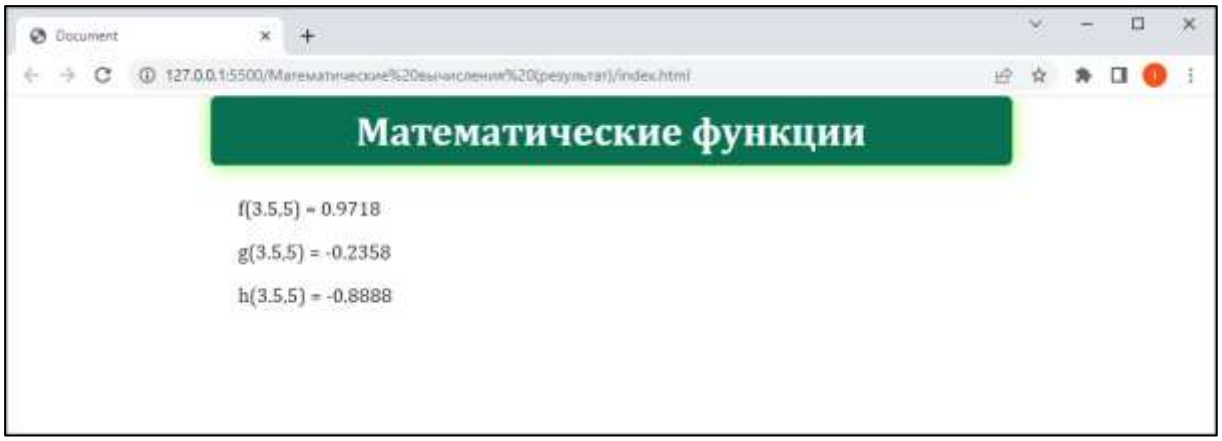


Рис. 4.47. Образец выполнения задания 3

4.4. Операторы условного выбора

4.4.1. Оператор if-else

Назначение и формы реализации

Определение

Инструкция ***`if`*** *реализует* ***разветвленное*** *выполнение блока инструкций в зависимости от истинности условия.*

Оператор `if` является основным для реализации разветвленных алгоритмов с проверкой условия, которые по значению логического выражения (условия выбора) осуществляют выполнение одной из двух ветвей. Для организации ветвлений с тремя и более выходами осуществляется вложение блоков `if-else`.

Оператор `if` допускает две основные формы реализации и одну расширенную.

1. Полная форма if

Определение

```
if (условие) {
    // делаем, когда истина
} else {
    // делаем, когда ложь
}
```

Предусматривает две ветви: первая содержит команды, которые сработают при истинном условии, а вторая – при ложном.

Важное замечание!

Круглые скобки в условии оператора обязательны!

2. Неполная форма if

Определение

```
if (условие) {
    // делаем, когда истина
}
```

В случае ложного условия ничего не происходит; осуществляется следующая после блока if команда.

Фигурные скобки, задающие тело оператора для полной и неполной реализации `if` не являются обязательными, если в них реализована одна инструкция. Тем не менее фигурные скобки блока рекомендуется писать в любом случае, чтобы исключить ошибки невнимательности.

Также заметим, что стилистика JavaScript предполагает, что открывающая скобка ставится после скобок условия, а закрывается отдельной строкой, на уровне ключевого слова if. Внутренние блоки сдвигаются вправо на заданную в редакторе величину отступа (2 или 4).

Пример

Рассмотрим пример. Пусть требуется вывести текущий год и определить, является ли он високосным.

Глава 4\Год\index.html

```
<h1>Год</h1>
<div id="information"></div>

<script src="code.js"></script>
```

Глава 4\Год\style.css

```
body {
    background: #f5f5f5;
    font: 16px Arial;
}

h1, h2, h3 {
    font-family: Tahoma, Geneva, Verdana, sans-serif;
    border-bottom: 2px solid #000;
}
```

Глава 4\Год\code.js

```
// Создаем ссылку на элемент с идентификатором
// "information":
const currentYear =
    document.getElementById("information");

let today = new Date();     // Создаем объект типа дата
let year = today.getFullYear();   // Получаем год

currentYear.innerHTML = "<p>Текущий год: " + year +
    ".</p>";

if (year % 4 == 0) {
    currentYear.innerHTML += "<p>Это високосный год</p>";
} else {
    currentYear.innerHTML += "<p>Это обычный год</p>";
}
```

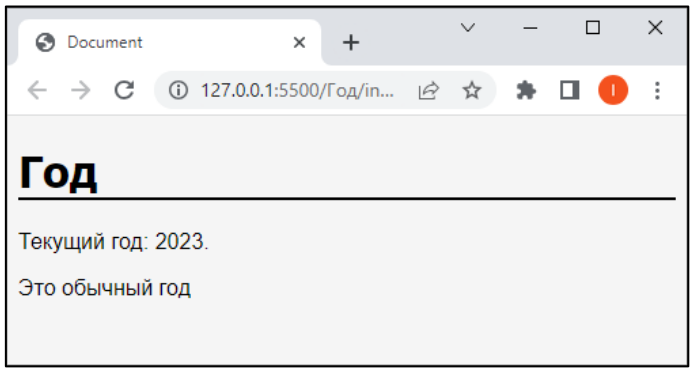


Рис. 4.48. Определяем текущий год

В этом примере мы впервые применили метод, позволяющий ссылаться на конкретный HTML-элемент по его идентификатору (атрибуту `id`):

```
const currentYear =
    document.getElementById("information");
```

Метод **`getElementById()`** из объекта **`document`** – это один из методов технологии *DOM*, позволяющий обращаться к элементу HTML-разметки по указанному идентификатору (`information`), который был задан пустому блоку `<div>` (см. HTML-разметку).

Объект `Date` является встроенным в JavaScript: он содержит ряд функций, позволяющих работать с датами. В частности метод `getFullYear()` возвращает год.

Еще одно нововведение – вместо вызова `document.write()` используется свойство объекта **`innerHTML`**, хранящее внутренний код разметки выбранного элемента в форме строки.

Это полезно знать!

Методы DOM позволяют JavaScript гибко управлять HTML-кодом, создавая более безопасную абстрактную модель дерева документа. Теги и их атрибуты в этой модели получают свойства и методы, которые можно использовать для обработки структуры, оформления и логики работы веб-страницы. Некоторые часто используемые особенности и приемы работы с DOM будут раскрываться в процессе изучения этого курса.

Фигурные скобки тела выражения

Фигурные скобки `{ }` определяют блоки оператора `if`. Если в блоке только одна инструкция, то их можно опустить. Так, в примере ранее допустимо сократить условие следующим образом:

```
if (year % 4 == 0)
    currentYear.innerHTML += "<p>Это високосный год</p>";
else
    currentYear.innerHTML += "<p>Это обычный год</p>";
```

Однако если в блоке несколько команд, то скобки опускать нельзя!

3. Расширенная форма if

Необходимость многократного выбора

Пусть требуется определить, является ли число положительным, равным нулю или отрицательным.

При решении этой задачи одной проверки недостаточно: есть три возможных варианта ответа, а инструкция `if` может организовать только две ветви.

Искомая задача может быть решена несколькими способами.

Первый способ: последовательно организовать три неполных проверки, тогда один (и только один) из трех случаев точно будет выполнен:

Глава 4\Число\code.js

```
let n = prompt("Введите число", "1");

if (n > 0)
    document.write("<p>Число положительное</p>");

if (n == 0)
    document.write("<p>Число равно нулю</p>");

if (n < 0)
    document.write("<p>Число отрицательное</p>");
```

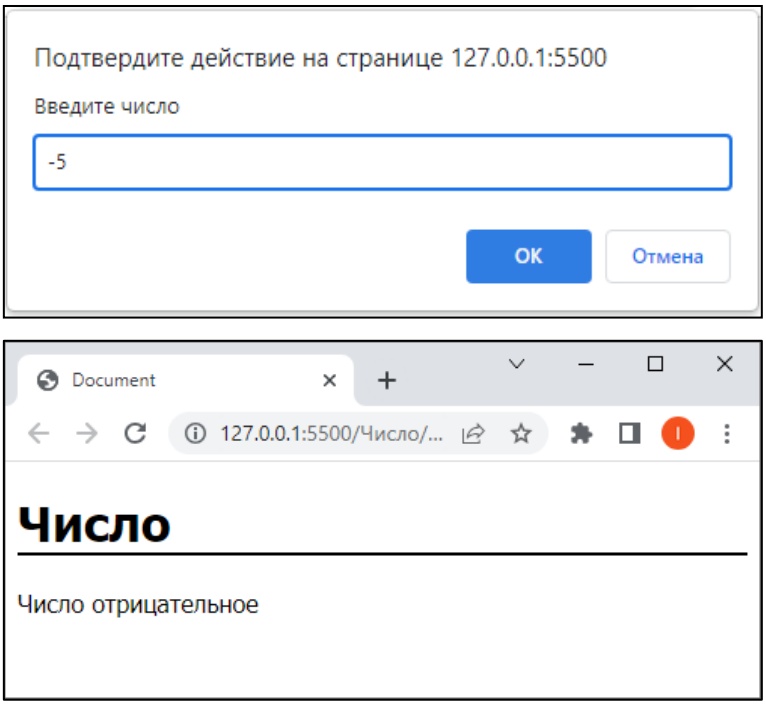


Рис. 4.49. Определение типа числа

Недостаток подхода: связанная по смыслу проверка разбита на отдельные блоки, неудобно анализировать код и легко допустить логическую ошибку.

Второй способ: последовательно вкладывать условия, пока не будут реализованы все случаи:

Глава 4\Число\code.js

```
let n = prompt("Введите число","1");

if (n > 0)
    document.write("<p>Число положительное</p>");
else {
    if (n == 0)
        document.write("<p>Число равно нулю</p>");
    else
        document.write("<p>Число отрицательное</p>");
}
```

Недостаток подхода: когда потребуется больше последовательных проверок, то получится множество вложений, что является потенциальным источником ошибок.

Третий способ заключается в использовании расширенного оператора if.

Синтаксис расширенного оператора

Определение

```
if (условие 1) {
    // условие 1 истинно
} else if (условие 2) {
    // условие 2 истинно
} else if (условие 3) {
    // условие 3 истинно
}
. . .
else {
    // все условия ложны
}
```

Инструкция ***if-else-if*** *последовательно проверяет условия и выполняет первый блок, в котором условие истинно; иначе выполняется блок* ***else*** *(необязателен). Инструкция является расширенной формой* ***if*** *с вложенными проверками, но в более компактной форме.*

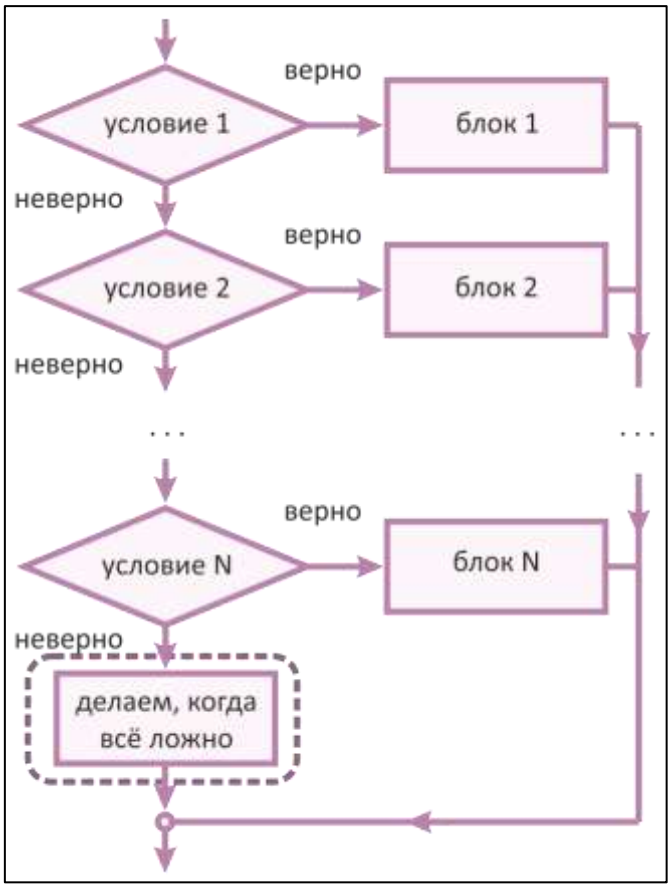


Рис. 4.50. Блок-схема расширенной проверки

Третий способ решения исходной задачи: используем расширенную форму оператора `if`:

Глава 4\Число\code.js

```
let n = prompt("Введите число", "1");

if (n > 0)
    document.write("<p>Число положительное</p>");
else if (n == 0)
    document.write("<p>Число равно нулю</p>");
else
    document.write("<p>Число отрицательное</p>");
```

Здесь для краткости фигурные скобки блоков опущены: в каждом всего лишь одна инструкция. Однако, как было замечено ранее, их лучше писать в любом случае.

Шаблонные строки в JavaScript

Определение

Шаблонная строка *– это способ создания строки, в которую можно осуществлять подстановку выражений. Шаблонная строка указывается в* ***косых кавычках*** ` `, *а подставляемое значение – внутри* ***${ }***.
Шаблонные строки поддерживаются, начиная со спецификации ES6.

Шаблонные строки повышают гибкость работы со строками и существенно их укорачивают, уменьшая вероятность потенциальных ошибок. Следующие строки кода эквивалентны:

```
let number = 20.22;

// Обычное склеивание
info.innerHTML = "<p>Число: " + number + "</p>";

// Шаблонная строка
info.innerHTML = `<p>Число: ${number}</p>`;
```

Так вместо `${number}` в строку подставляется значение переменной `number`. По сравнению с обычной конкатенацией удалось убрать операторы `+`, а также уменьшить число кавычек, сократив их до единой записи.

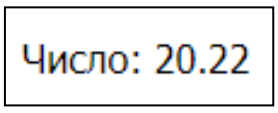


Рис. 4.51. Склеивание с помощью шаблонных строк

Для шаблонных строк используются косые кавычки (находятся на клавише с буквой *Ё*).

Шаблонные строки также часто называют *интерполированными строками*. Их следует использовать в случаях, когда в строковые литералы необходимо подставить значение переменных или констант.

4.4.2. Тернарный оператор

Синтаксис

Определение

Тернарный оператор является укороченной реализацией инструкции условного выбора ***if-else****. Удобен для выражений в одну команду. Обычно используется для сворачивания условной проверки в переменную.*

```
let rezult = (условие) ?
             значение_истина :
             значение_ложь;
```

Тернарный оператор позволяет оформить короткую инструкцию выбора в линейной форме. Однако в общем случае он не сможет заменить `if-else`.

В структуре оператора выделяются три части:

- условие (логическое выражение);
- значение, присваиваемое в `result` при `условие == true`;
- значение, присваиваемое в `result` при `условие == false`.

Пример

Пусть требуется записать в переменную `max` наибольшее из заданных чисел. С помощью обычно проверки это можно реализовать следующим образом:

```
let max;

if (a > b)
    max = a;
else
    max = b;
```

Тернарный оператор позволяет *линеаризировать* разветвленный алгоритм и записать его единым выражением:

```
let max = a > b ? a : b;
```

4.4.3. Оператор выбора switch

Синтаксис

Определение

Оператор **`switch`** *в зависимости от значения выражения осуществляет выполнение соответствующего блока кода. Важно отметить, что выражение может быть любого типа (и не обязательно перечислимого).*

```
switch (выражение) {
    case значение_1:   // if (выражение === значение_1)
        // блок операторов 1
        break;
    case значение_2:   // if (выражение === значение_2)
        // блок операторов 2
        break;
    . . .
    default:           // else
        // когда ни одно не подходит
        break;
}
```

Оператор **`break`** завершает выполнение блока **`case`** и выходит из тела оператора **`switch`**. Важно заметить, что для каждого case сверка выражения ведется с константой на условии тождественного равенства (т.е. и тип, и значения должны совпасть). В этой форме `switch` практически идентичен расширенной форме `if-else-if`.

Однако если `break` опустить в каком-либо блоке опустить, то блоки `case` ниже продолжат выполнение, при этом условия уже не проверяются! Такая реализация «проваливания» может быть полезна в ряде задач, но все же подобного приема лучше избегать

Задача «Светофор»

Пусть цвет светофора кодируется тремя числами, согласно следующей таблице:

Таблица 4.8. Состояния светофора

Код	Цвет
1	Красный
2	Желтый
3	Зеленый

Требуется определить цвет светофора, если в переменной `trafficLightsColor` задан числовой код его текущего состояния.

Глава 4\Светофор\index.html

```
<h1>Светофор</h1>
<p id="result"></p>

<script src="code.js"></script>
```

Глава 4\Светофор\style.css

```
body {
    font: 16px Tahoma;
}

h1, h2, h3 {
    border-bottom: 2px solid #000;
}
```

Глава 4\Светофор\code.js

```
const result = document.getElementById("result");

let trafficLightsColor = 2;     // Код цвета

switch (trafficLightsColor) {
    case 1:
        result.innerHTML = "Красный";
        break;
    case 2:
        result.innerHTML = "Желтый";
        break;
    case 3:
        result.innerHTML = "Зеленый";
        break;
    default:
        result.innerHTML = "Поломка!";
}
```

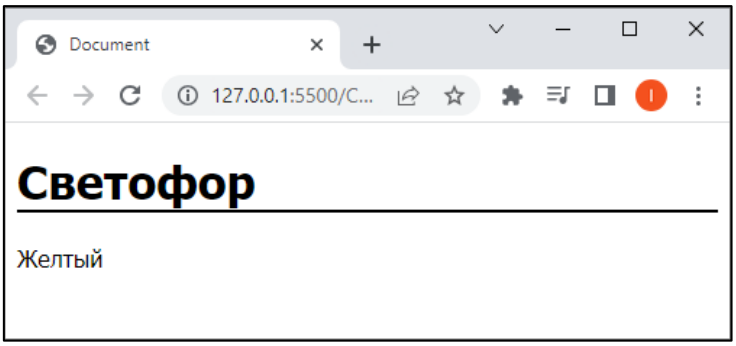


Рис. 4.52. Выбор цвета светофора

Как и в примере ранее, для обращения к HTML-элементу используем идентификатор: текст ответа будет вписан в абзац.

Дополнительно добавим четвертый вариант, который будет обозначать поломку светофора (т.е. ни одно из табличных чисел не содержится в `trafficLightsColor`).

Если опустить команду `break`, то можно реализовать достаточно интересную форму работы оператора `switch`:

```
switch (trafficLightsColor) {
    case 1:
    case 2:
    case 3:
        result.innerHTML = "Светофор работает";
        break;
    default:
        result.innerHTML = "Поломка!";
}
```

Здесь мы объединили первые три значения.

Замыкание с помощью объектов

Работа с переменной `trafficLightsColor` плохо организована. Не имея перед глазами таблицу сложно сказать, что подразумевает команда:

```
let trafficLightsColor = 2;    // Какой цвет "шифруется"
                               // под цифрой 2?
```

Во многих современных языках программирования для решения проблемы «магических чисел» есть возможность создавать тип данных, называемый *перечислением*.

В JavaScript подобной конструкции нет. Но похожую концепцию можно реализовать, используя объекты.

Рассмотрим самый простой способ.

Для начала создадим объект с тремя свойствами:

```
const colors = {
    red: 1,
    yellow: 2,
    green: 3
};
```

При записи значения в переменную ссылаемся на требуемое нам свойство объекта `colors`:

```
let trafficLightsColor = colors.yellow;     // желтый
```

Теперь это присваивание наделено смыслом для разработчика, а цифра «прячется» за более удобным текстовым синтаксисом.

Наконец, в операторе `case` вместо магических чисел указываем ссылки на соответствующие свойства объекта `colors`:

```
switch (trafficLightsColor) {
    case colors.red:
        result.innerHTML = "Красный";
        break;
    case colors.yellow:
        result.innerHTML = "Желтый";
        break;
    case colors.green:
        result.innerHTML = "Зеленый";
        break;
    default:
        result.innerHTML = "Поломка!";
}
```

Такой подход хорош и тем, что в некотором смысле создается *замыкание* (т.е. ограничение) относительно возможных значений переменной.

Кроме того, теперь Visual Studio Code выдает подсказки:

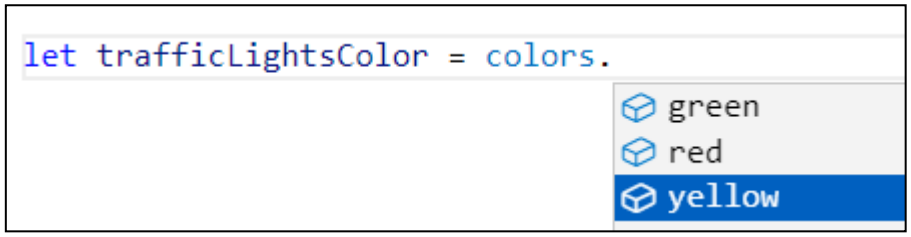


Рис. 4.53. Автоматическая подсказка значений перечисления как свойств объекта

Сравнение if и switch

Приведенные ранее примеры показывают, что `switch` может быть заменен расширенной реализацией оператора `if` без потери функционала. В каких случаях его стоит использовать?

Обычно `switch` удобен для обработки выражения одного типа с многочисленными значениями. Расширенная инструкция `if` удобнее при связывании нескольких условий в каждой ветке.

Как правило, `switch` рационально использовать в задачах по типу приведенного ранее примера, особенно для большого числа возможных вариантов. Разработчики по результатам тестов отмечают, что `switch` в таких случаях работает быстрее `if`.

Вопросы для самопроверки

1. Опишите структуру оператора `if` в полной и неполной форме.
2. Каким образом можно организовать проверку с тремя и более выходами?
3. Для каких проверок рациональнее использовать расширенную форму оператора `if`? А для каких проверок он не подходит?
4. Что представляют собой шаблонные строки и чем они упрощают синтаксис?
5. Приведите пример использования тернарного оператора в условии инструкции `if`.
6. Чем оператор `switch` отличается от расширенной формы `if`?
7. Для чего в операторе `switch` в выражении используются объекты?

Практикум

Задача 1

1. Создайте каталог *Координаты точки*.
2. Сгенерируйте стандартный начальный файл *index.html*.
3. Подключите таблицу стилей и скрипт. Скопируйте код в файлы, как указано в конце задания.

4. Внимательно изучите код разметки и фрагмент скрипта. Здесь приводятся примеры работы с кнопками и текстовыми полями, используется обработка событий.
5. Определить, попадает ли точка в сектор, если в текстовые поля заданы ее координаты:

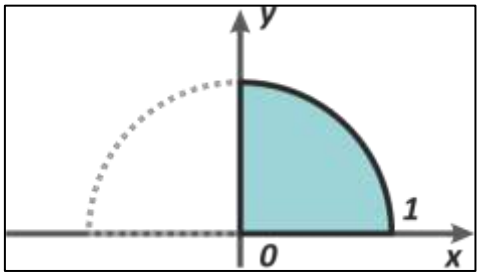


Рис. 4.54. Искомый сектор

6. Оформление и тестовые значения изображены на рис. 4.56.

Глава 4\Координаты точки\index.html

```
<h1>Попадание точки в область</h1>
<p>X: <input type="text" id="setX" value="0"></p>
<p>Y: <input type="text" id="setY" value="0"></p>
<button id="define">Определить попадание</button>
<div id="info"></div>

<script src="code.js"></script>
```

Глава 4\Координаты точки\style.css

```
body {
    background: whitesmoke;
    font-family: Arial, Helvetica, sans-serif;
    font-size: 18px;
}

h1 { border-bottom: 2px solid black; }
```

Глава 4\Координаты точки\code.js

```
// Получаем идентификаторы полей ввода, кнопки
// и блока вывода текста
const xCoord = document.getElementById("setX");
const yCoord = document.getElementById("setY");
const define = document.getElementById("define");
const info = document.getElementById("info");
```

```
// Подключаем функцию обработчик события клика на кнопку
define.onclick = function(){
    // Записываем координаты из текстовых полей
    // в переменные.
    // Свойство value возвращает запись из поля
    let x = xCoord.value;
    let y = yCoord.value;

    // Далее вывести координаты точки M в блоке
    // info (используйте свойство innerHTML).
    // Используйте шаблонную строку

    // Условием if проверить, попадает ли точка
    // и также вывести информацию в блоке info.
    // Используйте склеивание, добавляя к предыдущему
    // содержимому info новую
}
```

Задача 2

1. Скопируйте полную копию выполненного проекта *Координаты точки* (задание 1).
2. Добавьте в разметку еще одно текстовое поле, в которое будет вводится произвольный радиус круга.
3. Определить, попадает ли точка в сектор уже с заданным радиусом.

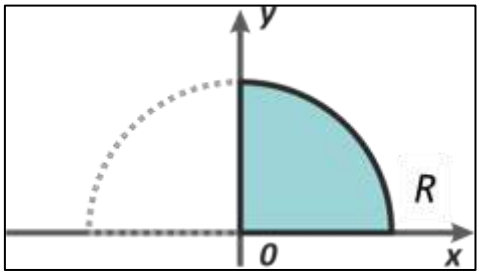


Рис. 4.55. Искомый сектор

4. Образец оформления результата изображен на рис. 4.57.

Задача 3

1. Создайте каталог *Рейтинг студента*.
2. Сгенерируйте стандартный начальный файл *index.html*.
3. Подключите таблицу стилей и скрипт. Скопируйте код в файлы, как указано в конце задания.

4. Изучите разметку. Узнайте через онлайн справочники роль дополнительных атрибутов, которые заданы текстовым полям. Также посмотрите на оформление стилей.
5. Перейдите в скрипт.
6. Пусть в текстовые поля задается ФИО и баллы студента. По нажатию на кнопку требуется определить и вывести его оценку согласно критериям из таблицы.
7. Измените по умолчанию текстовое поле на свою ФИО (образец изображен на рис. 4.58).

Таблица 4.9. Критерии оценивания студента

Баллы	**Оценка**
0-60	2
61-74	3
75-89	4
90	100

Глава 4\Рейтинг студента\index.html

```
<div class="rating">
    <h1 class="title">Карточка студента</h1>
    <div class="box">
        <p>ФИО: <input type="text"
                       id="fullname"
                       class="input"
                       value="Якубович Д.А."
                       size="30"
                       maxlength="50">
        </p>
        <p>Баллы: <input type="text"
                         id="rating"
                         class="input"
                         value="0"
                         size="3"
                         maxlength="3">
        </p>
        <button id="getData" class="start">
            Узнать оценку
        </button>
        <div id="student-mark"></div>
    </div>
</div>
```

```
<script src="code.js"></script>
```

Глава 4\Рейтинг студента\style.css

```
body {
    font: 16px Tahoma;
    background: #f8e299;
}

.box {
    margin: 10px;
}

.rating {
    background: #f8efcf;
    min-width: 320px;
    max-width: 560px;
    margin: 0 auto;
    border: 3px solid #46288d;
}

/* Заголовок */
.title {
    background: #46288d;
    color: azure;
    padding: 10px;
    margin: 0;
}

/* Текстовые поля */
.input {
    opacity: 0.6;
    color: #5f5f5f;
    font-size: 1.05em;
    border: 1px solid #b9b9b9;
    padding: 3px 7px;
    transition: 0.1s all;
}

.input:focus {
    opacity: 1.0;
    color: #d3188b;
    font-weight: bold;
    border-radius: 5px;
}
```

```css
/* Кнопка */
.start {
    display: table;
    margin: 0 auto;
    background: #fff;
    color: #46288d;
    border: 1px solid #46288d;
    border-radius: 5px;
    padding: 7px 18px;
}

.start:hover {
    background: #46288d;
    color: #fff;
}
```

Глава 4\Рейтинг студента\code.js

```js
// Получаем по id ссылки на текстовые поля, кнопку
// и область для вывода ответа
const fullname = document.getElementById("fullname");
const studentRating = document.getElementById("rating");
const getData = document.getElementById("getData");
const studentMark =
    document.getElementById("student-mark");

getData.onclick = function () {
    let fullName = fullname.value;       // ФИО
    let points = studentRating.value;    // Баллы
    let mark = 0;       // Сюда записать оценку (числом)

    // Далее с помощью расширенной инструкции if
    // получить и записать в mark корректную оценку
    // и вывести ее в блок studentMark, используя
    // свойство innerHTML
}
```

Задача 4

1. Создайте каталог *Элемент RadioButton*.
2. Сгенерируйте стандартный начальный файл *index.html*.
3. Подключите таблицу стилей и скрипт. Скопируйте код в файлы, как указано в конце задания.
4. Внимательно изучите разметку и скрипт.

5. Дополнительно можете изучит информацию по элементу Radio Button по ссылке: code.mu/ru/javascript.
6. Напишите скрипт, который в зависимости от выбранного флажка перекрашивает текстовую надпись в соответствующий цвет, а также делает его жирным.
7. Для работы вам потребуется обработать событие onchange (для каждой радио-кнопки).
8. Чтобы управлять CSS-свойствами с помощью JavaScript, используйте свойство style относительно каждой кнопки. Примеры работы с ним: doka.guide.
9. Предварительно активируйте первый флажок и оформите текст надлежащим образом, чтобы страница отображала корректный результат уже при загрузке (см. рис. 4.59).

Глава 4\Элемент RadioButton\index.html

```
<h1>Работа с флажками</h1>
<div class="box">
    <p><input type="radio"
              id="radio_red"
              name="group" checked>Красный
    </p>
    <p><input type="radio"
              id="radio_green"
              name="group">Зеленый
    </p>
    <p><input type="radio"
              id="radio_blue"
              name="group">Синий
    </p>
    <p id="text">Это текст будет менять цвет</p>
</div>

<script src="code.js"></script>
```

Глава 4\Элемент RadioButton\style.css

```
body { font: 16px Tahoma; }

h1 {
    background: #ac042e;
    color: #fff;
    font-size: 150%;
    padding: 8px;
```

```
    margin: 0;
}

.box {
    margin: 10px;
    padding: 10px;
    border: 1px solid #000000;
    border-radius: 6px;
}
```

Задача 5

1. Создайте каталог *Цифра*.
2. Сгенерируйте стандартный начальный файл *index.html*.
3. Подключите таблицу стилей и скрипт. Скопируйте код в файлы, как указано в конце задания.
4. Внимательно изучите разметку и скрипт.
5. Пользователь вводит в текстовое поле некоторую цифру или символ(ы). Вывести в абзац `digits` название этой цифры, либо текст «Это не цифра!» в противном случае. Используйте оператор `switch`.
6. Образец работы изображен на рис. 4.60.

Глава 4\Цифра\index.html

```
<h1>Цифра</h1>
<p>Введите цифру: <input type="text" id="inputDigit"
class="input" value="0" size="5" maxlength="5"></p>
<p id="digits"></p>

<script src="code.js"></script>
```

Глава 4\Цифра\style.css

```
body {
    font: 16px Tahoma;
}

h1, h2, h3 {
    border-bottom: 2px solid #000;
}
```

Глава 4\Цифра\code.js

```
// Ссылка на текстовое поле
const inputDigit =
    document.getElementById("inputDigit");
// Ссылка на абзац, в который выводим результат
const digits =
    document.getElementById("digits");

digits.innerHTML = `<p>Изначально: 0</p>`;

// Событие onchange срабатывает по окончании изменения
// в текстовом поле
inputDigit.onchange = function() {
    // this - ссылка на этот объект, т.е. в контексте
    // функции это inputDigit
    let digit = this.value;
    let result = "это не цифра!";

    // С помощью switch относительно digit определить
    // и записать в result название цифры,
    // . . .
    // Заметим что здесь в операторах case значения
    // следует брать в качестве строк: "0", "1" и т.д.
    // Это связано с тем, что значение из текстового поля
    // будет взято как строка.
    // А case сравнивает значения без приведения типа



    digits.innerHTML = `<p>Ввели цифру: ${result}</p>`;
}
```

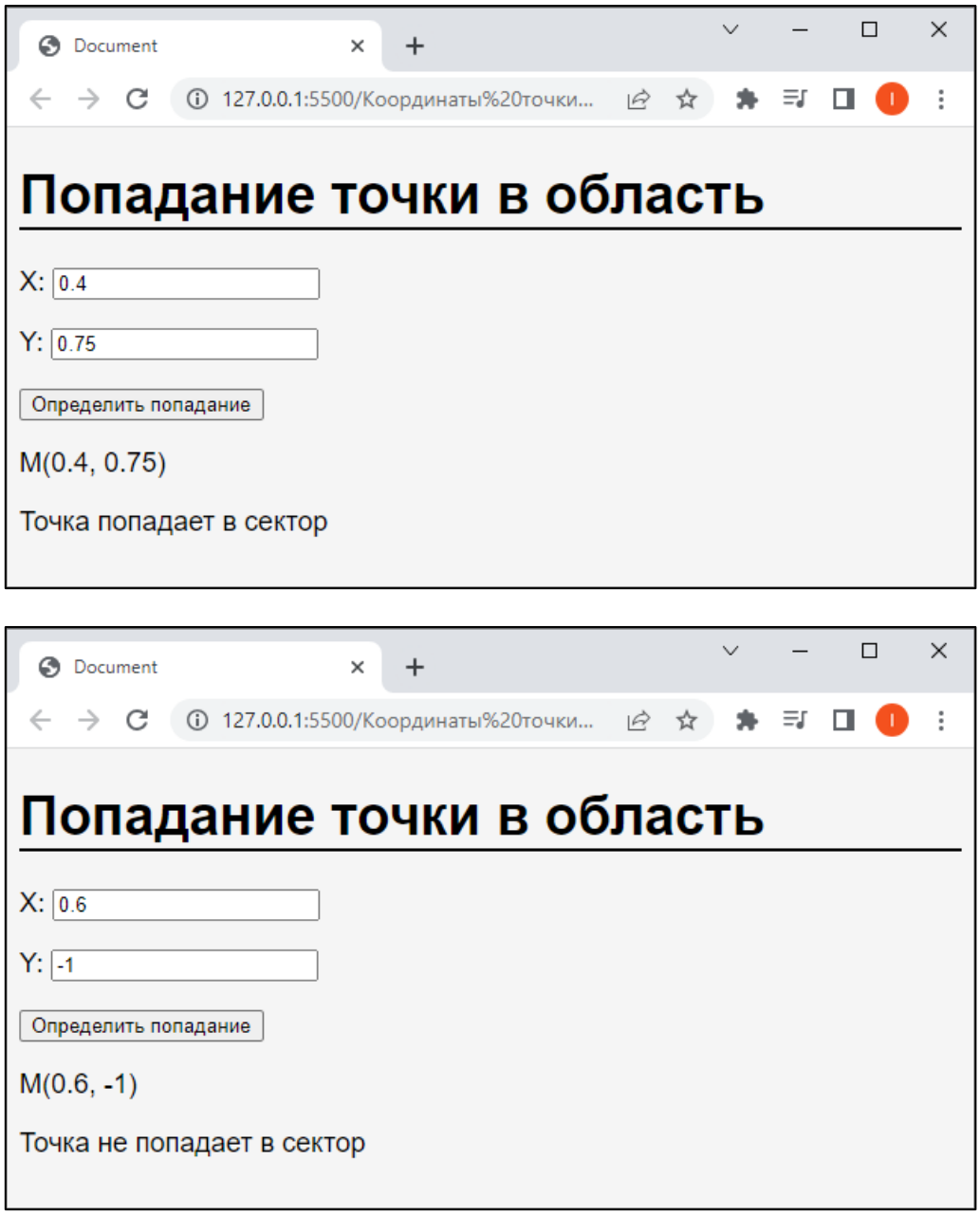


Рис. 4.56. Образец выполнения задания 1

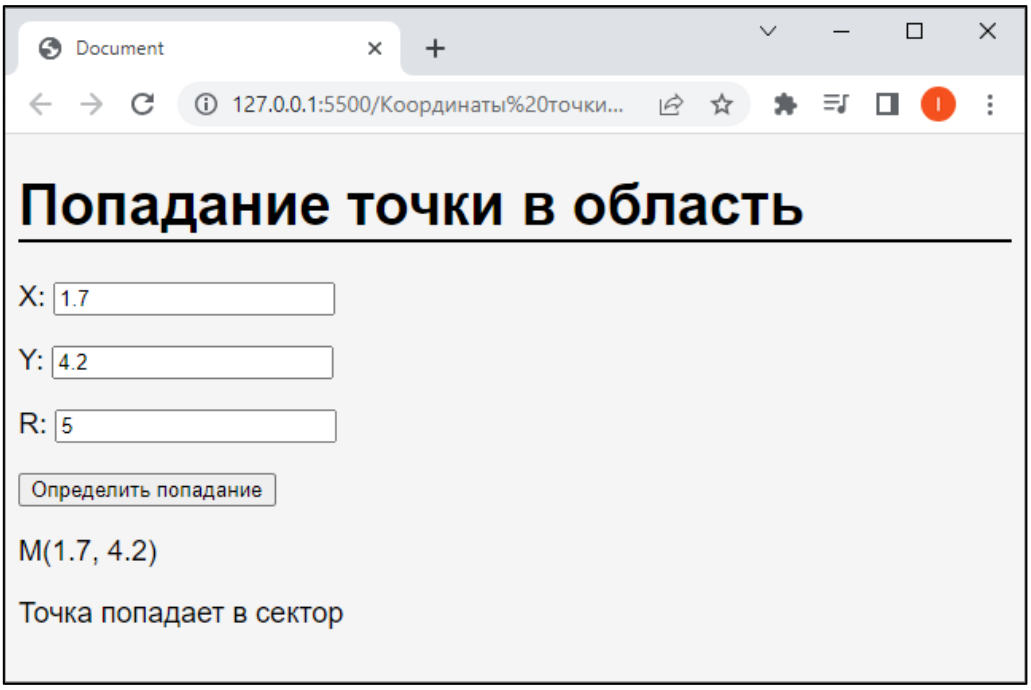


Рис. 4.57. Образец выполнения задания 2

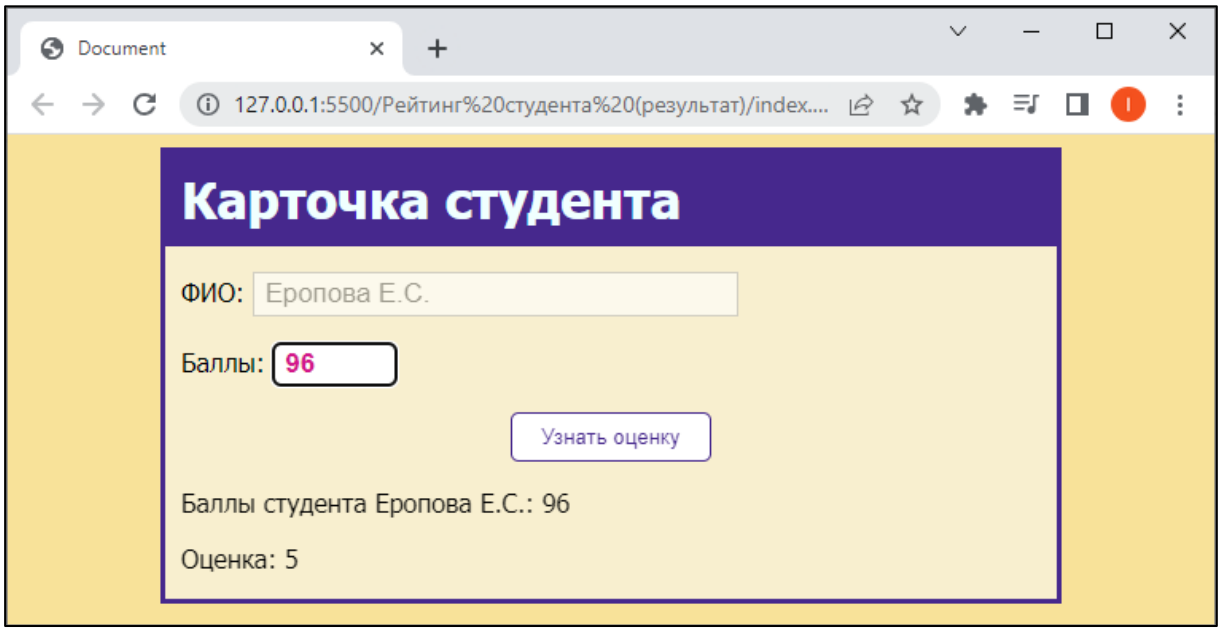


Рис. 4.58. Образец выполнения задания 3

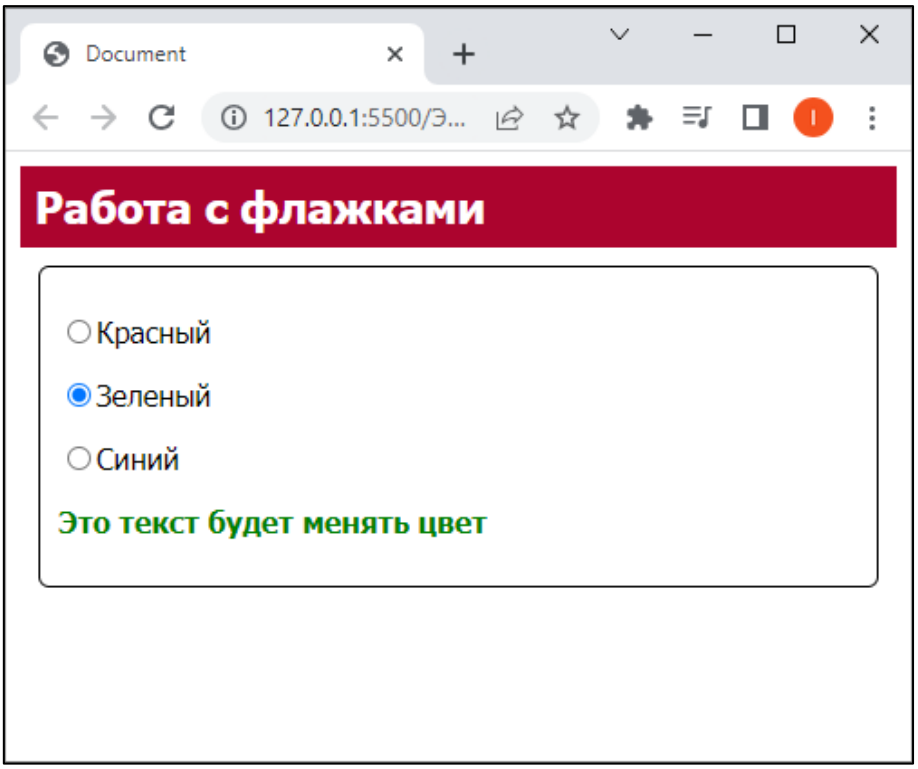


Рис. 4.59. Образец выполнения задания 4

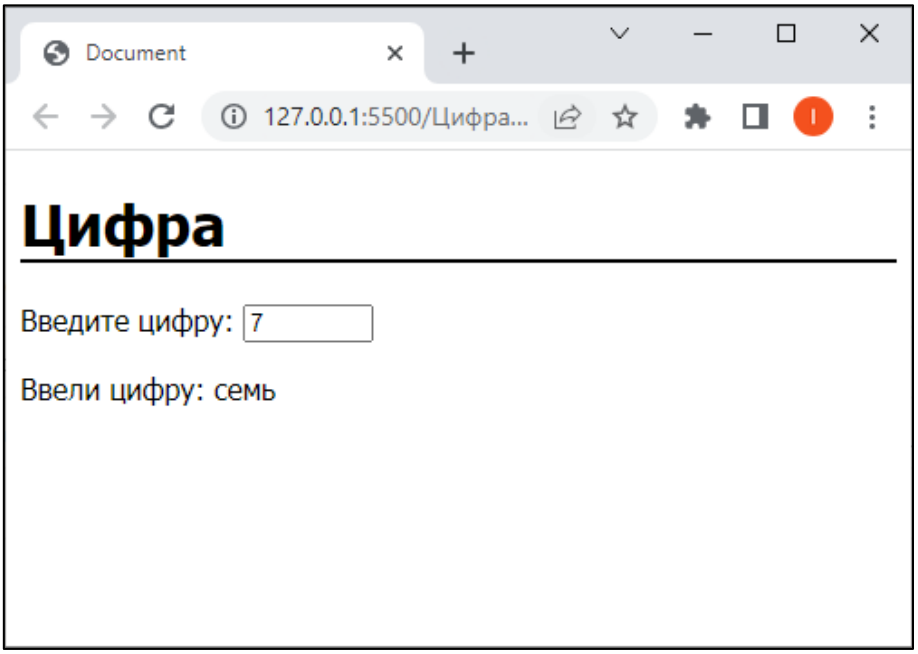


Рис. 4.60. Образец выполнения задания 5

4.5. Циклические инструкции

4.5.1. Цикл for

Синтаксис

Определение

```
for (начало; условие; шаг) {
    // тело цикла
}
```

*Инструкция **for** – это вариант **цикла с предусловием**, позволяющий инициализировать переменную перед началом цикла и указать код, выполняемый после одного витка.*

- ***начало*** *– выражение, задающее начальное значение параметру(ам) от которого зависит число повторов цикла; обычно используется, чтобы задать переменную-счётчик;*
- ***условие*** *– логическое выражение: пока оно истинно, тело цикло выполняется вновь; обычно оно зависит от переменной-счетчика;*
- ***шаг*** *– операция, выполняемая в конце очередного витка цикла; обычно используется для обновления или увеличения/уменьшения переменной-счётчика.*

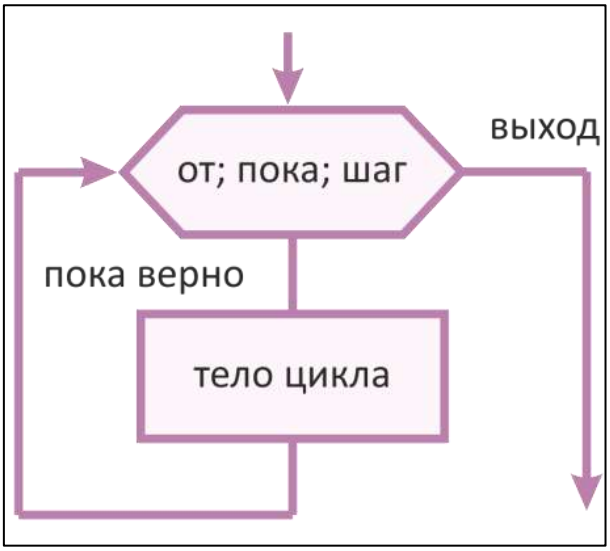


Рис. 4.61. Блок-схема цикла `for`

Рассмотрим структуру цикла на простом примере.

Выведем последовательно числа от 1 до 10 в отладочную консоль:

```
let i;

for (i = 1; i <= 10; i++) {
    console.log("Число " + i);
}
```

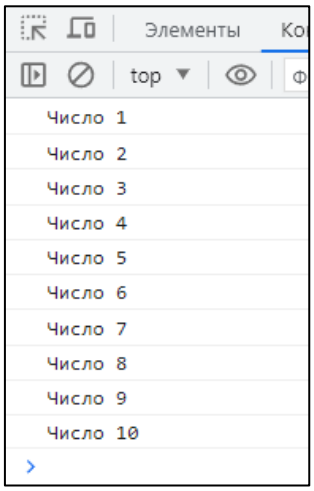


Рис. 4.62. Работа цикла

Переменная i выступает в роли счетчика. Начальное значение = 1, цикл выполняется, пока $i \leq 10$ (т.е. 10 раз). В теле цикла одна команда: выводит текст и текущее значение счетчика. После каждого витка счетчик увеличивается на 1 командой `i++`.

Локальное описание переменных

Предыдущий способ реализации цикла имеет серьезный недостаток: переменная-счетчик `i` описана до входа в цикл. Поэтому она будет доступна как внутри цикла, так и после его завершения, причем с уже измененным в цикле значением. Это небезопасный подход, поскольку переменная `i` может быть задействована в дальнейших операциях и нарушить логику работы скрипта.

Обычно переменные-счетчики в циклах используются локально и после выхода из цикла в них нет потребности.

В JavaScript хорошей практикой является описывать подобные переменные локально, в заголовочной части цикла. Приведенный пример может быть улучшен следующим образом:

```
for (let i = 1; i <= 10; i++) {
    console.log("Число " + i);
}
```

Здесь переменная-счетчик описана локально, т.е. будет доступна только внутри цикла `for`. Кроме того, в других циклах далее можно описывать счетчик с таким же именем: он всегда будет локально определенным и по факту – это совершенно другая переменная.

Вариации

Направление и шаг

Для цикла `for` условие может задаваться любым логическим выражением. Пока оно истинно, осуществляется очередная итерация.

В следующем примере переменная-счетчик уменьшается при каждом витке цикла на 6 (рис. 4.63a). Последний раз тело цикла выполнится при $i = 4$ ($i > 0$).

```
for (let i = 100; i > 0; i -= 6) {
    console.log("Число " + i);
}
```

Шаг может быть и дробным (рис. 4.63b):

```
for (let h = 0; h < 3; h += 0.25) {
    console.log("Число " + h);
}
```

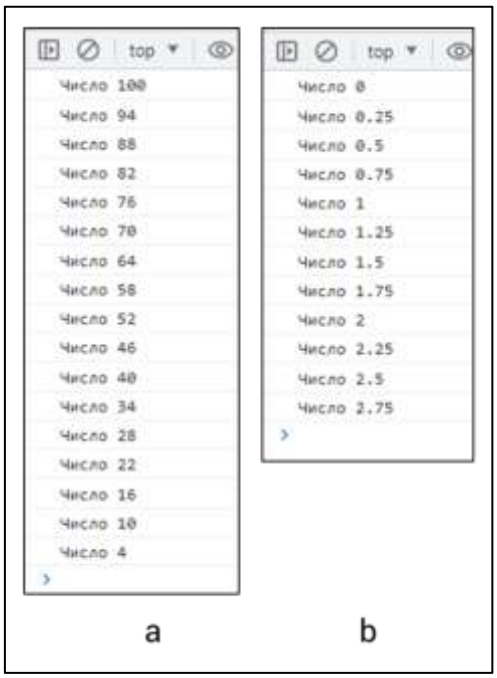


Рис. 4.63. Цикл по убыванию (a) и с дробным шагом (b)

Неполная запись

Определенные блоки заголовочной части цикла `for` возможно опускать.

Блок инициализации можно вынести до входа в цикл:

```
let i = 0;

for ( ; i <= 10; i++) {
    console.log("Число " + i);
}
```

Блок изменения шага также можно убрать. Однако в этом случае его обязательно необходимо контролировать в цикле. В данной вариации цикл `for` оформлен как цикл `while` (см. далее):

```
let i = 0;

for ( ; i <= 10; ) {
    console.log("Число " + i);
    i++;
}
```

Если из заголовка цикла убрать все блоки, то получится бесконечный цикл:

```
for ( ; ; ) {

}
```

Выход из бесконечного цикла можно осуществить с помощью команды **`break`** (см. далее). Впрочем, обычно подобные циклы рациональнее реализовать через цикл `while` или `do-while`.

Приемы работы с Visual Studio Code

*Для быстрой разметки цикла `for` в Visual Studio Code используйте сниппет **`for`**. После нажатия вставки цикла (пока не снято выделение) можно сменить название переменной счетчика (нажимайте **Tab**), рис. 4.64.*

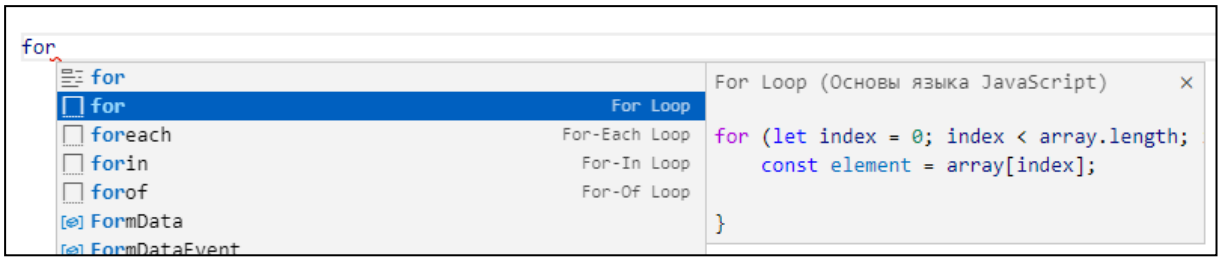


Рис. 4.64. Быстрая вставка шаблона цикла `for`

Несколько параметров

Цикл `for` – наиболее гибкий и универсальный в JavaScript. Он позволяет управлять сразу несколькими параметрами (счетчиками и внешними переменными).

В следующем примере контролируется два внутренних счетчика `i` и `j`. Один наращивается с единичным шагом, второй уменьшается на 5. Третья и уже внешняя переменная `allowedLimit` играет роль флажка. Условие зависит от всех трех переменных:

```
let allowedLimit = true;

for (let i = 0, j = 100; (i <= 100) && (j >= 0) &&
allowedLimit; i++, j -= 5) {
    const result = i * j;

    if (result < 500) {
        console.log(i, j, result);
    } else {
        allowedLimit = false;
    }
}
```

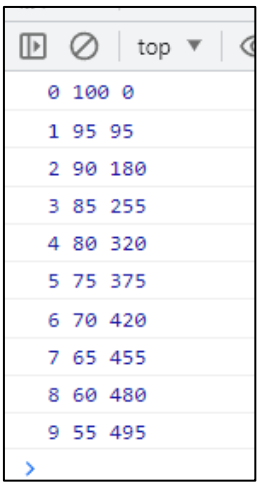


Рис. 4.65. Пример работы с несколькими параметрами цикла

Пример «Сумма ряда»

Рассмотрим простую задачу. В пределах от 0 до n найти сумму всех чисел, кратных 5 (см. рис. 4.66).

Глава 4\Цикл for\index.html

```
<h1>Сумма чисел, кратных 5</h1>
<p>Введите n:
    <input type="text"
           id="limit"
           value="1"
           size="8"
           maxlength="8">
</p>
<p id="result">Сумма кратных 5-ти от 0 до 1 = 0.</p>

<script src="code.js"></script>
```

Глава 4\Цикл for\style.css

```
body { font: 16px Tahoma; }

h1, h2, h3 { border-bottom: 2px solid #000; }

input { font-size: 1.025em; }
```

Глава 4\Цикл for\code.js

```
const last_number = document.getElementById("limit");
const result = document.getElementById("result");

last_number.onchange = function () {
    let n = this.value; // Запишем значение из
                        // текстового поля в n.
    let sum = 0;        // В эту переменную накапливаем
                        // сумму

    // Перебираем числа от 0 до n
    for (let i = 0; i <= n; i++) {
        if (i % 5 == 0) {   // Если число делится на 5
            sum += i;       // добавляем его к сумме
        }
    }

    result.innerText = `Сумма кратных 5-ти от 0 до
        ${n} = ${sum}.`;
}
```

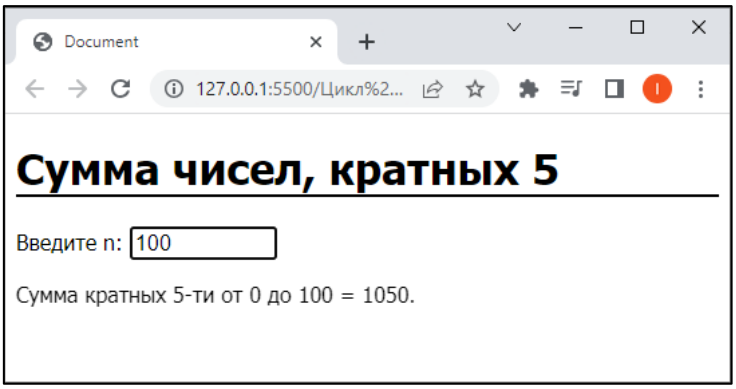


Рис. 4.66. Форма для вычисления суммы

Команда **this** внутри функции является *ссылкой на объект* `last_number`. Вместо `this` можно было указывать и название объекта.

Свойство **innerText**, в отличие `innerHTML`, принимает строку без анализа тегов, т.е. теги сохраняются как текст. В нашем случае окружать ответ тегами нет необходимости, поскольку он будет размещен внутри абзаца с идентификатором `result`.

Событие `onchange` срабатывает при изменении содержимого текстового поля, поэтому расчет суммы и вывод результата будет обновляться автоматически.

Работу цикла в примере можно оптимизировать. Проверять кратность на 5 необязательно. Достаточно начать с нуля и увеличивать счетчик на 5:

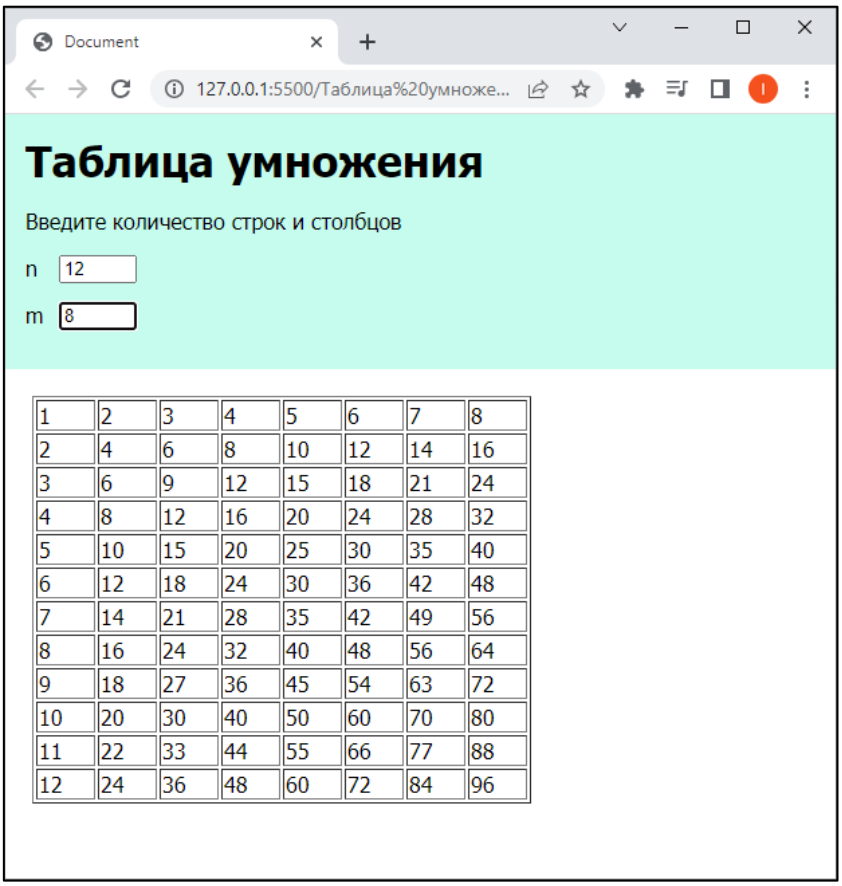
Глава 4\Цикл for\code.js

```
const last_number = document.getElementById("limit");
const result = document.getElementById("result");

last_number.onchange = function() {
    let n = this.value;
    let sum = 0;

    for (let i = 0; i <= n; i += 5) {
        sum += i;
    }

    result.innerText = `Сумма кратных 5-ти от 0 до
        ${n} = ${sum}.`;
}
```

Это существенно оптимизирует цикл: нет проверки условия и не перебираются лишние числа.

Пример «Таблица умножения»

Реализуем скрипт, отображающий таблицу умножения по заданному числу строк и колонок:



1	2	3	4	5	6	7	8
2	4	6	8	10	12	14	16
3	6	9	12	15	18	21	24
4	8	12	16	20	24	28	32
5	10	15	20	25	30	35	40
6	12	18	24	30	36	42	48
7	14	21	28	35	42	49	56
8	16	24	32	40	48	56	64
9	18	27	36	45	54	63	72
10	20	30	40	50	60	70	80
11	22	33	44	55	66	77	88
12	24	36	48	60	72	84	96

Рис. 4.67. Таблица умножения

Создадим каталог *Таблица умножения*. Добавим стандартный файл *index.html* с базовой разметкой и зададим код для тела веб-страницы:

Глава 4\Таблица умножения\index.html

```
<div class="container">
    <h1>Таблица умножения</h1>
    <p>Введите количество строк и столбцов</p>
    <p>n <input type="text"
               id="rows"
               class="input-rows"
               value="1"
               size="3"
               maxlength="3">
    </p>
```

```
    <p>m <input type="text"
               id="columns"
               class="input-rows"
               value="1"
               size="3"
               maxlength="3">
    </p>
    <div id="result"></div>
</div>
<div id="mul-table" class="mul-table"></div>

<script src="code.js"></script>
```

Также подключим в преамбуле стилевой файл:

Глава 4\Таблица умножения\style.css

```
body {
    font: 16px Tahoma;
    margin: 0;
}

h1, h2, h2 { margin: 0; }

.container {
    position: relative;
    background: #c5fcee;
    padding: 15px;
}

.input-rows,
.input-columns {
    position: absolute;
    left: 40px;
}

.mul-table { padding: 20px; }

.quad {
    font-weight: bold;
    background: #a8062f;
    color: #fff;
}
```

Для начала зададим ссылки на блок `mul-table` (в него разместим таблицу), текстовые поля `rows` и `columns` (в них вводятся размеры таблицы). Используем заданные тегам идентификаторы:

Глава 4\Таблица умножения\code.js

```
const result = document.getElementById("mul-table");
const rows = document.getElementById("rows");
const columns = document.getElementById("columns");
```

Сделаем так, чтобы таблица автоматически обновлялась при изменении любого из полей. Иными словами требуется, чтобы текстовые поля `rows` и `columns` могли реагировать на одно и то же событие и перестраивать таблицу.

Для этого воспользуемся методом **`addEventListener()`**. Первый параметр указывает название события, на которое подписывается объект. Второй – название функции, которая будет обрабатывать это событие.

Оба текстовых поля ссылаются на одну функцию обработчик `drawTable()`, опишем ее ниже:

Глава 4\Таблица умножения\code.js

```
const result = document.getElementById("mul-table");
const rows = document.getElementById("rows");
const columns = document.getElementById("columns");

rows.addEventListener('change', drawTable);
columns.addEventListener('change', drawTable);

function drawTable() {

}
```

Реализуем функцию `drawTable()`. Записываем введенные в поля значения в отдельные переменные `n` и `m`:

Глава 4\Таблица умножения\code.js

```
function drawTable() {
    let n = rows.value;
    let m = columns.value;
}
```

Код разметки таблицы будем накапливать в текстовую переменную `tableCode`. Сразу откроем тег и закроем его в конце, чтобы не забыть:

Глава 4\Таблица умножения\code.js

```
function drawTable() {
    let n = rows.value;
    let m = columns.value;

    let tableCode = "<table border='1'>";

    // Здесь будет код разметки таблицы

    tableCode += "</table>";
}
```

Нам понадобится два цикла:

- внешний для формирования строки (`<tr></tr>`);
- внутренний – для формирования ячеек строки (`<td></td>`), в которых будем писать соответствующее произведение:

Глава 4\Таблица умножения\code.js

```
function drawTable() {
    let n = rows.value;
    let m = columns.value;

    let tableCode = "<table border='1'>";

    for (let i = 1; i <= n; i++) {
        for (let j = 1; j <= m; j++) {
        }
    }

    tableCode += "</table>";
}
```

Каждая строка должна быть помещена в тег `<tr></tr>`:

Глава 4\Таблица умножения\code.js

```
function drawTable() {
    let n = rows.value;
    let m = columns.value;

    let tableCode = "<table border='1'>";
```

```
    for (let i = 1; i <= n; i++) {
        tableCode += "<tr>";

        for (let j = 1; j <= m; j++) {
        }

        tableCode += "</tr>";
    }

    tableCode += "</table>";
}
```

В i-й строке будет располагаться m ячеек `<td></td>`: произведения чисел $j = 1,2, \ldots m$ на i. Также зададим фиксированную ширину колонкам:

Глава 4\Таблица умножения\code.js

```
function drawTable() {
    let n = rows.value;
    let m = columns.value;

    let tableCode = "<table border='1'>";

    for (let i = 1; i <= n; i++) {
        tableCode += "<tr>";

        for (let j = 1; j <= m; j++) {
            tableCode += "<td width='40'>" +
                         i * j +
                         "</td>";
        }

        tableCode += "</tr>";
    }

    tableCode += "</table>";
}
```

Код таблицы сформирован. Копируем текст разметки в искомый блок `result`.

Глава 4\Таблица умножения\code.js

```
function drawTable() {
    let n = rows.value;
    let m = columns.value;

    let tableCode = "<table border='1'>";

    for (let i = 1; i <= n; i++) {
        tableCode += "<tr>";

        for (let j = 1; j <= m; j++) {
            tableCode += "<td width='40'>" +
                          i * j +
                         "</td>";
        }

        tableCode += "</tr>";
    }

    tableCode += "</table>";

    result.innerHTML = tableCode;
}
```

Следует учитывать, что пользователь вместо чисел может ввести буквы. Поэтому не лишним будет предварительно проверить значения и при необходимости их скорректировать.

Для этого опишем функцию `inputCorrenction`, которой передаем ссылку на проверяемое текстовое поле. Далее применим ее для каждого поля:

Глава 4\Таблица умножения\code.js

```
// Получаем ссылки (см выше)
// Подписываемся на события (см выше)

function drawTable() {
    let n = inputCorrection(rows);
    let m = inputCorrection(columns);

    // Далее остальной код на формирование таблицы
}
```

```
function inputCorrection(field) {
    if (isNaN(field.value)) {
        field.value = 1;
        return 1;
    }

    return Math.abs(field.value);
}
```

В случае, если в текстовое поле введено нечисловое значение, возьмем его единицей. Иначе это число, возвратим его модуль (на случай, если любопытный пользователь захочет проверить работоспособность и для отрицательного числа).

4.5.2. Циклы while и do-while

Синтаксис циклов

Определение

```
while (условие) {
    // тело цикла
}
```

Инструкция ***while*** *организует* ***цикл с предусловием****. Цикл выполняется, пока условие истинно.*

```
do {
    // тело цикла
} while (условие);
```

Инструкция ***do-while*** *организует* ***цикл с постусловием****. Цикл выполняется хотя-бы один раз и далее, пока условие истинно.*

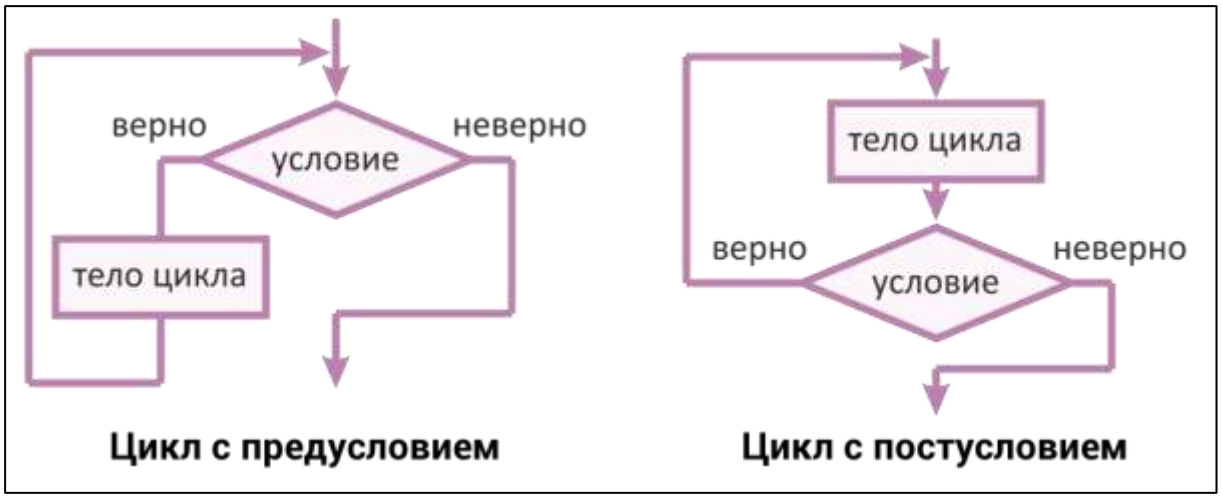


Рис. 4.68. Блок-схемы циклов `while` и `do-while`

Пример «Каноническое разложение числа»

Описание

Реализуем веб-страничку с формой, которая осуществляет разложение заданного натурального $n \geq 2$ на простые множители с указанием их кратности.

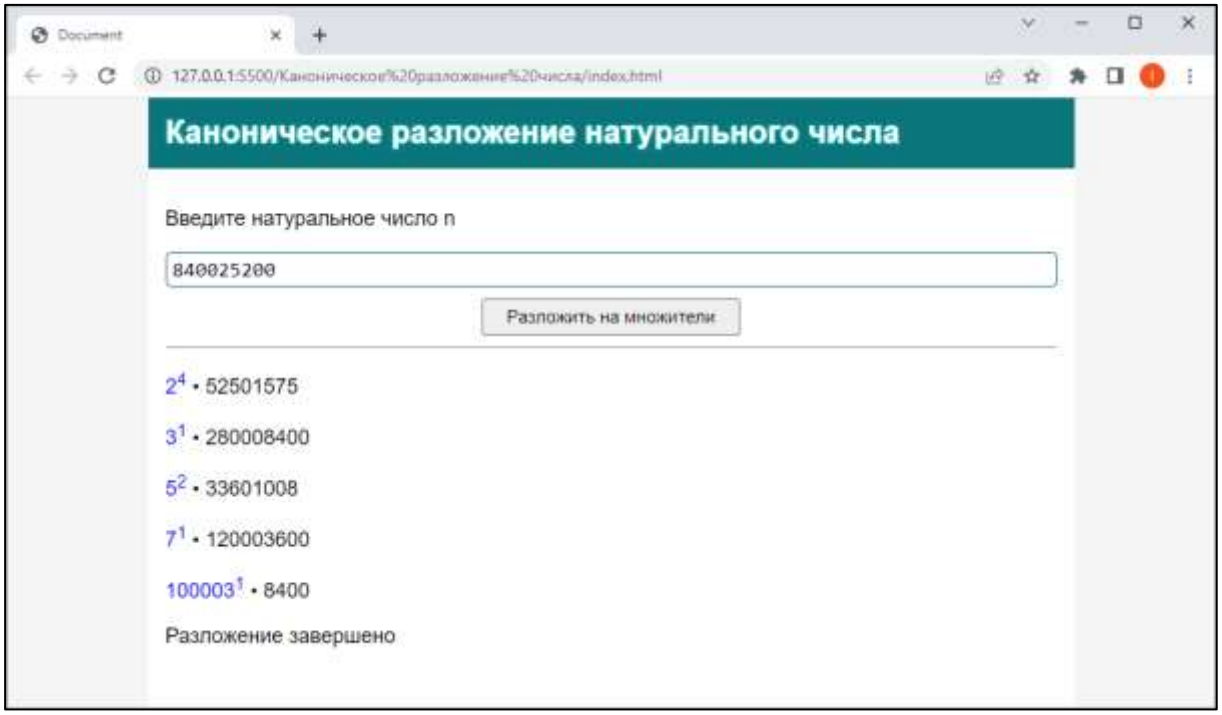


Рис. 4.69. Каноническое разложение на простые множители

Создадим одноименный каталог *Каноническое разложение числа*, подготовим разметку и стилевой файл.

Внешний дизайн

Глава 4\Каноническое разложение числа\index.html

```
<main class="content">
    <h1 class="content__title">
        Каноническое разложение натурального числа
    </h1>
    <div class="content__box">
        <p>Введите натуральное число n</p>
        <input type="text"
               id="setNumber"
               class="num-input"
               value="2">
        <button id="decomposition" class="start-btn">
            Разложить на множители
        </button>
        <hr>
        <div id="info"></div>
    </div>
</main>

<script src="code.js"></script>
```

Глава 4\Каноническое разложение числа\style.css

```
*  { box-sizing: border-box; }

body {
    background: whitesmoke;
    font-family: Arial, Helvetica, sans-serif;
    font-size: 18px;
    margin: 0;
}

.content {
    background: #ffffff;
    max-width: 800px;
    margin: 0 auto;
    height: 100vh;
}

.content__title {
    background: #0b757c;
    color: #ffffff;
    font-size: 150%;
    padding: 15px;
```

```
        margin: 0;
        text-shadow: 0 0 3px #1cfdf2;
    }

    .content__box { padding: 15px; }

    .num-input {
        display: block;
        width: 100%;
        font: 1.0em Consolas;
        border: 1px solid #0b757c;
        border-radius: 5px;
        padding: 3px 5px;
        margin-bottom: 10px;
    }

    .start-btn {
        display: block;
        margin: 0 auto;
        font-size: 0.85em;
        padding: 5px 20px;
    }
```

Алгоритм решения

Перед реализацией скрипта прокомментируем алгоритм разложения числа.

Будем сокращать число n на предполагаемые делители до тех пор, пока оно не станет равным 1.

В качестве первого делителя возьмем минимальное простое число – 2. Перебор делителей организуем с помощью цикла.

Поскольку, вообще говоря, делитель может входит в число в некоторой степени, то имеет смысл делить число n на текущий делитель до тех пор, пока это возможно. Деление также организуем в цикле (внутри основного, который перебирает возможные делители). В этом же цикле переменная счетчик будет подсчитывать степень делителя.

Наконец, если счетчик степени ненулевой (т.е. предполагаемый делитель является таковым), то выводим его вместе со степенью.

Далее переходим к проверке следующего кандидата в делители числа. Процесс повторится до наибольшего простого делителя числа.

Несмотря на простой циклический перебор предполагаемых делителей, предложенный алгоритм является достаточно эффективным:

в процессе число сокращается на простые делители, тем самым уменьшается количество возможных шагов.

Однако время поиска существенно увеличится, если:

- n само является простым (цикл повторится максимальное число раз);
- простые делители достаточно большие (как и само n) и входят в число лишь в первой степени.

Программная реализация

Для начала задаем ссылки на поле ввода числа `setNumber`, кнопку начала расчета `decomposition` и блок для вывода результата `info`.

В этом примере для разнообразия продемонстрирован еще один метод для работы с DOM-узлами – **`querySelector()`**. Он позволяет получить ссылку на указанный селектор, как если бы мы его описывали в CSS (т.е. с классами, идентификаторами и даже контекстным вложением). Иными словами, этот метод более универсальный, чем `getElementById()` и не ограничивается только обращением по идентификаторам.

Глава 4\Каноническое разложение числа\code.js

```
const number = document.querySelector("#setNumber");
const start = document.querySelector("#decomposition");
const info = document.querySelector("#info");
```

```
<p>Введите натуральное число n</p>
<input type="text" id="setNumber" class="num-input" value="2">
<button id="decomposition" class="start-btn">Разложить на множители</button>
<hr>
<div id="info"></div>
```

Рис. 4.70. Обращение по селектору-идентификатору

Подпишем кнопку на функцию обработки клика.

Каждое нажатие кнопки должно осуществить разложение числа и вывести эту разметку в блок. Выводить результат будем построчно, поскольку весь процесс разложения на множители больших чисел может занимать значительное время.

Новый расчет потребует очистит предыдущий вывод. Для этого опишем процедуру `clearCode()`, которая очищает разметку внутри указанного блока вывода `info`:

Глава 4\Каноническое разложение числа\code.js

```
const number = document.querySelector("#setNumber");
const start = document.querySelector("#decomposition");
const info = document.querySelector("#info");

start.onclick = function(){
    clearCode(info);

}

function clearCode(obj) {
    obj.innerHTML = "";
}
```

Считываем число из текстового поля в переменную. Подстрахуемся функцией `parseInt()`, которая отбросит дробную часть, если пользователь введет вещественное число.

Дополнительно сделаем копию числа (понадобится для подсчета парного множителя):

Глава 4\Каноническое разложение числа\code.js

```
start.onclick = function(){
    clearCode(info);

    let n = parseInt(number.value);
    let n_copy = n;

}
```

Первым простым делителем считаем 2. Цикл перебора повторяем, пока n не сократился вплоть до 1:

Глава 4\Каноническое разложение числа\code.js

```
start.onclick = function(){
    clearCode(info);

    let n = parseInt(number.value);
    let n_copy = n;
    let divisor = 2;

    while (n > 1) {

    }
}
```

Во внутреннем цикле подсчитываем кратность делителя (если является таковым) и парный ему:

Глава 4\Каноническое разложение числа\code.js

```
start.onclick = function(){
    clearCode(info);

    let n = parseInt(number.value);
    let n_copy = n;

    let divisor = 2;

    while (n > 1) {
        let degree = 0;   // Счетчик степени делителя

        // Пока и если делится, то:
        while (n % divisor == 0) {
            n /= divisor; // сокращаем n на множитель,
            degree++;     // наращиваем степень множителя
        }

        // Считаем делитель^k, парный множитель
        // и вставляем в разметку
        if (degree > 0) {
            let factor = Math.pow(divisor, degree);
            let pair_divisor =
                Math.round(n_copy / factor);
            info.innerHTML +=
                `<p>
                    <span style='color: blue'>
                        ${divisor}<sup>${degree}</sup>
                    </span>
                    •
                    ${pair_divisor}
                </p>`;
        }

        divisor++;
    }

    // Сообщаем о завершении расчета
    info.innerHTML += `<p>Разложение завершено</p>`;
}
```

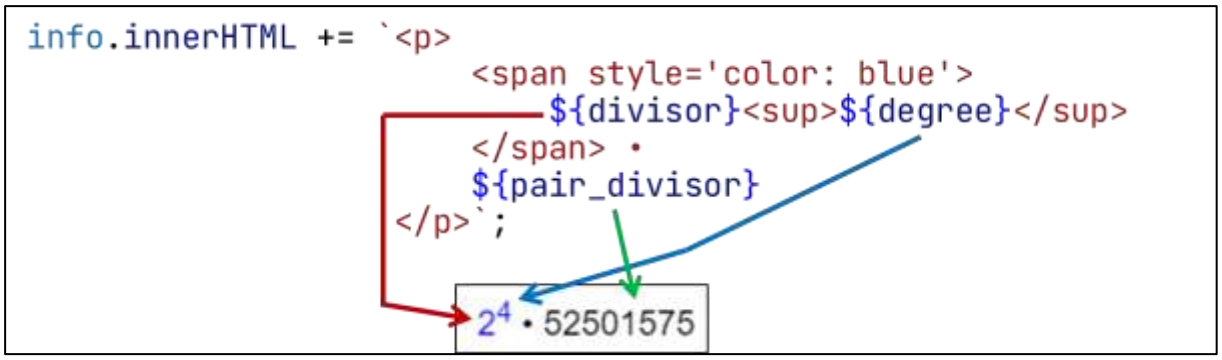


Рис. 4.71. Форматирование вывода

4.5.3. Общие возможности циклов

Прерывание циклов

1. Прерывание цикла

Команда **`break`** прерывает выполнение цикла. При вложении циклов действует относительно ближайшего.

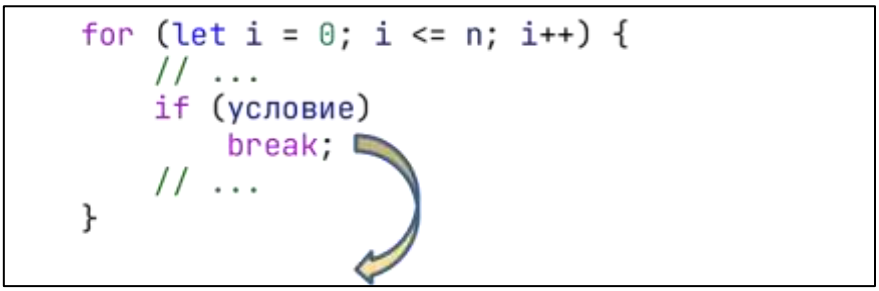


Рис. 4.72. Оператор `break`

2. Прерывание итерации цикла

Команда **`continue`** прерывает выполнение текущего витка цикла и переходит к следующему.

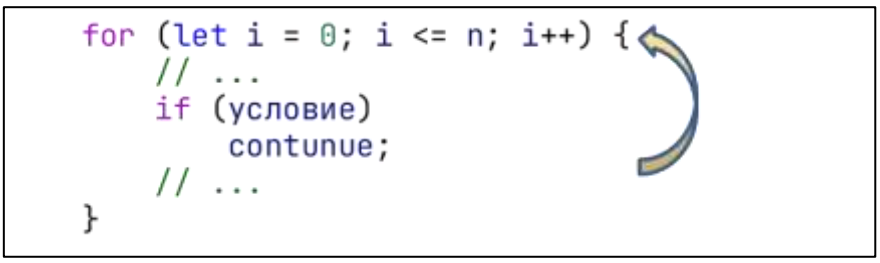


Рис. 4.73. Оператор `continue`

Переход на метку

Оба оператора также могут переходить на указанную метку:

```
exitFor:
for (let i = 0; i < n; i++) {
    if (условие)
        break exitFor;
}
```

Это полезно знать!

Операторы ***break*** *и* ***continue*** *работают во всех других типах циклов JavaScript.*

О циклах в целом

Вообще говоря, циклы `for`, `while` и `do-while` взаимозаменяемы.

Но для каждой ситуации лучше подбирать тот цикл, который дает более лаконичный и хорошо читаемый код.

Чаще всего удобно использовать:

- цикл `for`, если заранее известно количество витков или работаем с индексированными объектами;
- циклы `while` и `do-while`, если число витков заранее неизвестно и зависит от одного или более условий.

Вопросы для самопроверки

1. Опишите основные части цикла `for`.
2. Какие вариации цикла `for` возможны?
3. Чем отличается цикл `while` от `do-while`?
4. Каким образом можно организовать бесконечный цикл и как из него выйти?
5. Возможно ли реализовать из цикла `for` цикл `while`? А наоборот?
6. Опишите операторы для прерывания операций в циклах.
7. Какой из циклов наиболее универсален и почему?

Практикум

Задание 1

1. Создайте новый каталог *Четные и нечетные числа*.
2. Подготовьте типовой файл разметки, подключите таблицы стилей.
3. Скопируйте приведенный в конце занятия код.
4. Допишите скрипт. Требуется вывести четные и нечетные числа в пределах от 1 до n. Для четных код уже реализован.
5. Результат изображен на рис. 4.74.

Глава 4\Четные и нечетные числа\index.html

```
<h1>Четные и нечетные числа</h1>
<p>Введите n: <input type="text" id="limit" value="1"
size="5" maxlength="5"></p>
<div id="result" class="result"></div>

<script src="code.js"></script>
```

Глава 4\Четные и нечетные числа\style.css

```
body { font: 16px Tahoma; }

h1, h2, h3 { border-bottom: 2px solid #000; }

.answer {
    font: bold 1.05em 'Courier New', Courier, monospace;
    color: #1d59b4;
}
```

Глава 4\Четные и нечетные числа\code.js

```
// Получаем ссылки на текстовое поле и блок <div>
// для вывода результата
const last_number = document.getElementById("limit");
const result = document.getElementById("result");

// Обрабатываем изменение в текстовом поле
last_number.onchange = function() {
    // Получаем значение из текстового поля last_number
    let n = this.value;
    // В эту переменную последовательно накапливаем
    // разметку ответа
    let code = "<h2>Четные:</h2>";
    // Этот класс оформит числа отдельно
```

```
    code += "<p class='answer'>";

    for (let i = 2; i <= n; i += 2) {
        code += i + " | ";
    }

    code += "</p>";

    // Далее продолжите разметку code для нечетных
    // ...

    // Записываем полученный код в блок ответа
    result.innerHTML = code;
}
```

Задание 2

1. Создайте каталог *Таблица умножения*.
2. Воспроизведите пример, рассмотренный в п. 4.5.1.
3. Доработайте проект. Требуется выделить ячейки на диагонали. Вам необходимо к этим ячейкам подцепить класс `quad` (он уже описан в CSS).
4. Во внутреннем цикле потребуется конструкция `if-else`: для диагональных ячеек подцепить этот класс, а для недиагональных вывести как ранее. (Как определить, что ячейка стоит на диагонали?)
5. Образец оформления изображен на рис. 4.75.

Задание 3

1. Создайте каталог *Табулирование функции*.
2. Создайте типовой файл *index.html*, подключите внешнюю таблицу CSS-стилей.
3. Изучите приведенный ниже код разметки: каким образом размечаются текстовые поля с прокруткой и какие атрибуты для них задаются.
4. Также обратите внимание, как оформляются четные и нечетные строки таблицы (см. CSS-код).
5. Требуется построить таблицу для функции $(f(x) + g(x))/2$ (заданы в скрипте) на отрезке $[a, b]$ с шагом h.
6. В реализации используйте цикл `while`. Допишите код скрипта, следуя инструкциям в комментариях.
7. Образец оформления изображен на рис. 4.76.

Глава 4\Табулирование функции\index.html

```
<div class="container">
    <div class="input">
        <h1>Табулирование функции</h1>
        <p>Введите концы отрезка [a,b] и шаг h</p>
        <p>
            a <input type="number"
                     id="left-side"
                     class="field"
                     value="0"
                     min="-100"
                     max="100"
                     step="1">
        </p>
        <p>
            b <input type="number"
                     id="right-side"
                     class="field"
                     value="1"
                     min="-100"
                     max="100"
                     step="1">
        </p>
        <p>
            h <input type="text"
                     id="step"
                     class="field"
                     value="0.25"
                     size="10"
                     maxlength="10">
        </p>
        <button id="tabulate" class="start-btn">
            Рассчитать таблицу
        </button>
    </div>
</div>

<div class="container">
    <div id="result" class="result"></div>
</div>

<script src="code.js"></script>
```

Глава 4\Табулирование функции\style.css

```
body { font: 16px Tahoma; }
body {
    font: 16px Tahoma;
    margin: 0;
}

h1, h2, h2 {
    margin: 0;
}

.container {
    position: relative;
    width: 560px;
    margin: 0 auto;
}

.input {
    background: #c2eeff;
    padding: 15px;
}

.field {
    position: absolute;
    left: 40px;
}

.start-btn {
    display: block;
    margin: 0 auto;
    font-size: 0.85em;
    padding: 5px 20px;
}

.result {
    padding: 20px;
}

.result-table {
    border-collapse: collapse;
    border: 3px solid #000000;
    margin: 0 auto;
}
```

```
.result-table th,
.result-table td {
    padding: 5px;
}

/* Нечетные строки */
.result-table tr:nth-child(2n+1) {
    background: #c8ffd1;
}

/* Нечетные колонки */
.result-table th:nth-child(1) {
    width: 70px;
}

/* Четные колонки */
.result-table th:nth-child(2) {
    width: 120px;
}
```

Глава 4\Табулирование функции\code.js

```
// Ссылки на поле a, b, h, кнопку и блок вывода таблицы
const leftSide =
    document.getElementById("left-side");
const rightSide =
    document.getElementById("right-side");
const step =
    document.getElementById("step");
const tabulation =
    document.getElementById("tabulate");
const result =
    document.getElementById("result");

tabulation.onclick = function() {
    // Запрашиваем данные с полей, конвертируем их
    // в числа
    let a = parseFloat(leftSide.value);
    let b = parseFloat(rightSide.value);
    let h = parseFloat(step.value);

    // Открываем таблицу (оформляется классом, который
    // описан в CSS)
    let tableCode =
        "<table class='result-table'>";
```

```
    // Рисуем заголовочную строку из двух ячеек
    tableCode += "<tr><th>x</th><th>f(x)</th></tr>";

    let x = a;          // Начальная узловая точка

    // Далее с помощью while построчно вычислите
    // значение функции в точках x0, x, ..., xn <= b
    // и выставьте в разметку
    // . . . . .


    tableCode += "</table>";        // Закрываем таблицу

    // Вставляем разметку в блок
    result.innerHTML = tableCode;
}

// Математические функции f(x) и g(x)
function f(x) {
    return Math.sqrt(x*x + 1);
}

function g(x) {
    return 10 * Math.abs(Math.sin(x));
}
```

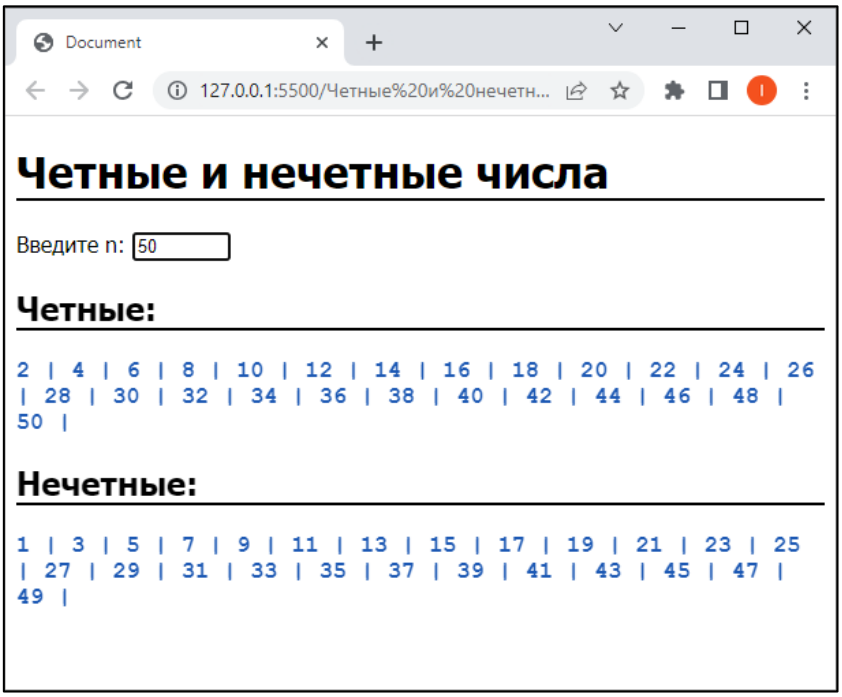
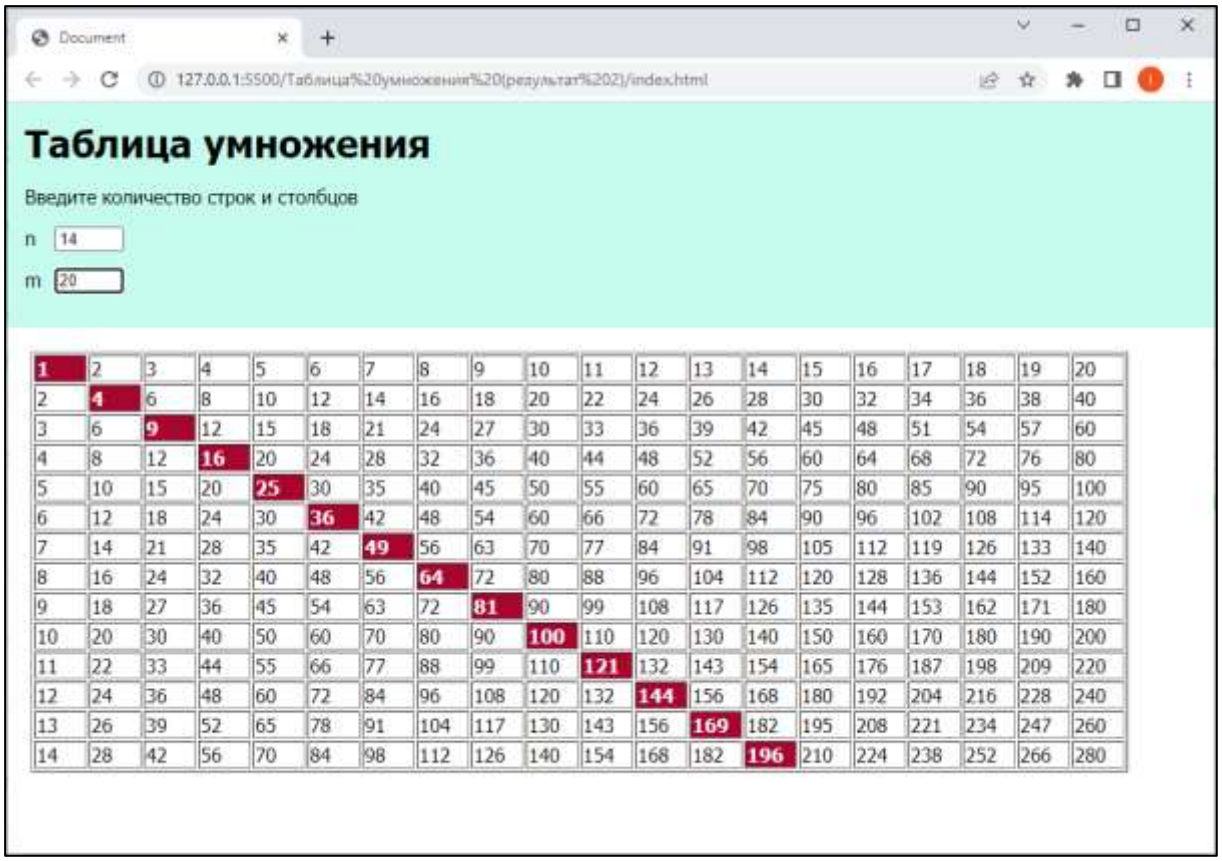
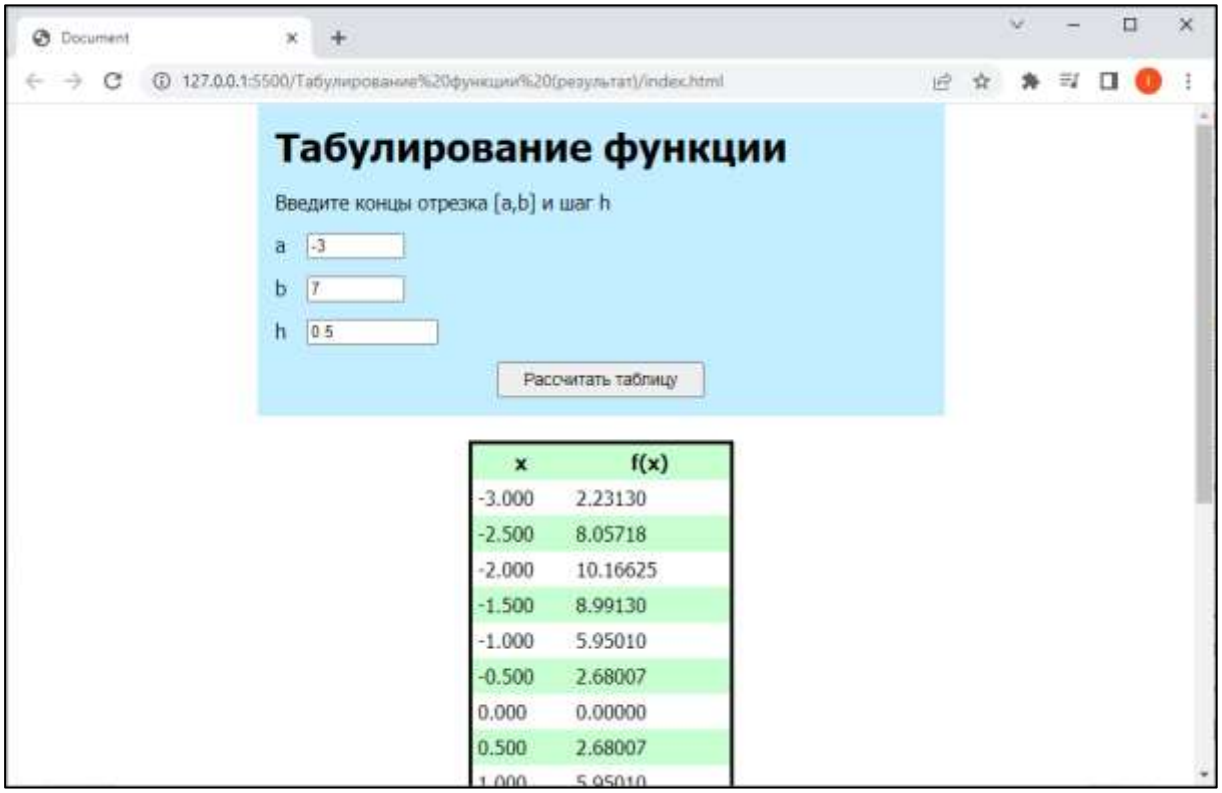


Рис. 4.74. Образец выполнения задания 1



1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
2	4	6	8	10	12	14	16	18	20	22	24	26	28	30	32	34	36	38	40
3	6	9	12	15	18	21	24	27	30	33	36	39	42	45	48	51	54	57	60
4	8	12	16	20	24	28	32	36	40	44	48	52	56	60	64	68	72	76	80
5	10	15	20	25	30	35	40	45	50	55	60	65	70	75	80	85	90	95	100
6	12	18	24	30	36	42	48	54	60	66	72	78	84	90	96	102	108	114	120
7	14	21	28	35	42	49	56	63	70	77	84	91	98	105	112	119	126	133	140
8	16	24	32	40	48	56	64	72	80	88	96	104	112	120	128	136	144	152	160
9	18	27	36	45	54	63	72	81	90	99	108	117	126	135	144	153	162	171	180
10	20	30	40	50	60	70	80	90	100	110	120	130	140	150	160	170	180	190	200
11	22	33	44	55	66	77	88	99	110	121	132	143	154	165	176	187	198	209	220
12	24	36	48	60	72	84	96	108	120	132	144	156	168	180	192	204	216	228	240
13	26	39	52	65	78	91	104	117	130	143	156	169	182	195	208	221	234	247	260
14	28	42	56	70	84	98	112	126	140	154	168	182	196	210	224	238	252	266	280

Рис. 4.75. Образец выполнения задания 2



x	f(x)
-3.000	2.23130
-2.500	8.05718
-2.000	10.16625
-1.500	8.99130
-1.000	5.95010
-0.500	2.68007
0.000	0.00000
0.500	2.68007
1.000	5.95010

Рис. 4.76. Образец выполнения задания 3

4.6. Циклы и массивы

4.6.1. Массивы в JavaScript

Описание и особенности массивов

Определение

Массив *– это упорядоченная коллекция элементов, в которой каждый элемент имеет индекс. Индексация элементов в JS начинается с нуля.*

Массивы наряду с объектами являются важнейшими элементами JavaScript. Поэтому изучению их основных возможностей уделим особое внимание.

Перечислим основные особенности массивов в JavaScript:

1. *Элементы массива могут иметь разные типы*. Во многих языках программирования массив – это набор элементов одного типа. В JavaScript этого ограничения нет: элементы могут быть любого типа, в т.ч. объектами или даже массивами элементов.
2. *Элементы массива индексируются, начиная с нуля*.
3. *Массивы в JavaScript являются динамическими*, т.е. в массив можно добавлять (удалять) элементы в любой момент времени.
4. *Массивы в JavaScript являются объектами*. Это открывает достаточно интересные возможности по их использованию.

Описание и инициализация массива осуществляется следующим образом:

```
let arr = [2, 4, -10, 12, 56];
```

К элементам массива можно обращаться по индексу (не забывайте, что индексирование начинается с нуля):

```
let first = arr[0];
```

Индекс также может храниться в переменной:

```
let i = 3;
arr[i] = 2023;
```

Примеры работы с массивами

Создание и операции с индексами

Для создания пустого массива используются пустые квадратные скобки:

```
let test= [];
```

Добавим в массив элемент, равный 1 и 10:

```
test[0] = 1;
test[1] = 10;

console.log(test);
```

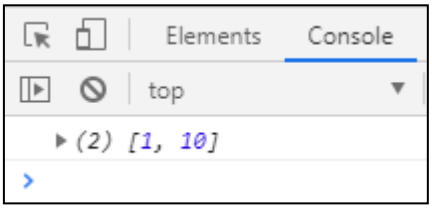


Рис. 4.77. Вывод массива в отладочную консоль браузера

Если заполнить элементы неупорядоченно, то на пропущенные элементы все равно выделяется память, а они считаются значением `undefined` (до того, как не будут записаны):

```
let test= [];

test[0] = 1;
test[1] = 10;
test[5] = 20;

console.log(test);
```

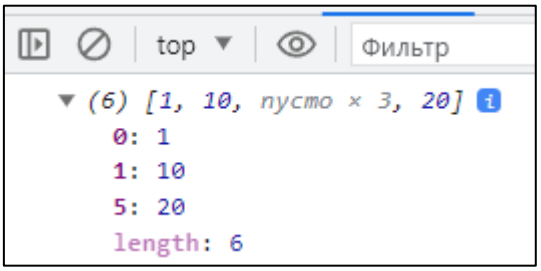


Рис. 4.78. Пропуски в массиве

Создадим массив из 5-ти элементов. Свойство **length** возвращает число элементов массива (автоматически обновляется при его изменении). В консоль отладки выводим первый и последний элементы:

```
let arr = [2, 4, -10, 12, 56];
let n = arr.length;

console.log(arr[0]);
console.log(arr[n-1]);
```

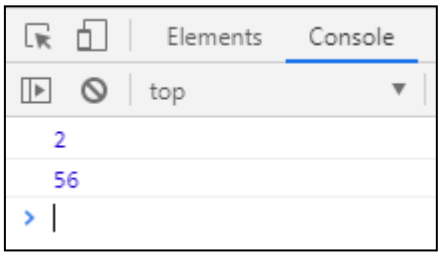


Рис. 4.79. Свойство length

Смешанные элементы

Создаем массив из четырех элементов разного типа: числовой, логический, строковый и объект. Запишем в отдельные переменные элементы массива. Обратите внимание, как идет обращение к полю `id` для объекта:

```
let objArr = [2022, true, "Денис", { id: 12345, fullname:
"Якубович Д.А."}];

let year = objArr[0];
let flag = objArr[1];
let name = objArr[2];
let fullName = objArr[3].id;
```

Выводим длину массива, сам массив и одно из свойств последнего объекта:

```
console.log(objArr.length);
console.log(objArr);
console.log(fullName);
```

Далее очищаем значение четвертого элемента оператором **delete**. При этом команда `delete` не меняет количества элементов:

```
delete objArr[3];

console.log(objArr.length);
```

```
console.log(objArr);
console.log(objArr[3]);
```

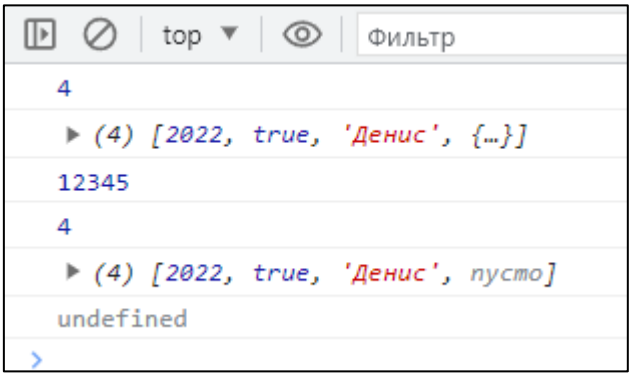


Рис. 4.80. Массивы со смешанными элементами и удаление элементов

Объекты как ассоциативные массивы

Ранее было отмечено, что массивы в JavaScript являются объектами. Более того, объекты в языке являются *ассоциативными массивами*, в которых обращение к значению свойства можно осуществлять по ключу:

```
const user = {
    fullName: "Якубович Денис Адреевич",
    email: "yakubovich.studylib@mail.ru"
};

// Обращение к свойству через оператор-точку
console.log(user.email);
// Работа как с ассоциативным массивом
console.log(user["email"]);
// Ключ предварительно можно записать в переменной
const myMail = "email";
console.log(user[myMail]);
```

Обработка массива в цикле

Чаще всего массивы обрабатываются в циклах. Например, стандартный обход элементов массива с помощью оператора цикла `for` можно задать следующим образом:

```
let arr = [2, 4, -10, 12, 56];

for (let i = 0; i < arr.length; i++){
    console.log(arr[i]);
}
```

Если в теле цикла требуется многократное обращение к *i*-му элементу массива и при этом нет необходимости явной работы с индексом, то полезным может быть создание *аббревиатур* (т.е. локальных переменных в виде копии значения):

```
let arr = [2, 4, -10, 12, 56];

for (let i = 0; i < arr.length; i++){
    const elem = arr[i];

    // Разные операции с элементом массива elem

    console.log(elem);
}
```

Задача «Проверка в объекте»

Пусть задан массив записей о студентах (объектов). Требуется вывести его в разметку и посчитать студентов, у которых не заданы имя или фамилия, или же возраст равен нулю.

Начнем с разметки и стилей:

Глава 4\Проверка в объекте\index.html

```
<h1>Работа с массивами</h1>
<h2>Вывод записей</h2>
<div id="data"></div>
<p id="result"></p>

<script src="code.js"></script>
```

Глава 4\Проверка в объекте\style.css

```
body {
    font-family: Verdana, Geneva, Tahoma, sans-serif;
    font-size: 16px;
}

h1, h2, h3 {
    font-family: 'Segoe UI', Tahoma, Geneva, Verdana,
sans-serif;
}

h1 { color: #107e81; }

h2 { color: #0caaaf; }
```

Перейдем к реализации скрипта. В начале зададим сам объект. Каждый элемент массива является объектом с тремя свойствами:

Глава 4\Проверка в объекте\code.js

```
// Массив объектов
const dataBase = [
    { name: "Денис", surname: "Якубович", age: 20 },
    { name: "Николай", surname: "", age: 0 },
    { name: "Вероника", surname: "Иванова", age: 21 },
    { name: "", surname: "Петрова", age: 19 },
    { name: "", surname: "", age: 21 },
    { name: "Александр", surname: "Михайлов", age: 0 }
];
```

Получаем ссылки на блоки, в которые далее выведем результаты (см. HTML-код):

Глава 4\Проверка в объекте\code.js

```
// См. код выше
const data = document.getElementById("data");
const result = document.getElementById("result");
```

Получаем число записей в переменную `rows`. Также зададим переменную `dataHTMLCode`, в которую будем накапливать текст разметки записей:

Глава 4\Проверка в объекте\code.js

```
// См. код выше
let rows = dataBase.length;
let dataHTMLCode = "";
```

Последовательно в цикле формируем записи. В конце записываем сформированную строку в блок `data`:

Глава 4\Проверка в объекте\code.js

```
// См. код выше
for (let i = 0; i < rows; i++) {
    dataHTMLCode += `<p>Имя: ${dataBase[i].name}</p>`;
    dataHTMLCode += `<p>Фамилия:
        ${dataBase[i].surname}</p>`;
    dataHTMLCode += `<p>Возраст: ${dataBase[i].age}</p>`;
    dataHTMLCode += "<hr>";
}

data.innerHTML = dataHTMLCode;
```

Далее перебираем объекты по индексу и осуществляем для каждого проверку (в целом эту проверку можно было разместить и в первом цикле, но разделим задачи вывода и поиска на отдельные циклы):

Глава 4\Проверка в объекте\code.js

```
// См. код выше
let incompleteRecords = 0;    // Счетчик пустых записей

for (let i = 0; i < rows; i++) {
    if (dataBase[i].name == "" ||
        dataBase[i].surname == "" ||
        dataBase[i].age == 0
    )
        incompleteRecords++;
}
```

Однако сравнивать с пустой строкой необязательно. Если строка не пустая, то в условии она отождествляется со значением `true` (но поскольку нас интересуют пустые строки, то берем отрицание):

Глава 4\Проверка в объекте\code.js

```
// См. код выше
let incompleteRecords = 0;
for (let i = 0; i < rows; i++) {
    if (!dataBase[i].name ||
        !dataBase[i].surname ||
        dataBase[i].age == 0
    )
        incompleteRecords++;
}
```

Наконец, выводим результат в блок `result`:

Глава 4\Проверка в объекте\code.js

```
// См. код выше
result.innerText = "Число неполных записей: " +
    incompleteRecords;
```

Результата работы скрипта изображен на рис. 4.81.

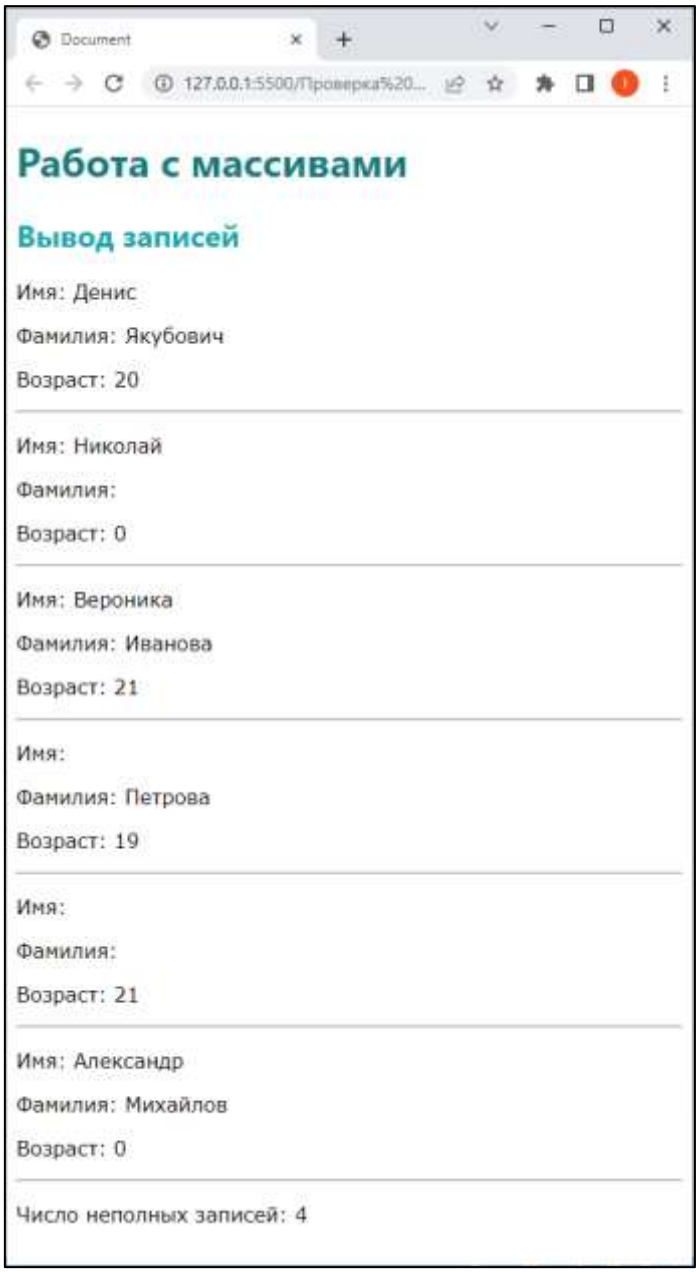


Рис. 4.81. Работа скрипта

Методы для работы с массивами

Добавление элементов в массив

Метод **`push()`** вставляет указанный(ые) элемент в конец массива (действует как стек).

Метод **`unshift()`** вставляет указанный(ые) элемент в начало массива (действует как очередь).

Дополнительно методы возвращают новое число элементов (если их вызывать, как функцию).

Извлечение элементов из массива

Метод **pop()** извлекает последний элемент массива (действует как стек).

Метод **shift()** извлекает первый элемент массива (действует как очередь).

Рассмотрим примеры:

```
const days = ["ср", "чт", "пт"];

days.push("сб");
const Saturday = days.pop();
const Friday = days.pop();
days.unshift("вт");
days.unshift("пн");
days.push(Friday, Saturday, "вс");
```

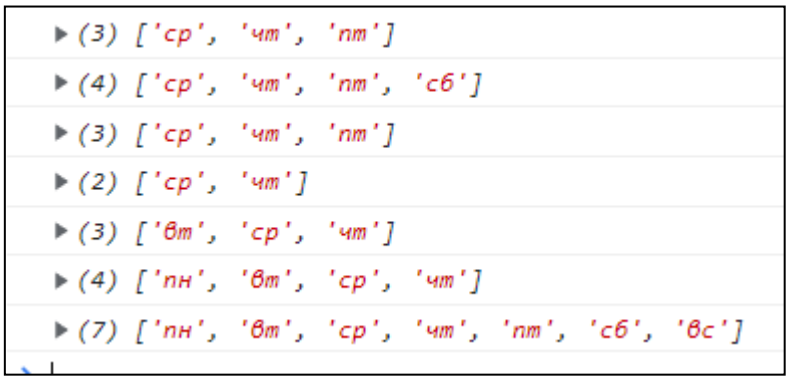


Рис. 4.82. Добавление и извлечение элементов массива

Добавление элементов в конец массива

Метод **concut()** добавляет в конец массива указанные элементы и возвращает копию массива (т.е. сам массив не изменяется).

Примеры:

```
const digits = [0, 1, 2];
const concut_1 = digits.concat(3, 4);
const concut_2 = digits.concat([3, 4, 5, 6, 7]);
const concut_3 = digits.concat([3, 4, 5, 6, 7, [8, 9]]);
```

Параметры можно передавать как отдельными элементами, так и в массиве. В последней строке массив `[8, 9]` уже вставляется как один элемент:

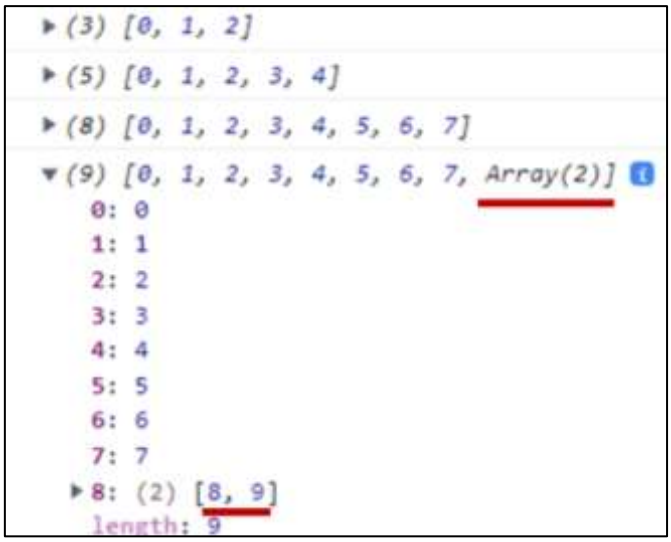


Рис. 4.83. Элемент массива сам является массивом

Получение подмассива

Метод **`slice()`** возвращает подмассив, не меняя искомого.

- Первый параметр – индекс начала подмасива.
- Второй параметр – индекс конца (не включая его!).
- Если указать только один аргумент, то копируются все элементы от текущего и до конца.
- Можно работать с отрицательными индексами, т.е. отсчитывать с конца.

Примеры:

```
const simbols = ["А", "Б", "В", "Г", "Д", "Е", "Ж", "З"];

const sub_arr_1 = simbols.slice(4);
const sub_arr_2 = simbols.slice(4, 6);
const sub_arr_3 = simbols.slice(-2);
const sub_arr_4 = simbols.slice(-6,-1);
```

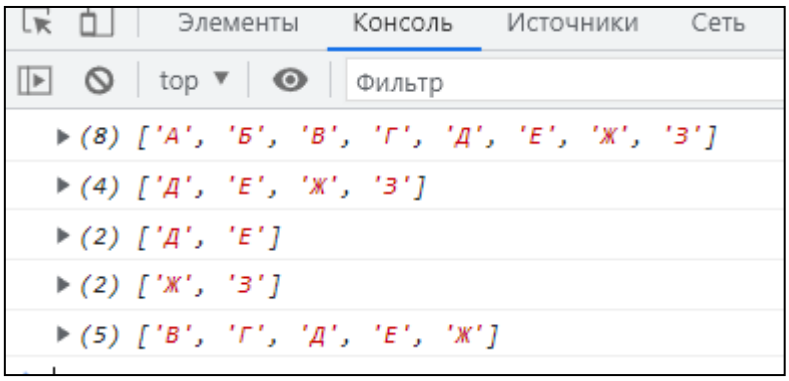


Рис. 4.84. Извлечение подмассива

Задача «Телефонная книга»

Пусть задан массив записей `records` о пользователях платформы дистанционного обучения. Сформировать массив с телефонными номерами студентов и вывести их построчно.

Зададим разметку и стили:

Глава 4\Телефонная книга\index.html

```
<h1>Справочник</h1>
<h2>Телефонная книга</h2>
<div id="tel"></div>

<script src="code.js"></script>
```

Глава 4\Телефонная книга\style.css

```
body {
    font-family: Verdana, Geneva, Tahoma, sans-serif;
    font-size: 16px;
}

h1, h2, h3 {
    font-family: 'Segoe UI', Tahoma, Geneva, Verdana,
sans-serif;
}

h1 { color: #107e81; }

h2 { color: #0caaaf; }

.total {
    background: #107e81;
    color: aliceblue;
    font-weight: bold;
    padding: 8px 20px;
}

.record {
    margin: 10px;
    padding: 8px;
    border: 1px solid #107e81;
    border-radius: 4px;
}
```

В объекте AccessRights задаем типы пользователей: гость, студент, преподаватель, администратор. В объекте records каждый объект в массиве описывают три свойства: ФИО, права доступа, телефон:

Глава 4\Телефонная книга\code.js

```
// Объект-перечисление: виды прав доступа
const AccessRights = {
    Guest: 0,
    Student: 1,
    Teacher: 2,
    Admin: 3
};

// Объект - логи о пользователях
const records = [
    {
        fullname: "Андреев Владислав Михайлович",
        rights: AccessRights.Student,
        telelphone: "89201234567"
    },
    {
        fullname: "Ивашова Ирина Владимировна",
        rights: AccessRights.Student,
        telelphone: "89101234567"
    },
    // Далее остальные записи...
];
```

Далее создаем ссылку на блок data, в который осуществим вывод результата. Запишем отдельно число элементов массива, создадим пустой массив для формирования новых записей с номерами телефонов искомых студентов. В цикле проверяем права пользователей и отбираем среди них студентов, добавляем их телефоны в конец массива:

Глава 4\Телефонная книга\code.js

```
// См. код выше
const data = document.getElementById("tel");

let all_users = records.length;
let tel_arr = [];
```

```
for (let i = 0; i < all_users; i++) {
    const user = records[i];

    if (user.rights == AccessRights.Student)
        tel_arr.push(user.telelphone);
}
```

Далее остается внедрить записи из массива `tel_arr` на страницу. Разумеется, можно с помощью цикла сформировать запись в одну строковую переменную, как это было сделано ранее в задаче *Проверка в объекте*. Однако продемонстрируем еще один подход к изменению кода разметки внутри тега – с использованием методов DOM для создания абстрактных элементов и динамического управления их атрибутами:

Глава 4\Телефонная книга\code.js

```
// См. код выше
for (let i = 0; i < tel_arr.length; i++) {
    const userTel = document.createElement("p");
    userTel.textContent = tel_arr[i];
    data.append(userTel);
}
```

В этом цикле создается DOM-узел: контейнер абзаца `<p></p>`. Далее в него добавляется текстовое содержимое (телефон). Наконец, узел добавляем внутрь контейнера `data`.

О преимуществах работы с DOM-узлами

Технология DOM предоставляет множество методов, которые позволяют динамически управлять разметкой документа и делать это более универсально, чем работа со свойством `innerHTML`.

Например, пусть в проекте используется блок `answer`:

```
<div id="answer" class="content"></div>
```

внутри которого в процессе должна выводиться информация о доходе:

```
const answer = document.getElementById("answer");

// Далее код, в результате которого вычисляем доход total
// ...
```

Пусть ответ должен быть помещен в блок `answer`. Сделать это можно, например, так:

```
answer.innerHTML = `<p>Итоговый доход: ${total}</p>`;
```

Однако допустим, что при определенных условиях требуется оформить абзац как-то иначе. Содержимое блока необходимо будет заменить, например так:

```
answer.innerHTML = `<p class='message report'>Итоговый
    доход: ${total}</p>`;
```

Такой подход уступает в гибкости. С одной стороны, нам потребовалось полностью заменить содержимое разметки блока `answer`. С другой – можно допустить опечатку: не закрыть кавычку, тег в строке или т.п.

Если работать с разметкой не как строкой, а как абстракцией, то создание элемента выглядит следующим образом:

```
const income = document.createElement("p");
income.textContent = `Итоговый доход: ${total}`;
answer.append(income);
```

Если потребуется добавить к элементу классы оформления, то сделать это можно следующим образом:

```
answer.firstChild.classList.add("message", "report");
```

(Обращаемся к первому дочернему элементу `answer`, списку его CSS-классов и добавляем к нему два указанных).

А если потребуется удалить какой-либо класс, то

```
answer.firstChild.classList.remove("message");
```

В отличие от `innerHTML`, методы DOM гораздо более удобны в работе с тегами внутри блоков, поскольку операции можно разбивать на отдельные части и оперировать ими в любой момент времени:

```
const income = document.createElement("p");
income.textContent = `Итоговый доход: ${total}`;
income.classList += "report";

const tax = document.createElement("p");
tax.textContent = `Налог: ${taxValue}`;

answer.append(income, tax);
```

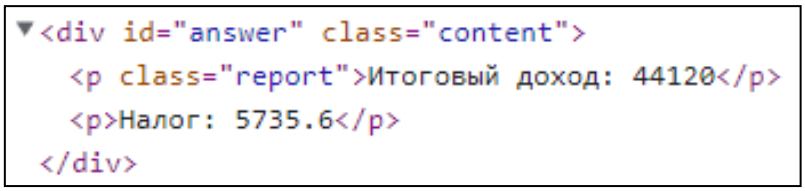


Рис. 4.85. Сформированный DOM-узел в блоке `answer`

Аналогичный код, но уже с `innerHTML`:

```
answer.innerHTML = `<p class='report'>Итоговый доход:
    ${total}</p>`;
answer.innerHTML += `Налог: ${taxValue}`;
```

Несмотря на то, что он получился короче, дальнейшее манипулирование содержимым потребует полной перезаписи `innerHTML`.

Rest и Spread операторы

Spread-оператор

Оператор `...` используется для разделения коллекций на отдельные элементы. В частности – *spread-оператор* разбивает массив на отдельные элементы.

Важно заметить: spread-оператор поддерживается начиная со спецификации ES6.

Примеры (см. описание ниже):

```
const nums = [45, 23, 8];

let max1 = Math.max(nums);
let max2 = Math.max(nums[0], nums[1], nums[2]);
let max3 = Math.max(...nums);

let nums_copy1 = nums;
let nums_copy2 = [nums];
let nums_copy3 = [...nums];

let arr1 = [4, 5];
let arr2 = [6, 7, 8, 9];
let concut = [1, 2, 3, ...arr1, ...arr2];
```

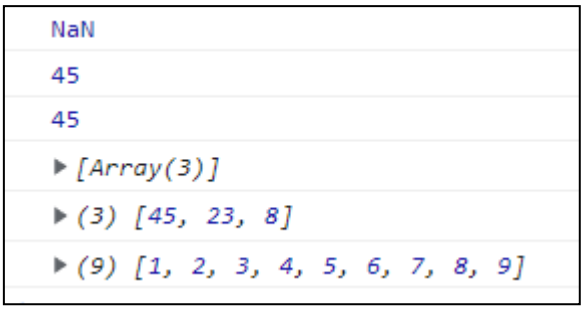


Рис. 4.86. Использование spread-оператора

В переменную `max1` будет записано `NaN`, т.к. функции `max()` числа нужно перечислять отдельно. В `max2` запишется 45, но требуется перечислять элементы по индексам. Переменная `max3` получит то же значение 45, но благодаря spread-оператору разбиение массива на отдельные элементы очень лаконичное.

Далее используем spread-оператор для организации полной копии массива `nums`.

- В `nums_copy1` будет записана ссылка на массив, т.е. фактически теперь `nums` и `nums_copy1` ссылаются на один и тоже массив. Такое копирование не создает независимый массив, а лишь копирует ссылки на один и тот же массив.
- В `nums_copy2` запишется массив из одного элемента, который является массивом `nums`. Этот вариант копирования также не подходит.
- В `nums_copy3` запишется полная и независимая копия массива `nums`, что и требовалось.

Наконец, spread-оператор может использоваться неоднократно. Такое очень часто требуется, когда необходимо создать новый массив из элементов других массивов.

Rest-оператор

При использовании spread-оператора **`...`** в описании параметров функции он становится *rest-оператором*, т.е. наоборот – собирает элементы в массив.

Важно заметить: spread-оператор поддерживается, начиная со спецификации ES6.

Например:

```
function rest(...arguments) {
    console.log(arguments);
}

rest(1);
rest(3, 7, 23);
rest(34, 67, 4, 1, 0, 46);
```

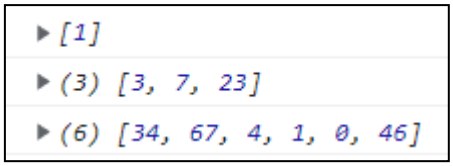


Рис. 4.87. Использование rest-оператора

В описании функции аргументы сворачиваются rest-оператором в массив `arguments`. Аргументы передаем как отдельные элементы.

Это полезно знать!

Таким образом, оператор `...` в зависимости от контекста может выполнять роль spread- либо rest-оператора.

Поиск по массиву

Поиск индекса искомого элемента

Метод **`indexOf()`** ищет индекс первого элемента, равного указанному.

Метод **`lastIndexOf()`** ищет индекс последнего элемента, равного указанному.

Метод **`findIndex()`** ищет индекс первого элемента, который удовлетворяет заданному условию.

Если заданный элемент не найден, возвращается число −1.

Примеры:

```
const array = [45, -5, 23, 56, 4, 23, 56, -5];

let a = array.indexOf(23);          // 2
let b = array.indexOf(0);           // -1
let c = array.lastIndexOf(23);      // 5
let d = array.lastIndexOf(0);       // -1
```

```
let firstEvenPosIndex = array.findIndex(
    i => (i > 0) && (i % 2 == 0)
);
```

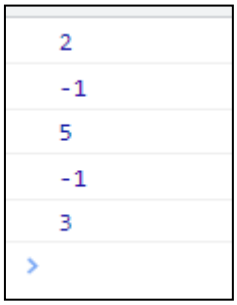


Рис. 4.88. Поиск индексов элементов, удовлетворяющих условию

Прокомментируем последнюю строку. Методу `finIndex()` в качестве параметра передается *предикат* – это функция, возвращающая логическое значение `true` либо `false`:

```
i => (i > 0) && (i % 2 == 0)
```

В данном случае он проверяет, является ли число положительным и четным. Параметр `i` пробегает каждый элемент из массива и может быть обозначен любым другим именем.

Второй момент – здесь функция описана в форме *стрелочной функции*, т.е. в более короткой и современной форме, появившейся в стандарте ES6. До ES6 приходилось использовать синтаксис *анонимной функции*:

```
let firstEvenPosIndex2 = array.findIndex(function (i) {
    return (i > 0) && (i % 2 == 0);
});
```

Синтаксис описания функций, в т.ч. стрелочных, рассматривается в следующих разделах.

Поиск искомого элемента

Метод **`find()`** ищет первый элемент, который удовлетворяет заданному условию (предикату). Если элемент не найден, возвращается значение **`null`**.

В следующем примере из массива объектов найдем первого совершеннолетнего студента:

```
const table = [
    { fullName: "Якубович Д.А.", age: 17},
    { fullName: "Петров Ю.С.", age: 18},
    { fullName: "Калинина О.И.", age: 19},
    { fullName: "Ильина Л.З.", age: 16},
];

let firstAdult = table.find(person => person.age >= 18);

console.log(firstAdult);
```

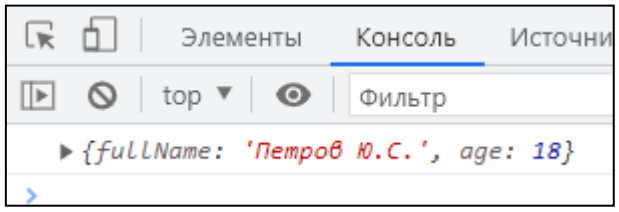


Рис. 4.89. Поиск первого элемента, удовлетворяющего условию

В этом примере также используем синтаксис стрелочных функций.

Преобразование и перебор в массиве

Преобразование с созданием нового массива

Метод **map()** создаёт новый массив, который вызывает указанную функцию, преобразующую каждый элемент массива. Исходный массив не меняется.

Примеры:

```
const array = [5, -4, 12, 7, -1];

const squares = array.map(n => n**2);
const roots = array.map(Math.sqrt);
const resetNegatives = array.map(n => (n > 0) ? n : 0);

console.log(squares);
console.log(roots);
console.log(resetNegatives);
```

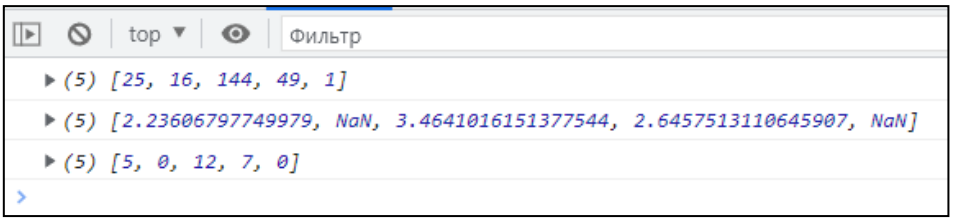


Рис. 4.90. Преобразование массива с помощью map()

В первом случае каждый элемент возводится в квадрат.

Во втором – к каждому элементу применяется функция вычисления квадратного корня `Math.sqrt()`.

В третьем – каждый отрицательный элемент заменяется нулем. Здесь используется тернарный оператор, хотя можно было записать инструкцию `if-else` в полной форме:

```
const resetNegatives = array.map(n => {
    if (n > 0)
        return n;
    else
        return 0;
});
```

Приведем еще один пример. Из массива объектов `table` в новый массив `users` отберем ФИО, а в `journal` – информацию по ФИО и возрасте в форме строк:

```
const table = [
    { fullName: "Якубович Д.А.", age: 17},
    { fullName: "Петров Ю.С.", age: 18},
    { fullName: "Калинина О.И.", age: 19},
    { fullName: "Ильина Л.З.", age: 16},
];

const users = table.map(person => person.fullName);

const journal = table.map(person =>
    `ФИО: ${person.fullName} Возраст: ${person.age}`
);

console.log(users);
console.log(journal);
```

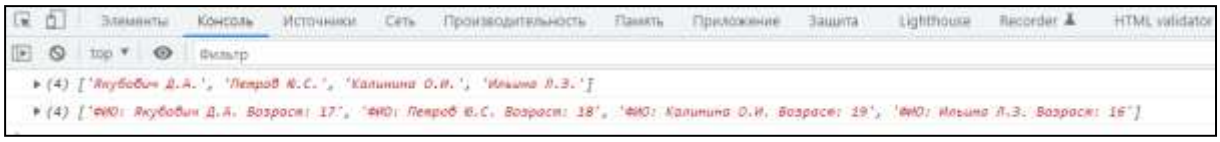


Рис. 4.91. Отбор данных из массива объектов

Перебор в массиве

Метод **`forEach()`** последовательно перебирает элементы массива с помощью функции, описанной в параметре.

Например, найдем сумму квадратов элементов заданного массива:

```
const array = [45, -5, 23, 56, 4, 23, 56, -5];

let sum = 0;
array.forEach(x => sum += x * x);

console.log(sum);
```

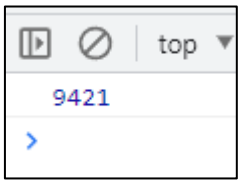


Рис. 4.92. Перебор по массиву для подсчета суммы

Для `forEach()` можно указать второй параметр – индекс текущего элемента. В приведенном примере дополнительно будем выводить квадрат каждого элемента с индексом:

```
const array = [45, -5, 23, 56, 4, 23, 56, -5];

let sum = 0;

array.forEach((x,i) => {
    const quad = x * x;
    sum += quad;
    console.log(`${i}. ${quad}`);
});
```

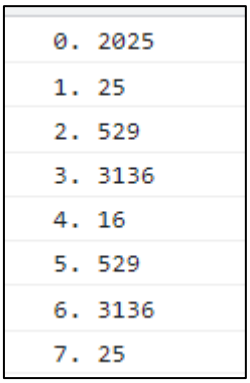


Рис. 4.93. Перебор по массиву

Фильтрация данных

Метод **`filter()`** создаёт новый массив, который вызывает указанную функцию, отбирающую только подходящие под условие элементы массива. Исходный массив не меняется.

Приведем пример. Из массива `array` отберем в `even_positive` положительные четные числа. А из массива `table` в `users` отберем совершеннолетних персон, у которых фамилия начинается с «Я»:

```
const array = [5, -4, 12, 7, -1];

const table = [
    { fullName: "Якубович Д.А.", age: 20},
    { fullName: "Петров Ю.С.", age: 18},
    { fullName: "Якушев О.И.", age: 19},
    { fullName: "Яковлева Л.З.", age: 17},
];

const even_positive = array.filter(
    n => (n > 0) && (n % 2 == 0)
);

const users = table.filter(
    person =>
    (person.age >= 18) && (person.fullName[0] == "Я")
);

console.log(even_positive);
console.log(users);
```

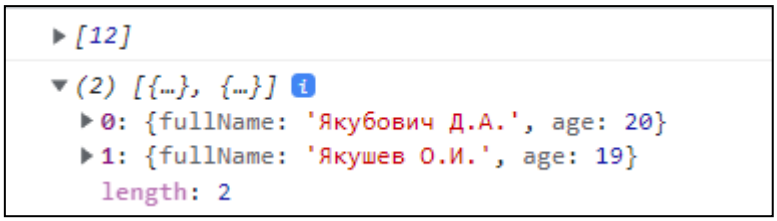


Рис. 4.94. Отбор элементов в новый массив

Объединение и разбиение

Метод join

Метод **join()** преобразует все элементы массива в строки, объединяет их и возвращает получившуюся строку. В качестве параметра можно указать значение, которое будет вставляться между соседними элементами (по умолчанию будут отделяться запятой).

Примеры:

```
const array = [-3, 0, 5, -10, -17];
const default_arr = array.join();
const my_arr = array.join(" || ");

console.log(default_arr);
console.log(my_arr);
```

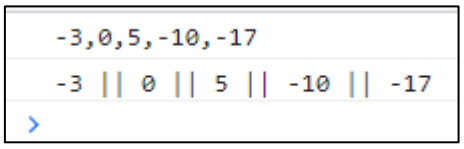


Рис. 4.95. Слияние элементов в строку с указанным разделителем между элементами

Метод split

Метод **split()** разбивает текстовую строку на массив строк с указанным в качестве параметра разделителем.

Примеры:

```
const text = "Мама мыла раму. Папа очень рад";
const words = text.split(" ");
const sentences = text.split(".");
const simbols = text.split("");

console.log(words);
console.log(sentences);
console.log(simbols);
```

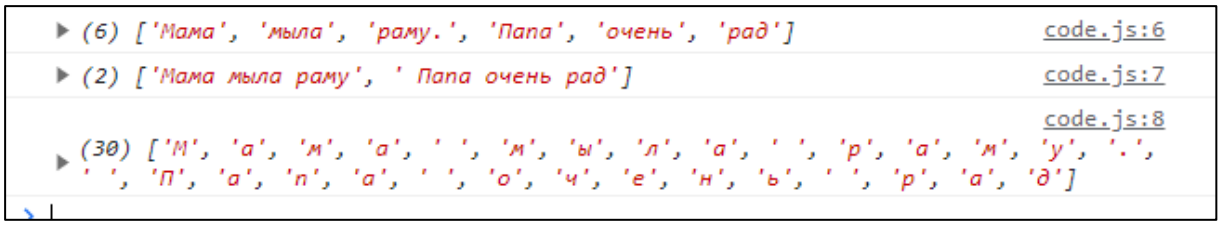


Рис. 4.96. Разбиение строк на массивы подстрок с указанным разделителем

Это полезно знать!

JavaScript позволяет объединять вызов методов в цепочки. Например, код

```
arr.reverse();
let mass = arr.join(" || ");
```

можно скомпоновать в одной команде:

```
let mass = arr.reverse().join(" || ");
```

(Методы будут вызываться в порядке следования)

4.6.2. Цикл for-of

Синтаксис

Определение

```
for (const элемент of коллекция) {
    // работа с элементом
}
```

*Инструкция **for-of** используется для перебора элементов итерируемых объектов. Цикл удобен тем, что не требует обращаться по индексу к элементам массивов.*

Удобство использования цикла проще всего продемонстрировать на примере. Пусть требуется реализовать простой обход и вывод всех элементов массива. С помощью цикла `for` реализация имеет следующий вид:

```
let array = [5, -10, -17, 22, 0, 5, -13];

for (let i = 0; i < array.length; i++) {
    console.log(array[i]);
}
```

В теле цикла отсутствует необходимость явной работы с индексом элемента. Цикл `for-of` делает реализацию алгоритма более компактной:

```
let array = [5, -10, -17, 22, 12, 5, -13];

for (const elem of array) {
    console.log(elem);
}
```

В этом цикле итерируемый объект – это массив `array`. В константу `elem` записывается очередной элемент массива. Здесь не требуется обращаться по индексу или получать число элементов.

При этом `elem` можно задать и переменной (с помощью `let`). В этом случае ее можно менять локально, но это никак не скажется на искомом массиве: `elem` является лишь локальной копией элемента.

Обработка числовых массивов

Вычислим сумму модулей чисел в массиве:

```
const array = [3.5, 6.2, -4.0, 2.0, 5.5, -0.9];
let sum = 0;

for (const n of array) {
    sum += Math.abs(n);
}

console.log(sum);
```

Следует учитывать особенности арифметики с плавающей точкой: дробная часть будет несколько отличаться от точной:

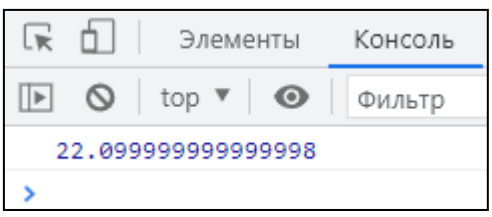


Рис. 4.97. Сумма модулей чисел в массиве

Обработка строковых массивов

Подсчитываем знаки препинания в текстовой строке.

Для начала сгруппируем символы пунктуации в массив.

С помощью цикла перебираем символы в строке. Метод **`includes()`** проверяет наличие хотя-бы одного символа препинания в массиве `punkt_marks`. Если элемент найден, возвращается `true`.

Метод поддерживается с ES6 и выше:

```
const text = "JS поддерживает несколько циклов: for,
    while, do/while, for/of, for/in.";
const punkt_marks = [",", ";", ":", "."];

let counter = 0;

for (const sim of text) {
    if (punkt_marks.includes(sim))
        counter++;
}

console.log(counter);
```

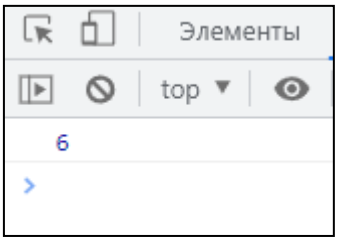


Рис. 4.98. Подсчет количества символов препинания в строке

4.6.3. Цикл for-in

Синтаксис

Определение

```
for (const свойство in объект) {
    // работа с свойством или его значением
}
```

*Инструкция **`for-in`** организует цикл перебора всех перечисляемых свойств объекта.*

Какие свойства относятся к перечислимым?

- В объектах, которые *создаются пользователем*, все свойства автоматически становятся перечислимыми. При необходимости допускается изменить специальный атрибут, который «спрячет» свойство, однако такая тонкая настройка свойств требуется не часто.
- К неперечислимым свойствам относят *свойства глобального объекта* **`Object`**, который является базовым для всех остальных, в т.ч. пользовательских.

Пример использования

Опишем объект, характеризующий прямоугольник: его задают ширина, высота, а также функция, которая возвращает периметр фигуры. С помощью цикла `for-in` осуществим обход его свойств и значений:

```
const Rectangle = {
    width: 2.5,
    height: 4.0,
    perimeter() {
        return 2*(this.width + this.height);
    }
};

for (const prop in Rectangle) {
    console.log(prop, ":", Rectangle[prop]);
}
```

```
width : 2.5
height : 4
perimeter : ƒ () {
        return 2*(this.width + this.height);
    }
```

Рис. 4.99. Вывод перечислимых свойств объекта

Так в переменную `prop` записывается очередной ключ (свойство) объекта. А чтобы получить значение свойства, используем синтаксис объектов как ассоциативных массивов:

```
Rectangle[prop]
```

(а вот запись `Rectangle.prop` не будет рабочей).

Вопросы для самопроверки

1. Какими особенностями обладают массивы в JavaScript?
2. Как обратиться к полю объекта, который является элементом массива?
3. Почему объекты также называют ассоциативными массивами и как это можно использовать в их обработке?
4. Что будет записано в промежуточные элементы массива, если задать значения только его первому и последнему элементу?
5. Какие циклы можно использовать при обработке массивов?
6. Перечислите команды, которые позволяют добавлять и извлекать элементы из массива.
7. Каким образом можно извлечь часть массива в новый массив?
8. Почему в общем случае рекомендуется работать с абстрактной моделью DOM-узлов, а не формировать структуру разметки с помощью свойства `innerHTML`?
9. Опишите назначение spread- и rest-операторов.
10. Перечислите методы, которые используются в поиске элементов или индексов элементов в массиве, которые удовлетворяют заданным условиям.
11. Что представляют собой анонимные функции и чем они отличаются от стрелочных?
12. Какие методы позволяют осуществлять преобразование и перебор элементов массива?
13. Каике методы помогают найти элементы, удовлетворяющие условию и на их основе создать новый массив?
14. Опишите методы, которые позволяют сворачивать массивы в строку и обратно разбивать на отдельные элементы.
15. Чем удобен цикл `for-of` и в каких случаях его рационально использовать?
16. Для каких задач предназначен цикл `for-in`?

Практикум

Задание 1

1. Создайте каталог *Проверка в объекте*.
2. Вернитесь к рассмотренному примеру в пункте 4.6.1.
3. Воспроизведите файлы для веб-страницы. Проверьте корректность работы скрипта.
4. Поправьте реализацию циклов, внеся в их тело локальную вспомогательную переменную (т.е. избавьтесь от работы с индексом).
5. Образец оформления изображен на рис. 4.81.

Задание 2

1. Создайте каталог *Отчет об операциях*.
2. Подготовьте типовую начальную разметку, подключите стилевой файл и скрипт. Скопируйте приведенные ниже заготовки кода в соответствующие файлы.
3. Изучите код скрипта. Пусть в массиве объектов `logs` записана информация о проведенных за определенный срок операциях с товарами.
4. Определите и выведите на странице информацию об утилизации продуктов.
5. Дополнительно посчитайте суммарную стоимость утилизации просроченного товара.
6. Образец оформления изображен на рис. 4.101.

Глава 4\Отчет об операциях\index.html

```
<h1>Отчет об операциях</h1>
<h2>Утилизация продукции:</h2>
<div id="liquidation"></div>

<script src="code.js"></script>
```

Глава 4\Отчет об операциях\style.css

```
body {
    font-family: Verdana, Geneva, Tahoma, sans-serif;
    font-size: 16px;
}
```

```
h1, h2, h3 {
    font-family: 'Segoe UI', Tahoma, Geneva, Verdana,
sans-serif;
}

h1 { color: #107e81; }

h2 { color: #0caaaf; }

.total {
    background: #107e81;
    color: aliceblue;
    font-weight: bold;
    padding: 8px 20px;
}
```

Глава 4\Отчет об операциях\code.js

```
// Задаем объект-перечисление с видами операций:
// - покупка
// - продажа
// - ликвидация товара с истекшим сроком годности
const Operations = {
    Purchase: 1,
    Sale: 2,
    Liquidation: 3
}

// Объект - логи об операциях с товаром
const logs = [
    {
        operation: Operations.Sale,
        date: "18.02.22",
        cost: 43000,
        description: "Продажа бытовой химии"
    },
    {
        operation: Operations.Sale,
        date: "18.02.22",
        cost: 3920,
        description: "Продажа фруктов"
    },
    {
        operation: Operations.Sale,
        date: "19.02.22",
        cost: 8020,
```

```
        description: "Продажа фруктов"
    },
    {
        operation: Operations.Sale,
        date: "19.02.22",
        cost: 38200,
        description: "Продажа товаров для дома"
    },
    {
        operation: Operations.Purchase,
        date: "19.02.22",
        cost: 6700,
        description: "Покупка расходников"
    },
    {
        operation: Operations.Sale,
        date: "20.02.22",
        cost: 59300,
        description: "Продажа товаров для дома"
    },
    {
        operation: Operations.Sale,
        date: "20.02.22",
        cost: 9200,
        description: "Продажа косметики"
    },
    {
        operation: Operations.Liquidation,
        date: "20.02.22",
        cost: 700,
        description: "Утилизация молочной продукции"
    },
    {
        operation: Operations.Liquidation,
        date: "20.02.22",
        cost: 900,
        description: "Утилизация испорченных фруктов"
    },
    {
        operation: Operations.Purchase,
        date: "21.02.22",
        cost: 3200,
        description: "Доставка товара"
    },
    {
```

```
        operation: Operations.Sale,
        date: "21.02.22",
        cost: 5080,
        description: "Продажа молочной продукции"
    },
    {
        operation: Operations.Sale,
        date: "21.02.22",
        cost: 10200,
        description: "Продажа косметики"
    },
    {
        operation: Operations.Sale,
        date: "21.02.22",
        cost: 8600,
        description: "Продажа фруктов"
    },
    {
        operation: Operations.Liquidation,
        date: "21.02.22",
        cost: 550,
        description: "Утилизация молочной продукции"
    },
];

// Ссылка на блок, в котором будем выводить информацию
const data = document.getElementById("liquidation");

// Сохраним число всех записей (объектов)
let all_operations = logs.length;
// В нее накапливаем разметку
let dataHTMLCode = "";
// Потери от утилизированного товара
let liq_sum = 0;

for (let i = 0; i < all_operations; i++) {
    // Заводим локально переменную "текущая_операция",
    // чтобы далее не возиться с индексом i
    const current_operation = logs[i];

    // Здесь проверить, является ли операция утилизацией
    // Если да - записать дату, стоимость и описание
    // операции и добавить к сумме затрат
}
```

```
dataHTMLCode += `<p class='total'>
                    Потери от утилизированного товара:
                    ${liq_sum}р.
                </p>`;

// Копируем код в блок
data.innerHTML = dataHTMLCode;
```

Задание 3

1. Скопируйте проект *Телефонная книга* из п. 4.6.1 и назовите его *Телефонный справочник*.
2. Воспроизведите веб-страницу, проверьте работу скрипта.
3. Еще раз внимательно изучите код скрипта.
4. Сформируйте массив, в который запишите всех преподавателей с ФИО и номерами телефонов. Т.е. каждый элемент нового массива – склеенная из двух значений строка.
5. С помощью методов DOM поместите каждую запись в блок, оформленный классом `record`:

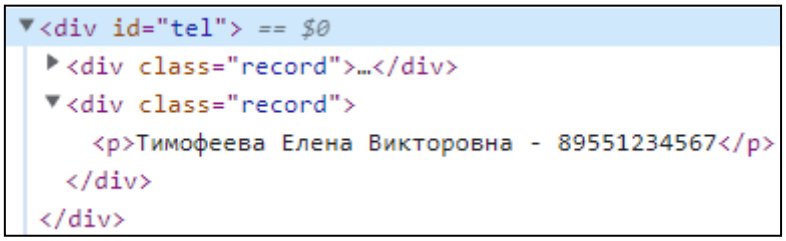


Рис. 4.100. Такие блоки должны формироваться в цикле

6. Образец оформления изображен на рис. 4.102.

Задание 4

1. Создайте проект *Поиск в массивах*.
2. Подготовьте типовую начальную разметку и подключите внешний файл скрипта (файл CSS-стилей не потребуется).
3. Для созданных массивов требуется последовательно осуществить описанные в комментариях преобразования и поиск. Результаты запишите в отдельные переменные.
4. Ответы вывести с использованием методов отладки `console.log()` или `console.table()` (какого-либо оформления не требуется).
5. Образец оформления изображен на рис. 4.103.

Глава 4\Поиск в массивах\code.js

```
const array = [-6, 56, 3, 8, 67, 15, 2, -3, -56, 1, 22,
3, 6, 12];
// 1. Вычислить минимальное значение
// 2. Создать копию в переменную arr_copy
// 3. Склеить array и arr_copy в новый массив double_arr
// 4. Найти индекс первого числа 3 и последнего числа 22
//    (используйте методы indexOf() и lastIndexOf())
// 5. Найти индекс первого нечетного числа в промежутке
//    от 10 до 20 (используйте метод findIndex())

const points = [
    { x: -4, y: 2.4 },
    { x: 0, y: -2 },
    { x: 1, y: 0},
    { x: -2, y: -1.9 },
    { x: 5.2, y: -1 },
    { x: -2.1, y: -0.5 }
];
// 6. Вывести первую точку с отрицательными координатами
//    (используйте find())
// 7. Вывести первую точку, для которой радиус вектор
//    меньше 5 (используйте find())
```

Задание 5

1. Создайте проект *Преобразования и фильтрация в массивах*.
2. Подготовьте типовую начальную разметку и подключите внешний файл скрипта (файл CSS-стилей не потребуется).
3. Скопируйте приведенный в конце задания код заготовки скрипта.
4. Для созданных массивов требуется сгенерировать новые, следуя описанию в комментариях.
5. Ответы вывести с использованием методов отладки `console.log()` или `console.table()` (какого-либо оформления не требуется).
6. Образец изображен на рис. 4.104.

Глава 4\Преобразования и фильтрация в массивах\code.js

```
const array = [-6, 56, 3, 8, 67, 15, 2, -3, -56, 1, 22,
    3, 6, 12];
// 1. Сгенерировать массив new_arr, преобразовав каждый
//    элемент x в (x + 3)^2 (используя map()).
// 2. Сгенерировать массив синусов двойного угла из array
//    (используя map()).
// 3. Вычислить натуральные логарифмы для всех x > 0,
//    иначе брать эти элементы нулями (используя map()).
// 4. Выделить два отдельных массива: с положительными и
//    отрицательными элементами (используя filter()).

const users = [
    { id: 1, login: "petya.pupkin", password: "12345" },
    { id: 2, login: "lera_holera", password: "OduvanchiK"
},
    { id: 3, login: "dark_hill", password: "BMA.user" },
    { id: 4, login: "zero_herro", password:
"ne_bug_a_ficha" },
    { id: 5, login: "solnishko", password: "nervniy_man"
}
];
// 5. Сгенерировать массив объектов из свойств id и
//    password (используя map())
// 6. Создать новый массив пользователей с id от 3 до 5
//    включительно (используя filter())

const test_arr = ["Массивы","являются","одними","из",
    "важнейших","элементов","JS"];
// 7. Преобразуйте массив в единую строку, слова отделите
//    пробелом (используя метод join())

const path = "d:/Documents/Документы/Работа/Руководство
ВКР/2020-2021";
// 8. Выделите массив с названием каталогов из пути
//    (используя метод split())
```

Задание 6

1. Создайте полную копию выполненного ранее задания *Телефонный справочник* (задание 3).
2. Замените циклы `for` на циклы `for-of`, сохранив при этом аналогичную рабочую способность кода.

Задание 7

1. Создайте каталог *Циклы for-in и for-of*.
2. Подготовьте разметку, стили и подключите скрипт. Код скопируйте в конце задания.
3. Изучите некоторые приемы обработки текста из текстового поля.
4. Замените циклы `for` на циклы `for-of`.
5. Допишите цикл `for-in` для перебора свойств объекта.
6. Образец изображен на рис. 4.105.

Глава 4\Циклы for-in и for-of\index.html

```
<h1>Циклы for</h1>

<h2>Цикл for/on</h2>
<p>Введите массив:</p>
<p><input type="text" id="inp_arr" class="text-input"
value="0"></p>
<button id="start" class="start-btn">Вычислить</button>
<div id="for_of"></div>

<h2>Цикл for/in</h2>
<div id="for_in"></div>

<script src="code.js"></script>
```

Глава 4\Циклы for-in и for-of\style.css

```
* { box-sizing: border-box; }

body { font: 1.0em Arial; }

h1, h2, h3 {
    font-family: Tahoma, Geneva, Verdana, sans-serif;
}

.text-input {
    display: block;
```

```
    width: 100%;
    font-size: 1.02em;
}

.start-btn {
    padding: 6px 20px;
    font-size: 1.02em;
    font-weight: bold;
}
```

Глава 4\Циклы for-in и for-of\code.js

```
// Ссылка на поле ввода, кнопку, блок для ввода и вывода
// информации
const inp_arr = document.getElementById("inp_arr");
const start = document.getElementById("start");
const for_of = document.getElementById("for_of");
const for_in = document.getElementById("for_in");

start.onclick = function () {
    // Берем текст из поля ввода, разбиваем его на
    // элементы (отделенные пробелом) и записываем в
    // массив
    const text_array = inp_arr.value.split(" ");
    // Каждый элемент массива строк переводим в число
    let array = text_array.map(x => parseFloat(x));

    // Сортируем массив по возрастанию
    array.sort();
    // Каждый элемент возводим в квадрат
    array = array.map(x => x * x);

    let new_array = [];

    // Заменить на for-of     <<-------------
    for (let i = 0; i < array.length; i++) {
        if ((10 <= array[i]) && (array[i] <= 100))
            new_array.push(array[i]);
    }

    let code = "";

    // Заменить на for-of     <<-------------
    for (let i = 0; i < new_array.length; i++) {
        code += `<p>${new_array[i]}<p>`;
    }
```

```
        for_of.innerHTML = code;

        const Vector = { x: 2, y: -3, z: 6};
        code = "";
        // С помощью for-in перебрать все свойства объекта
        // Vector и записать их code     <<-------------

        for_in.innerHTML = code;
    }
```

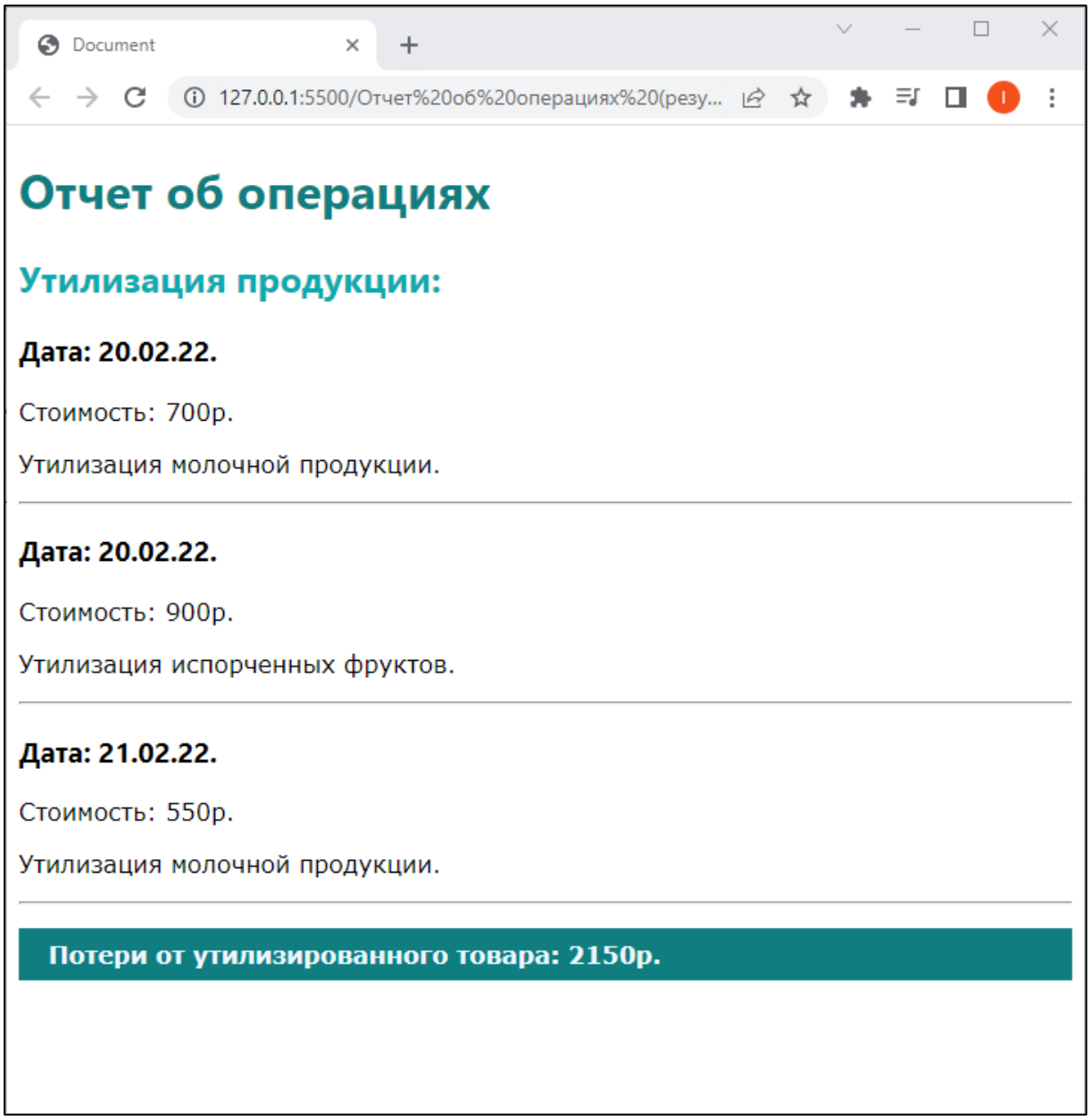


Рис. 4.101. Образец выполнения задания 2

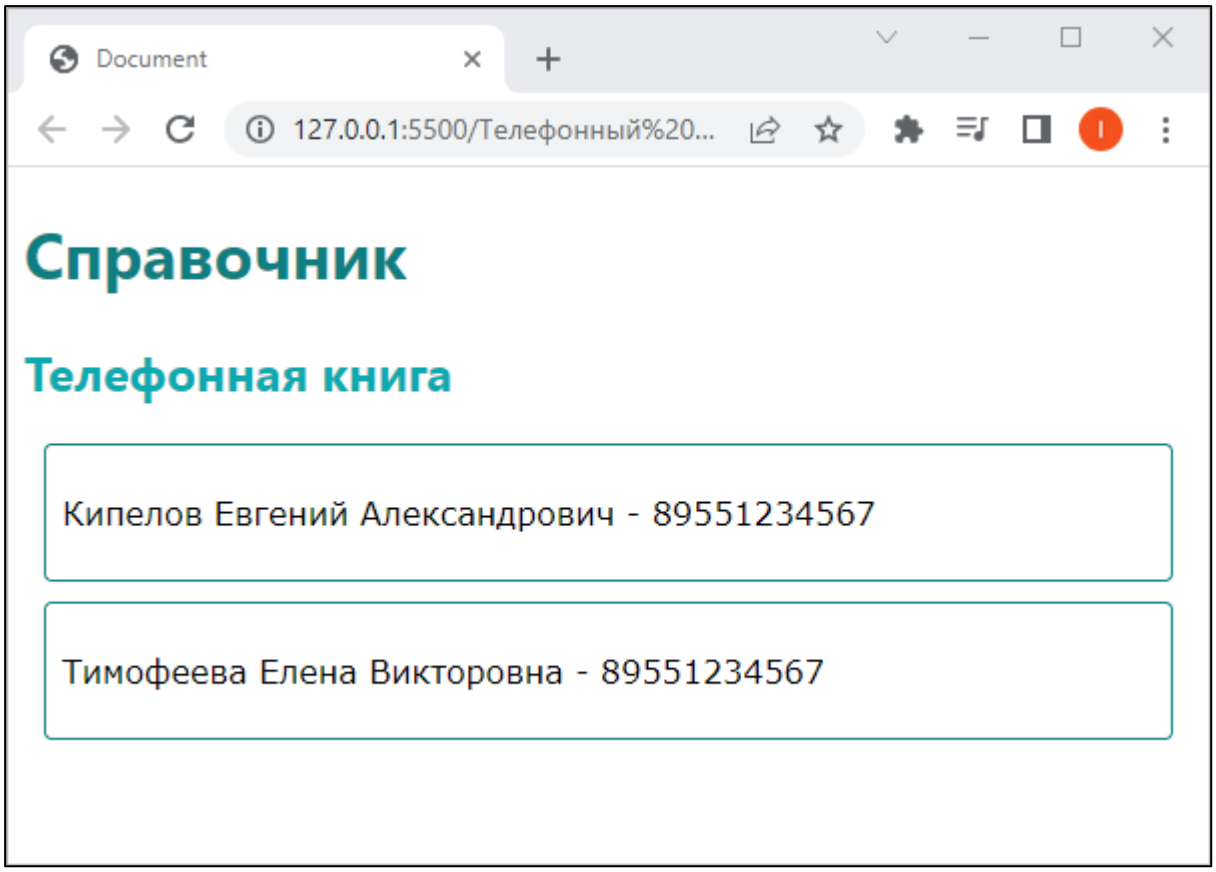


Рис. 4.102. Образец выполнения задания 3

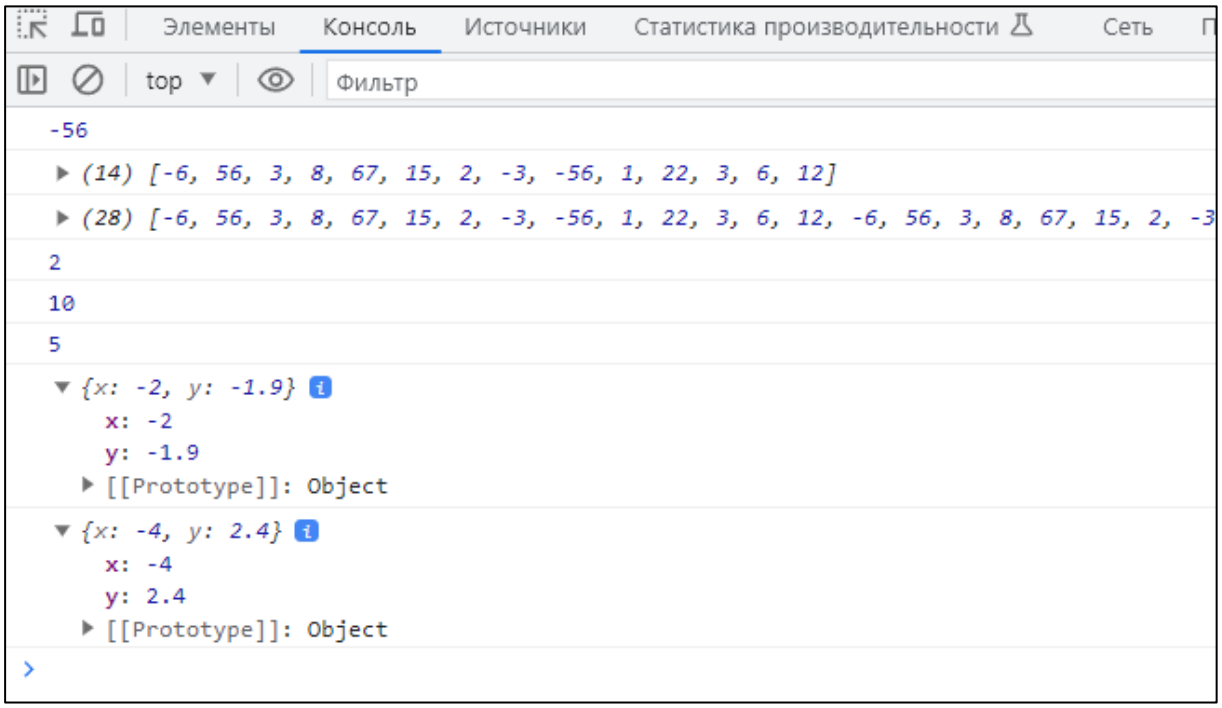


Рис. 4.103. Образец выполнения задания 4

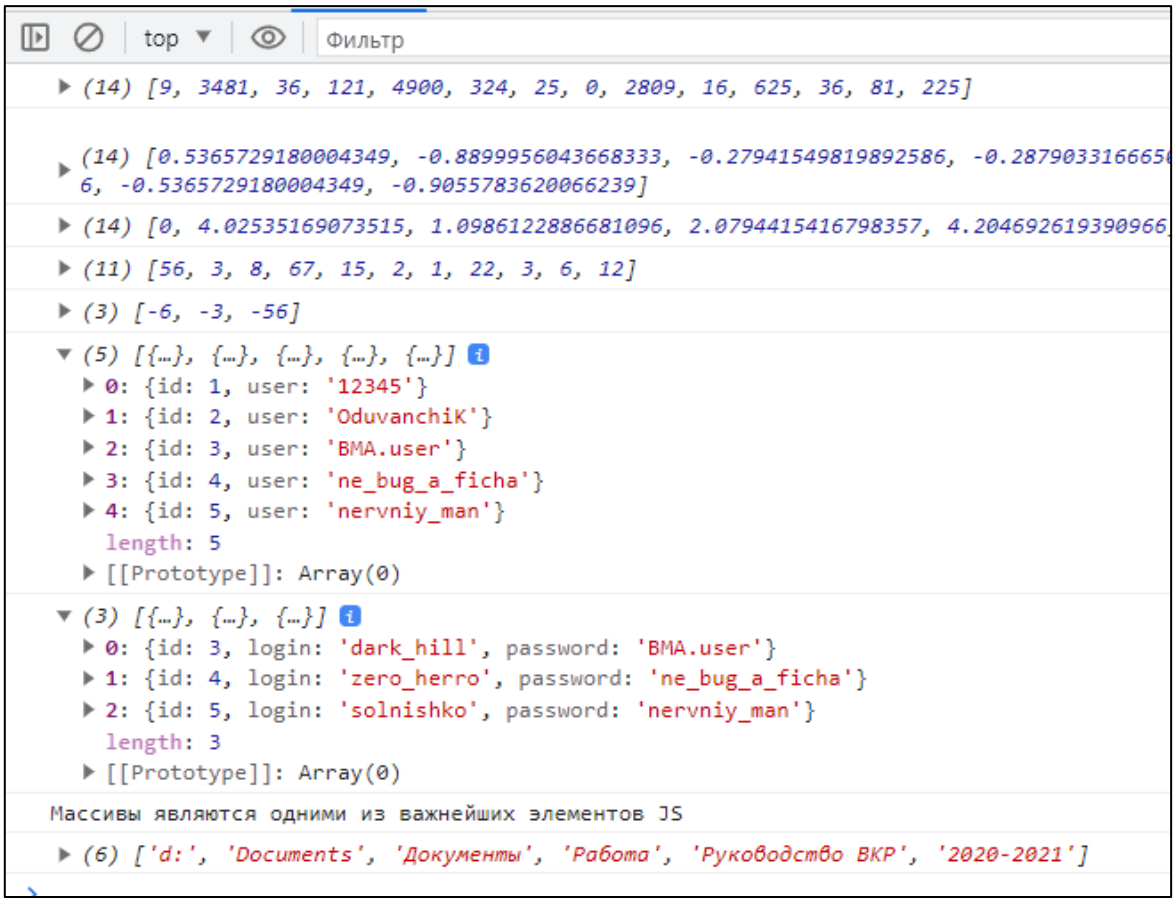


Рис. 4.104. Образец выполнения задания 5

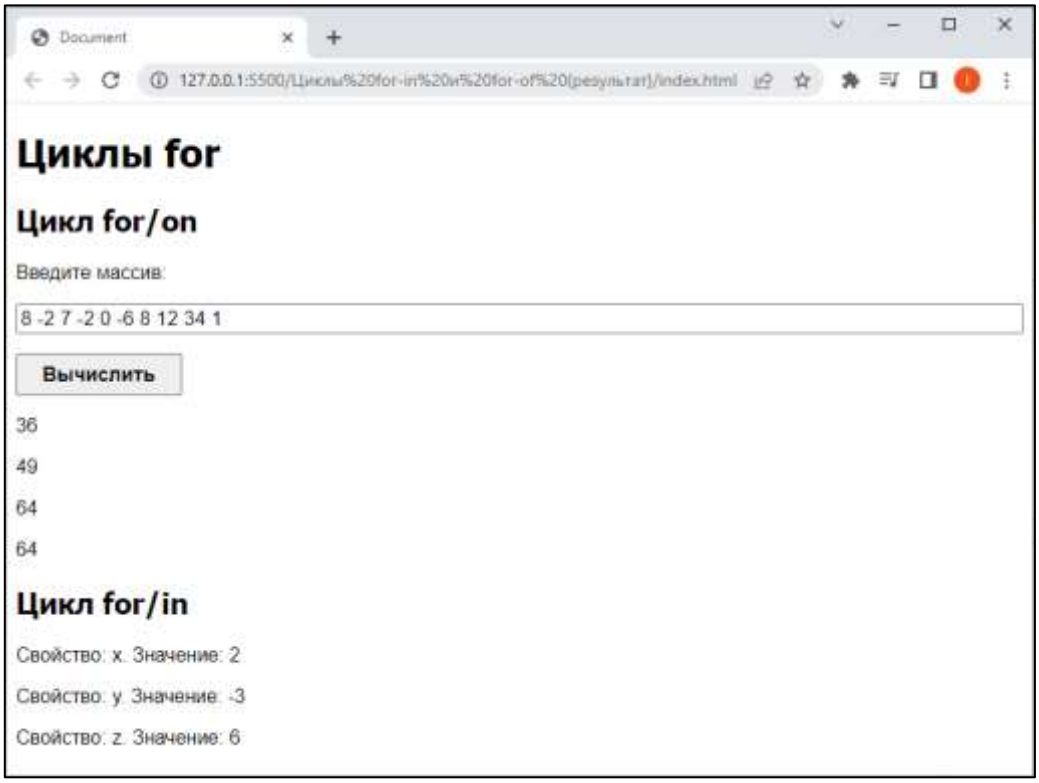


Рис. 4.105. Образец выполнения задания 7

4.7. Функции в JavaScript

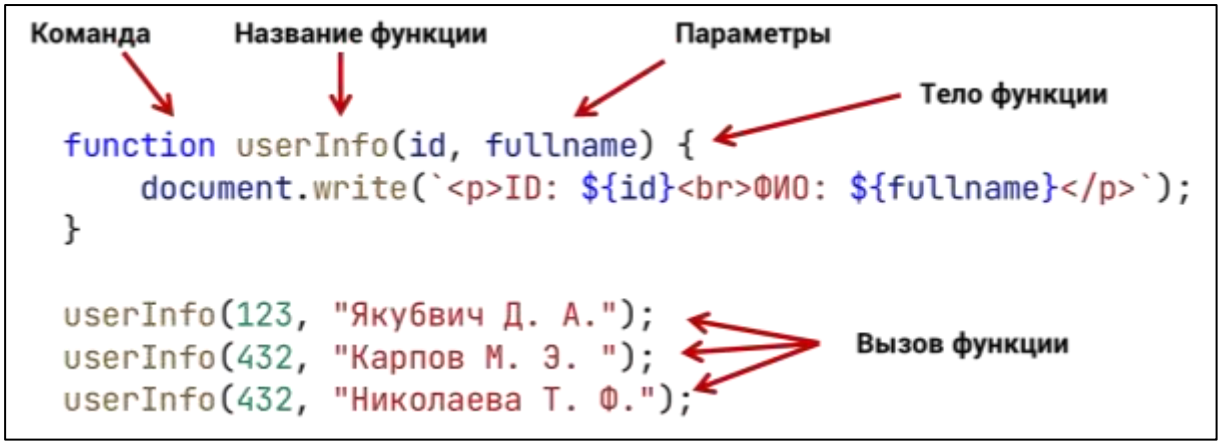
4.7.1. Понятие функции

Понятие и синтаксис

Определение

Функция *в JavaScript — объект, выполняющий роль подпрограммы. Основная задача функции – возвратить некоторое значение либо произвести ряд операций.*

Описание функции задается с помощью команды **`function`**. Параметры функции перечисляются через запятую в круглых скобках. Если параметров нет, то круглые скобки остаются пустыми. Тело функции задается в фигурных скобках.



ID: 123
ФИО: Якубвич Д. А.

ID: 432
ФИО: Карпов М. Э.

ID: 432
ФИО: Николаева Т. Ф.

Рис. 4.106. Пример описания функции и ее компоненты

Блок кода, заданный ключевым словом `function`, является *описанием функции*. Он выполняется только при *вызове функции*, т.е. указании ее имени и передаче параметров.

Роль функций в структуре скрипта

Функции решают целый ряд задач:

- убирают дублирование фрагментов повторяющегося кода;
- замыкают логически автономные блоки обработки данных и операции;
- упрощают чтение и сопровождение кода.

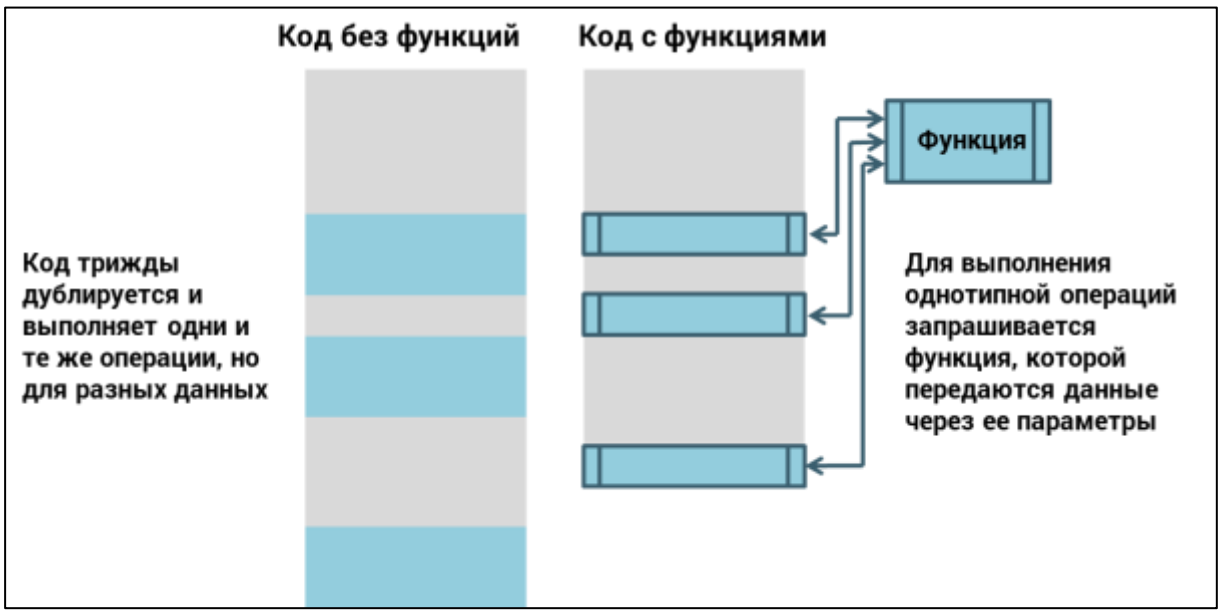


Рис. 4.107. Использование функций как подпрограмм

4.7.2. Виды функций

1. Функция без возврата значения

Функция без возврата значения не возвращает каких-либо значений. Задача такой функции – либо произвести определенные действия, либо преобразовать данные.

Функциям без возврата доступна команда **`return`**, которая принудительно завершает выполнение функции. В противном случае выход из функции происходит после выполнения последней команды в ее теле.

В других языках программирования такие функции часто называют *процедурами*.

Например, создадим функцию, которая выводит заданный параметр в окно консоли отладки браузера:

```
function print(text) {
    console.log(text);
}

print(2022);
print("Пример текста");
print({x:4, y:5});
```

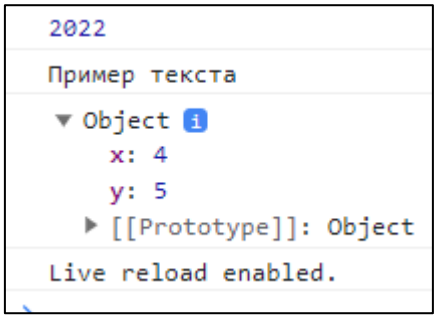


Рис. 4.108. Функция для вывода объекта в консоль отладки

2. Функция с возвратом значения

Функция с возвратом значения выполняет некоторую обработку данных и возвращает некоторое значение с помощью команды **`return`**. При этом любая возможная ветвь обязательно должна завершиться командой **`return`** с некоторым значением.

Результат работы такой функции можно присвоить другой переменной, либо использовать в операциях с переменными.

Например, создадим функцию вычисления суммы двух чисел:

```
function sum(a, b) {
    return a + b;
}

const s = 78 - sum(20, 34);
console.log(s);
```

Рис. 4.109. Функция для вычисления суммы двух чисел

Стоит заметить, что в силу динамической типизации функции можно передать вместо числовых строковые аргументы, и тогда она будет возвращать уже склеенную строку:

```
const s = sum("При", "вет!")
console.log(s);
```

В избежание подобных ситуаций следует предусмотреть возможные вариации данных и варианты их обработки.

4.7.3. Способы определения функции

Объявление функции

Объявление функции – это ее описание с использованием ключевого слова **function**. Объявление функции задает лишь ее описание и не работает само по себе.

```
function sum(a, b) {
    return a + b;
}
```

Вызов функции

Обращение к функции по имени и передача аргументов вызывает непосредственное ее *выполнение*.

```
const s = sum(20, 34);
```

Ссылка на функцию

В JavaScript функции являются объектами, поэтому их можно присваивать переменным, передавать в качестве параметров другим функциям и т.п.

В следующем примере создается переменная `operation`, которая позволяет ссылаться на функцию. При необходимости можно менять ссылку на другую функцию (операцию). При этом ссылке не указываются скобки при присваивании:

```
function sum(a, b) {
    return a + b;
}

function sub(a, b) {
    return a - b;
```

```
}

let operation = sub;
console.log(operation(23, 45));

operation = sum;
console.log(operation(8, 10));
```

```
-22
18
```

Рис. 4.110. Управление ссылками на функцию

4.7.4. Параметры функции

Параметры функции

Формальные параметры функции – это локальные переменные, описанные в ее заголовке. Они получают копии значений или ссылок на объекты, которые передаются функции при вызове.

```
function sum(a, b) {
    return a + b;
}
```

Рис. 4.111. Формальные параметры функции

Аргументы функции

Фактические аргументы функции – значения, которые передаются функции при ее вызове. Они копируются в формальные параметры функции.

```
const a = 10;
const b = 23;
const s = sum(a, b);
```

Рис. 4.112. Фактические аргументы функции

Особенности параметров

- Формальные параметры являются *копией* фактических аргументов.
- Изменение формального параметра не изменяет фактические аргументы, если они простого типа (числа, строки, логический тип).
- Для объектов изменение формального параметра влечет изменение компонент аргумента.

Это важно знать!

*Объекты в функцию передаются по **ссылке**. В этом случае параметр является не копией объекта, а копией ссылки на него в памяти. Поэтому внутри функции объекты можно модифицировать (см. далее).*

Рассмотрим несколько примеров.

При работе с простыми типами данных модификация формальных параметров в теле функции не влияет на внешние переменные.

```
function changeA(a, b) {
    a = 0;
    console.log(a, b);
}

const a = 10;
const b = 23;

console.log(a, b);
changeA(a, b);
console.log(a, b);
```

```
10 23
0 23
10 23
```

Рис. 4.113. Изменение формальных параметров простых типов

Здесь аргументы `a` и `b` являются числами, поэтому передаются в функцию по значению. Изменение параметра `a` внутри функции не влияет на внешнюю переменную `a`.

При работе с объектами ситуация меняется.

```
function changeFirst(array) {
    array[0] = 555;
}

function changeAllArray(array) {
    array = [1, 2, 3, 4, 5];
}

const arr = [10, 20, 30, 40];

console.log(arr);
changeFirst(arr);
console.log(arr);
changeAllArray(arr);
console.log(arr);
```

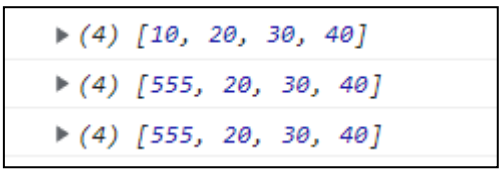


Рис. 4.114. Изменение ссылок на объекты

Массив является объектом и передается по ссылке. Функция `changeFirst()` меняет его первый элемент. При этом `array` является локальной копией ссылки на исходный массив `arr`.

А вот функция `changeAllArray()` меняет ссылку внутри функции, ссылаясь на новый массив. Иными словами, в этом месте разрывается связь между ссылками `arr` и `array`. Однако после выхода из функции эта локальная ссылка «разрушается», т.е. продолжаем работать с исходным массивом `arr`.

Важное замечание!

Важно разобраться, в чем разница между переменными значимого и ссылочного типа.

Необязательные параметры

1. Вариация числа аргументов

Функции можно передавать любое число аргументов.

Если аргумент не задан, то соответствующий параметр берется со значением `undefined`. Лишние же аргументы отсекаются.

```
function sum(a, b, c) {
    return a + b + c;
}

// a = undefined, b = undefined, c = undefined
console.log(sum());
// a = 1, b = undefined, c = undefined
console.log(sum(1));
// a = 1, b = 2, c = undefined
console.log(sum(1, 2));
// a = 1, b = 2, c = 3
console.log(sum(1, 2, 3));
// a = 1, b = 2, c = 3
console.log(sum(1, 2, 3, 4));
```

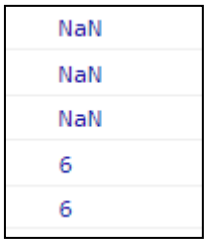


Рис. 4.115. Разное число аргументов и поведение функции

2. Контроль параметров

Параметры рационально проверять перед использованием. Если они не заданы, то следует определить некоторые значения по умолчанию (классический подход).

```
function sum(a, b, c) {
    if (a === undefined)
        a = 0;

    if (b === undefined)
        b = 0;

    if (c === undefined)
        c = 0;
```

```
    return a + b + c;
}

console.log(sum());             // a = 0, b = 0, c = 0
console.log(sum(1));            // a = 1, b = 0, c = 0
console.log(sum(1, 2));         // a = 1, b = 2, c = 0
console.log(sum(1, 2, 3));      // a = 1, b = 2, c = 3
```

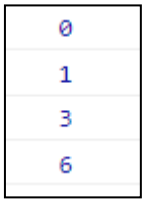


Рис. 4.116. Проверка и корректировка параметров

Есть и более лаконичный подход с использованием логического ИЛИ (классический подход):

```
function sum(a, b, c) {
    a = a || 0;
    b = b || 0;
    c = c || 0;
    return a + b + c;
}

console.log(sum());             // a = 0, b = 0, c = 0
console.log(sum(1));            // a = 1, b = 0, c = 0
console.log(sum(1, 2));         // a = 1, b = 2, c = 0
console.log(sum(1, 2, 3));      // a = 1, b = 2, c = 3
```

3. Параметры по умолчанию

В ES6 внедрена работа с *параметрами по умолчанию* (наилучшая стратегия обработки непереданных параметров).

```
function sum(a = 0, b = 0, c = 0) {
    return a + b + c;
}

console.log(sum());             // a = 0, b = 0, c = 0
console.log(sum(1));            // a = 1, b = 0, c = 0
console.log(sum(1, 2));         // a = 1, b = 2, c = 0
console.log(sum(1, 2, 3));      // a = 1, b = 2, c = 3
```

Задача «Комбинаторика»

Создадим каталог *Комбинаторика*. Требуется реализовать форму, которая осуществляет расчеты по некоторым формулам комбинаторики. А именно, необходимо создать функции:

- `factorial(n)` – возвращает факториал заданного числа;
- `nonrepCombo(n,m)` – возвращает число сочетаний без повторов из n по m.

Результаты вычисления требуется отобразить на странице:

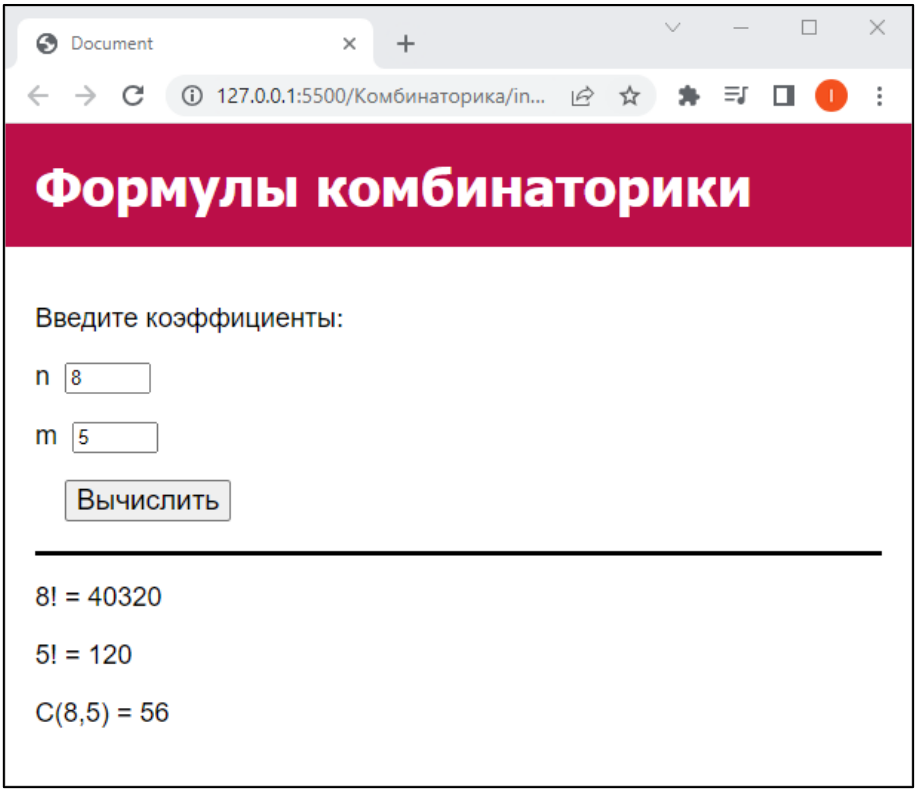


Рис. 4.117. Расчет по формулам

Код для интерфейса:

Глава 4\Комбинаторика\index.html

```
<header class="header">
    <h1 class="header__title">Формулы комбинаторики</h1>
</header>

<main class="main">
    <p>Введите коэффициенты:</p>
    <p>n<input type="text"
               id="koeffN"
               class="main__input"
```

```
                size="3"
                maxlength="3"
                value="1">
    </p>
    <p>m<input type="text"
                id="koeffM"
                class="main__input"
                size="3" maxlength="3"
                value="1">
    </p>
    <button id="start" class="main__button">
        Вычислить
    </button>
    <div id="result" class="answer"></div>
</main>

<script src="code.js"></script>
```

Глава 4\Комбинаторика\style.css

```
body {
    font-family: Arial, Helvetica, sans-serif;
    font-size: 18px;
    margin: 0;
}

h1, h2, h3 { font-family: Tahoma, sans-serif; }

.header__title {
    background: #bb0e48;
    color: #fff;
    margin: 0;
    padding: 20px;
}

.main { padding: 20px; }

.main__input {
    display: inline-block;
    margin-left: 10px;
}

.main__button {
    margin-left: 20px;
    margin-bottom: 20px;
    font-size: 105%;
```

```
}

.answer { border-top: 3px solid #000; }
```

Прейдем к реализации скрипта. В начале опишем функции.

Вычисление факториала организуем циклом. При $n = 0$ цикл не выполняется и сразу возвращается 1 (это соответствует определению факториала: $0! = 1$):

Глава 4\Комбинаторика\code.js

```
function factorial(n = 1) {
    n = Math.abs(n);

    let factor = 1;

    for (let i = 1; i <= n; i++) {
        factor *= i;
    }

    return factor;
}
```

Число комбинаций m объектов из n различных объектов вычисляется по формуле:

$$C_n^m = \frac{n!}{m!\,(n-m)!}.$$

Поскольку функция вычисления факториала уже реализована, то используем ее:

Глава 4\Комбинаторика\code.js

```
function nonrepCombo(n = 1, m = 1) {
    return factorial(n) /
           (factorial(m) * factorial(n - m));
}
```

Функции реализованы. Далее получаем ссылки на поля ввода, кнопку и блок вывода расчетов. В обработчике кнопки получаем значения из полей и формируем запись с результатами:

Глава 4\Комбинаторика\code.js

```
const koeffN = document.querySelector("#koeffN");
const koeffM = document.querySelector("#koeffM");
const start = document.querySelector("#start");
const data = document.querySelector("#result");
```

```
start.onclick = function () {
    // Считываем коэффициенты из текстовых полей
    // и переводим их в целые числа
    let n = parseInt(koeffN.value);
    let m = parseInt(koeffM.value);

    let code = `<p>${n}! = ${factorial(n)}</p>`;
    code += `<p>${m}! = ${factorial(m)}</p>`;
    code += `<p>C(${n},${m}) = ${nonrepCombo(n, m)}</p>`;

    data.innerHTML = code;
}
```

Rest и Spread операторы

Rest-оператор

Остаточные параметры (*rest-оператор*) позволяют собрать переменное число аргументов в массив. Если передается несколько параметров, то остаточный должен быть последним.

Rest-оператор поддерживается в ES6 и выше.

В следующем примере логическая функция проверяет, существует ли указанный элемент в заданном массиве (хотя-бы один). Здесь поиск организуется обычным перебором:

```
function elementExists(elem, ...array) {
    for (const e of array) {
        if (e === elem)
            return true;
    }

    return false;
}

console.log(elementExists(4, 5, 7, 4, 78, 2));
console.log(elementExists(0, 5, 2));
```

```
true
false
```

Рис. 4.118. Rest-оператор

Первый аргумент функции задает искомое число. Второй параметр представлен rest-оператором и позволяет передавать любое число параметров, которые будут сформированы в массив.

Spread-оператор

Оператор распространения (*spread-оператор*) позволяет разбить массив на список отдельных аргументов при передаче его функции.

Spread-оператор поддерживается в ES6 и выше.

В следующем примере spread-оператор позволяет корректно вычислить минимальное значение в массиве чисел, потому как функция `min()` объекта `Math` не может работать с массивом (только с набором элементов):

```
let array1 = [4, -3, 23, 0, 12];
let array2 = [6, -5, 2, 4];

console.log(Math.min(array1));
console.log(Math.min(...array1));
console.log(Math.min(...array1, ...array2));
```

```
NaN
-3
-5
```

Рис. 4.119. Spread-оператор

4.7.5. Функции как методы объектов

Функции внутри объектов

ООП основа

JavaScript является объектно-ориентированным языком программирования. Несмотря на то, что ранее мы создавали функции как самодостаточные модули на уровне скрипта, они являются частью глобального объекта **`window`**. Схема на рис. 4.120. демонстрирует, что объект `window` является корневым для остальных крупных объектов, используемых в модели DOM. Среди них – объект `document`, который мы использовали для доступа к функциям, управляющим структурой разметки.

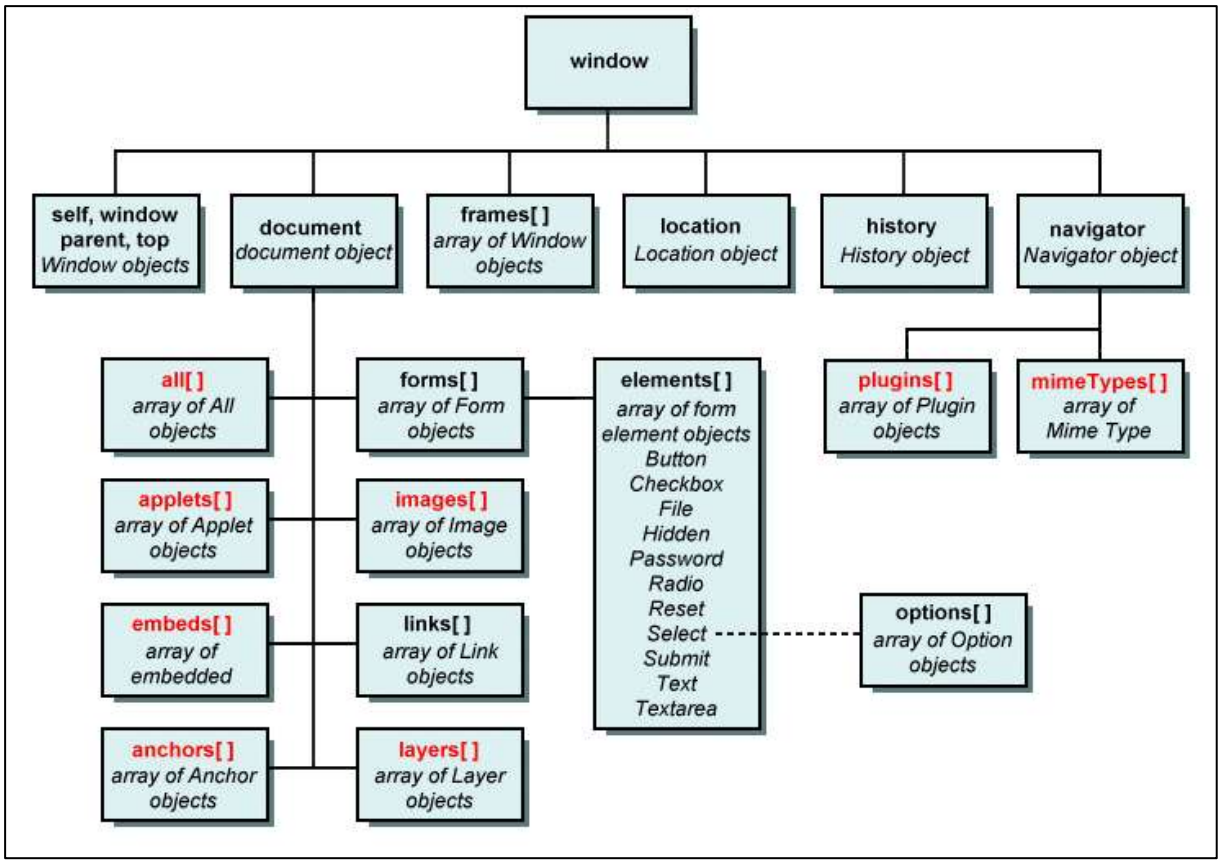


Рис. 4.120. Структура DOM

Вообще говоря, описывая какую-либо переменную или функцию на уровне скрипта с помощью `var` (для `let` это не распространяется), мы привязываем этот элемент к глобальному элементу `window`.

Иными словами, в следующем примере переменная `size` становится свойством объекта `window`, и оба обращения к ней равноценны:

```
var size = 680;

console.log(size);
console.log(window.size);
```

Однако находясь внутри объекта `window` уточнять его название не требуется, т.к. все свойства объекта доступны внутри него напрямую. То же правило распространяется и на функции.

Методы объекта

Таким образом функции являются подчиненными каким-либо объектам. Если свойства объекта характеризуют его количественно-качественные показатели, то функции выстраивают некоторую логику

действий и обработки данных, которые берутся из параметров функции или из свойств самого объекта.

Определение

Метод *– это функция, привязанная к объекту в качестве его свойства.*

Описание методов объектов несколько отличается от предложенного ранее декларативного подхода.

Рассмотрим простой пример. Опишем объект, характеризующий кота. Помимо некоторых свойств зададим функцию «мяуканья» `speak()`, которая возвращает строку:

```
const Barsik = {
    nickname: "Барсик",
    type: "кот",
    age: 3.7,

    speak: function () {
        return `${Barsik.nickname} говорит "Мяу"`;
    }
}

console.log(Barsik.speak());
```

```
Барсик говорит "Мяу"
>
```

Рис. 4.121. Вызов метода объекта

В контексте созданного объекта `speak()` тоже является его свойством. Чтобы методам были доступны свойства объекта, необходимо явно указать название объекта `Barsik`:

```
Barsik.nickname
```

Вызов метода извне также осуществляется с привязкой к объекту:

```
console.log(Barsik.speak());
```

Ссылка this

Предыдущий подход с явным указанием названия объекта внутри метода небезопасен и мало универсален. Например, при описании объекта похожей структуры можно случайно сослаться на другой объект. С другой стороны часто требуется создавать не один объект, а функцию *конструктор*, которая формирует объекты одного типа, но с разными параметрами.

Для этих целей в JavaScript используется специальная ссылка **`this`**, которая в контексте метода объекта ссылается на этот объект. Иными словами, более корректным будет следующее описание тела функции:

```
const Barsik = {
    nickname: "Барсик",
    type: "кот",
    age: 3.7,

    speak: function () {
        return `${this.nickname} говорит "Мяу"`;
    }
}
```

Важное замечание!

*Стоит отметить, что поведение команды **`this`** в JavaScript имеет существенные отличия от модели реализации этой ссылки в других ООП языках. Это связано с тем, что ее действие зависит от **контекста выполнения**. Мы не будем погружаться в нюансы, поскольку это требует дополнительной подготовки. В рамках текущего курса мы используем **`this`** в контексте методов, а значит ссылаемся на объект, который их содержит.*

Обновленный синтаксис

В ES6 появился более короткий способ описания методов внутри объекта. Теперь не требуется писать ключевое слово `function`:

```
const Barsik = {
    nickname: "Барсик",
    type: "кот",
    age: 3.7,

    speak() {
        return `${this.nickname} говорит "Мяу"`;
    }
}

console.log(Barsik.speak());
```

4.7.6. Формы описания функции

Роль формы

JavaScript предоставляет возможность описывать функции в нескольких формах. Это позволяет гибко оперировать ими.

1. Объявление функции (Function Declaration)

- Функция определяется в основном потоке кода и предполагает отложенный вызов.
- Функция обрабатывается интерпретатором до ее возможного вызова, т.е. она может быть описана выше или ниже места, откуда ее вызывают.
- Такая форма описания функции соответствует «классическому» понятию функции.

```
function degree(x, n) {
    return Math.pow(x, n);
}

let deg = degree(4.6, 3);
```

```
let deg = degree(4.6, 3);

function degree(x, n) {
    return Math.pow(x, n);
}
```

Оба варианта успешно работают с одинаковым результатом. Функция срабатывает только при явном вызове

Рис. 4.122. Объявление функции

2. Функциональное выражение (Function Expression)

- Функция определяется в контексте какого-либо выражения, например присваивания, в роли параметра другой функции и т.п.
- Функция обрабатывается в момент вызова.
- Такая функция не требует имени, поэтому часто называется *анонимной*.

```
let degree = function(x, n) {
    return Math.pow(x, n);
}

let deg1 = degree(4.6, 3);
let deg2 = degree(0.5, 2);
```

```
let deg1 = degree(4.6, 3);
let deg2 = degree(0.5, 2);

let degree = function(x, n) {
    return Math.pow(x, n);
}
```

Второй вариант вызова вызовет ошибку, т.к. функциональное выражение определено ниже вызова

Рис. 4.123. Функциональное выражение

Это полезно знать!

Описание функции в форме функционального выражения – очень распространенная практика в JavaScript. Объекты JS содержат множество встроенных методов, которым в качестве параметров передаются другие функции (вспомните методы ***`find()`****,* ***`map()`*** *и т.п.). В подобных случаях их вызов происходит однократно. А значит их гораздо удобнее описать прямо на месте в виде функционального выражения (см. далее).*

Стрелочная функция

Описание

В спецификацию ES6 ввели более короткую форму описания функциональных выражений – *стрелочные функции*.

Они позволяют:

- избавиться от слова `function`;
- опустить круглые скобки, если у функции один параметр;
- убрать `return`, если тело функции состоит только из одной команды.

Например, для поиска в массиве индекса первого положительного четного числа можно записать функциональное выражение в старой форме:

```
const array = [-56, -5, 23, 22, 67, 0, 23];

let firstEvenPositive = array.findIndex(function (x) {
    return (x > 0) && (x % 2 == 0);
});
```

Стрелочная функция сделает код более лаконичным:

```
const array = [-56, -5, 23, 22, 67, 0, 23];

let firstEvenPositive = array.findIndex(
    x => (x > 0) && (x % 2 == 0)
);
```

Использование анонимных или стрелочных функций и функциональных выражений очень распространено в JavaScript.

Отличие от обычных функций

Это важно знать!

Однако стрелочные функции не полностью аналогичны обычным! Важное отличие – стрелочные функции не имеют собственного контекста и они связываются с ближайшим в иерархии контекстом. Иными словами, ***this*** *внутри стрелочной функции указывает не на объект, где она описана, а на ближайшую родительскую сущность, имеющую свой контекст* ***this****.*
Контекст в JavaScript связан с местом определения функции – это называется ***лексическим окружением****.*

Рассмотрим простой пример. Создадим объект с одним свойством и двумя функциями (методами): простой и стрелочной.

Обратите внимание, что при вызове простой функции `this` ссылается на контекст объекта `obj`, где она описана. А вот стрелочная функция не имеет контекста и ближайшей для нее является контекст глобального объекта `window`:

```
const obj = {
    prop: 10,

    simpleFun() {
        return this;
    },

    arrowFun: () => this
}

console.log(obj.simpleFun());
console.log(obj.arrowFun());
```

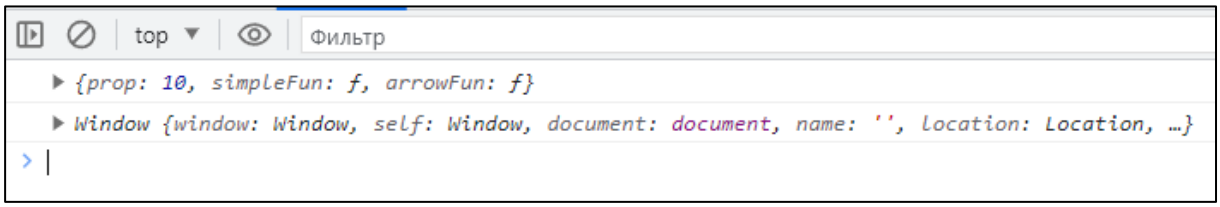


Рис. 4.124. Разница между контекстами обычных и стрелочных функций

4.7.7. События

Понятие событий

Определение

Событие *– это сигнал системы о некотором произошедшем действии, которое в дальнейшем может быть обработано.*

События являются одним из важнейших механизмов реакции программы на определенные условия.

Распространенный пример – нажатие на кнопку: ОС, приложение, браузер должны немедленно отреагировать на это событие и осуществить ряд соответствующих операций, «закрепленных» за этой кнопкой.

Очевидно, что нажатие на клавишу (клавиатуры) – это тоже событие, которому соответствует определенная серия действий и т.д.

При работе приложений и систем могут возникать условия, которые должны посылать сигнал другим системам. И здесь вновь востребованы события.

Принцип работы

При возникновении события система генерирует сигнал и предоставляет программный механизм реакции на событие.

В WEB события запускаются в рамках браузера и обычно привязываются к конкретному элементу. Это может быть один элемент, набор элементов, документ HTML, или все окно браузера.

Можно выделить следующие принципы работы событий:

1. Возможное событие привязывается к какому-либо элементу, например – DOM-элементу.
2. Каждое событие имеет *обработчик событий* – это функция(и) JavaScript, которая будет запускаться при срабатывании события.
3. Говорят, что обработчик события *подписывается* на событие. Если этого не происходит, то, вообще говоря, никакой реакции на событие не последует.
4. Часто обработчики событий еще называют *прослушивателями событий*.

1. Назначение обработчика события

Обычно обработчик события назначается в качестве свойства DOM-объекта. Ссылка на DOM-объект получается одним из методов объекта `document`, например – `getElementById()`.

Названия событий-свойств начинаются с приставки «on».

Глава 4\События\События 1\index.html

```
<h1>Подключение обработчика события</h1>
<button id="start">Нажми</button>

<script src="code.js"></script>
```

Глава 4\События\События 1\code.js

```
const start = document.getElementById("start");

start.onclick = function() {
    // Здесь размещается код обработки события:
}
```

В приведенном примере для кнопки указан атрибут `id` – идентификатор. С его помощью можно будет подключить к ней событие. С помощью метода `getElementById()` получаем ссылку на DOM-

элемент с идентификатором `start`, т.е. кнопку. Заметим, что название переменной/константы может совпадать с названием идентификатора.

Далее цепляем к свойству элемента `start` функцию-обработчик. Говорят, что элемент *подписывается* на событие `onclick`, срабатывающее при щелчке на элемент.

Разумеется, функция обработчик может быть описана отдельно и подключаться ссылкой:

Глава 4\События\События 1\code.js

```
const start = document.getElementById("start");
start.onclick = startButtonClick;

function startButtonClick() {
    // Здесь размещается код обработки события:
}
```

Здесь скобки для `startButtonClick` не нужны, т.к. делается ссылка на функцию.

2. Обработка нескольких событий одного элемента

Одному DOM-элементу можно назначить сразу несколько обработчиков на разные события.

Глава 4\События\События 2\index.html

```
<h1>Обработка нескольких событий одного элемента</h1>
<button id="start">Нажми или наведи курсор</button>

<script src="code.js"></script>
```

Глава 4\События\События 2\code.js

```
const start = document.getElementById("start");

start.onclick = function() {
    // Сработает при клике на кнопку:
}

start.onmouseover = function() {
    // Сработает при наведение курсора на кнопку:
}
```

3. Несколько обработчиков одного события

Однако предыдущий подход не позволяет к одному событию привязать сразу несколько обработчиков.

Решить эту проблему позволяет метод **addEventListener()**, которому передается два параметра:

- название события (приставка «on» здесь уже не нужна);
- функция обработчик или ссылка на нее.

Глава 4\События\События 3\index.html

```
<h1>Несколько обработчиков события</h1>
<button id="start">Нажми</button>

<script src="code.js"></script>
```

Глава 4\События\События 3\code.js

```
const start = document.getElementById("start");

start.addEventListener("click", function () {
    // Первый обработчик клика
});

start.addEventListener("click", function () {
    // Второй обработчик клика
});
```

Зачастую удобнее группировать подписку на обработчики по ссылкам:

Глава 4\События\События 3\code.js

```
const start = document.getElementById("start");

start.addEventListener("click", startButtonClickOne);
start.addEventListener("click", startButtonClickTwo);

function startButtonClickOne() {
    // Первый обработчик клика
}

function startButtonClickTwo() {
    // Второй обработчик клика
}
```

Это полезно знать!

Не трудно заметить, что в приведенных примерах продемонстрировано две формы подключения функции-обработчика события.
Первый – в качестве функционального выражения (анонимной функции):

```
start.onclick = function() {
    // Здесь размещается код обработки события:
}
```

Второй – в качестве объявления функции и ссылки на нее:

```
start.onclick = startButtonClick;

function startButtonClick() {
    // Здесь размещается код обработки события:
}
```

Очевидно, что функциональные выражения удобно использовать при обработке одного события.
Объявление функции и ссылка на нее более эффективна в случае подключения нескольких событий или обработчиков.

Это полезно знать!

События можно использовать, чтобы отслеживать завершение разметки страницы.
Ранее в примерах мы подключали скрипты в конце документа.
Однако скрипт можно подключить и в заголовке ***<head>****.*
В этом случае основной файл скрипта должен дождаться загрузки всех элементов страницы, используя следующую конструкцию с событием ***onload****:*

```
window.onload = function(){
    // Здесь код скрипта
}
```

Некоторые события

Таблица 4.10. События мыши

Событие	Описание
`onclick`	Возникает при клике на элемент.
`ondblclick`	Возникает при двойном клике на элемент.
`onmouseover`	Возникает при наведении курсора мыши на элемент.
`onmousedown / onmouseup`	Возникает при нажатии на кнопку мыши / ее отжатии.
`onmousemove`	Возникает при движении мыши.

Таблица 4.11. События клавиатуры

Событие	Описание
`onkeydown`	Возникает при нажатии клавиши.
`onkeyup`	Возникает при отпускании нажатой клавиши.
`onkeypress`	Возникает сразу после нажатия символьной клавиши, т.е. нажатие приводит к появлению символа.

Таблица 4.12. События окна браузера

Событие	Описание
`onload`	Возникает сразу после загрузки окна.
`onresize`	Возникает при изменении размеров окна.
`onscroll`	Возникает при использовании ползунка прокрутки.

4.7.8. Пример «Цветовая палитра»

Постановка задачи

Пусть требуется реализовать простейшую RGB-палитру с возможностью менять цвет с помощью ползунков:

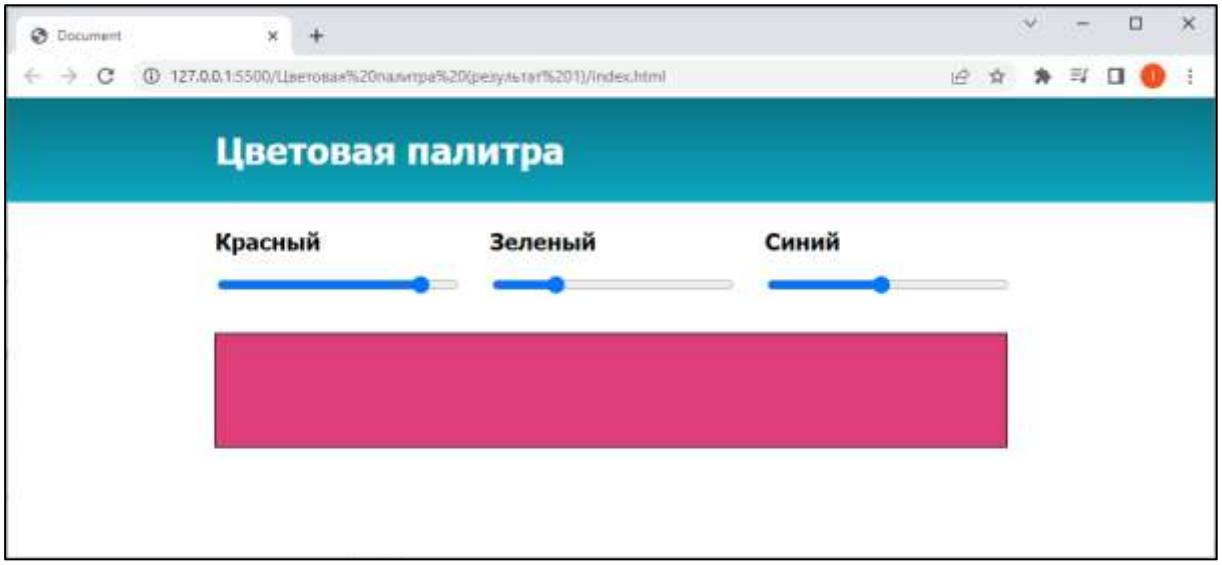


Рис. 4.125. Цветовая палитра

Внешний дизайн

Создадим проект *Цветовая палитра*. Вставим стандартную стартовую разметку. Подключим стилевой файл и код скрипта:

Глава 4\Цветовая палитра\index.html

```
<!DOCTYPE html>
<html lang="ru">

<head>
    <meta charset="UTF-8">
    <meta http-equiv="X-UA-Compatible" content="IE=edge">
    <meta name="viewport" content="width=device-width,
initial-scale=1.0">
    <title>Document</title>
    <link rel="stylesheet" href="style.css">
</head>

<body>
```

```
    <script src="code.js"></script>
</body>

</html>
```

В стилевой файл добавим базовую настройку тегов:

Глава 4\Цветовая палитра\style.css

```
* { box-sizing: border-box; }

body {
    font: 16px Arial;
    margin: 0;
}

h1, h2, h3 { font-family: Tahoma; }

h2 { font-size: 125%; }
```

Разметим область заголовка. Оформление далее будем задавать посредством подключаемых классов. Классы `row` и `colimn-X` будут описаны далее.

Глава 4\Цветовая палитра\index.html

```
<header class="header">
    <div class="row">
        <div class="column-1">
            <h1 class="header__title">
                Цветовая палитра
            </h1>
        </div>
    </div>
</header>
```

Рис. 4.126. Область заголовка

В стилях оформляем класс `row`, который будет содержать уровень с одной или несколькими колонками. Классы `column-X` делаем плавающими блоками, каждый из них будет задаваться шириной в 1, 1/2, 1/3 относительно родительского блока соответственно.

Глава 4\Цветовая палитра\style.css

```
.row {
    max-width: 720px;
    min-width: 480px;
    margin: 0 auto;
    overflow: auto;
}

.column-1,
.column-2,
.column-3 {
    float: left;
    overflow: auto;
    padding: 0.3em 2%;
}

.column-1 { width: 100%; }
.column-2 { width: 50%; }
.column-3 { width: 33.33%; }
```

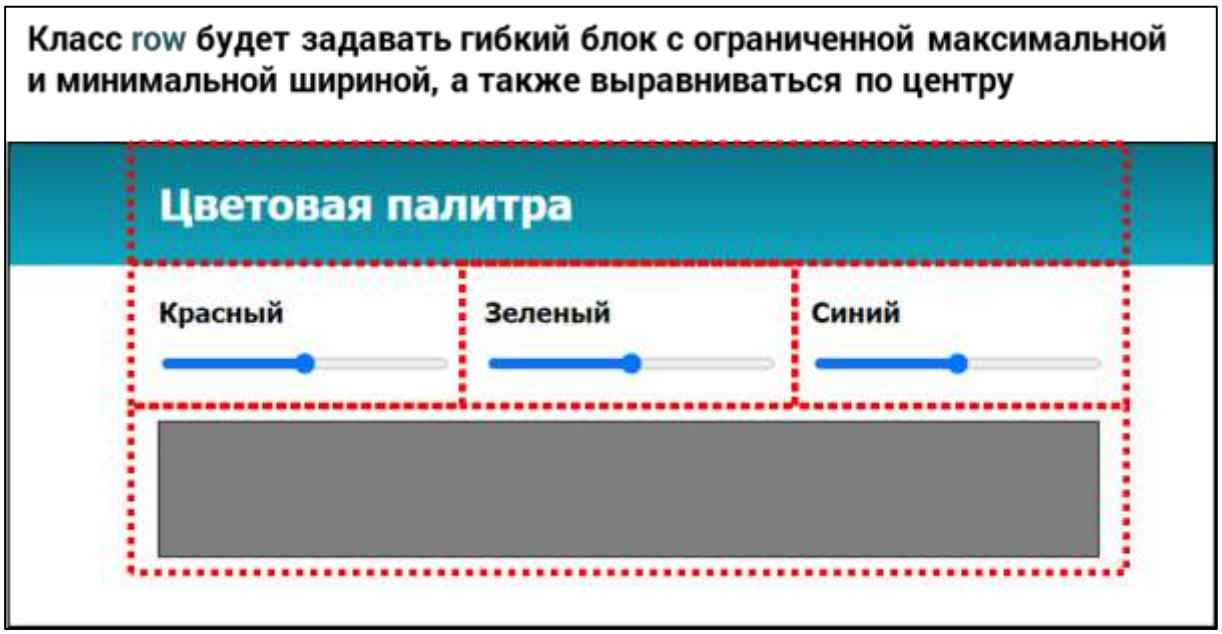


Рис. 4.127. Колонки в разметке

Оформляем блок с заголовком.

Глава 4\Цветовая палитра\style.css

```
.header {
    background: linear-gradient(#0b7486, #0da7c2);
}

.header__title {
    color: white;
}
```

В блоке `<main>` размечаем в три колонки ползунки. С помощью атрибутов `min` и `max` задаем минимальное и максимальное значение ползунка. Атрибут `step` определяет величину шага при перетаскивании ползунка. В блоке `color-block` будет отображаться цвет.

Глава 4\Цветовая палитра\index.html

```
<main>
    <div class="row">
        <div class="column-3">
            <h2>Красный</h2>
            <input type="range" id="red-color"
                   min="0" max="255"
                   step="1">
        </div>
        <div class="column-3">
            <h2>Зеленый</h2>
            <input type="range" id="green-color"
                   min="0" max="255"
                   step="1">
        </div>
        <div class="column-3">
            <h2>Синий</h2>
            <input type="range"
                   id="blue-color" min="0" max="255"
                   step="1">
        </div>
        <div class="column-1">
            <div id="color-block"></div>
        </div>
    </div>
</main>
```

Ползунки растянем по всей доступной ширине.

Блоку, отображающему цвет, задаем лишь рамку, фиксированную высоту и небольшие поля:

Глава 4\Цветовая палитра\style.css

```
#red-color,
#green-color,
#blue-color {
    width: 100%;
}

#color-block {
    border: 1px solid black;
    height: 100px;
    margin: 20px 0;
}
```

Программирование функционала

Переходим к реализации скрипта.

В начале создадим ссылки на DOM-элементы: каждый ползунок и блок с цветом:

Глава 4\Цветовая палитра\code.js

```
const rangeRed = document.getElementById("red-color");
const rangeGreen = document.getElementById("green-color");
const rangeBlue = document.getElementById("blue-color");
const palette = document.getElementById("color-block");
```

Любое изменение одного из трех ползунков должно менять цвет соответствующей красной, зеленой или голубой компоненты. Нам потребуется переменная, в которую будем сохранять текущие цвета. Для удобства сгруппируем все три цвета в одном объекте `color`.

Предварительно опишем объект `DefualtColor`, который задает цвета по умолчанию. Так мы внесем некоторую определенность, что за значения взяты изначально.

Для примера все компоненты изначально возьмем «половиной» из спектра от 0 до 255. Теперь магические константы спрятаны в объекте `DefaultColor`:

Глава 4\Цветовая палитра\code.js

```
const DefaultColor = {
    Red: 127,
    Green: 127,
    Blue: 127
}
```

```
let color = {
    red: DefaultColor.Red,
    green: DefaultColor.Green,
    blue: DefaultColor.Blue
};
```

Сразу после загрузки страницы зададим ползункам значения согласно установленным по умолчанию компонентам: их берем из свойств ранее инициализированного объекта `color`.

Также заливаем цвет фона блока. Здесь используем CSS функцию `rgb(x,y,z)`.

Кроме того, эту часть кода внесем в функцию, чтобы формально сгруппировать связанные по смыслу операции одной командой – вызовом функции `setDefaultColors()`:

Глава 4\Цветовая палитра\code.js

```
setDefaultColors();

function setDefaultColors() {
    rangeRed.value = color.red;
    rangeGreen.value = color.green;
    rangeBlue.value = color.blue;
    palette.style.background =
        `rgb(${color.red},${color.green},${color.blue})`;
}
```

Внутри функции ссылки на объекты `rangeRed`, `rangeGreen`, `rangeBlue`, `palette` и `color` видны, поскольку они описаны на одном уровне с функцией и являются для нее в некотором смысле глобальными. Напомним, что переменные, описанные внутри функции, получают уже локальную область видимости.

Изменение ползунка должно менять соответствующую цветовую компоненту. Здесь требуется подключение обработчика события **`oninput`**, который сработает при любом изменении положения (т.е. значения) ползунка. В отличите от `onchange` оно срабатывает при малейшем изменении положения ползунка.

Например, цепляем обработчик для ползунка красной компоненты. Обработчик немедленно должен скопировать новое значение красной компоненты в переменную `color` и обновить фон заливки блока.

Глава 4\Цветовая палитра\code.js

```
rangeRed.oninput = function () {
    color.red = this.value;
    palette.style.background =
        `rgb(${color.red},${color.green},${color.blue})`;
}
```

Здесь ссылка `this` ссылается на объект `rangeRed`.

Абсолютно такие же обработчики создаются для ползунков, меняющих зеленую и голубую компоненту:

Глава 4\Цветовая палитра\code.js

```
rangeGreen.oninput = function () {
    color.green = this.value;
    palette.style.background =
        `rgb(${color.red},${color.green},${color.blue})`;
}

rangeBlue.oninput = function () {
    color.blue = this.value;
    palette.style.background =
        `rgb(${color.red},${color.green},${color.blue})`;
}
```

Рефакторинг

Нетрудно заметить, что код реализации всех трех обработчиков очень похож: разница только в ссылках на ползунки и компоненты цвета, которые они меняют.

Поступим более рационально. Создадим функцию `changeColor()`, которая будет обрабатывать указанный ползунок и связанную с ним компоненту. Ссылку на ползунок и цвет передадим посредством параметров функции.

Тогда внутри каждого обработчика достаточно вызвать эту функцию и передать ей соответствующие аргументы:

Глава 4\Цветовая палитра\code.js

```
function changeColor(slider, current_component) {
    color[current_component] = slider.value;
    palette.style.background =
        `rgb(${color.red},${color.green},${color.blue})`;
}
```

```
// Изменение ползунка красного цвета
rangeRed.oninput = function () {
    changeColor(this, "red");
}

// Изменение ползунка зеленого цвета
rangeGreen.oninput = function () {
    changeColor(this, "green")
}

// Изменение ползунка синего цвета
rangeBlue.oninput = function () {
    changeColor(this, "blue")
}
```

Здесь `this` ссылается на объект-ползунок, относительно которого вызвано событие `oninput`.

Второй аргумент передает текстовое название свойства.

Обратите внимание, что внутри функции `changeColor()` параметр `current_component` используется для обращения к свойству объекта `color` через синтаксис ассоциативного массива. Мы не можем написать его как `color.current_component`.

Вопросы для самопроверки

1. Что представляют собой функции и для каких целей они используются?
2. В чем отличие между функциями с возвратом и без возврата значения?
3. Чем отличается объявление функции от ее вызова?
4. Приведите примеры ситуаций, когда рационально создавать переменные-ссылки на функции.
5. Что важно учитывать при работе с параметрами функции?
6. Опишите рекомендуемую практику работы с необязательными параметрами.
7. Для чего нужны rest- и spread-операторы при работе с функциями и в чем между ними разница?
8. Что называют методом объекта и как его можно описать?
9. Опишите особенности работы ссылки `this` в обычных и стрелочных функциях.

10. Перечислите возможности и ограничения синтаксиса объявления функции и функционального выражения.
11. В чем важность обработки событий при разработке ПО?
12. Как можно подписать несколько объектов на одно событие?
13. Опишите процедуру подписки объекта на одно и несколько событий.

Практикум

Задание 1

1. Создайте каталог *Комбинаторика*.
2. Повторно изучите и реализуйте рассмотренную одноименную задачу из пункта 4.7.4 (рис. 4.117).
3. Опишите третью функцию `repCombo(n,m)`, которая возвращает число сочетаний с повторами из n по m. (Самостоятельно найдите формулу в сети Интернет)
4. Образец оформления изображен на рис. 4.133.

Задание 2

1. Создайте каталог *Функции*.
2. Подготовьте разметку, подключите файл со стилями и скрипт. Скопируйте и изучите соответствующий код в конце задания.
3. Допишите реализацию функций:
   a. `isEvenPositive(x)`, возвращающую значение `true`, если число `x` положительное и четное, иначе `false`;
   b. `isOddNegative(x)`, возвращающую значение `true`, если число `x` отрицательное и нечетное, иначе `false`;
   c. `banner(text)`, которая оборачивает текстовую строку `text` внутрь тега `<div class='box'></div>` и возвращает эту модифицированную строку (класс `box` уже описан в CSS-файле).
4. Продемонстрируйте работу этих функций.
5. Образец оформления изображен на рис. 4.129.

Глава 4\Функции\index.html

```
<h1>Функции</h1>
<div id="result"></div>
```

```
<script src="code.js"></script>
```

Глава 4\Функции\style.css

```
body {
    font: 16px Tahoma;
}

h1, h2, h3 {
    border-bottom: 2px solid #000;
}

.box {
    background: #81ff68;
    font-weight: bold;
    font-size: 120%;
    padding: 10px;
    margin: 10px;
    border: 4px solid #05e98a;
    border-radius: 6px;
    text-align: center;
}
```

Глава 4\Функции\code.js

```
// Описание функций
function isEvenPositive(x){
    // Реализовать тело функции  <<-------------
}

function isOddNegative(x){
    // Реализовать тело функции  <<-------------
}

function banner(text){
    // Реализовать тело функции  <<-------------
}

const result = document.getElementById("result");

let number = 34;
let code = "";

// Используем как логические функции в проверке условий
if (isEvenPositive(number))
    code += `Число ${number} четное положительное`;
```

```
if (isOddNegative(number))
    code += `Число ${number} нечетное орицательное`;

code += banner("Привет!");
code += banner("Продам гараж");

result.innerHTML = code;
```

Задание 3

1. Создайте каталог *Операторы rest и spread*.
2. Подготовьте типовую начальную разметку, подключите CSS-файл и скрипт. Изучите и скопируйте в файлы код в конце задания.
3. Разберите код разметки, реализацию функции `getPositiveSum()`, возвращающую среднее арифметическое положительных чисел среди любого количества перечисленных аргументов.
4. Следуя комментариям, пошагово напишите код обработки действий кнопки, чтобы по нажатию на нее выводилось среднее от положительных чисел, введенных в текстовое поле через пробел.
5. Образец оформления изображен на рис. 4.130.

Глава 4\Операторы rest и spread\index.html

```
<header class="header">
    <h1 class="header__title">
        Операторы rest и spread
    </h1>
</header>
<main class="main">
    <p>Введите числа через пробел:</p>
    <p><input type="text" id="numbers"
              class="main__input" value="1">
    </p>
    <button id="start" class="main__button">
        Вычислить
    </button>
    <div id="result" class="answer"></div>
</main>

<script src="code.js"></script>
```

Глава 4\Операторы rest и spread\style.css

```css
body {
    font-family: Arial, Helvetica, sans-serif;
    font-size: 18px;
    margin: 0;
}

h1, h2, h3 { font-family: Tahoma, sans-serif; }

.header__title {
    background: #000000;
    color: #fff;
    margin: 0;
    padding: 20px;
}

.main { padding: 20px; }

.main__input {
    width: 100%;
    font-size: 110%;
}

.main__button {
    margin-bottom: 20px;
    font-size: 105%;
}

.answer { border-top: 3px solid #000; }
```

Глава 4\Операторы rest и spread\code.js

```js
// Функция считает среднее арифметическое среди
// положительных чисел
function positiveAverage(...array) {
    // Если массив пустой (аргументы не переданы),
    // то возвращаем нуль
    if (array == null)
        return 0;

    // В нее накапливаем сумму положительных чисел
    let sum = 0;
    // Счетчик положительных чисел
    let posit_counter = 0;
```

```
    // Перебираем элементы, ищем искомые
    for (const e of array) {
        if (e > 0) {
            sum += e;
            posit_counter++;
        }
    }

    // Если нашли хотя-бы одно положительное,
    // то считаем среднее и возвращаем его из функции
    if (posit_counter > 0)
        return sum / posit_counter;

    // Иначе положительных нет, возвращаем нуль
    return 0;
}

// Получаем ссылки на поле ввода, кнопку и блок
// вывода результата соответственно
const numbers = document.querySelector("#numbers");
const start = document.querySelector("#start");
const data = document.querySelector("#result");

// Подключаем обработку события для кнопки
start.onclick = function () {
    // 1. В переменную text запишите строку из текстового
    //    поля numbers, используя свойство value.
    // 2. В переменную text_arr запишите массив, который
    //    разбивает строку text на отдельные элементы,
    //    разделенные пробелом (используйте метод split).
    // 3. В переменную nums запишите массив, который
    //    получен из массива text_arr преобразованием
    //    каждого элемента из строки в вещ. число
    //    (используйте метод map)
    // 4. В переменную posit_av запишите результат вызова
    //    функции positiveAverage относительно параметра
    //    nums (не забудьте передать его с помощью
    //    rest-оператора)
    // 5. В блок data запишите строку с результатом
}
```

Задание 4

1. Создайте каталог *Треугольник*.
2. Подготовьте типовую начальную HTML-разметку, подключите внешний CSS-файл и JS-скрипт.
3. Изучите и скопируйте в файлы код в конце задания.
4. Самостоятельно опишите объект `Triangle`, который задается тремя свойствами `a`, `b`, `c` – стороны треугольника и тремя методами:
   a. `exists()` – метод проверяет, является ли фигура треугольником;
   b. `perimeter()` – возвращает периметр треугольника, иначе – 0.
   c. `square()` – возвращает площадь треугольника, если не существует, то– 0.
5. Последовательно вывести разметку (см. рис. 4.131).

Глава 4\Треугольник\index.html

```
<div class="box">
    <h1 class="title">Треугольник</h1>
    <div id="result"></div>
    <script src="code.js"></script>
</div>
```

Глава 4\Треугольник\style.css

```
body {
    background: url("wallpaper.jpg");
    font: 1.1em Arial;
    margin: 0;
}

.box {
    background: rgba(255, 255, 255, 0.6);
    padding: 15px 25px;
    width: 480px;
    margin: 0 auto;
    border-radius: 10px;
    margin-top: 40px;
}

.title { color: brown; }

p { color: #3d0a6d; }
```

Задание 5

1. Создайте каталог *Стрелочные функции*.
2. Подготовьте типовую начальную HTML-разметку, подключите внешний CSS-файл и JS-скрипт.
3. Изучите и скопируйте в файлы код в конце задания.
4. Перепишите вызовы метода `filter()`, используя синтаксис стрелочных функций.
5. Образец оформления изображен на рис. 4.132.

Глава 4\Стрелочные функции\index.html

```
<h1>Стрелочные функции</h1>
<div id="result"></div>

<script src="code.js"></script>
```

Глава 4\Стрелочные функции\style.css

```
body {
    font: 16px Arial;
}

h1, h2, h3 {
    font-family: Tahoma, Geneva, Verdana, sans-serif;
}
```

Глава 4\Стрелочные функции\code.js

```
const result = document.getElementById("result");

const assortment = [
    { goods: "Яблоки Голден", cost: 93.90},
    { goods: "Яблоки Гала", cost: 139.90 },
    { goods: "Яблоки Гренни смит", cost: 119.90 },
    { goods: "Бананы", cost: 89.90 },
    { goods: "Бананы фасованные", cost: 83.90 },
    { goods: "Авокадо", cost: 399.90 },
    { goods: "Апельсины", cost: 99.90 },
];

// Переписать стрелочными функциями
const bananas = assortment.filter(function(fruit) {
    return fruit.goods.includes("Бананы");
});

const econom = assortment.filter(function(fruit) {
```

```
    return (50.00 <= fruit.cost) && (fruit.cost <=
110.99);
});

let code = "<h2>Бананы в асортименте</h2>";

for (const e of bananas) {
    code += `<p>${e.goods} - ${e.cost}р.</p>`;
}

code += "<h2>Товары категории 'Эконом'</h2>";

for (const e of econom) {
    code += `<p>${e.goods} - ${e.cost}р.</p>`;
}

result.innerHTML = code;
```

Задание 6

1. Скопируйте ранее выполненный в рамках главы 3 проект *Видеомонтаж*.
2. Подключите в конце файл JS-скрипта.
3. Напишите скрипт, который по двойному щелчку *ЛКМ* будет подсвечивать желтым цветом фон любого текстового абзаца. Повторный щелчок отключает выделение.
4. Для реализации вам потребуется:
    a. в конце файла с CSS-стилями добавить класс `marker`, который задает желтый цвет фона;
    b. в скрипте получить ссылку на все теги в разметке (используйте метод `getElementByTagName()`);
    c. в цикле перебора по полученной коллекции абзацев для каждого подцепить событие `ondblclick`, в котором в зависимости от наличия класса `marker` у абзаца его подцепить, либо убрать, если он уже подключен (так будет работать переключатель);
    d. дополнительно самостоятельно изучите возможности свойства `classList` и его методов `contains()`, `add()` и `remove()`;
5. После успешной реализации попробуйте упростить код, используя метод `toggle()`.
6. Образец оформления изображен на рис. 4.133.

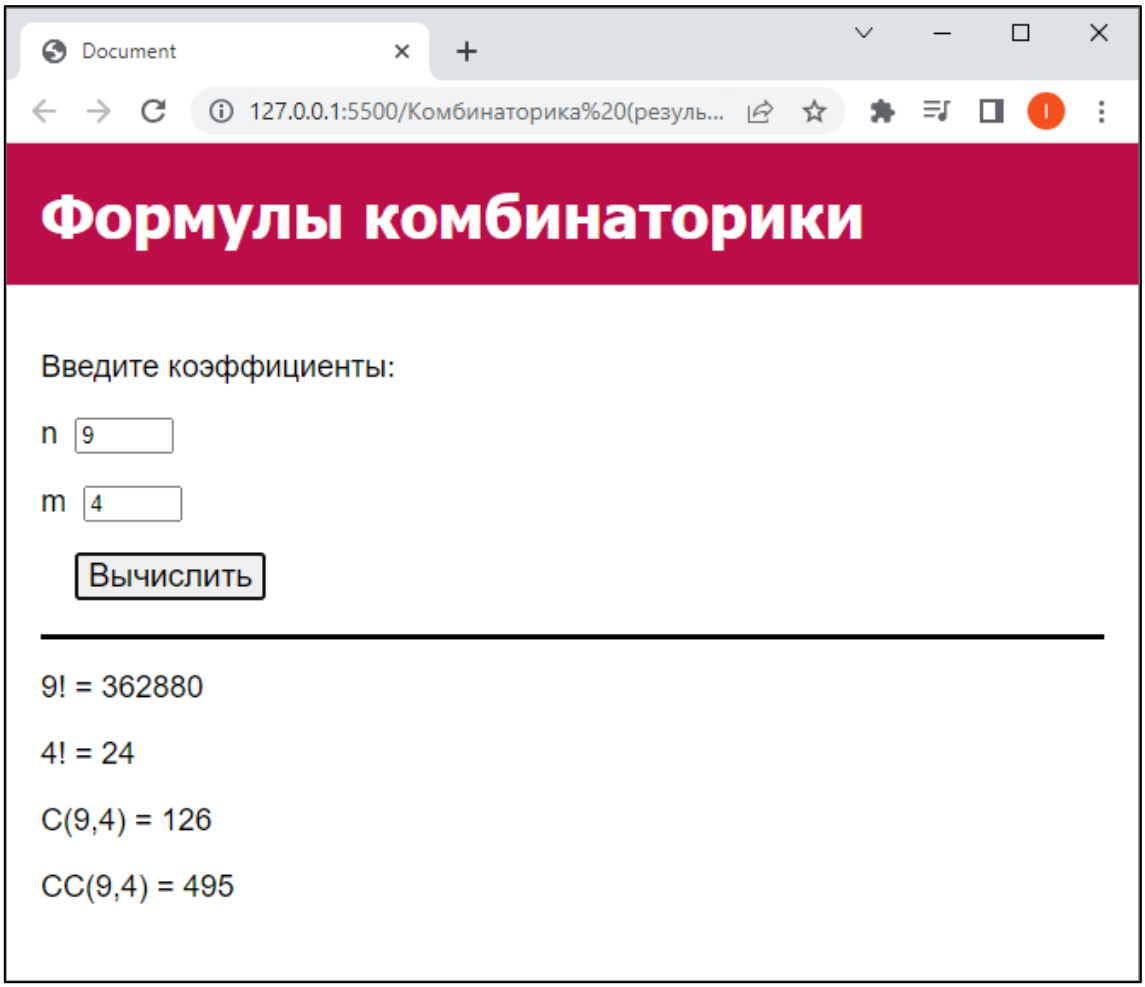


Рис. 4.128. Образец выполнения задания 1

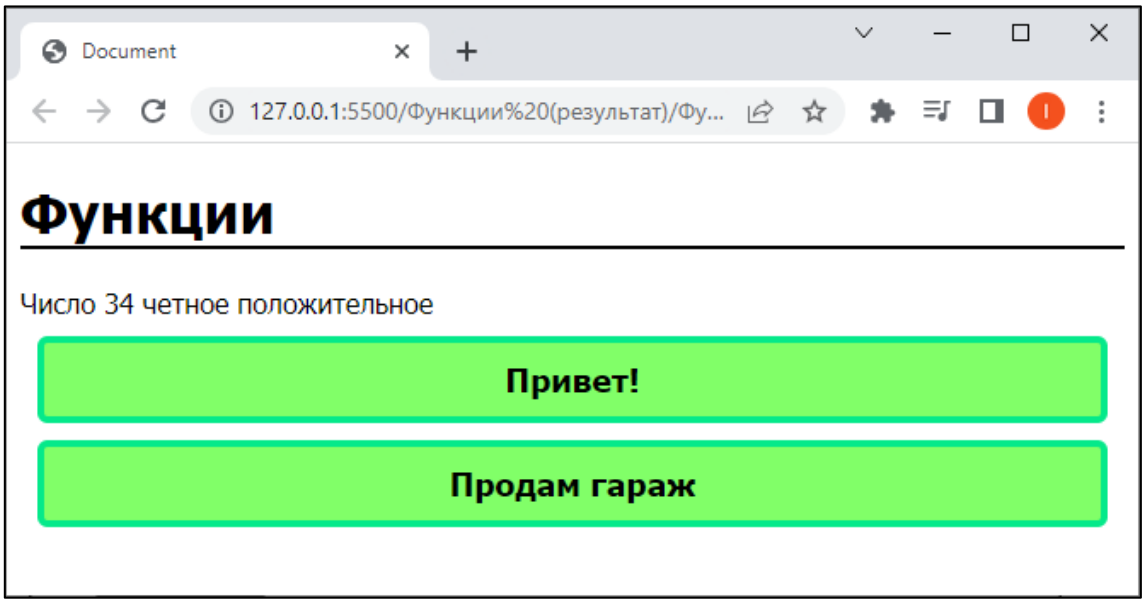


Рис. 4.129. Образец выполнения задания 2

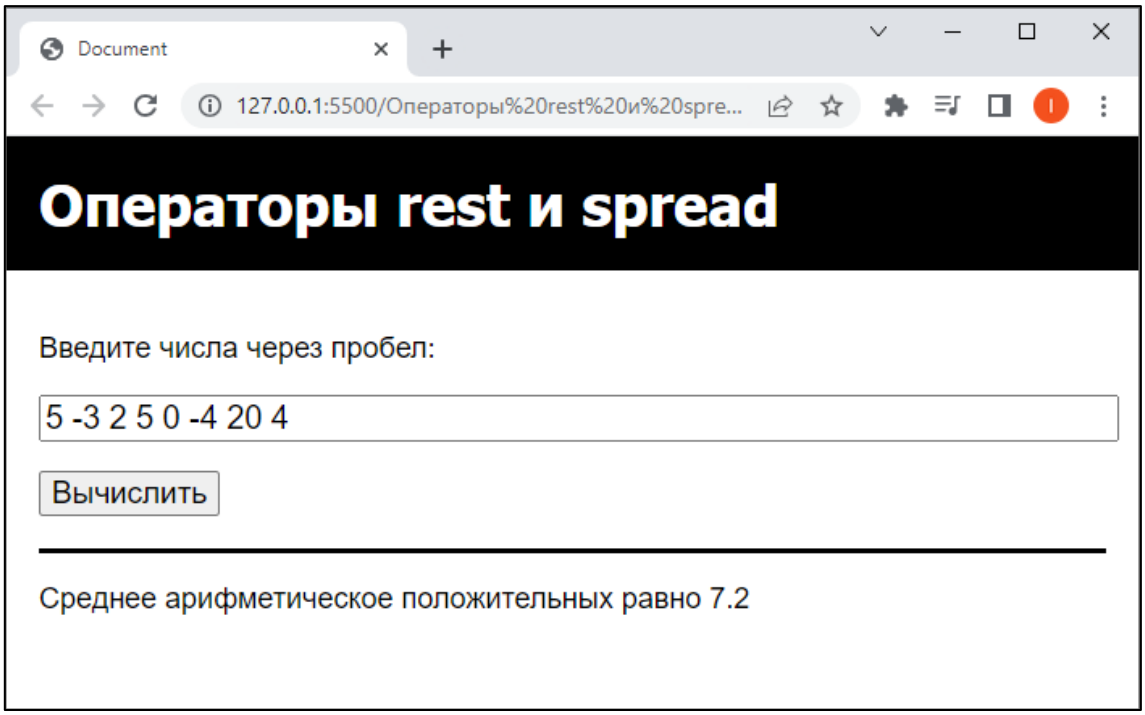


Рис. 4.130. Образец выполнения задания 3

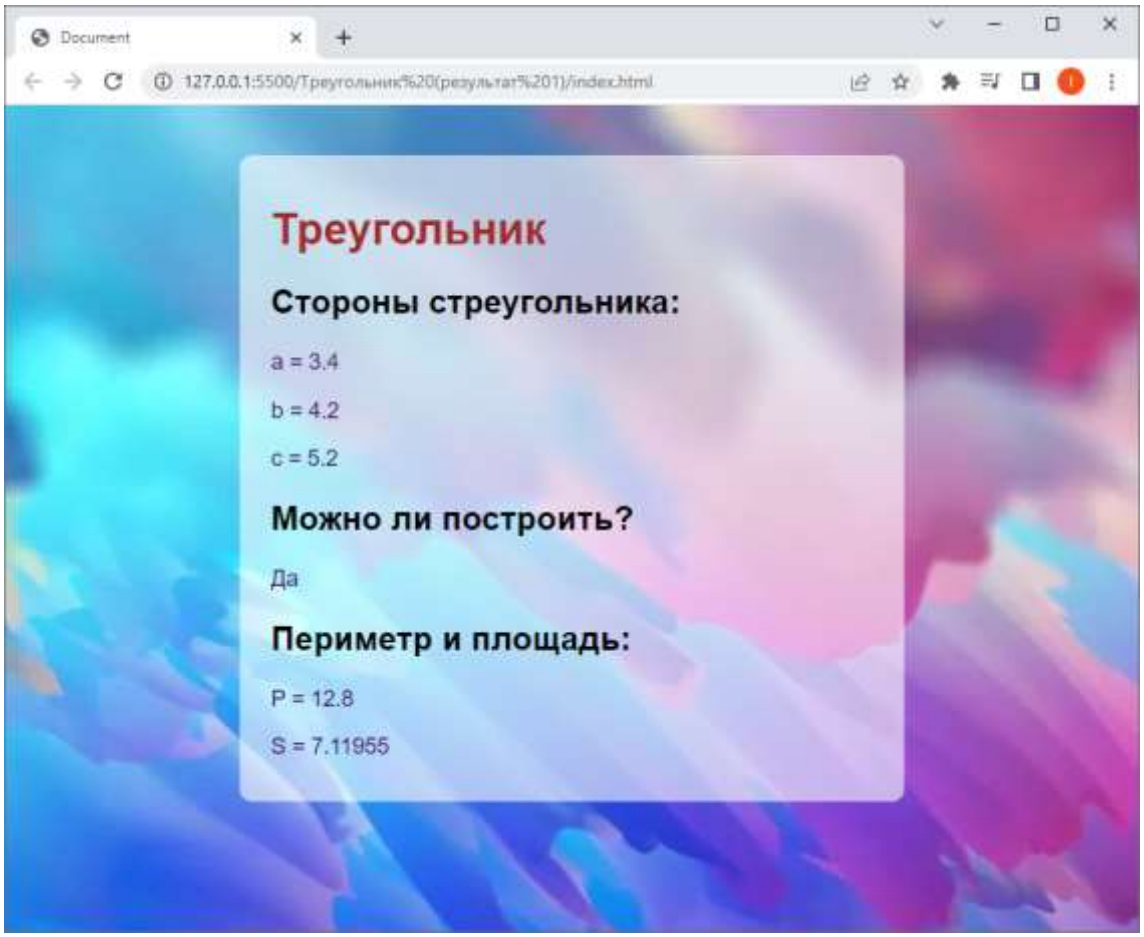


Рис. 4.131. Образец выполнения задания 4

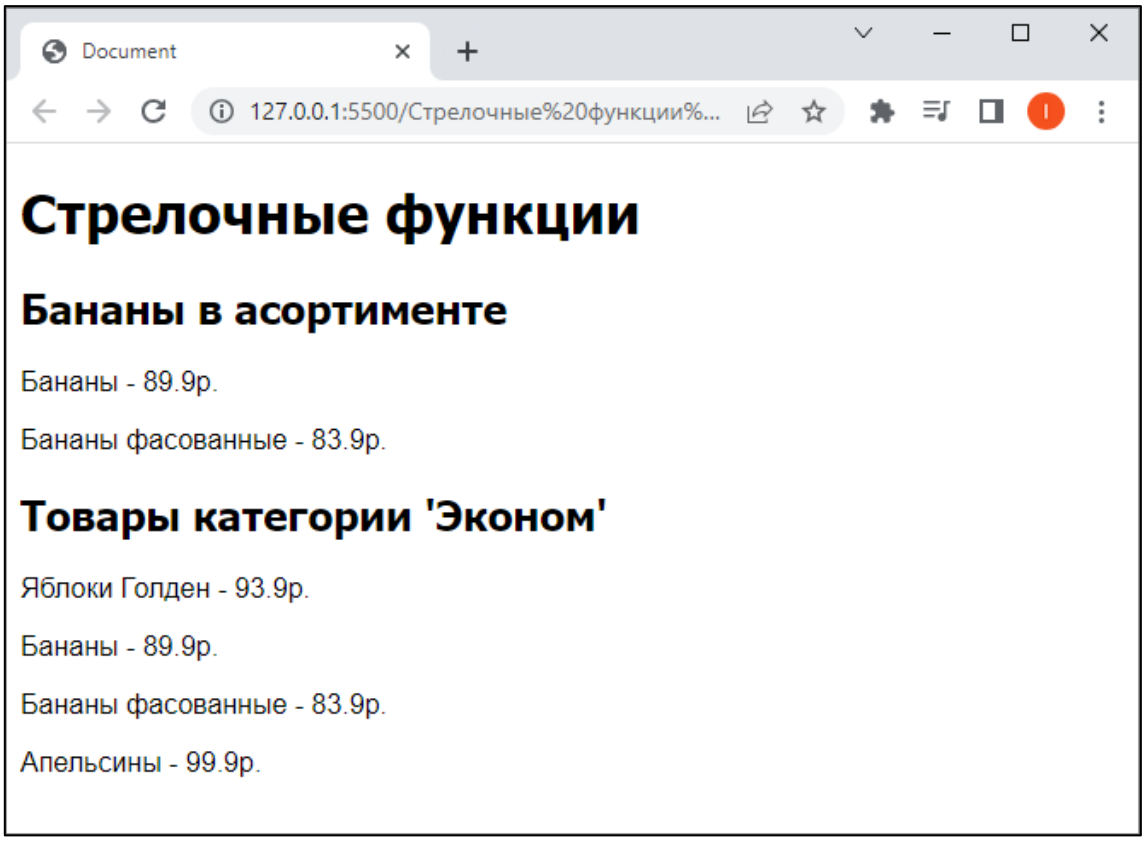


Рис. 4.132. Образец выполнения задания 5

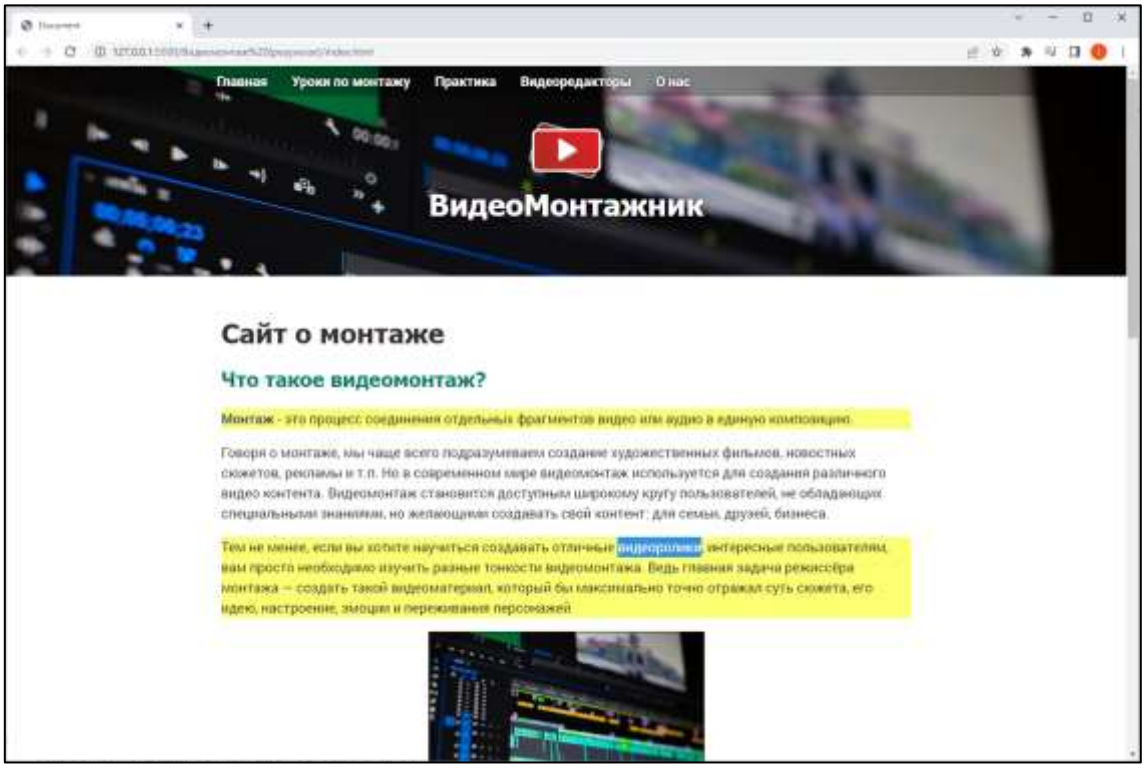


Рис. 4.133. Образец выполнения задания 6

ЗАКЛЮЧЕНИЕ

Рекомендации по дальнейшему изучению frontend-технологий

Технология HTML5

В главе 2 достаточно подробно были изложены основные возможности языка разметки HTML и нововведения технологии HTML5.

При дальнейшем изучении HTML5 особое внимание следует уделить следующим вопросам:

- оформление мультимедийных элементов (изображений, работа с видео и аудио, многочисленные атрибуты тегов);
- изучение примеров разметки с использованием тегов структурирования и секционирования;
- разметка элементов форм (рекомендуется изучать совместно с JavaScript);
- работа с метатегами (для SEO-оптимизации).

Также важно учиться работать с плагином ускоренной верстки Emmet, разбивать проект на отдельные страницы и каталоги.

Технология CSS3

Глава 3 была посвящена каскадным таблицам стилей CSS. Авторы постарались изложить наиболее фундаментальные и важные аспекты работы CSS3 и стилизации дизайна веб-страниц. Были затронуты простейшие принципы адаптивного дизайна.

Однако возможности CSS3 существенно шире и для более полного понимания и погружения в технологию следует изучить следующие вопросы:

- синтаксис селекторов атрибутов (вариации);
- псевдоклассы и псевдоэлементы;
- комбинированные селекторы;

- правила вычисления специфичности и проблемы конфликта стилей;
- режимы позиционирования элементов в блоках;
- импорт стилей из других файлов;
- переходы, трансформации и анимация;
- стилизация форм;
- медиа-запросы, их возможности и принципы адаптивного дизайна;
- устройство адаптивных сеток;
- модели CSS Grid и Flex Box.

После более детального изучения CSS3 следует переходить к знакомству со связанными технологиями:

- препроцессоры CSS (SASS, LESS и др), расширяющие возможности синтаксиса;
- изучение простых CSS-фреймворков с адаптивными сетками и классами;
- изучение хотя-бы одного CSS-фреймворка с поддержкой JavaScript (Bootstrap, Foundation, Bulma и пр.);
- методологию верстки (БЭМ, AMCSS или др.).

Язык программирования JavaScript

Глава 4 была посвящено языку программирования JavaScript. Изучение возможностей этого языка и связанных с ним технологий требует длительного времени и практики, если читатель планирует использовать полученные навыки веб-программирования в решении профессиональных задач.

Основываясь на изложенном в пособии материале, дальнейшее продвижение в JavaScript может предполагать изучение следующих вопросов:

- изучение возможностей базового синтаксиса (обработка исключений, особенности работы функций, составные типы и структуры данных);
- регулярные выражения в обработке текста;
- особенности реализации ООП модели в JS (работа с объектами, прототипы и принципы наследования);

- изучение нововведений ECMAScript начиная с ES6 (классы, импорт и экспорт модулей);
- промисы и async/await (асинхронное программирование);
- работа с объектной моделью документа DOM;
- браузерная модель BOM;
- работа с сетевыми запросами;
- обработка данных в браузере;
- анимация и графика.

После изучения основных возможностей JavaScript можно начать знакомство со множеством прикладных вопросов:

- настройка системы контроля версий;
- работа с пакетными менеджерами для проектов;
- изучение одного или нескольких популярных фреймворков (React, Angular, Vue.js, Svetle, Solid JS);
- изучение концепций современного CSS в паре с JS;
- методология тестирования веб-сайтов и веб-приложения;
- веб-компоненты;
- практика написания типизированного кода (использование TypeScript);
- изучение политики безопасности обработки данных и работа с аутентификацией;
- технологии рендеринга страниц на стороне сервера;
- генераторы статических сайтов;
- разработка мобильных веб-приложений и т.д.

Другие технологии

Множество вопросов веб-разработки было затронуто в главе 1. В частности, рассмотрены технологии frontend- и backend-разработки, ПО для разработчиков.

Отметим, что изложенный в практикуме материал может использоваться в профильном обучении школьников основам веб-разработки и программирования.

Об использовании пособия в обучении

Особенности курса

Текущее пособие разработано по учебно-методическим материалам, которые использовались авторами при подготовке студентов направления 44.03.05 «Педагогическое образование (с двумя профилями подготовки)». Чтение курса по HTML, CSS и JavaScript осуществлялось в период с 2017 по 2023 учебные годы в рамках дисциплин «Современные информационные технологии», «Информационные технологии в образовании», «Прикладная информатика», «Разработка веб-сайтов», «Практикум по решению задач на ЭВМ».

В пособии учтен опыт использования актуальных спецификаций рассматриваемых технологий и современного инструментального и прикладного ПО, необходимый разработчику. Материалы пособия дополнены новыми темами и расширены примерами.

Рекомендации преподавателям

При планировании рабочих программ дисциплины на прохождение предложенного в практикуме курса рекомендуется выделять не менее 36 ч для аудиторных занятий (18 лабораторных занятий) и не менее 54 ч для самостоятельной работы студента.

Полноценный курс нацелен на студентов физико-математического профиля подготовки:

- технология HTML5, глава 2 (8 ч – аудит., 12 ч – СРС);
- технология CSS3, глава 3 (12 ч – аудит., 18 ч – СРС);
- язык JavaScript, глава 4 (16 ч – аудит., 24 ч – СРС).

Для студентов гуманитарных направлений достаточно упрощенного курса из глав 2-3 (возможно частичное выполнение практических заданий по выбору преподавателя).

Для интенсификации курса преподавателям рекомендуется заранее подготовить файлы примеров и заданий, которые рассматриваются в практикуме, и приложить их в качестве раздаточного электронного материала (для последующей работы в редакторе кода).

БИБЛИОГРАФИЧЕСКИЙ СПИСОК

1. *Браун, Э.* Изучаем JavaScript: руководство по созданию современных веб-сайтов, 3-е изд. : Пер. с англ. – СПб. : ООО «Альфа-книга», 2017. – 368 с.
2. *Васильев, А. Н.* JavaScript в примерах и задачах / Алексей Васильев. – Москва : Издательство «Э», 2017. – 720 с.
3. *Дакетт, Дж.* JavaScript и jQuery. Интерактивная веб-разработка : Пер. с англ М. А. Райтмана. – М. : Издательство «Э», 2017. – 640 с.
4. *Дронов, В. А.* HTML5, CSS3 и Web 2.0. Разработка современных Web-сайтов. – СПб. : БХВ-Петербург, 2011. – 416 с.
5. *Закас, Н.* JavaScript для профессиональных веб-разработчиков : Пер. с англ. А. Лютича. – СПб. : Питер, 2015. – 960 с.
6. *Кириченко, А. В.* JavaScript для FrontEnd-разработчиков. Написание. Тестирование. Развертывание. – СПб. : Наука и Техника, 2020. – 320 с.
7. *Крокфорд, Д.* Как устроен JavaScript. – СПб. : Питер, 2019. – 304 с.
8. *Макфарланд, Д.* Большая книга CSS3. 3-е изд. – СПб. : Питер, 2014. – 608 с.
9. *Макфарланд, Д.* JavaScript и jQuery: исчерпывающее руководство / Дэвид Макфарланд. – Пер. с англ. М. А. Райтмана. – М. : Эксмо, 2015. – 880 с.
10. *Мейер, Э.* CSS – каскадные таблицы стилей. Подробное руководство. 3-е изд. – Пер. с англ. – СПб. : Символ-Плюс, 2008. – 576 с.
11. *Никсон, Р.* Создаем динамические веб-сайты с помощью PHP, MySQL, JavaScript, CSS и HTML5. 3-е изд. – СПб.: Питер, 2015. – 688 с.
12. *Печников, В. Н.* Создание Web-страниц и Web-сайтов. Самоучитель. – М. : Триумф, 2013. – 470 с.

13. *Прасти, Н.* Введение в ECMAScipt 6/. пер. с англ. Рагимов Р. Н. – М. : ДМК Пресс, 2016. – 176 с.
14. *Пьюривал, С.* Основы разработки веб-приложений. СПб. : Питер, 2016. – 272 с.: ил. – (Серия «Бестселлеры O'Reilly»).
15. *Роббинс, Дж.* HTML5, CSS3 и JavaScript. Исчерпывающее руководство / Дженифер Роббинс; [пер. с англ. М. А. Райтман]. – 4-е издание. М.: Эксмо, 2014 – 528 с.
16. *Роббинс, Дж.* HTML5: карманный справочник, 5-е издание.: Пер. с англ. – М. : ООО «И.Д. Вильямс», 2015. – 192 с.
17. *Сухов, К.* HTML5 – путеводитель по технологии. – М.: ДМК Пресс, 2013. – 352 с.
18. *Флэнаган, Д.* JavaScript. Полное руководство. 7-е изд. : Пер. с англ. – СПб. : ООО «Диалектика», 2021. – 720 с.
19. *Фрейн, Б.* HTML5 и CSS3. Разработка сайтов для любых браузеров и устройств. — СПб. : Питер, 2014. – 304 с.
20. *Хоган, Б.* HTML5 и CSS3. Веб-разработка по стандартам нового поколения. 2-е изд. – СПб. : Питер, 2014. – 320 с.
21. *Хольцнер, С.* HTML5 за 10 минут, 5-е изд. : Пер. с англ. – М. : ООО «И.Д. Вильямс», 2011. — 240 с.
22. *Якубович, Д. А.* Основы WEB-разработки: учебно-методическое пособие для проведения лабораторных занятий / Д. А. Якубович, Е. С. Еропова, И. А. Еропов / Владимирский гос. ун-т имени Александра Григорьевича и Николая Григорьевича Столетовых. – Владимир : Издательство «Шерлок-пресс», 2017. – 102 с.

Интернет-ресурсы

1. Простое пособие по сетевой модели OSI для начинающих [Электронный ресурс] / Режим доступа: https://selectel.ru/blog/osi-for-beginners/. – Дата обращения: 4.09.2022.
2. Структура сети Интернет: основные принципы работы. Основные характеристики интернета [Электронный ресурс] / Режим доступа: https://sukachoff.ru/noutbuki/. – Дата обращения: 5.09.2022.

3. Введение в WEB-технологии – понятие Интернет и всемирная паутина [Электронный ресурс] / Режим доступа: https://webonto.ru/vvedenie-v-web-tehnologii/. – Дата обращения: 7.09.2022.
4. Что такое модель OSI и зачем она нужна: препарируем слоёный пирог интернета [Электронный ресурс] / Режим доступа: https://skillbox.ru/. – Дата обращения: 9.09.2022.
5. Сетевые протоколы: базовые понятия и описание самых востребованных правил [Электронный ресурс] / Режим доступа: https://selectel.ru/blog/network-protocols/. – Дата обращения: 9.09.2022.
6. Что такое TCP/IP и как работает этот протокол [Электронный ресурс] / Режим доступа: https://timeweb.com/ – Дата обращения: 9.09.2022.
7. Как работает DNS (domain name system)? [Электронный ресурс] / Режим доступа: https://neoserver.ru/kak-rabotaet-dns – Дата обращения: 13.09.2022.
8. Служба WWW. Протокол HTTP [Электронный ресурс] / Режим доступа: https://moodle.kstu.ru/ – Дата обращения: 17.09.2022.
9. Службы Интернета [Электронный ресурс] / Режим доступа: http://kgau.ru/istiki/umk/ismar/c_8_3.htm – Дата обращения: 18.09.2022.
10. URI, URL, URN. Что это, чем отличаются [Электронный ресурс] / Режим доступа: https://alekseev74.ru – Дата обращения: 22.09.2022.
11. Кто такой веб-разработчик и чем он занимается? [Электронный ресурс] / Режим доступа: https://blog.skillfactory.ru/kto-takoj-veb-razrabotchik-i-chem-on-zanimaetsya/ – Дата обращения: 5.10.2022.
12. Как освоить фронтенд-разработку в 2022 году: дорожная карта [Электронный ресурс] / Режим доступа: https://tproger.ru/articles/frontend-roadmap-2021/#part6 – Дата обращения: 13.10.2022.
13. Как освоить бэкенд-разработку в 2022 году: дорожная карта [Электронный ресурс] / Режим доступа:

https://tproger.ru/articles/backend-roadmap-2021/ – Дата обращения: 18.10.2022.
14. И швец, и жнец. Кто такой Full-stack разработчик и как им стать? [Электронный ресурс] / Режим доступа: https://cdto.work/2021/04/22/full-stack/ – Дата обращения: 26.10.2022.
15. Фреймворк [Электронный ресурс] / Режим доступа: https://blog.skillfactory.ru/glossary/framework/ – Дата обращения: 6.11.2022.
16. Для чего нужен фреймворк и как его выбрать [Электронный ресурс] / Режим доступа: https://practicum.yandex.ru/blog/chto-takoe-framework/ – Дата обращения: 6.11.2022.
17. Обзор редакторов кода [Электронный ресурс] / Режим доступа: https://htmlacademy.ru/blog/soft/editors-for-the-coders – Дата обращения: 17.11.2022.
18. Какие программы использовать в веб-дизайне? [Электронный ресурс] / Режим доступа: https://vc.ru – Дата обращения: 23.11.2022.
19. Фронтенд-фреймворки [Электронный ресурс] / Режим доступа: https://blog.skillfactory.ru/ – Дата обращения: 23.11.2022.
20. Что такое вертска сайта [Электронный ресурс] / Режим доступа: https://web-valley.ru/articles/verstka-dlya-sajta – Дата обращения: 16.01.2023.
21. Ресурсы для разработчиков от разработчиков [Электронный ресурс] / Режим доступа: https://developer.mozilla.org/ru/ – Дата обращения: 23.01.2023.
22. Руководство по HTML5 и CSS3 [Электронный ресурс] / Режим доступа: https://metanit.com/web– Дата обращения: 7.02.2023.
23. Современный учебник по JavaScript [Электронный ресурс] / Режим доступа: https://learn.javascript.ru/ – Дата обращения: 2.03.2023.

ОБ АВТОРАХ

Якубович Денис Андреевич – старший преподаватель кафедры математического образования и информационных технологий Владимирского государственного университета.

Сфера научных интересов: дифференциальные уравнения в математической физике.

Преподаваемые дисциплины: «Информационные технологии в образовании»; «Информационные технологии в профессиональной деятельности»; «Практикум по решению задач на ЭВМ», «Прикладная информатика».

E-mail: yakubovich.studylib@mail.ru

Еропова Елена Станиславовна – кандидат педагогических наук, доцент кафедры математического образования и информационных технологий Владимирского государственного университета.

Сфера научных интересов: проблемы информационных технологий в образовании.

Преподаваемые дисциплины: «Информационные технологии в образовании»; «Информационные технологии в профессиональной деятельности»; «Основы математической обработки информации».

E-mail: eropova13061962@mail.ru

Учебное электронное издание

ЯКУБОВИЧ Денис Андреевич
ЕРОПОВА Елена Станиславовна

ПРАКТИКУМ ПО ПРИКЛАДНОЙ ИНФОРМАТИКЕ: ВЕБ-РАЗРАБОТКА НА БАЗЕ HTML5, CSS3 И JAVASCRIPT

Издается в авторской редакции

Системные требования: Intel от 1,3 ГГц; Windows XP/7/8/10; Adobe Reader; дисковод CD-ROM.

Тираж 25 экз.

Владимирский государственный университет
имени Александра Григорьевича и Николая Григорьевича Столетовых
Изд-во ВлГУ
rio.vlgu@yandex.ru

Педагогический институт
кафедра физико-математического образования и информационных технологий
yakubovich.studylib@mail.ru
eropova13061962@mail.ru