Владимирский государственный университет

А. А. ШАМЫШЕВ О. Н. ШАМЫШЕВА

ПРОГРАММИРОВАНИЕ НА PYTHON

РЕШЕНИЕ ЗАДАЧ ПО ОСНОВАМ АЛГОРИТМИЗАЦИИ

Практикум

Владимир 2025

Министерство науки и высшего образования Российской Федерации

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

А. А. ШАМЫШЕВ О. Н. ШАМЫШЕВА

ПРОГРАММИРОВАНИЕ НА PYTHON

РЕШЕНИЕ ЗАДАЧ ПО ОСНОВАМ АЛГОРИТМИЗАЦИИ

Практикум

Электронное издание

Владимир 2025

ISBN 978-5-9984-1999-7

УДК 004.43
ББК 32.973

Рецензенты:
Доктор технических наук, профессор
зав. научно-учебной лабораторией облачных и мобильных технологий,
профессор департамента программной инженерии факультета
компьютерных наук Национального исследовательского университета
«Высшая школа экономики»
Д. В. Александров

Кандидат технических наук, доцент
зав. кафедрой вычислительной техники и систем управления
Владимирского государственного университета
имени Александра Григорьевича и Николая Григорьевича Столетовых
К. В. Куликов

Издается по решению редакционно-издательского совета ВлГУ

Шамышев, А. А. ПРОГРАММИРОВАНИЕ НА PYTHON. Решение задач по основам алгоритмизации [Электронный ресурс] : практикум / А. А. Шамышев, О. Н. Шамышева ; Владим. гос. ун-т им. А. Г. и Н. Г. Столетовых. – Владимир : Изд-во ВлГУ, 2025. – 176 с. – ISBN 978-5-9984-1999-7. – Электрон. дан. (2,76 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 теоретические основы по составлению алгоритмов и написанию кода программ, разбор решенных задач на языке Python и примеры оформления лабораторных работ.

Предназначено для студентов начальных курсов вузов, обучающихся по техническим специальностям, а также студентов СПО.

Рекомендовано для формирования профессиональных компетенций в соответствии с ФГОС ВО.

Ил. 68. Библиогр.: 5 назв.

ISBN 978-5-9984-1999-7

ОГЛАВЛЕНИЕ

ВВЕДЕНИЕ 4

Тема 1. ВВОД/ВЫВОД ДАННЫХ 12

Тема 2. СОСТАВЛЕНИЕ АРИФМЕТИЧЕСКИХ ВЫРАЖЕНИЙ..... 28

Тема 3. СОСТАВЛЕНИЕ ЛОГИЧЕСКИХ ВЫРАЖЕНИЙ.
УСЛОВНЫЙ ОПЕРАТОР 52

Тема 4. ВЛОЖЕННЫЕ СТРУКТУРЫ ВЕТВЛЕНИЯ..... 65

Тема 5. ЦИКЛЫ 83

Тема 6. ЦИКЛЫ. ТЕКСТОВЫЕ ЗАДАЧИ..... 113

Тема 7. ГРАФИКА..... 135

Тема 8. СПИСКИ 151

ЗАКЛЮЧЕНИЕ..... 174

БИБЛИОГРАФИЧЕСКИЙ СПИСОК..... 175

ВВЕДЕНИЕ

Процесс создания программного обеспечения насколько увлекателен, настолько и непрост. Поставить задачу, спроектировать алгоритм решения, наконец, составить код программы на одном из языков программирования – всему этому надо учиться.

Практикум содержит восемь тем, освоение которых значительно облегчит учебный процесс.

Предложенные для разбора задачи написаны на языке программирования Python 3. Это один из самых популярных современных языков программирования, который широко применяется в коммерческих и некоммерческих проектах. К тому же он прост, лаконичен и имеет низкий порог вхождения: иногда для несложной программы бывает достаточно составить всего одну строчку кода.

Сегодня насчитывается более девяти тысяч языков программирования, однако на практике используется около двадцати. Изучение каждого последующего языка легче предыдущего. Практикум предлагает типовые конструкции и методы, применимые ко многим языкам программирования, не только к Python 3.

В издание вошли темы, посвященные базовым конструкциям языка и написанию программ начального уровня. Представленные задачи развивают навык работы по составлению кода программ со структурой ветвления и циклами, работы со списками и графической библиотекой Pillow. Подробно объяснены правила составления блок-схем программы, дано множество примеров. Благодаря освоению практикума будущие программисты или специалисты в IT-сфере научатся правильно оформлять код программы.

Прикладные задачи, решаемые с помощью Python

В современной IT-индустрии с помощью языка Python решается большое количество задач.

1. Автоматизация рутинных задач. Python – хороший инструмент для автоматизации рутинных задач. Если необходимо рассортировать файлы по папкам, скопировать на 100 различных серверов готовую конфигурацию, найти в логах нужные строчки, то для этого в Python есть большое количество встроенных функций и методов.

2. Разработка веб-приложений. Python – один из самых популярных языков для веб-разработки. В последнее время доступны десятки различных библиотек и фреймворков, которые позволяют создавать веб-сайты с использованием языка Python – от простейших одностраничных приложений до сложнейших маркетплейсов с посещаемостью 1 000 000 человек в сутки. Основными фреймворками для веб-разработки на Python являются Django, Flask, FastAPI, Tornado, Pyramid.

3. Искусственный интеллект, машинное обучение, анализ данных и научные вычисления. Python – один из основных инструментов для работы с разносторонними данными и создания программ с использованием искусственного интеллекта. Такие библиотеки, как TensorFlow, Keras, PyTorch, Scikit-learn, Pandas, NumPy, работают с Python и позволяют обработать любые данные и получить готовые модели для ответа на практически любой вопрос.

Особенности языка

Python – это интерпретируемый язык высокого уровня для работы в прикладных областях. Как все интерпретируемые языки, Python близок к решаемой задаче и сильно отдален от взаимодействия с аппаратной частью компьютера, а также от ручного управления доступом к ресурсам процессора и выделением памяти операционной системы для процесса выполнения программы. Для того чтобы решить задачу, используют высокоуровневые конструкции языка.

***Плюсы**:*

1. Во время выполнения программы (runtime) интерпретируемые языки программирования имеют возможность свободы действий в управлении объектами. Можно создавать, удалять, переопределять любой объект. Например, создать новую переменную или изменить существующую переменную, использовать имена стандартных функций в качестве идентификатора своей переменной. Если вам нравится название функции max, или min, или sum, можно использовать название этих функций для объявления своих переменных.

2. Нет необходимости в управлении памятью, так как выделение и освобождение памяти происходит автоматически.

3. Программу не нужно компилировать: этап компиляции отсутствует, и программа запускается сразу.

4. Реализованы очень многие низкоуровневые алгоритмы работы с данными, например добавление элемента в список, реверс списка и др.

5. В программах, написанных на языке Python, при объявлении переменной не нужно объявлять тип переменной, так как Python не является языком строгой типизации.

6. Python позволяет очень быстро писать программы, так как пользователь не отвлекается на типы данных, работу с памятью и т. д.

7. Python легко изучить. Некоторые отзываются о нем как о языке, из которого «выкинули все ненужное».

8. Python создан для прикладных областей, он очень популярен в среде непрограммистов.

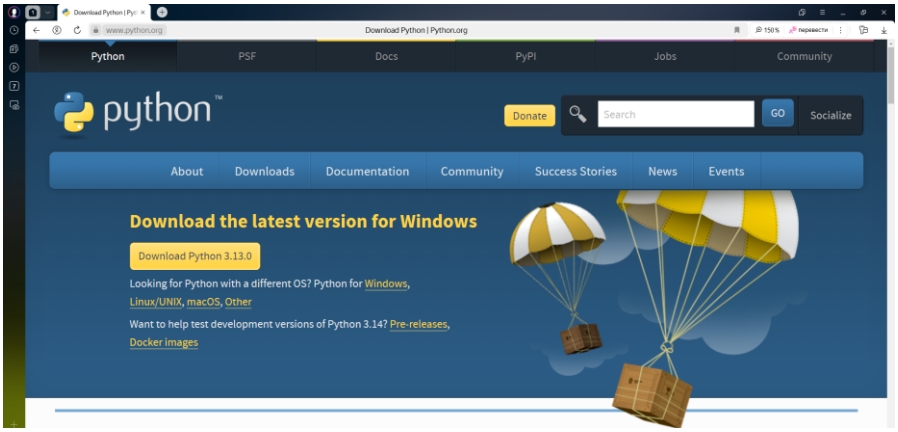
***Минусы**:*

1. Язык Python медленнее компилируемых языков, так как он интерпретируемый.

2. Необходим интерпретатор в памяти, чтобы выполнить любую программу, даже для вычисления такого простого выражения, как «2 + 2».

Среда разработки

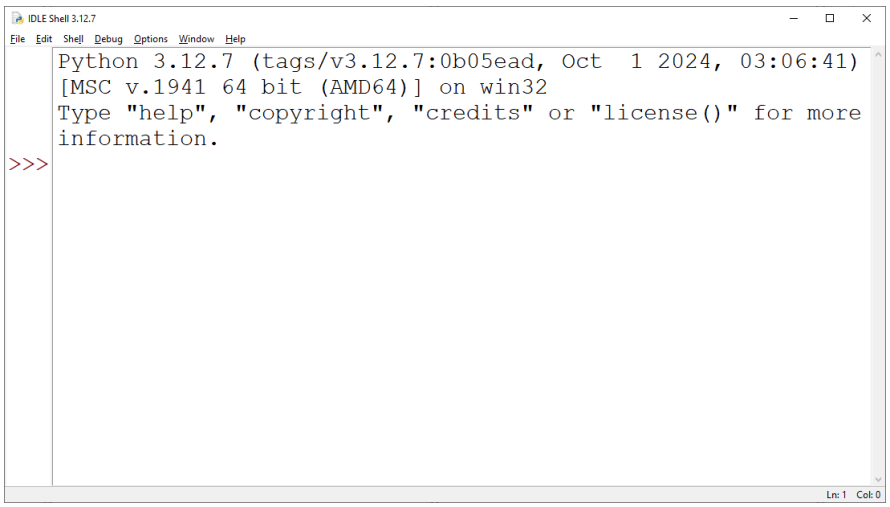
Скачать Python можно с официального сайта по ссылке https://www.python.org/downloads/



Страница загрузки версий Python

У языка Python есть несколько реализаций, написанных на других языках программирования. Реализация Python на C++ называется CPython и является эталонной версией.

В комплекте поставки CPython идет простая среда разработки IDLE. Она написана на Python и графическом фреймворке Tk. Многие относятся к ней скептически из-за ограниченных возможностей и бедного интерфейса, но если речь идет о небольших скриптах и знакомстве с Python, возможно, на первом этапе, когда хочется сосредоточиться на языке, а не на настройках среды, это хороший выбор.



Внешний вид IDLE после запуска

IDLE после старта переходит в командный режим, или, как его называют, REPL (от англ. read-eval-print-loop – «цикл “чтение – вычисление – вывод”»).

IDLE после ввода нескольких команд

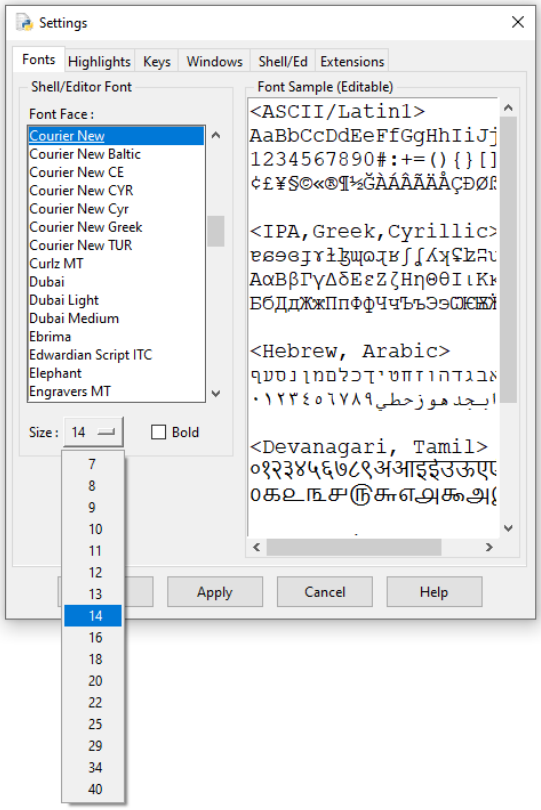
В процессе работы пользователь вводит операторы или блоки операторов по одному, а Python их выполняет, после чего можно переходить к следующей команде. Сама идея работать с небольшими интерактивными кусками кода была улучшена и переосмыслена в интерактивной среде Jupyter Notebook.

Если вы работали в средах типа Wolfram Mathematica или Matlab, то идея REPL должна быть знакома. Кроме того, командные оболочки Linux (например, bash) или Windows (PowerShell или cmd) работают также в режиме REPL.

Отметим несколько самых полезных и используемых возможностей IDLE, а также несколько особенностей:

– чтобы быстро скопировать уже вводившуюся команду, подведите к ней курсор и нажмите Enter;

– в любых средах советуем сделать шрифт побольше (меню Options → Configure IDLE).



Настройки шрифтов

F1 открывает полную коллекцию справочных материалов по CPython. Эти же документы всегда доступны по адресу https://docs.python.org/3/. Не игнорируйте этот материал. У Python одна из самых удобных и полных систем справки, правда, на английском языке.

В любой момент можно выйти из REPL и работать с файлами (меню File). Для того чтобы запустить файл на выполнение, нажмите F5.

Прервать выполнение блока команд можно, нажав CTRL + C.

Сочетания клавиш типа CTRL + C не будут работать в русской раскладке клавиатуры. Копировать выделенный текст можно с помощью всплывающего меню правой кнопкой мыши.

IDLE может «зависать», если объем выведенной в ее консоль информации будет очень большим. Это особенность Tk, так как тест представляет из себя последовательность графических объектов в памяти. Чистый CPython, естественно, не имеет такого недостатка.

Кроме IDLE очень популярны и другие среды разработки, такие как Wing Python IDE, PyCharm, Spyder, Notepad++, Jupyter Notebook, онлайн-редакторы и др. Их анализ останется за рамками практикума. Все скриншоты работающих программ сделаны в среде разработки IDLE.

Переменные, основные типы данных

Основная парадигма программирования в Python – объектно-ориентированное программирование (ООП), поэтому все переменные являются объектами.

Тип данных – это атрибут, определяющий, какого рода данные могут храниться в объекте. В Python определены такие типы данных, как логический тип данных `bool`, целые числа `int`, вещественные числа `float`, строки `str`, списки `list`, кортежи `tuple`, множества `set` и др.

Именами переменных может служить любая последовательность букв любого алфавита. Например, можно использовать русские буквы, греческие буквы или иероглифы.

Большинство операций с объектами осуществляется через методы. Методы можно представлять как функции, находящиеся внутри объекта и имеющие доступ к его содержимому.

То, что выглядит как функция в Python, на самом деле не всегда является функцией, а может быть именем класса, которое написали для вызова конструктора класса. Пока систематически не изучены особенности ООП в Python, будем считать, что это функция.

Создавая переменную, не указывают ее тип. Более того, тип переменной может меняться во время выполнения программы. Иногда это удобно, но лучше не использовать это часто в разработке. Используя функцию `dir()`, можно посмотреть состав любого объекта, например строки. Функция `help()` покажет справку по конкретному методу. Например, будет видно, что метод `swapcase()` меняет регистр у всех букв на противоположный.

Есть некоторое количество операций, которые можно проводить над числами: это классические сложение, умножение, вычитание, деление, возведение в степень. Существует также некоторое количество менее используемых операций. Полный их состав описан в документации по языку программирования, которая доступна на официальном сайте Python по ссылке https://docs.python.org/3/. Использование операции эквивалентно вызову соответствующего служебного метода объекта. Складывать можно любые объекты, у которых определен метод `__add__()`.

Интерфейсно строки и списки очень похожи, потому что являются контейнерами, элементы которых можно перебрать, например в цикле. Только строки содержат символы, а списки – произвольные элементы, в том числе другие списки. Нумерация элементов начинается с нуля. В отличие от списка изменить строку нельзя. Оба этих типа данных поддерживают слайсинг (срезы). Срезы позволяют получить часть строки или списка, начиная с элемента с индексом *a* по элемент с индексом *b* с определенным шагом. При отрицательном шаге элементы выбирают в обратном порядке – это один из способов инвертировать строку.

Списки в Python заменяют по своей функциональности массивы (потому что есть возможность обращаться к конкретным элементам по индексу и менять их). В списках можно произвольно выбирать элементы или добавлять новые. Списки содержат не сами объекты, а ссылки на них, поэтому можно помещать любые элементы в списки, в том числе другие списки с любым уровнем вложенности.

Логический тип данных в Python не отличается от такового в других языках программирования и привычен для тех, кто изучил какой-либо язык. Есть две предопределенные константы, `True` и `False`, которые возникают, например, в операциях сравнения. Есть и другие операции, которые возвращают этот тип. В арифметических операциях эти значения ведут себя как 0 и 1. Значения выражений в сравнениях автоматически конвертируются в логический тип. Поэтому, например, пустой список ведет себя в сравнениях как логическая константа `False`. Базовые типы данных позволяют начать писать содержательные алгоритмы.

Правила составления идентификаторов

Правила составления идентификаторов подробно описаны в «Самоучителе по Python» Дмитрия Мусина (гл. 25 «PEP 8 – руководство по написанию кода на Python», гл. 26 «Документирование кода в Python. PEP 257») [1].

Предлагаем подробно познакомиться с правилами составления идентификаторов в указанной книге. Знание правил позволит вам оформлять код профессионально и чисто, а также выбрать свой стиль из ряда стандартных стилей оформления кода и правил составления идентификаторов.

Тема 1. ВВОД/ВЫВОД ДАННЫХ

Примитивные типы данных в Python

К примитивным типам данных в Python относятся:

– тип `int` – целые числа, которые содержат знак и целую часть числа;

– тип `float` – вещественные числа, которые содержат знак, целую и дробную части числа. Дробная часть числа отделяется от целой части числа точкой;

– тип `str` – строки, которые состоят из символов; заключаются в одинарные или двойные кавычки;

– тип `bool` – логический тип данных, который может принимать одно из двух значений (`True` или `False`).

Инициализация переменных в Python может быть выполнена двумя способами:

1) методом присвоения значения новой переменной.

По типу присваиваемого значения определяется тип новой переменной:

```
a = 18                      # int
b = 18.5                    # float
c = True                    # bool
d = 'eighteen point five'   # str
```

Присвоение значений нескольким переменным возможно в одну строку:

```
a, b, c, d, = 18, 18.5, True, 'eighteen point five'
```

2) методом объявления типа новой переменной.

В этом случае в переменных после объявления будут храниться значения по умолчанию: для `int` – 0, для `float` – 0.0, `str` – пустая строка или `''`, `bool` – значение `False`:

```
a = int()                   # a = 0
b = float()                 # b = 0.0
c = str()                   # c = ''
d = bool()                  # d = False
```

Ввод/вывод строк

Процесс ввода/вывода данных – важный этап решения задач. Методы ввода позволяют получить входную информацию от пользователя программы, методы вывода – печатать сообщения в ходе выполнения кода и получать результат работы программы.

Рассмотрим функцию `input()`, которая читает строку из буфера клавиатуры, и функцию `print()`, которая выводит строки на экран монитора или в консоль.

Ввод строки с клавиатуры:

```
a = input()
```

Функция `input()` может содержать сообщение, которое выводится на экран перед вводом данных:

```
a = input("Введите строку - ")
```

Вывод значения переменной на экран:

```
print(a)
```

Следующий код программы вводит данные с клавиатуры в строковую переменную `a` и выводит значение переменной на экран:

```
a = input("Введите строку - ")
print(a)
```

При вводе строки «Привет!» результатом программы будет:

```
Привет!
```

Ввод/вывод целых чисел

Целое число содержит целую часть и знак числа:

```
a = 5  # положительное целое число
b = -5  # отрицательное целое число
```

Ввод целого числа с клавиатуры. Функция `int()` преобразует строковый тип данных к целочисленному типу данных:

```
a = int(input())
```

Вывод целого числа на экран:

```
print(a)
```

Следующий код программы сохраняет данные с клавиатуры в целочисленную переменную *a* и выводит значение переменной на экран:

```
a = int(input("Введите целое число - "))
print(a)
```

При вводе строки "5" результатом программы будет:

```
5
```

Форматированный вывод целых чисел. Вывод целого числа на экран можно выполнить так, чтобы число занимало на экране определенное количество символов. Форматированный вывод чисел с использованием оператора `%` называют методом подстановки значений в шаблоны строк с помощью плейсхолдеров. Плейсхолдеры – это места в строке, куда будет выполнена подстановка. Для подстановки целых чисел со знаком используется плейсхолдер `%d`.

Пусть необходимо напечатать число на экране в трех позициях. В строке необходимо указывать место для подстановки данных с помощью символа "%", далее указывать необходимое минимальное количество символов на экране "3" и формат выводимого значения "d". Формат "d" – формат вывода целого числа со знаком. После строки выставьте символ подстановки "%" и переменную для вывода:

```
a = 1
print("%3d" % a)
```

```
 1
---
```

Значение 1 выведено в трех позициях на экране, выравнивание выполнено по правому краю занимаемого диапазона.

В случае когда переменная имеет больше разрядов, чем указано в формате вывода, формат вывода игнорируется и на экран будут выведены все разряды числа.

Ввод/вывод вещественных чисел

Вещественное число также называют плавающей точкой (числом с плавающей запятой). Оно содержит знак, целую часть и дробную часть:

```
a = 5.2  # целая часть 5, дробная часть 0.2
```

Ввод вещественного числа с клавиатуры:

```
a = float(input())
```

Функция `float()` преобразует строковый тип данных к вещественному типу данных:

```
a = float(input())
```

Вывод вещественного числа на экран:

```
print(a)
```

Следующий код программы сохраняет данные с клавиатуры в вещественную переменную `a` и выводит значение переменной на экран:

```
a = float(input("Введите вещественное число - "))
print(a)
```

При вводе строки «5.5» результатом программы будет:

```
Введите вещественное число - 5.5
5.5
```

Для вывода вещественного числа на экран так, чтобы дробная часть содержала определенное количество символов, методом подстановки значений используется плейсхолдер `%f`. Вывод вещественного числа с точностью до трех знаков в дробной части выполняется следующим образом:

```
a = 5.5
print("%.3f" % a)
```

```
5.500
```

Рассмотрим пример вывода вещественного числа на экран так, чтобы всё число занимало на экране минимум семь символов (включая точку и три символа дробной части числа), с использованием метода подстановки значений.

```
a = 5.5
print("%7.3f" % a)
 5.500
-------
```

Ввод/вывод нескольких переменных

Ввод нескольких переменных с клавиатуры. Для ввода нескольких переменных с клавиатуры нужно использовать функцию `input()` необходимое количество раз.

Ниже написан код программы, в котором вводятся два числа с клавиатуры и вычисляется среднее значение этих чисел:

```
a = int(input("Введите a: "))
b = int(input("Введите b: "))
print("Среднее значение = %7.3f" % ((a+b)/2) )
```

При вводе $a = 2$ и $b = 3$ получим результат работы программы:

```
Введите a: 2
Введите b: 3
Среднее значение =  2.500
```

Рассмотрим пример ввода двух чисел с клавиатуры в одну строку через пробел с использованием функции `map()`:

```
a,b = map(int, input("Введите a b: ").split() )
print("Среднее значение = %7.3f" % ((a+b)/2) )
```

При вводе $a = 2$ и $b = 3$ получим результат работы программы:

```
Введите a b : 2 3
Среднее значение =  2.500
```

Разберем подробно этот пример. Функция `input()` возвращает строку, введенную с клавиатуры. Функция `split()` делит строку по пробелу и возвращает список строк. Функция `map(int,` итерируемый объект`)` применяет метод `int()` к каждому элементу итерируемого объекта. На выходе первое число – переменной *a*, второе число – переменной *b*. Например, вводят `"2 3"`:

```
input("Введите a b:")                          #"2 3"
input("Введите a b:").split()          #["2", "3"]
map(int,input("Введите a b:").split()) # a,b = 2,3
a, b = 2, 3
```

Результат работы оператора `print("Среднее значение = = %7.3f" % ((a+b)/2) )` выглядит следующим образом:

```
Среднее значение =  2.500
```

Вывод нескольких переменных на экран. При выводе нескольких переменных с помощью метода подстановок в строке указывают необходимое количество мест для подстановки (плейсхолдеров), а значения для подстановок записывают в круглых скобках через запятую после строки и знака `%`:

```
a, b = 1, 3
d = a/b
print(a,"/",b,"=",d)
print("%3d / %3d = %7.3f" % (a,b,d))
```

Результат:

```
1 / 3 = 0.3333333333333333
 1 /  3 =  0.333
```

В функции `print()` можно использовать параметры `sep` и `end`.

Параметр `sep` (от слова separate) содержит разделительные символы между выводимыми переменными функцией `print()`. По умолчанию `sep = " "` (равен пробелу).

Параметр `end` содержит символы, которые будут добавлены после вывода всех переменных функцией `print()`. По умолчанию `end = "\n"`. Символ "\n" означает перенос курсора на следующую строку:

```
a, b = 2, 3
print(a,b)
print(a,b, sep = ' ', end = '\n')
print(a,b, sep = '!!!', end = '\n\n')
```

Функции `print(a,b)` и `print(a,b,sep=' ',end='\n')` дадут одинаковый результат:

```
2 3
```

Функции `print(a,b, sep = '!!!', end = '\n\n!')` даст результат:

```
2!!!3


!
```

Рассмотрим пример вывода вещественного числа на экран так, чтобы всё число занимало на экране минимум *n* символов и *m* символов дробной части числа с использованием метода подстановки значений.

```
a, n, m = 5.5, 7, 3
s = '%'+str(n)+'.'+str(m)+'f'
print( s % a )
```

```
 5.500
-------
```

Работа с логическими переменными

Логические переменные принимают два значения: `True` или `False`. Функция `bool()` преобразует число или строку к логическому типу данных. Значения 0 и "" (пустая строка) дают значение `False`. При всех других значениях параметра функция `bool()` принимает значение `True`.

Примеры для значения `True`:

```
my_bool_true = True
print(my_bool_true)              #True
my_bool_true = bool(10)
print(my_bool_true)              #True
my_bool_true = bool('some')
print(my_bool_true)              #True
```

```
True
True
True
```

Примеры для значения `False`:

```
my_bool_false = False
print(my_bool_false)            #False
my_bool_false = bool(0)
print(my_bool_false)            #False
my_bool_false = bool('')
print(my_bool_false)            #False
my_bool_false = bool()
print(my_bool_false)            #False
```

```
False
False
False
False
```

Ввод логических переменных с клавиатуры:

```
my_bool = bool(input())
```

Работа со строковыми данными

Строки – это последовательности символов любой длины, заключённые в одинарные, двойные или тройные кавычки. Рассмотрим основные операции со строками.

Объявление пустой строки:

```
s = ""
s = ''
```

Конкатенация, или сцепление строк:

```
s1 = input()
s2 = " - это строка."
print(s1+s2)
```

Операция + выполняет конкатенацию строк *s*1 и *s*2.

При вводе с клавиатуры *s*1 = "Набор символов" получим результат:

```
Набор символов - это строка.
```

Обращение к первому символу строки:

```
s1 = "Набор символов"
print(s1[0])
```

```
H
```

Вычисление длины строки:

```
s1 ="Набор символов"
print( len(s1) )
```

```
14
```

Деление строки на несколько строк с использованием функции `split()`. Деление выполняется по определенному символу, называемому сепаратором. По умолчанию сепаратором является пробел:

```
s = "Группа ИР-123"
a,b = s.split()
print(a)
print(b)
a,b = s.split(" ")
print(a)
print(b)
```

Строки кода `a,b = s.split()` и `a,b = s.split(" ")` дают одинаковый результат:

```
Группа
ИР-123
```

Если в функции `split()` указать другой параметр деления строки, то строка будет разделена по указанному символу `'_'`.

```
s = '1_22_333'
a,b,c = s.split('_')
print(a)
print(b)
print(c)
```

```
1
22
333
```

```
x = 'Иванов Иван'
f,i = x.split()
print(f)
print(i)
```

```
Иванов
Иван
```

Вывод строковых переменных на экран:

```
x = 'Иванов Иван'
f,i = x.split()
print('Привет,', i,f,'!')
```

```
Привет, Иван Иванов !
```

Вывод строковых переменных с использованием метода подстановки. Формат вывода строк – использование плейсхолдера `%s`:

```
x = 'Иванов Иван'
f,i = x.split()
print('Привет, %s %s!' %(i,f))
```

```
Привет, Иван Иванов!
```

Преобразование данных в строку с помощью функции `str()`:

```
N,F,B = 0, 1.1, True
print(str(N), str(F), str(B))
```

```
0 1.1 True
```

Вывод на экран осуществляется и без преобразования переменных к типу `str`:

```
N,F,B = 0, 1.1, True
print(N, F, B)
```

```
0 1.1 True
```

Объединение слов в одну строку можно выполнить функцией `join()`. Между словами добавляется символ (или символы), который стоит перед обращением к методу `join`. Объединяемые слова нужно записать как параметры внутри круглых скобок `join ( слова )`. В следующем примере используется объединяющий символ – пробел `' '` между словами:

```
a,b,c = '1','2','3'
s = ' '.join((a,b,c))
print(s)
```

```
1 2 3
```

Обратите внимание, что объединяемые слова заключаются в круглые скобки и образуют кортеж. Также объединяемые слова можно заключить в квадратные скобки (для получения списка):

```
a,b,c = '1','2','3'
s = ' '.join([a,b,c])
print(s)
```

```
1 2 3
```

Собрать строку можно через другой разделительный символ или символы. В следующем примере между словами добавляется символ '///':

```
a,b,c = 1,2,3
s = '///'.join((str(a),str(b),str(c)))
print(s)
```

```
1///2///3
```

Рассмотрим пример вычисления целой и дробной частей вещественного числа с преобразованием к строковому типу:

```
a = input()
b = float(a)
c,d = a.split('.')
c,d = int(c),int(d)
print("Число = %.3f \n\
Целая часть = %d \n\
Дробная часть = 0.%d " % (b,c,d))
```

Результат при вводе $a = -1.234$:

```
Число = -1.234
Целая часть = -1
Дробная часть = 0.234
```

Разберем подробно следующую часть кода:

```
print("Число = %.3f \n\
Целая часть = %d \n\
Дробная часть = 0.%d " % (b,c,d))
```

Здесь в конце первой и второй строк используется символ обратного слэша «\». Символ необходим для написания оператора, который располагается в нескольких строках кода. Добавление символа обратного слэша позволяет продолжить писать текущий оператор в следующей строке.

Использование функции `map()` позволяет сократить код программы для задачи вычисления целой и дробной частей вещественного числа:

```
b = float(input())
c,d = map(int , str(b).split('.'))
print("Число = %.3f \n\
Целая часть = %d \n\
Дробная часть = %d " % (b,c,d))
```

Результат при вводе $b = 1.234$:

```
Число = 1.234
Целая часть = 1
Дробная часть = 0.234
```

Еще один пример написания оператора в нескольких строках:

```
a = 1
a = a +\
2
print(a)
3
```

Составить многострочные строки вывода можно с использованием тройных апострофов `'''`.

Пример написания многострочных строк:

```
a ='''Привет,
Иван
Иванович!'''
print(a)
Привет,
Иван
Иванович!
```

Определение типа (класса) переменной

Тип (класс) переменной можно определить с помощью функции type():

```
a = 18
b = 18.5
c = True
d = 'eighteen point five'
print(type(a))
print(type(b))
print(type(c))
print(type(d))
```

Результат работы программы:

```
<class 'int'>
<class 'float'>
<class 'bool'>
<class 'str'>
```

В Python существует специальный тип объекта, который указывает на отсутствие типа у переменной и отсутствие значения в этой переменной:

```
a = None
print(type(a))
<class 'NoneType'>
```

Пример выполнения лабораторной работы

Цель работы: познакомиться с основными типами данных в Python (целое число, вещественное число, строка, логический тип данных) и явным приведением типов. Научиться вводить данные основных типов с клавиатуры и выводить их на экран.

Задание № 1

1. Объявите пустую строку.

2. Введите с клавиатуры фамилию и имя в строку с приглашением «Введите свои фамилию и имя – ».

3. Выведите на экран приветствие с введенными фамилией и именем: «Привет, _ _!».

4. В одну строку выведите на экран количество символов в фамилии и количество символов в имени.

Решение задачи:

```
#1
null=''
null=""

#2
фамилия_имя = input("Введите свои фамилию и имя ")

#3
print("Привет,",фамилия_имя,"!")

#4
имя, фамилия = фамилия_имя.split()
print(len(имя), len(фамилия))
```

Результаты работы программы задания 1 представлены на рис. 1.1.

```
Введите свои фамилию и имя Иванов Иван
Привет, Иванов Иван !
11
6 4
```

Рис. 1.1. Скриншот результатов работы программы задания 1

Задание № 2

1. Объявите целочисленную переменную и переменную числа с плавающей точкой (вещественную переменную) со значениями, равными 0.

2. Введите значения объявленных переменных с клавиатуры, используя оператор ввода.

3. Выведите на экран класс каждой переменной (под классом понимается тип данных, который в ней хранится).

4. Переменную целочисленного типа приведите к числу с плавающей точкой и обратно.

5. Приведите числовую переменную с плавающей точкой к стоковому типу, выведите на экран ее длину (количество символов), приведите обратно к числовому типу.

6. Выведите целую часть вещественного числа и его дробную часть с точностью до *n* знаков после запятой (*n* вводит пользователь с клавиатуры).

Решение задачи:

```
#1
counter = 0
miles = 0.0

#2
counter = int(input("Введите целое число: "))
miles = float(input("Введите вещественное \
число: "))

#3
print("counter Тип данных: ",type(counter))
print("miles Тип данных: ",type(miles))

#4
counter = float(counter)
print("counter Тип данных: ",type(counter),\
counter)
counter = int(counter)
print("counter Тип данных: ",type(counter),\
counter)

#5
miles = str(miles)
print("miles Длина:",len(miles),\
   " ,miles Тип данных: ",type(miles))
miles=float(miles)
```

```
#6
n=int(input("Введите n: "))
s = "%."+str(n)+"f"
print( s % miles)
```

Результаты работы программы задания 2 представлены на рис. 1.2.

```
Введите целое число: 10
Введите вещественное число: 10.5
counter Тип данных:  <class 'int'>
miles Тип данных:  <class 'float'>
counter Тип данных:  <class 'float'> 10.0
counter Тип данных:  <class 'int'> 10
miles Длина: 4  ,miles Тип данных:  <class 'str'>
Введите n: 3
10.500
```

Рис. 1.2. Скриншот результатов работы программы задания 2

Задание № 3

1. Объявите логическую переменную.
2. Введите значение логической переменной с клавиатуры.
3. Выведите значение логической переменной на экран.
4. Приведите каждое значение логической переменной (`True` и `False`) к строковому и числовому типам данных. Выведите результат на экран.

Решение задачи:

```
#1
b= True

#2
b = bool(input("Введите:"))
print(b)

#3
print(type(b))
b_s = str(b)
b_n = int(b)
print(b_s)
print(b_n)
```

Результаты работы программы задания 3 представлены на рис. 1.3.

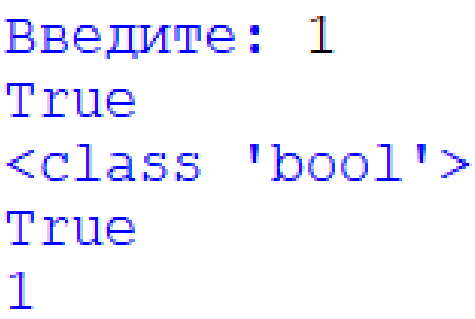


Рис. 1.3. Скриншот результатов работы программы задания 3

Вывод: Я познакомился с основными типами данных в Python (целое число, вещественное число, строка, логический тип данных) и явным приведением типов. Я научился вводить данные основных типов с клавиатуры и выводить их на экран.

Тема 2. СОСТАВЛЕНИЕ АРИФМЕТИЧЕСКИХ ВЫРАЖЕНИЙ

Работа с числовыми данными

Значения *a* и *b* вводят через пробел в одну строку:

```
a,b = map(int, input("Введите a b: ").split())
```
```
Введите a b: 2 3
```

Сложение:

```
print(a,"+",b,"=", a+b)
```
```
2 + 3 = 5
```

Вычитание:

```
print(a,"-",b,"=", a-b)
```
```
2 - 3 = -1
```

Умножение:

```
print(a,"*",b,"=", a*b)
```
```
2 * 3 = 6
```

Возведение в степень:

```
print(a,"**",b,"=", a**b)
2 ** 3 = 8
```

Деление:

```
d = a/b
print(a,"/",b,"=",d)
print("%3d /%3d = %7.3f" % (a,b,d))
2 / 3 = 0.6666666666666666
 2 / 3 =   0.667
```

Целочисленное деление:

```
print(a,"//",b,"=", a//b)
2 // 3 = 0

print("5 // 3 =", 5//3)
5 // 3 = 1
```

Остаток от деления:

```
print(a,"%",b,"=", a%b)
2 % 3 = 2

print("5 % 3 =", 5%3)
5 % 3 = 2
```

Округление до трех знаков после запятой:

```
print(a,"/",b,"=", round(a/b,3))
2 / 3 = 0.667
```

Модуль числа *x*:

```
x = 1.2
print("|",x,"| = ", abs(x) )
| 1.2 | = 1.2

x = -1.2
print("|",x,"| = ", abs(x) )
| -1.2 | = 1.2
```

Модуль math

Модуль `math` содержит функции работы с числами. Подключение модуля `math` необходимо выполнить до обращения к его функциям.

Подключение модуля `math`:

```
import math
```

Обращение к константам и функциям `math` выполняется через точку:

```
import math
print('пи = %.3f  e = %.3f ' % (math.pi, math.e))
```

```
пи = 3.142  e = 2.718
```

Вычисление sin(30°):

```
import math
print( math.sin( 30 * math.pi / 180))
print( "%7.3f" % math.sin( math.radians(30)) )
```

Функция `radians()` переводит градусы в радианы, функция `degrees()` – радианы в градусы.

```
0.49999999999999994
 0.500
```

Косинус X, синус X, тангенс X. Параметр функций `cos()`, `sin()`, `tan()` указывается в радианах:

```
import math
X = math.radians(int(input()))
print(round(math.cos(X),3))
print(round(math.sin(X),3))
print(round(math.tan(X),3))
```

Результат работы программы при вводе $X = 30°$:

```
30
0.866
0.5
0.577
```

Арккосинус X, арксинус X, арктангенс X. Результат функций вычисляется в радианах:

```
import math
X = math.radians(int(input()))
print(round(X,3))
f1 = math.cos(X)
f2 = math.sin(X)
f3 = math.tan(X)
print(round(math.acos(f1),3))
print(round(math.asin(f2),3))
print(round(math.atan(f3),3))
```

Результат работы программы при вводе $X = 30°$:

```
30
0.524
0.524
0.524
0.524
```

При необходимости результат можно преобразовать в градусы:

```
import math
X = math.radians(int(input()))
print(round(X,3))
f1 = math.cos(X)
f2 = math.sin(X)
f3 = math.tan(X)
print(round(math.degrees(math.acos(f1)),3))
print(round(math.degrees(math.asin(f2)),3))
print(round(math.degrees(math.atan(f3)),3))
```

Результат работы программы при вводе $X = 30°$:

```
30
0.524
30.0
30.0
30.0
```

Арктангенс *Y/X* (в радианах) с учетом четверти, в которой находится точка (*X, Y*):

```
import math
X,Y = map(float, input().split())
print(round(math.atan2(Y, X),3))
print(round(math.degrees(math.atan2(Y, X)),3))
```

Результат работы программы при вводе *X* = 1.5, *Y* = –2.0 рад:

```
1.5 -2.0
-0.927
-53.13
```

Функция `ceil()` округляет число в бóльшую сторону:

```
import math
print("ceil(1)  =", math.ceil(1))     # 1
print("ceil(1.1) =", math.ceil(1.1))    # 2
print("ceil(1.9) =", math.ceil(1.9))  # 2
print()
print("ceil(-1)  =", math.ceil(-1))   # -1
print("ceil(-1.1) =", math.ceil(-1.1))  # -1
print("ceil(-1.9) =", math.ceil(-1.9))  # -1
```

```
ceil(1)  = 1
ceil(1.1) = 2
ceil(1.9) = 2
ceil(-1)  = -1
ceil(-1.1) = -1
ceil(-1.9) = -1
```

Функция `floor()` округляет число в меньшую сторону:

```
import math
print("floor(1)  =", math.floor (1))   # 1
print("floor(1.1) =", math.floor (1.1))  # 1
print("floor(1.9) =", math.floor (1.9))  # 1
print()
print("floor(-1)  =", math.floor (-1))  # -1
print("floor(-1.1) =", math.floor (-1.1)) # -2
print("floor(-1.9) =", math.floor (-1.9)) # -2
```

```
floor(1)  = 1
floor(1.1) = 1
floor(1.9) = 1
floor(-1)  = -1
floor(-1.1) = -2
floor(-1.9) = -2
```

Функция `trunc()` усекает значение *x* до целого, функция `math.trunc(x)` преобразует вещественное число к целому:

```
import math
x = float(input())          # 12.90
y = math.trunc(x)
print(x,y)                  # 12.9 12
print(type(x),type(y))
                    # <class 'float'> <class 'int'>
```

Результат работы программы при вводе значения 12.90:

```
12.90
12.9 12
<class 'float'> <class 'int'>
```

Выделение целой части числа с помощью модуля `math`:

```
import math
a = float(input())
print( math.trunc(a) )
```

Результат при вводе $a = 1.234$:

```
1
```

Результат при вводе $a = -1.234$:

```
-1
```

Вычисление дробной части числа:

```
import math
a = float(input())
b = a - math.trunc(a)
print(b)
print(round(b,3))
```

Результат при вводе $a = 1.234$:

```
0.2339999999999999
0.234
```

Результат при вводе $a = -1.234$:

```
-0.2339999999999999
-0.234
```

Вычисление абсолютного значения дробной части числа:

```
import math
a = float(input())
b = abs(a - math.trunc(a))
print(b)
print(round(b,3))
```

Результат при вводе $a = 1.234$:

```
0.2339999999999999
0.234
```

Результат при вводе $a = -1.234$:

```
0.2339999999999999
0.234
```

Функция `exp(x)` вычисляет e^x:

```
import math
x = float(input())
print("e^%.3f = %.3f" % (x,math.exp(x)))
```

Результат при вводе $x = 1$:

```
1
e^1.000 = 2.718
```

Функция `log(x,base)` вычисляет логарифм x по основанию `base`:

```
import math
x = float(input("Введите число: "))
base = float(input("Введите основание: "))
print("log( %.3f ) по основанию %.3f = %.3f " \
%(x,base, math.log(x,base)))
```

При вводе $x = 2$, `base` = 0.5 получим следующий результат:

```
Введите число: 2
Введите основание: 0.5
log( 2.000 ) по основанию 0.500 = -1.000
```

Если `base` не указан, вычисляется натуральный логарифм:

```
import math
x = float(input("Введите число: "))
print("log( %.3f ) по основанию %.3f = %.3f " \
%(x,math.e, math.log(x)))
```

При вводе $x = 2.718$ получим следующий результат:

```
Введите число: 2.718
log( 2.718 ) по основанию 2.718 = 1.000
```

Вещественные числа могут быть записаны с помощью экспоненциальной формы записи числа. Число $0.01 = 10^{-2}$ записывается в экспоненциальной форме так: `1E-2` или `1e-2`.

```
print(1 == 1E-0)
True

print(0.5 == 5E-1)
True

print(0.00007 == 7E-5)
True

print(400 == 4E+2)
True
```

Функция `isclose(x, y, rel_tol = 1e-9, abs_tol = 0.0)` сравнивает вещественные числа на равенство. Функция содержит именованные параметры относительной `rel_tol` и абсолютной `abs_tol` погрешностей сравнения. Параметры погрешностей можно не указывать при обращении к методу, так как они имеют значения по умолчанию `rel_tol = 1e-9, abs_tol = 0.0`.

Абсолютная погрешность $abs\_tol \geq 0$ вычисляется по формуле

$$abs\_tol = |x - y| \quad (1)$$

Относительная погрешность $rel_{tol} \geq 0$ вычисляется по формуле

$$rel\_tol = \frac{|x-y|}{\max(|a|,|b|)} \quad (2)$$

Сравнение вещественного числа на равенство 0:

```
import math
x = float(input("Введите число: "))
print(math.isclose(x,0))
```

При вводе x = 0.000000000000000000001 получим результат:

```
Введите число: 0.000000000000000000001
False
```

При вводе в экспоненциальной форме `1E-20` получим:

```
Введите число: 1E-20
False
```

Проверка вещественного числа на равенство 0 с абсолютной погрешностью пять знаков после запятой, `abs_tol = 0.00001`:

```
import math
x = float(input("Введите число: "))
print(math.isclose(x,0, abs_tol = 0.00001))
```

При вводе в экспоненциальной форме `1E-20` получим результат:

```
Введите число: 1E-20
True
```

Сравнение двух вещественных чисел на равенство с абсолютной погрешностью `1e-5`:

```
import math
x = float(input("Введите число: "))
y = float(input("Введите число: "))
print(math.isclose(x,y, abs_tol = 1e-5))
```

При вводе чисел 5.00001 и 5.00000 получим результат `True` (так как |5.00001 – 5.00000| `<= abs_tol`:

```
Введите число: 5.00001
Введите число: 5.00000
True
```

При вводе чисел 5.00002 и 5.00000 получим результат `False` (так как |5.00002 – 5.00000| `> abs_tol`:

```
Введите число: 5.00002
Введите число: 5.00000
False
```

Модуль sys

Модуль `sys` обеспечивает доступ к некоторым переменным и функциям, взаимодействующим с интерпретатором Python.

Присвоение максимального значения целочисленной переменной выполняется с помощью константы `maxsize` из модуля `sys`:

```
import sys
x = sys.maxsize
print(x)
9223372036854775807
```

Присвоение минимального значения целочисленной переменной:

```
import sys
x = -sys.maxsize-1
print(x)
-9223372036854775808
```

Использовать `sys.maxsize` можно в тех случаях, когда необходимо установить порог значения для вычисления или перебора целочисленной переменной.

Следует помнить, что в Python возможны целочисленные значения больше, чем `sys.maxsize`, и меньше, чем `(-sys.maxsize-1)`.

Например, вычислим факториал 50!

```
import math
math.factorial(50)
30414093201713378043612608166064768844377641568960
512000000000000
```

Как видно из примера, результат 50! гораздо больше, чем значение sys.maxsyze. Ограничения размеров целых чисел как сверху, максимальным значением, так и снизу, минимальным значением, в языке Python не существует. Ограничение размеров целых чисел зависит от возможности устройства.

Информацию о типе int можно посмотреть в sys.int_info:

```
print(sys.int_info)
sys.int_info(bits_per_digit=30,sizeof_digit=4, de-
fault_max_str_di its=4300, str_dig-
its_check_threshold=640)
```

Информацию о типе float можно посмотреть в sys.float_info:

```
print(sys.float_info)
sys.float_info(max=1.7976931348623157e+308,
max_exp=1024, max_10_exp=308,
min=2.2250738585072014e-308, min_exp=-1021,
min_10_exp=-307, dig=15, mant_dig=53, epsi-
lon=2.220446049250313e-16, radix=2, rounds=1)
```

Минимальное вещественное значение:

```
print(sys.float_info.min)
2.2250738585072014e-308
```

Максимальное вещественное значение:

```
print(sys.float_info.max)
1.7976931348623157e+308
```

В Python также есть константа, обозначающая положительную бесконечность inf. Отрицательная бесконечность обозначается как inf:

```
print( float("inf"), float("-inf") )
inf -inf
```

Если параметр, переданный в функцию `float()`, представляет собой целое число, которое превышает максимальное значение чисел с плавающей точкой, то также возвращается значение `inf`:

```
print(float(1e+309))
inf
```

Модуль random

Модуль `random` содержит библиотеку функций для генерации случайных чисел.

Генерирование случайного целого числа в диапазоне от *x* до *y*:

```
import random
x,y = 0,100
z = random.randint(x,y)
print(z)
16
```

Генерирование случайного вещественного числа в диапазоне от 0 до 1:

```
import random
z = random.random()
print(z)
0.81916721945670 18
```

Генерирование случайного вещественного числа в диапазоне от *x* до *y*:

```
import random
x,y = 0,100
z = random.uniform(x,y)
print(z)
76.77397398038725
```

Приоритеты операций

Ниже перечислены операции по степени уменьшения приоритета по строкам. Операции, записанные в одной строке, имеют равный приоритет.

Операторы	*Описание*
`()`	Скобки
`**`	Возведение в степень
`+x, -x, ~x`	Унарный плюс/минус, битовое отрицание
`*, /, //, %`	Умножение, деление, целочисленное деление, остаток от деления
`+, -`	Сложение и вычитание
`<<, >>`	Битовые сдвиги
`&`	Битовое И
`^`	Битовое исключающее ИЛИ (`XOR`)
`|`	Битовое ИЛИ
`==, !=, >, >=, <, <=`	Сравнение
`is, is not`	Проверка идентичности
`in, not in`	Проверка вхождения
`not`	Логическое `not`
`and`	Логическое И
`or`	Логическое ИЛИ

Решение задач с целыми числами

Задача 1. Дано натуральное четырехзначное число *N*. Найдите все разряды числа *N*.

Решение задачи:

```
N = int(input("Введите N: "))

# выполняем операции: %,//
last =  N % 10      # разряд единиц
last_1 = N % 100 // 10   # разряд десятков
last_2 = N % 1000 // 100  # разряд сотен
last_3 = N // 1000     # разряд тысяч
print(last_3, last_2, last_1, last)
# или
last =  N % 10  // 1  # разряд единиц
last_1 = N % 100  // 10  # разряд десятков
last_2 = N % 1000 // 100 # разряд сотен
last_3 = N % 10000 // 1000 # разряд тысяч
print(last_3, last_2, last_1, last)
```

Результат работы программы:

```
Введите N: 1234
1 2 3 4
```

```
# выполняем операции: //,%
last =  N % 10           # разряд единиц
last_1 = N // 10  % 10   # разряд десятков
last_2 = N // 100 % 10   # разряд сотен
last_3 = N // 1000       # разряд тысяч
print(last_3, last_2, last_1, last)

# или
last =  N // 1  % 10 # разряд единиц
last_1 = N // 10  % 10   # разряд десятков
last_2 = N // 100 % 10   # разряд сотен
last_3 = N // 1000 % 10  # разряд тысяч
print(last_3, last_2, last_1, last)
```

Результат работы программы:

```
Введите N: 1234
1 2 3 4
```

Задача 2. Дано натуральное число N ($9 < N \leq 9999$). Поменяйте местами последнюю и предпоследнюю цифры этого числа.

Решение задачи:

```
N = int(input("Введите N: "))
last = N % 10
last_1 = N % 100 // 10
N -= N % 100
N += last*10 + last_1
print(N)
```

Результат работы программы:

```
Введите N: 1234
1243
```

Задача 3. Дано натуральное число N ($9 < N \leq 9999$). Поменяйте местами первую и вторую цифры этого числа.

Решение задачи:

```
N = int(input("Введите N: "))
# преобразуем строку в список символов
s = list(str(N))
s[0],s[1] = s[1],s[0]
N = int(''.join(s))
print(N)
```

Результат работы программы:

```
Введите N: 12345678
21345678
```

Задача 4. Функция `sum()` вычисляет сумму элементов последовательности. Последовательностью может быть список, кортеж или другой итерируемый объект.

Вычислите сумму чисел 1, 2, 3, 4 с использованием функции `sum()`.

Решение задачи:

```
a, b, c = 1, 2.3, 4
print(sum( (a, b, c) ))
```

Результат работы программы:

```
7.3
```

Задача 5. Дано натуральное число N. Вычислите сумму цифр числа с помощью метода `sum()`.

Решение задачи:

```
N = int(input("Введите N: "))
# преобразуем строку в список символов
s = list(str(N))
# преобразуем список символов в список целых чисел
d = map(int, s)
print( sum(d) )
```

```
# или
N = int(input("Введите N: "))
print(sum(map(int, list(str(N)))))
```

Результат работы программы:

```
Введите N: 1234
10
```

Задача 6. Дано натуральное число *N* с четным количеством цифр. Проверьте, равна ли сумма цифр первой половины числа *N* сумме цифр второй половины числа *N*.

Решение задачи:

```
import math
N = int(input("Введите N: "))
k = len(str(N))   # количество цифр в числе
c = 10**(math.ceil(k/2))
N1 = N % c      # правая половина числа
N2 = N // c      # левая половина числа
# Сумма цифр в половинах числа
sum_N1 = sum(map(int, list(str(N1))))
sum_N2 = sum(map(int, list(str(N2))))
print(sum_N1, sum_N2)
print(sum_N1 == sum_N2)
```

Перечислим переменные к задаче 6: *N* – число с четным количеством разрядов; *k* – количество цифр в числе; *N*1 – правая половина числа; *N*2 – левая половина числа; `sum_N1` и `sum_N2` – суммы цифр в соответствующих половинах числа.

Переменная *c* – число-маска, с помощью которого можно выделить левую и правую части числа. Переменная *c* – это число вида 10…0 (количество 0 равно количеству разрядов в правой половине числа).

Программа в первой строке выводит суммы частей числа. Программа во второй строке выводит `True`, если суммы частей числа равны, и `False`, если суммы не равны.

Запишем код программы задачи 6, исключив вспомогательные переменные:

```
import math
N = int(input("Введите N: "))
c = 10**( math.ceil(len(str(N)) / 2))
sum_N1 = sum(map(int, list(str(N % c))))
sum_N2 = sum(map(int, list(str(N // c))))
print(sum_N1, sum_N2)
print(sum_N1 == sum_N2)
```

Результат работы программы:

```
Введите N: 12340532
10 10
True
```

```
Введите N: 12340123
6 10
False
```

Решение задач с вещественными числами

Задача 7. Вычислите значение гипотенузы по двум катетам в прямоугольном треугольнике.

Решение задачи:

```
a,b = float(input("a = ")),float(input("b = "))
c = (a**2 + b**2) ** 0.5
print('c = %.3f ' % c)
```

Результат работы программы:

```
a = 4
b = 3
c = 5.000
```

Задача 8. Составьте арифметическое выражение и вычислите значение *y = f(x)* с точностью до трех знаков после запятой:

$$y = \cos(x) + \frac{\sqrt{|x|}}{x^2+1} - \sin(\frac{2}{3}\pi).$$

Если арифметическое выражение не может быть расположено в одной строке, то перенесите выражение с помощью символа "\".

Решение задачи:

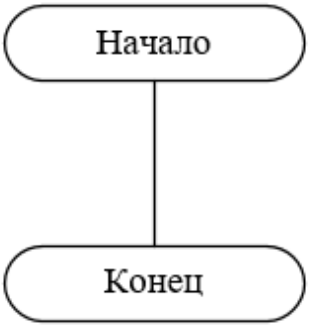
```
import math
x = float(input())
y = math.cos(x) + abs(x)**(1/2)/(x**2 + 1) \
- math.sin(2*math.pi/3)
print(round(y,3))
```

Результат работы программы:

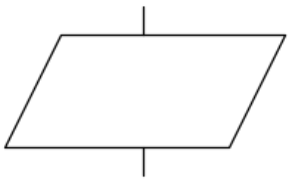
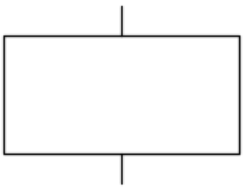
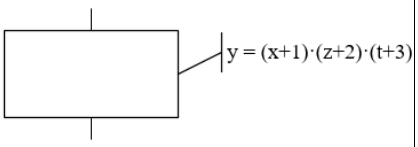
```
Введите x: 1.5
y = -0.418
```

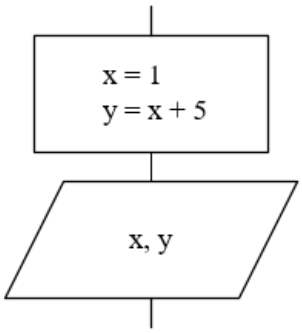
Составление блок-схем программ линейной структуры

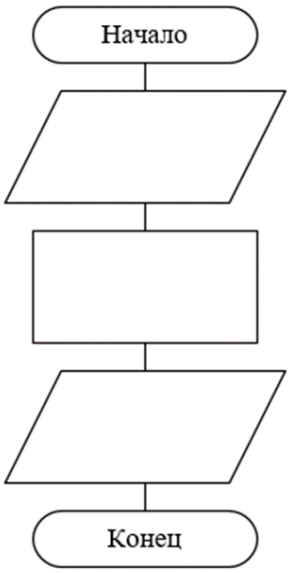
Рассмотрим правила оформления и графического изображения элементов блок-схем для программ линейной структуры.

Правило оформления	Графическое изображение
Блок-схема начинается с блока `begin` и заканчивается блоком `end`. Блок `begin` имеет только один выход. Является точкой входа в программу. Блок `end` имеет один вход. Является точкой выхода из программы. *Масштаб блока* высота : ширина = 1:4	

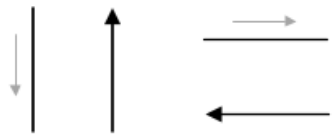
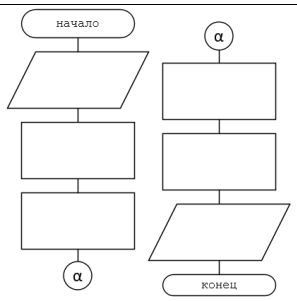
Продолжение

Правило оформления	Графическое изображение
Блоки ввода/вывода – параллелограммы. Блок ввода/вывода (блок обращения к внешним устройствам) имеет один вход и один выход. *Масштаб блока* высота : ширина = 1:2	
Блок вычислений (блок процесса) используется для отображения операций вычисления. Имеет один вход и один выход. *Масштаб блока* высота : ширина = 1:2	
Текст должен быть записан внутри блоков и не выходить за их границу. При необходимости можно использовать выноски и оформлять нужное количество текста вне блока внутри выноски.	

Если в коде программы идут подряд одинаковые операторы, то их можно объединять в один блок на блок-схеме.
 	
`x = 1`
`y = x + 5`
`print(x)`
`print(y)`
	

Все блоки блок-схемы должны иметь одинаковую ширину. Линии не должны пересекаться. Линии всегда входят в верхнюю границу блока. Входящие линии нельзя соединять с нижней или боковыми границами блока. Выходящие линии всегда направлены из нижней границы блока. Также выходящие линии нельзя соединять с верхней или боковыми границами блока.	

Окончание

Правило оформления	Графическое изображение
По правилам соединяющие линии, направленные вниз и вправо, могут не заканчиваться начертанием стрелки. Линии, направленные влево и вверх, должны заканчиваться начертанием стрелки.	
Достаточно часто возникает ситуация, когда алгоритм необходимо перенести на новую страницу. В этом случае следует использовать разделитель – небольшую окружность с меткой. Одинаковые метки ставят в два соответствующих разделителя, которые соединяют части блок-схемы.	

Примеры составления блок-схем программ линейной структуры

Задача 9. Вычислите квадратный корень числа *x.* Число *x* введите с клавиатуры. Результат выведите на экран.

Решение задачи:

```
x = int(input('x = '))
y = x**0.5
print(y)
```

Блок-схема программы задачи 9 представлена на рис. 2.1.

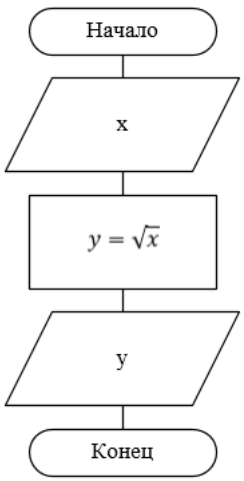


Рис. 2.1. Блок-схема программы задачи 9

Задача 10. Дано натуральное число N ($1000 < N \leq 9999$). Выделите все разряды данного числа.

Решение задачи:

```
N = int(input())
last =  N % 10
last_1 = N % 100 // 10
last_2 = N % 1000 // 100
last_3 = N // 1000
print(last_3, end = ' ')
print(last_2, end = ' ')
print(last_1, end = ' ')
print(last,  end = ' ')
```

Блок-схема программы задачи 10 представлена на рис. 2.2.

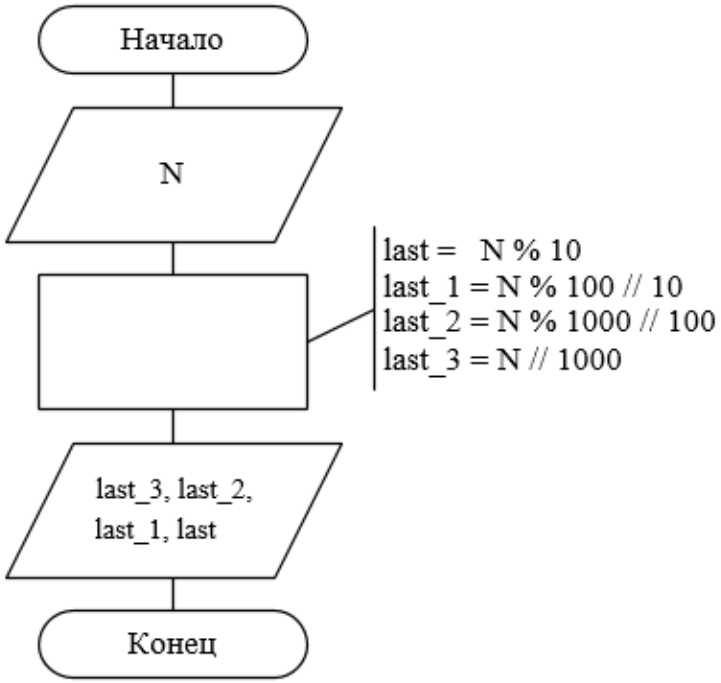


Рис. 2.2. Блок-схема программы задачи 10

Пример выполнения лабораторной работы

Цель работы: познакомиться с арифметическими функциями Python. Научиться составлять арифметические выражения и программы линейной структуры, а также блок-схемы к программам линейной структуры.

Задание № 1

Даны два натуральных числа M и N ($1000 \le M \le 9999$, $1000 < N \le 9999$). Составьте новое число F, состоящее из двух крайних разрядов числа M и двух средних разрядов числа N.

Пример: $M = 1234$, $N = 5678$, $F = 1674$.

Решение задачи:

```
"""
ЛР № 2. Задание 1. Вариант 1.
Программа вычисления арифметического выражения.

Программа составляет число F из двух крайних разря-
дов числа M и двух средних разрядов числа N.

студент гр. ПРИ-124 И.И. Иванов
8.09.2024
"""

M = int(input("M = "))
N = int(input("N = "))
m1 = M // 1000
m4 = M % 10
n2 = N // 100 % 10
n3 = N // 10 % 10
F = m1*1000 + n2*100 + n3*10 + m4
print(F)
```

Результаты работы программы задания 1 представлены на рис. 2.3.

```
M = int(input("M = "))
N = int(input("N = "))
m1 = M // 1000
m4 = M % 10
n2 = N // 100 % 10
n3 = N // 10 % 10
F = m1*1000 + n2*100 + n3*10 + m4
print(F)
```

```
M = 1234
N = 5678
1674
```

Рис. 2.3. Скриншот кода и результатов работы программы задания 1

Задание № 1*

Даны два натуральных числа M и N ($10 \le M$, $1000 \le N$, N имеет четное количество разрядов). Составьте новое число F, состоящее из двух крайних разрядов числа M и двух средних разрядов числа N.

Пример: $M = 12$, $N = 567890$. $F = 1782$.

Решение задачи:

```
import math
M = int(input("M = "))
N = int(input("N = "))
m1 = int(str(M)[0])
m4 = M % 10
c = 10**(math.ceil(len(str(N)) / 2))
n2 = N // c % 10
n3 = int(str(N % c)[0])
F = m1*1000 + n2*100 + n3*10 + m4
print(F)
```

Результаты работы программы задания 1* представлены на рис. 2.4.

```
import math
M = int(input("M = "))
N = int(input("N = "))
m1 = int(str(M)[0])
m4 = M % 10
c = 10**( math.ceil(len(str(N)) / 2))
n2 = N // c % 10
n3 = int(str(N % c)[0])
F = m1*1000 + n2*100 + n3*10 + m4
print(F)
```

```
M = 12
N = 567890
1782
```

*Рис. 2.4. Скриншот кода и результатов работы программы задания 1**

* Задание повышенной сложности.

Задание № 2

Запишите по правилам языка программирования заданную формулу. Вычислите значение выражения по формуле (все переменные принимают действительные значения):

$y = \sqrt[5]{x^2} - \sqrt[3]{\pi^2 + x^2} + \cos(x) + 1/e.$

Составьте блок-схему программы.

Решение задачи:

```
"""
ЛР № 2. Задание 2. Вариант 1.
Программа вычисления арифметического выражения.

Программа вычисляет значение y по формуле
y = x**(2/5) - (math.pi**2+x**2)**(1/3) + \math.cos(x)
+ 1/math.e

студент гр. ПРИ-124 И.И. Иванов
8.09.2024
"""

import math
x = float(input("Введите x = "))
y = x**(2/5)- (math.pi**2+x**2)**(1/3) \
  + math.cos(x)+1/math.e
print("y = ",round(y,3))
```

Результаты работы программы задания 2 представлены на рис. 2.5, блок-схема работы программы – на рис. 2.6.

```
import math
x = float(input("Введите x = "))
y = x**(2/5)- (math.pi**2+x**2)**(1/3) \
    + math.cos(x)+1/math.e
print("y = ",round(y,3))
```

```
Введите x = 1
y =  -0.307
```

Рис. 2.5. Скриншот кода и результатов работы программы задания 2

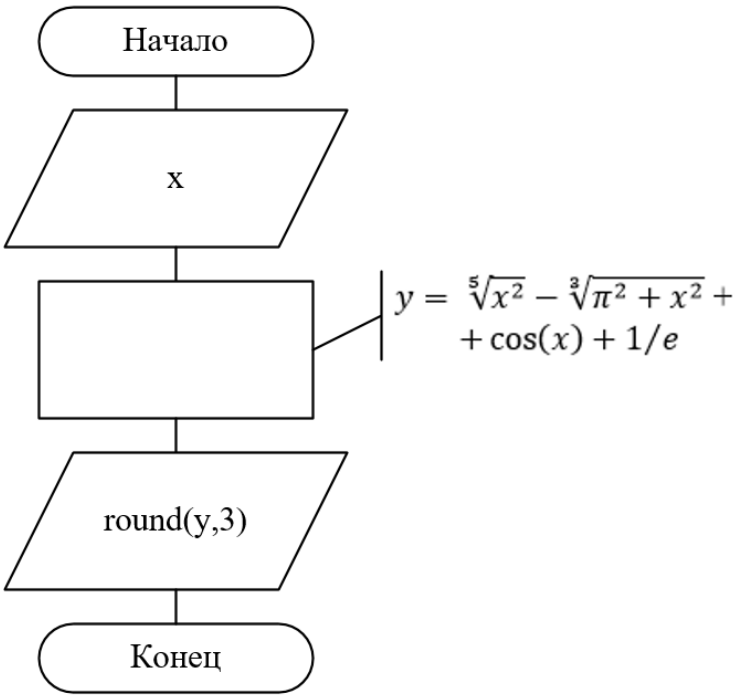


Рис. 2.6. Блок-схема программы задания 2

Вывод: Я познакомился с арифметическими функциями Python. Научился составлять арифметические выражения, программы линейной структуры и блок-схемы к программам линейной структуры.

Тема 3. СОСТАВЛЕНИЕ ЛОГИЧЕСКИХ ВЫРАЖЕНИЙ. УСЛОВНЫЙ ОПЕРАТОР

Логические переменные, операции и выражения

Логические переменные принимают два значения – `True` или `False`:

```
a = True
b = False
```

Операция сравнения дает результат логического типа:

```
a = 5>3
print(a) # True
a = 5<3
print(a) # False
```

Операция сравнения переменной с двумя значениями:

```
a = 5
print(0 < a < 10)    # True
```

Проверка на равенство:

```
print(1 == '1')    # False
print(1 == 1.0)        # True
```

Проверка на неравенство:

```
print(1 != '1')        # True
print(1 != 1.0)        # False
```

Логические операции дают результат логического типа.

Операция `and` – логическое И (конъюнкция):

```
print(False and False)    # False
print(False and True)    # False
print(True and False)    # False
print(True and True)    # True
```

Операция `or` – логическое ИЛИ (дизъюнкция):

```
print(False or False)    # False
print(False or True)    # True
print(True or False)    # True
print(True or True)    # True
```

Операция `xor` – исключающее ИЛИ (сложение по модулю 2). В Python нет оператора `xor` для логических переменных, поэтому `xor` необходимо записывать следующим образом:

```
print(False != False)    # False
print(False != True)    # True
print(True != False)    # True
print(True != True)    # False
```

Операция «Эквивалентность»:

```
print(False == False)    # True
print(False == True)    # False
print(True == False)    # False
print(True == True)    # True
```

Операция «Следование» (импликация):

```
print(not(False) or False) # True
print(not(False) or True)  # True
print(not(True) or False) # False
print(not(True) or True)  # True
```

Логические выражения с двумя переменными:

```
a,b = False,False
print(a and b or True)    # True
print(a or b and (1 < 2))  # False
print(a != b or a == b)   # True
```

Логические выражения с тремя переменными:

```
a,b,c = False,False,True
print(a and b and c)     # False
print(a or b or c)      # True
print(a or b and c)     # False
```

Оператор условия без ветки else

Условный оператор обязательно содержит слово `if`. После слова `if` необходимо написать логическое выражение и поставить знак двоеточия. Если выражение истинно, то выполняются операторы, записанные после двоеточия. Все операторы, записанные внутри `if`, должны иметь одинаковый отступ (обычно он содержит четыре пробела).

Задача 1. Выведите сообщение, если число положительное.

Решение задачи:

```
a = int(input("Введите число: "))
if a >= 0:
    print('Число положительное')
```

Результат работы программы:

```
Введите число: 5
Число положительное

Введите число: -5
```

Оператор условия с веткой else

Оператор условия может содержать слово `else`, после которого также необходимо поставить двоеточие. В таком случае при ложном условии выполняться будут операторы, записанные после слова `else`. Все операторы, которые находятся внутри оператора `if` по ветке `else`, также должны иметь одинаковые отступы.

Задача 2. Выведите сообщение о том, что число положительное или отрицательное.

Решение задачи:

```
a = int(input("Введите число: "))
if a >= 0:
    print('Число положительное')
else:
  print('Число отрицательное')
```

Оператор `if` может быть записан в одну строку:

```
a = int(input("Введите число: "))
print('Число положительное' if a >= 0 else \
   'Число отрицательное')
```

Один оператор после двоеточия можно записывать в той же строке:

```
a = int(input("Введите число: "))
if a >= 0: print('Число положительное')
else: print('Число отрицательное')
```

Результат работы программы:

```
Введите число: -5
Число отрицательное
```

```
Введите число: 0
Число положительное
```

```
Введите число: 5
Число положительное
```

Последовательное использование операторов условия

Следующая программа не требует вложенности операторов `if`. Все операторы условия выполняются последовательно.

Задача 3. Даны две строки. Выведите на экран сообщение о возможности преобразования строки к числу.

Решение задачи:

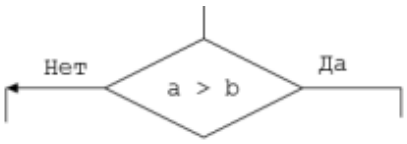
```
a = input()
b = input()
if a.isdigit(): print(a)
if b.isdigit(): print(b)
```

Результат работы программы:

```
1
b
1
```

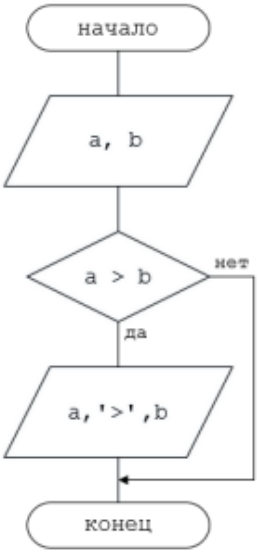
Составление блок-схем программ со структурой ветвления

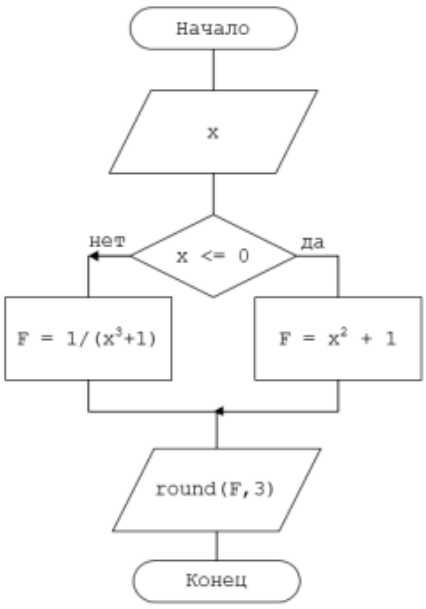
Рассмотрим правила оформления и графического изображения элементов блок-схем для программ со структурой ветвления.

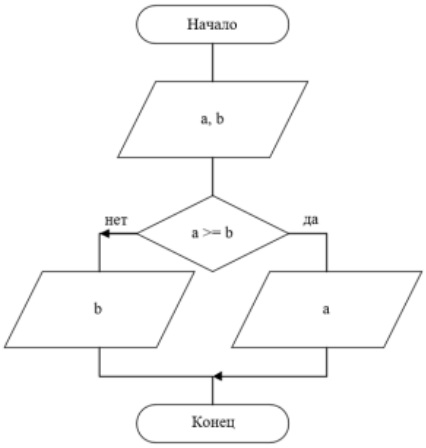
Правило оформления	Графическое изображение
Условный оператор содержит блок условия. Блок условия – ромб. Блок имеет один вход и два выхода. Два выхода необходимо рисовать всегда, даже если нет блока `else`. Над каждым из выходов следует писать слова "Да" и "Нет". Выходы будут соответствовать веткам, когда условие истинно и когда условие ложно. *Масштаб блока* высота : ширина = 1:2	

Окончание

Правило оформления	Графическое изображение

Оператор условия без ветки `else`.
 	
`a = int(input())`
`b = int(input())`
`if a > b :`
`  print(a, end = ' ')`
`  print('>', end = ' ')`
`  print(b, end = ' ')`
	

Оператор условия с веткой `else`.
 	
`x = float(input())`
`if x <= 0:`
`  F = x**2+1`
`else:`
`  F = 1/(x**3+1) print(round(F,3))`
	

Оператор условия, записанный в одну строку.
 	
`a = int(input())`
`b = int(input())`
`print(a if a >= b else b)`
	

Примеры составления блок-схем программ с ветвлением

Задача 4. Даны два целых числа. Найдите среди них максимальное и минимальное числа.

Решение задачи:

```
a = int(input())
b = int(input())
if a > b:
  print("max =",a)
  print("min =",b)
else:
  print("max =",b)
  print("min =",a)
```

Блок-схема задачи 4 представлена на рис. 3.1.

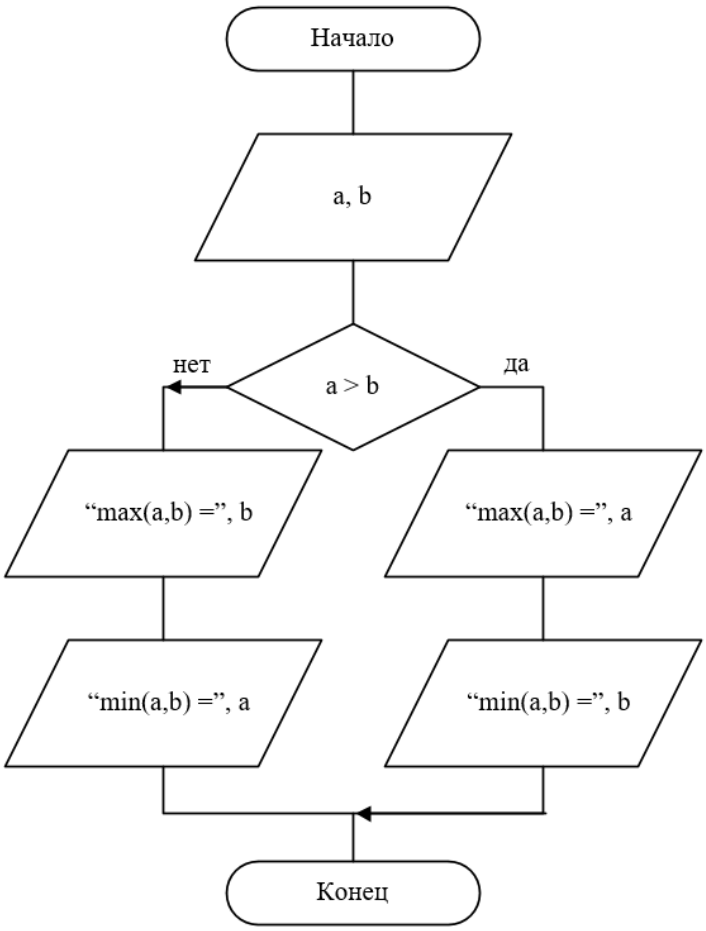


Рис. 3.1. Блок-схема программы задачи 4

Решение задачи с использованием функций `min()` и `max()`:

```
a = int(input())
b = int(input())
print("max =", max(a,b))
print("min =", min(a,b))
```

Задача 5. Даны три действительных числа. Возведите в квадрат те из них, значения которых неотрицательны, и в четвертую степень – отрицательные.

Решение задачи:

```
a,b,c = map(float, input().split())
a = a**2 if a >= 0 else a**4
b = b**2 if b >= 0 else b**4
c = c**2 if c >= 0 else c**4
print(a,b)
```

Блок-схема программы задачи 5 представлена на рис. 3.2.

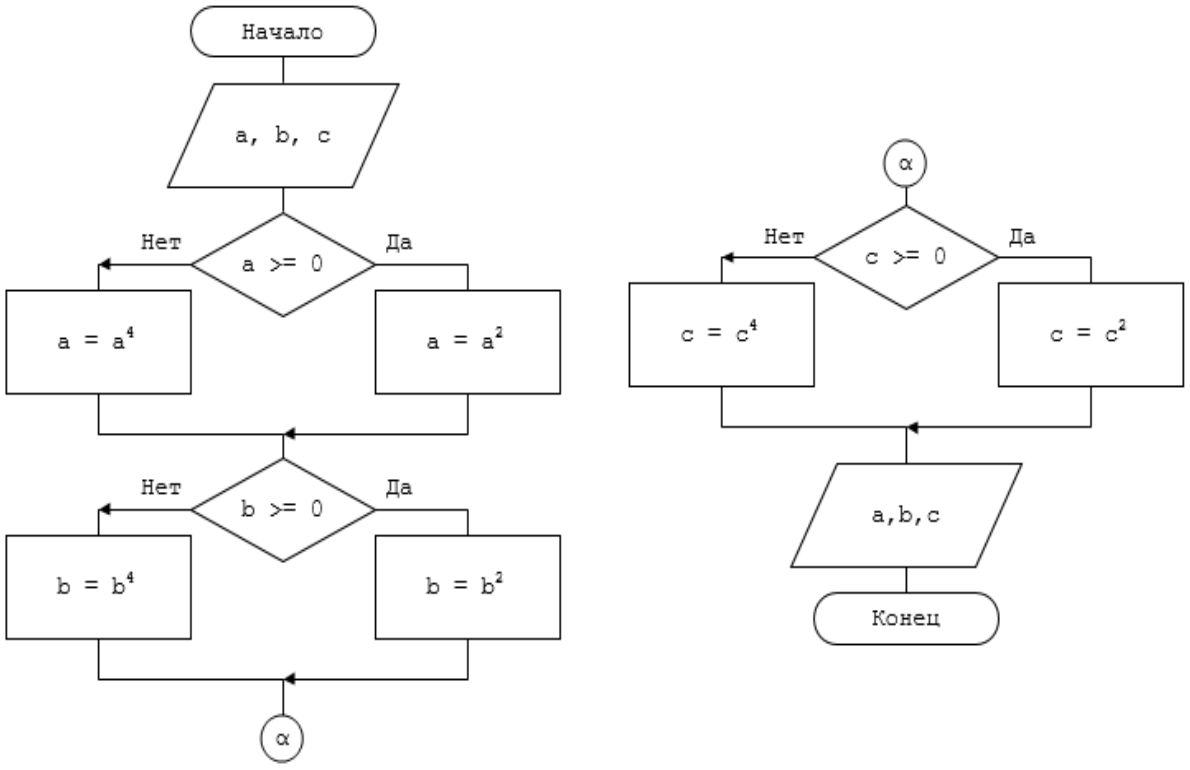


Рис. 3.2. Блок-схема программы задачи 5

Задача 6. Составьте программу вычисления арифметического выражения.

$$\frac{x}{5} - \frac{\lg\left(\sqrt{x^3 + 2\ln(x)}\right)}{\sin^2(x)}.$$

Проверьте область допустимых значений аргумента *x*. Если *x* не попадает в область допустимых значений, выведите соответствующее сообщение. Результат выведите с точностью до трех знаков после запятой.

Решение задачи:

Область допустимых значений	Условие проверки для выхода из программы
1) $\sin^2(x) \neq 0$ – знаменатель дроби 2) $x > 0$ – аргумент функции ln() 3) $x^3 + 2\ln(x) \geq 0$ – подкоренное выражение 4) $x^3 + 2\ln(x) > 0$ – аргумент функции lg()	1) $\sin^2(x) = 0$ 2) $x \leq 0$ 3) $x^3 + 2\ln(x) < 0$ 4) $x^3 + 2\ln(x) \leq 0$

Условие 4 частично (< 0) будет проверено условием 3. Поэтому оставляем в алгоритме в качестве условия 4 проверку на равенство 0: $x^3 + 2\ln(x) = 0$.

```
import math
import sys

eps = 1e-9

x = float(input("Введите значение аргумента x: "))

if math.isclose(math.sin(x), 0, abs_tol=eps):
   print("Ошибка деления на 0!")
   sys.exit()

if x <= 0:
   print("Ошибка аргумента функции ln(x)!")
   sys.exit()
```

```
if x**3 + 2*math.log(x) < 0:
   print("Ошибка вычисления корня!")
   sys.exit()

if math.isclose(x**3 + 2*math.log(x), 0,\
abs_tol=eps):
   print("Ошибка аргумента функции log(…)!")
   sys.exit()

y = x**3 + 2*math.log(x)
y = x/5 - math.log(y,10)/math.sin(x)**2

print(round(y,3))
```

Блок-схема программы задачи 6 представлена на рис. 3.3.

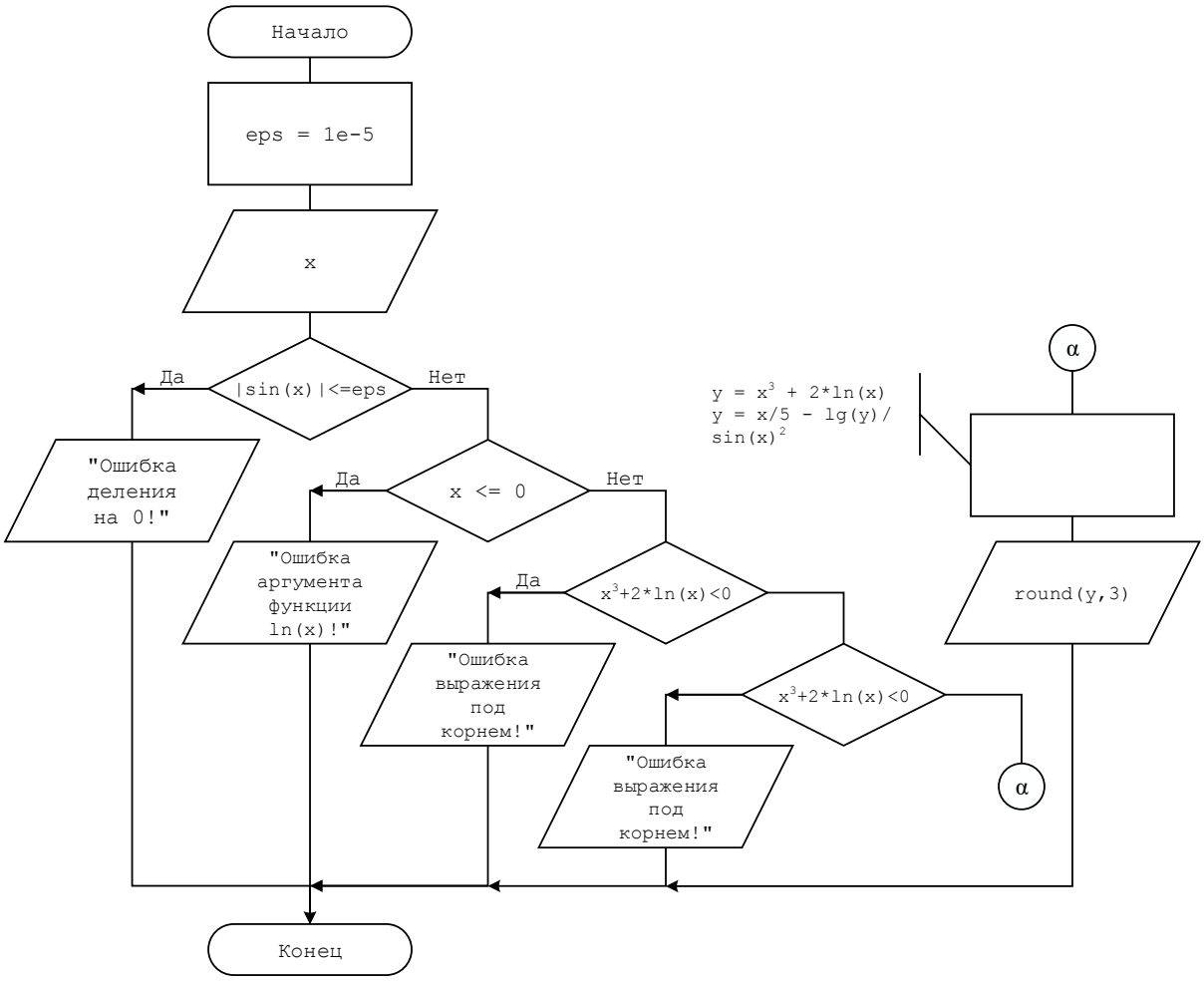


Рис. 3.3. Блок-схема программы задачи 6

Пример выполнения лабораторной работы

Цель работы: познакомиться с логическими операциями в Python. Научиться составлять логические выражения. Познакомиться с оператором ветвления. Научиться составлять программы со структурой ветвления, а также блок-схемы к программам с оператором ветвления.

Задание № 1

Вычислите значение функции с точностью до четырех знаков после запятой в соответствии с выданным вариантом задания.

$$F(x) = \begin{cases} x/2, & \text{если число } x \text{ целое} \\ (x-1)^2, & \text{если число } x \text{ дробное} \end{cases}$$

Решение задачи:

```
"""
ЛР № 3. Задание 1. Вариант 1.
Программа вычисления функции.

Программа вычисляет значение F(x). Число x вводится
с клавиатуры. Если число x целое, то F = x/2, иначе
F = (x-1)**2

студент гр. ПРИ-124 И.И. Иванов
15.09.2024
"""

x = float(input("x = "))
if x.is_integer():
    F = x/2
else:
    F =(x-1)**2
print("F =", round(F,4))
```

Результаты работы программы задания 1 представлены на рис. 3.4.

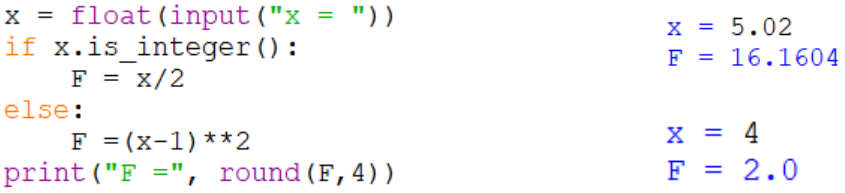


Рис. 3.4. Скриншот кода и результатов работы программы задания 1

Задание № 2

Решите задачу о принадлежности точки закрашенной области (рис. 3.5).

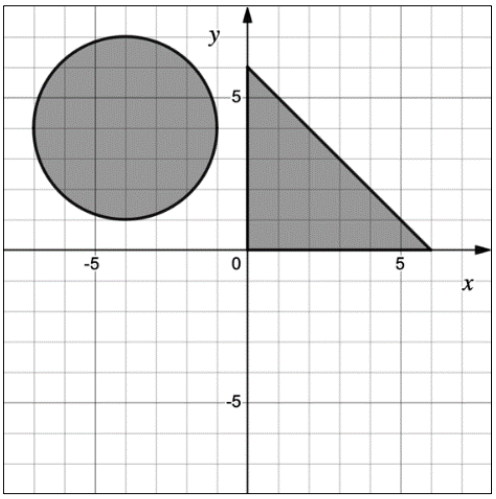


Рис. 3.5. Графическое представление области точек

Решение задачи:

```
"""
ЛР № 3. Задание 2. Вариант 1.
Программа проверки принадлежности точки области,
заданной на координатной плоскости.

Графически задана область, которая состоит из двух
частей. Программа получает на вход координаты точки
(x, y) и выдает сообщение "Точка принадлежит обла-
сти" или "Точка не принадлежит области".

студент гр. ПРИ-124 И.И. Иванов
15.09.2024
"""

x,y = map(float,input().split())
if (x+4)**2 +(y-4)**2 <= 3**2 or \ #окружность
x >= 0 and y >= 0 and y <= -x+6:  #треугольник
    print("Точка принадлежит области")
else:
    print("Точка не принадлежит области")
```

Результаты работы программы задания 2 представлены на рис. 3.6, блок-схема работы программы – на рис. 3.7.

```
x,y = map(float,input().split())
if (x+4)**2 +(y-4)**2 <= 3**2 or \
x >= 0 and y >= 0 and y <= -x+6 :
    print("Точка принадлежит области")
else:
    print("Точка не принадлежит области")
```

```
-5 5
Точка принадлежит области

2 2
Точка принадлежит области

0 0
Точка принадлежит области

-5 -5
Точка не принадлежит области
```

Рис. 3.6. Скриншот кода и результатов работы программы задания 2

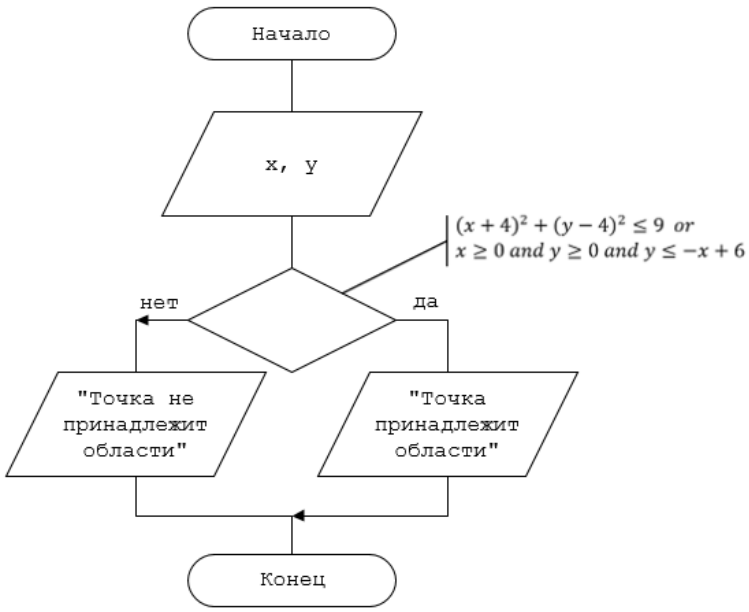


Рис. 3.7. Блок-схема программы задания 2

Вывод: Я познакомился с логическими операторами в Python. Научился составлять логические выражения. Познакомился с оператором ветвления. Научился составлять программы со структурой ветвления, а также блок-схемы к программам с оператором ветвления.

Тема 4. ВЛОЖЕННЫЕ СТРУКТУРЫ ВЕТВЛЕНИЯ

Внутри веток условного оператора могут быть расположены другие условные операторы – так называемые вложенные структуры.

Вложенные операторы могут располагаться как в обеих ветках оператора `if`, так и в одной из его веток. Другой вариант расположения – ветке `True`, операторы которой выполняются при истинном логическом выражении, или в ветке `else`, операторы которой выполняются при ложном логическом выражении.

Вложенный оператор в ветку True оператора if

Задача 1. Даны два угла треугольника (в градусах). Определите, существует ли треугольник с такими углами; если да, то будет ли он прямоугольным.

Решение задачи:

```
a,b = map(float, input().split())
c = 180 - a - b
if a > 0 and b > 0 and a + b < 180:
    print("Треугольник существует")
    if a == 90 or b == 90 or c == 90:
      print("Прямоугольный")
else:
    print("Треугольник не существует")
```

Результат работы программы:

```
60 60
Треугольник существует
```

```
30 60
Треугольник существует
Прямоугольный
```

```
30 150
Треугольник не существует
```

Блок-схема программы задачи 1 представлена на рис. 4.1.

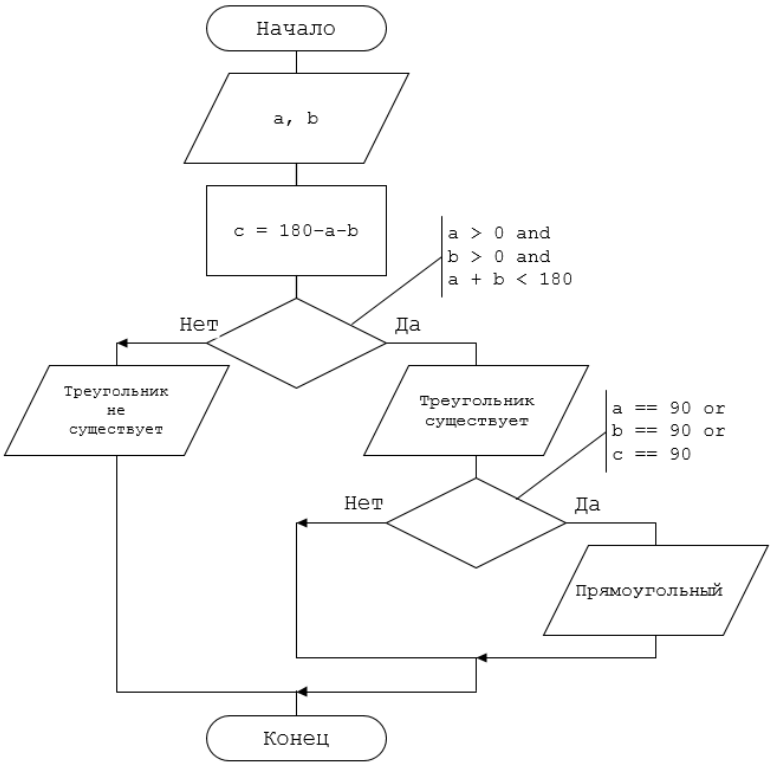


Рис. 4.1. Блок-схема программы задачи 1

Другие варианты составления блок-схемы программы задачи 1 представлены на рис. 4.2.

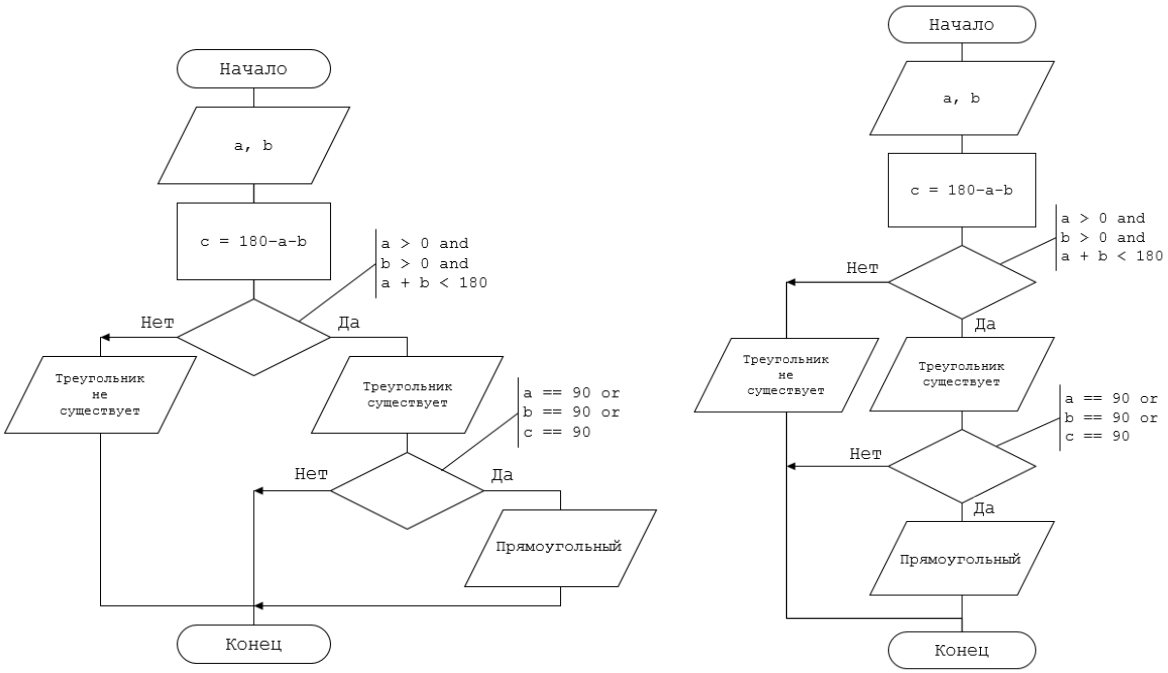


Рис. 4.2. Варианты составления блок-схемы программы задачи 1

Вложенные операторы в обе ветки оператора if

Задача 2. Даны три целых числа. Найдите максимальное среди чисел.

Решение задачи:

```
a = int(input("a: "))
b = int(input("b: "))
c = int(input("c: "))

if a > b:
    if a > c:
      print(a)
    else:
      print(c)
else:
    if b > c:
      print(b)
    else:
      print(c)
```

Результат работы программы:

```
a: 3
b: 2
c: 1
3
```

Варианты составленной блок-схемы программы задачи 2 представлены на рис. 4.3 и 4.4.

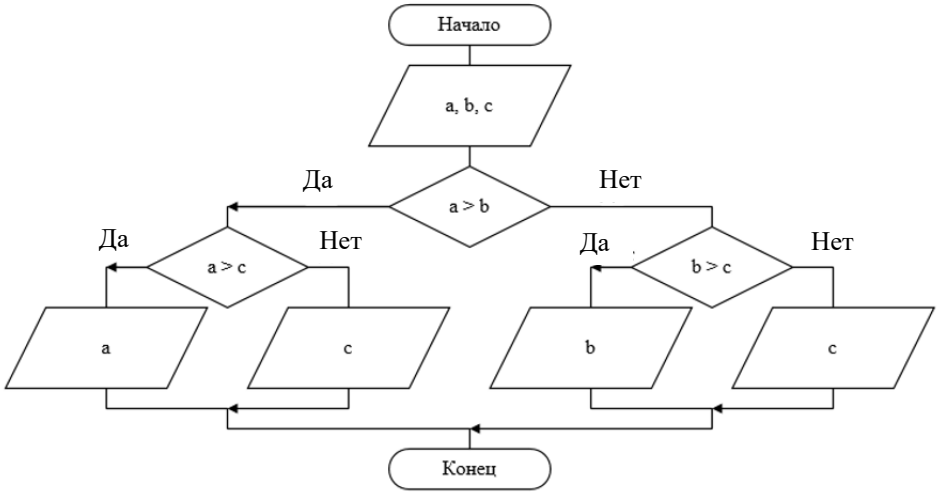


Рис. 4.3. Блок-схема программы задачи 2 (вариант 1)

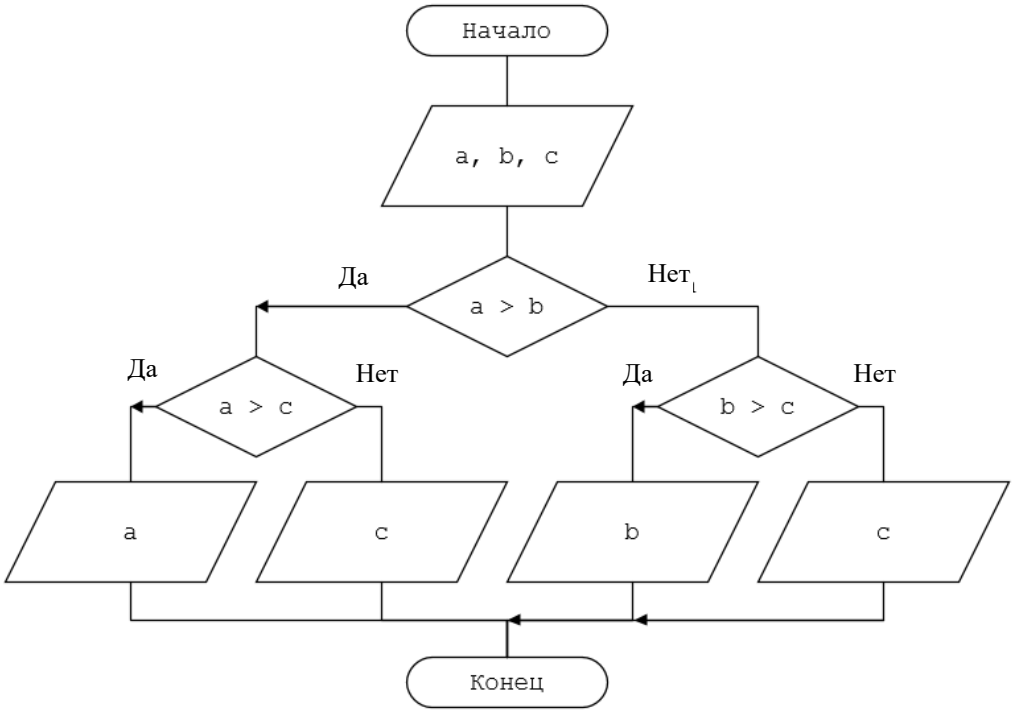


Рис. 4.4. Блок-схема программы задачи 2 (вариант 2)

Вложенный оператор в ветку else оператора if

Задача 2*. Даны три целых числа. Найдите максимальное среди чисел.

Решение задачи:

```
a, b, c = map(int, input("a b c : ").split())
if b < a > c:  # a > b and a > c
  print(a)
else:
    if b > c:
      print(b)
    else:
      print(c)
```

Результат работы программы:

```
Введите a b c : 1 2 3
3
```

Блок-схема программы задачи 2* представлена на рис. 4.5.

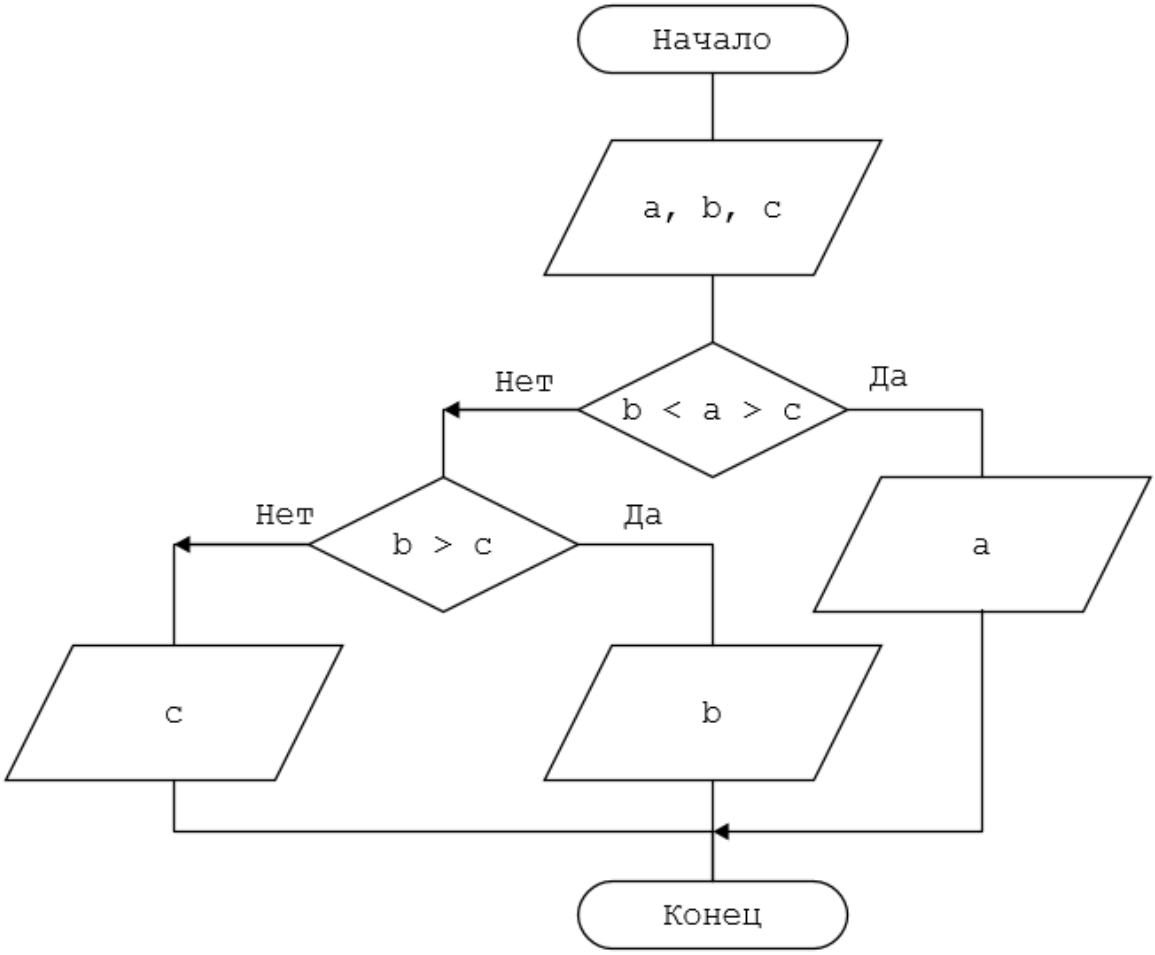


*Рис. 4.5. Блок-схема программы задачи 2**

Каскадный оператор elif

Задача 3. Дано целое число. Выведите одно из сообщений: "Число равно 0", "Число больше 0" или "Число меньше 0".

Задачу можно решить двумя способами: без использования `elif` (задача 3) и с использованием `elif` (задача 3*).

Решение задачи:

```
a = int(input("Введите число: "))
if a == 0:
    print('Число равно 0')
else:
  if a > 0:
        print('Число больше 0')
  else:
    print('Число меньше 0')
```

Результат работы программы:

```
Введите число: 5
Число больше 0
```

Оператор условия может содержать дополнительное условие `elif`. Условие `elif` будет проверяться, если первое условие `if` оказалось ложным. Ветка `else` будет выполняться в том случае, если условие `if` и все условия `elif` оказались ложными.

Такой синтаксис использования условного оператора называется каскадным оператором условия.

Задача 3*. Дано целое число. Выведите одно из сообщений: «Число равно 0», «Число больше 0» или «Число меньше 0». Программу напишите с использованием каскадного оператора условия.

Решение задачи:

```
a = int(input("Введите число: "))
if a == 0:
    print('Число равно 0')
elif a > 0:
    print('Число больше 0')
else:
  print('Число меньше 0')
```

Результат работы программы:

```
Введите число: -1
Число меньше 0
```

Блок-схема программы задач 3 и 3* представлена на рис. 4.6.

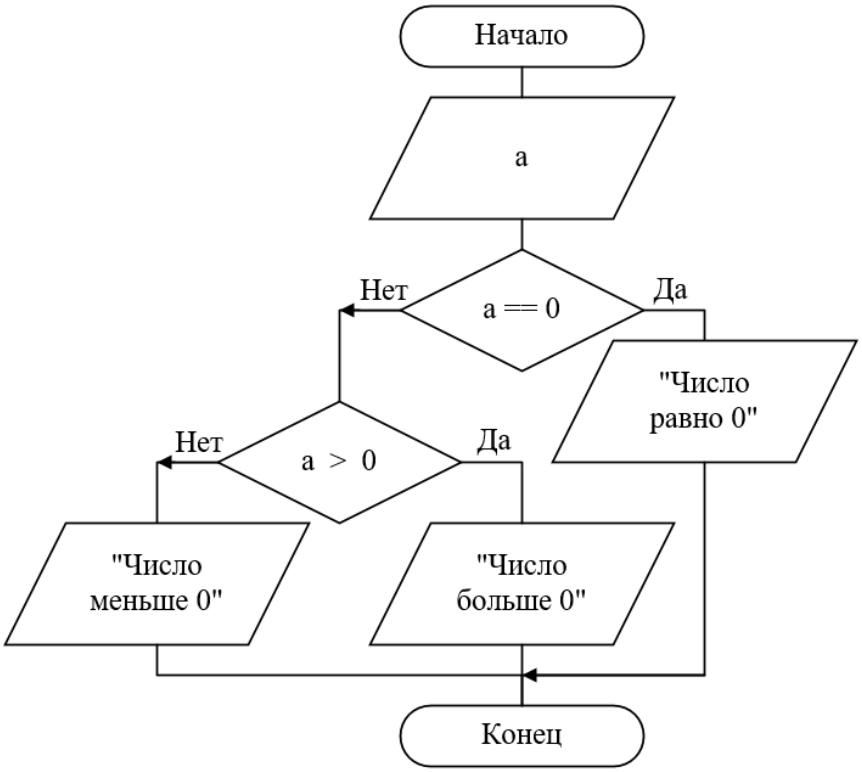


*Рис. 4.6. Блок-схема программы задач 3 и 3**

Задача 4. Дано число *n* – количество котят, *n* <= 20. Напишите программу, которая выведет сообщение о количестве котят с учетом правил склонения русского языка. При значении *n* = 0 необходимо вывести сообщение «нет котят».

Решение задачи:

```
n = int(input("n = "))
if n == 0:
print("нет котят")
elif n == 1:
print("1 котёнок")
elif n <= 4:
print(n, "котёнка")
elif n <= 20:
print(n, "котят")
else:
print("n > 20")
```

Результат работы программы:

```
n = 1
1 котёнок
```

```
n = 10
10 котят
```

Блок-схема программы задачи 4 представлена на рис. 4.7.

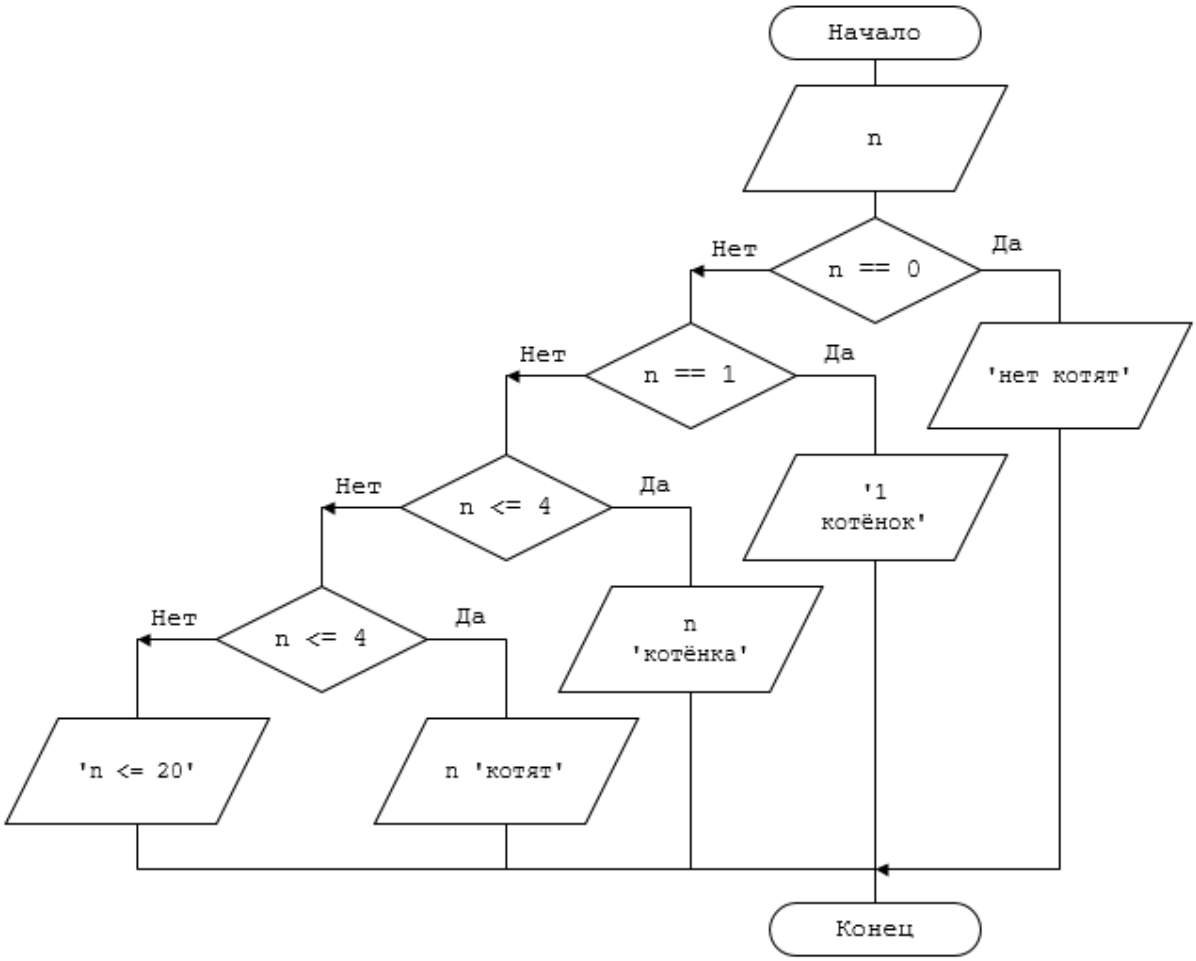


Рис. 4.7. Блок-схема программы задачи 4

Задача 5. Даны две точки $A(x_1, y_1)$ и $B(x_2, y_2)$. Составьте алгоритм, определяющий, какая из точек находится ближе к началу координат.

Решение задачи:

```
import math
print('Введите координаты точки через пробел')
x1,y1 = map(float, input('A(x1,y1) = ').split())
x2,y2 = map(float, input('B(x2,y2) = ').split())

OA = (x1**2 + y1**2)**0.5
OB = (x2**2 + y2**2)**0.5
if math.isclose(OA,OB, abs_tol = 1e-5):
    print('Точки на одинаковом расстоянии')
elif OA < OB:
    print('Точка A ближе к началу координат')
else:
    print('Точка B ближе к началу координат')
```

Блок-схема программы задачи 5 представлена на рис. 4.8.

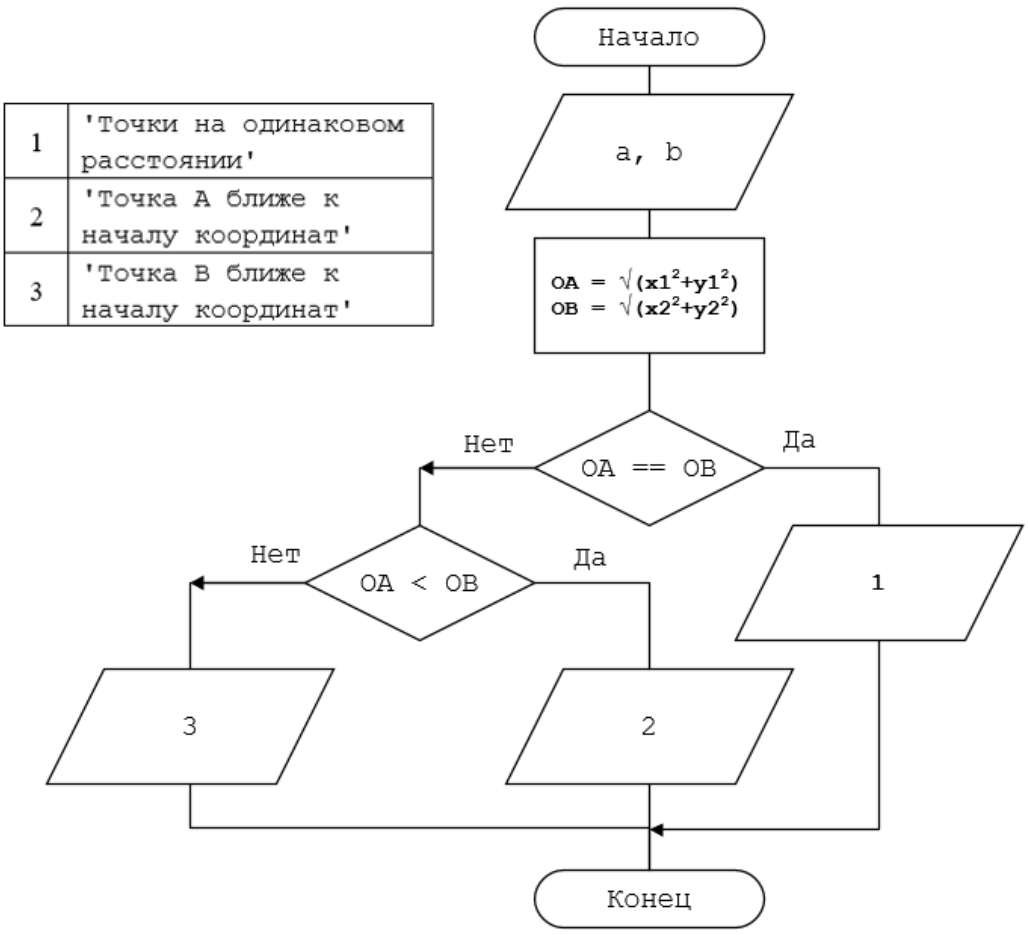


Рис. 4.8. Блок-схема программы задачи 5

Результат работы программы:

```
Введите координаты точки через пробел
A(x1,y1) = 1 1
B(x2,y2) = -2 -2
Точка A ближе к началу координат
```

```
Введите координаты точки через пробел
A(x1,y1) = 1.33333 2.33333
B(x2,y2) = -1.3333333 -2.3333333
Точки на одинаковом расстоянии
```

```
Введите координаты точки через пробел
A(x1,y1) = 5.1 5.2
B(x2,y2) = 1.5 2.5
Точка B ближе к началу координат
```

Оператор множественного выбора

Оператор множественного выбора `match-case` доступен с версии Python 3.10. Оператор по применению похож на оператор `if-elif-else`. Оператор `match-case` позволяет по значению ключа выбрать следующую ветку выполнения алгоритма. После слова `match` записывается ключ или выражение для расчета ключа. Блоки `case` являются ветками оператора множественного выбора. После слов `case` указываются конкретные значения ключа. При совпадении значения в одном из `case` выполняются операторы, расположенные в данной ветке. Ветка `case _:` повторяет блок `else` в операторе `if-elif-else`.

Задача 6. По введенному значению символа от "A" до "D" с клавиатуры выведите на экран название дисциплины с экзаменом. "A" – Информатика, "B" – Программирование, "C" – Основы веб, "D" – Математика. Используйте оператор `match-case`. При вводе других символов выведите сообщение "`Not symbol`".

Решение задачи:

```
grad = input()
match grad:
  case 'A':
    print('Информатика')
  case 'B':
    print('Программирование')
  case 'C':
    print('Основы веб')
  case 'D':
    print('Математика')
  case _:
    print("Not symbol")
```

Результат работы программы:

```
B
B
```

Блок-схема программы задачи 6 представлена на рис. 4.9.

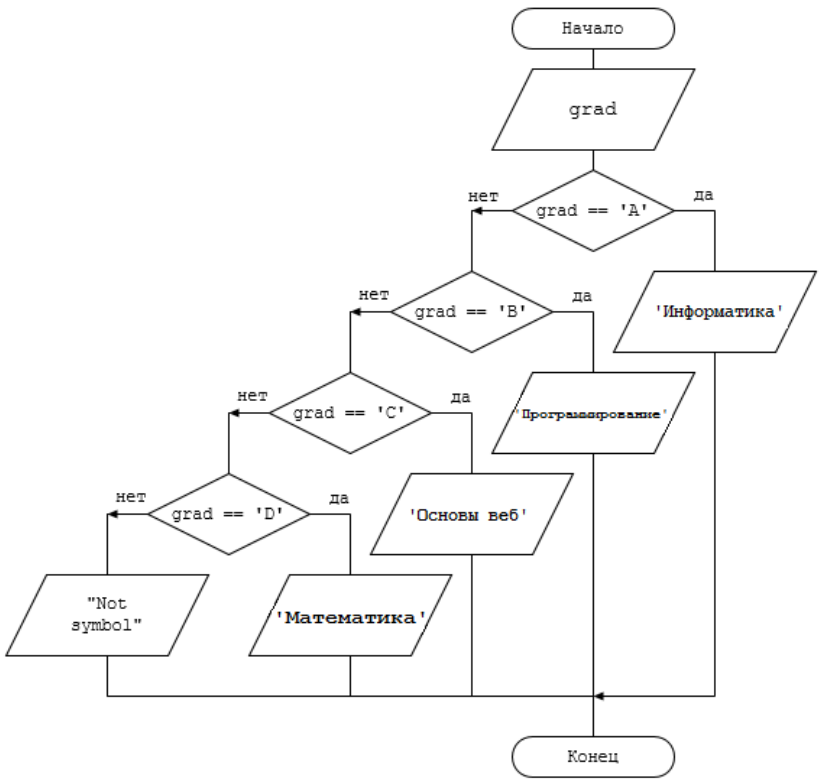


Рис. 4.9. Блок-схема программы задачи 6

Примеры составления блок-схем программ со вложенными структурами ветвления

Задача 7. Проверьте введенное число на четность и кратность 4.

Решение задачи:

```
a = int(input("Введите число: "))
if a % 2 == 0:
    print('Число четное')
    if a % 4 == 0:
       print('Число кратно 4')
   else:
      print('Число не кратно 4')
else:
    print('Число нечетное')
```

Блок-схема программы задачи 7 представлена на рис. 4.10.

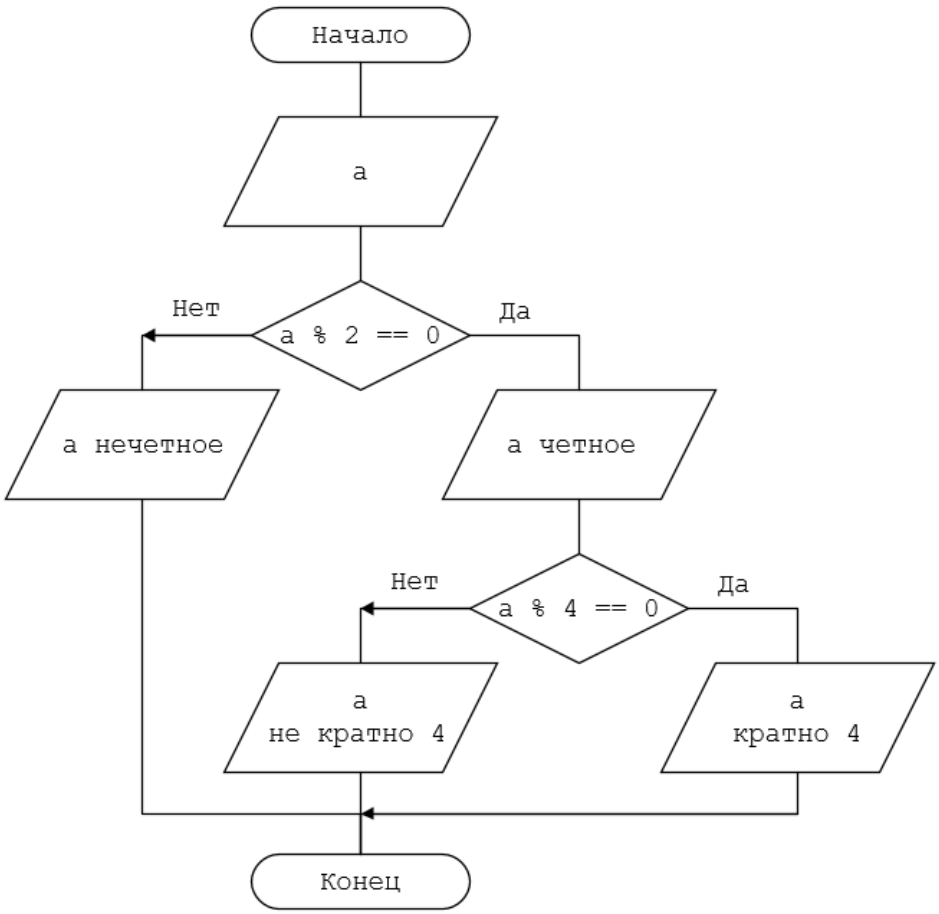


Рис. 4.10. Блок-схема программы задачи 7

Задача 8. Сравните два вещественных числа с точностью 10^{-5}.

Решение задачи:

```
import math
a = float(input("a = "))
b = float(input("b = "))
if math.isclose(a,b, abs_tol = 1e-5):
  print(a, "==", b)
elif a > b:
  print(a, ">", b)
else:
  print(a, "<", b)
```

Блок-схема программы задачи 8 представлена на рис. 4.11.

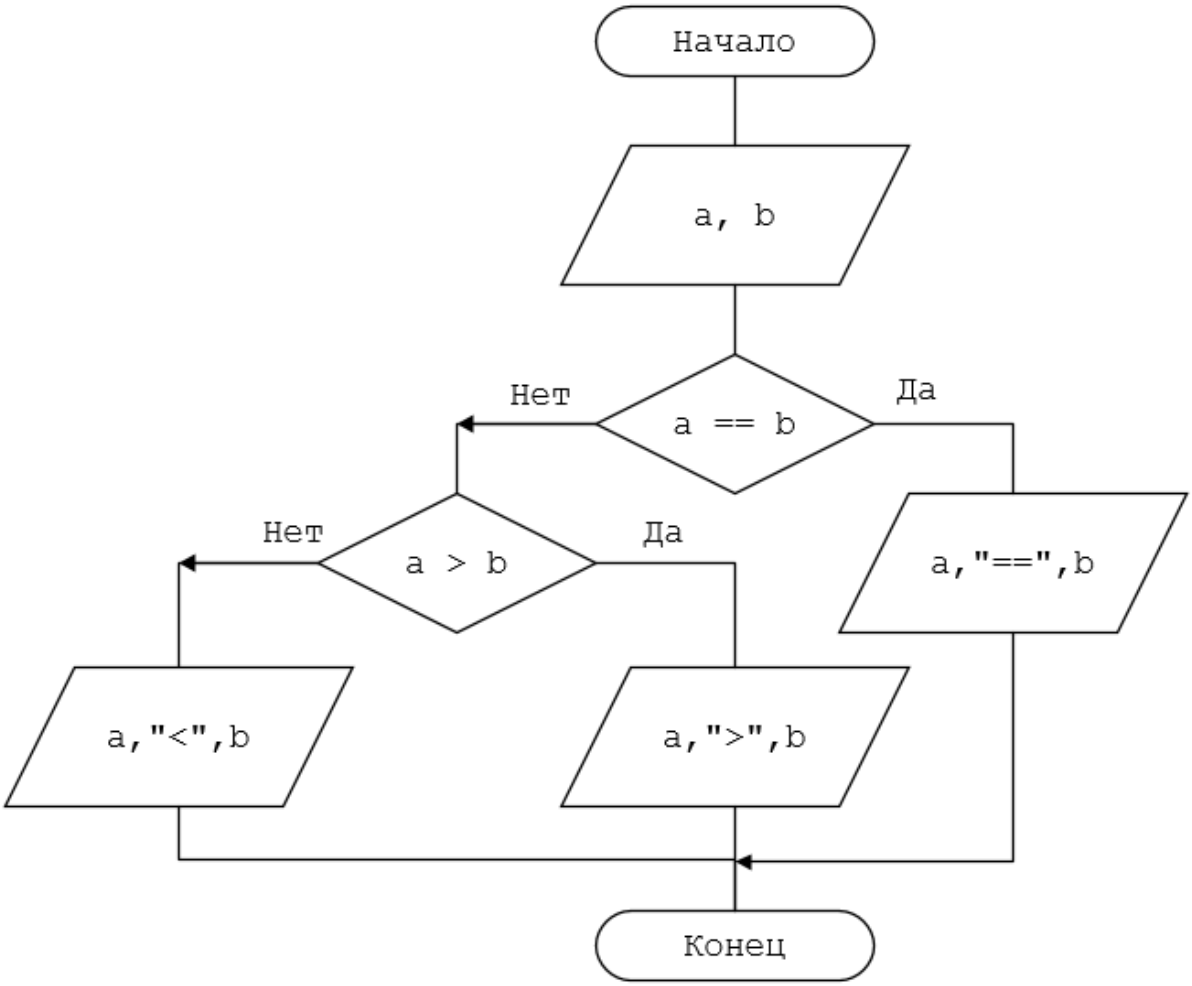


Рис. 4.11. Блок-схема программы задачи 8

Задача 9. Найдите корни квадратного уравнения.

Решение задачи:

```
    a,b,c = map(float,input("a b c : ").split())
D = b**2 - 4*a*c
if D < 0 :
    print("корней нет")
elif D == 0:
    print("x = %.3f" % (-b/(2*a)))
else:
    x1 = -b + D **(1/2)/(2*a)
    x2 = -b - D **(1/2)/(2*a)
    print("x1 = %.3f" % x1 )
    print("x2 = %.3f" % x2 )
```

Блок-схема программы задачи 9 представлена на рис. 4.12.

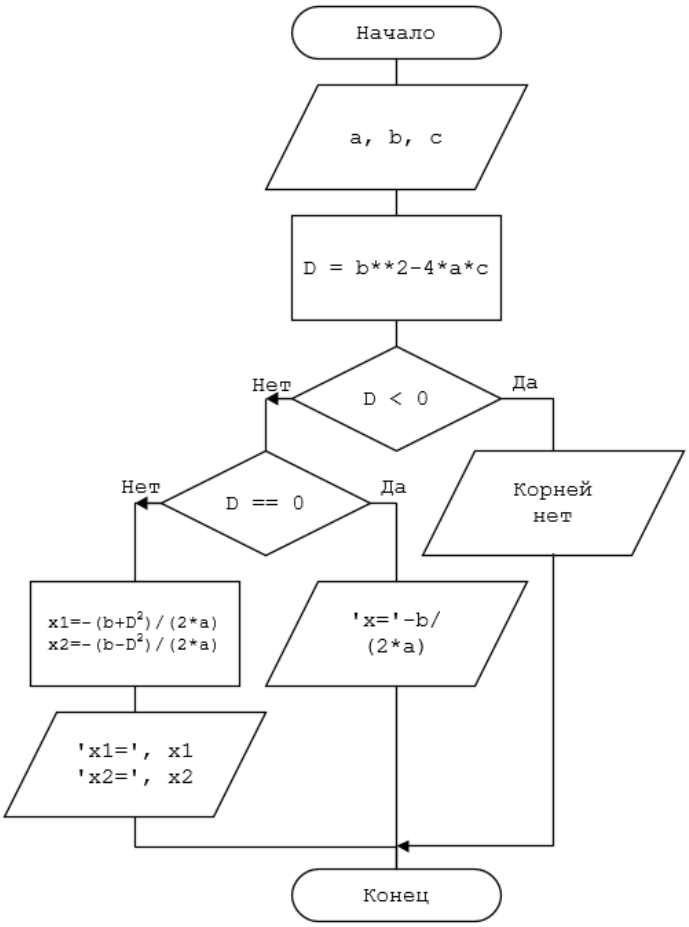


Рис. 4.12. Блок-схема программы задачи 9

Пример выполнения лабораторной работы

Цель работы: научиться составлять программы со вложенными структурами ветвления, а также блок-схемы к программам со вложенными операторами ветвления.

Задание № 1

На плоскости *XOY* задана своими координатами точка *A*. Укажите, где она расположена (на какой оси или в каком координатном угле).

Решение задачи:

```
"""
ЛР № 4. Задание 1. Вариант 1.
Программа проверки какой четверти принадлежит точка
A(x,y).

Координаты (x,y) точки A вводятся с клавиатуры. Программа печатает сообщение о том, какой четверти координатной плоскости принадлежит точка. Программа выводит сообщение, что точка является центром координат или расположена на осях координат с точностью 1E-5.

студент гр. ПРИ-124 И.И. Иванов
22.09.2024
"""

x,y = map(float, input().split())
if abs(y) <= 1E-5 and abs(x) <= 1E-5:
    print("Точка в центре координат")
elif abs(y) <= 1E-5:
    print("Точка на оси OX")
elif abs(x) <= 1E-5:
    print("Точка на оси OY")
elif x > 0:
    if y > 0 :
      print("1 четверть")
    else:
      print("2 четверть")
```

```
else:
    if y > 0 :
      print("4 четверть")
    else:
      print("3 четверть")
```

Результаты работы программы задания 1 представлены на рис. 4.13, блок-схема – на рис. 4.14.

```
x,y = map(float, input().split())
if abs(y) <= 1E-5 and abs(x) <= 1E-5:
    print("Точка в центре координат")
elif abs(y) <= 1E-5:
    print("Точка на оси OX")
elif abs(x) <= 1E-5:
    print("Точка на оси OY")
elif x > 0:
    if y > 0 :
        print("1 четверть")
    else:
        print("2 четверть")
else:
    if y > 0 :
        print("4 четверть")
    else:
        print("3 четверть")
```

```
10 10
1 четверть

1.234 -5.678
2 четверть

-10.987  -5.654
3 четверть

-3.4567 1.0203
4 четверть

-1 0.000001
Точка на оси OX

0.0000001 1
Точка на оси OY

0.000001 0
Точка в центре координат
```

Рис. 4.13. Скриншот кода и результатов работы программы задания 1

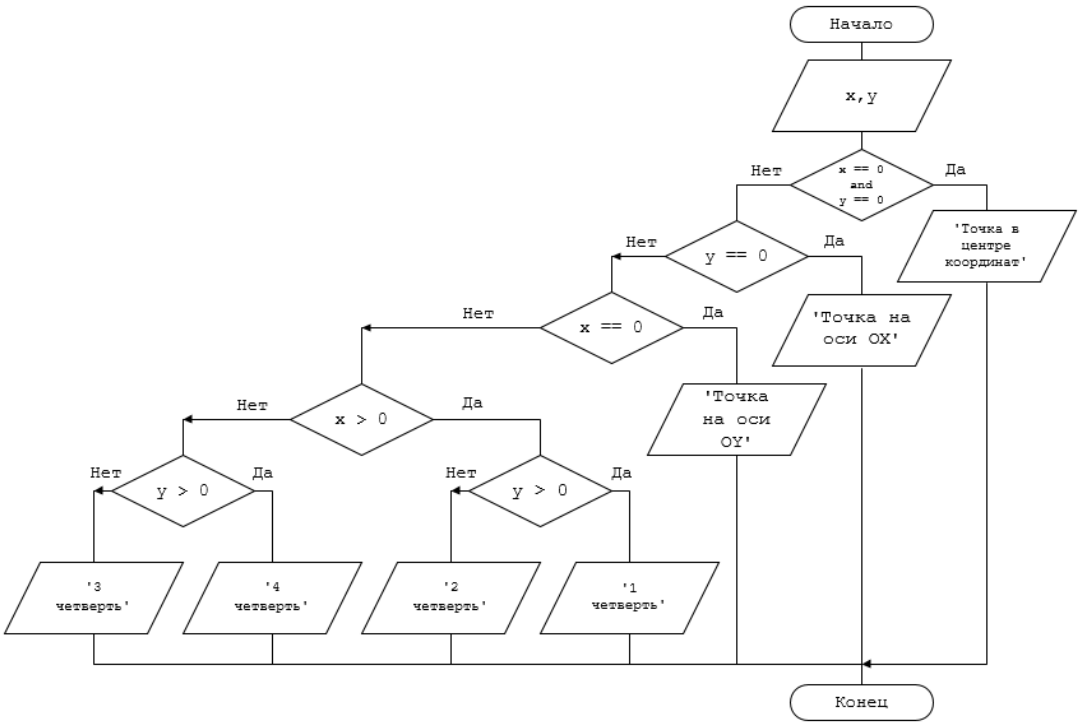


Рис. 4.14. Блок-схема программы задания 1

Задание № 2

Напишите программу, позволяющую по последней цифре числа определить последнюю цифру его квадрата.

Решение задачи:

```
"""
ЛР № 4. Задание 2. Вариант 1.
Программа вычисления последней цифры квадрата вве-
денного числа.

Число x вводится с клавиатуры. Программа печатает
последнюю цифру квадрата числа x, при этом число x
не возводится в квадрат. Последняя цифра квадрата
числа зависит от последней цифры числа.

студент гр. ПРИ-124 И.И. Иванов
22.09.2024
"""

x = int(input())
if x == 0: print(0)
elif x == 1 or x == 9: print(1)
elif x == 2 or x == 8: print(4)
elif x == 3 or x == 7: print(9)
elif x == 4 or x == 6: print(6)
else: print(5)
```

Результаты работы программы задания 2 представлены на рис. 4.15, блок-схема – на рис. 4.16.

```
x = int(input("x = "))
if x == 0:
    print(0)
elif x == 1 or x == 9:
    print(1)
elif x == 2 or x == 8:
    print(4)
elif x == 3 or x == 7:
    print(9)
elif x == 4 or x == 6:
    print(6)
else:
    print(5)
```

```
x = 0
0

x = 1
1

x = 5
5

x = 7
9

x = 9
1
```

Рис. 4.15. Скриншот работающей программы задания 2

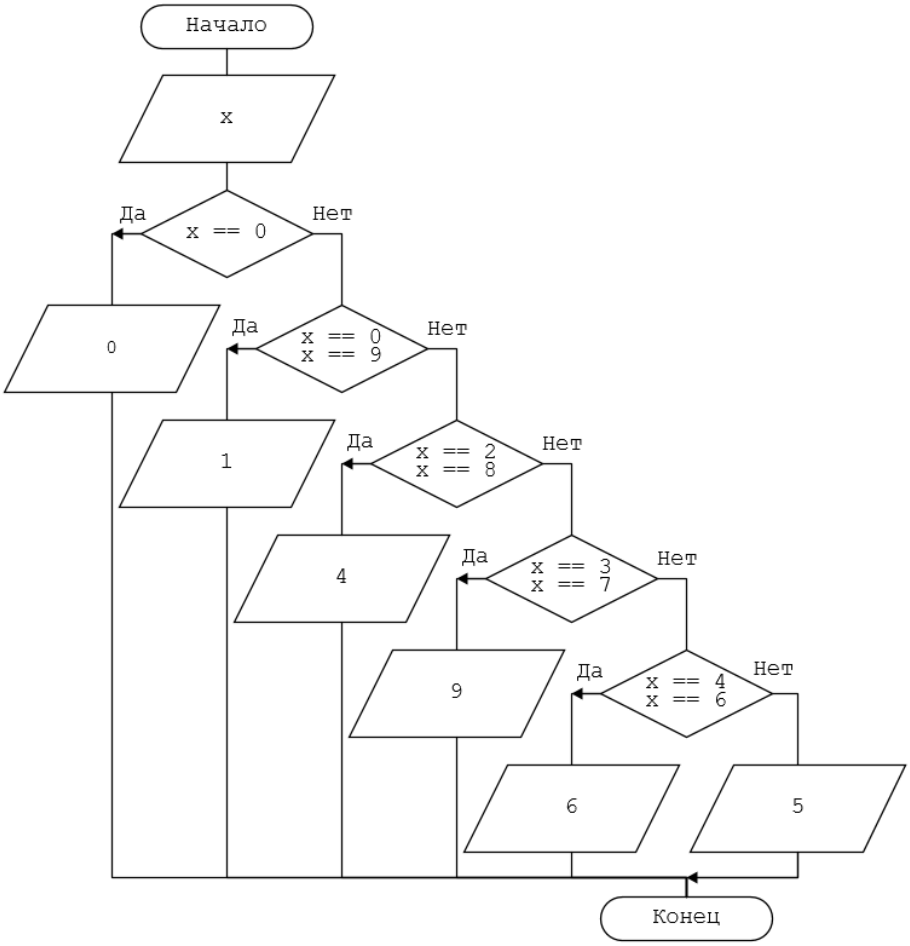


Рис. 4.16. Блок-схема программы задания 2

Вывод: Я научился составлять программы со вложенными структурами ветвления, а также блок-схемы к программам со вложенными операторами ветвления.

Тема 5. ЦИКЛЫ

В Python есть два оператора цикла: цикл с предусловием `while` и цикл со счетчиком `for`.

Цикл с предусловием while

Оператор `while` представляет собой оператор с предусловием. После слова `while` записывается логическое выражение, которое является условием выполнения цикла. Далее после двоеточия записываются операторы, которые находятся внутри цикла и составляют тело цикла.

Выполнение один раз тела цикла называется итерацией.

Все операторы тела цикла должны начинаться с одинакового отступа (обычно это четыре пробела).

Задача 1. Выведите на экран числа от 0 до 10.

Решение задачи:

```
x = 0
while x <= 10:
    print(x, end = ' ')
    x += 1
```

Результат работы программы:

```
0 1 2 3 4 5 6 7 8 9 10
```

В задаче 1 цикл выполняется 11 раз. Первая итерация цикла начинается при $x = 0$. Во время каждой итерации на экран выводится значение переменной x, далее значение этой переменной увеличивается на 1. Последняя итерация начинается при $x = 10$. Значение 10 выводится на экран, и переменная x становится равна 11. При $x = 11$ условие выполнения цикла становится ложным, цикл не выполняется и программа заканчивает свою работу.

Задача 2. Выведите на экран четные числа от 0 до 20.

Решение задачи:

```
x = 0
while x <= 20:
    print(x, end = ' ')
    x += 2
```

Результат работы программы:

```
0 2 4 6 8 10 12 14 16 18 20
```

Отличие задачи 2 от задачи 1 заключается в том, что переменная *x* с каждой итерацией цикла увеличивается не на 1, а на 2. При начальном значении *x* = 0 на экран будут выведены только положительные числа. Последнее значение выводимой последовательности равно 20, так как этим значением ограничивается вывод в условие выполнения цикла.

Если записать в качестве условия выполнения цикла значение `True`, то получим бесконечный цикл. Далее приведен цикл, который будет выполняться бесконечное число раз.

```
x = 0
while True:  # x >= 0
    print(x, end = ' ')
    x += 2
```

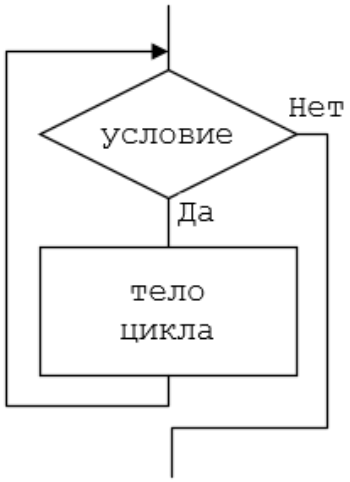
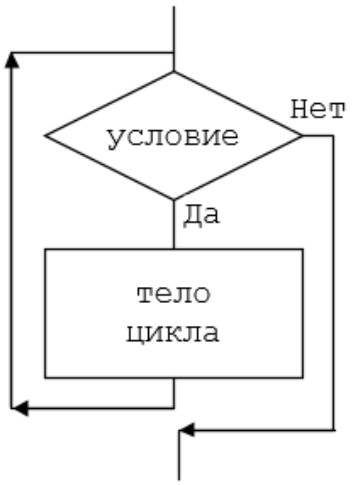
В качестве условия можно записать любое истинное логическое выражение. Также бесконечным будет цикл, если не выполнить внутри цикла действие, приближающее к завершению цикла. Например, не написать последнюю строку или написать *x* -= 2.

```
x = 0
while x <= 20:
    print(x, end = ' ')
    x += 2  # x -= 2
```

Такие ошибки называются зацикливанием программы. Если программа выполняется долго и не заканчивает работу, то ошибку следует искать прежде всего в циклах.

Составление блок-схем программ со структурой цикла while

Рассмотрим правила оформления и графического изображения элементов блок-схем для программ циклической структуры с оператором `while`.

Правило оформления	Графическое изображение
Оператор `while` на блок-схеме отображается минимум двумя блоками – блоком условия и блоком процесса. Блоки тела цикла располагаются под блоком условия. После последнего блока соединительная линия должна подниматься вверх и попадать в блок условия сверху. На рисунке справа представлена упрощенная схема отображения цикла с предусловием.	
Обратную стрелку цикла `while` необходимо оформлять так, чтобы линии, идущие вверх и справа налево, заканчивались стрелками.	

Построим блок-схему к задаче 2 (рис. 5.1.)

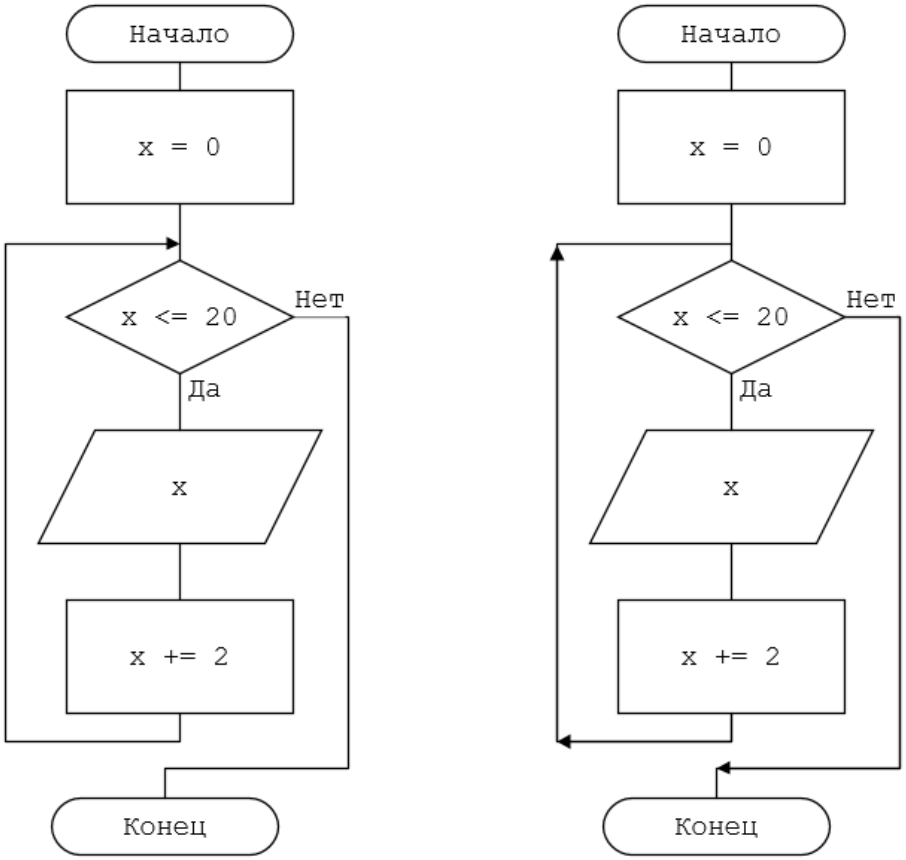


Рис. 5.1. Варианты построения блок-схемы программы задачи 2

Внутри цикла может находиться оператор или операторы условия.

Задача 3. Выведите на экран числа от 0 до 30, не кратные 3.

Решение задачи:

```
x = 0
while x <= 30:
  if x % 3 != 0 :
    print(x, end = ' ')
  x += 1
```

Результат работы программы:

```
1 2 4 5 7 8 10 11 13 14 16 17 19 20 22 23 25 26 28 29
```

Блок-схема работы программы задачи 3 представлена на рис. 5.2.

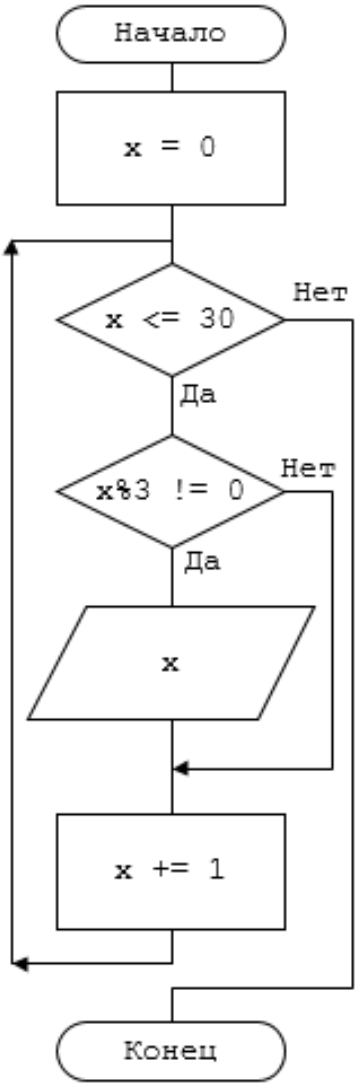


Рис. 5.2. Блок-схема программы задачи 3

Задача 4. Выведите на экран числа от 0 до 30, оканчивающиеся на 5.

Решение задачи:

```
x = 0
while x <= 30:
    if x % 10 == 5:
      print(x, end = ' ')
    x += 1
```

```
# или
x = 5
while x <= 30:
  print(x)
  x += 10
```

Результат работы программы:

```
5 15 25
```

Задача 5 (задача табулирования функции). Протабулируйте функцию $y = \sin(x)$ на интервале $x = 0$ до 2π с шагом $h = \pi/6$.

Решение задачи:

```
from math import sin, pi, degrees
x = 0
h = pi/6
print('%5s%9s' %('x','y'))
while x <= 2*pi+1E-5:
  y = sin(x)
  print("%7.3f %7.3f" %(degrees(x),y))
  x += h
```

Результат работы программы выводят на экран в табличной форме.

Результат работы программы:

```
    x      y
  0.000   0.000
 30.000   0.500
 60.000   0.866
 90.000   1.000
120.000   0.866
150.000   0.500
180.000   0.000
210.000  -0.500
240.000  -0.866
270.000  -1.000
300.000  -0.866
330.000  -0.500
360.000   0.000
```

Задача 6 (задача табулирования функции). Протабулируйте функцию *y = f(x)* на отрезке от –10 до 10 с шагом *h* = 2,5.

$$y = \begin{cases} x^2 + 5, & \text{если } x < 0 \\ x + 5, & \text{если } x \geq 0 \end{cases}$$

Решение задачи:

```
print('%5s%11s' %('x','y'))
a,b = -10,10
x = a; h = 1/3
while x <= b + 1E-5:
  if x < 0:
    y = x*x + 5
  else:
    y = x + 5
  print("%7.3f %10.3f" % (x,y) )
  x += h
```

Результат работы программы:

```
    X         y
-10.000  105.000
 -7.500   61.250
 -5.000   30.000
 -2.500   11.250
  0.000    5.000
  2.500    7.500
  5.000   10.000
  7.500   12.500
 10.000   15.000
```

Блок-схема работы программы задачи 6 представлена на рис. 5.3.

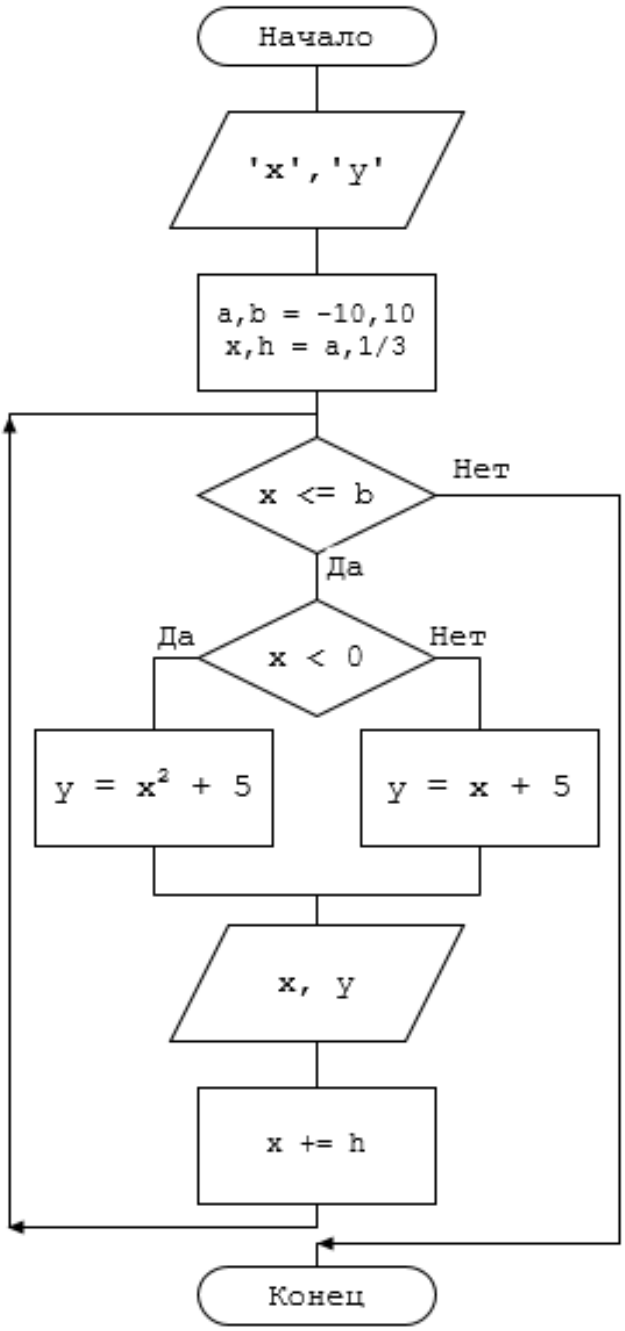


Рис. 5.3. Блок-схема программы задачи 6

Операторы continue и break

Оператор `continue` прерывает выполнение текущей итерации цикла и переводит цикл к выполнению следующей итерации.

Задача 7. Выведите на экран числа от 0 до 30, не кратные 3, с использованием оператора `continue`.

Решение задачи:

```
x = -1
while x < 30:
  x += 1
  if x % 3 == 0:
    continue
  print(x, end = ' ')
```

Результат работы программы:

```
1 2 4 5 7 8 10 11 13 14 16 17 19 20 22 23 25 26 28 29
```

Блок-схема работы программы задачи 7 представлена на рис. 5.4.

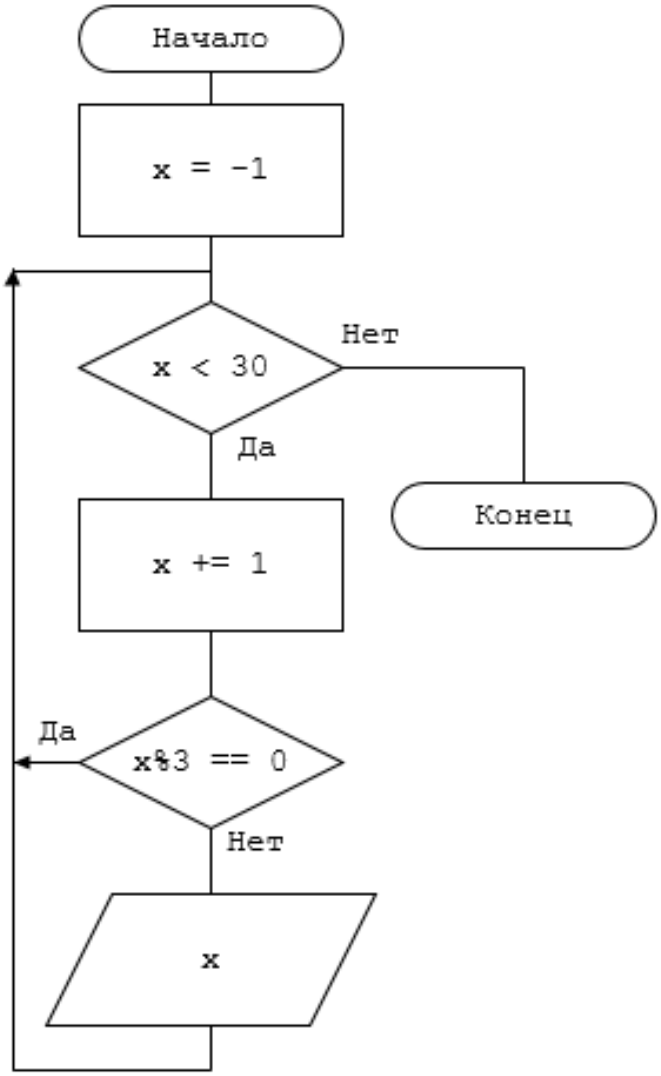


Рис. 5.4. Блок-схема программы задачи 7

При значении переменной *x*, которое кратно 3, вывод на экран данного значения не выполняется, так как после выполнения оператора `continue` выполнение программы переходит на проверку условия $x < 30$, т. е. в начало цикла.

На блок-схеме оператор `continue` не имеет отдельного блока. Он отображается в виде линии, которая должна попадать в начало цикла.

Для того чтобы выйти из цикла, можно использовать оператор `break`. Оператор `break` завершает выполнение цикла и передает управление оператору, следующему за циклом.

Задача 8. Выведите на экран числа от 0 до 30, не кратные 3, с использованием `break`.

Решение задачи:

```
x = 0
while True:
    x += 1
    if x % 3 != 0:
        print(x, end = ' ')
    if x >= 30:
       break
print("Вывод завершен")
```

Результат работы программы:

```
1 2 4 5 7 8 10 11 13 14 16 17 19 20 22 23 25 26 28 29
Вывод завершен
```

Блок-схема работы программы задачи 8 представлена на рис. 5.5.

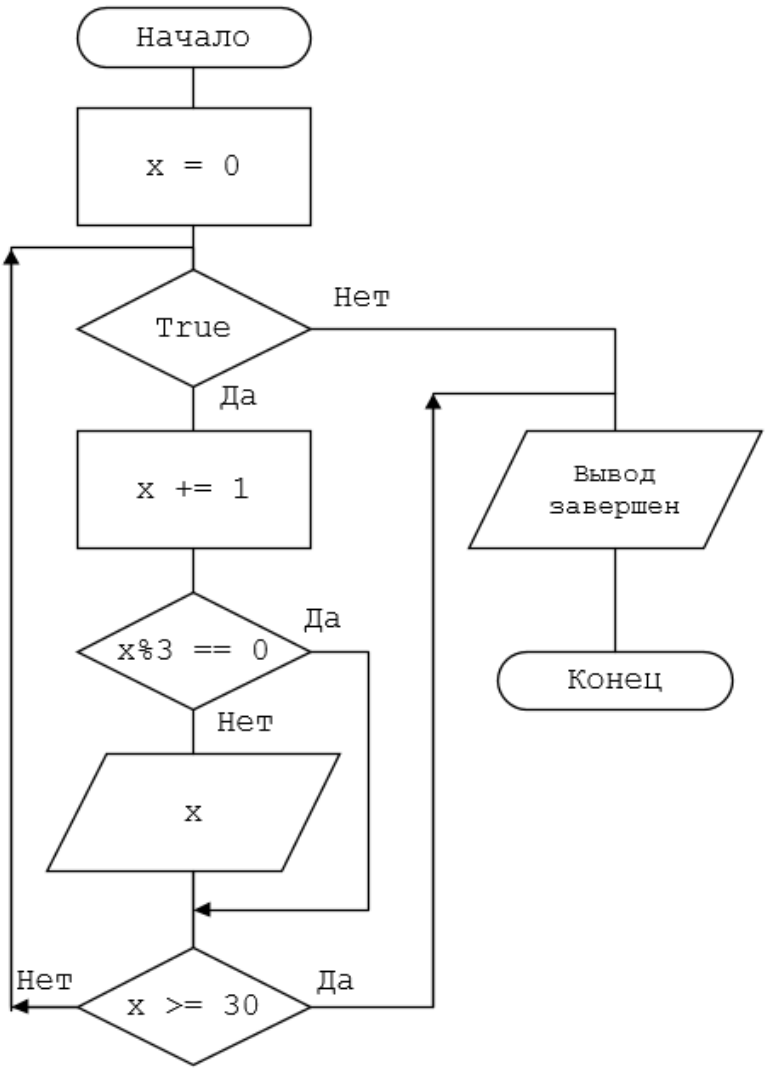


Рис. 5.5. Блок-схема программы задачи 8

Задача 9. Выведите на экран числа от 0 до 30, не кратные 3, с использованием операторов `continue` и `break`.

Решение задачи:

```
x = -1
while True:
    x += 1
    if x % 3 == 0:
      continue
    if x >= 30:
        break
    print(x, end = ' ')
print("Вывод завершен")
```

Результат работы программы:

```
1 2 4 5 7 8 10 11 13 14 16 17 19 20 22 23 25 26 28 29
Вывод завершен
```

Блок-схема работы программы задачи 9 представлена на рис. 5.6.

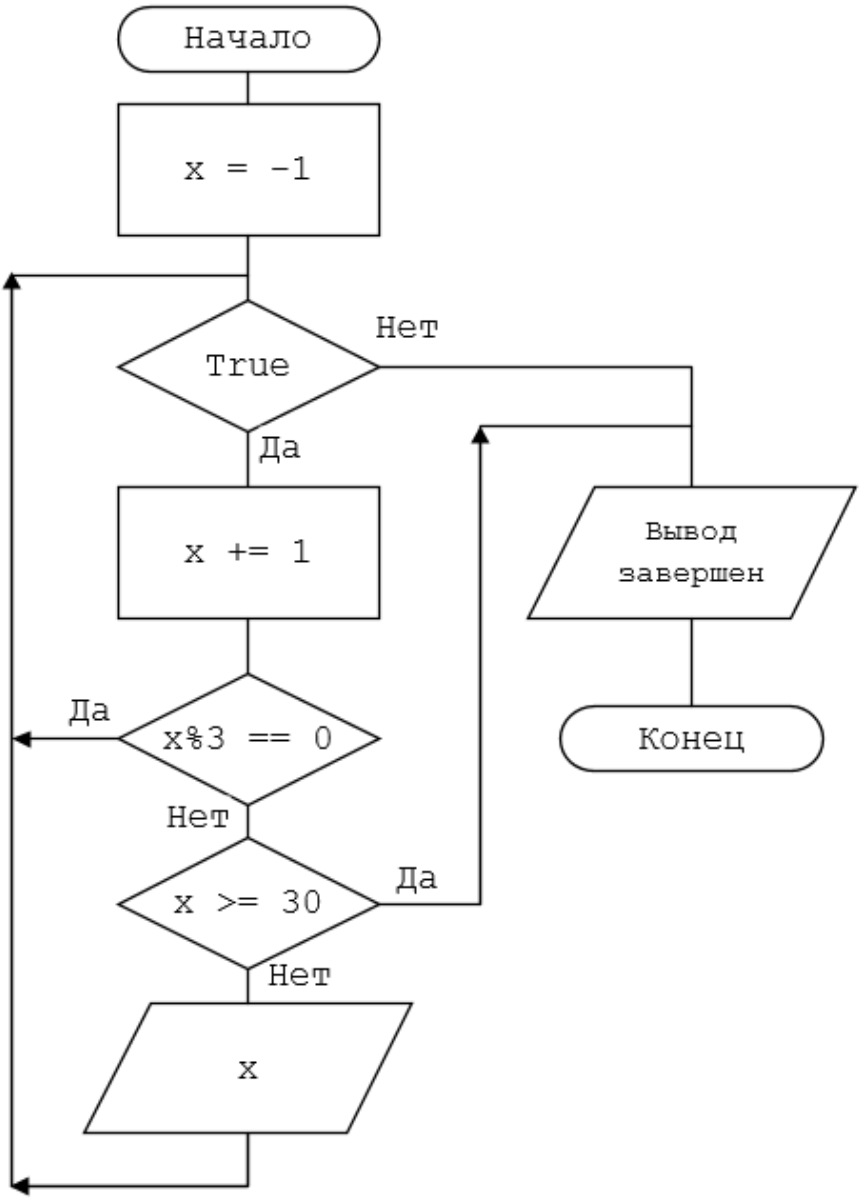


Рис. 5.6. Блок-схема программы задачи 9

На блок-схеме оператор `break` не имеет отдельного блока. Он отображается в виде линии, которая должна попадать в конец цикла.

Блок else оператора while

Оператор `while` может содержать блок `else`. Операторы блока `else` выполняются после последней итерации, если не было принудительного выхода из цикла оператором `break`.

Задача 10. Выведите на экран числа от 0 до 10 с шагом 2. После вывода элементов напечатайте сообщение «Вывод завершен». Если при выводе элементов встречается значение 5, то это значение выводить на экран не нужно. Завершите вывод элементов без печати сообщения.

Решение задачи:

```
x = int(input('x = '))
while x <= 10:
  print(x, end = ' ')
  x += 2
  if x == 5:
    break
else:
  print("Вывод завершен")
```

В данной программе при значении $x = 5$ выполняется принудительный выход из цикла. Если переменная x во время вычислений не будет равняться 5, то цикл завершается по условию $x > 10$. После последней итерации выполняются операторы блока `else`, т. е. выводится сообщение «Вывод завершен».

Результаты работы программы:

```
x = 0
0 2 4 6 8 10 Вывод завершен
```

```
x = 7
7 9 Вывод завершен
```

```
x = 1
1 3
```

Блок-схема работы программы задачи 10 представлена на рис. 5.7.

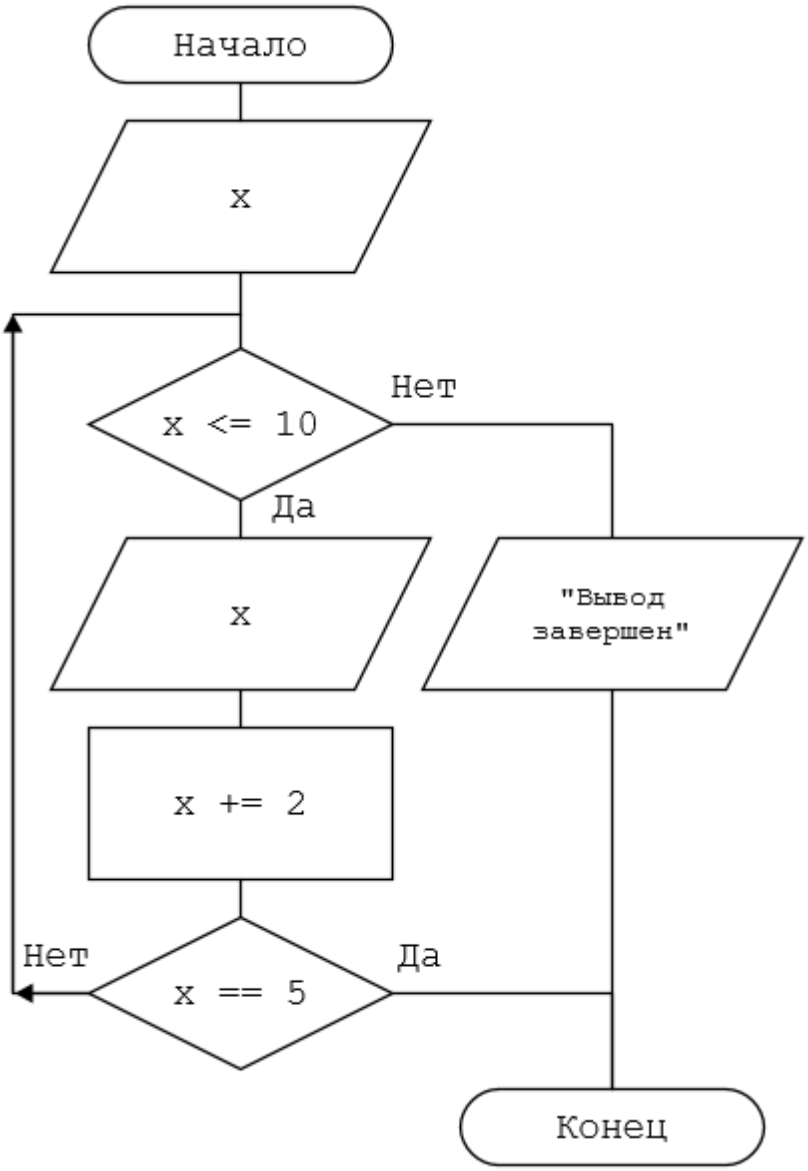


Рис. 5.7. Блок-схема программы задачи 10

Цикл со счетчиком for

Оператор `for` работает с обязательной переменной, которую называют счетчиком цикла. Переменная-счетчик *i* перебирает все элементы итерируемого объекта. Таким объектом может быть список, строка, кортеж. Переменная-счетчик должна совпадать с типом элементов итерируемого объекта.

Задача 11. Выведите на экран значения от 0 до 9.

Решение задачи:

```
for i in [0,1,2,3,4,5,6,7,8,9]:
    print(i, end = ' ')
print()
```

Результат работы программы:

```
0 1 2 3 4 5 6 7 8 9
```

Переменная-счетчик *i* перебирает все значения из списка [0, 1, 2, …, 9]. На экран будут выведены все значения от 0 до 9.

Функция `range(N)` возвращает список от 0 до $N - 1$:

```
range(10) = [0,1,2,3,4,5,6,7,8,9]
```

Последнее значение списка равно $N - 1 = 9$.

Задача 12. Выведите на экран значения от 0 до 9.

Решение задачи с использованием `range()`:

```
for i in range(10):
    print(i, end = ' ')
print()
```

Результат работы программы:

```
0 1 2 3 4 5 6 7 8 9
```

Другие варианты использования функции `range(N)`:

```
range(0,10,2) # [0,2,4,6,8]
```

Начальное значение – 0, шаг – 2, числа списка – строго меньше 10.

```
range(1,10,2) # [1,3,5,7,9]
```

Начальное значение – 1, шаг – 2, числа списка – строго меньше 10.

Переменная *i* принимает значения из итерируемого объекта, которым может быть любой список, объявленный в коде программы.

Задача 13. Выведите на экран названия времен года.

Решение задачи:

```
A = ['зима', 'весна', 'лето', 'осень']
for i in A:
    print(i, end = ' ')
print()
```

Результат работы программы:

```
зима весна лето осень
```

Переменная *i* принимает значения из итерируемого объекта, которым может быть строка.

Задача 14. Выведите на экран символы строки через пробел.

Решение задачи:

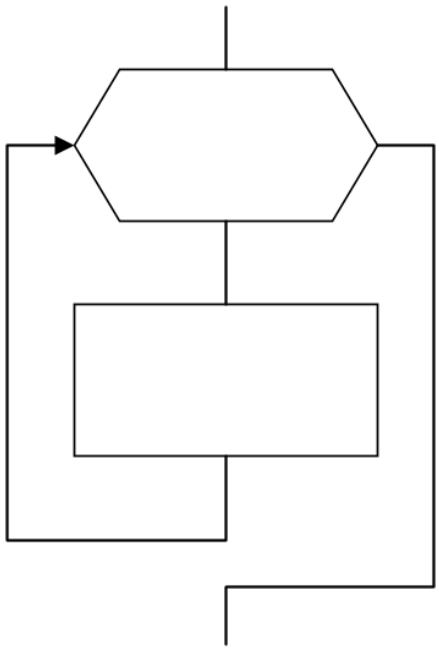
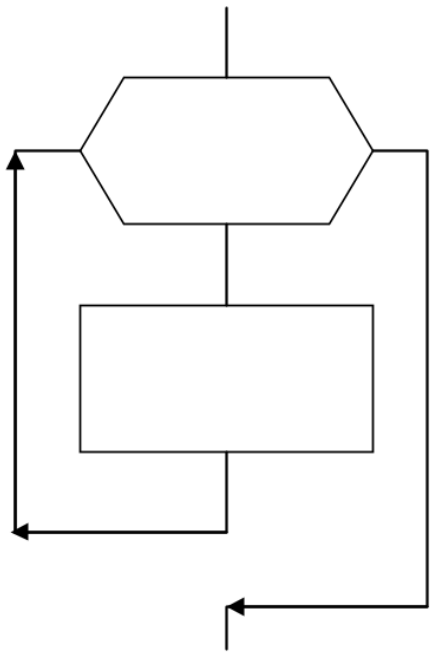
```
B = input()
for i in B:
  print(i, end=' ')
print()
```

Результат работы программы:

```
abcd
a b c d
```

Составление блок-схем программ со структурой цикла for

Рассмотрим правила оформления и графического изображения элементов блок-схем для программ циклической структуры с оператором `for`.

Правило оформления	Графическое изображение
Оператор `for` на блок-схеме отображается минимум двумя блоками – блоком итераций и блоком процесса. Блок итераций цикла – шестиугольник, имеет два входа и два выхода. Тело цикла располагается под блоком итераций. После последнего блока соединительная линия должна подниматься вверх и попадать в один из боковых входов блока итераций. На рисунке справа представлена упрощенная схема отображения цикла с предусловием.	
Обратную стрелку цикла `while` необходимо оформлять так, чтобы линии, идущие вверх и справа налево, заканчивались стрелками.	

Построим блок-схему к задаче 14 (рис. 5.8).

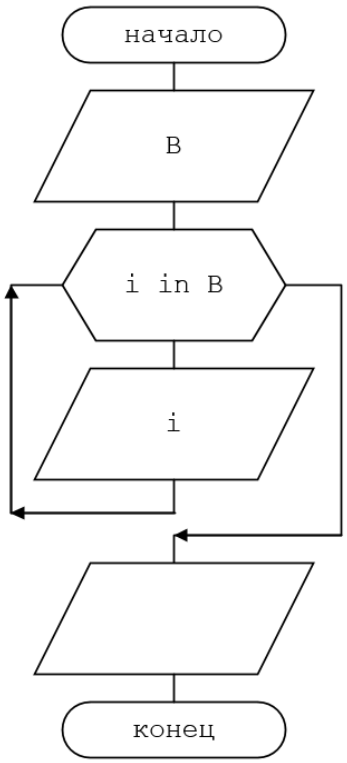


Рис. 5.8. Блок-схема программы задачи 14

В операторе `for` также можно использовать принудительный выход из цикла `break` и оператор `continue`. Оператор `for` может использовать блок `else`, который работает так же, как и в операторе `while`.

Задача 15. Выведите на экран символы строки и сообщение о наличии или отсутствии символа 'a' в строке.

Решение задачи:

```
C = input()
for i in C:
    print(i, end = ' ')
    if i == 'a':
      print("в слове есть символ ""a""")
      break
else:
    print("в слове нет символов ""a""")
```

Результат работы программы:

```
bcdefgh
b c d e f g h в слове нет символов a
```

Блок-схема работы программы задачи 15 представлена на рис. 5.9.

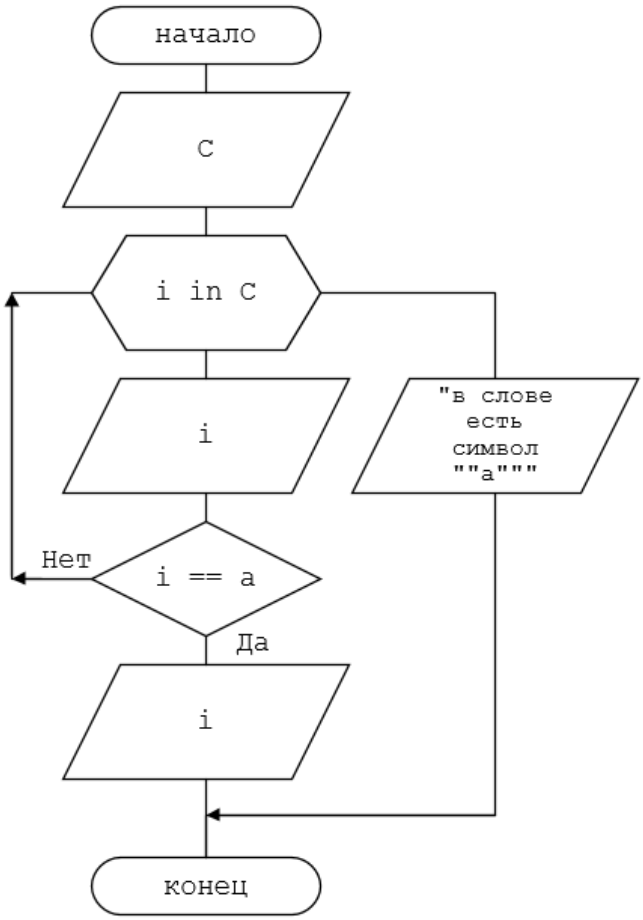


Рис. 5.9. Блок-схема программы задачи 15

Примеры программ со структурой цикла for

Задача 16. Вычислите сумму чисел от 1 до 10.

Решение задачи:

```
sum = 0
for a in range(11):
    sum += a
print("sum =",sum)
```

Результат работы программы:

```
sum = 55
```

Блок-схема работы программы задачи 16 представлена на рис. 5.10.

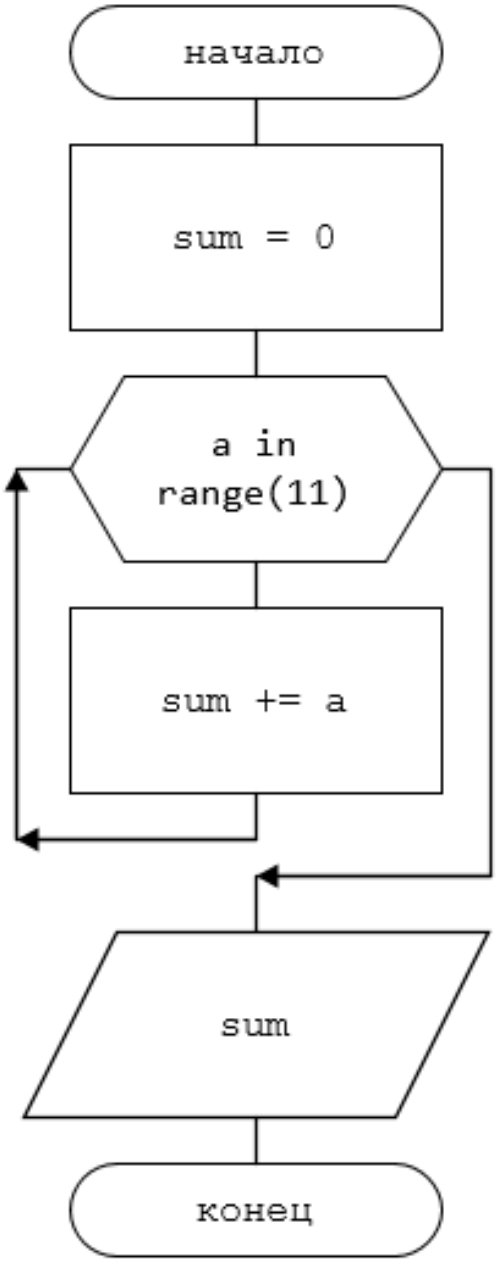


Рис. 5.10. Блок-схема программы задачи 16

Задача 17. Дано число *k*. Выведите на экран числа от 1 до 20, кратные *k* и заканчивающиеся цифрой 5.

Решение задачи:

```
k = int(input('k = '))
for a in range(21):
    if a % k == 0 and a % 10 != 5:
      print(a, end = ' ')
```

Результат работы программы:

```
k = 3
0 3 6 9 12 18
```

Блок-схема работы программы задачи 17 представлена на рис. 5.11.

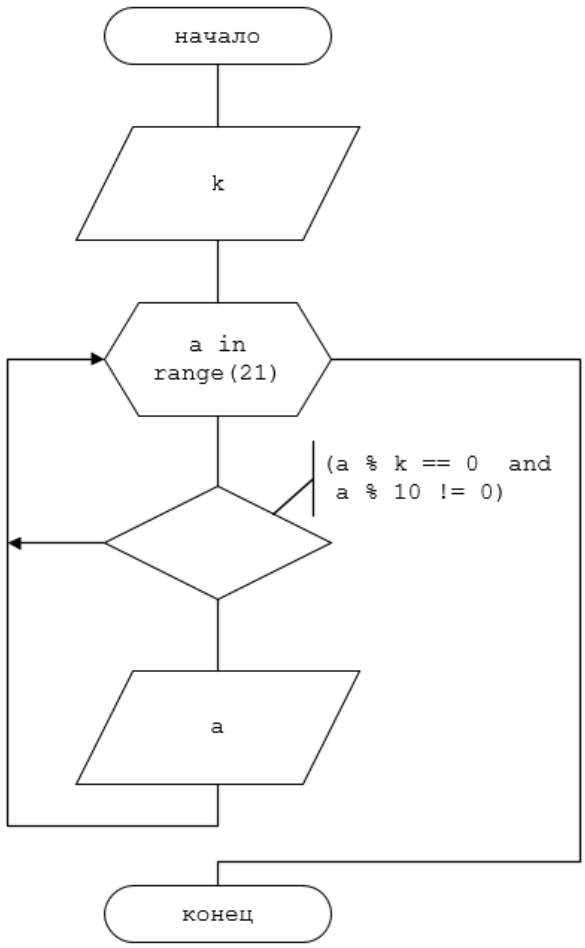


Рис. 5.11. Блок-схема программы задачи 17

Другой вариант решения задачи 17:

```
k = int(input('k = '))
for a in range(21):
    if a % k == 0 and a % 10 != 5:
      print(a, end = ' ')
```

Задача 18. Среди *n* случайных чисел подсчитайте количество положительных, отрицательных и равных 0.

Решение задачи:

```
from random import randint
n = int(input("n = "))
k0, k1 = 0, 0
for a in range(n):
    x = randint(-5,5)
    print(x, end = ' ')
    if x == 0: k0 += 1
    elif x > 0: k1 += 1

print("\nКоличество 0 = ",k0)
print("Положительных = ",k1)
print("Отрицательных = ",n - k0 -k1)
```

Результат работы программы:

```
n = 10
5 4 5 1 -3 -1 0 -1 3 3
Количество 0 = 1
Положительных = 6
Отрицательных = 3
```

Блок-схема работы программы задачи 18 представлена на рис. 5.12.

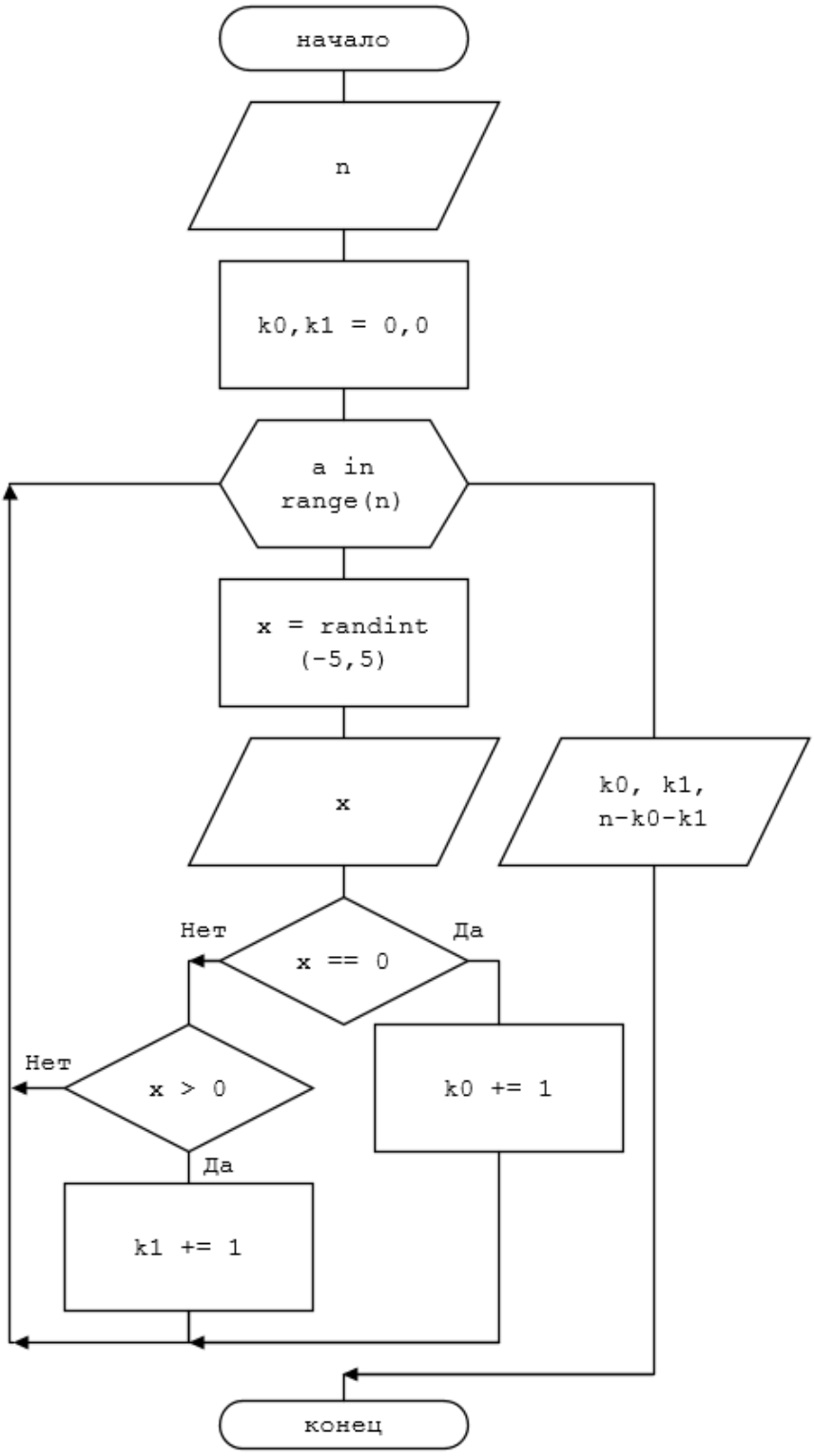


Рис. 5.12. Блок-схема программы задачи 18

Вложенные циклы

Задача 19. Составьте программу вывода таблицы умножения на экран. В программе используйте циклы `while`.

В этой задаче необходимо использовать структуру с вложенным циклом. Внешний цикл перебирает значения *x*, внутренний цикл – значения *y*.

Решение задачи:

```
x = 1                    # начальное значение x
while x <= 10:
    y = 1                # начальное значение y
    while y <= 10:
      print("%4d " % (x*y), end="")
      y += 1             # увеличиваем значение y
    print()              # увеличиваем значение x
    x += 1
```

Задача 20. Составьте программу вывода таблицы умножения на экран. В программе используйте циклы `for`.

```
for i in range(1,10):
  for j in range(1,10):
    print("%3d *%3d = %3d" % (i,j,i*j))
```

Блок-схема работы программы задачи 19 представлена на рис. 5.13, задачи 20 – на рис. 5.14.

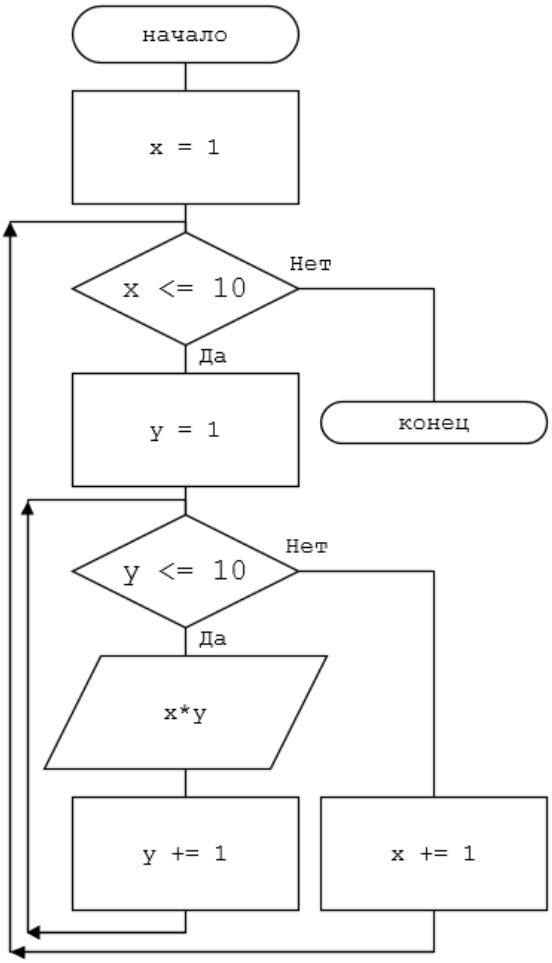


Рис. 5.13. Блок-схема программы задачи 19

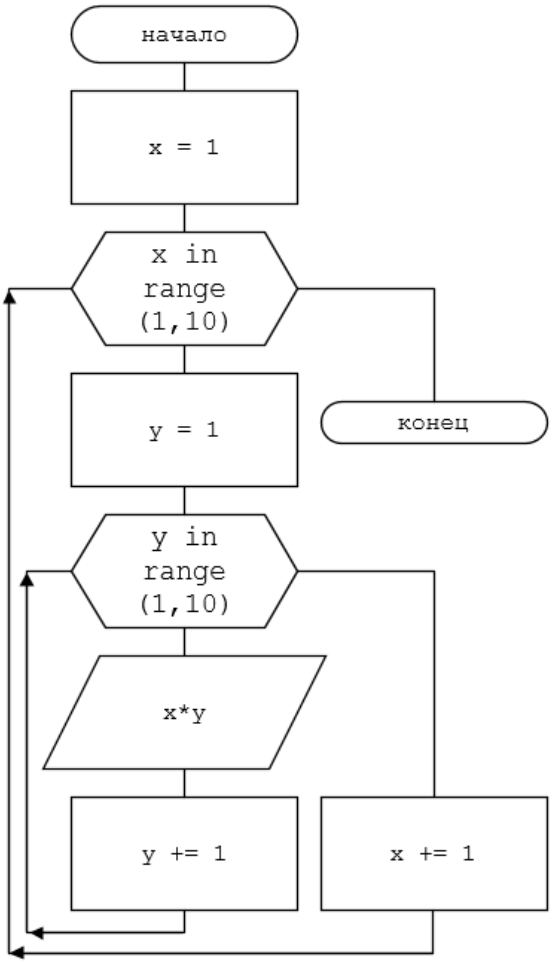


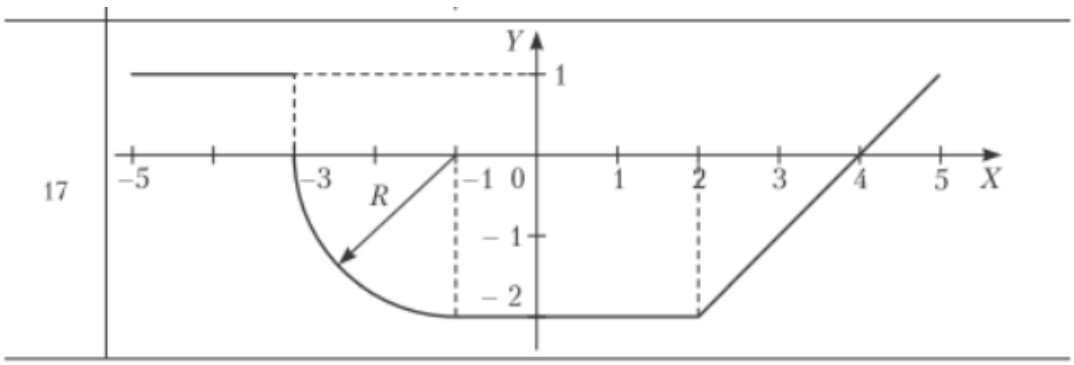
Рис. 5.14. Блок-схема программы задачи 20

Пример выполнения лабораторной работы

Цель работы: научиться составлять программы с циклами, а также блок-схемы к программам циклической структуры.

Задание № 1

Дан график функции:



Решите задачу табулирования функции. График функции задан графически. Значение шага h = 1,0. Решите задачу двумя способами: 1) с использованием цикла `while`; 2) с использованием цикла `for`.

Решение задачи способом 1 (1.1):

```
"""
ЛР № 5. Задание 1.1 Вариант 1.
Программа табулирования функции, заданной графиче-
ски.

Функция y(x), заданная графически, имеет 4 части. В
каждой части функция вычисляется по соответствующей
формуле. Согласно графику функции:
при -5 <= x <= 3    y = 1
при -3 <= x <= -1
```

$(x+1)^2 + y^2 = 2^2 \Rightarrow y = -\sqrt{4-(x+1)^2}$

```
при -1 < x <= 2    y = -2
при x > 2          y = x - 4
Результат вычисления выводится в форматированном
виде, похожем на таблицу.
Начальное значение x = -5, конечное значение x = 5.
Переменная x изменяется с шагом 0,5.
Для x = -3 выводится два значения y = (1,0), так
как это точка разрыва функции.
По условию задачи используется цикл while.

студент гр. ПРИ-124 И.И. Иванов
29.09.2024
"""

x,h = -5, 0.5
```

```
print("%5s  %5s" %("x","y"))
while x <= 5:
  if x < -3: y = 1
  elif x == -3: y = (1,0)
  elif x <= -1: y = -(4-(x+1)**2)**0.5
  elif x <= 2: y = -2
  else: y = x - 4
  if x == -3:
    print("%7.2f (%.2f,%.2f)" %(x,y[0],y[1]) )
  else:
    print("%7.2f %7.2f" %(x,y) )
      x += h
```

Результаты работы программы задания 1.1 представлены на рис. 5.15, блок-схема – на рис. 5.16.

```
x,h = -5, 0.5
print("%5s   %5s" %("x","y"))
while x <= 5:
    if x < -3: y = 1
    elif x == -3:y = (1,0)
    elif x <= -1:y = -(4-(x+1)**2)**0.5
    elif x <= 2: y = -2
    else: y = x - 4
    if x == -3:
        print("%7.2f (%.2f,%.2f)" %(x,y[0],y[1]) )
    else:
        print("%7.2f %7.2f" %(x,y) )
    x += h
```

```
    x       y
-5.00    1.00
-4.50    1.00
-4.00    1.00
-3.50    1.00
-3.00 (1.00,0.00)
-2.50   -1.32
-2.00   -1.73
-1.50   -1.94
-1.00   -2.00
-0.50   -2.00
 0.00   -2.00
 0.50   -2.00
 1.00   -2.00
 1.50   -2.00
 2.00   -2.00
 2.50   -1.50
 3.00   -1.00
 3.50   -0.50
 4.00    0.00
 4.50    0.50
 5.00    1.00
```

Рис. 5.15. Скриншот кода и результатов работы программы задания 1.1

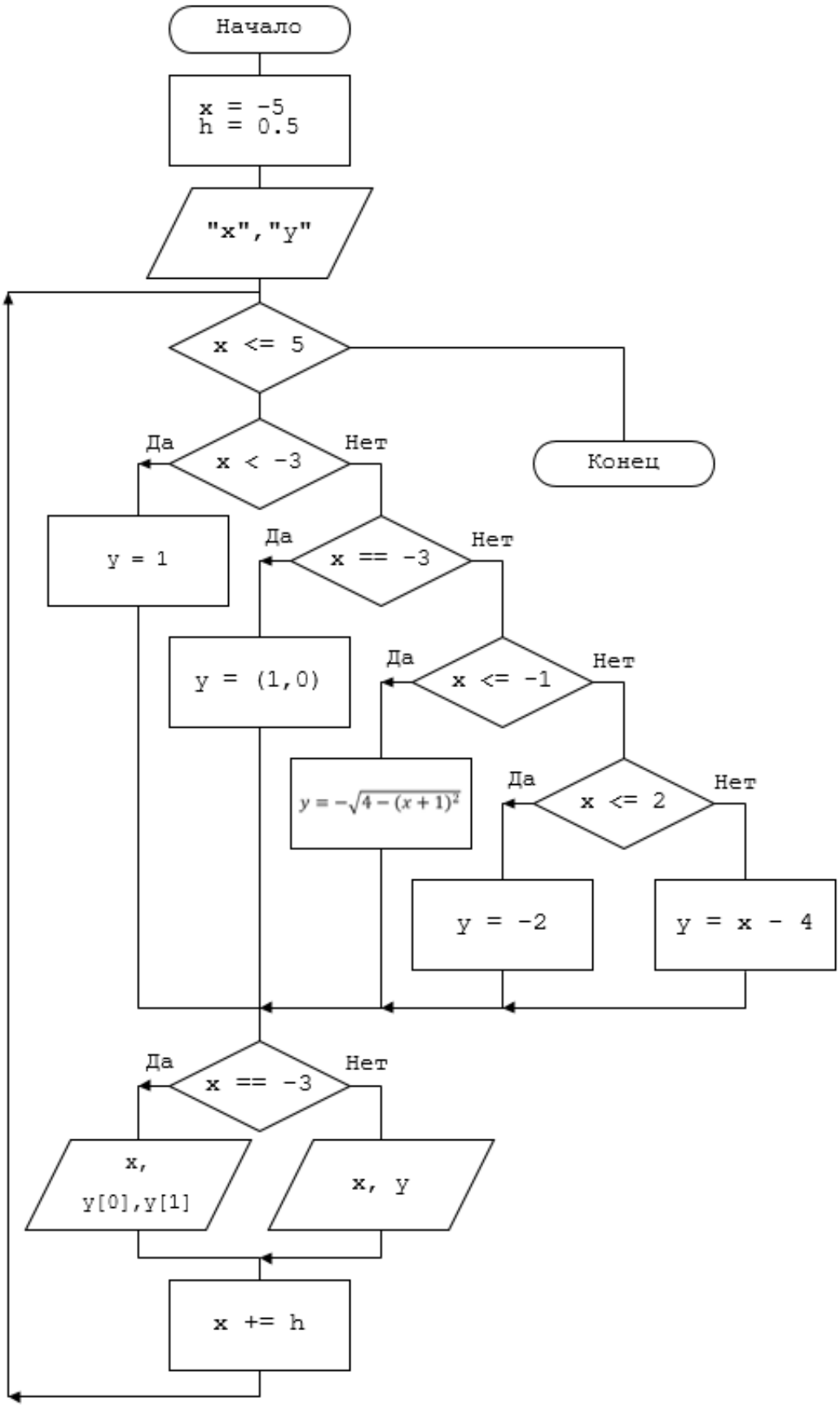


Рис. 5.16. Блок-схема программы задания 1.1

Решение задачи способом 2 (1.2):

```
"""
ЛР № 5. Задание 1.2. Вариант 1.
Программа табулирования функции, заданной графиче-
ски.

Описание задачи представлено в задании.
По условию задачи используется цикл for.
a - начальное значение переменной x,
b – конечное значение переменной x,
h – шаг изменения x,
k – количество точек расчета.

студент гр. ПРИ-124 И.И. Иванов
29.09.2024
"""

a,b,h = -5,5,0.5
k = math.floor((b-a)/h) + 1
print("%5s  %5s" %("x","y"))
for i in range(k):
  x = a + i*h
  if x < -3: y = 1
  elif x == -3: y = (1,0)
  elif x <= -1: y = -(4-(x+1)**2)**0.5
  elif x <= 2: y = -2
  else: y = x - 4
  if x == -3 :
    print("%7.2f (%.2f,%.2f)" %(x,y[0],y[1]) )
  else:
    print("%7.2f %7.2f" %(x,y) )
```

Результаты работы программы задания 1.2 представлены на рис. 5.17, блок-схема – на рис. 5.18.

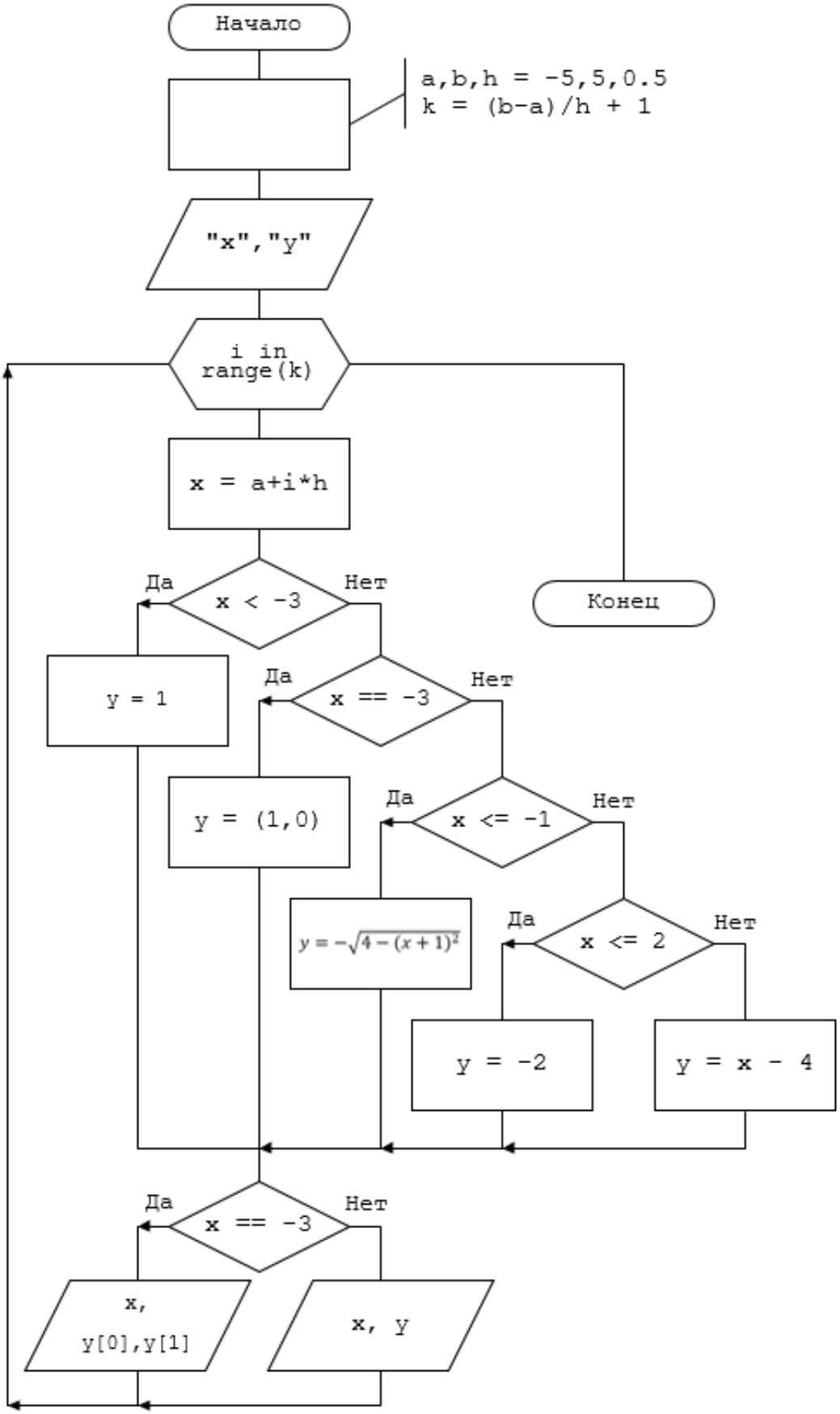


Рис. 5.17. Блок-схема программы задания 1.2

```
a,b,h = -5,5,0.5
k = math.floor((b-a)/h) + 1
print("%5s   %5s" %("x","y"))
for i in range(k):
    x = a + i*h
    if x < -3: y = 1
    elif x == -3: y = (1,0)
    elif x <= -1: y = -(4-(x+1)**2)**0.5
    elif x <= 2:  y = -2
    else: y = x - 4
    if x == -3 :
        print("%7.2f (%.2f,%.2f)" %(x,y[0],y[1]) )
    else:
        print("%7.2f %7.2f" %(x,y) )
```

```
   x        y
-5.00     1.00
-4.50     1.00
-4.00     1.00
-3.50     1.00
-3.00 (1.00,0.00)
-2.50    -1.32
-2.00    -1.73
-1.50    -1.94
-1.00    -2.00
-0.50    -2.00
 0.00    -2.00
 0.50    -2.00
 1.00    -2.00
 1.50    -2.00
 2.00    -2.00
 2.50    -1.50
 3.00    -1.00
 3.50    -0.50
 4.00     0.00
 4.50     0.50
 5.00     1.00
```

Рис. 5.18. Скриншот работающей программы задания 1.2

Вывод: Я научился составлять программы с циклами и блок-схемы к программам циклической структуры.

Тема 6. ЦИКЛЫ. ТЕКСТОВЫЕ ЗАДАЧИ

Решение текстовых задач по теме «Циклы»

Задача 1. Найдите наибольший общий делитель двух чисел $A = 9$ и $B = 24$.

Наибольшим общим делителем (*NOD*) двух целых чисел A и B называется наибольшее число, на которое A и B делятся без остатка.

Наименьшим общим кратным (*NOK*) двух целых чисел A и B называется наименьшее натуральное число, которое делится и на A, и на B.

1-й способ нахождения *NOD* – с помощью вычитания меньшего числа из большего:

A	*B*
9	24 (24 – 9 = 15)
9	15 (15 – 9 = 6)
9	6 (9 – 6 = 3)
3	6 (6 – 3 = 3)
3	3 (*A* = *B*)
NOD = 3	

Из большей переменной вычтите меньшую и запишите на место большей переменной.

Алгоритм выполняется до тех пор, пока переменные не станут равны: *NOD* = *A* = *B*

NOK и *NOD* связаны следующим соотношением:

$c \cdot d = NOK \cdot NOD.$

Поэтому *NOD* можно вычислить по формуле

$NOD = c \cdot d / NOK.$

Для того чтобы вычислить *NOD*, необходимо сохранить введенные значения чисел *c* и *d*.

Решение задачи:

```
c,d = map(int,input().split())
a,b = c,d
while a != b:
  if a > b:
    a = a-b
  else:
    b = b-a
NOD = a
print("NOD = ", NOD)
    print("NOK = ", c*d/NOD)
```

Результат работы программы:

```
9 24
NOD = 3
NOK = 72.0
```

2-й способ нахождения *NOD* – с помощью вычисления остатков от деления числа *A* на число *B*. Этот алгоритм называется алгоритмом Евклида:

A *B*	
9 24 (9%24 = 9)	
24 9 (24%9 = 6)	Сохраните переменную *B* в переменную *A*.
9 6 (9%6 = 3)	Остаток от деления *A* % *B* сохраните в переменную *B*.
6 3 (6%3 = 0)	
3 0	Алгоритм останавливается, когда *B = 0*.
NOD = 3	*NOD = A*

В переменной *A*, согласно алгоритму, значение будет больше, чем в переменной *B*, даже если при вводе *A* < *B*.

Решение задачи:

```
a,b = map(int,input().split())
c,d = a,b
while b !=0 :
  a,b = b, a%b
print("NOD = ", a)
print("NOK = ", c*d/a)
```

Результат работы программы:

```
9 24
NOD = 3
NOK = 72.0
```

Задача 2. Проверьте число *N* (*N* > 1) на простоту.

Число называют простым, если оно делится только на 1 и на себя. Простое число не имеет делителей среди чисел от 2 до *N* – 1.

Переменные задачи 2: N – проверяемое число; i – перебираемые делители числа N.

Проверку начинают с делителя $i = 2$. Необходимо проверить все возможные делители числа N от 2 до $N – 1$. Линейный алгоритм проверки неэффективен при решении данной задачи, так как будет выполнено $(N – 2)$ действия. Достаточно выполнить проверку до $\sqrt{N}$.

Если N делится на i, то число можно представить как произведение двух множителей: $N = i \cdot (N // i)$:
пусть $N = 100$, тогда $N = 100 = 2 \cdot 50 = 4 \cdot 25 = 5 \cdot 20 = 10 \cdot 10$.

Далее множители меняются местами, поэтому проверку заканчивают значением $\sqrt{N}$.

В алгоритме оператор `break` заканчивает цикл `while`, если найден делитель числа N, и число N определяем как составное.

Если делитель числа N не найден и цикл завершился без принудительного выхода `break`, то по завершении цикла выполняется блок `else`. Таким образом, выводится сообщение, что число N простое.

Решение задачи:

```
N = int(input())
i = 2
while i <= N**(1/2):
    if N % i == 0:
      print("%3d - число составное" % N)
      break
    i += 1
else: print("%3d - число простое" % N)
```

Результат работы программы:

```
21
 21 - число составное

23
 23 - число простое
```

Упростим алгоритм и выведем сообщение только в том случае, если N простое число.

Задача 2а. Проверьте число N ($N > 1$) на простоту.

Решение задачи:

```
N = int(input())
i = 2
while i <= N**(1/2):
    if N % i == 0:
      break
    i += 1
else: print(N," - число простое ")
```

Результат работы программы:

```
21
```

```
23
23 - число простое
```

Блок-схема работы программы задачи 2а представлена на рис. 6.1.

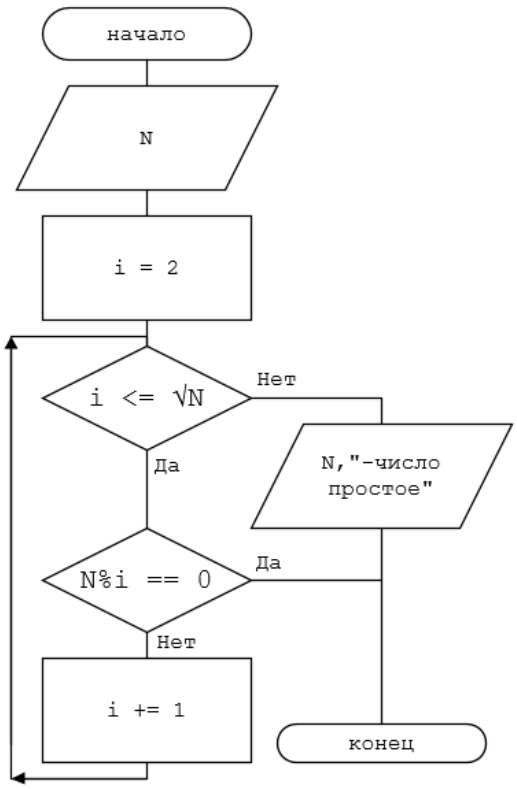


Рис. 6.1. Блок-схема программы 2а

Задача 3. Выведите все простые числа, которые меньше *B*.

Воспользуемся предыдущим алгоритмом и переберем значения *N* от 2 до *B*. На экран будут выводиться только значения *N*, которые являются простыми числами.

Решение задачи:

```
B = int(input())
N = 2
while N <= B:
  i = 2
  while i <= N**(1/2):
    if N % i == 0:
      break
    i += 1
  else: print(N, end=" ")
  N += 1
```

Результат работы программы:

```
30
2 3 5 7 11 13 17 19 23 29
```

Блок-схема работы программы задачи 3 представлена на рис. 6.2.

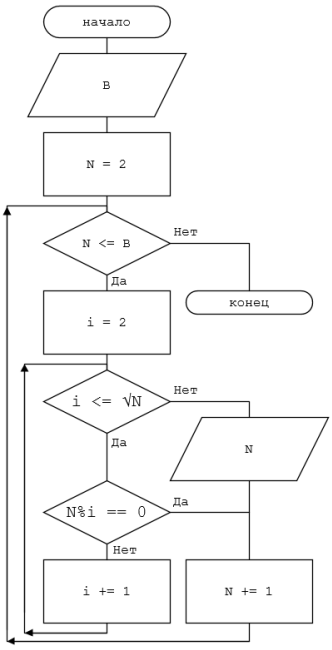


Рис. 6.2. Блок-схема программы задачи 3

Задача 4. Выведите на экран все делители числа N.

Переберите все возможные делители числа *N* с помощью переменной *i*. Начальное значение *i* = 1. Внутри цикла добавьте делитель *i* в переменную *s*1 и делитель *N//i* в переменную *s*2. В переменную *s*1 делители попадают в прямом порядке, в переменную *s*2 – в обратном. Последнее значение *i* возможно, когда $i = \sqrt{N}$ или $i \geq \sqrt{N}$. После выхода из цикла необходимо проверить, является ли число *N* полным квадратом, и добавить значение множителя в переменную *s*1.

Решение задачи:

```
N = int(input("N = "))
if N <= 0:
  print("Введите число > 0")
else:
  i = 1
  s1,s2 = '',''
  while i < N**(1/2):
    if N % i == 0:
      s1 = s1 + str(i)+' '
      s2 = str(N//i)+' '+s2
    i += 1
  if i*i == N: s1 = s1 + str(i)+' '
  s = s1 + s2
  print(s)
```

Результат работы программы:

```
N = 100
1 2 4 5 10 20 25 50 100
```

```
0
Введите число > 0
```

Рекуррентные вычисления

Рекуррентные вычисления – это такие вычисления, которые при вычислении текущего элемента последовательности используют предыдущий или предыдущие элементы последовательности и/или их номера.

Задача 5. Вычислите факториал числа N!

Факториал числа $N! = 1 \cdot 2 \cdot 3 \cdot \ldots \cdot (N-1) \cdot N$, $0! = 1$.

$1! = 1$

$2! = 1 \cdot 2 = 1! \cdot 2$

$3! = 1 \cdot 2 \cdot 3 = 2! \cdot 3$

$4! = 1 \cdot 2 \cdot 3 \cdot 4 = 3! \cdot 4$ и т. д.

Составьте выражение в общем виде: $N! = (N-1)! \cdot N$.

Такое выражение называется рекуррентным соотношением. Соотношение позволит вычислить текущее значение через предыдущее.

Соотношение в коде программы $F = F \cdot k$: слева от знака «равно» – новое значение, справа от знака «равно» – предыдущее значение факториала.

Решение задачи:

```
N = int(input("N = "))
F = 1 if N >= 0 else None
for k in range(1,N):
    F = F * k
print(F)
```

Результат работы программы:

```
N = -1
None
```

```
N = 0
1
```

```
N = 5
24
```

Задача 6. Вычислите сумму N факториалов: $Sum = 1! + 2! + 3! + \ldots + N!$

Каждое слагаемое – факториал числа, который можно вычислить, используя один цикл. Всю сумму можно вычислить, используя второй цикл.

Решение задачи:

```
N = int(input("N = "))
Sum = 1 if N > 0 else None

for P in range(1,N+1):      # Цикл вычисления суммы
    F = 1
    for k in range(1,P+1):  # Цикл вычисления
        F = F * k           # очередного слагаемого,
    Sum += F                # равного P!
print(Sum)
```

Вычислим результат алгоритма без программы:

$N = 1$ $Sum = 1! = 1$

$N = 2$ $Sum = 1! + 2! = 1 + 2 = 3$

$N = 3$ $Sum = 1! + 2! + 3! = 1 + 2 + 6 = 9$

Результат работы программы:

```
N = 0
None
```

```
N = 3
9
```

Такое решение неэффективно, так как алгоритм составлен из двух циклов. При использовании рекуррентного метода алгоритм будет выполнен быстрее, за один цикл.

Решение задачи:

```
N = int(input("N = "))
Sum = 0 if N > 0 else None
F = 1
```

```
for k in range(1,N+1):  # Цикл вычисления суммы
    F = F * k           # Вычисление F!=(F-1)!*k
    Sum += F            # Добавляем F! к сумме
print(Sum)
```

Результат работы программы:

```
N = 1
1
```

```
N = 3
9
```

Задача 7 (вычисление чисел Фибоначчи). Выведите на экран *N*-е число Фибоначчи.

Ряд Фибоначчи 1, 1, 2, 3, 5, 8, 13, 21, 34, 55…

Первые два числа равны 1, каждое следующее число равно сумме двух предыдущих чисел.

Решение задачи:

```
N = int(input())
if N <= 2 :
print("F(%d)= %d" % (N,1))
else:
F1,F2 = 1,1
i = 3
while i <= N :
    F1,F2 = F2, F1+F2
    i += 1
print("F(%d)= %d" % (N,F2))
```

Результат работы программы:

```
N = 1
F(1)= 1
```

```
N = 2
F(2)= 1
```

```
N = 10
F(10) = 55
```

Задача 8. Вычислите простейшую рациональную функцию $f(z) = \frac{1}{1-z}$ с точностью ε = 10^{-5}, используя рекуррентную функцию расчета для значений *z*, таких что $|z| < 1$.

Формула $f(z) = \frac{1}{1-z}$ может быть разложена в ряд Тейлора:

$$\frac{1}{1-z} = \sum_{n=0}^{\infty} z^n = 1 + z + z^2 + z^3 + \cdots$$

Как известно, это формула суммы бесконечной убывающей геометрической прогрессии. Функция $f(z) = \frac{1}{1-z}$ определена для всех *z*, кроме *z* = 1. Ряд Тейлора сходится только при условии $|z| < 1$.

Для вычисления слагаемых используйте переменную *d*. Заметим, что каждое следующее слагаемое больше предыдущего в *z* раз, поэтому рекуррентное соотношение будет следующим: *d* = *d* · *z*, начальное значение *d* = 1. Процесс добавления слагаемых к сумме заканчивается в тот момент, когда очередное слагаемое станет меньше заданной точности.

Программа вычисляет точное значение *f* и приближенное значение `f_model`. Также на экран необходимо вывести абсолютную погрешность `abs_tol =|f-f_model|`.

Решение задачи:

```
z = float(input("z = "))

f_model = 0
if abs(z) < 1:

eps = 1E-5

    d = 1
    while d > 1E-5:
      f_model += d
      d = d * z

f = 1/(1-z)
    abs_tol = abs(f-f_model)
```

```
print("Точное значение %.7f" %f)
    print("Приближенное значение %.7f" %f_model)
    print("Абсолютная погрешность %.7f" %abs_tol)
```

Результат работы программы:

```
z = 0.5
Точное значение 2.0000000
Приближенное значение 1.9999847
Абсолютная погрешность 0.0000153
```

Задача 9. Вычислите значение функции y = arcctg(x) с помощью стандартных функций и путем разложения в ряд Тейлора с заданной точностью $\varepsilon = 10^{-5}$. Выведите и примените рекуррентное соотношение для общей формулы слагаемого или его части. Разложение функции Y в ряд для $x \leq |1|$ выполняется по формуле

$$y = \text{arcctg}(x) = \frac{\pi}{2} - \text{arctg}(x) = \frac{\pi}{2} - \sum_{n=1}^{\infty} \frac{(-1)^{n-1}}{2n-1} x^{2n-1}$$

$$y = \text{arcctg}(x) = \frac{\pi}{2} - x + \frac{x^3}{3} - \frac{x^5}{5} + \frac{x^7}{7} + \cdots$$

Переменные задачи 9:

y – точное значение функции: $y = \text{arcctg}(x) = \frac{\pi}{2} - \text{arctg}(x)$;

y_model – приближенное значение функции; рассчитывается, как сумма элементов ряда Тейлора.

Заметим, что сохранять слагаемое в виде дроби не следует, так как знаменатель дроби в каждом слагаемом уникальный. Знаменатель зависит от номера слагаемого и вычисляется как ($2n$ – 1);

d – числитель слагаемого. Для вычисления числителя применяют рекуррентное соотношение $d = -d \cdot x \cdot x$. Умножение на (–1) необходимо, потому что сумма является знакочередующейся и знаки элементов, следующих друг за другом, меняются. Начальное значение $d = -x$.

Добавлять к сумме следует только те слагаемые, которые по модулю меньше заданной точности $\varepsilon = 10^{-5}$. Слагаемое вычисляется как $\frac{d}{2n-1}$. Начальное значение суммы $\frac{\pi}{2}$.

Решение задачи:

```
import math

eps = 1e-5
x = float(input("x = "))

y = math.pi/2 - math.atan(x)
y_model = math.pi/2
d,n = -x,1
while abs(d/(2*n-1)) > eps:
y_model += d/(2*n-1)
n += 1
d = - d *x*x

abs_tol = abs(y-y_model)
print("Точное значение %.7f" %y)
print("Приближенное значение %.7f" %y_model)
print("Абсолютная погрешность %.7f" %abs_tol)
```

Результат работы программы:

```
x = 0
Точное значение 1.5707963
Приближенное значение 1.5707963
Абсолютная погрешность 0.0000000
```

```
x = 0.5
Точное значение 1.1071487
Приближенное значение 1.1071564
Абсолютная погрешность 0.0000077
```

Задача 10. Вычислите значение функции *y* = arcctg*(x)* с помощью стандартных функций и путем разложения в ряд Тейлора с заданной точностью ε = 10^{-5} при изменении аргумента в указанном диапазоне [*a,b*]=[0, 1] с шагом $h = \frac{b-a}{10}$. Выведите и примените рекуррентное соотношение для общей формулы слагаемого или его части. Разложение функции *Y* в ряд выполняется по формуле для $x \leq |1|$

$$y = \text{arcctg}(x) = \frac{\pi}{2} - \text{arctg}(x) = \frac{\pi}{2} - \sum_{n=1}^{\infty} \frac{(-1)^{n-1}}{2n-1} x^{2n-1}.$$

Воспользуемся решением задачи 9 для вычисления *y*(*x*). В задаче 10 необходимы несколько значений *y*(*x*) для *x* на интервале [*a,b*] = [0,1] с шагом $h = \frac{b-a}{10}$. Для вычисления шага диапазон изменения *x* делится на 10 шагов или отрезков. Поэтому точек расчета *y*(*x*) – 11. Для вывода результатов добавим порядковый номер *k* для каждой рассчитываемой точки (*x, y*).

Решение задачи:

```
import math
eps = 1e-5
a,b = 0, 1
h = (b-a)/10
k = 1

print("%3s%5s%9s%16s%12s" % ('№ ','x', 'y',\
'y_model', 'abs_tol'))

x = a
while x <= b:
    y = math.pi/2 - math.atan(x)
    y_model = math.pi/2
    d,n = -x,1
    while abs(d/(2*n-1)) > eps:
```

```
        y_model += d/(2*n-1)
        n += 1
        d = - d *x*x
    abs_tol = abs(y-y_model)
    print("%3d %.3f %.8f %.8f %.8f" %(k,\
x, y, y_model, abs_tol))

    k += 1
    x += h
```

Результат работы программы:

№	x	y	y_model	abs_tol
1	0.000	1.57079633	1.57079633	0.00000000
2	0.100	1.47112767	1.47112966	0.00000199
3	0.200	1.37340077	1.37339899	0.00000177
4	0.300	1.27933953	1.27934157	0.00000204
5	0.400	1.19028995	1.19028659	0.00000336
6	0.500	1.10714872	1.10715644	0.00000772
7	0.600	1.03037683	1.03038436	0.00000754
8	0.700	0.96007036	0.96007405	0.00000369
9	0.800	0.89605538	0.89605975	0.00000437
10	0.900	0.83798123	0.83797678	0.00000444
11	1.000	0.78539816	0.78540316	0.00000500

Цепные дроби

Конечное или бесконечное выражение вида:

$$a_1 + \cfrac{1}{a_2 + \cfrac{1}{a_3 + \cfrac{1}{a_4 + \cdots \cfrac{1}{a_{n-1} + \cfrac{1}{a_n}}}}},$$

где a_1 – целое число; все другие a_n – натуральные числа, называется цепной дробью.

Для вычисления цепной дроби также необходимо использовать рекуррентную функцию.

Задача 11. Вычислите значение цепной дроби (см. с. 127), где $a_1 = 1$, $a_2 = 2$ … $a_n = 10$. Для решения вычислим начальное значение $d = 1/a_n$, т. е. $d = 1/10$. Рекуррентное соотношение $d = 1/(a_{n-1} + d)$. Заканчивается вычисление, когда первое слагаемое становится равно $n = 1$. Для получения ответа выполните последнее действие: добавьте к дроби d последнее слагаемое $n = 1$.

Решение задачи (способ 1):

```
d = 1/10
n = 9
while n >= 2:
  d = 1/(n + d)
  n -= 1
d = n + d
print(round(d,5))
```

Результат работы программы:

```
1.43313
```

Эту задачу можно решить иначе, если в качестве начального значения взять $d = a_n$, т. е. $d = 10$. Рекуррентное соотношение $d = a_{n-1} + 1/d$. Заканчивается вычисление, когда первое слагаемое становится равно 0. В этом способе сразу получаем ответ в переменной d.

Решение задачи (способ 2):

```
d = 10
n = 9
while n >= 1:
  d = n + 1/d
  n -= 1
print(round(d,5))
```

Результат работы программы:

```
1.43313
```

Задача 12. Вычислите с точностью ε = 10^{-5} цепную дробь вида

$$[1;\ 1,1,1\ldots] = 1 + \cfrac{1}{1 + \cfrac{1}{1 + \cfrac{1}{1 + \cdots}}}.$$

Это число называют золотым сечением. Оно представляет собой положительный корень уравнения $1 + \frac{1}{x} = x$ или $x^2 - x + 1 = 0$ $x = \frac{1+\sqrt{5}}{2} \approx 1{,}618$.

Золотое сечение – это такое отношение частей к целому, когда большая часть относится к меньшей так же, как целая – к большей.

Начальное значение знаменателя последней дроби *d* = 1. Рекуррентное соотношение *d* = 1 + 1/*d*. Переменная *d* по окончании алгоритма будет содержать результат работы алгоритма. Заканчивается вычисление, когда разница между текущим значением результата *d* и следующим результатом (1 + 1/*d*) станет меньше заданной точности `eps`.

Решение задачи:

```
eps = 1E-5

d = 1
while abs(d - (1 + 1/d)) >= eps:
    d = 1 + 1/d
print(round(d,5))
```

Результат работы программы:

```
1.61804
```

Задача 13. Вычислите с точностью ε = 10^{-5} цепную дробь вида

$$\frac{\pi}{4} = \cfrac{1}{1 + \cfrac{1^2}{2 + \cfrac{3^2}{2 + \cfrac{5^2}{2 + \cfrac{7^2}{\ldots \cfrac{K^2}{\ldots}}}}}}$$

Решение задачи сводится к задаче поиска значения K^2, которое должно стать последним в цепной дроби, так, чтобы значение цепной дроби совпало с точным значением $\frac{\pi}{4}$ минимум в четырех разрядах после запятой, а в пятом разряде отличия не превысили 1, так как точность приближенного ответа $\varepsilon = 10^{-5}$.

Начальное значение $d = 1$, рекуррентное соотношение $d = 2 + k \cdot k/d$. Значения k только нечетные, поэтому k += 2. После вычисления частей дроби с первым слагаемым 2 выполните последнее вычисление со слагаемым 1: $d = 1/(1 + 1/d)$. Начальное значение k принимаем равным K.

Вычисление цепной дроби начинается с последней дроби выражения. Сравнить результат вычисления с точным значением $\frac{\pi}{4}$ будет возможно только после вычисления цепной дроби. Поэтому необходимо вычислить дробь с разными значениями K, подбирая K до тех пор, пока не получим результат с заданной точностью.

Значение K вычисляется методом подбора, иначе нужно добавлять третий цикл по переменной K. Путем подбора $K = 50001$.

Решение задачи:

```
import math
eps = 1E-5

d, K = 1, 49997
f = math.pi/4
while abs(f - d) > eps:
    d = 1
    k = K
    while k > 1:
      d = 2+k*k/d
      k -= 2
    d = 1/(1 + 1/d)
    print("Приближенное значение" ,round(d,7),\
" K =",K)
  K += 2
print("Точное значение   ", round(f,7))
print("Абсолютная ошибка  ", round(abs(f-d),7))
```

Результат работы программы:

```
Приближенное значение 0.7853882 K = 49997
Приближенное значение 0.7854082 K = 49999
Приближенное значение 0.7853882 K = 50001
Точное значение    0.7853982
Абсолютная ошибка   1e-05
```

Пример выполнения лабораторной работы

Цель работы: получить навыки решения задач с использованием циклических конструкций на языке программирования Python.

Задание № 1

У гусей и кроликов вместе 64 лапы. Сколько может быть кроликов и гусей? (указать все сочетания)

Решение задачи:

```
"""
ЛР № 6. Задание 1. Вариант 1.
Программа вычисления всех сочетаний по заданию и их количества с использованием циклов.

По условию задачи дано N = 32 количество лап у всех кроликов и гусей. У кроликов 4 лапы, у гусей по 2 лапы. Формально необходимо подобрать значения r и g в выражении N = r + g, где r – число лап кроликов, или значение, кратное 4, g – число лап гусей, или число, кратное 2. Перебираем значения g от 0 до N включительно с шагом 2, вычисляя второе слагаемое r. Если r кратно 4, то нужное сочетание найдено – g//2 и r//4.

студент гр. ПРИ-124 И.И. Иванов
6.10.2024
"""

N = 32
print( ' g  r ')
for g in range(0, N+1, 2):
r = N - g
if r % 4 == 0:
    print("%3d %3d" %(g//2, r//4))
```

Результаты работы программы задания 1 представлены на рис. 6.3, блок-схема – на рис. 6.4.

```
N = 32
print( ' g    r ')
for g in range(0, N+1, 2):
    r = N - g
    if r % 4 == 0:
        print("%3d %3d" %(g//2, r//4))
```

```
 g   r
 0   8
 2   7
 4   6
 6   5
 8   4
10   3
12   2
14   1
16   0
```

Рис. 6.3. Скриншот кода и результатов работы программы задания 1

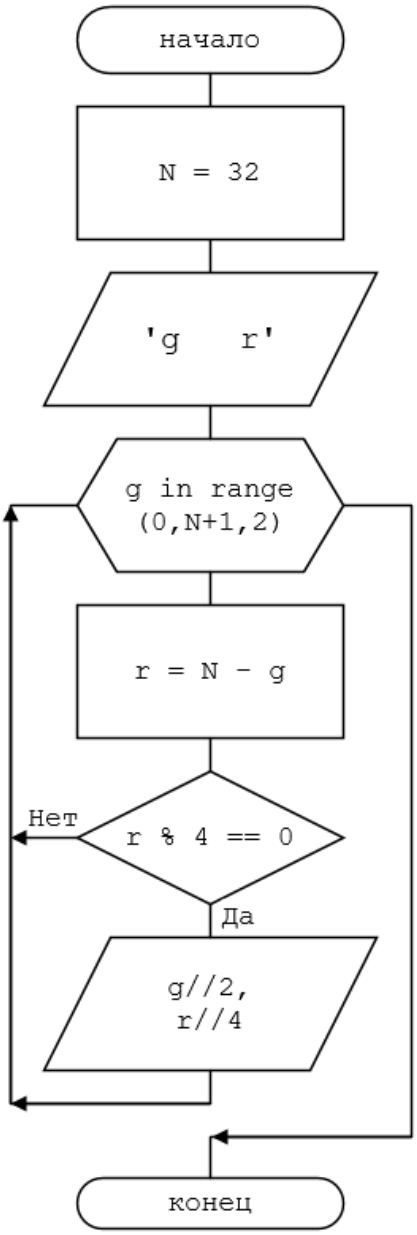


Рис. 6.4. Блок-схема программы задания 1

Задание № 2

Вычислите, используя рекуррентный метод, значение *y* по формуле

$$y = \sqrt{3 + \sqrt{3 + \sqrt{3 + \cdots \sqrt{3 + \sqrt{3}}}}},$$

где количество радикалов *n* = 10.

Решение задачи:

```
"""
ЛР № 6. Задание 2. Вариант 1.
Программа вычисления значения с применением рекур-
рентного метода.

Программа вычисляет значение функции y =
√(3+√(3+√(3+ … √3))), где количество радикалов n =
10. Рекуррентное соотношение d = √(3+d), начальное
значение d = 0.

студент гр. ПРИ-124 И.И. Иванов
6.10.2024
"""

d = 0
n = 10
while n > 0:
    d = (3+d)**0.5
    n -= 1
print(round(d,3))
```

Результаты работы программы задания 2 представлены на рис. 6.5, блок-схема – на рис. 6.6.

```
d = 0
n = 10
while n > 0:
    d = (3+d)**0.5
    n -= 1
print(round(d,7))
```

```
2.302775
```

Рис. 6.5. Скриншот кода и результатов работы программы задания 2

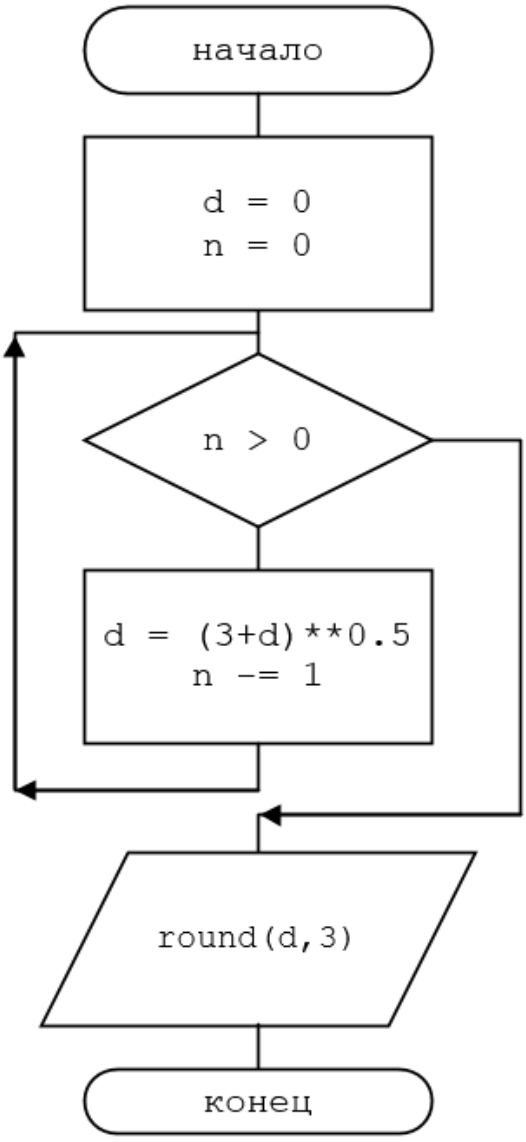


Рис. 6.6. Блок-схема программы задания 2

Вывод: Я получил навыки решения задач с использованием циклических конструкций на языке программирования Python.

Тема 7. ГРАФИКА

Установка библиотеки Pillow

Для работы с графикой необходимо установить библиотеку Pillow с помощью системы `pip` в командной строке.

`Pip` – это система управления пакетами, которая используется для установки и управления программными пакетами, написанными на Python.

Для установки библиотеки Pillow необходимо в командной строке написать команду:

```
pip install pillow
или
pip3 install pillow
и нажать клавишу enter.
```

Подключение библиотеки Pillow в коде программы

Для подключения библиотеки Pillow в коде программы используются объект «полотно» – `Image` и инструмент рисования по полотну – `ImageDraw` из пакета `Pillow`.

```
from PIL import Image, ImageDraw
```

Создаем объект – картинку размером 700×700 пикселей (система цветопередачи – RGB, цвет полотна – белый).

```
img = Image.new('RGB', (700, 700), 'white')
```

Создаем инструмент для рисования по полотну:

```
draw = ImageDraw.Draw(img)
```

В графическом режиме можно обратиться к каждому пикселю. Пиксель с координатами (0,0) находится в верхнем левом углу. Нумерация пикселей по горизонтали – слева направо, по вертикали – сверху вниз.

Рисование с помощью графических примитивов

Методы графических примитивов:

1) `draw.line()`

Рисует линию по указанным координатам, не обязательно прямую. Также можно указать цвет и толщину линии.

Пример:

```
draw.line([(0,0),(400,200),(200,500)], 'red', 5)
```

2) `draw.rectangle()`

Рисует прямоугольник, для которого надо указать координаты двух противоположных вершин. Также можно указать цвет заливки, цвет линий и толщину линии.

Пример:

```
draw.rectangle([(50,300),(100,350)],\
'magenta','green',5)
```

3)`draw.ellipse()`

Рисует круг или овал. Метод вписывает окружность или овал в прямоугольник, указанный координатами противоположных вершин. Также можно указать цвет заливки, цвет линий и толщину линии.

Пример:

```
draw.ellipse([(50,100),(200,200)],'green',None,0)
```

4)`draw.arc()`

Рисует дугу по заданным координатам, с указанием начала и конца, с определенной заливкой и толщиной. Дуга – это часть окружности, ограниченная двумя линиями, выходящими из центра окружности. Линии образуют угол с нулевым вектором. Нулевой вектор лежит на оси абсцисс системы координат с центром системы в центре окружности. Направление нулевого вектора совпадает с направлением оси абсцисс. Нулевой вектор соответствует углу, равному 0°. Углы отсчитываются против часовой стрелки.

Пример:

```
draw.arc([(200,100),(300,200)],0,270,'green',2)
```

5) `draw.chord()`

Дуга, ограниченная окружностью и отрезком, соединяющим ее концы; параметры те же самые, что и у `arc()`.

Пример:

```
draw.chord([(250,150),(350,250)],0,90,\
None,'blue',2)
```

6) `draw.point()`

Рисует точку с координатами; можно указать цвет.

Пример:

```
draw.point([(250,250)],'magenta')
```

7) `draw.polygon()`

Рисует полигон с определенным количеством вершин. Полигон – это замкнутая область, поэтому последняя точка в списке будет соединена с первой точкой.

Пример:

```
draw.polygon([(400,400),(400,500),(500,500)],\
'magenta',None)
```

8) `img.show()`

Отобразит картинку в формате .png в стандартном средстве просмотра картинок.

Задача 1. Нарисуйте картинку с использованием графических примитивов.

Решение задачи:

```
from PIL import Image, ImageDraw

img = Image.new("RGB",(500,500),"white")
draw = ImageDraw.Draw(img)
```

```
draw.line([(0,0),(400,200),(200,500)], 'red', 5)
draw.arc([(200,100),(300,200)],0,270,'green',2)
draw.chord([(250,150),(350,250)],0,90, \
None,'blue',2)
draw.ellipse([(50,100),(200,200)],'green',None,0)
draw.point([(250,250)],'magenta')
draw.rectangle([(50,300),(100,350)],\
'magenta','green',5)
draw.rectangle([(200,100),(300,200)],None,\
'blue',1)
draw.polygon([(400,400),(400,500),(500,500)],\
'magenta',None)

img.show()
```

Результат работы программы задачи 1 представлен на рис. 7.1.

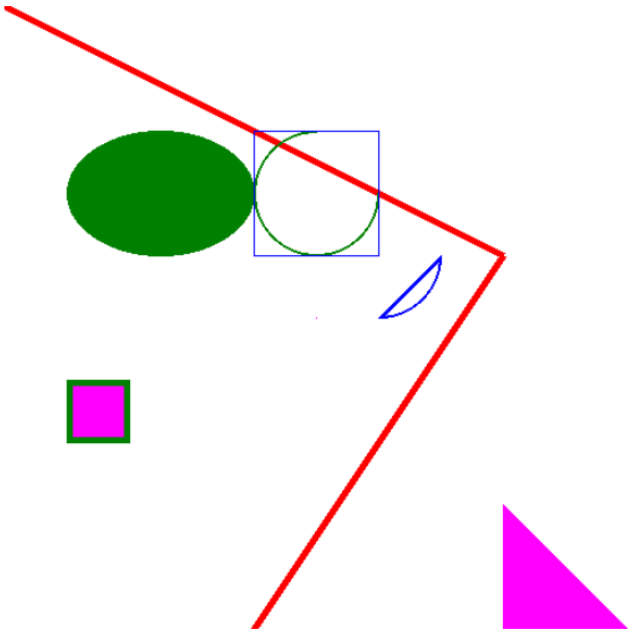


Рис. 7.1. Скриншот результата работы программы задачи 1

Задача 2. С помощью графических инструментов библиотеки Pillow нарисуйте спираль Фибоначчи.

Спираль Фибоначчи составлена из дуг, являющихся четвертями окружностей, вписанных в квадраты с длинами сторон, равными числам последовательности Фибоначчи.

Спираль Фибоначчи представлена на рис. 7.2. Прямоугольники располагаются относительно друг друга по часовой стрелке. К первому квадрату добавляется квадрат справа, затем к этим двум квадратам со сторонами 1 и 1 добавляется квадрат со стороной 2 снизу. Затем добавляется квадрат со стороной 3 слева, затем – квадрат со стороной 5 сверху и т. д.

Таким образом, правило добавления квадратов следующее: сначала добавляется квадрат справа, затем по очереди – снизу, слева, сверху, далее эта цепочка повторяется.

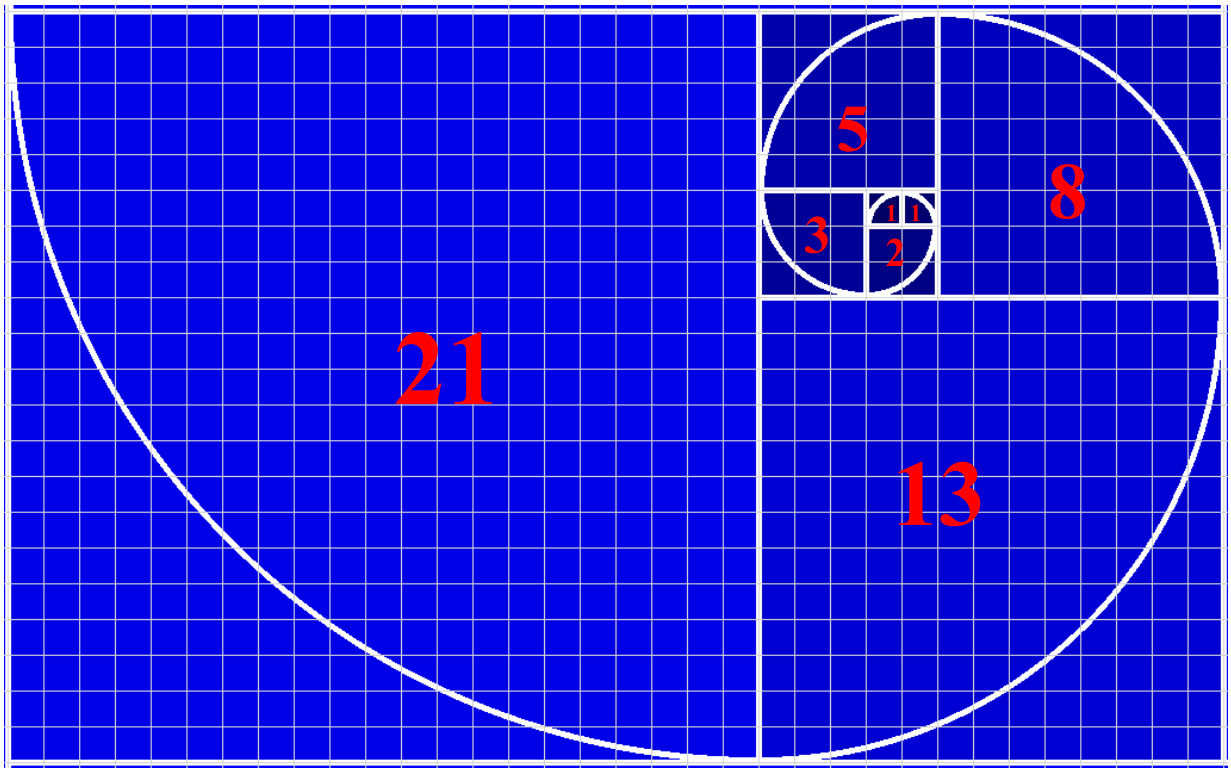


Рис. 7.2. Спираль Фибоначчи

Правило можно модифицировать и начать добавлять квадраты, например, снизу. Тогда цепочка будет выглядеть так: снизу, слева, сверху, справа и т. д.

Можно спираль закручивать в сторону, противоположную часовой стрелке. Параметры алгоритма рисования спирали Фибоначчи: стартовая сторона расположения добавленного квадрата и направление добавления новых квадратов дадут разные варианты картинки.

Переменные, которые используются в алгоритме:

$x0$, $y0$ – центр координат полотна;

m – масштаб;

x, y – координаты верхнего левого угла текущего прямоугольника;

*F, F*1 – числа последовательности Фибоначчи;

`color` – насыщенность синего цвета.

Решение задачи:

```
from PIL import Image, ImageDraw
img = Image.new("RGB",(1500,1500),"white")
draw = ImageDraw.Draw(img)
x0, y0 = img.width // 2, img.height // 2
m = 30

x, y = x0,y0
F, F1 = 1, 1
color = 10
draw.rectangle( \
    [(x,y),(x+F*m,y+F*m)],(0,0,color),"white",3)
draw.arc( \
    [(x,y),(x+2*F*m,y+2*F*m)],180,270,"white",5)

x += F*m

color += 10
draw.rectangle( \
    [(x,y),(x+F*m,y+F*m)],(0,0,color),"white",3)
draw.arc( \
    [(x-F*m,y),(x+F*m,y+2*F*m)],270,360,"white",5)

x, y = x + F*m, y + F*m

while F < 100 :
    color += 20
    F, F1 = F + F1, F
    draw.rectangle( \
    [(x-F*m, y),(x,y+F*m)],(0,0,color),"white",3)
    draw.arc( \
    [(x-2*F*m, y-F*m),(x,y+F*m)],0,90,"white",5)
```

```
    x, y = x - F*m, y + F*m

    color += 20
    F, F1 = F + F1, F
    draw.rectangle( \
    [(x-F*m, y-F*m),(x,y)],(0,0,color),"white",3)
    draw.arc( \
    [(x-F*m, y-2*F*m),(x+F*m,y)],90,180,"white",5)
    x, y = x - F*m, y - F*m

    color += 20
    F, F1 = F+F1, F
    draw.rectangle( \
    [(x, y-F*m),(x+F*m,y)],(0,0,color),"white",3)
    draw.arc( \
    [(x,y-F*m),(x+2*F*m,y+F*m)],180,270,"white",5)

    x, y = x + F*m, y - F*m

    color += 20
    F, F1 = F + F1, F
    draw.rectangle( \
    [(x, y),(x+F*m,y+F*m)],(0,0,color),"white",3)
    draw.arc( \
    [(x-F*m,y),(x+F*m,y+2*F*m)],270,360,"white",5)

    x, y = x + F*m, y + F*m

img.show()
```

В алгоритме квадраты рисуются методом `draw.rectangle()`. Квадрат задается противоположными точками вершин квадрата таким образом, что первая точка располагается на полотне рисования левее и выше, чем вторая точка. Поэтому вершины квадрата при вызове метода меняются соответствующим образом:

```
draw.rectangle( [(x,y),(x+F*m,y+F*m)] …)
draw.rectangle( [(x-F*m, y),(x,y+F*m)]…)
draw.rectangle( [(x-F*m, y-F*m),(x,y)]…)
```

Дуги рисуются методом `draw.arc()`. Они соединяют противоположные вершины текущего квадрата, поэтому их необходимо вписывать в квадрат со стороной 2**F*, чтобы получить четверть дуги:

```
draw.arc([(x,y),(x+2*F*m,y+2*F*m)]…)
draw.arc([(x-F*m,y),(x+F*m,y+2*F*m)]…)  и т. д.
```

В методе `draw.arc()` углы меняются в зависимости от вида дуги:

```
draw.arc( …180,270…)
draw.arc( …270,360…)
draw.arc( …0,90…)
draw.arc( …90,180…)  и т. д.
```

Результат работы программы задачи 2 представлен на рис. 7.3.

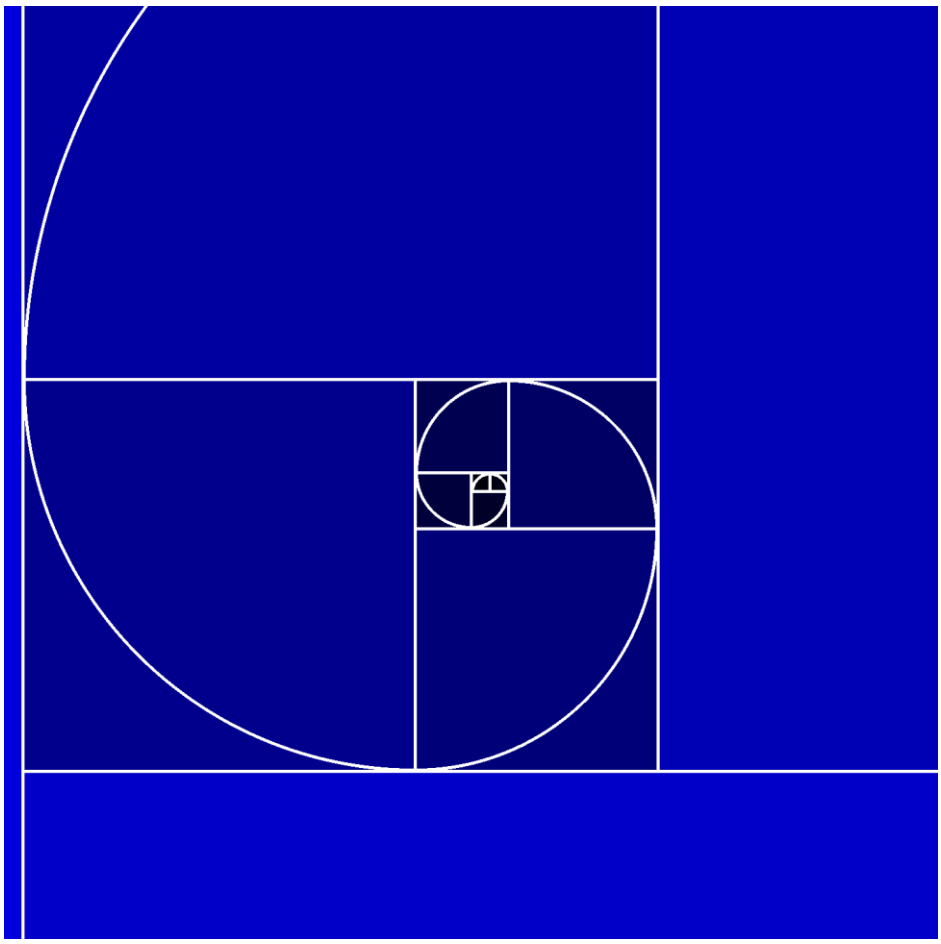


Рис. 7.3. Скриншот результата работы программы задачи 2

В программе цвет заливки фона квадратов сохраняется в формате *RGB* в виде кортежа из трех значений: интенсивность красного цвета – *R*, интенсивность зеленого цвета – *G*, интенсивность синего цвета – *B*.

Например:

```
draw.rectangle(…(0,0,color)…)
```

В переменной `color` сохраняется интенсивность синего фона. Для каждого следующего квадрата цвет фона будет более насыщен синим цветом, так как переменная `color` увеличивается.

Самостоятельно поэкспериментируйте с насыщенностью цвета фона. Варианты изменения параметров вызова метода `rectangle()`:

1) `draw.rectangle(…(0,color,color)…)`
2) `draw.rectangle(…(color,0,color)…)`
3) `draw.rectangle(…(color,color,color)…)`
4) `draw.rectangle(…(0,color,256-color)…)` и т. д.

Рисование графика функции

Задача 3. Напишите программу, которая с помощью библиотеки Pillow создаст файл картинки в формате `png` и нарисует график функции $y = \sin(x)$.

В алгоритме рисования функции воспользуйтесь алгоритмом табулирования значений функции, в котором получатся точки $(x, y(x))$, принадлежащие функции. Данные точки необходимо соединить между собой прямыми линиями, получив кусочно-линейную функцию. При малом шаге, т. е. большом количестве рассчитанных точек, такая подмена будет не видна.

Итак, для алгоритма отображения функции необходимо рассчитать координаты начальной точки (x, y). Далее на каждой итерации вычислите новую точку $(x1, y1)$. Имея координаты двух точек, соедините точки методом `draw.line()`. Сохраните координаты $(x1, y1)$ как точку (x, y), алгоритм переходит к следующей итерации. Процесс продолжается до тех пор, пока не закончится диапазон изменения аргумента функции x.

Другие переменные в программе следующие:
h – шаг изменения аргумента функции;
mX, mY – масштаб по осям *OX* и *OY*;
*x*0, *y*0 – центр координат.

Наконечники стрелок осей координат рисуют методом `draw.polygon()`.

Решение задачи:

```
from math import sin, pi
from PIL import Image, ImageDraw, ImageFont
imgWidth, imgHeight = 700, 300
img = Image.new('RGB', (imgWidth, imgHeight),\
        'white')
draw = ImageDraw.Draw(img)

h = pi/12   # шаг
mX,mY = 50,50   # масштаб по осям OX и OY

# центр координат, точка (0,0)
x0,y0 = imgWidth // 2, imgHeight // 2

# ось OY
draw.line([(x0,0),(x0,imgHeight)],"black",3)
# ось OX
draw.line([(0,y0),(imgWidth,y0)],"black",3)

# стрелки на осях
draw.polygon([(x0,0),(x0+5,20),(x0-5,20)],'black')
draw.polygon([(imgWidth,y0),(imgWidth-20,y0-5),\
              (imgWidth-20,y0+5)],'black')

# Раздел рисования графика функции
# начальная точка (x,y)
x, x1, y = -2*pi, -2*pi, sin(-2*pi)
```

```
while x <= 2*pi:

    x1 += h          # новая точка (x1,y1)
    y1 = sin(x1)

    # рисуем линию между точками (x,y)и(x1,y1)
    draw.line((round(x*mX+x0),round(y0-y*mY),\
          round(x1*mX+x0),round(y0-y1*mY)),\
          fill = 'blue', width = 3)

    x, y = x1,y1     # переходим к последней
                     # точке (x1,y1)
img.show()
```

Результат работы программы задачи 3 представлен на рис. 7.4.

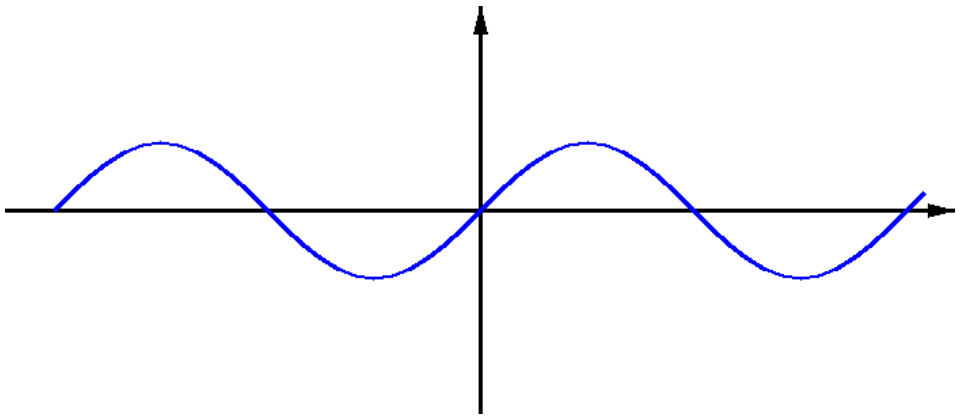


Рис. 7.4. Скриншот результата работы программы задачи 3

Для того чтобы добавить сетку координатной плоскости и подписи на шкале *OX*, необходимо дополнить программу до блока «Раздел рисования графика функции» кодом, представленным в решении задачи 3*. Код добавляется до основного алгоритма рисования графика для того, чтобы график рисовался поверх сетки, а не наоборот.

Для добавления текста на полотно используется метод `draw.text()`:

```
draw.text((x,y), str(text), color, font),
```

где (*x, y*) – куда добавляем, `str(text)` – что добавляем, `color` – цвет текста, `font` – шрифт текста.

Шрифт текста устанавливается методом `ImageFont.truetype()`: `ImageFont.truetype(path, size)`,
где `path` – путь к папке, где хранится файл шрифта, `size` – размер шрифта.

Можно использовать шрифты из системной папки Windows:

```
fnt = ImageFont.truetype(\
            r"C:\Windows\Fonts\sylfaen.ttf", 18)
```

Символ *r* перед началом строки `r"C:\Windows…"` означает, что в строке будут игнорироваться все входящие экранированные символы `'\n', '\t'` и т. д.

Решение задачи 3*:

```
# линии вертикальной сетки
for j in range( img.height // (2*mY)):
    draw.line([(0,y0+j*mY),(img.width,y0+j*mY)],\
         'lightgrey' )
    draw.line([(0,y0-j*mY),(img.width,y0-j*mY)],\
         'lightgrey' )

# подключение шрифта из системной папки
fnt = ImageFont.truetype(\
         r"C:\Windows\Fonts\sylfaen.ttf", 18)

# Вывод значения 0 в центре координат
draw.text((x0,y0+10), "0", "grey", font = fnt)

# Подпись оси OX и шкала

x, key = 1, 1

while x <= 2*pi:

    match key : # Выбор подписи
      case 1: str = 'pi/3'
      case 2: str = '2pi/3'
```

```
  case 3: str = 'pi'
  case 4: str = '4pi/3'
  case 5: str = '5pi/3'
  case _: str = '2'

# линии горизонтальной сетки
draw.line([(x0+x*mX,0),(x0+x*mX,img.height)],\
     'lightgrey')
draw.line([(x0-x*mX,0),(x0-x*mX,img.height)],
     'lightgrey')

# черточки для шкалы оси OX
draw.line([(x0+x*mX,y0+5), (x0+x*mX,y0-5)],
     "black",3)
draw.line([(x0-x*mX,y0+5), (x0-x*mX,y0-5)],
     "black",3)

# подписи значений шкалы OX
draw.text((x0+x*mX-5*len(str),y0+10), str,
     "grey", font = fnt)
draw.text((x0-x*mX-5*len(str),y0+10), "-"+str,
     "grey",font = fnt)

key += 1

x += 4*h
```

Результат работы программы задачи 3* представлен на рис. 7.5.

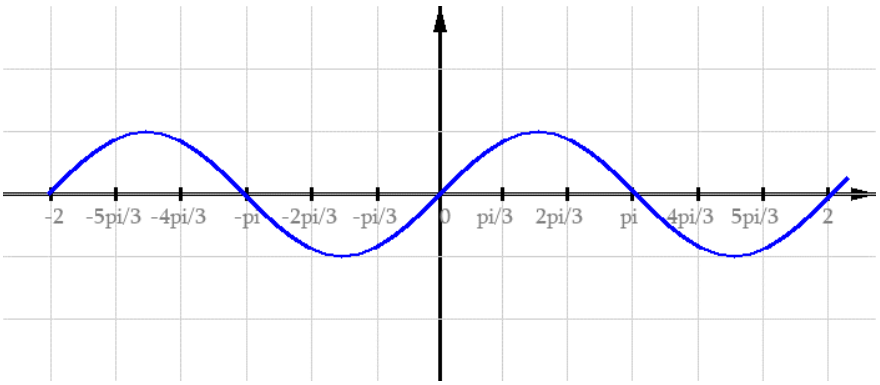


*Рис. 7.5. Скриншот результата работы программы задачи 3**

Пример выполнения лабораторной работы

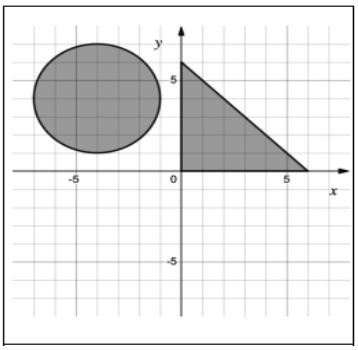


Рис. 7.6. Закрашенная область

Цель работы: получить навыки создания изображений на языке Python с использованием библиотеки Pillow.

Задание № 1

Напишите программу, которая с помощью библиотеки Pillow создаст файл картинки в формате png и нарисует закрашенную область (см. лаб. работу к теме 3, задание 2), рис. 7.6.

Решение задачи:

```
"""
ЛР № 7. Задание 1. Вариант 1.
Программа рисования картинки заданной области.

Программа просматривает все точки координатной
плоскости. Если точка принадлежит заданной обла-
сти, то точка заполняется нужным цветом.

студент гр. ПРИ-124 И.И. Иванов
13.10.2024
"""

from math import sin, pi
from PIL import Image, ImageDraw
imgWidth,imgHeight = 700,700
img = Image.new('RGB', (imgWidth, imgHeight),
'white')
draw = ImageDraw.Draw(img)

x0,y0,h = imgWidth // 2, imgHeight // 2, pi/12
draw.line([(x0,0),(x0, imgHeight)], "black", 3)
draw.line([(0,y0),(imgWidth,y0)], "black", 3)
```

```
mX,mY = 30,30
for x in range(-x0,x0):
  for y in range(-y0,y0):
    if (x/mX+4)**2 +(y/mY-4)**2 <= 3**2 \
      or x/mX>0 and y/mY>0 and y/mY<-x/mX+5:
      draw.point([round(x+x0),round(y0-y)],\
                 "blue")
img.show()
```

Результаты работы программы задания 1 представлены на рис. 7.7.

```
from math import sin, pi
from PIL import Image, ImageDraw
imgWidth,imgHeight = 700,700
img = Image.new('RGB', (imgWidth, imgHeight),\
                'white')
draw = ImageDraw.Draw(img)

x0,y0,h = imgWidth // 2, imgHeight // 2, pi/12
draw.line([(x0,0),(x0, imgHeight)], "black", 3)
draw.line([(0,y0),(imgWidth,y0)], "black", 3)

mX,mY = 30,30
for x in range(-x0,x0):
    for y in range(-y0,y0):
        if (x/mX+4)**2 +(y/mY-4)**2 <= 3**2 or \
            x/mX>0 and y/mY>0 and y/mY<-x/mX+5:
            draw.point(\
                [round(x+x0),round(y0-y)],"blue")
img.show()
```

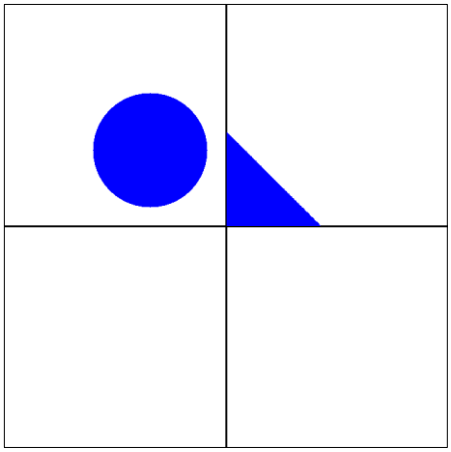


Рис. 7.7. Скриншот кода и результатов работы программы задания 1

Задание № 2

Напишите программу, которая с помощью библиотеки Pillow создаст файл картинки в формате png и нарисует картинку в соответствии с вариантом задания. Для рисования картинки используйте стандартные примитивы (прямоугольники, окружности, полигоны и т. д.).

Решение задачи:

```
"""
ЛР № 7. Задание 2. Вариант 1.
Программа рисования картинки.
```

```
Нарисована картинка с использованием графических
примитивов библиотеки Pillow.

студент гр. ПРИ-124 И.И. Иванов
13.10.2024
"""

from PIL import Image, ImageDraw

img = Image.new('RGB', (700, 700), 'white')
draw = ImageDraw.Draw(img)

draw.polygon([(300, 300),(300, 325),(375,\
325),(375, 300)], 'green', None)
draw.line([(375,325),(390,325)],'black',1)
draw.polygon([(382, 287),(382, 325),(435,\
325),(435, 287)], 'green', None)
draw.ellipse((315,325,340,350),None,'black',6)
draw.ellipse((385,325,410,350),None,'black',6)
draw.ellipse((425,303,432,316),'yellow','yell',2)
draw.polygon([(320,249),(320,299),\
(340,299),(340,249)],'#FFD700','black')
draw.polygon([(340,249),(340,299),\
(360,299),(360,249)], '#FFD700', 'black')
draw.polygon([(382,260),(382,287),\
(397,287),(397,260)],(240, 255, 255),'black')
draw.polygon([(397,260),(397,287),\
(412,287),(412,260)],(240, 255, 255),'black')
draw.polygon([(412,260),(412,287),(435,287)],\
'white', 'black')
draw.line([(335,274),(335,278)], 'black', 2)
draw.line([(344,274),(344,278)], 'black', 2)

img.show()
```

Результаты работы программы задания 2 представлены на рис. 7.8.

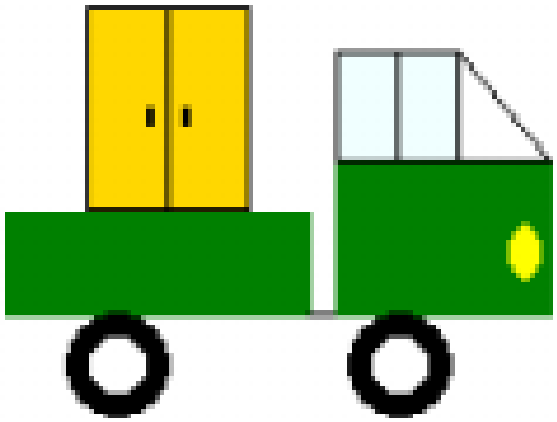


Рис. 7.8. Скриншот результатов работы программы задания 2

Вывод: Я получил навыки создания изображений на языке Python с использованием библиотеки Pillow.

Тема 8. СПИСКИ

Объявление списка

Список – это индексируемый набор элементов. Количество элементов в списке называется длиной списка.

Объявление пустого списка:

```
A = []
```

В пустом списке нет элементов. Длина пустого списка равна 0.

Объявление списка из пяти элементов, равных 0:

```
A = [0]*5
```

Объявление списка:

```
A = ["First",0,1,2,-1.5,'a',[1,2,3],[],True]
```

В списке *A* девять элементов, т. е. длина списка равна 9. В одном списке одновременно могут находиться элементы разных типов: строки, целые числа, вещественные числа, логические переменные. Также элементами списков могут быть списки, в том числе пустые.

К каждому элементу списка можно обратиться по индексу. Индексация элементов начинается с 0.

Обращение к первому элементу списка:

```
print(" A[0] = ", A[0])
A[0] = First

print(" A[8] = ", A[8])
A[8] = True
```

Последний элемент списка имеет индекс на 1 меньше, чем длина списка.

Список является объектом класса `list` в Python. У класса `list` есть множество готовых методов работы со списками. С этими методами можно познакомиться в пункте «Встроенные функции и методы списков».

Длину списка возвращает один из встроенных методов – метод `len()`:

```
print("Длина списка -",len(A))
9
```

Функция `append()` добавляет элемент в конец списка. Например, добавим в список *A* элемент, равный 1:

```
A.append(1)
print("Длина списка -",len(A))
10
```

Длина списка *A* увеличилась до 10, так как был добавлен один элемент.

Ввод/вывод элементов списка

Вывод списка на экран в квадратных скобках:

```
print(A)
['First', 0, 1, 2, -1.5, 'a', [1, 2, 3], [], True]
```

Вывод списка на экран без квадратных скобок:

```
print(*A)
First 0 1 2 -1.5 a [1, 2, 3] [] True
```

Перебор всех элементов списка и вывод каждого элемента с новой строки:

```
for i in range( len(A) ):
  print(A[i], end = '_')
First_0_1_2_-1.5_a_[1, 2, 3]_[]_True_
```

Ввод списка с клавиатуры. Окончание ввода – 'end':

```
C = []
print('Ввод элементов списка. Окончание ввода - слово end')
x = input("Введите элемент списка ")
while x != 'end':
    C.append( int(x) )
    x = input("Введите элемент списка ")
print(*C)
Ввод элементов списка.
Окончание ввода - слово end
Введите элемент списка 1
Введите элемент списка 2
Введите элемент списка 3
Введите элемент списка end
1 2 3
```

Ввод списка с клавиатуры. Окончание ввода – 'end':

```
D = []
while True:
    x = input()
    if x == 'end' : break
    D.append(int(x))
```

```
print(*D)
1
2
end
1 2
```

Ввод списка из пяти элементов с клавиатуры:

```
n = 5
B =[0]*5
for i in range( len(B) ):
  B[i]=int(input("Введите значение B[%1d] " % i))
print(*B)
Введите значение B[0] 1
Введите значение B[1] 2
Введите значение B[2] 3
Введите значение B[3] 4
Введите значение B[4] 5
1 2 3 4 5
```

Ввод списка в строку выполняется с помощью `map()`:

```
print("Введите элементы списка в одной строке \че-
рез пробел")
D = map(int, input().split() )
print("Список:", *D)
Введите элементы списка в одной строке через пробел
1 2 3 4 5
Список: 1 2 3 4 5
```

Поэлементный вывод элементов списка:

```
B =[1,2,3,4]
for i in range( len(B) ):
    print("B[%1d] = %d " % (i,B[i]))
B[0] = 1
B[1] = 2
B[2] = 3
B[3] = 4
```

Одновременный перебор индексов и элементов списка в цикле `for` можно выполнить с использованием конструкции `enumerate()`:

```
B = [1,2,3,4]
for i, val in enumerate(B):
    print("B[%1d] = %d " % (i, val))
```

```
B[0] = 1
B[1] = 2
B[2] = 3
B[3] = 4
```

`Enumerate()` позволяет читать кортеж из пары чисел – индекс элемента и значение элемента с данным индексом. Такой способ очень удобен в циклах, где по алгоритму требуется обращение и к индексу, и к значению элемента одновременно.

Перебор элементов списка без использования индексов тоже возможен в Python:

```
B = [1,2,3,4]
for x in B:
    print(x, end = ' ')
```

```
1 2 3 4
```

В последнем примере в переменную *x* попадают элементы из списка *B*. Как элементы итерируемого объекта элементы списка можно прочитать без обращения к ним по индексу. Но такой способ позволит только прочитать элементы, но не изменять их, сохраняя новые значения.

Заполнение списка случайными числами

Заполнение списка из *N* элементов случайными числами:

```
import random
N = int(input("Количество чисел: "))
H = [0]*N
for i in range(N):
    H[i] = random.randint(0,100)
print(*H)
```

```
Количество чисел: 3
18 100 85
```

Копирование списков

Неверное копирование списков:

```
A =[1,2,3,4,5,6,7,8,9]
B = A
B[0] = 100
print("A =", A)
print("B =", B)
```

```
A = [100, 2, 3, 4, 5, 6, 7, 8, 9]
B = [100, 2, 3, 4, 5, 6, 7, 8, 9]
```

Копирование списков оператором $B = A$ невозможно, так как в этом случае будет скопирована только ссылка на список.

Верное копирование списков:

```
A =[1,2,3,4,5,6,7,8,9]
B = A[:]
B[0] = 100
print("A =", A)
print("B =", B)
```

```
A = [1, 2, 3, 4, 5, 6, 7, 8, 9]
B = [100, 2, 3, 4, 5, 6, 7, 8, 9]
```

Решение элементарных задач со списками

Задача 1. Вычислите сумму элементов списка [1, 2, 3, 4, 5].

Решение задачи:

```
C = [1,2,3,4,5]
sum = 0                    #Начальное значение sum
for i in range(len(C)):
    sum += C[i]
print(sum)
```

Результат работы программы:

```
15
```

Задача 2. Вычислите среднее значение элементов списка [1, 2, 3, 4, 5].

Решение задачи:

```
D = [1,2,3,4,5]
sum = 0                    #Начальное значение sum
for i in range(len(D)):
    sum += D[i]
print("Average = %7.3f" % (sum/len(D)) )
```

Результат работы программы:

```
Average =  3.000
```

Задача 3. Найдите сумму четных элементов списка X = [1, 2, 3, 4, –3, 6, 7, 8]. Перебор элементов осуществите по индексу. Выведите четные элементы на экран.

Решение задачи:

```
X =[1,2,3,4,-3,6,7,8]
N = len(X)
summ = 0
for i in range(N):
    if X[i] % 2 == 0 :
      print(X[i], end = ' ')
      summ += X[i]
print()
print("Сумма =",summ)
```

Результат работы программы:

```
2 4 6 8
Сумма = 20
```

Задача 4. Найдите сумму четных элементов списка X = [1, 2, 3, 4, –3, 6, 7, 8]. Перебор элементов осуществите без использования индекса. Выведите четные элементы на экран.

Решение задачи:

```
X =[1,2,3,4,-3,6,7,8]
summ = 0
for x in X:
    if x % 2 == 0 :
      print(x, end = ' ')
      summ += x
print()
print(summ)
```

Результат работы программы:

```
2 4 6 8
20
```

Задача 5. Инвертируйте элементы списка. Перебор элементов осуществите по индексу.

Решение задачи:

```
A = [0,1,2,3,4,5,6,7,8,9]
for i in range(len(A)):
    A[i] *= -1
print(*A)
```

Результат работы программы:

```
0 -1 -2 -3 -4 -5 -6 -7 -8 -9
```

Задача 6. Инвертируйте элементы списка. Перебор элементов осуществите без индекса.

Решение задачи:

```
A = [0,1,2,3,4,5,6,7,8,9]
for x in A:
    x *= -1
print(*A)
```

Результат работы программы:

```
0 1 2 3 4 5 6 7 8 9
```

Как видно из последнего примера, задача 6 не выполнена, элементы списка *A* не инвертированы. Это произошло потому, что перебор элементов без индекса возможен только для чтения элементов. В переменную *x* элементы попадают из списка *A*, но из переменной *x* значения обратно в список *A* не возвращаются.

Для того чтобы решить задачу 6, необходимо сохранить инвертированные элементы в новом списке.

Решение задачи 6*:

```
A = [0,1,2,3,4,5,6,7,8,9]
B = []
for x in A:
    B.append(-x)
print(*A)
print(*B)
```

Результат работы программы задачи 6*:

```
0 1 2 3 4 5 6 7 8 9
0 -1 -2 -3 -4 -5 -6 -7 -8 -9
```

Конкатенация списков

Задача 7. Выполните конкатенацию двух списков [1, 2, 3] и [4, 5, 6].

Решение задачи:

```
B = [1,2,3]+[4,5,6]
print(*B)
```

Результат работы программы:

```
1 2 3 4 5 6
```

Рассмотренная ранее операция объявления списка из *N* элементов, заполненного 0, также выполняется как операция конкатенации нескольких списков:

```
N = 5
A = [0]*N
print(A)
B = [0]+[0]+[0]+[0]+[0]
```

```
print(B)
[0, 0, 0, 0, 0]
[0, 0, 0, 0, 0]
```

Ввод элементов списка также можно осуществить с помощью конкатенации:

```
N = 3
C = []
for i in range(N):
  C += [ int(input()) ]
print(*C)
1
2
3
1 2 3
```

Решение задач со списками

Задача 8. Четные элементы запишите без изменений, нечетные – инвертируйте. Порядок вхождения элементов не меняйте.

Решение задачи:

```
A = [0,1,2,3,4,5,6,7,8,9]
J = []
for x in A :
    if x % 2==0 :
      J.append(x)
    else:
      J.append(-x)
print(*J)
```

Результат работы программы:

```
0 -1 2 -3 4 -5 6 -7 8 -9
```

Задача 9. Четные элементы запишите без изменений, нечетные – инвертируйте. Порядок вхождения элементов следующий: сначала выбираем все четные элементы, затем – все нечетные.

Решение задачи:

```
A = [0,1,2,3,4,5,6,7,8,9]
I,J = [],[]
for x in A :
    if x % 2==0 :
      I.append(x)
    else:
      J.append(-x)
I += J
print(*I)
```

Результат работы программы:

```
0 2 4 6 8 -1 -3 -5 -7 -9
```

Задача 10. Заполните список из пяти элементов случайными вещественными числами, значения которых варьируются от –1 до 1.

Решение задачи:

```
import random
N = 5
M = []
for i in range(N):
    M.append(round(random.uniform(-1,1),3))
print(M)
```

Результат работы программы:

```
[0.829, 0.507, 0.702, 0.294, -0.722]
```

Задача 11. Найдите максимум в списке из пяти случайных вещественных чисел, значения которых варьируются от –1 до 1.

В задаче необходимо сохранить в переменную *max* начальное значение. В качестве начального значения сохраняем *M*[0]. Далее перебираем все элементы списка: если встретился элемент больший, чем *max*, то сохраняем этот элемент в переменную *max*, т. е. обновляем максимум.

Решение задачи:

```
import random
N, M = 5, []
for i in range(N):
  M.append(round(random.uniform(-1,1),3))
print(M)

max = M[0]                # начальное значение max
for x in M:
  if x > max: max = x
print("max =", max)
```

Результат работы программы:

```
[0.903, 0.699, -0.254, 0.082, 0.144]
max = 0.903
```

Задача 12. Найдите второй максимум в списке *M* = [0.903, 0.699, –0.254, 0.082, 0.144].

В задаче необходимо сохранить в переменную *max*1 и *max*2 начальные значения. В качестве начальных значений сохраняем два первых значения списка *M*[0] и *M*[1] таким образом, что больший из них попадает в переменную *max*1, другой – в переменную *max*2. Далее перебираем все элементы списка. При этом могут возникнуть две ситуации:

1) встретился новый первый максимум. Тогда старый максимум сохраняем в переменную *max*2, а в переменную *max*1 сохраняем текущий элемент списка, который является новым первым максимумом;

2) встретился элемент, меньший, чем первый максимум, но больший, чем второй максимум. Тогда следует сохранить текущий элемент списка как второй максимум в переменную *max*2.

Решение задачи:

```
M = [0.903, 0.699, -0.254, 0.082, 0.144]
max1, max2 = M[0],M[1]
if max2 > max1:
    max1, max2 = max2, max1
```

```
for i in range(2,len(M)):
     if M[i] > max1: max1, max2 = M[i], max1
     else:
       if M[i] > max2: max2 = M[i]
print(max1, max2)
```

Результат работы программы:

```
0.903 0.699
```

Задача 13. Найдите максимум среди отрицательных чисел списка из пяти элементов *M*, заполненного случайными числами.

В задаче нельзя использовать первый элемент списка в качестве начального значения переменной *max*, потому что на элементы для поиска накладываются ограничения – они должны быть отрицательными. Может возникнуть такая ситуация, что *M*[0] >=0, тогда в переменную *max* попадет значение заведомо больше, чем любое отрицательное число, и обновление *max* никогда не состоится.

Поэтому в задаче необходимо сначала найти первое отрицательное число в списке и сохранить его как начальное значение в переменную *max.*

Далее алгоритм аналогичен простому поиску максимального числа, только в условие необходимо добавить ограничение на отрицательные числа.

Решение задачи:

```
import random
N, M = 5, []
for i in range(N):
    M.append(round(random.uniform(-1,1),3))
print(M)

max = 0
for x in M:
    if x < 0:
      max = x
      break
```

```
if max == 0:
    print(" Отрицательных чисел нет!")
else:
    for x in M:
      if (x < 0) and (x > max):
        max = x
    print(max)
```

Результат работы программы:

```
[0.157, -0.923, -0.25, -0.092, -0.512]
-0.092
```

Задача 14. Найдите индекс минимального числа в списке A = [0.806, –0.763, 0.673, –0.565, 0.995].

Решение задачи:

```
A = [0.806, -0.763, 0.673, -0.565, 0.995]
index = 0
for i in range(len(A)):
    if A[i] < A[index]:
      index = i
print('A[%d]= %.3f' %(index, A[index] ))
```

Результат работы программы:

```
A[1]= -0.763
```

Задача 15. Дан список A = [0.806, –0.763, 0.673, –0.565, 0.995]. Вставьте значение элемента $A[0]$ после минимального элемента списка.

В задаче первоначально необходимо определить индекс минимального элемента `index`. Далее добавляем один элемент в список для того, чтобы длина списка увеличилась на один элемент. Затем следует сдвинуть все элементы вправо до элемента с индексом `index`. Сдвигаются элементы дублированием $A[i] = A[i – 1]$, причем список в цикле `for` просматриваем справа налево, уменьшая индекс i. После сдвига элементов массива копируем $A[0]$ в элемент `A[index+1]` – в следующий элемент после минимального.

Решение задачи:

```
A = [0.806, -0.763, 0.673, -0.565, 0.995]
index = 0
for i in range(len(A)):
    if A[i] < A[index]:index = i

A.append(0)        # добавляем элемент в список
for i in range(len(A)-1,index,-1):
    A[i] = A[i-1] # сдвигаем элементы вправо
A[index+1] = A[0] # вставляем A[0] после min
print(A)
```

Результат работы программы:

```
[0.806, -0.763, 0.806, 0.673, -0.565, 0.995]
```

Задача 16. Даны два списка: A = [0.806, –0.763, 0.673, –0.565, 0.995] и B = [–0.645, –0.706, –0.156, 0.282, 0.736]. Получите новый список C, элементы которого равны сумме соответствующих элементов списка A и B.

Решение задачи:

```
A = [0.806, -0.763, 0.673, -0.565, 0.995]
B = [-0.645, -0.706, -0.156, 0.282, 0.736]
C = []
for i in range(len(A)):
  C.append(round(A[i]+B[i],3))
print("C =", C)
```

Результат работы программы:

```
C = [0.161, -1.469, 0.517, -0.283, 1.731]
```

Задача 17. Найдите индекс последнего отрицательного элемента в списке A = [0.806, –0.763, 0.673, –0.565, 0.995].

В задаче просматривать элементы рациональнее справа налево. Как только встретился первый отрицательный элемент – выполняем принудительный выход из цикла `break`. Если цикл завершился без `break`, то будет выведено сообщение о том, что отрицательных элементов нет.

Решение задачи:

```
A = [0.806, -0.763, 0.673, -0.565, 0.995]
for i in range(len(A)-1,-1,-1):
    if A[i] < 0:
      print('A[%d]= %.3f' %(i, A[i]))
      break
else:
    print("Отрицательных элементов в цикле нет")
```

Результат работы программы:

```
A[3]= -0.565
```

Встроенные функции и методы списков

К спискам можно применять встроенные функции – `min()`, `max()`, `sum()` и др.

Поиск максимального значения списка:

```
M = [0.903, 0.699, -0.254, 0.082, 0.144]
print("max =", max(M))
```

```
max = 0.903
```

Основные методы списков:

`list.append(x)` – добавляет элемент в конец списка.

`list.extend(L)` – расширяет список `list`, добавляя в конец все элементы списка *L*.

`list.insert(i, x)` – вставляет в *i*-й элемент значение *x*.

`list.remove(x)` – удаляет первый элемент в списке, имеющий значение *x*. Возникает `ValueError`, если такого элемента не существует.

`list.pop([i])` – удаляет *i*-й элемент и возвращает его. Если индекс не указан, удаляется последний элемент.

`list.index(x, [start [, end]])` – возвращает положение первого элемента со значением *x* (при этом поиск ведется от `start` до `end`).

`list.count(x)` – возвращает количество элементов со значением *x*.

`list.sort([key=функция])` – сортирует список на основе функции.

`list.reverse()` – разворачивает список.

`list.clear()` – очищает список.

Например, найти индекс максимального элемента можно следующим образом:

```
M = [0.903, 0.699, -0.254, 0.082, 0.144]
maxElem = max(M)
print("max =", maxElem)
indexMaxElem = M.index(maxElem)
print(("indexMaxElem =", indexMaxElem)
```

```
max = 0.903
indexMaxElem = 0
```

Перевернуть элементы списка можно с помощью метода `reverse()`:

```
X = [1,2,3,4,5,6,7,8,9]
X.reverse()
print(X)
```

```
[9, 8, 7, 6, 5, 4, 3, 2, 1]
```

Сортировать элементы списка можно с помощью метода `sort()`:

```
X = [1,-5,3,4,-15,6,-7,8,-9]
X.sort()
print(X)
```

```
[-15, -9, -7, -5, 1, 3, 4, 6, 8]
```

Метод `count()` возвращает количество элементов в списке, равных ключу:

```
X = [1,5,3,4,5,6,5,8,5]
print(X.count(5))
```

```
4
```

Удалить все элементы, равные ключу, в списке можно следующим образом:

```
X = [1,5,3,4,5,6,5,8,5]
while X.count(5) != 0 : X.remove(5)
print(X)
```

```
[1, 3, 4, 6, 8]
```

Пример выполнения лабораторной работы

Цель работы: получить навыки реализации одномерных массивов на языке программирования Python.

Задание № 1

Найдите сумму и количество положительных элементов массива, оканчивающихся на 4. Элементы массива вводятся в строку.

Решение задачи:

```
"""
ЛР № 8. Задание 1. Вариант 1.
Программа нахождения суммы четных элементов массива, оканчивающихся на 4.

Вводим элементы массива в строку с помощью map(). Сумму элементов накапливаем в переменной sum, количество – в переменной count. Сохраняем начальное значение sum = 0, count = 0. Просматриваем все элементы массива, добавляя к сумме только положительные и оканчивающиеся на 4.

студент гр. ПРИ-124 И.И. Иванов
20.10.2024
"""
```

```
print("Введите массив в строку числа\
разделены пробелами: ")
arr = list(map(int, input().split()))

sum, count = 0, 0
for i in range(len(arr)):
  if arr[i] % 10 == 4 and arr[i] >= 0:
    sum += arr[i]
    count += 1
print("сумма:", sum)
print("count:", count)
```

Результаты работы программы задания 1 представлены на рис. 8.1, блок-схема – на рис. 8.2.

```
print("Введите массив в строку числа \
разделены пробелами: ")
arr = list(map(int, input().split()))

sum, count = 0, 0

for i in range(len(arr)):
    if arr[i] % 10 == 4 and arr[i] >= 0:
        sum += arr[i]
        count += 1
print("сумма:", sum)
print("count:", count)
Введите массив в строку числа разделены пробелами:
1 2 3 4 5 6 7 8 9 10 11 12 13 14
сумма: 18
count: 2
```

Рис. 8.1. Скриншот кода и результатов работы программы задания 1

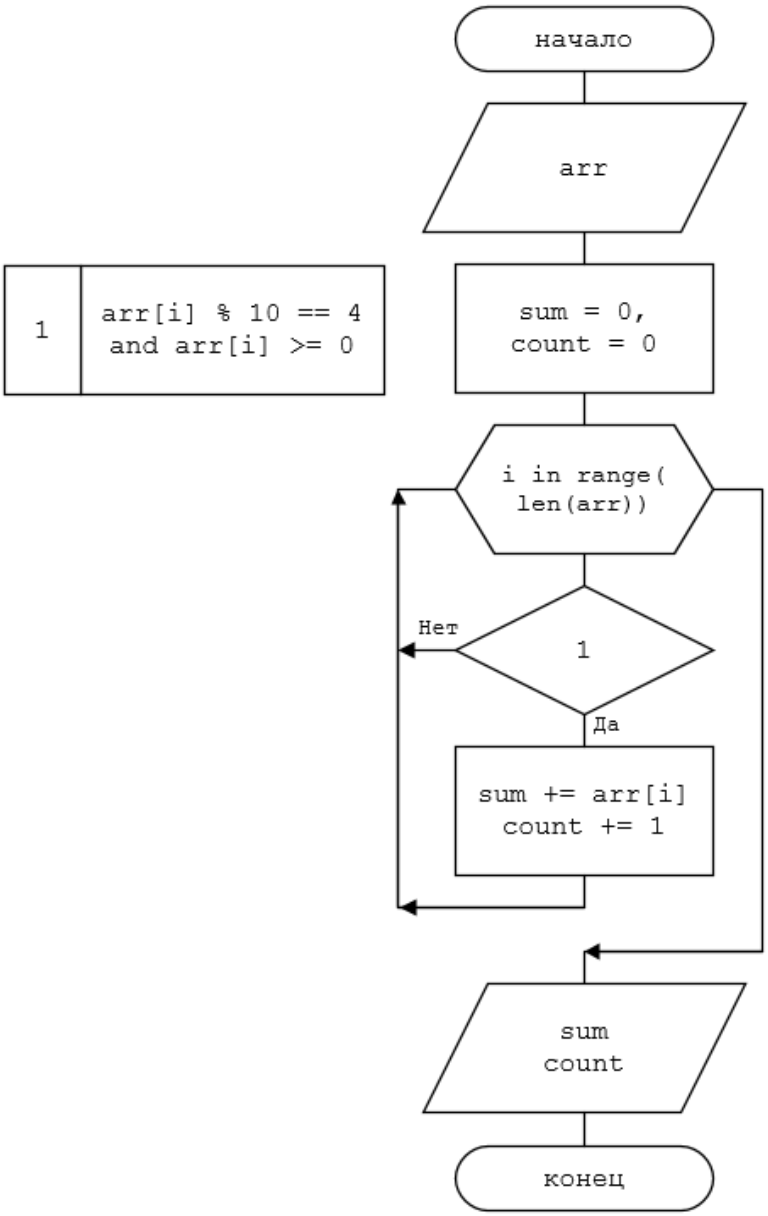


Рис. 8.2. Блок-схема программы задания 1

Задание № 2

Задана последовательность из *N* вещественных чисел. Определите, сколько среди них чисел, меньших *K*, равных *K* и больших *K*.

Решение задачи:

```
"""
ЛР № 8. Задание 2. Вариант 1.
В последовательности из N вещественных чисел опре-
делите, сколько среди них чисел, меньших K, равных
K и больших K.
```

```
Массив чисел вводится в строку с клавиатуры. Значение K вводится с клавиатуры. В переменных less_k, equal_k, greater_k накапливаем количество элементов, соответственно меньших K, равных K, больших K. Просматривая элементы списка, подсчитываем less_k, equal_k. Значение greater_k = N - less_k - equal_k.

студент гр. ПРИ-124 И.И. Иванов
20.10.2024
"""

import math

print("Введите последовательность действительных \
чисел (разделенных пробелами): ")
nums = list(map(float, input().split()))

k = float(input("Введите k: "))

less_k, equal_k = 0, 0

for num in nums:
  if math.isclose(num,k):
    equal_k += 1
  elif num < k:
    less_k += 1
greater_k = len(nums) - less_k - equal_k
print("Количество чисел меньше k:", less_k)
print("Число, равное k:", equal_k)
print("Количество чисел, превышающих k:", \
      greater_k)
```

Блок-схема работы программы задания 2 представлена на рис. 8.3, результаты – на рис. 8.4.

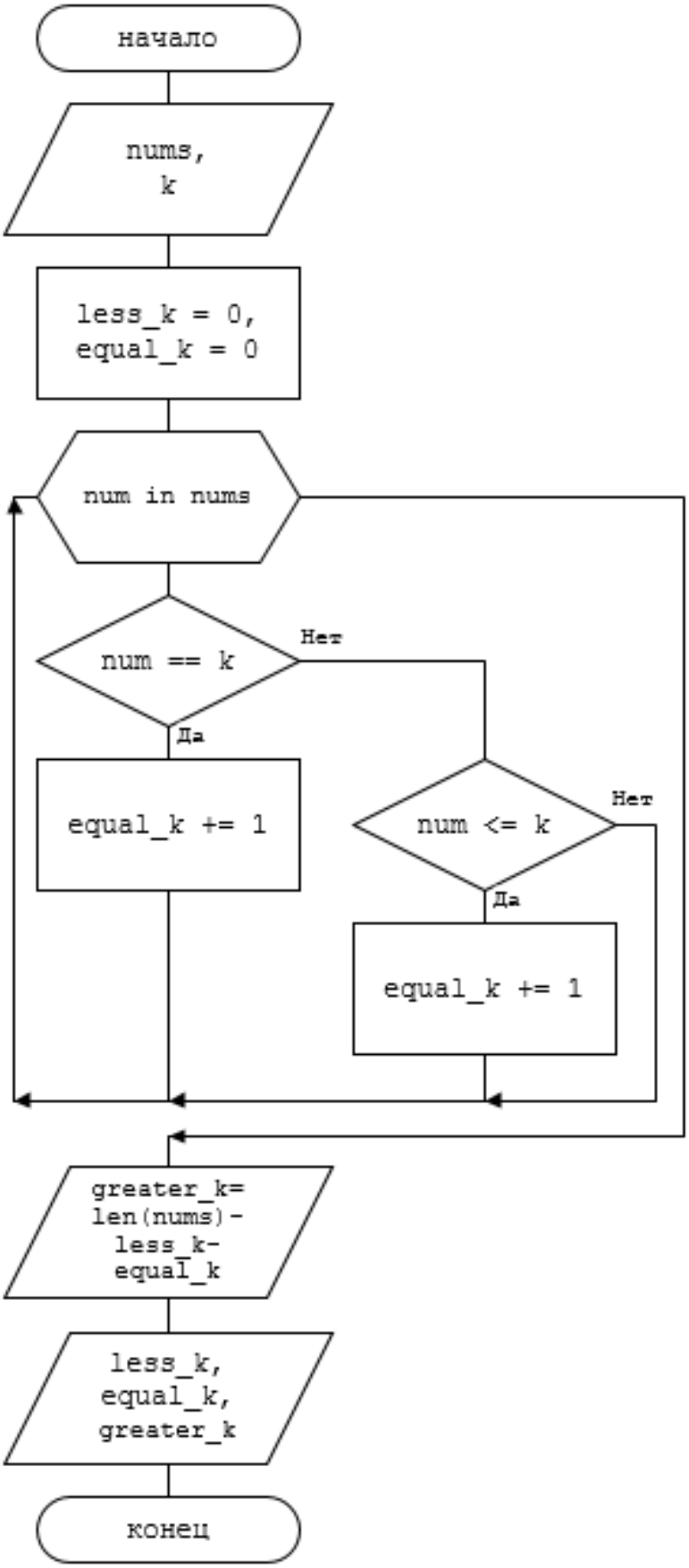


Рис. 8.3. Блок-схема программы задания 2

```
import math

print("Введите последовательность действительных \
чисел (разделенных пробелами): ")
nums = list(map(float, input().split()))

k = float(input("Введите k: "))

less_k, equal_k = 0, 0

for num in nums:
    if math.isclose(num,k):
        equal_k += 1
    elif num < k:
        less_k += 1
greater_k = len(nums) - less_k - equal_k
print("Количество чисел меньше k:", less_k)
print("Число, равное k:", equal_k)
print("Количество чисел, превышающих k:", \
      greater_k)
```

```
Введите последовательность действительных чисел (разделенных пробелами):
1.1 2.4 3.2 5.1 1.5 2.8 3.7 4.3 4.2
Введите k: 3
Количество чисел меньше k: 4
Число, равное k: 0
Количество чисел, превышающих k: 5
```

Рис. 8.4. Скриншот кода и результатов работы программы задания 2

Вывод: Я овладел навыками создания одномерных массивов на языке программирования Python, а также изучил правила составления блок-схем к программам с циклами.

ЗАКЛЮЧЕНИЕ

Большинство приведенных в практикуме алгоритмов – необходимые инструменты в работе каждого программиста. Базовые задачи и различные способы их решения служат основой для решения более сложных задач и разработки программ профессионального уровня. Невозможно написать сложную программу, не обладая знаниями об условиях и циклах, а также о списках и алгоритмах из базовой обработки.

Конечно, с развитием языков программирования все больше и больше средств и методов программирования включаются либо в сам язык, либо в библиотеки классов для этого языка. Python хорошо иллюстрирует это, поскольку в его состав входит большое количество библиотек.

Практикум – только первый шаг на пути изучения такого популярного и востребованного языка программирования, как Python. Ни одна книга не может превратить заурядного программиста в хорошего и грамотного специалиста, если он не будет постоянно пополнять знания и совершенствовать на практике свои навыки.

Желаем успехов в дальнейшем изучении Python!

БИБЛИОГРАФИЧЕСКИЙ СПИСОК

1. *Мусин, Д.* Самоучитель Python [Электронный ресурс] / Д. Мусин. – Режим доступа: http://server.aesc.msu.ru/materials/PYTHON/pythonworldru.pdf (дата обращения: 29.10.2024).

2. *Маккинли, У.* Python и анализ данных [Электронный ресурс] / У. Маккинли. – М. : ДМК Пресс, 2015. – 482 с. – ISBN 978-5-97060-315-4. – URL: https://www.studentlibrary.ru/book/ ISBN9785970603154.html (дата обращения: 29.10.2024). – Режим доступа: по подписке.

3. *Рашка, С.* Python и машинное обучение [Электронный ресурс] / С. Рашка ; пер. с англ. А. В. Логунова. – М. : ДМК Пресс, 2017. – 418 с. – ISBN 978-5-97060-409-0. – URL: https://www.studentlibrary.ru/book/ISBN9785970604090.html (дата обращения: 29.10.2024). – Режим доступа: по подписке.

4. *Царёв, Р. Ю.* Алгоритмы и структуры данных (CDIO) [Электронный ресурс] : учебник / Р. Ю. Царёв. – Красноярск : СФУ, 2016. – 204 с. – ISBN 978-5-7638-3388-1. – URL: https://www.studentlibrary.ru/book/ISBN9785763833881.html (дата обращения: 29.10.2024). – Режим доступа: по подписке.

5. *Вирт, Н.* Алгоритмы и структуры данных [Электронный ресурс] / Н. Вирт ; пер. с англ. Ф. В. Ткачева. – 3-е изд., стер. – М. : ДМК Пресс, 2023. – 274 с. – ISBN 978-5-89818-313-4. – URL: https://www.studentlibrary.ru/book/ISBN9785898183134.html (дата обращения: 29.10.2024). – Режим доступа: по подписке.

Учебное электронное издание

ШАМЫШЕВ Антон Андреевич
ШАМЫШЕВА Ольга Николаевна

ПРОГРАММИРОВАНИЕ НА PYTHON
Решение задач по основам алгоритмизации

Практикум

Редактор Е. А. Лебедева
Технические редакторы Ш. Ш. Амирсейидов, Н. В. Пустовойтова
Компьютерная верстка Л. В. Макаровой
Корректор О. В. Балашова
Выпускающий редактор А. А. Амирсейидова

Системные требования: Intel от 1,3 ГГц; Windows XP/7/8/10;
Adobe Reader; дисковод CD-ROM.

Тираж 9 экз.

Издательство Владимирского государственного университета
имени Александра Григорьевича и Николая Григорьевича Столетовых.
600000, Владимир, ул. Горького, 87.