Министерство науки и высшего образования Российской Федерации

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

Ф. Ж. ТАННИНГ

F. `J. TANGNING

ОСНОВЫ ПРОГРАММИРОВАНИЯ В PYTHON: ТРИ В ОДНОМ

THE BASICS OF PYTHON PROGRAMMING: THREE IN ONE

Учебное пособие (The course book)

В четырех томах (In four volumes)

Том 1 (*Volume I)*

Владимир 2021

UDC 004.432
BBK 32.973.2
T19 (engl.)

Reviewers:
Candidate of Technical Sciences, Associate Professor
Associate Professor of the Department of Information Systems
of Tyumen State University
Yu. E. Karyakin

Doctor of Technical Sciences, Professor
head of the Department of Computer Engineering and Control Systems
of Vladimir State University named after Alexander and Nikolai Stoletovs
V. N. Lantsov

Tangning J., F.

T19 The basics of Python programming: Three in One : the course book. In 4 vol. Vol. 1 / F. J. Tangning ; Vladimir State University named after Alexander and Nikolai Stoletovs. – Vladimir : Ed. VlSU, 2021. – 339 p.

ISBN 978-5-9984-1228-8 (vol. 1). – ISBN 978-5-9984-1227-1.

The course book contains a step by step manual to start programming in Python. In addition, the book describes and analyzes the most important points in learning this programming language. The author focuses on schematic algorithmic knowledge in order to increase efficiency in solving problems. The teaching material in the book enables foreign (non-Russian) students to master the discipline by overcoming the language barrier. Students who master Russian can improve their knowledge of French and English through the synchronization of the material presented in these languages.

The publication is recommended for students of various technical fields, teachers and anyone who is starting to take their first steps in programming, using the Python language as a programming tool.

It is also recommended for the acquisition of general professional skills in accordance with the higher education standards of the Russian Federation.

Tabl. 11. Il. 12. Bibliogr.: 22 titl.

UDC 004.432
BBK 32.973.2

ISBN 978-5-9984-1228-8 (vol. 1)
ISBN 978-5-9984-1227-1

УДК 004.432
ББК 32.973.2
Т18 (рус.)

Рецензенты:
Кандидат технических наук, доцент
доцент кафедры информационных систем
Тюменского государственного университета
Ю. Е. Карякин

Доктор технических наук, профессор
зав. кафедрой вычислительной техники и систем управления
Владимирского государственного университета
имени Александра Григорьевича и Николая Григорьевича Столетовых
В. Н. Ланцов

Таннинг Ж., Ф.

Т18 Основы программирования в Python: три в одном : учеб. пособие. В 4 т. Т. 1 / Ф. Ж. Таннинг ; Владим. гос. ун-т им. А. Г. и Н. Г. Столетовых. – Владимир : Изд-во ВлГУ, 2021. – 339 с.
ISBN 978-5-9984-1228-8 (т. 1). – ISBN 978-5-9984-1227-1.

В учебном пособии представлены последовательные шаги для начала программирования в Python. Кроме того, в книге описываются и анализируются основные первые важные моменты в изучении этого языка программирования. Сделан акцент на знание алгоритмических схем для повышения эффективности в решении задач. Обучающий материал пособия также дает возможность иностранным студентам освоить определенные дисциплины, преодолевая языковой барьер. Студенты, владеющие русским языком, могут совершенствовать свои знания на французском и английском благодаря синхронизации материала, представленного на этих языках.

Издание рекомендовано для студентов различных технических направлений, преподавателям вузов и всем, кто начинает делать свои первые шаги в программировании, используя язык Python в качестве инструмента программирования.

Рекомендовано для формирования общепрофессиональных компетенций в соответствии с ФГОС ВО.

Табл. 11. Ил. 12. Библиогр.: 22 назв.

УДК 004.432
ББК 32.973.3

ISBN 978-5-9984-1228-8 (т. 1)
ISBN 978-5-9984-1227-1

RU СПИСОК ОСНОВНЫХ СОКРАЩЕНИЙ
EN MAIN ABBREVIATIONS
FR LES ABRÉVIATIONS PRINCIPALES

RU – означает, что за этой частью главы на русском языке следует перевод на английский и французский языки
EN – означает, что за этой частью главы на английском языке следует перевод на русский и французский языки
FR – означает, что за этой частью главы на французском языке следует перевод на русский и английский языки

RU – *means that this part of the chapter in Russian is followed by a translation into English and French*
EN – *means that this part of the chapter in English is followed by a translation into Russian and French*
FR – *means that this part of the chapter in French is followed by a translation into Russian and English*

RU – *signifie que cette partie du chapitre en russe est suivie d'une traduction en anglais et en français*
EN – *signifie que cette partie du chapitre en anglais est suivie d'une traduction en russe et en français*
FR – *signifie que cette partie du chapitre en français est suivie d'une traduction en russe et en anglais*

ОГЛАВЛЕНИЕ

TABLE OF CONTENTS

TABLE DES MATIÈRES

ПРЕДИСЛОВИЕ

Идея написания книги появилась после того, как в 2020 году была издана книга под названием «Информатика и информационные технологии: три в одном» = «*Informatics and Information Technology: Three in One»*. В этой книге среди прочего мы попытались сделать сравнение между языком Pascal и языком Python, показать способ обучающимся простого перехода в программировании с языка Pascal на Python, тот язык, который по своему опыту автор считает ценным инструментом для будущих разработчиков приложений или какого-либо другого программного обеспечения.

Программируя коды, мы учимся понимать, как устроены наши собственные мыслительные процессы, в той мере, в какой хотели бы, чтобы пользователь понял, что исполняющая программы машина (компьютер) – это некое самостоятельно мыслящее существо, способное к логическим рассуждениям, но, конечно же, без чувств, без веры и души.

В учебном пособии предлагается подход к обучению, основанный на конкретных примерах. Мы убеждены, что для лучшего понимания необходимо не только преподать теорию, но и проиллюстрировать эту теорию на разборе реальных и понятных примеров. Для Python существует огромное количество бесплатных инструментов разработки приложений благодаря тому, что это открытый, свободный и портативный язык кода, который поддерживают разные операционные системы.

Почему именно Python?

Это универсальный язык, который особенно эффективен.

Это бесплатный язык, с открытым исходным кодом, что означает, он может бесплатно использоваться и свободно изменяться.

Это относительно простой язык для начинающих благодаря очень простому синтаксису, близкому к синтаксису английского языка.

Это язык с несколькими парадигмами, и он также может использоваться на нескольких платформах.

Изначально это был язык сценариев, который вскоре после появления превратился в объектно-ориентированный и универсальный язык.

Это язык, код которого может выполняться как из интерпретатора, так и компилятора.

Он популярен и привлекателен тем, что позволяет разработчику меньше работать за счет того, что остальную часть работы делает за него компьютер.

Python является *динамическим* (это означает, что состояние переменных изменяется при последовательном выполнении набора инструкций, мы можем изменять тип переменной в течение ее жизни), *конструктивным* (одна концепция в значительной степени достаточна для создания очень богатых конструкций), *рефлективным* (поддерживает метапрограммирование, например, способность объекта добавлять или удалять атрибуты или методы или даже менять классы во время выполнения) и *интроспективным* (т. е. большое количество инструментов разработчика, таких как ***debugger*** или ***profiler***, имплантированы в самом Python).

Установка

Язык был создан голландским разработчиком, *Гвидо ван Россумом,* в 1991 году (дата выхода первой версии). Python поддерживает такое программирование, как императивное (описывает операции в последовательных инструкциях, выполняемых компьютером для изменения состояния программы), структурированное, функциональное (дающее один результат для

каждого набора входных значений) и объектно-ориентированное (использующее классы (*Java*) или прототипы, т. е. бесклассовые (*JavaScript*)).

Python имеет меньше синтаксических конструкций, чем многие языки, такие как C, Pascal и др. Многие предприятия, организации и в том числе известные компании, используют Python в своем развитии: *Google, LibreOffice.org, Apache OpenOffice, Corel Paint Shop Pro, FreeCAD, Blender, QGIS, National Aeronautics and Space Administration* и т.д.

Существуют среды разработки, используемые для программирования в Python, эти среды иногда включают в себя множество пакетов, предназначенных для решения конкретных задач. Мы можем привести здесь некоторые из них:

Anaconda (сайт: *https://www.anaconda.com/*) – доступно в бесплатной версии и коммерческой.

ActivePython (сайт: *https://www.activestate.com/products/python/*) – доступно в бесплатной версии и коммерческой.

PyChram (сайт: *https://www.jetbrains.com/pycharm/*) – доступно в бесплатной версии и коммерческой.

PyDev (сайт: *https://www.pydev.org/*) – доступно в бесплатной версии.

WingWare (сайт: *https://wingware.com/*) – доступно в бесплатной версии и коммерческой.

Komodo IDE (сайт: *https://www.activestate.com/products/komodo-ide/*).

The Eric Python (сайт: *https://eric-ide.python-projects.org/*) – предоставляется в бесплатной версии.

Python-IDLE (часть стандартной библиотеки Python – сайт: *https://www.python.org/*).

Spyder (сайт: *https://www.spyder-ide.org/*) – доступно в бесплатной версии для пользователей Windows.

Какую среду разработки выбрать для редактирования кодов?

Поскольку в этой книге мы все еще изучаем элементарные основы программирования, мы можем рекомендовать вам установить дистрибутив *Anaconda*, если у вас еще нет опыта в этой области или у вас не хватает времени для индивидуальной установки некоторых пакетов Python. Программное обеспечение *Anaconda* устанавливает большое количество пакетов по умолчанию, которых будет достаточно, чтобы неплохо освоить основы программирования. Чтобы узнать разницу между этими средами разработки, имеет смысл изучить каждое программное обеспечение (дистрибутив), перечисленное выше, чтобы точно знать, какое из них лучше всего подходит вашему уровню программирования на языке Python.

Если вы не хотите устанавливать среду разработки Python на свой компьютер по какой-либо причине, вы можете использовать онлайн-компиляторы для интерпретации ваших кодов Python. Вот некоторые ссылки на них – URL:

- *https://www.onlinegdb.com/online_python_compiler*
- *https://www.programiz.com/python-programming/online-compiler/*
- *https://repl.it/languages/python3*
- *https://rextester.com/l/python3*
- *https://www.online-python.com/*
- *https://www.pythonanywhere.com/* (требуется предварительная регистрация перед использованием)
- *https://ideone.com/l/python-3*

и т. д.

Версия языка

Язык Python, как и любой другой инструмент программирования, модернизируется, вследствие чего с течением времени появляются новые версии с целью улучшения или усовершенствования продукта. Его поддерживает сообщество активных и ответственных за язык разработчиков, большинство из которых являются приверженцами свободного программного

обеспечения (*free software*). В учебном пособии, мы разберем некоторые примеры и упражнения с использованием версии 3.x, если более конкретно – 3.8.1. Мы учли изменения, которые ожидаются в версии 4, для будущего использования в ней некоторых методов. Как бы то ни было, коды приведенной нами версии всегда будут совместимы с новыми версиями Python, если только политика модернизации продукта не будет изменена радикально.

Порядок чтения

Начнем с того, что первый том издания состоит из шести глав. Каждая из которых включает курс, примеры для объяснения того, что было изложено, и упражнений.

1. ***Глава I*** *рассказывает о том, с чего начать программирование на языке Python: переменные, типы данных, способ отображения и ввода данных на экране, различные операторы и тому подобное.*

2. ***Глава II*** *вводит нас в мир матриц, где мы учимся обращаться с ними и применять их, показывая плюсы и минусы применений.*

3. ***Глава III*** *зарезервирована для библиотеки NumPy, которая поддерживает многомерные массивы (включая матрицы) с операциями над математическими функциями высокого уровня.*

4. ***Глава IV*** *вводит нас в использование исключений для обработки ошибок, которые могут произойти при выполнении кода.*

5. ***Глава V*** *посвящена функциям, таким, которые создаются разработчиками, и собственным функциям в Python. В этой главе мы покажем важность функций в коде Python.*

6. ***Глава VI*** *отведена упражнениям и решению некоторых из них.*

В основу первого тома издания положено изучение основ программирования на языке Python, направленное также на преодоление языковых барьеров учащимися из франко- и англоязычных стран в Российской Федерации.

Автор считает своим долгом выразить искреннюю благодарность заведующему кафедрой информатики и защиты

информации, д.т.н., профессору М. Ю. Монахову; директору института Информационных технологий и радиоэлектроники, профессору А. А. Галкину, за поддержку и помощь в издании пособия.

Автор будет благодарен обучающимся, читателям за замечания и предложения по содержанию книги, которые просит направлять по электронному адресу: tajifirmin2@yahoo.com.

Продолжение на странице 26

PREFACE

The idea of writing this manual came to me after the previous course book was prepared in 2020, entitled «*Информатика и информационные технологии: три в одном*» = «*Informatics and Information Technology: Three in One*». In this book, among other things, we tried to make a comparison between the Pascal and the Python language and to help learners and students easily switch programming from Pascal to Python, a language that, in my experience, I consider a precious tool for future developers of applications or any other software.

By programming codes, we learn to understand how our own thought processes are arranged, to the extent that we would like the user to believe that the machine (computer) that executes programs is a kind of an independently thinking being, capable of logical reasoning but, of course, without feelings, without faith and without a soul.

In this manual, we offer a case-based approach to learning. We are convinced that for a better understanding, it is necessary not only to teach the theory, but also to "illustrate" this theory by analyzing palpable and understandable examples. There is a huge number of free application development tools for Python, due to the fact that it is an open, free and portable code language for different operating systems.

Why Python?

It is a universal language that is particularly effective.

It is a free language and an open source, which means that it can be used for free and freely modified.

It is a relatively easy language to learn, even for beginners, thanks to its very simple syntax close to that of English.

It is a multi-paradigm language and it can also be used on multiple platforms.

It was first a scripting language, which soon after its appearance evolved into an object-oriented and universal language.

It is a language whose code can be executed from both the interpreter and the compiler.

It is popular and attractive because it allows the developer to work less due to the fact that the rest of the work is done for him by the computer itself.

Python is dynamic (this means that the state of variables changes when a set of instructions is executed sequentially; we can change the type of a variable during its lifetime), constructive (one concept is pretty much sufficient to create very rich designs), reflective (it supports metaprogramming, such as the ability of an object to add or remove attributes or methods, or even change classes at runtime), and introspective (i.e. a large number of developer tools, such as debugger or profiler, are implanted in Python itself).

Installation

The programming language was created by a Dutch developer, *Guido van Rossum*, in 1991 (the release date of the first version). Python supports such as: imperative programming (which describes operations in sequences of instructions executed by a computer to change the state of a program), structured programming, functional programming (which produces one result for each set of input values), and object-oriented programming (which uses classes (as in *Java*) or prototypes, i.e. classless (*JavaScript*)).

Python has fewer syntactic constructions than many other programming languages such as ***C***, *Pascal*, and many others. Many enterprises, organizations, and even well-known companies use Python in their development: *Google*, *LibreOffice.org*, *Apache OpenOffice*, *Corel Paint Shop Pro*, *FreeCAD*, *Blender*, *QGIS*, *National Aeronautics and Space Administration*, etc.

There are certain development environments used for programming in Python: these environments sometimes include many packages designed to solve specific tasks. We can cite some of them here:

Anaconda (URL: https://www.anaconda.com/) - available in a free or commercial version.

ActivePython (URL: https://www.activestate.com/products/python/) - available in a free or commercial version

PyCharm (URL: https://www.jetbrains.com/pycharm/) - available in a free or commercial version.

PyDev (URL: https://www.pydev.org/) - available in a free version.

WingWare (URL: https://wingware.com/) - available in a free or commercial version.

Komodo IDE (URL: https://www.activestate.com/products/komodo-ide/).

The Eric Python (URL: https://eric-ide.python-projects.org/) - available in a free version.

Python-IDLE (part of the standard Python library – URL: https://www.python.org/).

Spyder (URL: https://www.spyder-ide.org/) – available in a free version for Windows users.

Which development environment should I choose for editing codes?

Since we are still learning the rudimentary basics of programming in this course book, we can recommend that you install the *Anaconda* distribution if you don’t have any experience in this area or don’t have enough time to install some Python packages individually. The Anaconda software installs a large number of packages by default, which will be enough to get a good grasp of the basics of programming. To learn the difference between these development environments, it makes sense to study each software (distribution) listed above to know exactly which one suits your level of Python programming best.

If you don’t want to install the Python development environment on your computer for some reason, you can also use online compilers to interpret your Python codes. Here are some URL links to them:

https://www.onlinegdb.com/online_python_compiler

https://www.programiz.com/python-programming/online-compiler/
https://repl.it/languages/python3
https://rextester.com/l/python3
https://www.online-python.com/
https://www.pythonanywhere.com/ (requires pre-registration before use)
https://ideone.com/l/python-3
etc.

Language version

Python, like any other programming tool, is being upgraded, and as a result, new versions are being released over time to improve the product. It is supported by a community of active and responsible developers, most of whom are free software adherents. In this book, we will look at some examples and exercises using version 3.x, more specifically 3.8.1. We have taken into account the changes that are expected in version 4, for the future use. In spite of this fact, the codes of the above version will always be compatible with new versions of Python, unless the product upgrade policy is radically changed.

Reading order

To begin with, this first volume of the textbook consists of 6 chapters. Each of them consists of a course, examples to explain what was stated, and exercises.

1. Chapter I explains how to start programming in Python: variables, data types, how to display and enter data on the screen, various operators, and so on.

2. Chapter II introduces us to the world of matrices, where we learn how to handle and apply them, showing the pros and cons of their applications.

3. Chapter III is reserved for the NumPy library, which supports multidimensional arrays (including matrices) with operations on high-level mathematical functions.

4. Chapter IV introduces us to the use of exceptions to handle errors that may occur during code execution.

5. Chapter V deals with functions: such as those created by developers, and native functions in Python. In this chapter, we will show the importance of functions in Python code.

6. Chapter VI is devoted to exercises and the solution of some of them.

The first volume of the course book is focused on the study of the basics of programming in Python; it is also aimed at overcoming language barriers for learners from French-speaking and English-speaking countries in the Russian Federation.

The author considers it his duty to express his sincere thanks to the Head of the Computer Science Department, Doctor of Technical Sciences, Professor Monakhov M. Yu; to the Principle of the Institute of Information Technology and Radio-Electronics – Professor Galkin A. A. for their support and assistance in the publication of the manual.

The author would like to thank the users of the manual for their comments and suggestions on the content of the book, which he requests to send to the following e-mail address: tajifirmin2@yahoo.com.

Continued on page 30

LA PRÉFACE

L'idée de rédiger cet ouvrage est apparue après la rédaction du précédent en 2020 intitulé: «*Информатика и информационные технологии: три в одном*» = «*Informatique et la technologie de l'information: Trois en un*». Dans ce livre, entre autre, nous avons essayé de faire une comparaison entre le langage Pascal et celui de Python; une façon d'amener les apprenants du Pascal à migrer facilement dans la programmation avec la langage Python, qui selon mon expérience est considéré comme un bijou pour les futurs développeurs d'applications ou autre.

En programmant les codes, nous apprenons à construire notre propre raisonnement, dans la mesure où nous voulons faire croire à l'utilisateur, que la machine (ordinateur) est une personne vivante accompagnée d'un raisonnement logique et bien sûr que sans sentiments, sans foi et âme.

Dans ce manuel, nous proposons une démarche d'apprentissage basée sur des exemples bien déterminés. Nous sommes convaincu que pour mieux comprendre, il est indispensable non pas seulement de donner la théorie, mais aussi d'illustrer cette théorie à travers des exemples palpables et compréhensibles. Avec Python, il existe un nombre important légal d'outils pour développer une application, étant donné qu'il est un langage à code ouvert, libre et portable sur différents systèmes d'exploitations.

Pourquoi le langage Python?

Il s'agit d'un langage généraliste, particulièrement performant.

Il est gratuit et open source, ce qui veut dire qu'il peut être utilisé et modifié librement.

Il est un langage relativement facile à apprendre même pour les débutants, de par sa syntaxe très simple et proche de l'anglais.

C'est un langage multi-paradigme et il peut être aussi utilisé sur un maximum de plates-formes.

C'est un langage de script, qui a vite évolué pour enfin devenir aussi un langage orienté objet et polyvalent.

C’est un langage où le code peut être exécuté à partir d’un interpréteur ou à partir d’un compilateur.

Il est populaire et attractif parce qu'il permet au développeur de travailler moins au détriment de l'ordinateur qui doit faire le reste.

Python est dynamique (ce qui veut dire que l'état des variables est modifié lors d'une exécution séquentielle d'une suite d'instructions; nous pouvons changer le type d’une variable au cours de sa vie), constructif (un concept est largement suffisant pour engendrer des constructions très riches), réflectif (il supporte la méta-programmation, par exemple la capacité pour un objet de se rajouter ou de s'enlever des attributs ou des méthodes, ou même de changer de classe en cours d'exécution), et introspectif (un grand nombre d'outils de développement, comme le ***debugger*** ou le ***profiler***, sont implantés en Python lui-même).

Installation

Le langage a été crée par *Guido van Rossum* en 1991 (Date de sortie de la première version). Guido est un développeur de nationalité Néerlandaise. Python favorise la programmation impérative (décrit les opérations en séquences d'instructions exécutées par l'ordinateur pour modifier l'état du programme) structurée, fonctionnelle (donne un seul résultat pour chaque ensemble de valeurs en entrée) et orientée objet ou par objet (langages à classes (*Java*) ou à prototypes c’est-à-dire sans classe (*JavaScript*)).

Python possède moins de constructions syntaxiques que de nombreux langages tels que C, Pascal et bien d’autres. Beaucoup entreprises, organismes ou sociétés reconnues utilisent Python dans leur développement: *Google, LibreOffice.org, Apache OpenOffice, Corel Paint Shop Pro, FreeCAD, Blender, QGIS, National Aeronautics and Space Administration etc*.

Il existe des environnements de développement intégrés utilisé pour programmer en Python, ces environnements incluent parfois beaucoup de paquets dédiés à résoudre des tâches bien précises. Nous pouvons citer quelques une:

Anaconda (*https://www.anaconda.com/*) – disponible en version gratuite ou commerciale.

ActivePython (*https://www.activestate.com/products/python/*)- disponible en version gratuite ou commerciale

PyCharm (*https://www.jetbrains.com/pycharm/*) – disponible en version gratuite ou commerciale.

PyDev (*https://www.pydev.org/*) – disponible en version gratuite.

WingWare (*https://wingware.com/*) – disponible en version gratuite ou commerciale.

Komodo IDE (*https://www.activestate.com/products/komodo-ide/*).

The Eric Python (*https://eric-ide.python-projects.org/*) – disponible en version gratuite.

Python-IDLE (Une partie de la bibliothèque Python standard – *https://www.python.org/*).

Spyder (*https://www.spyder-ide.org/*) – disponible en version gratuite pour les utilisateurs de Windows.

Quel environnement choisir pour éditer les codes?

Étant donnée que dans ce livre, nous étudions encore les bases élémentaires de la programmations, nous pouvons vous recommandez d'installer la distribution Anaconda si vous n’avez pas encore d’expérience dans le domaine ou vous manquez de temps pour installer individuellement certains paquets du Python. Le logiciel Anaconda installe un très grand nombre de paquets par défaut qui seront largement suffisants pour mieux maîtriser les bases de la programmation. Pour savoir la différence qui existe entre ces environnements de développement, il sera plus logique d’étudier chaque logiciel (distributif) énuméré plus haut afin de savoir exactement celui qui vous correspond le mieux selon votre niveau de programmations en Python.

Dans le cas où vous ne voulez pas installer l'environnement de développement Python sur votre ordinateur pour des raisons personnelles,

vous pouvez utiliser des compilateurs en ligne pour interpréter vos codes Python. Voici quelques liens URL:

https://www.onlinegdb.com/online_python_compiler

https://www.programiz.com/python-programming/online-compiler/

https://repl.it/languages/python3

https://rextester.com/l/python3

https://www.online-python.com/

https://www.pythonanywhere.com/ (nécessite un pré-enregistrement avant utilisation)

https://ideone.com/l/python-3

etc.

Version du langage

Le langage Python, comme tout autre instrument de programmation, se modernise et c'est de cette manière que les versions changent au fil des temps dans le but d'améliorer ou perfectionner le produit. Il est soutenu par une communauté de développeurs actifs et responsables, dont la plupart sont des adhérents du logiciel libre. Dans le manuel suivant, nous avons résolue certains exemples et exercices en utilisant la version 3.x et plus précisément 3.8.1. Nous avons pris en compte les changements qui se produiront dans la version 4 sur l'utilisation de certaines méthodes. Qu'a cela ne tienne, les codes de cette version seront toujours compatibles avec les nouvelles versions du Python, sauf changement radical de la politique de modernisation du produit.

Ordre de lecture

Tout d'abord, nous dirons que ce premier volume du manuel est constitué de 6 chapitres. Chaque chapitre est constitué d'un cours, des exemples pour expliciter ce qui a été relatée et des exercices.

1. ***Le chapitre I*** *, nous apprenons sur quoi commencer dans la programmation du langage Python: variables, types de données, mode d'affichage et d'introduction des données sur écran, les différents opérateurs et autres.*

*2. **Le chapitre II** nous introduit dans le monde des matrices où nous apprenons à les utiliser et manipuler tout en montrant les avantages et les inconvénients des uns et des autres.*

*3. **Le chapitre III** est réservé à la bibliothèque NumPy qui prend en charge les tableaux multidimensionnels (y compris les matrices) avec des opérations sur les fonctions Mathématiques de haut niveau.*

*4. **Le chapitre IV** nous introduit dans l'utilisation des exceptions pour gérer les erreurs qui peuvent arriver dans l'exécution d'un code.*

*5. **Le chapitre V** est basé sur les fonctions créées par le développeur et les fonctions natives. Dans ce chapitre, nous montrons l'importance des fonctions dans un code.*

*6. **Le chapitre VI** est réservé aux exercices et à la résolution de quelques uns.*

Le premier volume de cet ouvrage est concentré sur l'étude des bases de la programmation en Python; il vise également à surmonter les barrières linguistiques pour les apprenants des pays francophones et anglophones de la Fédération de Russie.

L'auteur estime de son devoir d'exprimer ses sincères remerciements à son chef du département d'informatique, docteur en sciences techniques, professeur Monakhov M. Yu; au directeur de l'Institut des Technologies de l'Information et de la Radio-électronique – professeur Galkin A. A. pour le soutien et l'assistance dans la publication du manuel.

L'auteur sera reconnaissant aux utilisateurs du manuel pour leurs commentaires et suggestions sur le contenu du livre, qu'il demande d'envoyer à l'adresse électronique suivante: tajifirmin2@yahoo.com.

Suite à la page 34

ВВЕДЕНИЕ

Можно ли научиться правильно программировать на языке, не зная основ? С таким же успехом можно было бы задать вопрос, возможно ли правильно изучать язык (например, русский, французский, английский или какой-нибудь другой), не зная его алфавита? Прежде чем ответить на эти вопросы, давайте попробуем определить, что такое программирование в области информационных технологий.

С развитием технологий и компьютерных устройств программирование стало основой одной из профессий будущего – программист, успешная карьера этого специалиста в современном цифровом обществе гарантирована. Но, давайте обо всем по порядку!

Программирование в той или иной мере – это кодирование, состоящее из совокупности выполняемых действий (задач – инструкций), которые приводят к созданию программного обеспечения. Для достижения этой цели важно иметь алгоритмические знания, предпочтительно схематические, они будут использоваться для написания кода. Код пишется в соответствии с правилами, которые подходят для четко определенного языка. Это означает, что для программирования нам необходим язык программирования, включая среду разработки на нем, в то время как для создания самого алгоритма нужно просто знать шаги, которые требуется выполнить для решения поставленной задачи. Эти шаги предпочтительно лучше представить в виде схематического блока (состоящего из шагов). В случае, если мы не знаем, как шаги представить в виде схемы, мы можем отобразить решение с помощью словесного алгоритма.

Написание программы включает в себя перенос и интерпретацию на компьютере человеческого мышления (в пошаговом виде), находящего решение конкретной проблемы с помощью определенного языка. Предположим, что нас просят отсортировать в порядке возрастания целую последовательность, состоящую из 5 разных чисел: 2, 0, -3, 6 и 1. Человек, обладающий интеллектуальными способностями, сможет за долю секунды

непосредственно упорядочить её и получить следующий результат: -3, 0, 1, 2, 6.

Возникает вопрос, как сообщить это умение компьютеру, который является просто машиной, чтобы получить тот же результат, который получает человек. Чтобы передать это умение машине, сначала необходимо найти различные пути решения, которые могут привести к одним и тем же результатам, а затем в зависимости от заданного контекста выбрать наиболее эффективный из них.

Другой пример, который можем привести: предположим, нам необходимо кодировать и декодировать текст, чтобы сжать определенную информацию. Для этого следует выбрать и использовать один или несколько существующих методов, а именно: кодирование *Шеннона-Фано*, кодирование *Хаффмана*, кодирование словаря (*Lempel-Zi* (*LZW*)), повторное кодирование (*RLE – Run-Length Encoding*), алгоритм *MTF* (*Move-To-Front*), преобразование *BWT* (*Burrows-Wheeler Transform*) и так далее или даже создать собственный алгоритм.

Поэтому, чтобы внедрять эти алгоритмы в компьютере, мы вынуждены программировать их с помощью определенного компьютерного языка. Языки программирования предназначены для передачи набора инструкций, которые компьютер должен выполнить для получения определенного результата. С помощью такого языка мы, люди, в качестве разработчиков приложений, можем превратить компьютер в нечто “умное”, отсюда и берет свое начало понятие “искусственный интеллект”. Отметим, что каждой инструкции в коде соответствует одно и только одно действие “*мозга*” компьютера, которым является его процессор. Существуют более сотни компьютерных языков. В Википедии мы находим более 700 языков. Эти языки программирования можно разделить на ряд категорий:

1. **Машинные языки** (родные языки процессора; например, *Ассемблер;* их труднее понять, потому что они представлены битами);

2. **Машинно-независимые** (языки высокого уровня).

Машинные языки – это языки низкого уровня, требующие указания мелких деталей процесса обработки данных.

Языки высокого уровня подражают естественным языкам, используя некоторые обычные человеческие слова и определенные математические операции. Эти языки адаптированы к нормальному восприятию человеческим мозгом. Языки высокого уровня делятся по способу описания:

- на ***детализированный*** – этот способ часто используется для введения в программирование; вхождения и сопутствия осмыслению самой концепции программирования; освоения и понимания взаимодействий алгоритмического мышления и программного кода. Примеры языков: *Pascal, Basic, C* и др.
- ***логический*** – здесь встречаются программы, предназначенные для логического мышления с использованием формальных языков, называемых языками предикатов в области искусственного интеллекта. Примеры языков: *Prolog, Lisp* и др.
- ***объектно-ориентированный*** – это новый способ программирования, который отличается от классического способа (первый способ – детализированный); основным инструментом является объект (как объекты используют: классы, методы, экземпляры, наследования и т. д.), где переменные и значения отходят на второй план в программировании. Примеры языков: *Python, Java, C, C++, C#, Delphi, Visual Basic, PHP* и т. д.

Легко заметить, что язык, используемый процессором, человеку воспринимать трудно. Поэтому и существуют “промежуточные” языки (высокого уровня), понятные человеку, но позволяющие давать инструкции процессору. Отметим также, что не все процессоры имеют один и тот же язык, поэтому программное обеспечение, разработанное для компьютера “***A***”, не обязательно сможет работать на компьютере “***B***”, у которого другой процессор. В связи с этим при написании кода будет важно адаптировать программу на компьютерах с разными процессорами. Эта трудность часто возникает, когда мы напрямую используем низкоуровневый язык, близкий к машинному.

Все языки предназначены для выполнения представленных в них инструкций. Их объединяет также наличие правил, позволяющих эти инструкции применять.

Когда обучающийся хочет начать изучение языка программирования, он должен ответить на ряд вопросов:

1. В какой области я собираюсь программировать, т. е. какая область программирования представляется мне наиболее интересной?
2. Могу ли я нарисовать эффективный алгоритм?
3. Если вы выбираете проприетарное программное обеспечение (proprietary software), задайтесь вопросом, сможете ли вы приобрести программное обеспечение для редактирования кода программ? Следует также учитывать стабильность поддержки этого программного обеспечения во времени.
4. Какие существуют эффективные языки, специализирующиеся в этой области?
5. Какова стоимость миграции ваших данных в случае банкротства компании, которая несет ответственность за выбранный вами язык и развивает его?
6. Какие права использования предоставляет вам лицензия на данный язык? Может ли использование этого языка в ваших проектах иметь юридические риски?

Наша цель не состоит в том, чтобы представить различные возможности, которые предлагает язык Python, а в том, чтобы изложить и разобрать основные шаги, которые необходимо пройти, при изучении языка программирования, понять, как работает большинство методов, и, наконец, научиться писать код. Учебное пособие также поможет повысить уровень владения иностранным языком учащегося путем синхронизации курса на двух других языках (французском и английском).

Для достижения нашей цели мы предлагаем примеры и упражнения с решениями.

Продолжение на странице 38

INTRODUCTION

Is it possible to learn how to program correctly in a language without knowing the basics? One might as well ask, is it possible to learn a language correctly (for example, Russian, French, English, or some other language) without knowing the alphabet if it exists? Before answering these questions, let's try to define what *information technology programming* is.

With the development of technology and computer devices, programming has become one of the professions of the future, in which a successful career is almost certainly guaranteed for you in this new digital society. But, let's talk about everything in order!

Programming in one way or another is coding, consisting of a set of actions (tasks – instructions) that lead to the creation of software. To achieve this goal, it is important to have algorithmic knowledge, preferably schematic, that will be used to write code. This code is written according to the rules that are appropriate for a well-defined language. This means that for programming, we need a programming language, including a development environment in it, while for creating the algorithm itself, we just need to know the steps that need to be performed to solve the problem. These steps are preferably better represented as a schematic block (consisting of the steps that need to be performed to solve the task). If we do not know how to represent it in the form of a diagram, then we can display the solution using a verbal algorithm.

Writing a program involves transferring and interpreting human thinking on a computer (in a step-by-step form), finding a solution to a specific problem, using a specific language. Suppose that we are asked to sort in ascending order, an entire sequence consisting of 5 different numbers: 2, 0, -3, 6, and 1. A person with intellectual abilities will be able to directly order it in a fraction of a second and get the following result: -3, 0, 1, 2, 6.

The question arises, how to communicate this skill to a computer that is just a machine, in order to get the same result that a human gets? To

transfer this skill to the machine, you first need to find different solutions that can lead to the same results, and then, depending on the given context, choose the most effective one.

Another example we can give is: suppose we need to encode and decode text in order to compress certain information. To do this, you need to select and use one or more existing methods, namely: *Shannon-Fano* encoding, *Huffman* encoding, dictionary encoding *(Lempel-Zi (LZW)), re – encoding (RLE-Run-Length Encoding), MTF (Move-To-Front), BWT (Burrows-Wheeler Transform),* etc., or even create your own algorithm.

Therefore, in order to implement these algorithms in a computer, we are forced to program them using a specific computer language. These programming languages are designed to convey a set of instructions that a computer must execute to produce a specific result. With the help of such a language, we humans, as application developers, can turn a computer into something "smart," hence the concept of "artificial intelligence." Note that each instruction in the code corresponds to one and only one action of the computer's "brain," which is its processor. There are more than a hundred computer languages. In the free encyclopedia Wikipedia, we find more than 700 languages. These programming languages can be divided into a number of categories:

1. **Computer languages** (native processor languages, for example: the Assembly language; they are harder to understand because they are represented by bits) – machine languages;

2. **Machine-independent language** (high-level languages).

Computer languages (machine languages) are low-level languages that require specifying small details of the data processing.

High-level languages mimic natural languages by using some common human words and certain mathematical operations. These languages are adapted to the normal perception of the human brain. High-level languages are divided by the way they are described into:

- ***detailed language*** – this method is often used for introduction to programming; for entering and accompanying the understanding of the programming concept itself; for mastering and understanding

the interactions of algorithmic thinking and program code. Examples of languages: *Pascal*, *Basic*, *C*, and others.

- ***logical language*** – here you can find programs designed for logical thinking using formal languages called predicate languages in the field of artificial intelligence. Examples of languages: *Prolog*, *Lisp*, and others.
- ***an object-oriented language*** is a new way of programming that differs from the classical way (the first way is detailed); the main tool is the object (how objects use: classes, methods, instances, inheritance, etc.), where variables and values take a back seat in programming. Examples of languages: *Python*, *Java*, *C*, *C++*, *C#*, *Delphi*, *Visual Basic*, *PHP*, and so on.

It is easy to see that the language used by the processor is difficult for a person to perceive. Therefore, there are "intermediate" languages (high-level) that are understandable to a person, but allow you to give instructions to the processor. Note also that not all processors have the same language, so software developed for computer "***A***" will not necessarily be able to run on computer "***B***," which has a different processor. Therefore, when writing a code, it will be important to adapt the program on computers with different processors. This difficulty often arises when we directly use a low-level language that is close to a machine language.

All languages are designed to follow the instructions provided in them. They are also united by the existence of rules that allow these instructions to be applied.

When a student wants to start learning a programming language, they must answer a number of questions:

1. In which area am I going to program, i.e. which area of programming seems to me the most interesting?
2. Can I draw an efficient algorithm?

3. If you choose proprietary software, ask yourself, will you be able to purchase software for editing the program code? You should also consider the stability of support for this software over time;

4. What effective languages are there that specialize in this area?

5. What is the cost of migrating your data in the event of the bankruptcy of the company that is responsible for the language you choose and develops it?

6. What user rights does the license for this language grant you? Can the use of this language in your projects have any legal risks?

Our goal here is not to present the various features that Python offers, but to outline and parse the basic steps that you need to go through to learn a programming language, understand how most methods work, and finally learn how to write a code. This book will also help to improve the student's foreign language proficiency by synchronizing the course in two other languages (Russian and French).

To achieve our goal, we offer examples and exercises with a solution.

Continued on page 43

INTRODUCTION

Peut-on apprendre à bien programmer dans un langage sans savoir les notions de base?. De la même manière, peut-on se poser la question de savoir s'il est possible d'apprendre correctement une langue (telle que le russe, le français, l'anglais etc) sans connaître l'alphabet s'il existe ? Avant de répondre à ces questions, essayons de définir ce que c'est qu'une programmation dans le domaine de la Technologie de l'Information.

Avec le progrès de la technologie et des appareils d'ordinateurs, la programmation est devenue une des professions d'avenir où la carrière est presque totalement garantie dans cette nouvelle société numérique. De quoi s'agit-il?

La programmation d'une manière ou d'une autre est un codage constitué de l'ensemble des activités (instructions) réalisées qui conduisent à la création d'un logiciel. Pour y parvenir, il est important d'avoir des connaissances algorithmiques et de préférence schématiques qui servira de rédiger le code. Ce code est écrit selon des règles qui sont appropriées à un langage bien déterminé. Ce qui veut dire que pour programmer, nous avons besoin d'un langage de programmation y compris d'un environnement de développement, alors que pour élaborer un algorithme, on aura juste besoin de savoir les étapes à suivre pour résoudre le problème posé; de préférence sous forme de bloc schématique (constitué des étapes à suivre pour résoudre le problème posé). Dans le cas où nous ignorons comment le représenter graphiquement, nous pouvons le faire à travers un algorithme écrit – verbal.

L'écriture d'un programme consiste à traiter et interpréter la pensée humaine (à une vitesse amoindrie) dans l'ordinateur, sur un problème posé par le truchement d'un langage donnée. Considérons qu'on nous demande de trier selon un ordre croissant, une suite entière constituée de 5 nombres différents: 2, 0, -3, 6 et 1. L'être humain ayant une capacité intellectuelle pourra directement l'arranger en une fraction de seconde et obtenir le résultat suivant: -3, 0, 1, 2, 6.

La question qui se pose est de savoir comment transmettre cette intelligence à l'ordinateur qui est juste une machine afin de retrouver le même résultat qu'une personne? Pour le transmettre cette connaissance à la machine, il faut d'abord commencer par établir les différents cheminements qui peuvent mener au même résultats et après selon le contexte donné, choisir le plus efficace parmi ces cheminements. Un autre exemple: il peut également nous arriver de coder et décoder un texte, dans le but de compresser certaines données. Pour cela, nous devons également choisir et utiliser l'une des méthodes existantes (ou alors créer la notre): *Codage de Shannon-Fano, codage de Huffman, codage par dictionnaire (Lempel-Zi (LZW)), codage par repetition (RLE – Run-Length Encoding), l'algorithme MTF (Move-To-Front), la transformée BWT (Burrows-Wheeler Transform) etc*.

Alors pour mettre en application ces algorithmes dans l'ordinateur, nous sommes obligés de les programmer en utilisant un langage informatique précis. Les langages informatiques sont destinés à relater l'ensemble des instructions qu'un ordinateur doit exécuter dans le but d'obtenir un résultat déterminé. À travers ce langage, nous, considéré comme des humains, sous le couvert des développeurs d'application, pouvons transformer l'ordinateur à quelque chose d'intelligent, d'où l'expression de l'intelligence artificielle. Notons qu' à chaque instruction dans le code, correspond une et une seule action du "cerveau" de l'ordinateur qu'est le processeur. Il existe plus d'une centaine de langages informatiques. Dans l'encyclopédie libre *Wikipédia*, nous y trouvons plus de 700 langages. Ces langages peuvent être classés en un certain nombre de catégories:

1. **Langage machine** – c'est le langage natif du processeur (par exemple ***Assembleur***) et difficile à comprendre, car il est représenté par des bits;

2. **Langage indépendant de la machine** – langage de haut niveau.

Les langages machine sont des langages de bas niveau qui nécessitent la spécification de petits détails du processus de traitement des données.

Les langages de haut niveau imitent les langues naturelles en appliquant des mots ordinaires et des opérations mathématiques. Ces

langues sont adaptées à la perception du cerveau humain. Les langues de haut niveau sont divisées en:

- ***langage détaillé*** – cette méthode est utilisée pour l'introduction à la programmation; pour l'entrée et la promotion de la compréhension du concept de programmation; pour adapter et comprendre l'interaction de la pensée algorithmique et le code de programmation. Exemples de langues: *Pascal, Basic, C* et autres.
- ***langage logique*** – nous avons dans ce niveau des programmes conçus pour la pensée logique en utilisant des langages formels appelés langages de prédicats dans le domaine de l'intelligence artificielle. Exemples de langues: *Prolog, Lisp* et autres.
- ***langage orienté objet*** – est une nouvelle façon de programmer qui est différente de la méthode classique (première méthode: détaillée); l'outil principal est l'objet (est utilisé: la classe, la méthode, l'instance, l'héritage, etc.), où les variables et les valeurs sont reléguées au second plan dans la programmation. Exemples de langues: *Python, Java, C, C++, C#, Delphi, Visual Basic, PHP*, etc.

Nous avons vu que le langage utilisé par le processeur était difficile à comprendre par l'être humain, raison pour laquelle il existe des langages “intermédiaires” (langages de haut niveau) et compréhensible par l'être humain qui permettent de donner des instructions au processeur. Notons également que tous les processeurs n'ont pas le même langage; donc un logiciel développé pour un ordinateur «***A***» ne pourra pas fonctionner obligatoirement sur un ordinateur «***B***» avec un processeur différent. Ce qui fait qu'en écrivant un code, il sera important d'adapter le programme sur des ordinateurs avec des processeurs différents. Cette difficulté s'observe fréquemment lorsque nous utilisons directement le langage de bas niveau qui est proche du langage machine.

Tous les langages ont pour but d'exécuter les instructions qui sont soumises. Ils ont en commun aussi la présence des règles qui permettent leur manipulation. Lorsqu'un étudiant (apprenant) veut apprendre nouvellement un langage de programmation, il doit répondre à un certain nombre de questions:

1. Dans quel domaine devrais-je programmer c’est-à-dire quelle aspiration je ressens de la programmation?

2. Suis-je capable de dessiner un algorithme efficace?

3. Si vous choisissez un logiciel non libre, posez-vous la question de savoir si vous êtes capable de s’en procurer pour éditer des programmes codes? il faut tenir compte également de sa stabilité dans le temps;

4. Quels sont les langages efficaces spécialisés dans ce domaine?

5. Quel est le coût de migration de vos données en cas de faillite de l'entreprise qui est responsable et développe le langage choisi ?

6. Quel droit peut vous accorder la licence de ce langage? peut-il avoir des risques juridiques dans l'utilisation de ce langage dans vos projets?

Mon but ici n'est pas de venter les différentes possibilités qu'offre le langage Python, mais d’élaborer des étapes essentielles qu'il faut parcourir pour apprendre un langage de programmation, de comprendre comment fonctionne la plus part des méthodes et enfin d’apprendre à écrire un code. Ce manuel permet également d’améliorer le niveau de connaissance de langue étrangère d'un apprenant à travers une synchronisation du cours en deux autres langues que sont le russe et l’anglais.

Pour atteindre notre but, nous offrons des exemples et exercices avec résolutions à l'appui.

Suite à la page 47

RU Глава I. ИЗУЧЕНИЕ ОСНОВ ПРОГРАММИРОВАНИЯ НА ЯЗЫКЕ PYTHON
EN Chapter I. LEARNING THE BASICS OF PYTHON PROGRAMMING
FR Chapitre I. APPRENDRE LES BASES DE LA PROGRAMMATION DANS LE LANGAGE PYTHON

RU I.1. Основные сведения

Переменные – используются для резервного копирования некоторых данных. Данные, находящиеся в этих переменных, могут быть преобразованы, т. е. претерпеть изменения. Имя переменной почти всегда начинается с латинской буквы. Категорически запрещается использовать ключевые или специальные символы.

Типы данных – в Python существует несколько типов данных. Наиболее часто используемые:

Int (от слова "Integer" - предназначается для целых чисел);

float (предназначается для действительных чисел);

str (от слова “String” – предназначается для слов или фраз - строка символов);

bool (от слова "Boolean” - предназначается для двух данных: “true” (истина) или “false” (ложь))

Пример 1 Определите тип данной переменной x.

```
x = 4          # x принимает значение 4
y = (type(x)) # Результат будет: <class 'int'>, после напечатания.
```

Комментарии – используются для объяснения команды или процесса в программном коде.

Используется символ “#”. Он ставится перед фразой (фраза не должна быть длиннее строки). Для текстового блока используются следующие символы « ''' » в начале и в конце текста. Когда эти символы используются, Python не пытается исправить или проверить синтаксис того, что написано в качестве комментария.

Отображение на экране – существует возможность отображения информации на экране в соответствии с определенной структурой.

Используется команда: *print()*;

Чтобы понять различные возможности, давайте попробуем рассмотреть несколько примеров:

a) предположим, что мы хотим вывести на экран следующеее предложение: «Python “легко” изучать». Тогда наличие двойных кавычек в слове “легко” даёт синтаксическую ошибку, если мы используем эти кавычки в команде: print("Python "легко" изучать").

Чтобы исправить, необходимо писать так:

print("Python \"легко\" изучать")

или вместо двойных кавычек команды *print*() мы используем единичные кавычки:

print('Python "легко" изучать')

b) Предположим, что мы хотим отобразить эту фразу: «Python – мой любимый язык.» в двух строках: первая строка: “Python –”; вторая строка: “мой любимый язык.”.

Итак, у нас есть **первый вариант:**

print("Python – **n**мой любимый язык.");

второй вариант:

print("Python – ");

print("мой любимый язык.").

c) Предположим, что мы хотим добавить в существующее предложение значение любой переменной. Например, в предложении: «Общая оценка ...» на месте символов “...” должно быть представлено значение, которое находится, допустим, в переменной ***S***. Тогда мы получим:

print(f"Общая оценка {S}") или

print("Общая оценка "+str(S))

Если S была переменной типа String (т. е. str), то мы напишем:

print("Общая оценка "+S)

Это означает, что функция **str**() преобразует значение в «String» – строку слов или символов. Буква ***f*** или ***F*** представляет собой отформатированную буквенную строку.

d) Рассмотрим тот же пример, что и в пункте “*c)*” – мы хотим добавить в существующее предложение значение любой переменной. Например: «Общая оценка равна ...»; пустота, отмеченная символами в кавычках “…”, должна быть представлена значением, находящимся, например, в переменной T. Для этого используется символ процента «**%**» или фигурные скобки «**{}**». Тогда у нас будут следующие возможности в зависимости от типа:

вариант №1: T – является натуральным целым числом

```
T=5
messag = ("Общая оценка равна %d – Отлично" %(T))
print(messag)
# В результате мы получим:
# Общая оценка равна 5 – Отлично
```

вариант №2: T – это действительное число

```
T=4.85
messag = ("Общая оценка равна %f – Отлично" %(T))
print(messag)
# В результате мы получим:
# Общая оценка равна 4.850000  – Отлично
```

вариант №3: T – это действительное число, с ограниченным числом цифр после запятой (точка)

```
T=4.85
messag = ("Общая оценка равна %.3f – Отлично" %(T))
print(messag)
# В результате мы получим 3 цифры после запятой (точка)
# Общая оценка равна 4.850 – Отлично
```

вариант №4: T – это строка символов

```
T="4.85" # или T=4.85
messag = ("Общая оценка равна %s – Отлично" %(T))
print(messag)
```

```
# В результате мы получим:
# Общая оценка равна 4.85 – Отлично
```

вариант №5: (*Вернитесь к этому примеру после изучения следующей части о «Вводе данных»*)

```
imia = input('Введите ваше имя: ')
predmiet= input('Введите изучаемый предмет: ')
note = 4
print("Меня зовут {0}. Я получила оценку {2} во время экзамена по {1} в университете." .format(imia, predmiet, note))
# Он эквивалентен:
print("Меня зовут ", imia , ". Я получила оценку ", note , "во время экзамена по ", predmiet , "в университете.")
#Результат:
Введите ваше имя: Алис
Введение изучаемого материала: информатика
Меня зовут Алис. Я получила оценку 4 во время экзамена по информатика в университете
# по информатике (должно быть)
```

е) Выполнение операций во время процедуры отображения

```
a=13.8
b=16.4
print("Значение 'a' – %s, а значение 'b' – %s. Мы можем также сделать сумму значения 'a' и 'b', которая дает c=%s. "%(a, b, a+b))

#Результат:
Значение 'a' – 13.8, а значение 'b' – 16.4. Мы можем также сделать сумму значения 'a' и 'b', которая дает c=30.2.
```

Примечание:

Используя следующие комбинации символов: %s, %d, %f, мы должны запомнить, что буквы **s**, **d**, **f** обозначают следующие типы

данных: строку, натуральное целое число и вещественное число соответственно.

Ввод данных после выполнения кода:

Как попросить пользователя ввести значение **X**, которое будет сохранено в другой переменной **Y** в интерактивном режиме?

Для этого мы можем использовать команду *input()*; например:

```
Y = input("X = ")
```

На экране программа попросит нас ввести значение **X**, которое будет храниться в переменной **Y**. Отметим, что данные, введенные функцией *input()*, считаются строкой символов типа **str**().

Упражнение Попробуйте использовать полученную информацию, чтобы получить имя, фамилию и возраст пользователя. После получения информации, выведите на экран сообщение в виде предложения, которое подтверждает, что пользователь получил эти данные, указав год рождения и напомнив имя и фамилию.

Решение

```
familia = input("Введите вашу фамилию: ")
imia = input("Введите ваше имя: ")
vozrast = input("Введите свой возраст здесь: ")
god = 2020-int(vozrast)
print("Господин "+familia+" "+imia+", вы, родившиеся в "+str(god)+", действительно получили ваши данные. ")
```

или

```
familia = input('Введите вашу фамилию: ')
imia = input('Введите ваше имя: ')
vozrast = input('Введите свой возраст здесь: ')
god = 2020-int(vozrast)
print('Господин '+familia+' '+imia+', вы, родившиеся в '+str(god)+
', действительно получили ваши данные. ')
```

Продолжение на странице 52

EN I.1. Basic information

Variables – used to back up some data. The data contained in these variables can be transformed, i.e. undergo changes. The variable name almost always starts with a Latin letter. It is strictly forbidden to use key or special characters.

Data Types – There are several data types in Python. Most frequently used:

Int (from the word “Integer” – intended for integers);

float (intended for real numbers);

str (from the word “String” – intended for words or phrases – a string of characters);

bool (from the word “Boolean” – is intended for two data: “true” or “false”)

Example Define the type of this variable x.

```
x = 4          # x takes the value 4
y = (type(x)) # The result will be: <class 'int'>, after printing.
```

Comments – used to explain a command or process in the program code.

The “#” symbol is used. It is placed before the phrase (the phrase should not be longer than a line). For a text block, the following characters are used « ''' » at the beginning and end of the text. When these characters are used, Python does not attempt to correct or check the syntax of what is written as a comment.

Display on the screen – it is possible to display information on the screen in accordance with a certain structure.

The command used is: ***print()***;

To understand the various possibilities, let's try to look at a few examples:

a) Suppose we want to display the following sentence: «Python is “easy” to learn». Then the presence of double quotes in the word “easy”

gives a syntax error if we use these quotes in the command: print ("Python is "easy" to learn").

To fix it, you need to write like this:

print ("Python is \"easy\" to learn")

or instead of double quotes of the *print()* command, we use single quotes:

print ('Python is "easy" to learn')

b) Suppose that we want to display this sentence: «Python is my favorite language.» in two lines: the first line: “Python is”; the second line: “my favorite language.”.

So we have the **first option:**

print("Python is **\n**my favorite language.");

the second option:

print("Python is");

print("my favorite language.").

c) Suppose we want to add the value of any variable to an existing sentence. For example, in the sentence: «Total score ...» in place of the characters “...” a value that is, say, in the variable S must be represented. Then we get:

print(f"Total score {S}") or

print("Total score "+str(S))

If S was a String variable (i.e. str), then we would write:

print("Total score " +S)

This means that the **str()** function converts the value to a «String» – a string of words or characters. The letter ***f*** or ***F*** is a formatted letter string.

d) Consider the same example as in “*c)*” – we want to add the value of any variable to an existing sentence.

For example: «The total score is ...»; the void marked with quotation marks “ ... ” must be represented by a value that is, for example, in the variable T. To do this, use the percent symbol «**%**»or curly brackets «**{}**». Then we will have the following possibilities depending on the type:

option №1: T – is a natural integer

```
T=5
messag = ("Total score is %d – Excellent" %(T))
print(messag)
# As a result, we will get:
# The overall score is 5 – Excellent
```

option №2: T is a real number

```
T=4.85
messag = ("Total score is %f – Excellent" %(T))
print(messag)
# As a result, we will get:
# The total score is 4.850000 – Excellent
```

option №3: T is a real number, with a limited number of digits after the decimal point (dot)

```
T=4.85
messag = ("Total score is %.3f – Excellent" %(T))
print(messag)
# As a result, we will get 3 digits after the decimal point (dot)
# Total score is 4.850 – Excellent
```

option №4: T is a string of characters

```
T="4.85" # or T=4.85
messag = ("Total score is %s – Excellent" %(T))
print(messag)
# As a result, we get:
# Total score is 4.85 – Excellent
```

option №5: *(Come back to this example after learning the next part about «Data Entry»)*

```
imia = input('Enter your name: ')
predmiet= input('Enter the subject you are studying: ')
note = 4
```

print("My name is {0}. I got a grade of {2} during the {1} exam at the university.".format(imia, predmiet, note))

It is equivalent to:

print("My name is", imia,". I got a grade of ", note , "during the ", predmiet, "exam at the university.")

#Result:

Enter your name: Alice

Enter the subject you are studying: computer science

My name is Alice. I got a grade of 4 during the computer science exam at the university

in computer science

e) Performing operations during the display procedure

a=13.8

b=16.4

print("The value of 'a' is %s, and the value of 'b' is %s. We can also make the sum of the values of 'a' and 'b', which gives c=%s. "%(a, b, a+b))

Result:

The value of '***a***' is 13.8, and the value of '***b***' is 16.4. We can also make the sum of the values of '***a***' and '***b***', which gives *c*=30.2.

Note:

Using the following character combinations: %s, %d, %f, we must remember that the letters **s**, **d**, **f** represent the following data types: string, natural integer, and real number, respectively.

Entering data after the code is executed:

How do I ask the user to enter the value of **X**, which will be saved in some other variable **Y** in interactive mode?

To do this, we can use the *input()* command; for example:

```
Y = input("X = ")
```

On the screen, the program will ask us to enter the value of X, which will be stored in the variable Y. Note that the data entered by the *input()* function is considered a string of characters of the **str()** type.

Example

Try using this information to get the user's first name, last name, and age. After receiving the information, display a message in the form of a sentence that confirms that the user has received this data, indicating the year of birth and reminding the first and last name.

Answer

```
familia = input ("Enter your last name:")
imia = input ("Enter your first name: ")
vozrast = input ("Enter your age here: ")
god = 2020 – int(vozrast)
print("Mister" +familia+" "+imia+", born in "+str(god)+ ", have indeed received your data.")
```

or

```
familia = input('Enter your last name:')
imia = input('Enter your name:')
vozrast = input('Enter your age here:')
god = 2020-int(vozrast)
print('Mister' +familia+' '+imia+', born in '+str(god)+ ', have indeed received your data. ')
```

Continued on page 55

FR I.1. Informations de base

Les variables – elles sont utilisées pour sauvegarder certaines données. Les données qui se trouvent dans ces variables peuvent être transformées c'est-à-dire subir des modifications. Le nom d'une variable commence presque toujours par une lettre latine. Il est strictement interdit d'utiliser les symboles clés ou spéciaux.

Les types de données – il existe plusieurs types de données dans Python. Les plus utilisées sont:

int(venant du mot “Integer” – réservé pour les nombres entiers);

float (réservé pour les nombres réels);

str (venant du mot “String” – réservé pour les mots ou phrases – chaîne de caractères);

bool (venant du mot “Boolean” – réservé pour deux données: “true” (vraie) ou “false” (faux))

***Exemple* :** Déterminer le type d’une variable *x* donnée.

```
x = 4          # x prend la valeur 4
y = (type(x)) # Le résultat sera: <class 'int'> s’il est imprimé.
```

Les commentaires – ils sont utilisés pour expliquer une commande ou un processus dans un programme code.

Le symbole utilisé est “#”. Il est placé avant la phrase. Pour un bloc de texte, on utilise les symboles suivants: « ''' » au début et à la fin du texte. Lorsque ces symboles sont utilisés, Python ne cherche pas à corriger ou à vérifier le syntaxe de ce qui est écrit comme commentaire.

Affichage sur l’écran – Il existe des possibilités d'affichage des informations sur l’écran afin de respecter une structure déterminée.

La commande utilisée est: ***print()***;

Pour comprendre les différentes possibilités, essayons de voir quelques exemples:

a) Considérons que nous voulons afficher cette phrase: «Le Python est “facile” à apprendre», alors la présence des doubles guillemets sur le mot “facile” nuit, si nous utilisons ces guillemets dans la commande ***print()***. Pour remédier à cela, nous écrirons de cette manière:

```
print("Le Python est \"facile\" à apprendre")
```

ou alors à la place des doubles guillemets de la commande print(), nous remplacerons par des guillemets unitaires:

```
print('Le Python est "facile" à apprendre')
```

b) Considérons que nous voulons afficher cette phrase: «Le Python est mon langage préféré.» sur deux lignes: première ligne: Le Python est; deuxième ligne: mon langage préféré.

Alors, nous avons la première possibilité:

print("Le Python est\nmon langage préféré. ");

la deuxième possibilité est:

print("Le Python est");

print("mon langage préféré. ").

c) Considérons que nous voulons ajouter dans une phrase existante, la valeur d'une variable quelconque. Par exemple: La note totale est de …; le vide marqué par les symboles … doivent être représentées par une valeur qui se trouve dans une variable **S** – par exemple. Alors nous aurions:

print(f"La note totale est de {S}")

print("La note totale est de "+str(S))

Si **S** était une variable de type String (c'est-à-dire **str**), alors nous écrirons:

print("La note totale est de "+S)

Cela veut dire que la fonction **str()** transforme la valeur en «String» – chaîne de mots ou de symboles. La lettre ***f ou F*** represente une chaîne de caractères littérale formatée.

d) Considérons le même exemple que celui de la partie «c», nous voulons ajouter dans une phrase existante, la valeur d'une variable quelconque. Par exemple: La note totale est de …; le vide marqué par les symboles entre guillemets «…» doit être représenté par une valeur qui se trouve dans la variable **T** par exemple. Pour y parvenir, on utilise le symbole de pourcentage «**%**» ou les accolades «**{}**». Alors nous aurions les possibilités suivantes selon le type:

variante 1 : T – est un entier naturel

T=5

messag = ("La note totale est de **%d** – Excellent" %(T))

print(messag)

Comme résultat, on aura:

La note totale est de 5 – Excellent

variante 2 : T – est un nombre réel

T=4.85

messag = ("La note totale est de **%f** – Excellent" %(T))

print(messag)

```
# Comme résultat, on aura:
# La note totale est de 4.850000 – Excellent
```

variante 3 : T – est un nombre réel, avec un nombre de chiffre limité après la virgule(le point)

```
T=4.85
messag = ("La note totale est de %.3f – Excellent" %(T))
print(messag)
# Comme résultat, on aura 3 chiffres après la virgule(le point)
# La note totale est de 4.850 – Excellent
```

variante 4 : T – est une chaîne de caractères

```
T="4.85" # ou  T=4.85
messag = ("La note totale est de %s – Excellent" %(T))
print(messag)
# Comme résultat, on aura:
# La note totale est de 4.85 – Excellent
```

variante 5 : (*Revenir sur cette exemple après avoir appris la partie suivante sur l'introduction des données*)

```
nom = input('Introduire votre prénom: ')
matière= input('Introduire une matière étudiée: ')
note = 4
print("Je m'appelle {0}. J'ai reçu la note {2} pendant mon examen en {1} à l'université.".format(nom, matière, note))
# Il est équivalent à:
print("Je m'appelle",nom,". J'ai reçu la note",note,"pendant mon examen en",matière, " à l'université.")

#Résultat:
Introduire votre prénom: Alice
Introduire une matière étudiée: Informatique
```

Je m'appelle Alice. J'ai reçu la note 4 pendant mon examen en Informatique à l'université.

Je m'appelle Alice . J'ai reçu la note 4 pendant mon examen en Informatique à l'université.

e) Effectuer les opérations pendant la procédure d'affichage

```
a=13.8
b=16.4
print("La valeur de 'a' est %s, et celle de 'b' est %s. Nous pouvons même faire une somme de 'a' et 'b' qui donne c=%s. "%(a, b,a+b))
#Résultat:
```

La valeur de 'a' est 13.8, et celle de 'b' est 16.4. Nous pouvons même faire une somme de 'a' et 'b' qui donne c=30.2.

Remarque:

En utilisant les combinaisons de caractères suivants ***%s***, ***%d***, ***%f,*** nous devons retenir que les lettres ***s***, ***d***, ***f*** désignent respectivement les types de données suivants: chaîne, entier naturel et nombre réel.

Introduire les données après exécution du code:

Comment demander à l'utilisateur d'introduire une donnée ***X***, qui va se sauvegarder dans une autre variable Y quelconque en mode interactif?

Pour cela, nous pouvons utiliser la commande ***input()***; par exemple:

```
Y = input("X = ")
```

A l'écran, le logiciel nous demandera d'introduire la valeur de ***X*** qui sera gardée dans la variable ***Y***. Notons que les données introduites par la fonction ***input()*** sont considérées comme étant une chaîne de symboles de type **str()**.

Exemple

Essayez d'utiliser les informations obtenues pour avoir le nom, prénom et l'age de l'utilisateur. Après avoir obtenu ses informations, afficher sur l'écran le message en une phrase qui atteste à l'utilisateur

d'avoir bien reçu ces données en lui donnant son année de naissance et tout en lui rappelant son nom et prénom.

Réponse:

```
nom = input("Introduisez votre nom: ")
prenom = input("Introduisez votre prénom: ")
age = input("Introduisez votre age ici: ")
annee = 2020-int(age)
print("Monsieur "+nom+" "+prenom+", vous qui êtes né en "+str(annee)+", avons bel et bien reçu vos données.")
```

ou alors

```
nom = input('Introduisez votre nom: ')
prenom = input('Introduisez votre prénom: ')
age = input('Introduisez votre age ici: ')
annee = 2020-int(age)
print('Monsieur '+nom+' '+prenom+', vous qui êtes né en '+str(annee)+', avons bel et bien reçu vos données.')
```

Suite à la page 57

RU I.2. Различные операторы в Python

RU I.2.1. Основные операторы

Рассмотрим два числа в следующих переменных: m=27 и n=3. *res* – это переменная, содержащая результат выполненных операций.

Сложение: res = 27+3
m=m+n, что эквивалентно m+=n

вычитание: res = 27–3
m=m–n, что эквивалентно m–=n

умножение: res = 27*3
m=m*n эквивалентно m*=n

деление: res= 27/3
m=m/n, что эквивалентно m/=n

остаток от деления: res = 27%3
m=m%n эквивалентно m%=n

деление и остаток одновременно: res = divmod (27,3)

Целая часть результата деления
res = m // n
повышение степени: res = 27**3
m=m**n эквивалентно m**=n

Присвоить знак минус значению (унарный минус): m=–m

Округление в меньшую сторону
import math
res = math.flood(2.78) = 2

Округление в большую сторону
import math
res = math.ceil(2.28) = 3

RU I.2.2. Операторы сравнения, двоичные и арифметические

Оператор (см. табл. 1.1) представлен специальным символом или зарезервированным словом, которое можно использовать для выполнения операции между операндами (переменными).

Таблица 1.1 – Разные операторы в Python

Операторы	Название	В какой инструкции использовать ?
==	сравнение	if, while, …
!=	не равно	if, while, …
<	меньше	if, while, …
>	больше	if, while, …
<=	меньше или равно	if, while, …
>=	больше или равно	if, while, …

Продолжение таблицы 1.1

& или and	и	двоичный
| или or	или	двоичный
~ или not	отрицание	двоичный
is	существует	двоичный
is not	не существует	двоичный
in	внутри	двоичный
not in	не внутри (не в)	двоичный
<<	Нулевой заполненный Сдвиг влево	двоичный
>>	Смещение вправо	двоичный
+		арифметический
-		арифметический
*		арифметический
/		арифметический
%		арифметический
**		арифметический
//		арифметический

RU I.2.3. Операторы присваивания

Операторы присваивания используются для записи значения в переменную. Эти операторы находятся в таблице 1.2.

Таблица 1.2 – Разные операторы присваивания

Операторы	Примеры	Соответствует
=	y = 10	-
-=	y -=10	y = y - 10
+=	y +=10	y = y + 10
*=	y *=10	y = y * 10
**=	y **=10	y = y ** 10
/=	y /=10	y = y / 10
//=	y //=10	y = y // 10
&=	y &=10	y = y & 10
|=	y |=10	y = y | 10
^=	y ^=10	y = y ^ 10
%=	y %=10	y = y % 10

Продолжение таблицы 1.2

Операторы	Примеры	Соответствует
>>=	y >>=10	y = y >> 10
<<=	y <<=10	y = y << 10

Продолжение на странице 60

EN I.2. Various operators in Python

EN I.2.1. Basic operators

Consider two numbers in the following variables: m=27 and n=3. *res* is a variable containing the result of the operations performed.

Addition: res = 27+3
m=m+n, which is equivalent to m+=n

subtraction: res = 27–3
m=m-n, which is equivalent to m – =n

multiplication: res = 27*3
m=m*n is equivalent to m*=n

division: res= 27/3
m=m/n, which is equivalent to m/=n

remainder of division: res = 27%3
m=m%n equivalent to m%=n

division and remainder at the same time: res = divmod (27,3)

Integer part of the division result
res = m // n
increasing the degree: res = 27**3
m=m** n is equivalent to m**=n

Assign a minus sign to the value: m=–m
Rounding down

import math
res = math.flood(2.78) = 2

Rounding up
import math
res = math.ceil(2.28) = 3

EN I.2.2. Comparison operators, binary and arithmetic

An operator (see table 1.1) is represented by a special symbol or a reserved word that can be used to perform an operation between operands (variable).

Table 1.1 – Different operators in Python

Operators	Name	In what instructions to use ?
==	comparison is	if, while, …
!=	not equal to	if, while, …
<	less than	if, while, …
>	more than	if, while, …
<=	less than or equal to	if, while, …
>=	more than or equal to	if, while, …
& or and	and	binary
| = or	or	binary
~ или not	negation	binary
is	exists does	binary
is not	not exist	binary
in	inside	binary
not in	not inside (not in)	binary
<<	zero filled	binary
>>	left shift	binary
+		arithmetic
-		arithmetic
*		arithmetic

Continuation of table 1.1

Operators	Name	In what instructions to use ?
/		arithmetic
%		arithmetic
**		arithmetic
//		arithmetic

EN I.2.3. Assignment Operators

Assignment operators are used to write a value to a variable. These operators are found in table 1.2.

Table 1.2 – Different assignment operators

Operators	Examples	Respond
=	y = 10	-
-=	y -=10	y = y - 10
+=	y +=10	y = y + 10
*=	y *=10	y = y * 10
**=	y **=10	y = y ** 10
/=	y /=10	y = y / 10
//=	y //=10	y = y // 10
&=	y &=10	y = y & 10
|=	y |=10	y = y | 10
^=	y ^=10	y = y ^ 10
%=	y %=10	y = y % 10
>>=	y >>=10	y = y >> 10
<<=	y <<=10	y = y << 10

Continued on page 63

FR I.2. Les différents opérateurs dans Python

FR I.2.1. Opérateurs de base

Considérons deux nombres dans les variables suivantes: m=27 et n=3. ***res*** est la variable contenant le résultat des opérations effectuées.

Addition: *res* = 27+3
m=m+n équivalent à m+=n

soustraction: *res* = 27-3
m=m–n équivalent à m–=n

multiplication: *res* = 27*3
m=m*n équivalent à m*=n

division: *res*= 27/3
m=m/n équivalent à m/=n

reste de la division: *res* = 27%3
m=m%n équivalent à m%=n

division et reste en même temps: *res* = divmod(27,3)

La partie entière du résultat de la division
res = m//n
élévation en puissance: *res* = 27**3
m=m**n équivalent à m**=n

attribuer le signe moins à une valeur: m=–m

arrondir du coté inférieur
import math
res = math.flood(2.78) = 2

arrondir du coté supérieur
import math
res = math.ceil(2.28) = 3

FR I.2.2. Les opérateurs de comparaison, binaires et arithmétiques

Un opérateur (voir table 1.1) est représenté par un symbole special ou un mot réservé que l'on peut utiliser pour effectuer une opération entre des opérandes (variable).

Table 1.1 – Les différents opérateurs

Opérateurs	Nom	Dans quelle instruction utiliser ?
==	comparaison	if, while, …
!=	différent	if, while, …
<	inférieur	if, while, …
>	supérieur	if, while, …
<=	inférieur égale	if, while, …
>=	supérieur ou égale	if, while, …
& ou and	et	binaire
| ou or	ou	binaire
~ ou not	négation	binaire
is	est	binaire
is not	n'appartient pas	binaire
in	dans	binaire
not in	pas dans	binaire
<<	zéro rempli; décalage à gauche	binaire
>>	décalage à droite	binaire
+		arithmétiques
-		arithmétiques
*		arithmétiques
/		arithmétiques
%		arithmétiques
**		arithmétiques
//		arithmétiques

FR I.2.3. Les opérateurs d'affectation

Les opérateurs d'affectation sont utilisés pour écrire une valeur dans une variable. Ces opérateurs se trouvent à la table 1.2.

Table 1.2 – Les exemples d'opérateurs d'affectation

Opérateurs	Exemple	Equivalent
=	y = 10	-
-=	y -=10	y = y - 10
+=	y +=10	y = y + 10
*=	y *=10	y = y * 10
**=	y **=10	y = y ** 10

Suite de la table 1.2

/=	y /=10	y = y / 10
//=	y //=10	y = y // 10
&=	y &=10	y = y & 10
|=	y |=10	y = y | 10
^=	y ^=10	y = y ^ 10
%=	y %=10	y = y % 10
>>=	y >>=10	y = y >> 10
<<=	y <<=10	y = y << 10

Suite à la page 65

RU I.3. Условный оператор

Используется основная команда «**if**». Вот различные случаи:

Условие if (означает “если”)

Пример №1 Предположим, что мы хотим определить, является ли введенное число x кратным двум. Тогда мы будем иметь следующие коды:

```
# Попросить исполнителя ввести любое значение
x = input('ввести любое число x = ')
# Определить остаток деления этого числа на 2
reste = float(x)%2
# Проверьте, если этот остаток равен нулю (синтаксис if)
if (reste == 0):
   print(f'Число {x} кратно 2')
   # Напечатать фразу : «число x кратно 2»
   # (только тогда,когда, остаток равен нулю).
```

Условие if ... else (означает “если ... иначе”) ;

Пример №2 Добавим продолжение к упражнению в примере 1: нам необходимо отобразить сообщение «число x не кратно двум» на экране, если условие не удовлетворяется (остаток от деления не равен 0). Для этого необходим следующий синтаксис описания:

```
x = input('ввести любое число x = ')
reste = float(x)%2
if (reste == 0):
   print(f'Число {x} кратно 2')
else:
   print(f" число {x} не кратно 2")
   # Распечатать сообщение: «Число x не кратно 2»
```

Условие if ... elif ... else (означает “если ... иначе если ... иначе ... ”)

Пример №3 Чтобы понять этот тип условия, попробуем решить еще одно упражнение. Предположим, что мы хотим определить окончательную оценку студента по следующим трём критериям:

- если начальная оценка (полученная от учителя) строго больше 10, то она увеличивается на 4;
- если оценка находится в интервале [5, 10], то она увеличивается на 3;
- если она меньше 5, то отнимается 2 балла.

Прежде чем решить это упражнение, необходимо сначала построить алгоритм, показанный на рисунке №1.1, состоящий из 7 блоков.

Упражнение №1 с решением

Написать код в python, который решает квадратичное уравнение в виде: $ax^2+bx+c=0$, зная, что, имеет$\neq 0$. (Использовать метод дискриминанта).

Решение находится в главе VI

Упражнение №1.2

Предположим, что у нас имеется три разных целых числа (***a***, ***b*** и ***c***). Напишите код, который позволяет найти число (по его величине), которое лежит между двумя другими. Запрещается сравнивать три числа подряд (например, ***a<b<c***). Можно только сравнивать их по два (из предыдущего примера мы сначала получим сравнение между ‘***a***’ и ‘***b***’, а потом ‘***b***’ и ‘***c***’ или наоборот).

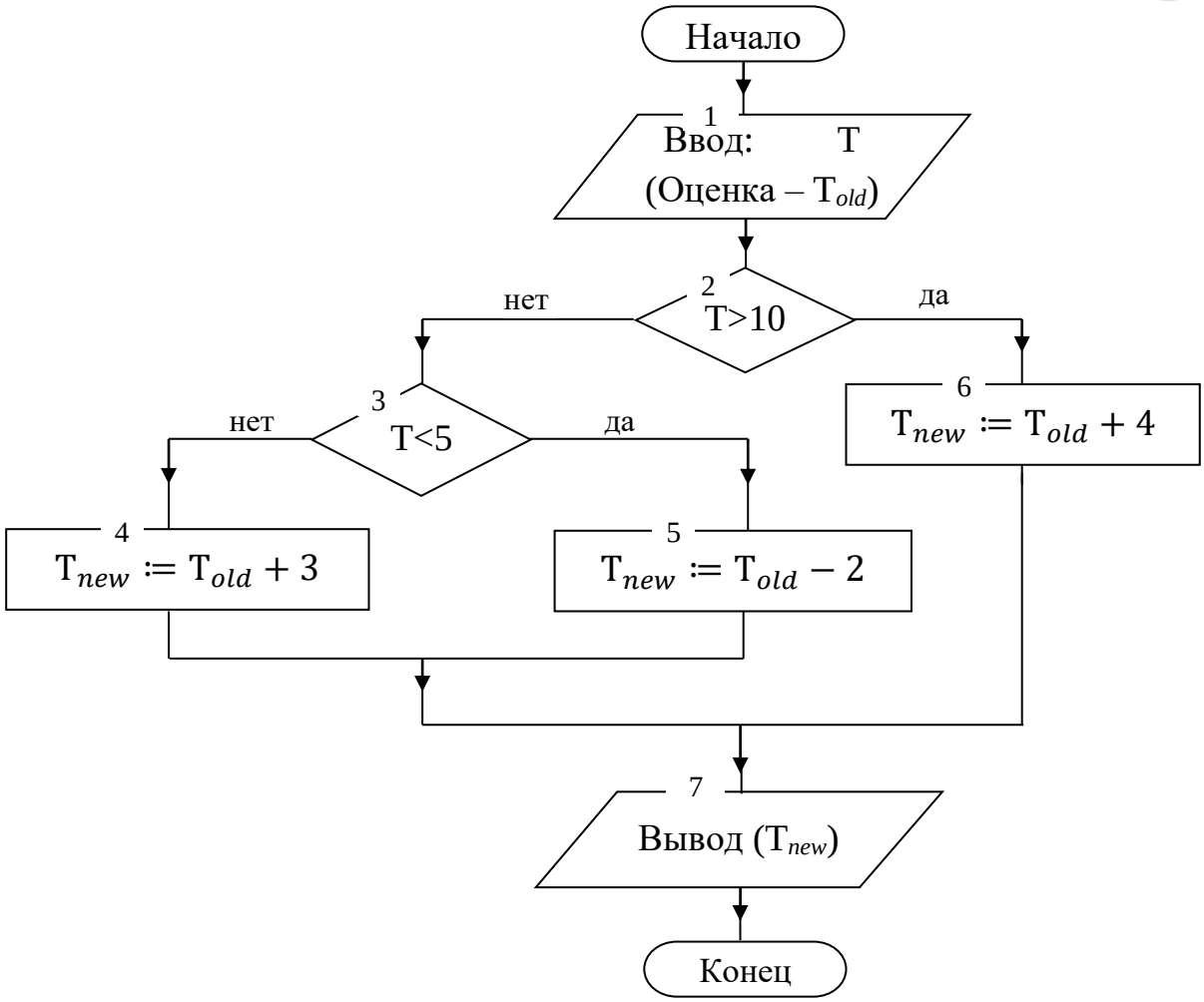


Рис. 1.1. Иллюстрация к решению примера №3

С Python имеется следующее решение:

```
note = int(input("Введите оценку студента = "))
if (note>10):
    note+=4
elif (note<5):
    note-=2
else:
    note+=3
print(f" Окончательная оценка студента – {note}.")
```

Блок-схема №1.1
Блок 1
Блок 2
Блок 6
Блок 3
Блок 5
–
Блок 4
Блок 7

Продолжение на странице 68

EN I.3. Conditional operator

The basic «**if**» command is used. Here are the different cases:

If condition

Example 1 Suppose we want to determine whether the entered number ***x*** is a multiple of two. Then we will have the following codes:

```
# Ask the performer to enter any value
x = input('enter any number x = ')
# Determine the remainder of dividing this number by 2
reste = float(x)%2
# Check if this remainder is zero (the syntax of the "if")
if (reste == 0):
   print(f'The number {x} is a multiple of 2')
   # Print the phrase: "The number x is a multiple of 2"
   # (only if, the remainder is zero).
```

The «if ... else» condition

Example 2 Adding a continuation to the exercise in example 1: we need to display the message «the number *x* is not a multiple of two» on the screen if the condition is not satisfied (the remainder of the division is not equal to 0). This requires the following description syntax:

```
x = input('Enter any number x = ')
reste = float(x)%2
if (reste == 0):
   print(f'The number {x} is a multiple of 2')
else:
   print(f"The number {x} is not a multiple of 2")
   # Print the message: «The number x is not a multiple of 2»
```

The «if ... elif ... else» condition

Example 3 To understand this type of condition, let's try to solve another exercise. Suppose we want to determine the final grade of a student by the three following criteria:

- if the initial grade (received from the teacher) is strictly greater than 10, then it increases by 4;
- if the score is in the range [5, 10], then it is increased by 3;
- if it is less than 5, then 2 points are deducted.

Before solving this exercise, you must first build the algorithm shown in figure 1.1, which consists of 7 blocks.

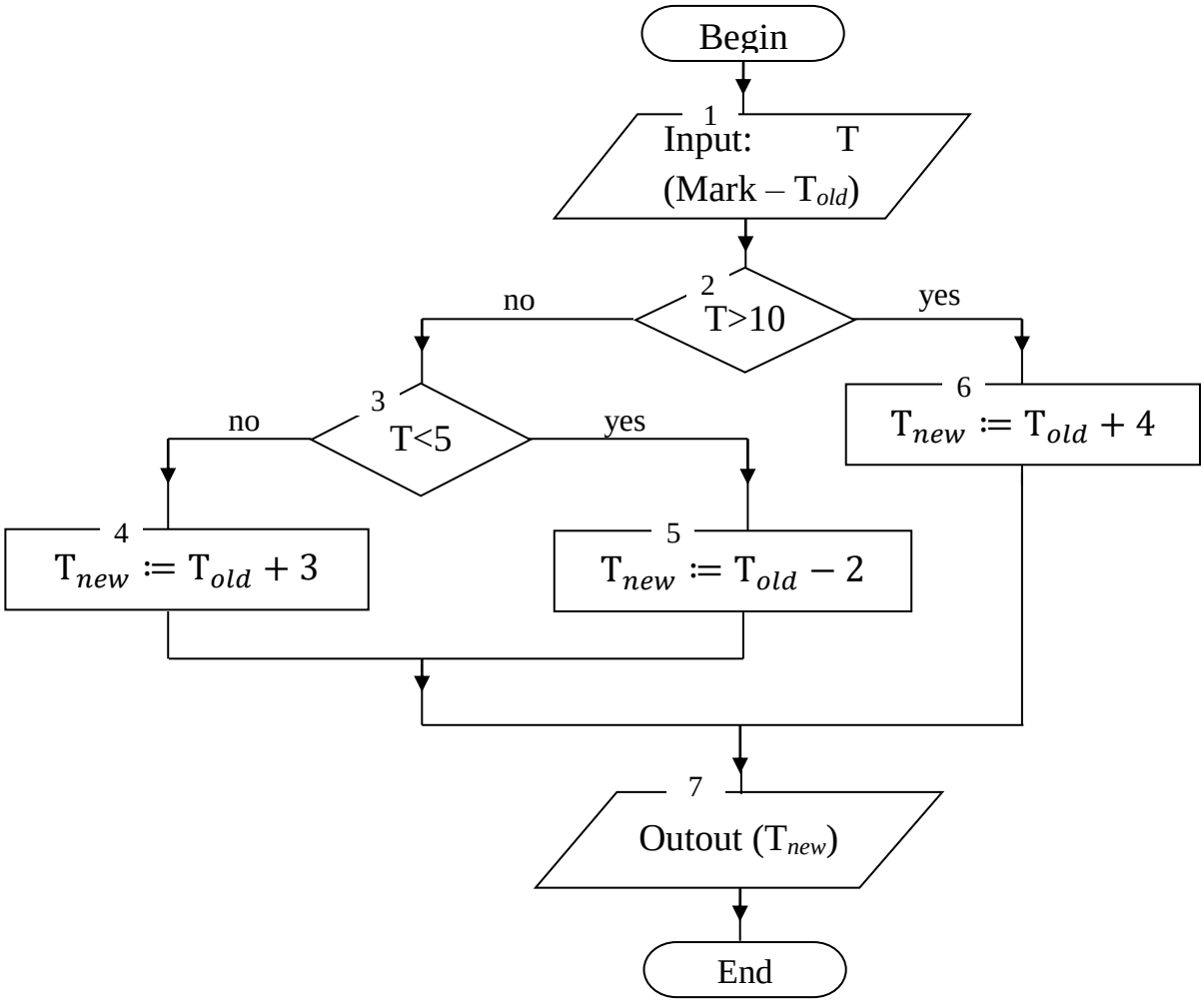


Fig. 1.1. Illustration to the solution of Example 3

With Python, there is the following solution:

Block diagram 1.1

```
note = int(input("Enter the student's grade = "))
if (note>10):
    note+=4
elif (note<5):
    note–=2
else:
    note+=3
print(f"Final mark of the student – {note}.")
```

Block 1
Block 2
Block 6
Block 3
Block 5
–
Block 4
Block 7

Exercise 1.1 with a solution

Write code in python that solves a quadratic equation in the form: $ax^2+bx+c=0$, knowing that, $a\neq0$. (Use the discriminant method).

The resolution can be found in Chapter VI

Exercise 1.2

Suppose we have three different integers (***a***, ***b***, and ***c***). Write code that allows you to find a number (by its value) that lies between the other two. It is forbidden to compare three consecutive numbers (for example, ***a<b<c***). You can only compare them two at a time (from the previous example, we will first get a comparison between ***a*** and ***b***, and then ***b*** and ***c***, or vice versa).

Continued on page 71

FR I.3. Instruction conditionnelle

La principale commande utilisée est «**if**». Voici les différents cas:

La condition if (veut dire “si”)

Exemple 1 Considérons que nous voulons déterminer si le nombre x introduit est un multiple de deux, alors nous aurons les codes suivants:

```
# Demander à l'exécuteur d'introduire une valeur quelconque
x = input('introduire un nombre quelconque x = ')
# Déterminer le reste de la division de ce nombre par 2
reste = float(x)%2
#Vérifier si ce reste est égale à zéro (syntaxe de if)
if (reste == 0):
   print(f'le nombre {x} est un multiple de 2')
   # Imprime la phrase : «le nombre x est un multiple de 2»
   # (si et seulement si, le reste est égale à zéro).
```

La condition if…else (veut dire "si…sinon")

Exemple 2 Ajoutons une suite à l'exercice de l'exemple 1: nous voulons également afficher un message (Le nombre x n'est pas un multiple de deux) sur l'écran, dans le cas où la condition n'est pas vérifiée (le reste de la division n'est pas 0). Nous aurons la syntaxe suivante:

```
x = input('introduire un nombre quelconque x = ')
reste = float(x)%2
if (reste == 0):
   print(f'Le nombre {x} est un multiple de 2')
else:
   print(f"Le nombre {x} n'est pas un multiple de 2")
   # Imprime la phrase: «Le nombre x n'est pas un multiple de 2»
```

La condition if…elif…else (veut dire "si…sinon si… sinon")

Exemple 3 Pour comprendre cette condition, essayons de résoudre un autre exercice. Considérons que nous voulons déterminer la note finale de l'étudiant selon les trois critères suivants:

- si la note initiale (reçue de l'enseignant) est strictement supérieure à 10, alors elle est augmentée de 4;
- si la note se trouve dans l'intervalle [5, 10], alors elle est augmentée de 3;
- si elle est inférieure à 5, alors nous pénalisons l'étudiant en retranchant 2 points.

Avant de résoudre cet exercice, il faudra d'abord construire l'algorithme illustré à la figure №1.1 constitué de 7 blocs.

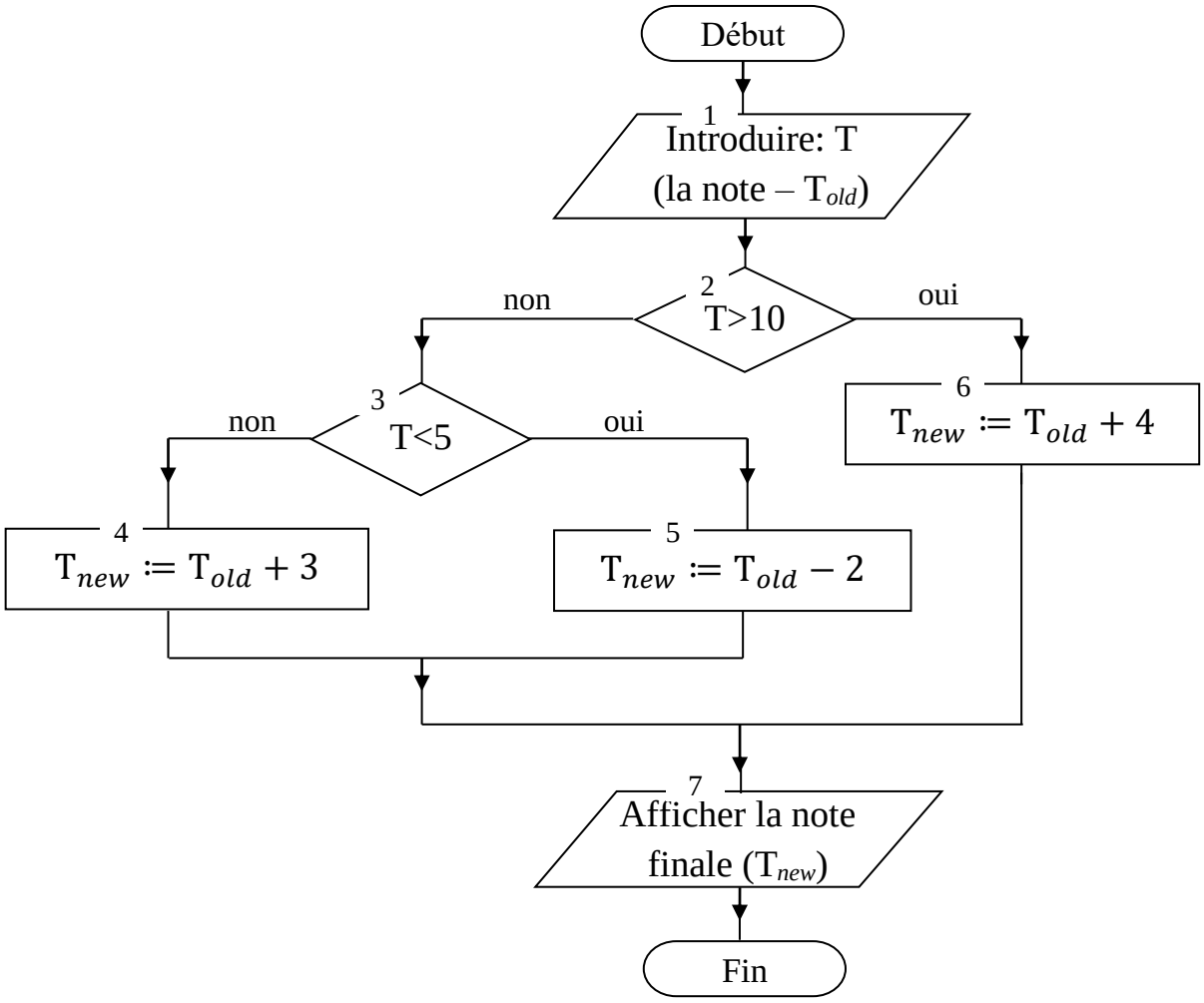


Fig. 1.1. Illustration de la résolution de l'exemple 3

Avec Python, nous avons la solution suivante:

```
note = int(input("Introduire la note de l'étudiant = "))
if (note>10):
   note+=4
elif (note<5):
   note-=2
else:
   note+=3
print(f"La note finale de l'étudiant est de {note}.")
```

Schéma №1.1
Bloc 1
Bloc 2
Bloc 6
Bloc 3
Bloc 5
–
Bloc 4
Bloc 7

Exercice 1.1 corrigé

Écrire un code en python qui permet de résoudre l'équation du second degré (équation quadratique) $ax^2+bx+c=0$, sachant que, $a\neq0$. (Utiliser la méthode du discriminant).

Voir résolution au chapitre VI

Exercice 1.2

Considérons que nous avons trois nombres entiers différents. Écrire le code permettant de trouver le nombre (par sa grandeur) qui se trouve entre les deux autres. Il est interdit de comparer trois nombres successivement (Par exemple a<b<c). Vous ne pouvez que les comparer deux par deux (D'après l'exemple précédent, nous aurons d'abord la comparaison entre a et b, et après b et c ou vice versa).

Suite à la page 74

RU I.4. Инструкции по формированию алгоритмических циклов

Напомним, что цикл – это процедура, предназначенная для выполнения части кода несколько раз подряд. Иногда бывает так, что нам приходится вычислять значения в соответствии с одними и теми же условиями для разных значений.

Для этого необходимо выполнить следующие действия:

1) указать в начале цикла начальное значение переменной, которое будет постепенно меняться.

2) Изменять переменную перед каждым новым повторением цикла;

3) Поставить условие, чтобы проверить, должен ли цикл повторяться или завершаться.

4) Управлять циклом в соответствии с его состоянием (продолжать или направить его на выход).

Основными используемыми командами являются «**for**» и «**while**».

Синтаксис цикла с «for» (что означает – «для») – как он работает?

Чтобы лучше понять, давайте решим следующие примеры:

Пример №1 Вычислить и вывести на печать значения функции $g(x) = \frac{a^3}{2a^2+x^2}$. при значении x, изменяющемся от 0 до 25 с шагом 2, a – константа не равна 0.

Чтобы лучше решить, построим схематический алгоритм (блок-схему), который облегчит понимание кода (см. рис. №1.2).

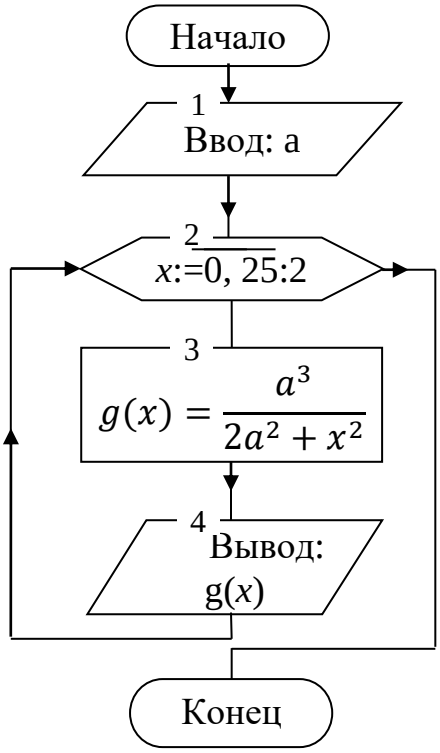


Рис. № 1.2. Иллюстрация к решению примера 1 с помощью циклического блока

Коды:

```
a = int(input(' Введите значение a = '))
for x in range(0,26,2):
    g = (a**3)/(2*(a*a)+(x*x))
    print(f"Для x = {x}, g({x}) = {g}")
```

Блок-схема №1.2
Блок 1
Блок 2
Блок 3
Блок 4

Строка блока 2 будет интерпретироваться таким образом: для каждого значения x, которое строится или находится в интервале от 0 до 26 (кроме 26) и имеет приращение 2, выполните строку блока 3.

Вторичные объяснения для «for»:

for *x* in range(3): # значит, x = цифры от 0 до 2

for *x* in range(4,9): # значит, x = цифры от 4 до 8

for *x* in range(1,10,2): # значит, x = цифры 1,3,5,7 и 9.

Пример №2 Предположим, что в какой-то таблице имеем список, содержащий дни недели, и эту таблицу нам нужно отобразить на экране. Тогда код будет следующим:

```
spisok = ['Понедельник', 'Вторник', 'Среда', 'Четверг', 'Пятница', 'Суббота', 'Воскресенье']
print(' Дни недели:')
y=1
for x in spisok:
    if y==1: print(f' {y}-ый день недели является “{x}”\n')
    else: print(f'{y}-ой(-ый) день недели является “{x}”\n')
    y+=1
```

Синтаксис цикла с «while» (что означает – «пока») – как он работает?

Чтобы лучше понять, давайте изменим пример №1.

Пример №3 Вычислить и вывести на печать значения функции $g(x) = \frac{a^3}{2a^2+x^2}$. при значении x, изменяющемся от 0 до 25 с шагом 0.5, a – константа не равна 0.

Схематический алгоритм (блок-схема) остается таким же, что и на рисунке №1.2. С помощью инструкции «**for**» мы не можем решить его из-за приращения, которое является действительным числом. Инструкция «**for**» использует только целочисленное значение для этой части. Это одна из особенностей, которая отличает его от

инструкции **«while»**. С помощью инструкции **«while»** мы вынуждены указывать в начале цикла начальное значение переменной, которое будет меняться постепенно, а после менять эту переменную перед каждым новым повторением в цикле. Таким образом, мы находим, что инструкция **«for»** более «умная», чем инструкция **«while»**.

Коды:

```
a = int(input(' Введите значение a ='))
x = 0
while x<=25:
    g = (a**3)/(2*(a*a)+(x*x))
    print(f"для x = {x}, g({x}) = {g}")
    x += 0.5
```

Блок-схема №1.2

Блок 1
Блок 2
Блок 2
Блок 3
Блок 4
Блок 2

Продолжение на странице 77

EN I.4. Instructions for forming algorithmic loops

Recall that a loop is a procedure designed to execute a piece of code several times in a row. Sometimes it happens that we have to calculate values according to the same conditions for different values.

To do this, follow these steps:

1) specify at the beginning of the loop the initial value of the variable, which will gradually change.

2) Change the variable before each new iteration of the loop;

3) Set a condition to check whether the loop should be repeated or completed.

4) Control the loop according to its state (continue or direct it to exit).

The main commands used are «**for**» and «**while**».

Loop syntax with «for» – how does it work?

To better understand, let's solve the following example:

Example 4 Calculate and print the values of the function $(x) = \frac{a^3}{2a^2+x^2}$, if the value of x varies from 0 to 25 in increments of 2, the a – constant is not equal to 0.

To better solve the problem, we will build a schematic algorithm (flowchart) that will make it easier to understand the code (see Fig. 1.2).

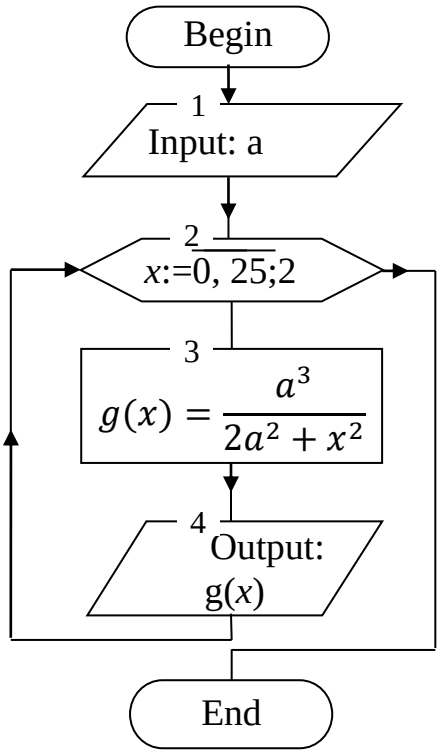


Fig. 1.2. Illustration of the solution of Example 1 using a cyclic block

Program code	Block diagram 1.2

a = int(input('Enter a value *a* = '))
for *x* in range(0,26,2):
  g = (a**3)/(2*(a*a)+(*x**x))
  print(f"For x = {*x*}, g({*x*}) = {g}")
	Block 1
	Block 2
	Block 3
	Block 4

The block 2 string will be interpreted as follows: for each value of x that is plotted or is in the range from 0 to 26 (except 26) and has an increment of 2, execute the block 3 string.

Secondary explanations for Statement «for»:

for *x* in range(3): # so *x* = values from 0 to 2
for *x* in range(4,9): # so *x* = values from 4 to 8
for *x* in range(1,10,2): # so *x* = values 1,3,5,7 and 9.

Example2 Suppose that in some table we have a list containing the days of the week, and we need to display this table on the screen. Then the code will be as follows:

```
spisok = ['Monday', 'Tuesday', 'Wednesday', 'Thursday', 'Friday',
'Saturday', 'Sunday']
print('The days of the week: ')
y=1
for x in spisok:
    if y==1: print(f'The {y}-st day of the week is “{x}”\n')
    else: print(f'The {y} day of the week is “{x}”\n')
    y+=1
```

Loop syntax with «while» – how does it work?

To better understand, let's change example №1.

Example 3

Calculate and print the values of the function $(x) = \frac{a^3}{2a^2+x^2}$, if the value of x varies from 0 to 25 in increments of 0.5, the *a* – constant is not equal to 0.

The schematic algorithm (flowchart or block diagram) remains the same as in Figure 1.2. With the statement «**for**», we can't solve it because

of the increment, which is a real number. The «**for**» operator only uses an integer value for this part. This is one of the features that distinguishes it from the **«while»** operator. Using the **«while»** operator, we are forced to specify the initial value of the variable at the beginning of the loop, which will change gradually, and then change this variable before each new repetition in the loop. So we find that the **«for»** operator is more **«smart»** than the **«while»** operator.

Program code	Block diagram 1.2
a = int(input('Enter a value *a* ='))	Block 1
x = 0	Block 2
while *x*<=25:	Block 2
g = (a**3)/(2*(a*a)+(*x***x*))	Block 3
print(f"For x = {*x*}, g({*x*}) = {g}")	Block 4
x += 0.5	Block 2

Continued on page 88

FR I.4. Instructions pour former des boucles

Rappelons qu'une boucle est une procédure destinée à exécuter une portion de code plusieurs fois de suite. Il arrive parfois que nous devons calculer plusieurs fois un exercice avec les mêmes conditions pour différentes valeurs.

Pour cela, il faut procéder de la manière suivante:

1) Indiquer au début de la boucle la valeur initiale de la variable qui va changer progressivement.

2) Changer la variable avant chaque nouvelle répétition dans la boucle;

3) Mettez une condition pour vérifier si le cycle doit être répété ou terminé.

4) Contrôler la boucle en fonction de son état (continuer ou le diriger vers la sortie si la condition n'est plus valable).

Les principales commandes utilisées sont «**for**» et «**while**».

Syntaxe de la boucle «for» (qui veut dire – «pour») – comment elle fonctionne?

Pour mieux comprendre, résolvons cet exercice en forme d'exemple.

Exemple 1 Calculer et afficher les valeurs *x* de la fonction $g(x) = \frac{a^3}{2a^2+x^2}$. Les valeurs de *x* varient de 0 à 25 par incrément de 2. ***a*** est une constante qui n'est pas 0.

Pour mieux résoudre, construisons l'algorithme schématique (organigramme) qui facilitera la compréhension du code (voir la figure №1.2).

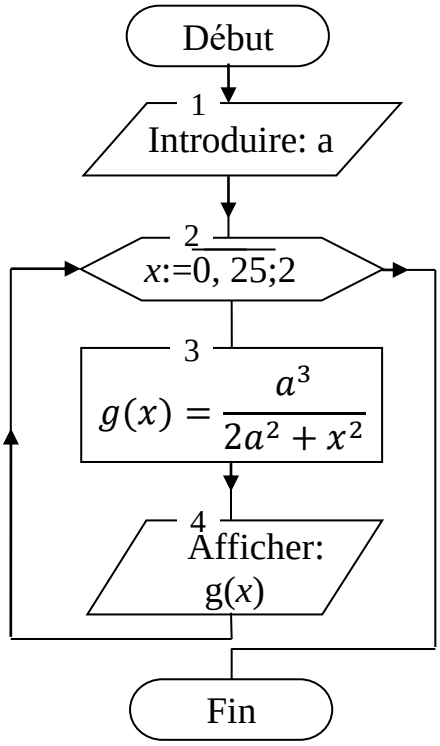


Fig. 1.2 Illustration de la résolution de l'exemple 1 en utilisant la boucle

Le code:

	Schéma №1.2

a = int(input('Introduire la valeur de a = '))
for *x* in range(0,26,2):
  g = (a**3)/(2*(a*a)+(*x**x))
  print(f"Pour x = {*x*}, g({*x*}) = {g}")
	Bloc 1
	Bloc 2
	Bloc 3
	Bloc 4

La ligne du bloc 2 sera interprétée de cette manière: pour chaque valeur de *x* qui rangée ou alors qui se trouve dans l'intervalle de 0 à 26 (26 exclu) et ayant un incrément de 2, exécuter la ligne du bloc 3.

Explications secondaires pour «for»:
for *x* in range(3): # veut dire *x* = aux chiffres de 0 à 2
for *x* in range(4,9): # veut dire *x* = aux chiffres de 4 à 8
for *x* in range(1,10,2): # veut dire *x* = aux chiffres 1,3,5,7 et 9.

Exemple 2 Considérons avoir une liste contenant les jours de la semaine dans un tableau quelconque que nous devons juste afficher sur l'écran. Alors le code sera le suivant:

```
la_liste = ['lundi', 'mardi', 'mercredi', 'jeudi', 'vendredi', 'samedi', 'dimanche']
print('Les jours de la semaine sont:')
y=1
for x in la_liste:
    if y==1: print(f'Le {y}-ier jour de la semaine est {x}\n')
    else: print(f'Le {y}-ième jour de la semaine est {x}\n')
    y+=1
```

Syntaxe de la boucle «while» (qui veut dire – «tant que») – comment elle fonctionne?

Pour mieux comprendre, modifions l'exemple №1.

Exemple 3 Calculer et afficher les valeurs *x* de la fonction $g(x) = \frac{a^3}{2a^2+x^2}$. Les valeurs de *x* varient de 0 à 25 par incrément de 0.5. ***a*** est une constante qui n'est pas 0.

L'algorithme schématique (organigramme) reste le même que celui de la figure №2. Avec l'instruction «**for**», nous ne pouvons pas le résoudre à cause de l'incrément qui est un réel. L'instruction «**for**» utilise uniquement la valeur entière sur cette partie. C'est l'une des caractéristiques qui la différencie de l'instruction «**while**». Avec l'instruction «**while**», nous sommes obligés d'indiquer au début du cycle la valeur initiale de la variable qui va changer progressivement et après modifier cette variable avant chaque nouvelle répétition dans la boucle. Ainsi, nous constatons que l'instruction «for» est plus «intelligente» que l'instruction «**while**».

Le code:

	Schéma №1.2
a = int(input('Introduire la valeur de a = '))	Bloc 1
x = 0	Bloc 2
while *x*<=25:	Bloc 2
g = (a**3)/(2*(a*a)+(*x***x*))	Bloc 3
print(f"Pour x = {*x*}, g({*x*}) = {g}")	Bloc 4
x += 0.5	Bloc 2

Suite à la page 98

RU I.5. Обработка данных с помощью конструкции массивов

Существует четыре способа хранения данных в форме строки: через «*list*», «*tuple*», «*set*» и «*dictionary*».

«**list**» – это коллекция, которая упорядочена и может модифицироваться. Позволяет дублировать элементы. Пример применения «***list***»

```
maliste = ["я", "папа", "мама"] или
maliste = ['я', 'папа', 'мама']
```

Методы с «list» (см. таб. 1.3)

Таблица 1.3 – Различные методы в списке – «*list*»

Методы	Описания
append()	Добавляет элемент в конец списка
clear()	Удаляет все элементы из списка
copy()	Возвращает копию списка
count()	Возвращает количество элементов с указанным значением
extend()	Добавить элементы списка в конец текущего списка
index()	Возвращает индекс первого элемента с указанным значением
insert()	Добавляет элемент в указанную позицию
pop()	Удаляет элемент в указанной позиции
remove()	Удаляет элемент с указанным значением
reverse()	Разворачивает список
sort()	Сортирует список

Пример №1 Изменить порядок списка “*maliste*”.

```
maliste = ["я", "папа", "мама"]
maliste.reverse()
print(maliste)
```

```
Результат :
['мама', 'папа', 'я']
```

Пример №2 Добавить элемент “брат” после первого слова в данном списке “*maliste*”.

```
maliste = ["я", "папа", "мама"]
maliste.insert(1, "брат")
print(maliste)
```

```
Результат:
['я', 'брат', 'папа', 'мама']
```

«Tuple» представляет собой набор данных, который является упорядоченным и неизменяемым. Позволяет дублировать элементы. Этот тип использует круглые скобки вместо квадратных скобок. Пример применения «**tuple**»:

```
mytuple = ("я", "папа", "мама") или
mytuple = ('я', 'папа', 'мама')
```

Методы с «tuple» (см. таб. 1.4)

Таблица 1.4 – Различные методы с использованием «tuple»

Методы	**Описания**
count()	Возвращает количество раз, когда заданное значение происходит в «tuple»
index()	Выполняет поиск в «tuple» указанного значения и возвращает местоположение, где оно было найдено

Пример №1 Найдите в «tuple»: “*mytuple*” указанное значение элемента «папа» и верните его индекс, т. е. номер, где находится данный элемент.

```
mytuple = ("я", "папа", "мама")
x=mytuple.index("папа")
print(f"Номер элемента «{mytuple[1]}» в «tuple» = "+str(x))
```

```
Результат:
Номер элемента «папа» в «tuple» = 1
```

Примечание Метод ***index()*** создает исключение, если значение не найдено.

Пример №2 Найдите в «tuple» *mytuple*, сколько раз находится элемент «папа».

```
mytuple = ("я", "папа", "мама", "папа")
n=mytuple.count("папа")
print(f"Количество раз, когда встречается элемент «папа» = "+str(n))
```

```
Результат:
Количество раз, когда встречается элемент «папа» = 2
```

Как изменить значение в «tuple»?

«tuple» нужно преобразовать в список, чтобы его можно было изменить.

```
mytuple = ("я", "папа", "мама")
y = list(mytuple)
y[1] = "дядя"
mytuple = tuple(y)

print(mytuple)
```

```
Результат:
("я", "дядя", "мама")
```

«set» – это коллекция, которая не упорядочена и не индексирована. Нет дубликатов членов. Этот тип использует фигурные скобки вместо квадратных скобок или обычных. Пример применения «**set**»:

```
myset = {"я", "папа", "мама"} или
myset = {'я', 'папа', 'мама'}
```

Методы с использованием «set» (см. таб. 1.5)

Таблица 1.5 – Различные методы с использованием «set»

Методы	**Описания**
add()	Добавляет элемент в набор
clear()	Удаляет все элементы
copy()	Возвращает копию набора
difference()	Возвращает набор, содержащий разницу между двумя или более наборами
difference_update()	удаляет элементы из этого набора, которые также включены в другой указанный набор
discard()	Удаляет указанный элемент
intersection()	Возвращает набор, который является пересечением двух других рядов
intersection_update ()	удаляет элементы из этого множества, которые отсутствуют в других наборах, указанных в ***set()***
isdisjoint()	возвращает, если два множества имеют пересечение или нет
issubset()	возвращает, если какое-то подмножество находится в другом множестве
issuperset()	возвращает, если определенное подмножество находится в другом множестве
pop()	удаляет элемент из множества
remove()	удаляет указанный элемент
symmetric_difference()	возвращает множество с симметричными различиями двух наборов (множеств)
symmetric_difference_ update()	вставляет симметричные различия этого подмножества и другого
union()	возвращает множество, содержащее объединение множеств
update ()	обновляет исходное множество путём объединения других наборов

Пример №1 Печать элементов из заданного множества *myset*.

```
myset = {"я", "брат", "папа", "мама"}
for x in myset:
   print(x)
```

Результат (вариант 1):
папа
я
мама
брат

Результат (вариант 2):
брат
мама
папа
moi

Пример №2 Проверьте, находится ли элемент «брат» в множестве *myfamily*; если да, напечатать фразу: "У нас один и тот же отец и одна и та же мать!".

```
myfamily = {"я", "брат", "папа", "мама"}
if "брат" in myfamily:
   print("У нас один и тот же отец и одна и та же мать!")
```

Результат:
У нас один и тот же отец и одна и та же мать!

Пример №3 Ввести в существующее множество оценку «отлично» и результат вывести на экран.

```
atsenka = {"хорошо", "плохо", "удовлетворительно"}
appréciation.add("отлично")
print(atsenka)
```

Результат:
{'хорошо', 'отлично', 'плохо', 'удовлетворительно'}
один из возможных результатов.

«dictionary» – это коллекция, которая неупорядочена, редактируется и индексируется. Нет дубликатов членов. Они обозначены через фигурные скобки, кроме того, у них есть ключевые слова со значением.

```
aboutme_dict ={
  "familia": "Иванов",
  "imia": "Иван",
  "sexe": " мужской ",
  "vozrast": 30,
  "status": "женат"
}
print(aboutme_dict)
```

или ещё

```
aboutme_dict ={
  "familia": "Иванов",
  "imia": "Иван",
  "sexe": " мужской ",
  "vozrast": 30,
  "status": "женат"
}
print(aboutme_dict)
```

```
Результат:
{'familia': 'Иванов', 'имя': 'Ivan', 'sexe': 'мужской', 'vozrast': 30,
'status': 'женат'}
```

Пример №1 Получить значение объекта «vozrast», зная, что имеем «dictionary» *aboutme_dict* (см. предыдущий словарь) и вывести на экран.

```
znacheniye_1 = aboutme_dict["vozrast"]
print(f" Значение объекта 'vozrast' (1) равно = {znacheniye_1}")
# ou alors
znacheniye_2 = aboutme_dict.get("vozrast ")
print(f"Значение объекта 'vozrast' (2) равно = {znacheniye_2}")
```

```
Результат :
Значение объекта 'vozrast' (1) равно = 30
# ou alors
Значение объекта 'vozrast' (2) равно = 30
```

Пример №2 Вывести на экран ключевые слова, длина которых равна 6 символам (также вывести на экран найденное значение). При этом использовать входные данные из словаря *aboutme_dict* (см. предыдущий словарь).

Создаем цикл с использованием «for»

```
for cle in aboutme_dict:
    if (len(str(aboutme_dict[cle]))==6):
        print(f'Ключевое слово = {cle}')
        print(f'Значение ключевого слова "{cle}" является:
                                        {aboutme_dict[cle]}')
```

```
Результат:
Ключевое слово = familia
Значение ключевого слова "familia" является: Иванов
Ключевое слово = status
Значение ключевого слова "status" является: женат
```

Пример №3 Вывести на экран значения, длина которых равна 6 символам. Необходимо использовать входные данные из «dictionary» *aboutme_dict* (см. предыдущий словарь).

```
for val in aboutme_dict.values():
    if (len(str(val))==6):
        print(f'У нас здесь значение "{val}" без ключевого слова')
         print(f'Оно здесь: {aboutme_dict.values()}')
```

```
Результат:
У нас здесь значение "Иванов" без ключевого слова
Оно здесь: dict_values(['Иванов', 'Иван', 'мужской', 30, 'женат'])
У нас здесь значение "женат" без ключевого слова
Оно здесь: dict_values(['Иванов', 'Иван', 'мужской', 30, 'женат'])
```

Пример №4 Вывести на экран ключевые слова, длина которых равна 6 символам, другим способом (также вывести найденные значения). Используйте входные данные из «dictionary» *aboutme_dict* (см. предыдущий пример №2).

Создаем цикл с использованием «for» + метод items()

```
for cle, val in aboutme_dict.items():
    if (len(str(val))==6):
        print(f'Ключ = {cle}')
        print(f'Значение ключа "{cle}" является : {val}')
```

```
Результат:
Ключ = familia
Значение ключа "familia" является : Иванов
Ключ = status
Значение ключа "status" является : женат
```

Пример №5 Создать «dictionary», содержащий три маленькие Dictionary. Например, группа студентов с их соответствующими характеристиками.

Основной «dictionary» называется GS_2020. Маленькие словари: *Ivanova*, *Petrov* и *Sidorov*. Каждый из них должен иметь: тип студента (*стипендиат* или *нет*), пол(*мужской* или *женский*), национальность (*русский* или *иностранный*) и год рождения.

```
GS_2020 = {
  "Ivanova" : {
     "typ_stud"    : " стипендиат",
     "pol"           : "женский",
     "nationalnost"  : "русский",
     "god_rogd" : 2001
  },
  # далее…
```

```
  "Petrov" : {
     "typ_stud"    : " стипендиат",
     "pol"         : "мужской",
     "nationalnost"  : "иностранный",
     "god_rogd" : 2002
  },
  "Sidorov" : {
     "typ_stud"    : " стипендиат",
     "pol"         : "мужской",
     "nationalnost"  : "русский",
     "god_rogd" : 1999
  }
}
```

```
#Результат: GS_2020
{'Ivanova': {'typ_stud': 'стипендиат', 'pol': 'женский',
'nationalnost': 'русский', 'god_rogd': 2001}, 'Petrov': {'typ_stud':
'стипендиат', 'pol': 'мужской', 'nationalnost': 'иностранный',
'god_rogd': 2002}, 'Sidorov': {'typ_stud': 'стипендиат', 'pol':
'мужской', 'nationalnost': 'русский', 'god_rogd': 1999}}
```

Другой способ: применяется в случае, если вы хотите ввести три словаря, которые уже существуют в «dictionary»:

```
Ivanova = {
     "typ_stud"    : "стипендиат",
     "pol"         : "женский",
     "nationalnost"  : "русский",
     "god_rogd" : 2001
  }
Petrov = {
     "typ_stud"    : "стипендиат",
     # далее ...
```

```
...
    "pol"          : "мужской",
    "nationalnost"  : "иностранный",
    "god_rogd" : 2002
  }
Sidorov = {
    "typ_stud"   : "стипендиат",
    "pol"          : "мужской",
    "nationalnost"  : "русский",
    "god_rogd" : 1999
  }
GS_2020 = {
  "Ivanova" : Ivanova,
  "Petrov"  : Petrov,
  "Sidorov" : Sidorov
}
```

```
Результат: (то же, что и предыдущий)
GS_2020
{'Ivanova': {'typ_stud': 'стипендиат', 'pol': 'женский', 'nationalnost': 'русский', 'god_rogd': 2001}, 'Petrov': {'typ_stud': 'стипендиат', 'pol': 'мужской', 'nationalnost': 'иностранный', 'god_rogd': 2002}, 'Sidorov': {'typ_stud': 'стипендиат', 'pol': 'мужской', 'nationalnost': 'русский', 'god_rogd': 1999}}
```

Методы использования «dictionary» (см. таб. 1.6)

Таблица 1.6 – Различные методы с использованием «dictionary»

Методы	Описание
clear()	Удаляет все элементы из словаря
copy()	Возвращает копию словаря
fromkeys()	Возвращает словарь с указанными ключами и значением
get()	Возвращает значение указанного ключа

Продолжение таблицы 1.6

Методы	Описание
items()	Возвращает список, содержащий Tuple для каждой пары ключевых значений
keys()	Возвращает список, содержащий словарь ключей
pop()	Удаляет элемент с указанным ключом
popitem()	Удаляет последнюю вставленную пару ключ-значение
setdefault()	Возвращает значение указанного ключа. Если ключ не существует: вставьте ключ, с указанным значением
update()	Обновляет словарь с парами ключ-значение
values()	Возвращает список всех значений словаря

Продолжение на странице 108

EN I.5. Data processing using array construction

There are four ways to store data in the form of a string: through «*list*», «*tuple*», «*set*» and «*dictionary*».

“*list*” is a collection that is ordered and can be modified. Allows you to duplicate elements. Example of using «*list*»:

```
mylist = ["me", "dad", "mom"] or
mylist = ['me', 'dad', 'mom']
```

Methods with «list»: (see table 1.3)

Table 1.3 – The different methods of the list – «*list*»

Methods	**Description**
append()	Adds an item to the end of the list
clear()	Removes all items from the list
copy()	Returns a copy of the list
count()	Returns the number of elements with the specified value
extend()	Add list items to the end of the current list

Продолжение таблицы 1.3

Methods	Description
index()	Returns the index of the first element with the specified value
insert()	Adds an element to the specified position
pop()	Deletes an item at the specified position
remove()	Deletes an element with the specified value
reverse()	Expands the list
sort()	Sorts the list

Example 1 Change the order of the “mylist” list.

```
mylist = ["me", "dad", " mom"]
mylist.reverse()
print(mylist)
```

```
The result :
['mom', 'dad', 'me']
```

Example 2 Add the element “brother” after the first word in this list “mylist”.

```
mylist = ["me", "dad", "mom"]
mylist.insert(1, "brother")
print(mylist)
```

```
The result :
['me', 'brother', 'dad', 'mom']
```

“tuple” is a data set that is ordered and immutable. It allows you to duplicate elements. This type uses parentheses instead of square brackets. Example of using «*tuple*»:

```
mytuple = ("me", "dad", "mom") or
mytuple = ('me', 'dad', 'mom')
```

Methods with «tuple»: (see table 1.4)

Table 1.4 – The different methods on the «*tuple*»

Methods	Description
count()	Returns the number of times a given value occurs in *«tuple»*
index()	Searches in *«tuple»* for the specified value and returns the location where it was found

Example 1 Find the tuple "*mytuple*" the specified value «dad» and return its index, i.e. the number where this element is located.

```
mytuple = ("me", "dad", "mom")
x=mytuple.index("dad")
print(f"Item number «{mytuple[1]}» in «tuple» = "+str(x))
```

```
Результат:
Item number «dad» in «tuple» = 1
```

Note: The ***index()*** method throws an exception if the value is not found.

Example 2 Find in tuple "*mytuple*" how many times the element «dad» is found.

```
mytuple = ("me", "dad", "mom", "dad")
n=mytuple.count("dad")
print(f"Number of times the «dad» element is encountered = "+str(n))
```

```
The result:
Number of times the «dad» element is encountered = 2
```

How can I change the value in «tuple»?

«tuple» needs to be converted to a list so that it can be changed.

```
mytuple = ("me", "dad", "mom")
y = list(mytuple)
y[1] = "uncle"
mytuple = tuple(y)

print(mytuple)
```

```
The result:
("me", "uncle", "mom")
```

"***set***" is a collection that is not ordered or indexed. No duplicate members. This type uses curly brackets instead of square brackets or regular ones. Example of using «*set*»:

```
myset = {"me", "dad", "mom"} or
myset = {'me', 'dad', 'mom'}
```

Methods using the «set» (see table 1.5)

Table 1.5 – The different methods with "set"

Methods	Description
add()	Adds an item to the set
clear()	Deletes all elements
copy()	Returns a copy of the set
difference()	Returns a set containing the difference between two or more sets
difference_update()	Removes items from this set that are also included in another specified set
discard()	Deletes the specified element
intersection()	Returns a set that is the intersection of two other rows

Continuation of table 1.5

Methods	Description
intersection_update ()	Removes elements from this set that are missing from other sets specified in set ()
isdisjoint()	Returns if two sets have an intersection or not
issubset()	Returns if a subset is contained in the other set
issuperset()	Returns if a certain subset is in another set
pop()	Removes an element from the set
remove()	Deletes the specified element
symmetric_difference()	Returns a set with symmetric differences between two sets (sets)
symmetric_difference_update()	Inserts the symmetric differences of this subset and the other
union()	Returns a set containing a union of sets
update ()	Updates the original set by combining other sets

Example 1 Printing elements from a given *myset* set.

```
myset = {"me", "brother", "dad", "mom"}
for x in myset:
    print(x)
```

The result (option 1):
dad
me
mom
brother

The result (option 2):
brother
mom
dad
me

Example 2 Check if the «brother» element is in the *myfamily* set; if so, print the phrase: “We have the same father and the same mother!”

```
myfamily = {"me", "brother", "dad", "mom"}
if "brother" in myfamily:
    print("We have the same father and mother!")
```

```
The result:
We have the same father and mother!
```

Example 3 Enter an «excellent» rating in an existing set and display the result on the screen.

```
atsenka = {"good", "bad", " satisfactory"}
appréciation.add("excellent")
print(atsenka)
```

```
The result:
{'good', 'excellent', 'bad', 'satisfactory'}
# one of the possible results.
```

“dictionary” is a collection that is unordered, edited, and indexed. There are no duplicate members. They are indicated by curly brackets; in addition, they have keywords with meaning.

```
aboutme_dict ={
  "surname": "Ivanov",
  "name": "Ivan",
  "sexe": "masculin",
  "age": 30,
  "statut": "marier"
}
print(aboutme_dict)
```

or

```
aboutme_dict ={
  "surname": "Ivanov",
  "name": "Ivan",
  "sexe": "masculin",
  "age": 30,
  "statut": "marier"
}
print(aboutme_dict)
```

```
The result:
{'surname': 'Ivanov', 'name': 'Ivan', 'sexe': 'masculin', 'age': 30,
'statut': 'marier'}
```

Example 1 Get the value of the "vozrast" object, knowing that we have the Dictionary *aboutme_dict* (see previous) and display it on the screen.

```
znacheniye_1 = aboutme_dict["vozrast"]
print(f"The value of the object 'vozrast' (1) is equal to =
{znacheniye_1}") # or
znacheniye_2 = aboutme_dict.get("vozrast ")
print(f"The value of the object 'vozrast' (2) is equal to =
{znacheniye_2}")
```

```
The result :
The value of the object 'vozrast' (1) is equal to = 30
# or
The value of the object 'vozrast' (2) is equal to = 30
```

Example 2 display keywords with a length of 6 characters (also display the found value). In this case, use the input data from the *aboutme_dict* «dictionary» (see previous).

Creating a loop using operator «for»

```
for cle in aboutme_dict:
    if (len(str(aboutme_dict[cle]))==6):
        print(f'The keyword = {cle}')
        print(f' The meaning of the keyword "{cle}" is:
                                        {aboutme_dict[cle]}')
```

```
The result:
The keyword = familia
The meaning of the keyword "surname" is: Ivanov
The keyword = status
The meaning of the keyword "status" is: marier
```

Example 3 Display values that are 6 characters long. You must use the input data from the Dictionary *aboutme_dict*.

```
for val in aboutme_dict.values():
    if (len(str(val))==6):
        print(f'We have the value of "{val}" without the keyword')
         print(f'He is here: {aboutme_dict.values()}')
```

```
The result:
We have the value of "Ivanov" without the keyword
He is here: dict_values(['Ivanov', 'Ivan', 'masculin', 30, 'marier'])
We have the value of "marier" without the keyword
He is here: dict_values(['Ivanov', 'Ivan', 'masculin', 30, 'marier'])
```

Example 4 display keywords with a length of 6 characters in a different way (also display the found values). Use the input data from «dictionary» *aboutme_dict* (see previous example №2).

Creating a loop using operator «for» and items method()

```
for cle, val in aboutme_dict.items():
    if (len(str(val))==6):
        print(f'Key = {cle}')
        print(f' The value of the key "{cle}" is: {val}')
```

```
The result:
Key = surname
The value of the key "surname" is: Ivanov
Key = status
The value of the key "status" is: marier
```

Example 5 Create a "*dictionary*" containing three small "*dictionaries*", for example: a group of students with their respective characteristics.

The main "*dictionary*" is called GS_2020. Small "*dictionaries*": Ivanova, Petrov and Sidorov. Each of them should have: type of student (scholarship holder or not), gender (male or female), nationality (Russian or foreign) and year of birth.

```
GS_2020 = {
  "Ivanova" : {
     "typ_stud"    : "scholarship",
     "genre"         : "female",
     "nationality"  : "Russian",
     "year_birth" : 2001
  },
  "Petrov" : {
     "typ_stud"    : "scholarship",
     "genre"         : "male",
     "nationality"  : "foreign",
     "year_birth" : 2002
  },
  "Sidorov" : {
     "typ_stud"    : "scholarship",
     "genre"         : "male",
     "nationality"  : "Russian",
     "year_birth" : 1999
  }
}
```

```
The result: GS_2020
{'Ivanova': {'typ_stud': 'scholarship', 'genre': 'female', 'nationality': 'Russian', 'year_birth': 2001}, 'Petrov': {'typ_stud': 'scholarship', 'genre': 'male', 'nationality': 'foreign', 'year_birth': 2002}, 'Sidorov': {'typ_stud': 'scholarship', 'genre': 'male', 'nationality': 'Russian', 'year_birth': 1999}}
```

Another way is used if you want to enter three "*dictionaries*" that already exist in the "*dictionary*":

```
Ivanova = {
    "typ_stud"    : "scholarship",
    "genre"         : "female",
    "nationality"  : "Russian",
    "year_birth" : 2001
  }
Petrov = {
    "typ_stud"    : "scholarship",
    "genre"         : "male",
    "nationality"  : "foreign",
    "year_birth" : 2002
  }
Sidorov = {
    "typ_stud"    : "scholarship",
    "genre"         : "male",
    "nationality"  : "Russian",
    "year_birth" : 1999
  }
GS_2020 = {
  "Ivanova" : Ivanova,
  "Petrov"  : Petrov,
  "Sidorov" : Sidorov
}
```

```
The result: (same as the previous one)
GS_2020
{'Ivanova': {'typ_stud': 'scholarship', 'genre': 'female', 'nationality': 'Russian', 'year_birth': 2001}, 'Petrov': {'typ_stud': 'scholarship', 'genre': 'male', 'nationality': 'foreign', 'year_birth': 2002}, 'Sidorov': {'typ_stud': 'scholarship', 'genre': 'male', 'nationality': 'Russian', 'year_birth': 1999}}
```

Methods using the «dictionary» (see table 1.6)

Table 1.6 – The different methods with “dictionary”

Methods	Description
clear()	Removes all elements from the dictionary
copy()	Returns a copy of the dictionary
fromkeys()	Returns a dictionary with the specified keys and value
get()	Returns the value of the specified key
items()	Returns a list containing Tuple for each pair of key values
keys()	Returns a list containing a key dictionary
pop()	Deletes an element with the specified key
popitem()	Deletes the last key-value pair inserted
setdefault()	Returns the value of the specified key. If the key does not exist: insert the key with the specified value
update()	Updates the dictionary with key-value pairs
values()	Returns a list of all dictionary values

Continued on page 111

FR I.5. Collection des données sur tableaux (Arrays)

Il existe quatre façons de garder les données en forme de rangée: par les type «***list***», «***tuple***», «***set***» et «***dictionary***».

«list» est une collection qui est ordonnée et modifiable. Permet de dupliquer les membres. Exemple avec «**list**» :

```
maliste = ["moi", "papa", "maman"] ou
maliste = ['moi', 'papa', 'maman']
```

Les Méthodes sur la liste – «list» (voir table 1.3)

Table 1.3 – Les différentes méthodes de la liste – «list»

Les méthodes	Description
append()	Ajoute un élément à la fin de la liste
clear()	Supprime tous les éléments de la liste
copy()	Renvoie une copie de la liste
count()	Renvoie le nombre d'éléments avec la valeur spécifiée
extend()	Ajouter les éléments d'une liste, à la fin de la liste actuelle
index()	Renvoie l'index du premier élément avec la valeur spécifiée
insert()	Ajoute un élément à la position spécifiée
pop()	Supprime l'élément à la position spécifiée
remove()	Supprime l'élément avec la valeur spécifiée
reverse()	Inverse l'ordre de la liste
sort()	Trie la liste

Exemple 1 Inverser l'ordre de la liste *maliste*.

```
maliste = ["moi", "papa", "maman"]
maliste.reverse()
print(maliste)
```

```
Résultat :
['maman', 'papa', 'moi']
```

Exemple 2 Ajouter l'élément «frère» après le premier mot de la liste *maliste* donnée.

```
maliste = ["moi", "papa", "maman"]
maliste.insert(1, "frère")
print(maliste)
```

```
Résultat :
['moi', 'frère', 'papa', 'maman']
```

«tuple» est une collection des données qui est ordonnée et immuable (inchangeable). Permet de dupliquer les membres. Ce type utilise les parenthèses à la place des crochets. Nous utilisons la méthode «tuple».

```
mytuple = ("moi", "papa", "maman") ou
mytuple = ('moi', 'papa', 'maman')
```

Les Méthodes sur le tuple (voir table 1.4)

Table 1.4 – Les différentes méthodes sur le «tuple»

Les méthodes	Description
count()	Renvoie le nombre de fois qu'une valeur spécifiée se produit dans un tuple
index()	Recherche dans le tuple une valeur spécifiée et renvoie la position de l'endroit où elle a été trouvée

Exemple 1. Rechercher dans le tuple *mytuple* la valeur spécifiée «papa» et renvoyer son index, c'est-à-dire le numéro où l'élément donné se trouve.

```
mytuple = ("moi", "papa", "maman")
x=mytuple.index("papa")
print(f"Le numéro de l' élément «{mytuple[1]}» est = "+str(x))
```

```
Résultat :
Le numéro de l' élément «papa» est = 1
```

Remarque La méthode ***index()*** lève une exception si la valeur n'est pas trouvée. ☹

Exemple 2 Trouver dans le tuple *mytuple* le nombre de fois que l'élément «papa» se trouve.

```
mytuple = ("moi", "papa", "maman", "papa")
n=mytuple.count("papa")
print(f"Le nombre de fois qu'on a l' élément «papa» est = "+str(n))
```

```
Résultat :
Le nombre de fois qu'on a l' élément «papa» est = 2
```

Comment changer une valeur dans un tuple ?

Il faut converti le tuple en une liste pour pouvoir le changer.

```
mytuple = ("moi", "papa", "maman")
y = list(mytuple)
y[1] = "oncle"
mytuple = tuple(y)

print(mytuple)
```

```
Résultat :
("moi", "oncle", "maman")
```

«set» est une collection qui n'est pas ordonnée et non indexée. Aucun duplicata de membres. Ce type utilise les accolades à la place des crochets ou des parenthèses. Nous utilisons la méthode «set».

```
myset = {"moi", "papa", "maman"} ou
myset = {'moi', 'papa', 'maman'}
```

Les Méthodes sur le «set» (voir table 1.5)

Table 1.5 – Les différentes méthodes avec «set»

Les méthodes	Description
add()	Ajoute un élément à l'ensemble
clear()	Supprime tous les éléments de l'ensemble

Suite de la table 1.5

copy()	Renvoie une copie de l'ensemble
difference()	Retourne un ensemble contenant la différence entre deux ensembles ou plus
difference_update()	supprime les éléments de cet ensemble qui sont également inclus dans un autre ensemble spécifié
discard()	Retirer l'élément spécifié
intersection()	Retourne un ensemble, qui est l'intersection de deux autres séries
intersection_update ()	supprime les éléments de cet ensemble qui ne sont pas présents dans d'autres ensembles spécifiés par ***set()***
isdisjoint()	retourne si deux ensembles ont une intersection ou non
issubset()	retourne si un autre ensemble contient cet ensemble ou non
issuperset()	retourne si cet ensemble contient un autre ensemble ou non
pop()	Supprime un élément de l'ensemble
remove()	Supprime l'élément spécifié
symmetric_difference()	renvoie un ensemble avec les différences symétriques de deux ensembles
symmetric_difference_update()	insère les différences symétriques de cet ensemble et d'un autre
union()	renvoie un ensemble contenant l'union des ensembles
update ()	Mettre à jour l'ensemble avec l'union de cet ensemble et d'autres

Exemple 1 Imprimer les éléments de l'ensemble *myset* donnée.

```
myset = {"moi", "frère", "papa", "maman"}
for x in myset:
    print(x)
```

Résultat (variante 1):
papa
moi
maman
frère

Résultat (variante 2):
frère
maman
papa
moi

Exemple 2 Vérifier si l'élément «frère» se trouve dans l'ensemble *myfamily*; si oui, imprimer la phrase: "Nous avons un même père et mère!".

```
myfamily = {"moi", "frère", "papa", "maman"}
if "frère" in myfamily:
    print("Nous avons un même père et mère!")
```

Résultat :
Nous avons un même père et mère!

Exemple 3 Introduire l'appréciation «excellent» dans l'ensemble existant et imprimer le résultat.

```
appréciation = {"bon", "mal", "mauvais"}
appréciation.add("excellent")
print(appréciation)
```

Résultat :
{'mal', 'excellent', 'mauvais', 'bon'} # un des résultats possible

«dictionary» est une collection qui est non ordonnée, modifiable et indexée. Aucun duplicata de membres. Ils sont écrits avec des accolades, en plus, ils ont des mots clés avec des valeurs.

```
aboutme_dict ={
  "nom": "Ivanov",
  "prenom": "Ivan",
  "sexe": "masculin",
  "age": 30,
  "statut": "marier"
}
print(aboutme_dict)
```

ou bien

```
aboutme_dict ={
  "nom": "Ivanov",
  "prenom": "Ivan",
  "sexe": "masculin",
  "age": 30,
  "statut": "marier"
}
print(aboutme_dict)
```

```
Résultat :
{'nom': 'Ivanov', 'prenom': 'Ivan', 'sexe': 'masculin', 'age': 30,
'statut': 'marier'}
```

Exemple 1 Obtenir la valeur de l'objet «age» sachant que nous avons un «dictionary» *aboutme_dict* (voir précédant) et imprimer.

```
valeur1 = aboutme_dict["age"]
print(f"La valeur de l'objet age est (variante 1) = {valeur1}")
# ou alors
valeur2 = aboutme_dict.get("age")
print(f"La valeur de l'objet age est (variante 2) = {valeur2}")
```

```
Résultat :
La valeur de l'objet age est (variante 1) = 30
# ou alors
La valeur de l'objet age est (variante 2) = 30
```

Exemple 2 Imprimer les mots clés dont la valeur a une longueur égale à 6 symboles (imprimer également la valeur trouvée). Utilisons les données d'entrée du «dictionary» *aboutme_dict* (voir précédant).

Nous utilisons la boucle «for»

```
for cle in aboutme_dict:
    if (len(str(aboutme_dict[cle]))==6):
        print(f'La clé est = {cle}')
        print(f'La valeur de la clé "{cle}" est : {aboutme_dict[cle]}')
```

```
Résultat :
La clé est = nom
La valeur de la clé "nom" est : Ivanov
La clé est = statut
La valeur de la clé "statut" est : marier
```

Exemple 3 Imprimer les valeurs qui ont une longueur égale à 6 symboles. Utiliser les données d'entrée du «dictionary» *aboutme_dict*.

```
for val in aboutme_dict.values():
    if (len(str(val))==6):
        print(f'Ici, nous avons juste la valeur "{val}" sans le mot clé')
         print(f'Elle est ici: {aboutme_dict.values()}')
```

```
Résultat :
Ici, nous avons juste la valeur "Ivanov" sans le mot clé
Elle est ici: dict_values(['Ivanov', 'Ivan', 'masculin', 30, 'marier'])
Ici, nous avons juste la valeur "marier" sans le mot clé
Elle est ici: dict_values(['Ivanov', 'Ivan', 'masculin', 30, 'marier'])
```

Exemple 4 Imprimer d'une autre façon les mots clés dont la valeur a une longueur égale à 6 symboles (imprimer également la valeur trouvée). Utiliser les données d'entrée du «dictionary» *aboutme_dict* (voir précédant exemple 2).

Nous utilisons la boucle «for»* avec *«items()»

```
for cle, val in aboutme_dict.items():
    if (len(str(val))==6):
        print(f'La clé est = {cle}')
        print(f'La valeur de la clé "{cle}" est : {val}')
```

```
Résultat :
La clé est = nom
La valeur de la clé "nom" est : Ivanov
La clé est = statut
La valeur de la clé "statut" est : marier
```

Exemple 5 Créer un «dictionary» qui contient trois dictionnaires. Par exemple, un groupe d'étudiants avec leurs caractéristiques resectifs.

Le «dictionary» principal est appelé GS_2020. Les petits «dictionary» sont: Ivanova, Petrov et Sidorov. Chacun d'eux doit avoir: le type d'étudiant(boursier ou pas), le sexe(masculin ou féminin), nationalité (russe ou étrangère) et année de naissance.

```
GS_2020 = {
  "Ivanova" : {
      "type_stud"    : "boursière",
      "sexe"         : "féminin",
      "nationalité"  : "russe",
      "année_naiss" : 2001
  },
  "Petrov" : {
      "type_stud"    : "Boursier",
      "sexe"         : "masculin",
      "nationalité"  : "étrangère",
      "année_naiss" : 2002
  },
  "Sidorov" : {
      "type_stud"    : "Boursier",
      "sexe"         : "masculin",
      "nationalité"  : "russe",
      "année_naiss" : 1999
  }
}
```

```
Résultat : GS_2020
{'Ivanova': {'type_stud': 'boursière', 'sexe': 'féminin', 'nationalité': 'russe', 'année_naiss': 2001}, 'Petrov': {'type_stud': 'Boursier', 'sexe': 'masculin', 'nationalité': 'étrangère', 'année_naiss': 2002}, 'Sidorov': {'type_stud': 'Boursier', 'sexe': 'masculin', 'nationalité': 'russe', 'année_naiss': 1999}}
```

Une autre possibilité dans le cas où vous voulez introduire trois dictionnaires qui existent déjà dans un Dictionary:

```
Ivanova = {
    "type_stud"    : "boursière",
    "sexe"         : "féminin",
    "nationalité"  : "russe",
    "année_naiss" : 2001
  }
Petrov = {
    "type_stud"    : "Boursier",
    "sexe"         : "masculin",
    "nationalité"  : "étrangère",
    "année_naiss" : 2002
  }
Sidorov = {
    "type_stud"    : "Boursier",
    "sexe"         : "masculin",
    "nationalité"  : "russe",
    "année_naiss" : 1999
  }
GS_2020 = {
  "Ivanova" : Ivanova,
  "Petrov"  : Petrov,
  "Sidorov" : Sidorov
}
```

Résultat : (même chose comme le précédent)
GS_2020
{'Ivanova': {'type_stud': 'boursière', 'sexe': 'féminin', 'nationalité': 'russe', 'année_naiss': 2001}, 'Petrov': {'type_stud': 'Boursier', 'sexe': 'masculin', 'nationalité': 'étrangère', 'année_naiss': 2002}, 'Sidorov': {'type_stud': 'Boursier', 'sexe': 'masculin', 'nationalité': 'russe', 'année_naiss': 1999}}

Les Méthodes sur le «dictionary» (voir table 1.6)

Table 1.6 – Les différentes méthodes avec «dictionary»

Les méthodes	Description
clear()	Supprime tous les éléments du dictionnaire
copy()	Renvoie une copie du dictionnaire
fromkeys()	renvoie un dictionnaire avec les clés et la valeur spécifiées
get()	Retourne la valeur de la clé spécifiée
items()	renvoie une liste contenant un tuple pour chaque paire de valeurs clés
keys()	Renvoie une liste contenant le dictionnaire clés
pop()	Supprime l'élément avec la clé spécifiée
popitem()	supprime la dernière paire clé-valeur insérée
setdefault()	Renvoie la valeur de la clé spécifiée. Si la clé n'existe pas: insérez la clé, avec la valeur spécifiée
update()	met à jour le dictionnaire avec les paires clé-valeur
values()	renvoie une liste de toutes les valeurs du dictionnaire

Suite à la page 114

RU I.6. Список строковых методов и их описания

В таблице 1.7 представлены возможные методы и их описания при обработке строки данных.

Таблица 1.7 – Строковые Описание методов в строке

Методы	**Описание**
lower()	Возвращает строку в нижний регистр
upper()	Возвращает строку в верхний регистр
capitalize()	Преобразует первый символ в верхний регистр
casefold ()	Преобразует строку в нижний регистр
centre()	Возвращает строку по центру
count()	Определяет количество раз, когда указанное значение встречается в строке
encode()	Возвращает кодированную версию строки
endswith()	Возвращает «True», если строка заканчивается указанным значением
expandtabs()	Задает размер табуляции строки
find()	Выполняет поиск в строке указанного значения и возвращает позицию, в которой оно было найдено
format()	Форматирует значения, указанные в строке
format_map()	Форматирует значения, указанные в строке
index()	Поиск в строке указанного значения и возврат позиции, где оно было найдено
isalnum()	Возвращает «True», если все символы в строке являются буквенно-цифровыми
isalpha()	Возвращает «True», если все символы в строке являются буквами в алфавите
isdecimal()	Возвращает «True», если все символы в строке являются десятичными знаками
isdigit()	Возвращает «True», если все символы в строке являются цифрами
Isidentifier()	Возвращает «True», если строка является идентификатором
islower()	Возвращает «True», если все символы в строке в нижнем регистре
isnumeric()	возвращает «True», если все символы в строке числовые

Продолжение таблицы 1.7

isprintable()	Возвращает «True», если все символы в строке доступны для печати
isspace()	Возвращает «True», если все символы в строке являются пробелами
istitle()	Возвращает «True», если строка соответствует правилам заголовка
isupper()	Возвращает «True», если все символы в строке прописные
join()	Уплотнение элементов в iterable в конце строки
ljust()	Возвращает подходящую версию влево от строки
lower()	Преобразует строку в нижний регистр
lstrip ()	Удаляет символы в начале строки
maketrans()	Возвращает таблицу перевода для использования в переводах
partition()	Возвращает tuple, где строка разделена на три части
replace()	Возвращает строку, в которой одно указанное значение заменяется другим указанным значением. Заменяет строку другой строкой
rfind()	Возвращает самый большой индекс в строке, на котором была найдена указанная подстрока. Если не находит заданную подстроку после поиска, то выдает значение “-1”, а не *ValueError*.
rindex()	Возвращает самый большой индекс в строке, на котором была найдена указанная подстрока. Если не находит заданную подстроку после поиска, то выдает *ValueError*, а не значение “-1”.
rjust()	Возвращает подходящую версию вправо от строки
rpartition()	Возвращает один tuple, в котором строка разделена на три части
rsplit()	делит строку на указанный разделитель и возвращает список
rstrip()	Удаляет символы в конце строки

Окончание таблицы 1.7

split()	Делит строку на указанный разделитель и возвращает список. Разбивает строку на подстроки, если она находит экземпляры разделителя
splitlines()	Разбивает строку на разрывы строк и возвращает список
startswith()	Возвращает «True», если строка начинается с указанного значения
strip()	Возвращает усеченную версию строки, очищает пустые символы.
swapcase()	Строчные буквы становятся прописными и наоборот
titre()	Преобразует первый символ каждого слова в верхний регистр
translate()	Возвращает переведенную строку
upper()	Преобразует строку в верхний регистр
zfill()	Заполняет строку заданным числом значений 0 в начале

Продолжение на странице 117

EN I.6. List of string methods and their descriptions

Table 1.7 shows the methods and their descriptions for processing a data row.

Table 1.7 – Description of methods on the data row

Methods	Description
lower()	Returns the string in lower case
upper()	Returns a string in uppercase
capitalize()	Converts the first character to uppercase
casefold ()	Converts a string to lowercase
centre()	Returns a string in the center
count()	Specifies the number of times the specified value occurs in a string
encode()	Returns the encoded version of the string

Continuation of table 1.7

endswith()	Returns “True” if the string ends with the specified value
expandtabs()	Sets the tab size of the line
find()	Searches the string for the specified value and returns the position at which it was found
format()	Formats the values specified in the string
format_map()	Formats the values specified in the string
index()	Search in a string for the specified value and return the position where it was found
isalnum()	Returns “True” if all characters in the string are alphanumeric
isalpha()	Returns “True” if all characters in the string are letters in the alphabet
isdecimal()	Returns “True” if all characters in the string are decimal places
isdigit()	Returns “True” if all characters in the string are digits
Isidentifier()	Returns “True” if the string is an identifier
islower()	Returns “True” if all characters in the string are lowercase
isnumeric()	returns “True” if all characters in the string are numeric
isprintable()	Returns “True” if all characters in the string are printable
isspace()	Returns “True” if all characters in the string are spaces
istitle()	Returns “True” if the string matches the header rules
isupper()	Returns “True” if all characters in the string are uppercase
join()	Compacting elements in an iterable at the end of a line
ljust()	Returns the appropriate version to the left of the line
lower()	Converts a string to lowercase

End of table 1.7

lstrip ()	Removes characters at the beginning of a line
maketrans()	Returns the translation table for use in translations
partition()	Returns tuple, where the string is divided into three parts
replace()	Returns a string in which one specified value is replaced by another specified value. Replaces a string with another string
rfind()	Returns the largest index of the string on which the specified substring was found. If it does not find the specified substring after the search, it returns the value “-1” and not *ValueError*.
rindex()	Same function as the previous one, but only here, if it does not find the specified substring after the search, it returns *ValueError* and not the value “-1”.
rjust()	Returns the appropriate version to the right of the line
rpartition()	Returns a single tuple in which the string is divided into three parts
rsplit()	divides the string by the specified separator and returns a list
rstrip()	Removes characters at the end of a line
split()	Divides the string by the specified delimiter and returns a list. Splits a string into substrings if it finds instances of the delimiter
splitlines()	Splits a string into line breaks and returns a list
startswith()	Returns “True” if the string starts with the specified value
strip()	Returns the truncated version of the string, clears empty characters.
swapcase()	Lowercase letters become uppercase and vice versa
titre()	Converts the first character of each word to uppercase
translate()	Returns the translated string
upper()	Converts a string to uppercase
zfill()	Fills the string with the specified number of values 0 at the beginning

Continued on page 118

FR I.6. Les différentes méthodes sur une chaîne et leurs descriptions en forme de tableau

Le tableau 1.7 présente les méthodes possibles et leurs descriptions lors du traitement d'une chaîne de données.

Table 1.7 – Description des méthodes sur une chaîne

Les méthodes	Description
lower()	Renvoie la chaîne en minuscules
upper()	Renvoie la chaîne en majuscules
capitalize()	Convertit le premier caractère en majuscule
casefold ()	convertit la chaîne en minuscules
centre()	Renvoie une chaîne centré
count()	renvoie le nombre de fois qu'une valeur spécifiée apparaît dans une chaîne
encode()	Renvoie une version codée de la chaîne
endswith()	Renvoie «True» si la chaîne se termine par la valeur spécifiée
expandtabs()	Définit la taille de tabulation de la chaîne
find()	Recherche dans la chaîne une valeur spécifiée et renvoie la position où elle a été trouvée
format()	Met en forme les valeurs spécifiées dans une chaîne
format_map()	Met en forme les valeurs spécifiées dans une chaîne
index()	Recherche dans la chaîne une valeur spécifiée et renvoie la position de l'endroit où elle a été trouvée
isalnum()	Renvoie «True» si tous les caractères de la chaîne sont alphanumériques
isalpha()	Renvoie «True» si tous les caractères de la chaîne sont dans l'alphabet
isdecimal()	Renvoie «True» si tous les caractères de la chaîne sont des décimales

Suite de la table 1.7

isdigit()	Renvoie «True» si tous les caractères de la chaîne sont des chiffres
Isidentifier()	Renvoie «True» si la chaîne est un identifiant
islower()	Renvoie «True» si tous les caractères de la chaîne sont en minuscules
isnumeric()	Renvoie «True» si tous les caractères de la chaîne sont numériques
isprintable()	Renvoie «True» si tous les caractères de la chaîne sont imprimables
isspace()	Renvoie «True» si tous les caractères de la chaîne sont des espaces
istitle()	Renvoie «True» si la chaîne suit les règles d'un titre
isupper()	Renvoie «True» si tous les caractères de la chaîne sont en majuscules
join()	Joint les éléments d'un itérable à la fin de la chaîne
ljust()	Renvoie une version justifiée à gauche de la chaîne
lower()	Convertit une chaîne en minuscules
lstrip ()	Supprime les caractères au début d'une chaîne
maketrans()	Renvoie une table de traduction à utiliser dans les traductions
partition()	Renvoie un tuple où la chaîne est séparée en trois parties
replace()	Renvoie une chaîne dans laquelle une valeur spécifiée est remplacée par une valeur spécifiée. Remplace une chaîne par une autre chaîne
rfind()	Renvoie le plus grand index de la chaîne sur lequel la sous-chaîne spécifiée a été trouvée. Si elle ne trouve pas la sous-chaîne spécifiée après la recherche, elle renvoie la valeur “-1” et non *ValueError.*

Fin de la table 1.7

rindex()	Même fonction que la précédente, mais seulement ici, si elle ne trouve pas la sous-chaîne spécifiée après la recherche, elle renvoie *ValueError* et non la valeur "-1".
rjust()	Renvoie une version justifiée à droite de la chaîne
rpartition()	Renvoie un tuple où la chaîne est séparée en trois parties
rsplit()	divise la chaîne au niveau du séparateur spécifié et renvoie une liste
rstrip()	Supprime les caractères à la fin d'une chaîne
split()	Divise la chaîne au niveau du séparateur spécifié et renvoie une liste. *(Divise la chaîne en sous-chaînes si elle trouve des instances du séparateur)*
splitlines()	Divise la chaîne aux sauts de ligne et renvoie une liste
startswith()	Renvoie true si la chaîne commence avec la valeur spécifiée
strip()	Renvoie une version tronquée de la chaîne, efface les caractères vides.
swapcase()	Échange les cas, les minuscules deviennent les majuscules et vice versa
titre()	Convertit le premier caractère de chaque mot en majuscule
translate()	Renvoie une chaîne traduite
upper()	Convertit une chaîne en majuscules
zfill()	Remplit la chaîne avec un nombre spécifié de valeurs 0 au début

Suite à la page 120

RU I.7. Роль модуля «array» и чем он отличается от предыдущих (*“list”, “tuple”, “set” и “dictionary”*)

Мы можем обнаружить или констатировать, что модуль « array » также определяет массивы в Python, который имеет три типа элементарных значений: символы, целые числа, вещественные (см. таб. 1.8). Эти массивы очень похожи на списки “*list*”, но с ограничением на некоторые параметры: тип данных и размер каждого элемента. Эти два приведенных нами параметра определяются при создании массива и могут принимать следующие значения:

Таблица 1.8 – Определение типов в модуле «**array**»

Код индикации типа	**Тип C**	**Тип Python**	**Минимальный размер в байтах**
'b'	signed char	*int*	1
'B'	Без знака «–» char	*int*	1
'u'*	Py_UNICODE*	Символ *Юникод	2*
'h'	signed short	*int*	2
'H'	Без знака «–» short	*int*	2
'i'	signed int	*int*	2(4)
'I'	Без знака «–» int	*int*	2(4)
'l'	signed long	*int*	4
'L'	Без знака «–» long	*int*	4
'q'	signed long long	*int*	8
'Q'	Без знака «–» long long	*int*	8
'f'	*float*	*float*	4
'd'	double	*float*	8

** 'u' будет удален вместе с остальной частью API Py_UNICODE в версии Python 4.0. Надо по возможности избегать использования этого типа.*

Например, мы можем создать массив целочисленного типа, действительного типа “*float*” или символьного типа через *Unicode*. Если нам нужен тип *float*, мы будем использовать букву ‘**d**’ или ‘**f**’ из таблицы №1.8.

Разница с другими по-прежнему в том, чтобы использовать эти таблицы «array» в Python, необходимо импортировать стандартный модуль «array», поскольку он не является базовым типом данных, таким как целое число, действительное, символы и т. д.

Пример использования: *from array import* ***

Этот модуль представлен по-разному в зависимости от типа измерения массива.

Особенности ***numpy arrays*** :

- мы имеем массив однородных значений;
- мы имеем широкий спектр функций, то есть больше возможностей;
- расчеты быстрее, с *numpy*.

Продолжение на странице 122

EN I.7. What is the «*array*» module and how does it differ from the previous ones (“*list*”, “*tuple*”, “*set*” and “*dictionary*”)

We can detect or state that the «array» module also defines arrays in python, which has three types of elementary values: symbols, integers, and real values (see table 1.8). These arrays are very similar to «list» lists, but with a restriction on some parameters: the data type and size of each element. These two parameters are defined when creating the array and can take the following values in table 1.8.

Table 1.8 – Definition of types in the «**array**» module

Code indicating the type of	**Type C**	**Type Python**	**Minimum size in bytes**
'b'	signed char	*int*	1
'B'	Unsigned «–» char	*int*	1
'u'*	Py_UNICODE*	*Unicode Character	2*
'h'	signed short	*int*	2
'H'	Unsigned «–» short	*int*	2
'i'	signed int	*int*	2(4)
'I'	Unsigned «–» int	*int*	2(4)
'l'	signed long	*int*	4
'L'	Unsigned «–» long	*int*	4
'q'	signed long long	*int*	8
'Q'	Unsigned «–» long long	*int*	8
'f'	*float*	*float*	4
'd'	double	*float*	8

** 'u' will be removed along with the rest of the Py_UNICODE API in Python 4.0. Avoid using this type if possible.*

For example, we can create an array of integer type, real type "*float*", or character type via *Unicode*. If we need the float type, we will use the letter '**d**' or '**f**' from table 1.8.

The difference with others is still to use these «*array*» tables in Python if you have to import the standard «*array*» module, since it is not a basic data type, such as integer, real, characters, etc.

Example of use: *from array import* *******

This module is represented differently depending on the type of array dimension.

Features of ***numpy arrays***:

- we have an array of uniforms values;

- we have a wide range of functions, that is, more features;
- calculations are faster with numpy.

Continued on page 123

FR I.7. C'est quoi le module «*array*» et quelle différence avec les précédents («*list*», «*tuple*», «*set*» et «*dictionary*»)?

Nous constatons que le module «array» définit aussi des tableaux en python qui possède trois types de valeurs élémentaires: caractères, entiers, flottants (voir table 1.8). Ces tableaux sont très similaires aux listes «list», mais avec une restriction sur certains paramètres: le type de données et la taille de chaque élément. Ces deux paramètres que nous venons de citer sont déterminés lors de la création du tableau et peuvent prendre les valeurs de la table 1.8.

Table 1.8 – La définition des types dans le module «**array**»

Code d'indication du type	Type C	Type Python	Taille minimale en octets
'b'	signed char	*int*	1
'B'	sans le signe «–» char	*int*	1
'u'*	Py_UNICODE*	Caractère* Unicode	2*
'h'	signed short	*int*	2
'H'	sans le signe «–» short	*int*	2
'i'	signed int	*int*	2(4)
'I'	sans le signe «–» int	*int*	2(4)
'l'	signed long	*int*	4
'L'	sans le signe «–» long	*int*	4
'q'	signed long long	*int*	8
'Q'	sans le signe «–» long long	*int*	8
'f'	*float*	*float*	4
'd'	double	*float*	8

** 'u' sera supprimé avec le reste de l'API Py_UNICODE en version de Python 4.0. Il faut éviter d'utiliser ce type autant que possible.*

Par exemple, nous pouvons créer un tableau de type entier ou de type *float* ou encore de type caractère via *Unicode*. Si nous voulons le type *float*, nous utiliserons la lettre '**d**' ou '**f**' venant de la table 1.8.

La différence avec les autres se trouve encore sur le faite que, pour utiliser ces tableaux «array» dans Python, if faudra importer le module standard «array», car il n'est pas un type de données de base comme un entier, un réel (flottant), caractères etc.

Exemple d'utilisation: *from array import* *******

Ce module est représenté de manière différente selon le type de dimension du tableau.

Avec ***numpy arrays***, nous avons les informations – caractéristiques suivantes:

- nous avons un tableau de valeurs uniformes;
- nous avons diversité de fonctionnalités à grande échelle, c'est-à-dire plus de possibilités;
- les calculs sont plus rapide avec *numpy*.

Suite à la page 124

RU Глава II. МАССИВЫ
EN Chapter II. ARRAYS
FR Chapitre II. UTILISATION DES MATRICES

RU Определение

Массив – упорядоченная (элементы массива пронумерованы), линейная (равноправие всех элементов) совокупность однородных (элементами могут быть данные лишь одного какого-то типа, или все элементы имеют одинаковую структуру и размер) данных.

Количество индексов, определяющих положение элемента в массиве, называется мерностью массива.

EN Definition

Array – an ordered (array elements are numbered), linear (equality of all elements) set of homogeneous (elements can be data of only one type, or all elements have the same structure and size) data.

The number of indexes that determine the position of an element in an array is called *the dimension of the array*.

FR Définition

La matrice est une collection ordonnée (les éléments du tableau sont numérotés) et linéaire (égalité de tous les éléments) de données homogènes (les éléments peuvent être des données d'un seul type, ou tous les éléments ont la même structure et la même taille).

Le nombre d'indices qui détermine la position d'un élément dans un tableau ou matrice est appelé *la dimension du tableau ou matrice*.

RU II.1. Одномерный массив

Если индекс единственный, то массив называется одномерным. Существуют два вида представления этого массива: векторная строка (см. рис. 2.1) и векторный столбец (см. рис. 2.2).

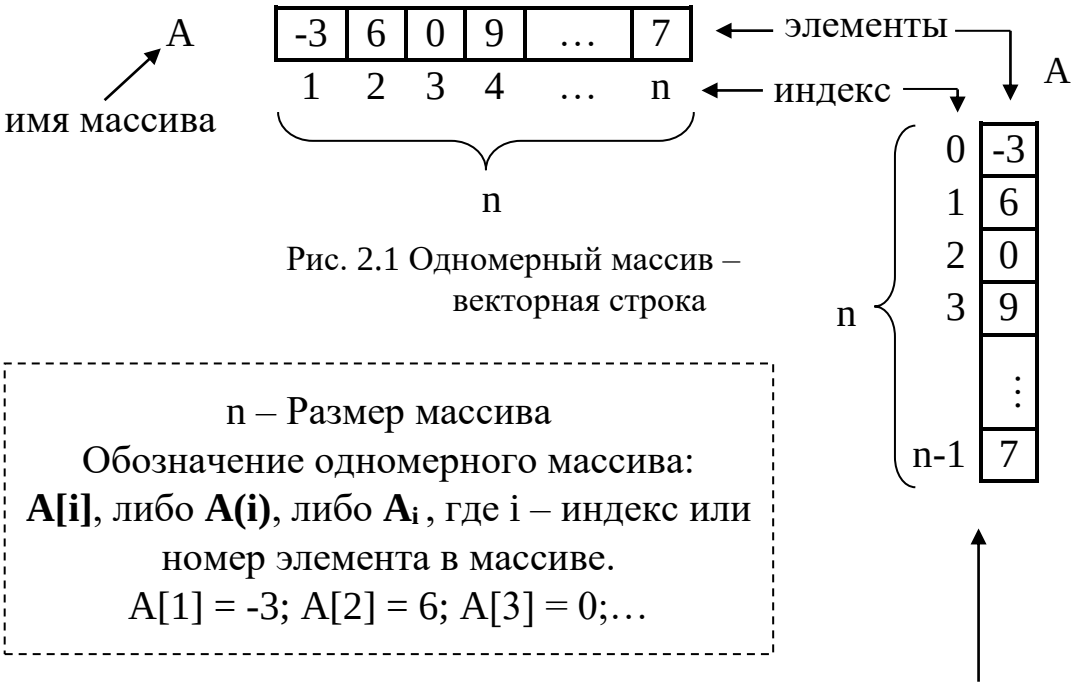


Рис. 2.1 Одномерный массив – векторная строка

Рис. 2.2 Одномерный массив – векторный столбец

Продолжение на странице 125

EN II.1. One-dimensional array

If the index is unique, then the array is called one-dimensional. There are two types of representation of this array: a "*vector row*" (see fig. 2.1) and a "*vector column*" (see fig. 2.2).

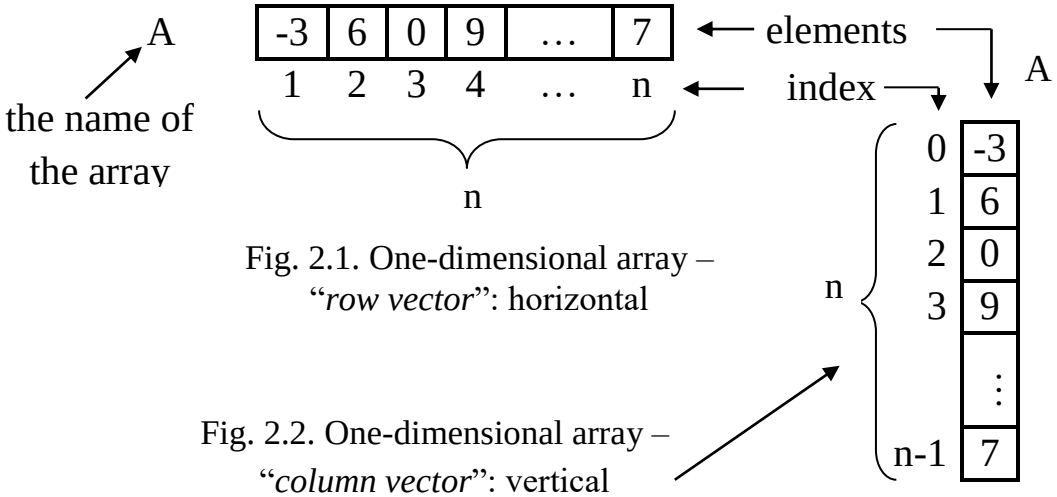


Fig. 2.1. One-dimensional array – "*row vector*": horizontal

Fig. 2.2. One-dimensional array – "*column vector*": vertical

n – the size of the array
Designation of a one-dimensional array:
A[i], or **A(i)**, or $\mathbf{A_i}$, where ***i*** is the index or number of the element in the array.
A[1] = -3; A[2] = 6; A[3] = 0;…

Continued on page 127

FR II.1. Tableaux d'effectifs unidimensionnel

Si l'index est unique, alors le tableau est appelé unidimensionnel. Il existe deux types de représentation de ce tableau: la «*ligne vectorielle*» (voir fig. 2. 1) et la «*colonne vectorielle*» (voir fig. 2.2).

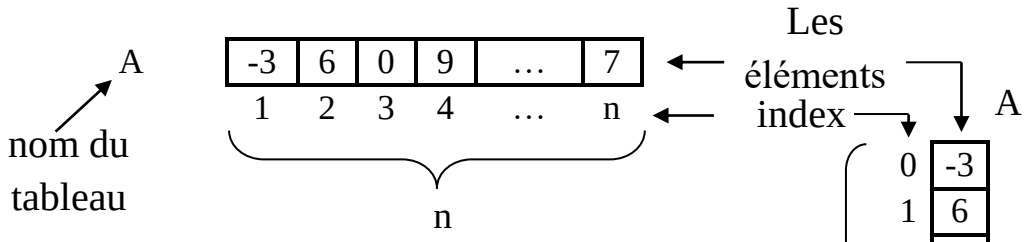


Fig. 2.1 Tableau unidimensionnel – «ligne vectorielle»: horizontale

n – dimension du tableau
Notation d'un tableau unidimensionnel:
A[i], ou **A(i)**, ou bien $\mathbf{A_i}$, où *i* – l'index ou numéro de l'élément dans le tableau.
A[1] = -3; A[2] = 6; A[3] = 0;…

Fig. 2.2 Tableau unidimensionnel – «la colonne vectorielle»: verticale

Suite à la page 129

RU II.2. Двумерный массив

Массив, элементы которого имеют два индекса, называется двумерным или матрицей (см. таб. 2.3).

Обозначение двумерного массива: B[i][j]

B i \ j	1	2	3	…	m
1	B[1][1]	B[1][2]	B[1][3]	…	B[1][m]
2	B[2][1]	B[2][2]	B[2][3]	…	B[2][m]
3	B[3][1]	B[3][2]	B[3][3]	…	B[3][m]
…	…	…	…	…	…
n	B[n][1]	B[n][2]	B[n][3]	…	B[n][m]

i – номер строки; *j* – номер столбца

n – количество строк; *m* – количество столбцов

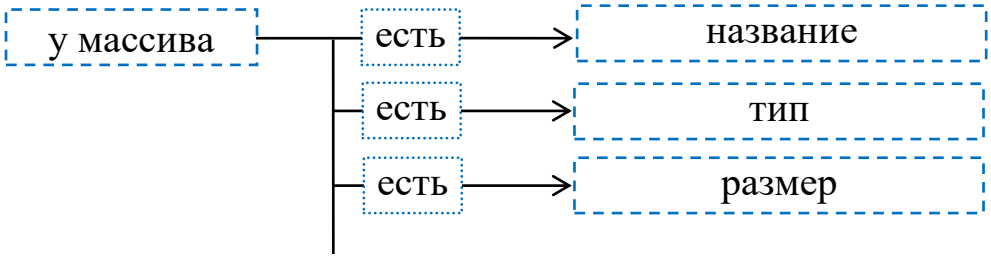
Рис. 2.3 Двумерный массив

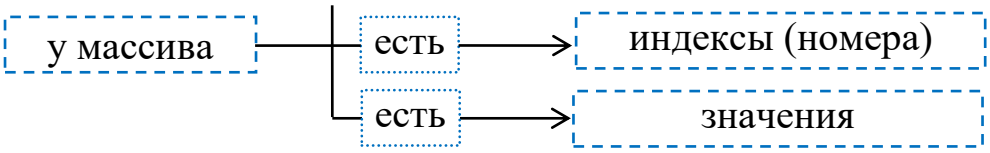
Пример двумерного массива Массив латинского квадрата D[3×3] имеет три различных элемента. Массив будет латинским, если каждый элемент ячейки отображается в каждом столбце и в каждой строке только один раз.

1	3	2
2	1	3
3	2	1

B[1][1]=1;
B[1][2]=3;
B[1][3]=2;
…
B[3][3]=1;

Следует заметить, что:





Имя = название;
тип → числовой, текстовой и так далее;
размер = количество элементов;
индекс = номер элемента;
значение = элемент массива.

Упражнение 2.1

Вы имеете матрицу магического квадрата порядка 3 (она уникальна, магическая сумма равна 15, а центральная цифра – 5): найдите её.

Упражнение 2.2

Используя полученные знания напишите код на Python, который создаст массив типа магического квадрата (см. рис. 2.4) с *n* порядками (где *n* – нечетное число), если он существует.

Один из возможных методов:

***Шаг №1*:**

Рассмотрим *n* – нечетное и равно 5.

***Шаг №2*:**

Примем, что числа используются от 1 до 25, а магическая сумма – 65.

***Шаг №3*:**

Центральная ячейка равна 13, а все противоположные пары относительно этой центральной ячейки дают сумму, равную 2×13 = 26.

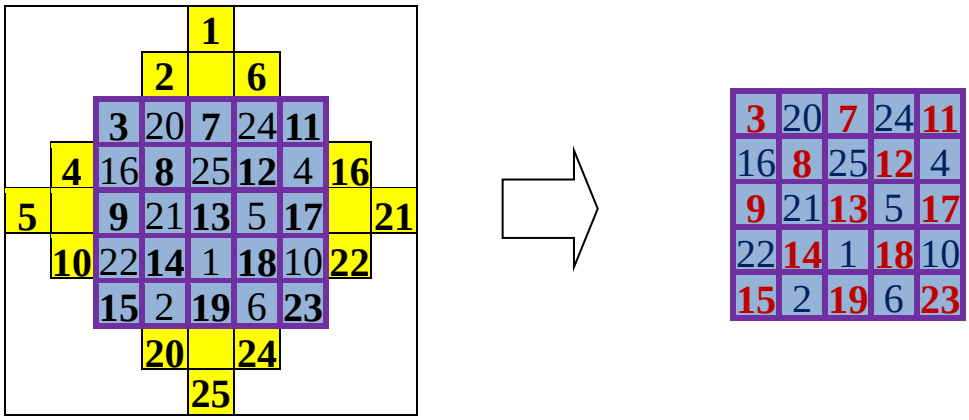


Рис. 2.4 Формирование магического квадрата

Числа от 1 до 25 располагаются друг за другом по диагоналям. Когда числа выходят за пределы, они переносятся на другую сторону.

Продолжение на странице 132

EN II.2. Two-dimensional array

An array whose elements have two indexes is called a two-dimensional array or matrix (see fig. 2.3).

Designation of a two-dimensional array: B[i][j]

B i \ j	1	2	3	…	m
1	B[1][1]	B[1][2]	B[1][3]	…	B[1][m]
2	B[2][1]	B[2][2]	B[2][3]	…	B[2][m]
3	B[3][1]	B[3][2]	B[3][3]	…	B[3][m]
…	…	…	…	…	…
n	B[n][1]	B[n][2]	B[n][3]	…	B[n][m]

i – row number; j – column number
n – number of rows; m – number of columns

Fig. 2.3. Two-dimensional array

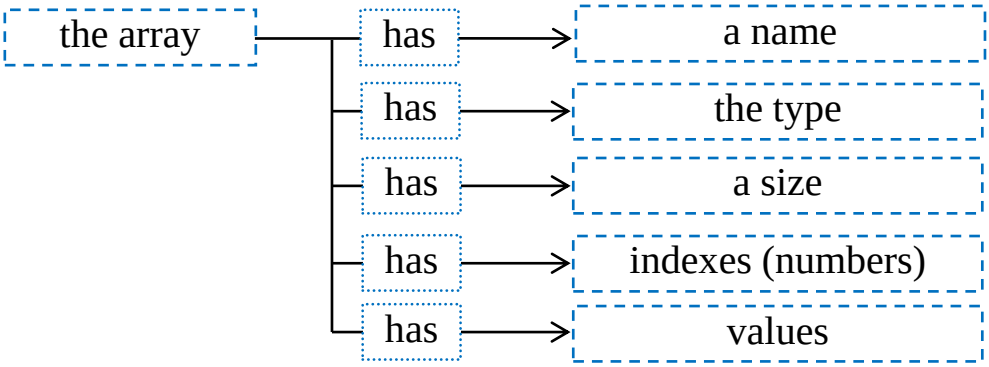
Example of a two-dimensional array

The Latin square array *D*[3×3] has three different elements. The array will be Latin if each cell element is displayed in each column and in each row only once.

1	3	2
2	1	3
3	2	1

```
B[1][1]=1;
B[1][2]=3;
B[1][3]=2;
…
B[3][3]=1;
```

It should be noted that:



type → numeric, text, and so on;
size = number of elements;
index = element number;
value = array element.

Exercise 2.1

You have a matrix of the magic square type of order 3 (it is unique, the magic sum is 15, and the central number is 5): can you represent this matrix?

Exercise 2.2

Using this knowledge, write Python code that will create an array of the magic square type with n orders (where *n* is an odd number), if it exists.

One of the possible methods:

Step №1

Consider *n* – odd and equal to 5.

Step №2

Let's assume that the numbers are used from 1 to 25, and the magic sum is 65.

Step №3

The central cell is 13, and all opposite pairs relative to this central cell give a sum equal to 2 × 13 = 26.

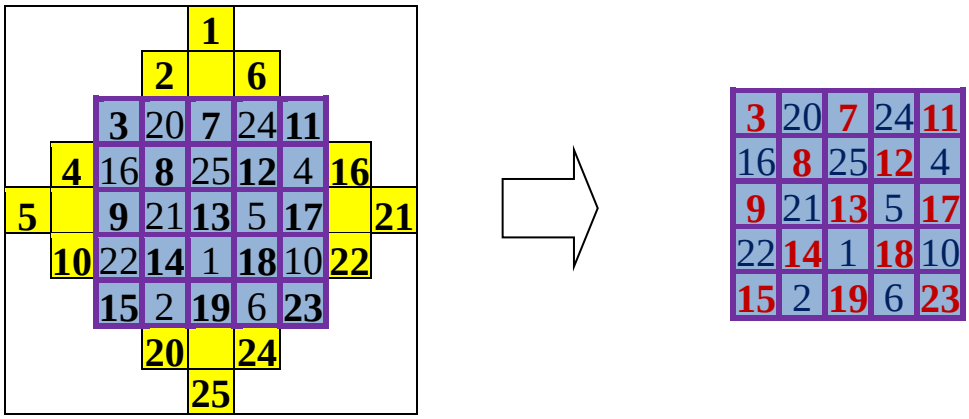


Fig. 2.4. Representation of the magic square

The numbers from 1 to 25 are arranged diagonally one after the other. When the numbers go out of bounds, they are transferred to the other side.

Continued on page 134

FR II.2. Tableaux d'effectifs bidimensionnel

Un tableau (voir fig. 2.3) dont les éléments ont deux index est appelé *bidimensionnel* ou *matrice*.

Notation d'un tableau bidimensionnel:
B[i][j]; B[i,j]; B(i,j);
ou alors $B_{i,j}$

B i \ j	1	2	3	…	m
1	B[1][1]	B[1][2]	B[1][3]	…	B[1][m]
2	B[2][1]	B[2][2]	B[2][3]	…	B[2][m]
3	B[3][1]	B[3][2]	B[3][3]	…	B[3][m]
…	…	…	…	…	…
n	B[n][1]	B[n][2]	B[n][3]	…	B[n][m]

i – numéro de ligne; *j* – numéro de colonne;
n – le nombre de lignes; *m* – le nombre de colonnes.

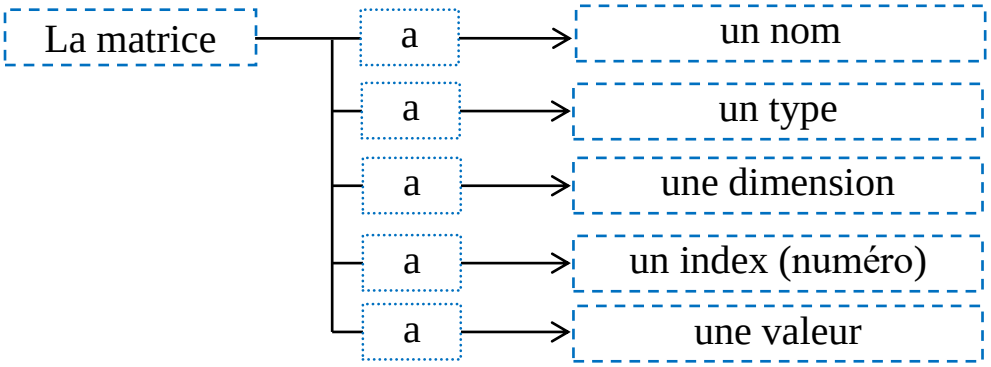
Fig. 2.3. Tableau bidimensionnel

Exemple de tableau bidimensionnel: le carré latin ***D*** d'ordre 3 a trois éléments distincts. Le tableau sera «*latin*» si chaque élément de la cellule n'apparaît qu'une seule fois dans chaque colonne et dans chaque ligne (chaque ligne et chaque colonne ne contient qu'un seul exemplaire).

1	3	2
2	1	3
3	2	1

```
D[1][1]=1;
D[1][2]=3;
D[1][3]=2;
…
D[3][3]=1;
```

Il convient de remarquer que:



Exercise 2.1 Vous avez une matrice de type carré magique d'ordre 3 (notons qu'il est unique, la somme magique est 15 et le chiffre du centre est 5): pouvez vous représenter cette matrice?

Exercice 2.2

Utiliser les connaissances acquises pour écrire un code en Python qui pourra créer un carré magique (voir fig. 2.4) à ***n*** ordres (n – un nombre impair) s'il existe.

Une des méthodes possible pour déterminer:

Étape №1

Prenons n – impair et égale à 5.

Étape №2

Les nombres à placer vont de 1 à 25 et la somme magique est 65.

Étape №3

La case centrale est 13 et tous les couples opposés par rapport à cette case centrale donnent une somme égale à 2 x 13 = 26.

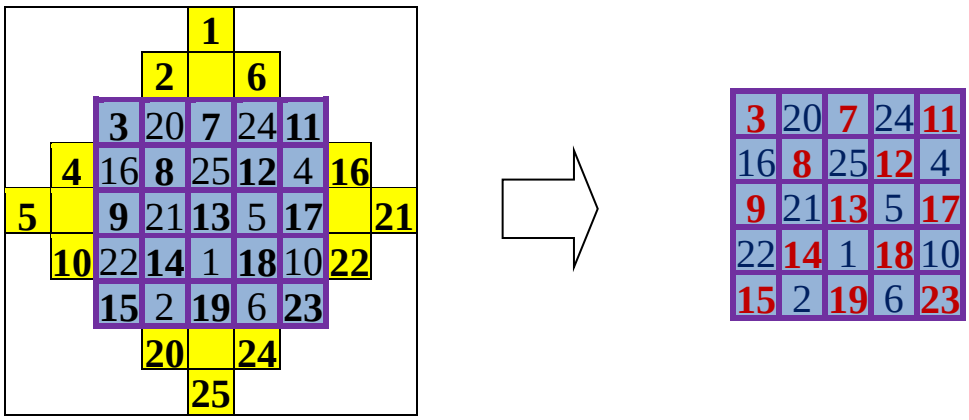


Fig. 2.4. Représentation du carré magique

Les nombres de 1 à 25 sont placés à la suite les uns des autres, le long des diagonales. Ceux qui débordent sont "enroulés" de l'autre côté.

Suite à la page 137

RU II.3. Работа над одномерным массивом

1) Массив, состоящий из N элементов, который имеет название A и, кроме того, все элементы массива получают значение, равное 0.

```
A = [0]×N
# A=[]×N не оставляйте квадратную скобку без элемента внутри
A = [0]×3
# [0, 0, 0]
A = [2]×4
# [2, 2, 2, 2]
```

2) Вы можете установить значения элементов массива при их объявлении

```
# Массив D имеет 4 элемента
D=[3, 6, 9, 0]
# Или можно ввести эти элементы шаг за шагом.
# Во-первых, объявляем массив под именем D с 4 элементами.
D=[0]×4
# Введение элементов
D[0]=3
D[1]=6
D[2]=9
D[3]=0
```

 # Объявление массива с неизвестным количеством элементов в Python

```
F=[]
# Введение элементов в массив G
G=[3,6,9,0]
# Введение элемента в конце массива G
G.append(11)
# Результат будет так:
```

```
G=[3,6,9,0,11]
# Если мы выполняем ту же команду с другой цифрой
G.append(12)
# результат будет следующим (11 будет заменен на 12, потому
# что массив не был объявлен с неизвестным количеством
# элементов):
G=[3,6,9,0,12]
```

3) Как интерактивно вводить данные в одномерный массив?

```
B=[]
n=int(input())
for i in range(n):
    B.append(int(input()))
```

```
# объвление массив “резиновый”
# порядок массива (минус 1)
# начало цикла с 0.
# введение в массив
```

Чтобы стало более понятно и определенно, покажем:

```
B=[]
n=int(input("Введите размер массива = "))
for i in range(n):
    B.append(int(input("B["+str(i)+"]= ")))
```

Как использовать массив этого одномерного типа?

```
# Импортировать модуль
from array import *
# Инициализировать переменную как массив array
# valeur = array('u',['2', '5', '4', '5', '5', '9', '4'])
valeur = array('i',[2, 5, 4, 5, 5, 9, 4])
# Распечатать весь одномерный массив, для проверки
print(valeur)
# Результат: array('I', [2, 3, 4, 7])
# Распечатать первое значение одномерного массива для проверки
```

```
print(valeur[0])
# Результат: 2
# Распечатать последнее значение одномерного массива для проверки
print(valeur[-1])
# Результат: 4
print(valeur.buffer_info())
print(valeur.count(5)) # Подсчитайте количество раз, когда в таблице значений встречается цифра 5
# Определите длину элемента массива в байтах в соответствии с типом кодом.
print('Длина в байтах. = '+str(valeur.itemsize)+' для типа '"+(valeur.typecode)+"")
print(valeur[:])  # Печать всей таблицы(или print (valeur)).
print(valeur[:2]) # Печать первых двух элементов
print(valeur[1:]) # Распечатать все элементы, которые стоят после первого

# Удалить элемент из списка
import array as mas
new_mass = mas.array('d', [5.3, 12, 4.9, -7.2, 4])
new_mass.remove(12) # Удалить элемент '12' из списка
print(new_mass)

# Результат: array('d', [5.3, 4.9, -7.2, 4.0])
```

Продолжение на странице 140

EN II.3. Working on a one-dimensional array

1) An array consisting of N elements, which has the name A and, in addition, all elements of the array get a value equal to 0.

```
A = [0]×N
# A=[]×N don't leave a square bracket without an element inside
A = [0]×3
# [0, 0, 0]
A = [2]×4
# [2, 2, 2, 2]
```

2) You can set the values of array elements when declaring them

```
# Array D has 4 elements
D=[3, 6, 9, 0]
# Or you can enter these elements step by step.
# First, declare an array named D with 4 elements.
D=[0]×4
# The introduction of elements
D[0]=3
D[1]=6
D[2]=9
D[3]=0
```

 # Declaring an array with an unknown number of elements in Python

```
F=[]
# The introduction of elements in the array G
G=[3,6,9,0]
# Introducing an element at the end of the array G
G. append(11)
# The result will be:
G=[3,6,9,0,11]
# If we execute the same command with a different digit
G. append(12)
# the result will be as follows (11 will be replaced by 12, because
# that the array was not declared with an unknown number
```

# of elements):

```
G=[3,6,9,0,12]
```

3) How to interactively enter data into a one-dimensional array?

```
B=[]
n=int(input())
for i in range(n):
   B.append(int(input()))
```

```
# ads with an array of “rubber”
# array order (subtract 1)
# start of loop with 0.
# introduction to the array
```

To make it more clear and definite, we will show:

```
B=[]
n=int(input("Enter the size of the array = "))
for i in range(n):
   B.append(int(input("B["+str(i)+"]= ")))
```

How do I use an array of this one-dimensional type?

```
# Import a module
from array import *
# Initialize a variable as an array
# valeur = array('u',['2', '5', '4', '5', '5', '9', '4'])
valeur = array('i',[2, 5, 4, 5, 5, 9, 4])
# Print the entire one-dimensional array, for verification
print(valeur)
# The result: array('I', [2, 3, 4, 7])
# Print the first value of a one-dimensional array for verification
print(valeur[0])
# The result: 2
# Print the last value of a one-dimensional array for verification
print(valeur[-1])
# The result: 4
print(valeur.buffer_info())
```

print(valeur.count(5)) # Count the number of times the number 5 occurs in the value table

Determine the length of the array element in bytes according to the type of code.

print('Length in bytes. = '+str(valeur.itemsize)+' for the type " '+(valeur.typecode)+' " ')

print(valeur[:]) # Print the entire table (or print (valeur)).

print(valeur[:2]) # Print the first two elements

print(valeur[1:]) # Print all items that are after the first one

Remove an item from a given list

import array as mas

new_mass = mas.array('d', [5.3, 12, 4.9, -7.2, 4])

new_mass.remove(12) # Remove item '12' from the list

print(new_mass)

The result: array('d', [5.3, 4.9, -7.2, 4.0])

Continued on page 146

FR II.3. Travail sur les tableaux d'effectifs unidimensionnel

1) Un tableau composé de N éléments qui a pour nom A et en plus tous les éléments du tableau reçoivent une valeur égale à 0.

A = [0]*N

A=[]*N interdit sans élément dans le crochet

A = [0]*3

[0, 0, 0]

A = [2]*4

[2, 2, 2, 2]

2) Vous pouvez définir la valeur des éléments du tableau lorsque vous les déclarez.

Tableau D ayant 4 éléments

```
D=[3, 6, 9, 0]
# Ou alors nous pouvons introduire ces éléments pas à pas
# Premièrement nous déclarons le tableau au nom D avec 4 éléments
D=[0]*4
# Introduction des éléments
D[0]=3
D[1]=6
D[2]=9
D[3]=0

# Déclaration d'un tableau avec un nombre inconnu d'éléments en python
F=[]
#Introduction des éléments dans le tableau G
G=[3,6,9,0]
#Introduction d'un élément à la fin du tableau G
G.append(11)
# le résultat est le suivant:
G=[3,6,9,0,11]
# Si nous exécutons la même commande avec un autre chiffre
G.append(12)
# le résultat sera le suivant (11 sera remplacé par 12, parce que le
# tableau n'a pas été déclaré avec un nombre inconnu d'éléments):
G=[3,6,9,0,12]
```

3) Comment introduire interactivement les données dans un tableau unidimensionnel ?

```
B=[]
n=int(input())
for i in range(n):
    B.append(int(input()))
```

```
# déclaration du tableau "élastique"
# l'ordre du tableau (moins 1)
# début de la boucle avec 0
# introduction dans le tableau
```

Pour être plus compréhensible et explicite, nous aurons:

```
B=[]
n=int(input("Introduire la taille du tableau = "))
for i in range(n):
   B.append(int(input("B["+str(i)+"]= ")))
```

Comment utiliser array dans ce type unidimensionnel?

```
# Importer le module
from array import *
# Initialiser une variable comme tableau array
# valeur = array('u',['2', '5', '4', '5', '5', '9', '4'])
valeur = array('i',[2, 5, 4, 5, 5, 9, 4])
# Imprimer toute le tableau unidimensionnel pour vérifier
print(valeur)
# Résultat: array('I', [2, 3, 4, 7])
# Imprimer la première valeur du tableau unidimensionnel pour vérifier
print(valeur[0])
# Résultat: 2
# Imprimer la dernière valeur du tableau unidimensionnel pour vérifier
print(valeur[-1])
# Résultat: 4
print(valeur.buffer_info())
print(valeur.count(5)) # Compter le nombre de fois qu'il y a 5 dans le tableau valeur
# Déterminer la longueur en octets d'un élément du tableau selon le code d'indication du type.
print('La longueur en octets = '+str(valeur.itemsize)+' pour le type " '+(valeur.typecode)+' " ')
print(valeur[:])  #Imprimer tout le tableau (ou encore print(valeur))
print(valeur[:2]) #Imprimer les deux premiers éléments
```

print(valeur[1:]) #Imprimer tous les éléments qui viennent après le premier

```
# Supprimer un élément d'une liste donnée
import array as mas
new_mass = mas.array('d', [5.3, 12, 4.9, -7.2, 4])
new_mass.remove(12) # Retirer de la liste l'élément '12'
print(new_mass)

# Résultat: array('d', [5.3, 4.9, -7.2, 4.0])
```

Suite à la page 152

RU II.4. Работа над двумерными массивами (матрицами)

Представление матрицы: (см. рис. 2.5) на языке программирования Python она представлена как список строк элементов

$B_{[1][1]}$	$B_{[1][2]}$	$B_{[1][3]}$
$B_{[2][1]}$	$B_{[2][2]}$	$B_{[2][3]}$

Рис. 2.5 Матрица из 2 строк и 3 столбцов

Это выглядит следующим образом в Python:

$$B = [[B_{[1][1]}, B_{[1][2]}, B_{[1][3]}],[B_{[2][1]}, B_{[2][2]}, B_{[2][3]}]]$$

Обозначается так B[i][j], где i – номер строки; j – номер столбца.

2) Индивидуальная обработка элементов.

Для обработки каждого элемента можно использовать следующие коды:

```
n=len(B)                # количество строк
i=0                     # начальное значение (любое)
m=len(B[i])             # количество столбцов
for i in range(n):      # выбор строк
    for j in range(m):  # выбор столбцов
        B[i][j]+=3      # B[i][j]= B[i][j]+3
```

Оптимизировав программу, будем иметь:

```
for i in range(len(B)):
    for j in range(len(B[i])):
        B[i][j]+=3
```

С более подробной информацией получим:

```
# Ввод данных в матрицу B
B=[[2,4,1,5],[-3,0,5,6]]
n=len(B)          # количество строк здесь равно 2 (от 0 до 1)
i=0 # начальное значение для определения размера m
m=len(B[i])       # количество строк здесь равно 4 (от 0 до 3)
for i in range(n):
    for j in range(m):
        # добавить к каждому элементу матрицы B значение 3
        B[i][j]+=3 # или B[i][j]= B[i][j]+3
print(B)          # выводить матрицы
print(B[0])       # выводить первой строки
print(B[1])       # выводить второй строки
```

Еще один способ печати массива в форме матрицы

```
for i in range(len(B)):
    for j in range(len(B[i])):
        B[i][j]+=3
        print(B[i][j], end=' ')
    print()
```

Печать матрицы с помощью инструкций «***row***» и «***elem***»

```
for row in B:          # для каждой строки в матрице сделайте:
   for elem in row: #для каждого элемента в строке сделайте:
      print(f'для каждой строки = {row}, имеем эл-т = {elem}')
```

Печать матрицы с помощью инструкции «join» (прикрепить)

```
for row in B:
   print('  '.join(list(map(str, row))))
```

3) Как интерактивно вводить данные в матрицу?

Сначала узнаем, как работает метод «*split*»?

Пример № 1

```
texte = "он разбивает строку"
M1 = texte.split()
print(M1)
```

```
Результат:
['он', 'разбивает', 'строку']
```

Пример №2

```
texte = "Разделите строку, используя запятую, за
которой следует пробел, в качестве разделителя"
M2 = texte.split(",  ")
print(M2)
```

```
Результат:
['Разделите строку', 'используя запятую', 'за которой
следует пробел', 'в качестве разделителя']
```

Пример №3

```
texte = "Разделите строку на 2, используя первую запятую, за которой следует пробел, в качестве разделителя"
M3 = texte.split(",  ", 1) #
```

```
Результат:
['Разделите строку на 2', 'используя первую запятую, за которой следует пробел, в качестве разделителя']
```

Как работает метод «*map*»?

Функция ***map()*** представлена следующим образом (ее синтаксис):

map(function, iterables),

где «***function»*** – функция, которая решает какую-либо операцию; «***iterables***» – последовательность элементов, которая будет проходить эту операцию (преобразование). Таким образом, функция ***map()*** позволяет выполнять указанную функцию для каждого элемента «iterable». Результат просматривается в списке функцией ***list*** или в ***typle***; повторяющийся элемент отправляется функции в качестве параметра.

Пример №1 Предположим, что у нас есть группа из 6 слов. Определите, сколько раз в каждом слове встречается латинская буква 'a' и отобразите результат на экране.

```
# ввести слова в «tuple» или в «list»
z = ['maman','panthère','magasinage','liste','amour','tomate']
# Создать функцию, которая будет подсчитывать
количество раз, когда мы находим латинскую букву 'a'
в слове
def fcompter(mot):
    n = mot.count('a')
    return n
# Используйте функцию map(), чтобы вывести этот
# подсчет на все слова в списке
m = map (fcompter, z)
# выводить результат используя функции list или tuple
print(list(m))
```

Смотрите, как def работает в разделе о функций

```
# Результат:
[2, 1, 3, 0, 1, 1]
```

Пример №2 Допустим, что у нас есть список (tuple) из 5 студентов, и каждый из них имеет 3 оценки по 3 различным дисциплинам:

№ студентов	Оценки по информатике	Оценки по истории	Оценка по экономике
1	13	15	14
2	5	5	5
3	11	2	5
4	5	6	10
5	7	9	11

Задача состоит в том, чтобы найти среднюю оценку каждого студента и распечатать.

Если мы хотим использовать модуль ***map()***, мы должны сначала создать функцию, которая позволит найти среднее значение из 3 оценок. Объяснения функции ***def*** – см. раздел, предназначенный для функций.

```
def mafonction(note1, note2, note3):
    avg = (note1+note2+note3)/3
    return avg
```

После введем оценки каждой дисциплины в 3 списка. Таким образом, мы будем иметь:

```
liste1 = [13, 5, 11, 5, 7]
liste2 = [15, 5, 2, 6, 9]
liste3 = [14, 5, 5, 10, 11]
```

Ввести функции ***map()***, чтобы запустить функцию «mafonction» на каждый iterable.

```
x = map (mafonction, liste1, liste2, liste3)
```

Печать результата путем интеграции функции ***list()*** или ***tuple()***

```
print(list(x))
# Результат: [14,0, 5,0, 6,0, 7,0, 9,0]
```

Для того, чтобы обобщить предложим следующую функцию:

```
def mafonction(note1, note2, note3):
    avg = (note1+note2+note3)/3
    return avg
x = map (mafonction, [13, 5, 11, 5, 7],[15, 5, 2, 6, 9],[14,
5, 5, 10, 11])
print(list(x))
# Результат: [14,0, 5,0, 6,0, 7,0, 9,0]
```

Вариант №1

```
n=5          # количество строк, например 5
D = []       # объявить таблицу D – "резиновая"
for i in range(n):
   D.append(list(map(int, input().split())))
```

Вариант №2

```
n=3          # количество строк
D = [list(map(int, input().split())) for i in range(n)]
```

Вариант №3

```
D = [list(map(int, input(f'Элементы строки {i+1} расположены
        на расстоянии друг от друга: ').split())) for i in range(3)]
```

```
# Давайте представим данные так:
Элементы строки 1 расположены на расстоянии друг от друга:
                                        6 -3 7 0
Элементы строки 2 расположены на расстоянии друг от друга:
                                        2 8
Элементы строки 3 расположены на расстоянии друг от друга:
                                        -4 1
```

С использованием print(D), получим следующий результат:
[[6, -3, 7, 0], [2, 8], [-4, 1]]

Продолжение на странице 159

EN II.4. Working on two-dimensional arrays (matrices)

1) Matrix representation: (see fig 2.5). In the Python programming language, it is represented as a list of strings of elements

$B_{[1][1]}$	$B_{[1][2]}$	$B_{[1][3]}$
$B_{[2][1]}$	$B_{[2][2]}$	$B_{[2][3]}$

Fig. 2.5. Matrix of 2 rows and 3 columns

It looks like this in Python:

$B = [[B_{[1][1]}, B_{[1][2]}, B_{[1][3]}],[B_{[2][1]}, B_{[2][2]}, B_{[2][3]}]]$

Denoted as B[i][j], where ***i*** is the row number; ***j*** is the column number.

2) Individual processing of elements

You can use the following codes to process each element:

```
n=len(B)                # number of rows
i=0                     # initial value (any)
m=len(B[i])             # number of columns
for i in range(n):      # select rows
    for j in range(m):  # select columns
        B[i][j]+=3      # B[i][j]= B[i][j]+3
```

After optimizing the program, we will have:

```
for i in range(len(B)):
    for j in range(len(B[i])):
        B[i][j]+=3
```

With more detailed information, we will get:

```
# Entering data into matrix B
B=[[2,4,1,5],[-3,0,5,6]]
n=len(B)          # the number of rows here is 2 (from 0 to 1)
i=0 # initial value for determining the size of m
m=len(B[i])       # the number of rows here is 4 (from 0 to 3)
for i in range(n):
    for j in range(m):
        # add the value 3 to each element of matrix B
        B[i][j]+=3 # or B[i][j]= B[i][j]+3
print(B)          # output matrices
print(B[0])       # output the first line
print(B[1])       # output the second line
```

Another way to print an array in the form of a matrix

```
for i in range(len(B)):
    for j in range(len(B[i])):
        B[i][j]+=3
        print(B[i][j], end=' ')
    print()
```

Printing a matrix using the «***row***» and «***elem***» instructions

```
for row in B:          # for each row in the matrix, do:
    for elem in row: # for each element in the row, do:
        print(f'for each row = {row}, we have element = {elem}')
```

Printing the matrix using the «join» instruction (attach)

```
for row in B:
    print('  '.join(list(map(str, row))))
```

3) How do I interactively enter data into a matrix?

First, let's find out how the «split» method works?

Example 1

```
texte = "it splits the string "
M1 = texte.split()
print(M1)
```

```
The result:
['it', 'splits', 'the', 'string']
```

Example 2

```
texte = "Separate the string, using a comma, followed by a space, as the separator"
M2 = texte.split(",  ")
print(M2)
```

```
The result:
['Separate the string, using a comma, followed by a space, as the separator']
```

Example 3

```
texte = " Divide the string by 2, using the first comma, followed by a space as the separator"
M3 = texte.split(", ", 1) #
print(M3)
```

```
The result:
['Divide the string by 2', 'using the first comma, followed by a space as the separator']
```

The ***map()*** function is represented as follows (its syntax is):

map(function, iterables),

where «***function»*** is a function that solves an operation; «***iterables***» is a sequence of elements that will undergo this operation (transformation). Thus, the ***map()*** function allows you to perform the specified function for each «iterable» element. The result is viewed in a list by the ***list*** function or in ***typle***; the duplicate element is sent to the function as a parameter.

Example 1 Suppose we have a group of 6 words. Determine how many times the Latin letter '***a***' occurs in each word and display the result on the screen.

```
# enter words in «tuple» or in «list»
z = ['maman','panthère','magasinage','liste','amour','tomate']
# # Create a function that will count the number of times
we find the Latin letter 'a' in a word

def fcompter(mot):
    n = mot.count('a')
    return n
# Use the map() function to output this
# counting on all words in the list
m = map (fcompter, z)
# output the result using the list or tuple functions
print(list(m))
```

See how "***def***" works in the section about functions

```
# The result:
[2, 1, 3, 0, 1, 1]
```

Example 2 Let's say we have a list via «*tuple*» of 5 students, and each of them has 3 grades in 3 different disciplines:

Assessment of the economy.

Number of students	Mark in computer science	History mark	Mark in economics
1	13	15	14
2	5	5	5
3	11	2	5
4	5	6	10
5	7	9	11

The task is to find the average grade of each student and print it out.

If we want to use the ***map()*** module, we must first create a function that will allow us to find the average of 3 estimates. For explanations of the ***def*** function, see the section dedicated to functions.

```
def mafonction(note1, note2, note3):
    avg = (note1+note2+note3)/3
    return avg
```

After that, we will enter the grades of each discipline in 3 lists. So we will have:

```
liste1 = [13, 5, 11, 5, 7]
liste2 = [15, 5, 2, 6, 9]
liste3 = [14, 5, 5, 10, 11]
```

Enter the *map()* function to run the «*myfonction*» function on each iterable.

```
x = map (myfonction, liste1, liste2, liste3)
```

Print the result by integrating the ***list()*** or ***tuple()*** function

```
print(list(x))
# The result: [14,0, 5,0, 6,0, 7,0, 9,0]
```

To generalize, we propose the following function:

```
def myfonction(note1, note2, note3):
    avg = (note1+note2+note3)/3
    return avg
x = map (myfonction, [13, 5, 11, 5, 7],[15, 5, 2, 6, 9],[14, 5, 5, 10, 11])
print(list(x))
# The result: [14,0, 5,0, 6,0, 7,0, 9,0]
```

Option 1

```
n=5          # number of rows, for example 5
D = []       # declare table D – “elastic”
for i in range(n):
   D.append(list(map(int, input().split())))
```

Option 2

```
n=3          # number of rows
D = [list(map(int, input().split())) for i in range(n)]
```

Option 3

```
D = [list(map(int, input(f'The elements of row {i+1} are located at a
        distance from each other:').split())) for i in range(3)]
```

```
# Let's present the data like this:
The elements of row 1 are located at a distance from each other: 6 -
                                                    3 7 0
The elements of row 2 are located at a distance from each other: 2 8
The elements of row 3 are located at a distance from each other: -4
                                                    1
```

Using print(D), we get the following result:
[[6, -3, 7, 0], [2, 8], [-4, 1]]

Continued on page 161

FR II.4. Travail sur les tableaux d’effectifs bidimensionnel (matrices)

1) Représentation de la matrice: (voir fig. 2.5). Dans le langage de programmation Python, elle est représentée comme une liste de chaînes d’éléments.

$B_{[1][1]}$	$B_{[1][2]}$	$B_{[1][3]}$
$B_{[2][1]}$	$B_{[2][2]}$	$B_{[2][3]}$

Fig. 2.5 – Matrice de 2 lignes et 3 colonnes

Elle se présente de la manière suivante dans Python:

$$B = [[B_{[1][1]}, B_{[1][2]}, B_{[1][3]}],[B_{[2][1]}, B_{[2][2]}, B_{[2][3]}]]$$

La notation est B[i][j] où ***i*** - numéro de rangée; ***j*** - numéro de colonne.

2) Traitement individuel des éléments: Pour traiter chaque élément, vous pouvez utiliser les codes suivants:

```
n=len(B)                  # Nombre de rangées
i=0                       # Valeur initiale (n'importe laquelle)
m=len(B[i])               # Nombre de colonnes
for i in range(n):        # choix des rangées
    for j in range(m):    # choix des colonnes
        B[i][j]+=3        # B[i][j]= B[i][j]+3
```

Pour optimiser le programme, nous aurons:

```
for i in range(len(B)):
    for j in range(len(B[i])):
        B[i][j]+=3
```

Avec plus de détails, nous aurons:

```
# Introduire les données dans la matrice B
B=[[2,4,1,5],[-3,0,5,6]]
n=len(B)        # Nombre de rangées ici égale à 2 (de 0 à 1)
i=0 # Valeur initiale, elle est utile pour déterminer la taille m
m=len(B[i])     # Nombre de colonnes ici égale à 4 (de 0 à 3)
for i in range(n):
    for j in range(m):
        # Ajouter à chaque élément de la matrice B la valeur 3
        B[i][j]+=3 # ou bien B[i][j]= B[i][j]+3
print(B)          # Imprimer la matrice
print(B[0])       # Imprimer la première rangée
print(B[1])       # Imprimer la deuxième rangée
```

Une autre façon d'imprimer le tableau en forme de matrice

```
for i in range(len(B)):
    for j in range(len(B[i])):
        B[i][j]+=3
        print(B[i][j], end=' ')
    print()
```

Imprimer la matrice en utilisant les instructions «***row***» et «***elem***»

```
for row in B:              # pour chaque rangée dans la matrice faites:
    for elem in row:       # pour chaque élément dans la rangée faites:
        print(f'Pour la rangée = {row}, on a l'élément = {elem}')
```

Imprimer la matrice en utilisant l'instructions «***join***» ***(joindre)***

```
for row in B:
    print('  '.join(list(map(str, row))))
```

3) Comment introduire interactivement les données dans une matrice?

Savoir d'abord comment fonctionne la méthode «*split*»?

Exemple 1

```
texte = "il décompose une chaîne"
M1 = texte.split()
print(M1)
```

```
Résultat:
['il', 'décompose', 'une', 'chaîne']
```

Exemple 2

```
texte = "Diviser la chaîne, en utilisant une virgule, suivie d'un espace, comme séparateur"
M2 = texte.split(",  ")
print(M2)
```

```
Résultat :
['Diviser la chaîne', 'en utilisant une virgule', 'suivie d'un espace', 'comme séparateur']
```

Exemple 3

```
texte = "Diviser la chaîne en 2, en utilisant la 1-ère virgule, suivie d'un espace, comme séparateur"
M3 = texte.split(",  ", 1) #
print(M3)
```

```
Résultat :
['Diviser la chaîne en 2', 'en utilisant la 1-ère virgule, suivie d'un espace, comme séparateur']
```

Comment fonctionne la méthode «*map*»?

La fonction ***map()*** est présentée sous la forme (sa syntaxe):

map(function, iterables),

où «***function***» est une fonction qui résous une opération quelconque et «***iterables***» – une suite d'éléments qui subira cette opération (transformation). Donc, la fonction ***map()*** permet d'exécuter une fonction spécifiée pour chaque élément d'un itérable. Le résultat est visionné dans une liste par la fonction ***list*** ou dans un **tuple**; l'élément de l'itérable est envoyé à la fonction en tant que paramètre.

Exemple 1 Considérons que nous avons un groupe de 6 mots. Déterminer le nombre de fois qu'on rencontre la lettre latine 'a' dans chaque mot et afficher le résultat.

```
#Introduire les mots dans un tuple ou dans une liste
z = ['maman','panthère','magasinage','liste','amour','tomate']
# Créer la fonction qui comptera le nombre de fois qu'on
# trouve la lettre latine 'a' sur un mot
def fcompter(mot):
    n = mot.count('a')
    return n
# Utiliser la fonction map() pour ressortir ce comptage sur
# tous les mots de la liste
m = map (fcompter, z)
# Imprimer le résultat en utilisant la fonction list ou tuple
print(list(m))
```

voir comment fonctionne def dans la partie des fonctions

```
# Résultat:
[2, 1, 3, 0, 1, 1]
```

Exemple 2 Considérons que nous avons une liste (*tuple*) de 5 étudiants et chacun d'eux possède 3 notes sur 3 disciplines différentes:

№ des étudiants	Les notes de l'Informatique	Les notes de l'Histoire	Les notes de l'Économie
1	13	15	14
2	5	5	5
3	11	2	5
4	5	6	10
5	7	9	11

Le devoir consiste à trouver la valeur moyenne de chaque étudiant et de l'imprimer.

Si nous voulons utiliser le module ***map()***, nous devons tout d'abord créer une fonction qui permet de trouver la valeur moyenne des 3 notes. Les explications sur la fonction def – voir la partie réservée aux fonctions.

```
def mafonction(note1, note2, note3):
    avg = (note1+note2+note3)/3
    return avg
```

Après, introduire les notes de chaque matière dans 3 listes. Ainsi, nous aurons:

```
liste1 = [13, 5, 11, 5, 7]
liste2 = [15, 5, 2, 6, 9]
liste3 = [14, 5, 5, 10, 11]
```

Introduire la fonction ***map()*** pour exécuter la fonction «mafonction» sur chaque itérable.

```
x = map (mafonction, liste1, liste2, liste3)
```

Imprimer le résultat en intégrant la fonction ***list()*** ou ***tuple()***

```
print(list(x))
# Résultat: [14.0, 5.0, 6.0, 7.0, 9.0]
```

Pour condenser tout ce qui a été dit en un seul bloc on aura la fonction suivante:

```
def mafonction(note1, note2, note3):
    avg = (note1+note2+note3)/3
    return avg
x = map (mafonction, [13, 5, 11, 5, 7],[15, 5, 2, 6, 9],[14, 5, 5, 10, 11])
print(list(x))
# Résultat: [14.0, 5.0, 6.0, 7.0, 9.0]
```

Variante 1

```
n=5          # Nombre de rangées, par exemple 5
D = []       # déclaration du tableau D “élastique”
for i in range(n):
    D.append(list(map(int, input().split())))
```

Variante 2

```
n=3          # Nombre de rangées
D = [list(map(int, input().split())) for i in range(n)]
```

Variante 3

```
D = [list(map(int, input(f'les éléments de la rangée {i+1} espacés:
                                                    ').split())) for i in range(3)]
```

```
# Introduisons les données de cette manière:
Les éléments de la rangée 1 espacés: 6 -3 7 0
Les éléments de la rangée 2 espacés: 2 8
Les éléments de la rangée 3 espacés:-4 1
```

Avec print(D), nous aurons comme résultat:
[[6, -3, 7, 0], [2, 8], [-4, 1]]

Suite à la page 163


RU Глава III. ОПЕРАЦИЯ С NUMPY
EN Chapter III. OPERATION WITH NUMPY
FR Chapitre III. OPÉRATION AVEC NUMPY

RU III.1. Библиотека NumPy и некоторые демонстрации на массивах

1) Обновить пакет *pip*

Открыть программу python (python.exe) и выполнить команду с терминала Windows (*cmd или powershell*) со статусом или правом администратора:

...\ python\python.exe -m pip install --upgrade pip

2) Установка библиотеки

Введите команду из программы Python с терминала Windows (cmd или powershell) со статусом или правом администратора:

C:\>...*pip install numpy*

или так

...\python\python.exe pip install numpy

или так:

a) запишите имя и, если возможно, версию (numpy==1.19.1) библиотеки для установки в файл (например: requirements.txt); вы могли бы установить несколько библиотек за один раз, выровняв эти библиотеки по строкам в файле; с помощью команды *pip freeze* можно получить список установленных библиотек;

b) установите библиотеки из файла, запустив эту команду с терминала Windows (cmd или powershell) со статусом или правом администратора: ***pip install-r requirements.txt*** (чтобы удалить: *pip uninstall-r requirements.txt*).

3) если пункт 1 и 2 не дают ожидаемого результата, воспользуйтесь следующими шаги для установки библиотеки numpy:

a) загрузите библиотеку numpy, которая совместима с вашим установленным программным обеспечением Python (*numpy-*.whl*) из этого источника (*https://www.lfd.uci.edu/~gohlke/pythonlibs/#numpy*) и сохраните его в любую папку.

b) откройте терминал *Windows* (*cmd* или *powershell*) со статусом или правом администратора.

c) введите следующую команду:

pip install Путь_папка\имя_библиотеки.whl

4) импорт библиотеки в приложение

import numpy или *import numpy as np* (np – альтернативное имя)

Пример №1 Создайте одномерный массив из 6 натуральных элементов из NumPy.

Вариант №1

```
import numpy
tabl_unid = numpy.array([2, 6, 3, 8, 10, 7])
print(tabl_unid)
```

```
# Результат:
[2  6  3  8  10  7]
```

Вариант №2 (Использование *list*)

```
import numpy as np
tabl_unid = np.array([2, 6, 3, 8, 10, 7])
print(tabl_unid)
```

```
# Результат:
[2  6  3  8  10  7]
```

Вариант №3 (Использование *set*)

```
import numpy as np
tabl_unid = np.array({2, 6, 3, 8, 10, })
print(tabl_unid)
```

```
# Результат:
{2  6  3  8  10  7}
```

Можно также использовать *tuple*.

Пример №2 Создайте двумерный массив натуральных элементов из 3 строк и 4 столбцов из NumPy.

Решение

```
import numpy as np
tabl_bidim = np.array([[3, 9, 3, 2],  [1, 7, 5, 0]])
print(tabl_bidim)
```

```
# Результат:
 [[3  9  3  2]
 [1  7  5  0]]
```

Продолжение на странице 166

EN III.1. The NumPy library and some demonstrations on tables

NB. Read the preface for more information on installing Python.

1) Update the pip package

Open the Python program (*python.exe*) and run the command from a Windows terminal (*cmd* or *powershell*) with the status or administrator rights:

...\ python\python.exe -m pip install --upgrade pip

2) Installing the library

Enter a command from the Python program from a Windows terminal (cmd or powershell) with the status or administrator rights:

C:\>...*pip install numpy*

or so

...\python\python.exe pip install numpy

or so:

a) write down the name and, if possible, the version (numpy==1.19.1) of the library to install in the file (for example: requirements.txt); you could install multiple libraries at a time by aligning these libraries to the lines in the file; using the *pip freeze* command, you can get a list of installed libraries;

b) install the libraries from the file by running this command from a Windows terminal (*cmd or powershell*) with the status or administrator rights: ***pip install-r requirements.txt*** (to remove: *pip uninstall-r requirements.txt*).

3) if item 1 and 2 do not give the expected result, use the following steps to install numpy:

a) load the numpy library which is compatible with your software installed Python (*numpy-*.whl*) from this source URL: *https://www.lfd.uci.edu/~gohlke/pythonlibs/#numpy* and save it in any folder.

b) open a *Windows terminal* (*cmd or powershell*) with the status or administrator rights.

c) type the following command:
pip install path_folder\library_name.whl

4) import the library to the application

import numpy или *import numpy as np* (*np* is an alternative name)

Example №1 Create a one-dimensional array of 6 natural elements from numpy.

Option 1

```
import numpy
tabl_unid = numpy.array([2, 6, 3, 8, 10, 7])
print(tabl_unid)
```

```
# The result:
[2  6  3  8  10  7]
```

Option №2 (Using ***list***)

```
import numpy as np
tabl_unid = np.array([2, 6, 3, 8, 10, 7])
print(tabl_unid)
```

```
# The result:
[2  6  3  8  10  7]
```

Option 3 (Using *set*)

```
import numpy as np
tabl_unid = np.array({2, 6, 3, 8, 10, })
print(tabl_unid)
```

```
# The result:
{2  6  3  8  10  7}
```

You can also use ***tuple***.

Example 2 Create a two-dimensional array of natural elements with 3 rows and 4 columns from NumPy.

```
import numpy as np
tabl_bidim = np.array([[3, 9, 3, 2],  [1, 7, 5, 0]])
print(tabl_bidim)
```

```
# The result:
 [[3  9  3  2]
 [1  7  5  0]]
```

Continued on page 168

FR III.1. La bibliothèque NumPy et quelques démonstrations sur tableaux

NB. Lire la préface pour avoir plus d'informations sur l'installation du Python.

1) Mettre à jour le paquet *pip*

Ouvrir le lieu(dossier) où se trouve le programme Python (*python.exe*) et exécuter la commande à partir d'un terminal *Windows* (*cmd* ou *powershell*) avec un statut ou profil d'administrateur:

...\python\python.exe -m pip install --upgrade pip

2) Installation de la bibliothèque

Taper la commande hors du programme Python à partir d'un terminal Windows (*cmd ou powershell*) avec un statut ou profil d'administrateur:

C:\>…*pip install numpy*

ou alors

...\python\python.exe *pip install numpy*

ou alors:

a) écrire le nom et si possible la version (numpy==1.19.1) de la bibliothèque à installer dans un fichier (par exemple: requirements.txt); vous pouviez installer plusieurs bibliothèques en une exécution en alignant ces bibliothèques ligne par ligne dans le fichier; avec la commande pip freeze, nous avons la liste des bibliothèques installées;

b) installer la bibliothèques à partir du fichier en lançant cette commande à partir d'un terminal Windows (*cmd* ou *powershell*) avec un statut ou profil d'administrateur: ***pip install –r requirements.txt*** (pour désinstaller: *pip uninstall –r requirements.txt*).

3) Dans le cas où le point 1 et 2 ne donne pas le résultat escompté, alors utiliser les étapes suivantes pour installer la bibliothèque numpy:

a) Téléchargez la bibliothèque ***numpy*** qui est compatible à votre logiciel Python installé (*numpy-*.whl*) *de cette source (https://www.lfd.uci.edu/~gohlke/pythonlibs/#numpy*) et sauvegarder le dans un dossier quelconque.

b) Ouvrez un terminal Windows (*cmd* ou *powershell*) avec un statut ou profil d'administrateur.

c) Tapez la commande suivante:

pip install Chemin_d'accès_au_dossier\nom_ bibliothèque.whl

4) Importation de la bibliothèque dans l'application

import numpy ou alors *import numpy as np* (np – nom alternatif)

Exemple 1. Créer un tableau unidimensionnel de 6 éléments naturels à partir de NumPy.

Variante 1

```
import numpy
tabl_unid = numpy.array([2, 6, 3, 8, 10, 7])
print(tabl_unid)
```

```
# Résultat:
[2  6  3  8  10  7]
```

Variante 2 (Utilisation d'une liste «*list*»)

```
import numpy as np
tabl_unid = np.array([2, 6, 3, 8, 10, 7])
print(tabl_unid)
```

```
# Résultat:
[2  6  3  8  10  7]
```

Variante 3 (Utilisation d'un «s*et*»)

```
import numpy as np
tabl_unid = np.array({2, 6, 3, 8, 10, })
print(tabl_unid)
```

```
# Résultat:
{2  6  3  8  10  7}
```

Nous pouvons également utiliser un «tu*ple*».

Exemple 2 Créer un tableau bidimensionnel d'éléments naturels de 3 rangées et 4 colonnes à partir de ***NumPy***.

```
import numpy as np
tabl_bidim = np.array([[3, 9, 3, 2],  [1, 7, 5, 0]])
print(tabl_bidim)
```

```
# Résultat:
 [[3  9  3  2]
 [1  7  5  0]]
```

Suite à la page 169

RU III.2. Изменение вида массива

Преобразуйте одномерный массив в двумерный

Пример №1

Предположим, что у нас есть список из 12 элементов. Составьте матрицу 3×4, то есть массив должен имеет 3 строки и 4 столбца.

Чтобы достичь цели, давайте используем метод ***reshape***. Чтобы не было ошибок, список должен содержать число элементов, равное размеру 3×4.

```
import numpy as np

liste1 = np.array([2, 4, 6, 8, 3, 5, 7, 9, -1, -2, -3, -4])
matrice1 = liste1.reshape(3, 4)
print(matrice1)
```

```
# Результат:
[[ 2  4  6  8]
 [ 3  5  7  9]
 [-1 -2 -3 -4]]
```

Преобразуйте двумерный массив в одномерный массив и переверните его.

Пример №2

Предположим, что имеется массив размером 3×4, то есть массив, содержит 3 строки и 4 столбца. Преобразуем простой список с помощью функции ***reshape(-1)***. После этого давайте перевернем его (с помощью функции ***flip(list, axis))***, чтобы первый элемент стал последним, а последний может занять место первого и так далее.

2	4	6	8
3	5	7	9
-1	-2	-3	-4

[[2, 4, 6, 8], [3, 5, 7, 9], [-1, -2, -3, -4]]

```
import numpy as np

matrice2 = np.array([[ 2,  4,  6,  8],  [ 3,  5,  7,  9],  [-1, -2, -3, -4]])
liste2 = matrice2.reshape(-1)
print(f'Полученный список   : {liste2}')
liste3 = np.flip(liste2, 0)
print(f' Перевернутый список: {liste3}')
```

```
# Результат:
Полученный список: [ 2  4  6  8  3  5  7  9 -1 -2 -3 -4]
Перевернутый список: [-4 -3 -2 -1  9  7  5  3  8  6  4  2]
```

Продолжение на странице 171

EN III.2. Changing the type of array

Convert a one-dimensional array to a two-dimensional one

Example 1

Suppose we have a list of 12 elements. Make a 3×4 matrix, that is, the array should have 3 rows and 4 columns.

To achieve the goal, let's use the reshape method. To avoid errors, the list must contain a number of elements equal to the size of 3×4.

```
import numpy as np

liste1 = np.array([2, 4, 6, 8, 3, 5, 7, 9, -1, -2, -3, -4])
matrice1 = liste1.reshape(3, 4)
print(matrice1)
```

```
# The result:
[[ 2  4  6  8]
 [ 3  5  7  9]
 [-1 -2 -3 -4]]
```

Convert a two-dimensional array to a one-dimensional array and flip it.

Example 2

Suppose there is a 3×4 array, that is, an array containing 3 rows and 4 columns. Transform a simple list using the ***reshape(-1)*** function. Then let's turn it on (using the function ***flip(list, axis)***) to the first element became last and the last may take place first and so on.

2	4	6	8
3	5	7	9
-1	-2	-3	-4

[[2, 4, 6, 8], [3, 5, 7, 9], [-1, -2, -3, -4]]

```
import numpy as np

matrice2 = np.array([[ 2,  4,  6,  8], [ 3,  5,  7,  9], [-1, -2, -3, -4]])
liste2 = matrice2.reshape(-1)
print(f'The resulting list : {liste2}')
liste3 = np.flip(liste2, 0)
print(f'The inverted list: {liste3}')
```

```
# The result:
The resulting list: [ 2  4  6  8  3  5  7  9 -1 -2 -3 -4]
The inverted list: [-4 -3 -2 -1  9  7  5  3  8  6  4  2]
```

Continued on page 177

FR III.2. Changer la forme d'un tableau

Transformez un tableau unidimensionnel à un tableau bidimensionnel

Exemple 1

Considérons que nous avons une liste de 12 éléments, composer une matrice de 3×4, c'est-à-dire que le tableau doit avoir 3 rangées et 4 colonnes.

Pour atteindre notre but, utilisons la méthode ***reshape***. Pour ne pas avoir d'erreurs, la liste doit contenir un nombre d'élément égale à la dimension 3×4.

```
import numpy as np

liste1 = np.array([2, 4, 6, 8, 3, 5, 7, 9, -1, -2, -3, -4])
matrice1 = liste1.reshape(3, 4)
print(matrice1)
```

```
# Résultat:
[[ 2  4  6  8]
 [ 3  5  7  9]
 [-1 -2 -3 -4]]
```

Convertissez le tableau bidimensionnel en un tableau unidimensionnel et renversez le.

Exemple 2

Considérons que nous avons un tableau de dimension 3×4, c'est-à-dire un tableau contenant 3 rangées et 4 colonnes, convertissons le une simple liste avec la fonction ***reshape(-1)***. Après, renversons le (avec la fonction ***flip(liste, axis)***) de manière à ce que le premier élément devienne le dernier et que le dernier puisse prendre la place du premier et ainsi de suite.

2	4	6	8
3	5	7	9
-1	-2	-3	-4

[[2, 4, 6, 8], [3, 5, 7, 9], [-1, -2, -3, -4]]

```
import numpy as np

matrice2 = np.array([[ 2,  4,  6,  8], [ 3,  5,  7,  9], [-1, -2, -3, -4]])
liste2 = matrice2.reshape(-1)
print(f'La liste obtenue  : {liste2}')
liste3 = np.flip(liste2, 0)
print(f'La liste renversée: {liste3}')
```

```
# Résultat:
La liste obtenue  : [ 2  4  6  8  3  5  7  9 -1 -2 -3 -4]
La liste renversée: [-4 -3 -2 -1  9  7  5  3  8  6  4  2]
```

Suite à la page 183

RU III.3. Некоторые полезные функции для библиотеки NumPy

 # Объединение двух одномерных массивов

```
import numpy as np
liste1 = np.array([0,1,2,3])
liste2 = np.array([4,5,6,7])
liste_unie = np.concatenate((liste1, liste2))
print(f' Объединенный список: {liste_unie}')
```

```
# Результат:
Объединенный список: [0 1 2 3 4 5 6 7]
```

 # Объединение двух двумерных массивов

```
import numpy as np
matrice1 = np.array([[0,1,2], [6,7,8]])
matrice2 = np.array([[3,4,5], [9,10,11]])
liste_unie = np.concatenate((matrice1, matrice2), axis=1)
print(f'Объединенный список №2: {liste_unie}')
```

```
# Результат:
Объединенный список №2: [[ 0  1  2  3  4  5]
 [ 6  7  8  9 10 11]]
```

 # Объединение двух двумерных массивов

```
import numpy as np
matrice1 = np.array([[0,1,2], [6,7,8]])
matrice2 = np.array([[3,4,5], [9,10,11]])
liste_unie = np.concatenate((matrice1, matrice2), axis=0)
print(f'Объединенный список №3: {liste_unie}')
```

```
# Результат:
Объединенный список №3: [[ 0  1  2]
 [ 6  7  8]
 [ 3  4  5]
 [ 9 10 11]]
```

 # Разделение массива на два одномерных массива

```
import numpy as np
tab_a_separer = np.array([1, 2, 3, 4, 5, 6, 7])
ntables = np.array_split(tab_a_separer, 2)
print(f' Первый список: {ntables[0]}')
print(f'Второй список: {ntables[1]}')
```

```
# Результат:
Первый список: [1 2 3 4]
Второй список: [5 6 7]
```

 # Разделение двумерного массива на два двумерных массива

```
import numpy as np
tabi_a_separ = np.array([[1, 2, 3], [4, 5, 6], [-1, -2, -3], [-4, -5, -6]])
ntables = np.array_split(tabi_a_separ, 2)
print(f'Массивы: {ntables}')
print(f'Размер начальной таблицы: {tabi_a_separ.ndim}')
```

```
# Результат:
Массивы: [array([[1, 2, 3],
       [4, 5, 6]]), array([[-1, -2, -3],
       [-4, -5, -6]])]
Размер начальной таблицы: 2
```

 # Найдите индекс элемента, зная его значение

Пример 1 Предположим, что у нас есть список следующих природных элементов: [3, 5, 8, 2, 0, 5, 7]. Найдите индекс(индексы) элемента 5 и распечатайте его(их).

```
import numpy as np
laliste = np.array([3, 5, 8, 2, 5, 9, 7])
mass_i = np.where(laliste==5)
print(f"Все индексы цифры 5: {mass_i[0]}")
for x in range(mass_i[0].size):
    print(f"Индекс №{x+1} для цифры 5 = {mass_i[0][x]}")
```

```
# Результат:
Все индексы цифры 5: [1 4]
Индекс №1 для цифры 5 = 1
Индекс №2 для цифры 5 = 4
```

 # Выполняет поиск в списке и присваивает индекс значению указанное будет вставлено, сохраняя порядок сортировки

Пример №2 Найдите индекс, в котором значение 0 должно быть вставлено в этот список [-7, -3, 1, 3, 6, 9, 12] который предварительно отсортирован (выстроен по порядку).

```
import numpy as np
laliste = np.array([-7, -3, 1,  3, 6, 9, 12])
j = np. searchsorted (laliste, 0)
print(f" Индекс, с которым будет вставлено значение 0 : {j}")
```

```
# Результат:
Индекс, с которым будет вставлено значение 0 : 2
```

Пример №3 это упражнение является продолжением примера №2. После нахождения индекса, где значение 0 должно быть вставлено в список [-7, -3, 1, 3, 6, 9, 12], введите это значение в этот список и распечатайте его.

```
import numpy as np
laliste = np.array([-7, -3, 1,  3, 6, 9, 12])
j = np. searchsorted (laliste, 0)
laliste = np.insert(laliste, j, values=0, axis=0)
print(f" Новый список со значением 0: {laliste}")
```

```
# Результат:
Новый список со значением 0: [-7 -3  0  1  3  6  9 12]
```

 # Размещение элементов списка в упорядоченной последовательности

Пример №4 расположите следующие имена ['Иванов', 'Андреева', 'Петров', 'Антонов', 'Мисонов'] в алфавитном порядке и распечатайте их.

```
import numpy as np
family_stud = np.array(['Иванов', 'Андреева', 'Петров', 'Антонов',
'Мисонов'])
list_family=np.sort(family_stud)
print(f" Имена в алфавитном порядке: {list_family}")
```

```
# Результат:
Имена в алфавитном порядке:
['Андреева' 'Антонов' 'Иванов' 'Мисонов' 'Петров']
```

 # Фильтрация элементов в списке (одномерный массив)

Пример №5

Пусть ***d*** список, содержащий семь натуральных целых чисел [2, 7, 5, 4, 10, 3, 8], создайте второй список, который будет содержать только нечетные цифры. Решить с помощью функции ***delete()***.

Вариант №1

Использование метода ***delete()*** вне цикла.

```
import numpy as np

d = np.array([2, 7, 5, 4, 10, 3, 8])
index = []
i=0
for x in d:
    if (x%2==0):
        index.append(i)
    i+=1
new_d = np.delete(d, index)
print(f'Новый список с нечетными элементами: {new_d}')
```

```
# Результат:
Новый список с нечетными элементами: [7 5 3]
```

Вариант №2

Использование метода ***delete()*** в цикле.

```
import numpy as np

d = np.array([2, 7, 5, 4, 10, 3, 8])
i=0
for x in d:
    if (x%2==0):
        d = np.delete(d, [i])
        i-=1 # Поскольку новый список сокращается на 1 элемент
    i+=1
print(f'Новый список с нечетными элементами: {d}')
```

```
# Результат:
Новый список с нечетными элементами: [7 5 3]
```

Вариант №3

Создание нового списка на основе элементов «**True**» (если условие проверено, данные будут содержаться в массиве или списке

отфильтрованных элементов) и «**False**» (в случае, если элемент исключен из отфильтрованного списка).

```
import numpy as np
d = np.array([2, 7, 5, 4, 10, 3, 8])
new_d_filtre = []
for x in d:
    if (x%2==1):
        new_d_filtre.append(True)
    else:
        new_d_filtre.append(False)
# Здесь у нас есть список, состоящий из логических элементов
# new_d_filtre = [False, True, True, False, False, True, False]

new_d = d[new_d_filtre]
print(f'Новый список с нечетными элементами: {new_d}')
```

```
# Результат:
Новый список с нечетными элементами: [7 5 3]
```

Упражнение №3.1 с решением

В списке содержится фамилии десяти студентов с их соответствующими оценками (таблица ниже). Задача состоит в том, чтобы автоматически составлять два разных списка, состоящих из студентов, прошедших аттестацию по всем предметам, и тех, кто должен пересдать предмет (по тем дисциплинам, которые не сданы). Мы считаем, что студент должен повторно сдавать предмет, когда оценка строго меньше 3.

Фамилия	Оценки
Ivanov	3
Andreeva	4
Petrov	5
Antonov	2
Misonov	0
Sidorov	3
Davidov	5
Shelkova	4
Timkov	2
Belova	3

Используя библиотеку NumPy, можно будет отфильтровать массив, используя список нулевых индексов (т. е. содержащий значения **True** и/или **False**).

Решение находится в главе VI

Продолжение на странице 189

EN III.3. Some useful features for the NumPy library

Combining two one-dimensional arrays

```
import numpy as np
liste1 = np.array([0,1,2,3])
liste2 = np.array([4,5,6,7])
liste_unie = np.concatenate((liste1, liste2))
print(f' Объединенный список: {liste_unie}')
```

```
# The result:
The combined list: [0 1 2 3 4 5 6 7]
```

Combining two two-dimensional arrays

```
import numpy as np
matrice1 = np.array([[0,1,2], [6,7,8]])
matrice2 = np.array([[3,4,5], [9,10,11]])
liste_unie = np.concatenate((matrice1, matrice2), axis=1)
print(f'The combined list No. 2: {liste_unie}')
```

```
# The result:
The combined list No. 2: [[ 0  1  2  3  4  5]
 [ 6  7  8  9 10 11]]
```

Combining two two-dimensional arrays

```
import numpy as np
matrice1 = np.array([[0,1,2], [6,7,8]])
matrice2 = np.array([[3,4,5], [9,10,11]])
liste_unie = np.concatenate((matrice1, matrice2), axis=0)
print(f'The combined list No. 3: {liste_unie}')
```

```
# The result:
The combined list No. 3: [[ 0  1  2]
 [ 6  7  8]
 [ 3  4  5]
 [ 9 10 11]]
```

Splitting an array into two one-dimensional arrays

```
import numpy as np
tab_a_separer = np.array([1, 2, 3, 4, 5, 6, 7])
ntables = np.array_split(tab_a_separer, 2)
print(f'The first list: {ntables[0]}')
print(f'The second list: {ntables[1]}')
```

```
# The result:
The first list: [1 2 3 4]
The second list: [5 6 7]
```

Splitting a two-dimensional array into two two-dimensional arrays

```
import numpy as np
tabi_a_separ = np.array([[1, 2, 3], [4, 5, 6], [-1, -2, -3], [-4, -5, -6]])
ntables = np.array_split(tabi_a_separ, 2)
print(f'Arrays: {ntables}')
print(f'Size of the initial table: {tabi_a_separ.ndim}')
```

```
# The result:
Arrays: [array([[1, 2, 3],
       [4, 5, 6]]), array([[-1, -2, -3],
       [-4, -5, -6]])]
Size of the initial table: 2
```

Find the index of an element by knowing its value.

Example 1 Suppose we have a list of the following natural elements: [3, 5, 8, 2, 0, 5, 7]. Find the index(s) of element 5 and print it(s).

```
import numpy as np
laliste = np.array([3, 5, 8, 2, 5, 9, 7])
mass_i = np.where(laliste==5)
print(f"The indexes of the number 5 are  {mass_i[0]}")
for x in range(mass_i[0].size):
   print(f"The index №{x+1} of the number 5 is: {mass_i[0][x]}")
```

```
# The result:
The indexes of the number 5 are: [1 4]
The index №1 of the number 5 is: 1
The index №2 of the number 5 is: 4
```

Searches the list and assigns an index to the value
the specified will be inserted, keeping the sort order

Example 2 Find the index where the value 0 should be inserted into this list [-7, -3, 1, 3, 6, 9, 12] which is pre-sorted (arranged in order).

```
import numpy as np
laliste = np.array([-7, -3, 1,  3, 6, 9, 12])
j = np. searchsorted (laliste, 0)
print(f"Index with which the value 0 will be inserted: {j}")
```

```
# The result:
Index with which the value 0 will be inserted: 2
```

Example 3 This exercise is a continuation of example №2. After finding the index where the value 0 should be inserted into the list [-7, -3, 1, 3, 6, 9, 12], enter this value in this list and print it out.

```
import numpy as np
laliste = np.array([-7, -3, 1,  3, 6, 9, 12])
j = np.searchsorted(laliste, 0)
laliste = np.insert(laliste, j, values=0, axis=0)
print(f"New list with value 0: {laliste}")
```

```
# The result:
New list with value 0: [-7 -3  0  1  3  6  9 12]
```

Placement of list elements in an ordered sequence

Example 4 Arrange the following names ['Ivanov',' Andreeva',' Petrov',' Antonov',' Misonov'] in alphabetical order and print them out.

```
import numpy as np
family_stud = np.array(['Ivanov', 'Andreeva', 'Petrov', 'Antonov',
'Misonov'])
list_family=np.sort(family_stud)
print(f"Names in alphabetical order: {list_family}")
```

```
# The result:
Names in alphabetical order:
['Andreeva' 'Antonov' ' Misonov' ' Petrov' 'Ivanov']
```

Filtering elements in a list (one-dimensional array)

Example 5

Let ***d*** be a list containing seven natural integers [2, 7, 5, 4, 10, 3, 8], create a second list that will only contain odd numbers. Solve it using the ***delete()*** function.

Option 1

Using the ***delete()*** method outside of the loop.

```
import numpy as np

d = np.array([2, 7, 5, 4, 10, 3, 8])
index = []
i=0
for x in d:
    if (x%2==0):
        index.append(i)
    i+=1
new_d = np.delete(d, index)
print(f'New list with odd elements: {new_d}')
```

```
# The result:
New list with odd elements: [7 5 3]
```

Option 2

Using the ***delete()*** method in a loop.

```
import numpy as np

d = np.array([2, 7, 5, 4, 10, 3, 8])
i=0
for x in d:
    if (x%2==0):
        d = np.delete(d, [i])
        i-=1 # Since the new list is reduced by 1 element
    i+=1
print(f'New list with odd elements: {d}')
```

```
# The result:
New list with odd elements: [7 5 3]
```

Option 3

Creating a new list based on the elements «**True**» (if the condition is checked, the data will be contained in an array or list of filtered elements) and «**False**» (if the element is excluded from the filtered list).

```
import numpy as np
d = np.array([2, 7, 5, 4, 10, 3, 8])
new_d_filtre = []
for x in d:
    if (x%2==1):
        new_d_filtre.append(True)
    else:
        new_d_filtre.append(False)
# Here we have a list consisting of logical elements
# new_d_filtre = [False, True, True, False, False, True, False]

new_d = d[new_d_filtre]
print(f'New list with odd elements: {new_d}')
```

```
# The result:
New list with odd elements: [7 5 3]
```

Exercise 3.1 with a solution

The list contains the names of ten students with their respective grades (table below). The task is to automatically make two different lists, consisting of students who have passed the exams in all disciplines and those who must re-take the exam (for those disciplines that they haven't passed). We believe that a student should re-take the exam when the grade is strictly less than 3.

Beginning

Surname	Grade
Ivanov	3
Andreeva	4
Petrov	5
Antonov	2
Misonov	0

Continuation

Surname	Grade
Sidorov	3
Davidov	5
Shelkova	4
Timkov	2
Belova	3

Using the NumPy library, it will be possible to filter the array using a list of null indexes (i.e. containing the values **True** and/or **False**).

The resolution can be found in Chapter VI

Continued on page 193

FR III.3. Quelques fonctions utiles pour la bibliothèque NumPy

#Joindre deux tableaux unidimensionne1s

```
import numpy as np
liste1 = np.array([0,1,2,3])
liste2 = np.array([4,5,6,7])
liste_unie = np.concatenate((liste1, liste2))
print(f'La liste unie: {liste_unie}')
```

```
# Résultat:
La liste unie: [0 1 2 3 4 5 6 7]
```

#Joindre deux tableaux bidimensionne1s

```
import numpy as np
matrice1 = np.array([[0,1,2], [6,7,8]])
matrice2 = np.array([[3,4,5], [9,10,11]])
liste_unie = np.concatenate((matrice1, matrice2), axis=1)
print(f'La liste unie №2: {liste_unie}')
```

```
# Résultat:
La liste unie №2: [[ 0  1  2  3  4  5]
 [ 6  7  8  9 10 11]]
```

#Joindre deux tableaux bidimensionne1s

```
import numpy as np
matrice1 = np.array([[0,1,2], [6,7,8]])
matrice2 = np.array([[3,4,5], [9,10,11]])
liste_unie = np.concatenate((matrice1, matrice2), axis=0)
print(f'La liste unie №3: {liste_unie}')
```

```
# Résultat:
La liste unie №3: [[ 0  1  2]
 [ 6  7  8]
 [ 3  4  5]
 [ 9 10 11]]
```

Division d'un tableau en deux tableaux unidimensionnels

```
import numpy as np
tab_a_separer = np.array([1, 2, 3, 4, 5, 6, 7])
ntables = np.array_split(tab_a_separer, 2)
print(f'La première liste: {ntables[0]}')
print(f'La deuxième liste: {ntables[1]}')
```

```
# Résultat:
La première liste: [1 2 3 4]
La deuxième liste: [5 6 7]
```

Division d'un tableau bidimensionnel en deux tableaux bidimensionnels

```
import numpy as np
tabi_a_separ = np.array([[1, 2, 3], [4, 5, 6], [-1, -2, -3], [-4, -5, -6]])
ntables = np.array_split(tabi_a_separ, 2)
print(f'Les tableaux: {ntables}')
print(f'La dimension du tableau de départ est: {tabi_a_separ.ndim}')
```

```
# Résultat:
Les tableaux: [array([[1, 2, 3],
       [4, 5, 6]]), array([[-1, -2, -3],
       [-4, -5, -6]])]
La dimension du tableau de départ est: 2
```

Recherchez l'index d'un élément connaissant sa valeur

Exemple 1 Considérons que nous avons une liste d'éléments naturels suivants: [3, 5, 8, 2, 0, 5, 7]. Retrouvez l'index(les index) du chiffre 5 et imprimez le(les).

```
import numpy as np
laliste = np.array([3, 5, 8, 2, 5, 9, 7])
mass_i = np.where(laliste==5)
print(f"Les index du chiffre 5 sont : {mass_i[0]}")
for x in range(mass_i[0].size):
    print(f"L'index №{x+1} du chiffre 5 est = {mass_i[0][x]}")
```

```
# Résultat:
Les index du chiffre 5 sont : [1 4]
L'index №1 du chiffre 5 est = 1
L'index №2 du chiffre 5 est = 4
```

Effectue une recherche dans une liste et renvoie l'index où la valeur spécifiée serait insérée en maintenant l'ordre de tri

Exemple 2 Trouvez l'index où la valeur 0 doit être insérée dans cette liste [-7, -3, 1, 3, 6, 9, 12] qui est au préalable triée (arrangée suivant un ordre).

```
import numpy as np
laliste = np.array([-7, -3, 1,  3, 6, 9, 12])
j = np. searchsorted (laliste, 0)
print(f"L'index où la valeur 0 serait insérée est : {j}")
```

```
# Résultat:
L'index où la valeur 0 serait insérée est : 2
```

Exemple 3 Cet exercice est une continuation de l'exemple 2. Après avoir trouvé l'index où la valeur 0 doit être insérée dans la liste [-7, -3, 1, 3, 6, 9, 12]; introduisez cette valeur dans cette liste et imprimez là.

```
import numpy as np
laliste = np.array([-7, -3, 1,  3, 6, 9, 12])
j = np. searchsorted (laliste, 0)
laliste = np.insert(laliste, j, values=0, axis=0)
print(f"La nouvelle liste avec la valeur 0 est: {laliste}")
```

```
# Résultat:
La nouvelle liste avec la valeur 0 est: [-7 -3  0  1  3  6  9 12]
```

Mettre les éléments d'une liste dans une séquence ordonnée

Exemple 4 Classez les noms suivants ['Ivanov', 'Andreeva', 'Petrov', 'Antonov', 'Misonov'] par ordre alphabétique et imprimez les.

```
import numpy as np
family_stud = np.array(['Ivanov', 'Andreeva', 'Petrov', 'Antonov',
'Misonov'])
list_family=np.sort(family_stud)
print(f"Les noms classés par ordre alphabétique: {list_family}")
```

```
# Résultat:
Les noms classés par ordre alphabétique:
['Andreeva' 'Antonov' 'Ivanov' 'Misonov' 'Petrov']
```

Filtrage des éléments dans une liste (tableau unidimensionnel)

Exemple 5

Soit ***d*** la liste contenant sept entiers naturels [2, 7, 5, 4, 10, 3, 8], créer une seconde liste qui va contenir uniquement les chiffres impairs. Résoudre en utilisant la fonction ***delete()***.

Variante 1

Utilisation de la méthode ***delete()***en dehors de la boucle.

```
import numpy as np

d = np.array([2, 7, 5, 4, 10, 3, 8])
index = []
i=0
for x in d:
    if (x%2==0):
        index.append(i)
    i+=1
new_d = np.delete(d, index)
print(f'La nouvelle liste avec les éléments impairs: {new_d}')
```

```
# Résultat:
La nouvelle liste avec les éléments impairs: [7 5 3]
```

Variante 2

Utilisation de la méthode ***delete()*** dans la boucle.

```
import numpy as np

d = np.array([2, 7, 5, 4, 10, 3, 8])
i=0
for x in d:
    if (x%2==0):
        d = np.delete(d, [i])
        i – =1 # Car la nouvelle liste est réduite d'un élément
    i+=1
print(f'La nouvelle liste avec les éléments impairs: {d}')
```

```
# Résultat:
La nouvelle liste avec les éléments impairs: [7 5 3]
```

Variante 3

Création d'une nouvelle liste à base des éléments «**True**» (si la condition est vérifiée, ainsi sera contenu dans le tableau ou la liste des éléments filtrés) et «**False**» (dans le cas où l'élément est exclu de la liste filtrée).

```
import numpy as np
d = np.array([2, 7, 5, 4, 10, 3, 8])
new_d_filtre = []
for x in d:
    if (x%2==1):
        new_d_filtre.append(True)
    else:
        new_d_filtre.append(False)
# Ici, nous avons une liste composée des éléments Booléens
# new_d_filtre = [False, True, True, False, False, True, False]

new_d = d[new_d_filtre]
print(f'La nouvelle liste avec les éléments impairs: {new_d}')
```

```
# Résultat:
La nouvelle liste avec les éléments impairs: [7 5 3]
```

Exercice 3.1 corrigé

Dans une liste contenant une dizaine d'étudiants avec leurs notes respectives (voir tableau ci-dessous). Le devoir consiste à établir automatiquement deux listes différentes constituées des étudiants ayant validés toutes les matières et ceux qui doivent aller au rattrapages (pour les

matières non validées). Nous considérons qu'un étudiant doit aller au rattrapage d'une matière, lorsque la note est strictement inférieure à 3.

Noms	Notes
Ivanov	3
Andreeva	4
Petrov	5
Antonov	2
Misonov	0
Sidorov	3
Davidov	5
Shelkova	4
Timkov	2
Belova	3

En utilisant la bibliothèque NumPy, il sera possible de filtrer un tableau à l'aide d'une liste d'index **Booléen** (c'est-à-dire contenant les valeurs **True** et/ou **False**).

Voir la résolution au chapitre VI

Suite à la page 196

RU III.4. Случайные числа

Функция ***random*** позволяет генерировать случайные целые числа.

Различные методы на функции (см. таб. 3.1)

Перед любой операцией необходимо импортировать библиотеку *random* таким образом: ***from numpy import random***

Таблица 3.1 – Некоторые методы и описания случайных чисел

Методы	**Описания**
randint(N)	Генерирует случайным образом, целое число от 0 до N, например: *x= random.randint(20).* *x* будет принимать любое значение между 0 и 20
randint(N, size=(k))	Случайным образом генерирует целое число от 0 до N, **k** количество элементов в одномерном массиве. Например: *randint(20, size=(3))*=[13 5 7]
randint(N, size=(k,m))	Случайным образом генерирует целое число от 0 до N, **k** – количество строк и **m** – количество столбцов, в двумерном массиве Например: *randint(20, size=(2, 3))*= [[16 14 12] [8 10 9]]
rand()	Возвращает действительное число от 0 до 1 *x=random.rand() ;* *например x=random.rand()=0,67*
	Если мы добавим натуральное целое число в качестве параметра, оно определит количество вещественных чисел, которое будет сгенерировано в одномерном массиве. Например *x=random.rand(3)*=[0,67 0,23 0,79]

Продолжение таблицы 3.1

Методы	Описания
rand()	Аналогичным образом, с двумя параметрами он будет генерировать двумерный массив. Первый параметр будет обозначать количество строк, а второй – количество столбцов. Например *x=random.rand(2, 3)*= [[0,61352542 0,72491567 0,48139524] [0,9237479 0,47582258 0,88221277]]
choice()	Позволяет генерировать случайное значение на основе массива указанных значений Например: *x = random.choice([1, -4, 2, 0])* = -4
	x = random.choice([1, -4, 2, 0], size=(2, 3)) = [[1 1 -4] [2 2 0]]
choice()	Введение возникновения вероятности; сумма вероятностей должна быть равна 1. Для одномерного массива: *x = random.choice([3, 5], p=[0,1, 0,9], size=(10))* = [3 3 5 5 3 5 5 3 5 5] или же [5 5 5 5 5 5 5 5 5 5] ...

Окончание таблицы 3.1

Методы	Описания
choice()	Для двумерного массива: *x = random.choice([4, 2, 5], p=[0,2, 0,5, 0,3], size=(2, 3))=* [[2 4 2] [2 5 2]] | [[2 5 5] [5 5 5]] | …
shuffle() и permutation()	Позволяет случайным образом изменять положение элементов в массиве. Например: *import numpy as np* *x = np.array([5, 4, 3, 2, 1, 0])* *random.shuffle(x) ;* В результате мы получим один из следующих вариантов при печати *x* на экране: [1 4 5 2 3 0]
	С помощью этой функции: *import numpy as np* *x = np.array([5, 4, 3, 2, 1, 0])* *print(random.permutation(x))* исходный список *x* остается неизменным

 # Генерировать целые числа

arange(a, b) – эта функция генерирует одномерный массив с целыми числами от a до b.

Пример

```
import numpy as np

mas = np.arange(2, 10)
print(f'Сгенерированные элементы:{mas}')
```

```
# Результат:
Сгенерированные элементы:[2 3 4 5 6 7 8 9]
```

Продолжение на странице 200

EN III.4. Random numbers

The ***random*** function allows you to generate random integers.

Different methods on the function (see table 3.1).

Before any operation, you must import the random library in this way: ***from numpy import random***

Table 3.1 – Some methods and descriptions about random numbers

Methods	Description
randint(N)	Generates a random integer from **0** to ***N***, for example: *x= random.randint(20).* *x* will take any value between *0* and *20*
randint(N, size=(k))	Randomly generates an integer from ***0*** to ***N***, ***k*** the number of elements in a one-dimensional array. For example: *randint(20, size=(3))*=[13 5 7]
randint(N, size=(k,m))	Randomly generates an integer from 0 to N, ***k*** is the number of rows and ***m*** is the number of columns, in a two-dimensional array For example: *randint(20, size=(2, 3))*= [[16 14 12] [8 10 9]]

Continuation of table 3.1

rand()	Returns a valid number from ***0*** to ***1*** *x=random.rand();* *for example x=random.rand()=0,67*
	If we add a natural integer as a parameter, it will determine the number of real numbers that will be generated in the one-dimensional array. For example: *x=random.rand(3)*=[0,67 0,23 0,79]
	Similarly, with two parameters, it will generate a two-dimensional array. The first parameter will indicate the number of rows, and the second parameter will indicate the number of columns. For example: *x=random.rand(2, 3)=* [[0,61352542 0,72491567 0,48139524] [0,9237479 0,47582258 0,88221277]]
choice()	Allows you to generate a random value based on an array of specified values. For example: *x = random.choice([1, -4, 2, 0])* = -4
	x = random.choice([1, -4, 2, 0], size=(2, 3)) = [[1 1 -4] [2 2 0]]

End of table 3.1

<table>
<tr><td>choice()</td><td>Introduction of probability occurrence; the sum of probabilities must be equal to 1.
For a one-dimensional array:
x = random.choice([3, 5], p=[0,1, 0,9], size=(10)) =
[3 3 5 5 3 5 5 3 5 5]
or
[5 5 5 5 5 5 5 5 5 5]
...</td></tr>
<tr><td>choice()</td><td>For a two-dimensional array:
x = random.choice([4, 2, 5], p=[0,2, 0,5, 0,3], size=(2, 3))=
<table><tr><td>[[2 4 2]
[2 5 2]]</td><td>[[2 5 5]
[5 5 5]]</td><td>…</td></tr></table></td></tr>
<tr><td rowspan="2">shuffle()
и
permutation()</td><td>Allows you to randomly change the position of elements in the array.
For example:
import numpy as np
x = np.array([5, 4, 3, 2, 1, 0])
random.shuffle(x) ;
As a result, we will get one of the following options when printing x on the screen:
[1 4 5 2 3 0]</td></tr>
<tr><td>Using this function:
import numpy as np
x = np.array([5, 4, 3, 2, 1, 0])
print(random.permutation(x))
the original list x remains unchanged</td></tr>
</table>

Generate integers

arange(a, b) – this function generates a one-dimensional array with integers from ***a*** to ***b***.

Example

```
import numpy as np

mas = np.arange(2, 10)
print(f'The generated elements: {mas}')
```

```
# The result:
The generated elements: [2 3 4 5 6 7 8 9]
```

Continued on pages 200 and 201

FR III.4. Nombres aléatoires

La fonction ***random*** permet de générer ces entiers aléatoires.

Les differentes méthodes sur la fonction (voir la table 3.1)

Il faudra importer la bibliothèque random de cette manière avant toute opération: ***from numpy import random***

Table 3.1 – Quelques méthodes et descriptions sur les nombres aléatoires

Les méthodes	Description
randint(N)	Génère de manière aléatoire un nombre entier de 0 à **N** : Par exemple : *x= random.randint(20).* x prendra un nombre compris entre 0 et 20
randint(N, size=(k))	Génère de manière aléatoire un nombre entier de 0 à **N**, **k** nombre fois dans un tableau unidimensionnel Par exemple : *randint(20, size=(3))*=[13 5 7]

Suite de la table 3.1

randint(N, size=(k,m))	Génère de manière aléatoire un nombre entier de 0 à **N**, **k** – nombre de rangée et **m** – nombre de colonne, dans un tableau bidimensionnel Par exemple : *randint(20, size=(2, 3))*= [[16 14 12] [8 10 9]]
rand()	Renvoie un nombre réel compris entre 0 et 1 *x=random.rand() ;* *par exemple x=random.rand()=0.67*
	Si nous ajoutons un entier naturel comme paramètre, il déterminera le nombre de réel qu'il doit générer dans un tableau unidimensionnel. Par exemple *x=random.rand(3)*=[0.67 0.23 0.79]
	De la même manière, avec deux paramètres, il générera un tableau bidimensionnel. Le premier paramètre désignera le nombre de rangée, et le deuxième – le nombre de colonne Par exemple *x=random.rand(2, 3)*= [[0.61352542 0.72491567 0.48139524] [0.9237479 0.47582258 0.88221277]]
choice()	Elle permet de générer une valeur aléatoire basée sur un tableau de valeurs indiquées Par xemple : *x = random.choice([1, -4, 2, 0])* = -4
	x = random.choice([1, -4, 2, 0], size=(2, 3)) = [[1 1 -4] [2 2 0]]

Fin de la table 3.1

<table>
<tr><td>choice()</td><td>Introduction de la probabilité d'apparition; la somme des probabilités doit être égale à 1. Pour un tableau unidimensionnel:
x = random.choice([3, 5], p=[0.1, 0.9], size=(10)) =

[3 3 5 5 3 5 5 3 5 5]
ou alors
[5 5 5 5 5 5 5 5 5 5]
...</td></tr>
<tr><td>choice()</td><td>Pour un tableau bidimensionnel :
x = random.choice([4, 2, 5], p=[0.2, 0.5, 0.3], size=(2, 3))=
<table><tr><td>[[2 4 2]
[2 5 2]]</td><td>[[2 5 5]
[5 5 5]]</td><td>…</td></tr></table></td></tr>
<tr><td rowspan="2">shuffle()
et
permutation()</td><td>Permet de changer aléatoirement la position des éléments sur un tableau
Par exemple :

import numpy as np
x = np.array([5, 4, 3, 2, 1, 0])
random.shuffle(x) ;

Nous aurons comme résultat l'une des variantes suivantes en imprimant x:
[1 4 5 2 3 0]</td></tr>
<tr><td>Avec cette fonction:

import numpy as np
x = np.array([5, 4, 3, 2, 1, 0])
print(random.permutation(x))

la liste x d'origine reste inchangée.</td></tr>
</table>

 # Générer des nombres entiers

arange(a, b) – cette fonction génère un tableau unidimensionnel avec des nombres entiers allant de ***a*** à *b – 1*.

Exemple

```
import numpy as np

mas = np.arange(2, 10)
print(f'Les éléments générés sont:{mas}')
```

```
# Résultat:
Les éléments générés sont:[2 3 4 5 6 7 8 9]
```

Suite aux pages 200 et 203

RU Глава IV. КАК ОБРАБАТЫВАТЬ ОШИБКИ ВЫПОЛНЕНИЯ ОПЕРАЦИЙ
EN Chapter IV. HOW TO HANDLE OPERATION EXECUTION ERRORS
FR Chapitre IV. COMMENT GÉRER LES ERREURS D'EXÉCUTION D'OPÉRATIONS

RU Введение

В программировании всегда есть высокая вероятность того, что тот, кто пишет программу на определенном языке, может сделать ошибки кода или не учесть некоторые детали при запуске программы. Например, вы просите пользователя ввести действительное число, и в коде забываете проверить введенное значение (значение должно автоматически иметь символ "**.**", а не "**,**"), или после заданного пути вы обнаружите, что делите число на 0. Введение запятой вместо точки или затем деление на 0 приведет к набору закодированных сообщений, которые не совсем приятно видеть. Будет похоже, что программа не работает. Возвращаемое сообщение будет на английском языке и, вероятно, пользователю трудно будет исправить ситуацию. Отметим также, что эти ошибки относятся к стандартным классам, которые нужно знать программисту. Помимо этих классов, необходимо будет ввести инструкции типа "*try*", "*except*", "*else*", "*finally*" и "*raise*", чтобы программа могла нормально завершиться. Как мы уже упоминали, язык Python имеет стандартные методы для обработки ошибок и исключений. Ошибки представлены набором ключевых слов для их обнаружения и обработки. Например:

- *ValueError*: путаница с типами, введенное значение не соответствует указанным параметрам;
- *ZeroDivisionError*: деление на ноль (0);
- *TypeError*: значение в программном коде не соответствует типу; операция с несовместимыми типами и т. д.

Пример №1

Написать программу, которая реализует функции простого калькулятора на основе четырех операций из двух чисел: сложение, вычитание, умножение и деление.

Решение без инструкций try, except, else, finally или raise

```
x = float(input('Введите первое действительное число x = '))
y = float(input('Введите второе действительное число y = '))
print("1-сложение; 2-деление; 3-умножение; 4-вычитание")
sign = str(input("Выберите тип операции :"))
if sign == "1":
   mess="Результат операции "+str(x)+"+"+str(y)+" = "+ str(x+y)
elif (sign == "2"):
   mess="Результат операции "+str(x)+"/"+str(y)+" = "+ str(x/y)
elif sign == "3":
   mess=" Результат операции "+str(x)+"*"+str(y)+" = "+ str(x*y)
elif sign == "4":
   mess="Результат операции "+str(x)+"-"+str(y)+" = "+ str(x-y)
else:
   mess="Вы ввели символ " + sign+", вам нужно было выбрать
   только число от 1 до 4."
print(f"{mess}")
```

```
# Результат: (одна из возможностей)
Введите первое действительное число x = 15,4
Введите второе действительное число y = 7
1-сложение; 2-деление; 3-умножение; 4-вычитание
Выберите тип операции : 4
Результат операции 15,4-7,0 = 8,4
```

Продолжение на странице 205

EN Introduction

In programming, there is always a high probability that someone who writes a program in a particular language may make code errors or fail to

take some details into account when running the program. For example, you ask the user to enter a valid number, and in the code you forget to check the entered value (the value should automatically have the character "**.**", not "**,**") or after a given path you find yourself dividing the number by 0.

The introduction of the "comma" instead of the "point" or the division by Zero, will result in a series of coded messages, which will not be understandable to a simple user; we will be led to believe that we have an incorrect program code in the set. The message returned by the program code will be in English language and probably difficult for the user to correct (remedy) the situation.

Note also that these errors belong to standard classes that you will need to know as a programmer (developer). To remedy these likely incidents, the instructions "*try*", "*except*", "*else*", "*finally*" and "*raise*" will have to be introduced so that the program can normally complete.

As we have already mentioned, Python has standard methods for handling errors and exceptions. Errors are represented by a set of keywords for detecting and processing them. ***For example:***

- ***ValueError***: confusion with types, the entered value does not match the specified parameters;
- ***ZeroDivisionError***: division by zero (0);
- ***TypeError***: the value in the program code does not match the type; operation with incompatible types, etc.

Example 1

Write a program that performs the following functions of a simple calculator based on four operations of two numbers: addition, subtraction, multiplication and division.

Solution without «try», «except», «else», «finally» or «raise» statements

```
x = float(input('Enter the first valid number x = '))
y = float(input('Enter the second valid number y = '))
print("1 – addition; 2 – division; 3– multiplication; 4– subtraction")
sign = str(input("Select the type of operation:"))
if sign == "1":
   mess="The result of the operation "+str(x)+"+"+str(y)+" = "+ str(x+y)
elif (sign == "2"):
   mess="The result of the operation "+str(x)+"/"+str(y)+" = "+ str(x/y)
elif sign == "3":
   mess="The result of the operation "+str(x)+"*"+str(y)+" = "+ str(x*y)
elif sign == "4":
   mess=" The result of the operation "+str(x)+"-"+str(y)+" = "+ str(x-y)
else:
   mess="You entered the character " + sign+", you only had to select a
   number from 1 to 4."
print(f"{mess}")
```

```
# Result: (one of the possibilities)
Enter the first valid number x = 15,4
Enter the second valid number y = 7
1 – addition; 2 – division; 3– multiplication; 4– subtraction
Select the type of operation : 4
The result of the operation 15,4-7,0 = 8,4
```

Continued on page 206

FR Introduction

En programmation, il y a toujours de forte chance que celui qui rédige le programme dans un langage précis puisse effectuer des erreurs de codes ou alors oublier de tenir compte de certains détails lors de l'exécution du programme. Par exemple vous demandez à l'utilisateur d'introduire un nombre réel et dans le code vous oubliez de faire une

vérification de la valeur introduite (la valeur doit automatiquement avoir le symbole “.” et non “,”), ou alors après un cheminement bien donné, on se retrouve entrain de diviser un nombre par 0. *L'introduction de la virgule à la place du point ou alors la division par zéro, entraînera une suite de messages codés, qui ne sera pas compréhensible à un simple utilisateur, nous aurons l’impression d’avoir un code programme incorrecte dans l'ensemble. Le message renvoyé par le code programme sera en langue anglaise et probablement difficile à l'utilisateur de remédier à la situation.* Notons également que ces erreurs appartiennent à des classes standards qu’il faudra connaître en tant que programmiste (développeur). Pour remédier à ces probables incidents, il faudra introduire les instructions “*try*”, “*except*”, “*else*”, “*finally*” et “*raise*” pour que le programme puisse “*normalement*” s'achever. Comme nous l’avions mentionné, le langage Python possède des classes d'objets représentant les erreurs qu'on peut obtenir en effectuant des opérations. Ces erreurs sont représentées par un ensemble de mots clés qui permettent de les détecter et de les traiter:

- *ValueError*: confusion entre les types, la valeur introduite ne correspond pas aux paramètres indiqués;
- *ZeroDivisionError*: division par zéro (0);
- *TypeError*: la valeur dans le code du programme ne correspond pas au type; opération avec des types incompatibles entre eux etc.

Exemple

Écrire un programme qui réalise les fonctions d'une simple calculatrice basée sur quatre opérations de deux nombres: l'addition, la soustraction, la multiplication et la division.

Résolution sans les instructions «try», «except», «else», «finally» ou «raise»

```
x = float(input('Introduire un premier nombre réel x = '))
y = float(input('Introduire un second  nombre réel y = '))
print("1-addition; 2-division; 3-multiplication; 4-soustraction")
sign = str(input("Choisissez le type d'opération à faire : "))
if sign == "1":
   mess="Le résultat de l'opération "+str(x)+"+"+str(y)+" = "+ str(x+y)
elif (sign == "2"):
   mess="Le résultat de l'opération "+str(x)+"/"+str(y)+" = "+ str(x/y)
elif sign == "3":
   mess="Le résultat de l'opération "+str(x)+"*"+str(y)+" = "+ str(x*y)
elif sign == "4":
   mess="Le résultat de l'opération "+str(x)+"-"+str(y)+" = "+ str(x-y)
else:
   mess="Vous aviez introduit le symbole "+sign+", vous deviez choisir
   uniquement un nombre compris entre 1 et 4."
print(f"{mess}")
```

```
# Résultat: (une des possibilités)
Introduire un premier nombre réel x = 15.4
Introduire un second  nombre réel y = 7
1-addition; 2-division; 3-multiplication; 4-soustraction
Choisissez le type d'opération à faire : 4
Le résultat de l'opération 15.4-7.0 = 8.4
```

Suite à la page 207

RU IV.1. Типы ошибок

Важно знать разницу между синтаксической ошибкой и исключением.

Синтаксическая ошибка – это неточное описание конструкции после анализа синтаксического кода.

Пример

Если мы пишем:

Print("message")

чтобы вывести сообщение на экран, мы получим синтаксическую ошибку:

«NameError: name '**P**rint' is not defined»

то есть ошибка имени функции "Print", которая не определена. Чтобы исправить, нужно будет написать функцию таким образом:

print("message")

Исключением можно назвать ошибку, обнаруженную во время выполнения кода. Они генерируют сообщения об ошибках, которые можно покажем ниже. Они не всегда фатальны во время выполнения кодов.

Пример

Найти значение z = x/y. Если x = 13 и y = 0, у нас будет несколько сообщений, среди которых мы выделим:

«ZeroDivisionError: division by zero»,

после выполнения следующего кода:

```
x = 13
y = 0
z = x/y
```

То есть запрещено делить ненулевое число на ноль. Чтобы исправить это, необходимо написать программу, которая будет поддерживать это исключение, чтобы увеличить отказоустойчивость кода. Таким образом, выполнение программы будет защищено от потенциального сбоя, который может привести к ее преждевременному завершению.

Продолжение на странице 208

EN IV.1. Types of errors

It is important to know the difference between a syntax error and an exception.

A **syntax error** is an inaccurate writing of a statement coming after parsing a code.

Example If we write:

Print("message")

to display the message on the screen, we will get the following syntax error:

*«NameError: name '**P**rint' is not defined»*

i.e., the error of the function name “Print”, which is not defined. To fix it, you will need to write the function this way:

print("message")

An exception is an error detected during code execution. They generate error messages, which can be shown below. They are not always fatal during code execution.

Example: Find the value of z = x/y. If x = 13 and y = 0, we will have several messages, among which we will highlight:

«ZeroDivisionError: division by zero»,

after executing the following code:

```
x = 13
y = 0
z = x/y
```

That is, it is forbidden to divide a non-zero number by zero. To fix this, you need to write a program that will support this exception to increase the fault tolerance of the code. Thus, the execution of the program will be protected from potential failure, which can lead to its premature termination.

Continued on page 213

FR IV.1. Les types d’erreurs

Il est important de savoir la difference entre une erreur de syntaxe et une exception d’erreur.

L'erreur de syntaxe est une écriture inexacte d'une instruction venant après l'analyse d'un code.

Exemple

Si nous écrivons:

Print("message")

pour faire sortir le message sur l'écran, alors nous obtiendrons une erreur de syntaxe suivante: «*NameError: name '**P**rint' is not defined*»,c'est-à-dire une erreur du nom de la fonction *«Print»* qui n'est pas définie. Pour remédier à cela, il faudra bien écrire la fonction de cette manière:

print("message")

Une exception d'erreur est une erreur détectée pendant l'exécution du code. Ces erreurs génèrent des messages d'erreurs qu'on peut voir un peu plus bas. Elles ne sont pas toujours fatales pendant l'exécution des codes. Exemple : Trouver la valeur z = x/y. Si x = 13 et y = 0, on aura plusieurs messages parmi lesquels, nous distinguerons:

«*ZeroDivisionError: division by zero*»,

après l'exécution du code suivant:

x = 13

y = 0

z = x/y

C'est-à-dire qu'il est interdit de diviser un nombre non nul par zéro. Pour remédier à cela, il faudra écrire un programme qui prendra en charge cette exception afin d'augmenter la tolérance de la panne du code. Ainsi l'exécution du programme sera protégée contre la défaillance potentielle pouvant entraîner son arrêt prématuré.

Suite à la page 218

RU IV.2. Инструкции "*try*" и "*except*"

Эти две команды работают вместе. Команда «**try**» используется для проверки кода, который может иметь ошибки выполнения и дать через команду «**except**» указания, которые следует использовать, если исключение действительно встречается. Давайте проследим на примере, чтобы лучше понять:

Решение с инструкциями «try» и «except»

```
while True:
    try:
        print("4 операции в калькуляторе с 2 числами: +,/,*,–")
        x = float(input('Введите первое число x = '))
        y = float(input('Введите второе число y = '))
        answer = ' Осторожно!!! Нужно выбрать от 1 до 4 \n '
        while True:
            print("1-сложение; 2-деление; 3-умноже*; 4-вычитание")
            operation = input("Выберите тип операции:")
            sign = operation.strip()
            if sign == "1":
                answer = f"{x}+{y} = {x+y}"
            elif sign == "2":
                answer = f"{x}/{y} = {x/y}"
            elif sign == "3":
                answer = f"{x}*{y} = {x*y}"
            elif sign == "4":
                answer = f"{x}–{y} = {x-y}"
            else:
                print(answer)
                continue
            break
        print(f"Результат операции: {answer}\n")
    except TypeError:
        print("Операция с несовместимыми типами \n")
        continue
    except ValueError:
        print("Введенное значение не является верным числом \n")
        continue
    except ZeroDivisionError:
        print("Запрещено делить на ноль (0)\n")
        continue
    break
```

**** – умножение***

 # Комментарии предшествующего кода

`while True:`	Начало цикла1 (предположим верно)
`  try:`	
`    print("4 операции в ...`	Вывести сообщение ...
`    x = float(input('Введите...`	Введите число в *x*
`    y = float(input('Введите...`	Введите число в *y*
`    answer = 'Осторожно!!! ...`	Сообщение в случае, если это не 1-4
`    while True:`	Служит для повторения, если он не 1-4
`      print("1-сложение; 2-д..`	
`      operation = input("Вы...`	Выбор операции
`      sign = operation.strip()`	Очистить пробелы до и после
`      if sign == "1":`	Если вы нажмете символ «1»,
`        answer = f"{x}+{y} ..`	то …
`      elif sign == "2":`	В случае если «2», а не «1»,
`        answer = f"{x}/{y}…`	тогда ...
`      elif sign == "3":`	В случае если «3», а не «2»,
`        answer = f"{x}*{y}...`	тогда ...,
`      elif sign == "4":`	В случае если «4», а не «3»,
`        answer = f"{x}-{y}…`	то ...
`      else:`	В случае, если не 1, 2, 3 и не 4,
`        print(answer)`	вывести сообщ. переменной *answer*.
`        continue`	Начать с начала цикла.
`      break`	Выход из цикла №2, если были 1-4.
`    print(f" Результат опера...`	
`  except TypeError:`	Если встречается исключение этого
`    print("Операция с несо…`	типа, то вывести сообщение и
`    continue`	начать с начала цикла №1
`  except ValueError:`	} Объяснение как предыдущее
`    print("Введенное значе…`	
`    continue`	
`  except ZeroDivisionError:`	} То же самое
`    print("Запрещено дели…`	
`    continue`	
`  break`	Выход из цикла №1 и конец.

Различные возможные результаты выполнения

```
# Результат: (одна из возможностей)
4 операции в калькуляторе с 2 числами: +,/,*,–
Введите первое число x = 15.3
Введите второе число y = 3
1-сложение; 2-деление; 3-умножение; 4-вычитание
Выберите тип операции : 2
Результат операции: 15.3/3.0 = 5.1000000000000005
```

```
# Результат: (одна из возможностей – выбор за пределами 1-4)
4 операции в калькуляторе с 2 числами: +,/,*,–
Введите первое число x = 15.3
Введите второе число y = 3
1-сложение; 2-деление; 3-умножение; 4-вычитание
Выберите тип операции : 5
Внимание!!! Нужно выбрать от 1 до 4

1-сложение; 2-деление; 3-умножение; 4-вычитание
Выберите тип операции : 4
Результат операции : 15.3-3.0 = 12.3
```

```
# Результат:
# (одна из возможностей – y принимает строковое значение)
4 операции в калькуляторе с 2 числами: +,/,*,–
Введите первое число x = 15.3
Введите второе число y = a
Введенное значение не является верным числом

4 операции в калькуляторе с 2 числами: +,/,*,–
Введите первое число x = b
Введенное значение не является верным числом
```

```
# Результат: (одна из возможностей – знаменатель равен 0)
4 операции в калькуляторе с 2 числами: +,/,*,–
Введите первое число x = 12
Введите второе число y = 0
1-сложение; 2-деление; 3-умножение; 4-вычитание
Выберите тип операции : 2
Запрещено делить на ноль (0)

4 операции в калькуляторе с 2 числами: +,/,*,–
Введите первое число x = 12
Введите второе число y = 3
1-сложение; 2-деление; 3-умножение; 4-вычитание
Выберите тип операции : 2
Результат операции: 12.0/3.0 = 4.0
```

```
# Результат:
# (одна из возможностей – числитель и знаменатель равны 0)

4 операции в калькуляторе с 2 числами: +,/,*,–
Введите первое число x = 0
Введите второе число y = 0
1-сложение; 2-деление; 3-умножение; 4-вычитание
Выберите тип операции: 2
Запрещено делить на ноль (0)
```

Примечание

Чтобы программа могла запустить этот фрагмент кода:

```
except TypeError:
    print("Операция с несовместимыми типами \n")
    continue
```

необходимо внести некоторые изменения в эту программу, чтобы она могла иметь конфликт типов данных во время обработки. Например,

изменить операцию «x = *float(input('Введите первое число x = '))*» на «x = input('*Введите первое число x = '*)». Таким образом, программа может быть остановлена только при принудительном прерывании.

```
while True:
    try:
        print("4 операции в калькуляторе с 2 числами: +,/,*,–")
        # x = float(input('Введите первое число x = '))
        x = input('Введите первое число x = ')
        y = float(input('Введите второе число y = '))
        answer = 'Осторожно!!! Нужно выбрать от 1 до 4 \n'
        while True:
            print("1-сложение; 2-деление; 3-умноже; 4-вычитание")
            operation = input("Выберите тип операции : ")
            ...
    …
                # Пример, которого следует избегать –
    потому что он повлечёт за собой бесконечное выполнение
                программы без выхода
```

```
# Результат: (случаи, когда x и y разных типов)
4 операции в калькуляторе с 2 числами: +,/,*,–
Введите первое число x = 15.3
Введите второе число y = 5
1-сложение; 2-деление; 3-умножение; 4-вычитание
Выберите тип операции: 3

Операция с несовместимыми типами
```

Продолжение на странице 224

EN IV.2. «Try» and «except» statements

These two teams work together. The «**try**» command is used to check code that may have execution errors and to give instructions via the «**except**» command that should be used if an exception does occur. Let's follow an example to better understand:

Solution with «try» and «except» statements

```
while True:
    try:
        print("4 operations in the calculator with 2 numbers: +,/,*,–")
        x = float(input('Enter the first number x = '))
        y = float(input(' Enter the second number y = '))
        answer = ' Attention!!! You need to choose from 1 to 4 \n '
        while True:
            print("1 – addition; 2– division; 3– multip*; 4– subtraction")
            operation = input("Select the type of operation:")
            sign = operation.strip()
            if sign == "1":
                answer = f"{x}+{y} = {x+y}"
            elif sign == "2":
                answer = f"{x}/{y} = {x/y}"
            elif sign == "3":
                answer = f"{x}*{y} = {x*y}"
            elif sign == "4":
                answer = f"{x}–{y} = {x-y}"
            else:
                print(answer)
                continue
            break
        print(f"The result of the operation: {answer}\n")
    except TypeError:
        print("Operation with incompatible types \n")
        continue
    except ValueError:
        print("The entered value is not a valid number \n")
        continue
    except ZeroDivisionError:
        print("It is forbidden to divide by zero (0)\n")
        continue
    break
```

**** – multiplication***

Comments of the previous program code

while True:	Start of Loop 1 (suppose true)
try:	
print("4 operations in ...	Print a message ...
x = float(input('Enter the...	Enter a number in x
y = float(input('Enter the...	Enter a number in y
answer = 'Attention!!! ...	Message in case it is not “1” – “4”
while True:	Serves for repetition if it is not “1” – “4”
print("1 – addition; 2...	
operation = input("Sel...	Selecting an operation
sign = operation.strip()	Clear spaces before and after
if sign == "1":	If you press the “1” symbol,
answer = f"{x}+{y} .	then …
elif sign == "2":	And if “2”, and not “1”,
answer = f"{x}/{y}…	then ... .
elif sign == "3":	If “3”, and not “2”,
answer = f"{x}*{y}...	then ...,
elif sign == "4":	If you press “4”,
answer = f"{x}-{y}…	then ...
else:	If not 1, 2, 3, or 4,
print(answer)	output the variable message *answer*.
continue	Start from the beginning of the cycle.
break	Exit loop №2, if there were “1” – “4”.
print(f"The result of the...	
except TypeError:	If an exception of this type occurs, then
print("Operation with in…	print the message and start from the
continue	beginning of loop №1
except ValueError:	} Explanation of how the previous
print("The entered value…	
continue	
except ZeroDivisionError:	} Same thing
print("It is forbidden to …	
continue	
break	Exit from loop №1 and end.

The different possible results

```
# Result: (one of the possibilities)
4 operations in the calculator with 2 numbers: +,/,*,–
Enter the first number x = 15.3
Enter the second number y = 3
1 – addition; 2 – division; 3– multip; 4– subtraction
Select the type of operation: 2
The result of the operation: 15.3/3.0 = 5.1000000000000005
```

```
# Result: (one of the possibilities – selection outside “1” – “4”)
4 operations in the calculator with 2 numbers: +,/,*,–
Enter the first number x = 15.3
Enter the second number y = 3
1 – addition; 2 – division; 3– multip; 4– subtraction
Select the type of operation: 5
Attention!!! You need to choose from 1 to 4

1 – addition; 2 – division; 3– multip; 4– subtraction
Select the type of operation: 4
The result of the operation: 15.3-3.0 = 12.3
```

```
# Result: (one of the possibilities – “y” takes a string value)
4 operations in the calculator with 2 numbers: +,/,*,–
Enter the first number x = 15.3
Enter the second number y = a
The entered value is not a valid number

4 operations in the calculator with 2 numbers: +,/,*,–
Enter the first number x = b
The entered value is not a valid number
```

```
# Result: (one of the possibilities – the denominator is 0)
4 operations in the calculator with 2 numbers: +,/,*,–
Enter the first number x = 12
Enter the second number y = 0
1 – addition; 2 – division; 3– multip; 4– subtraction
Select the type of operation: 2
It is forbidden to divide by zero (0)

4 operations in the calculator with 2 numbers: +,/,*,–
Enter the first number x = 12
Enter the second number y = 3
1 – addition; 2 – division; 3– multip; 4– subtraction
Select the type of operation: 2
The result of the operation: 12.0/3.0 = 4.0
```

```
# Result: (one of the possibilities – the numerator and denominator
                                                    are 0)
4 operations in the calculator with 2 numbers: +,/,*,–
Enter the first number x = 0
Enter the second number y = 0
1 – addition; 2 – division; 3– multip; 4– subtraction
Select the type of operation: 2
It is forbidden to divide by zero (0)
```

Note

So that the program can pass on the following code:

```
except TypeError:
    print("Operation with incompatible types \n")
    continue
```

you need to make some changes to this program so that it can have a data type conflict during processing.

For example, change the operation «x = *float(input('Enter the first number x = '))*» to «x = input('*Enter the first number x* = ')». Thus, the program can only be stopped by a forced interruption.

```
while True:
   try:
      print("4 operations in the calculator with 2 numbers: +,/,*,–")
      # x = float(input('Enter the first number x = '))
      x = input('Enter the first number x = ')
      y = float(input('Enter the second number y = '))
      answer = ' Attention!!! You need to choose from 1 to 4 \n'
      while True:
         print("1 – addition; 2 – division; 3– multip; 4– subtraction")
         operation = input("Select the type of operation: ")
         ...
   …
                              # An example to avoid –
because it will cause the program to run indefinitely without exiting
```

```
# Result: (cases where x and y are different types)
4 operations in the calculator with 2 numbers: +,/,*,–
Enter the first number x = 15.3
Enter the second number y = 5
1 – addition; 2 – division; 3– multip; 4– subtraction
Select the type of operation: 3

Operation with incompatible types
```

Continued on page 227

FR IV.2. Instructions *try* et *except*

Ces deux commandes fonctionnent ensemble. La commande «**try**» permet de tester un code qui peut avoir des erreurs d'exécutions et donner à travers la commande «**except**» les indications à suivre si une exception est effectivement rencontrée. Suivons cet exemple résolu pour mieux comprendre:

Résolution avec les instructions «try» et «except»

```
while True:
    try:
        print("4 opérations dans la calculatrice de 2 nombres: +,/,*,–")
        x = float(input('Introduire un premier nombre x = '))
        y = float(input('Introduire un second  nombre y = '))
        answer = 'Attention!!! Il faut choisir entre 1 à 4 \n'
        while True:
            print("1-addition; 2-division; 3-multiplication; 4-soustract")
            operation = input("Choisissez le type d'opération à faire : ")
            sign = operation.strip()
            if sign == "1":
                answer = f"{x}+{y} = {x+y}"
            elif sign == "2":
                answer = f"{x}/{y} = {x/y}"
            elif sign == "3":
                answer = f"{x}*{y} = {x*y}"
            elif sign == "4":
                answer = f"{x}–{y} = {x-y}"
            else:
                print(answer)
                continue
            break
        print(f"Résultat de l'opération: {answer}\n")
    except TypeError:
        print("Opération avec des types incompatibles entre eux \n")
        continue
    except ValueError:
        print("La valeur entrée n'est pas un réel \n")
        continue
    except ZeroDivisionError:
        print("Interdit de diviser par zéro (0) \n")
        continue
    break
```

 # Les commentaires du code précédant

`while True:`	Début de la boucle 1 (considérons vrai)
`  try:`	
`    print("4 opérations dans ...`	Imprime le message ...
`    x = float(input('Introduir...`	Introduire un nombre qui sera considéré
`    y = float(input('Introduir...`	comme réel dans *x* et *y*
`    answer = 'Attention!!! Il ...`	Message dans le cas où ce n'est pas 1-4
`    while True:`	Sert pour la répétition s'il n'est pas 1-4
`      print("1-addition; 2-di...`	
`      operation = input("Ch...`	Introduire un nombre
`      sign = operation.strip()`	Effacer les espaces avant et après
`      if sign == "1":`	Si on a appuyé le symbole «1»,
`        answer = f"{x}+{y} ..`	alors …
`      elif sign == "2":`	Dans le cas où c'est «2» et non «1»,
`        answer = f"{x}/{y}…`	alors ...
`      elif sign == "3":`	Dans le cas où c'est plutôt «3»,
`        answer = f"{x}*{y}...`	alors ...
`      elif sign == "4":`	Dans le cas où c'est plutôt «4»,
`        answer = f"{x}-{y}…`	alors ...
`      else:`	Si le nombre n'est ni 1, 2, 3 ou ni 4,
`        print(answer)`	alors imprimer le message d'en haut.
`        continue`	Recommencer au début de la boucle.
`      break`	Sortir de la boucle №2, si 1, 2, 3 ou 4.
`    print(f"Résultat de l'opér...`	
`  except TypeError:`	S'il rencontre l'exception de ce type,
`    print("Opération avec …`	Imprimer le message et
`    continue`	recommencer au début de la boucle №1
`  except ValueError:`	} Même explication ici.
`    print("La valeur entrée n…`	
`    continue`	
`  except ZeroDivisionError:`	} Même explication ici.
`    print("Interdit de diviser…`	
`    continue`	
`  break`	Sortir de la boucle №1 et Fin

Les différents résultats possibles

```
# Résultat: (une des possibilités)
4 opérations dans la calculatrice de 2 nombres: +,/,*,–
Introduire un premier nombre x = 15.3
Introduire un second  nombre y = 3
1-addition; 2-division; 3-multiplication; 4-soustraction
Choisissez le type d'opération à faire : 2
Résultat de l'opération: 15.3/3.0 = 5.1000000000000005
```

```
# Résultat: (une des possibilités – choix en dehors de 1 à 4)
4 opérations dans la calculatrice de 2 nombres: +,/,*,–
Introduire un premier nombre x = 15.3
Introduire un second  nombre y = 3
1-addition; 2-division; 3-multiplication; 4-soustraction
Choisissez le type d'opération à faire : 5
Attention!!! Il faut choisir entre 1 à 4

1-addition; 2-division; 3-multiplication; 4-soustraction
Choisissez le type d'opération à faire : 4
Résultat de l'opération: 15.3-3.0 = 12.3
```

```
# Résultat: (une des possibilités – y prend la valeur d'une chaîne)
4 opérations dans la calculatrice de 2 nombres: +,/,*,–
Introduire un premier nombre x = 15.3
Introduire un second  nombre y = a
La valeur entrée n'est pas un réel

4 opérations dans la calculatrice de 2 nombres: +,/,*,–
Introduire un premier nombre x = b
La valeur entrée n'est pas un réel

4 opérations dans la calculatrice de 2 nombres: +,/,*,–
Introduire un premier nombre x =
```

```
# Résultat: (une des possibilités – dénominateur zéro)
4 opérations dans la calculatrice de 2 nombres: +,/,*,–
Introduire un premier nombre x = 12
Introduire un second  nombre y = 0
1-addition; 2-division; 3-multiplication; 4-soustraction
Choisissez le type d'opération à faire : 2
Interdit de diviser par zéro (0)

4 opérations dans la calculatrice de 2 nombres: +,/,*,–
Introduire un premier nombre x = 12
Introduire un second  nombre y = 3
1-addition; 2-division; 3-multiplication; 4-soustraction
Choisissez le type d'opération à faire : 2
Résultat de l'opération: 12.0/3.0 = 4.0
```

```
# Résultat: (une des possibilités – numérateur et dénominateur zéro)
4 opérations dans la calculatrice de 2 nombres: +,/,*,–
Introduire un premier nombre x = 0
Introduire un second  nombre y = 0
1-addition; 2-division; 3-multiplication; 4-soustraction
Choisissez le type d'opération à faire : 2
Interdit de diviser par zéro (0)

4 opérations dans la calculatrice de 2 nombres: +,/,*,–
Introduire un premier nombre x =
```

Remarque

Pour que le programme puisse passer sur le code suivant:

```
    except TypeError:
        print("Opération avec des types incompatibles entre eux \n")
        continue
```

il faudra apporter quelques modifications dans ce programme pour qu'il puisse avoir conflit de type des données pendant le traitement. Par exemple, changer l'opération «*x = float(input('Introduire un premier nombre x = '))*» par «*x = input('Introduire un premier nombre x = ')*». Ainsi le programme ne pourra s'arrêter que par interruption forcée.

```
while True:
   try:
      print("4 opérations dans la calculatrice de 2 nombres: +,/,*,–")
      # x = float(input('Introduire un premier nombre x = '))
      x = input('Introduire un premier nombre x = ')
      y = float(input('Introduire un second  nombre y = '))
      answer = 'Attention!!! Il faut choisir entre 1 à 4 \n'
      while True:
         print("1-addition; 2-division; 3-multiplication; 4-soustract")
         operation = input("Choisissez le type d'opération à faire : ")
         ...
   …
                              # Exemple à éviter –
                  car exécution infinie du programme sans sortie
```

```
# Résultat: (cas où x et y sont de types différents)
4 opérations dans la calculatrice de 2 nombres: +,/,*,–
Introduire un premier nombre x = 15.3
Introduire un second  nombre y = 5
1-addition; 2-division; 3-multiplication; 4-soustraction
Choisissez le type d'opération à faire : 3
Opération avec des types incompatibles entre eux

4 opérations dans la calculatrice de 2 nombres: +,/,*, –
Introduire un premier nombre x =
```

Suite à la page 228

RU IV.2. Встроенные исключения

RU Иерархия классов EN Hierarchy FR Hiérarchie des classes

SystemExit	–	
KeyboardInterrupt	–	
GeneratorExit	–	
Exception	+	

StopIteration	–	
StopAsyncIteration	–	
ArithmeticError	+	

FloatingPointError
OverflowError
ZeroDivisionError

AssertionError	–	
AttributeError	–	
BufferError	–	
EOFError	–	
ImportError	+	

FloatingPointError

LookupError	+	

IndexError
KeyError

MemoryError	–	
NameError	+	

UnboundLocalError

OSError	+	

BlockingIOError	–	
ChildProcessError	–	
ConnectionError	+	

BrokenP...
Connec…
Connec…
Connec…

RU *Продолжение иерархии классов* EN *Continuation of the class hierarchy* FR *Suite de la hiérarchie des classes*

OSError	+			
		FileExistsError	–	
		FileNotFoundError	–	
		InterruptedError	–	
		IsADirectoryError	–	
		NotADirectoryError	–	
		PermissionError	–	
		ProcessLookupError	–	
		TimeoutError	–	

ReferenceError	–			
RuntimeError	+			
		NotImplementedError	–	
		RecursionError	–	

SyntaxError	+			
		IndentationError	+	
				TabError

SystemError	–			
TypeError	–			
ValueError	+			
		UnicodeError	+	
				Unic…D...
				Unic…E...
				Unic…T...

Warning	+			
		DeprecationWarning		
		PendingDeprecationWarning		
		RuntimeWarning		

RU *Продолжение иерархии классов* EN *Continuation of the class hierarchy* FR *Suite de la hiérarchie des classes*

Warning	+	

SyntaxWarning
UserWarning
FutureWarning
ImportWarning
UnicodeWarning
BytesWarning
ResourceWarning

RU Расшифровка слов из таблицы (Имя классы – исключения)

EN Decryption of words from the table (Name of classes – exceptions)

FR Décryptage de certains mots (nom des classes)

BrokenPipeError	⇦	BrokenP...
ConnectionAbortedError	⇦	Connec…
ConnectionRefusedError	⇦	Connec…
ConnectionResetError	⇦	Connec…

UnicodeDecodeError	⇦	Unic…D...
UnicodeEncodeError	⇦	Unic…E...
UnicodeTranslateError	⇦	Unic…T...

Значение некоторых исключений

ZeroDivisionError – это происходит, когда знаменатель деления равен нулю, а числитель – ненулевой аргумент.

OverflowError – происходит, когда невозможно объявить результат (арифметической) операции, который получается слишком большим для представления.

ConnectionError – при возникновении проблем с подключением к сети.

FileExistsError – это происходит, когда вы хотите создать файл или папка, которая уже существует.

FileNotFoundError – он появляется, когда файл или каталог запрашивается, пока он ещё не существует.

PermissionError – когда вы хотите выполнить операцию без разрешения или необходимых прав.

TimeoutError – это происходит, когда вы хотите запустить системную функцию, время которой истекло.

EOFError – появляется, когда находит конец файла там, где он этого не ожидал.

OSError – исключение возникает, когда возникает системная ошибка, например: "файл не найден" или "полный диск".

ValueError – при возникновении путаницы между типами, введенное значение не соответствует указанным параметрам;

TypeError – возникает, когда значение в программном коде не соответствует типу; когда операция с несовместимыми друг с другом типами и т. д.

Продолжение на странице 229

EN IV.3. Built-in exceptions

Hierarchy (*see pages 222 and 223*)

Meaning of some exceptions

ZeroDivisionError – this occurs when the division denominator is zero and the numerator is a non-zero argument.

OverflowError – occurs when it is not possible to declare the result of an (arithmetic) operation that is too large to represent.

ConnectionError – if there are problems with connecting to the network.

FileExistsError – this happens when you want to create a file or folder that already exists.

FileNotFoundError – it appears when a file or directory is requested while it does not yet exist.

PermissionError – when you want to perform an operation without permission or the necessary permissions.

TimeoutError – this happens when you want to run a system function that has timed out.

EOFError – appears when it finds the end of the file where it did not expect it.

OSError – an exception occurs when a system error occurs, such as “file not found” or “full disk”.

ValueError – if there is confusion between types, the entered value does not match the specified parameters;

TypeError – occurs when a value in the program code does not match the type; when an operation with incompatible types, etc.

Continued on page 231

FR IV.3. Les classes d’exceptions natives

La hiérarchie des classes (*voir les pages 222 et 223*)

Signification de certaines exceptions

ZeroDivisionError – Il intervient, lorsque le dénominateur d'une division est zéro alors que le numérateur est argument non nul.

OverflowError – Intervient lorsqu'il est impossible d'annoncer le résultat d'une opération (arithmétique) qui semble etre trop grand.

ConnectionError – lorsqu'il y a des problèmes de connections à un réseau.

FileExistsError – Il intervient lorsqu'on veut créer un fichier ou répertoire(dossier) qui existe déjà.

FileNotFoundError – Il apparaît lorsque le fichier ou un répertoire est demandé alors qu'il n'existe pas encore.

PermissionError – lorsqu'on veut exécuter une opération sans avoir la permission ou les droits requis.

TimeoutError – Il intervient lorsqu'on veut exécuter une fonction du système dont le temps a expiré.

EOFError – Intervient lorsqu'il trouve la fin du fichier là où il ne l'attendait pas.

OSError – l'exception est déclenchée lorsqu'on a une erreur liée au système par exemple: "fichier non trouvé" ou "disque plein".

ValueError – Intervient lorsqu'il y a confusion entre les types, la valeur introduite ne correspond pas aux paramètres indiqués;

TypeError – Intervient lorsque la valeur dans le code du programme ne correspond pas au type; lorsque l'opération avec des types incompatibles entre eux etc.

Suite à la page 232

RU IV.4. Инструкции "*try*", "*except*" и "*else*"

Было уже рассмотрено, что команды «**try**» и «**except**» работают попарно. Для дополнения, можно добавить еще одну команду: «**else**». Она используется для контроля случаев, когда команда **«try»** не вызывает никаких исключений. Для ясности, перепишем предыдущий пример, чтобы лучше понять:

Решение с инструкциями «try», «except» и «else»

```
while True:
    try:
        print("4 операции в калькуляторе с 2 числами: +,/,*,–")
        x = float(input('Введите первое число x = '))
        y = float(input('Введите второе число y = '))
        answer = 'Внимание!!! Нужно выбрать от 1 до 4 \n'
        while True:
            print("1-сложение; 2-деление; 3-умноже*; 4-вычитание")
            operation = input("Выберите тип операции:")
            sign = operation.strip()
            # продолжение следует
```

Продолжение кода программы

```
            if sign == "1":
                answer = f"{x}+{y} = {x+y}"
            elif sign == "2":
                answer = f"{x}/{y} = {x/y}"
            elif sign == "3":
                answer = f"{x}*{y} = {x*y}"
            elif sign == "4":
                answer = f"{x}–{y} = {x-y}"
            else:
                print(answer)
                continue
            break
        # print(f"Результат операции: {answer}\n")
    except TypeError:
        print("Операция с несовместимыми типами \n")
        continue
    except ValueError:
        print("Введенное значение не является верным числом \n")
        continue
    except ZeroDivisionError:
        print("Запрещено делить на ноль (0) \n")
        continue
    else:
        print(f"Результат операции: {answer}\n")

    break
```

```
# Результат: (одна из возможностей – то же самое, что и
                                        первый пример)
4 операции в калькуляторе с 2 числами: +,/,*,–
Введите первое число x = 15.3
Введите второе число y = 3
1-сложение; 2-деление; 3-умножение; 4-вычитание
Выберите тип операции : 2
Результат операции 15.3/3.0 = 5.1000000000000005
```

Продолжение на странице 234

EN IV.4. «Try», «except» and «else» statements

It has already been considered that the «**try**» and «**except**» commands work in pairs. To supplement it, you can add another command: «**else**». It is used to control cases when the **«try»** command does not raise any exceptions. For clarity, we will rewrite the previous example to better understand:

```
while True:
    try:
        print("4 operations in the calculator with 2 numbers: +,/,*,-")
        x = float(input('Enter the first number x ='))
        y = float(input('Enter the second number y = '))
        answer = 'Attention!!! You need to choose from 1 to 4 \n'
        while True:
            print("1 – addition; 2– division; 3– multip*; 4– subtraction")
            operation = input("Select the type of operation:")
            sign = operation.strip()
            if sign == "1":
                answer = f"{x}+{y} = {x+y}"
            elif sign == "2":
                answer = f"{x}/{y} = {x/y}"
            elif sign == "3":
                answer = f"{x}*{y} = {x*y}"
            elif sign == "4":
                answer = f"{x}–{y} = {x-y}"
            else:
                print(answer)
                continue
            break
        # print(f"The result of the operation: {answer}\n")
    except TypeError:
        print("Операция с несовместимыми типами \n")
        continue                                    # To be continued
```

Continuation of the program code

```
    except ValueError:
        print("The entered value is not a valid number \n")
        continue
    except ZeroDivisionError:
        print("It is forbidden to divide by zero (0)\n")
        continue
    else:
        print(f"The result of the operation {answer}\n")

    break
```

```
# Result: (one of the possibilities – same as the first example)
4 operations in the calculator with 2 numbers: +,/,*,-
Enter the first number x = 15.3
Enter the second number y = 3
1 – addition; 2– division; 3– multip*; 4– subtraction
Select the type of operation: 2
The result of the operation 15.3/3.0 = 5.1000000000000005
```

Continued on page 236

FR IV.4. Instructions «try», «except» et «else»

Nous avons vu que les commandes «**try**» et «**except**» fonctionnaient ensemble. Nous pouvons ajouter une autre commande: «**else**». Elle permet de controler le cas où aucune exception n'a été levée par la commande «*try*». Réécrivons l'exemple précédant pour mieux comprendre:

Résolution avec les instructions «try», «except» et «else»

```
while True:
    try:
        print("4 opérations dans la calculatrice de 2 nombres: +,/,*,-")
        x = float(input('Introduire un premier nombre x = '))
        y = float(input('Introduire un second  nombre y = '))
        # À suivre
```

Suite du code du programme

```
        answer = 'Attention!!! Il faut choisir entre 1 à 4 \n'
        while True:
            print("1-addition; 2-division; 3-multiplication; 4-soustract")
            operation = input("Choisissez le type d'opération à faire : ")
            sign = operation.strip()
            if sign == "1":
                answer = f"{x}+{y} = {x+y}"
            elif sign == "2":
                answer = f"{x}/{y} = {x/y}"
            elif sign == "3":
                answer = f"{x}*{y} = {x*y}"
            elif sign == "4":
                answer = f"{x}–{y} = {x-y}"
            else:
                print(answer)
                continue
            break
        # print(f"Résultat de l'opération: {answer}\n")
    except TypeError:
        print("Opération avec des types incompatibles entre eux \n")
        continue
    except ValueError:
        print("La valeur entrée n'est pas un réel \n")
        continue
    except ZeroDivisionError:
        print("Interdit de diviser par zéro (0) \n")
        continue
    else:
        print(f"Résultat de l'opération: {answer}\n")

    break
# Fin
```

```
# Résultat: (une des possibilités – même chose que le 1ier exemple)
4 opérations dans la calculatrice de 2 nombres: +,/,*,–
Introduire un premier nombre x = 15.3
Introduire un second  nombre y = 3
1-addition; 2-division; 3-multiplication; 4-soustraction
Choisissez le type d'opération à faire : 2
Résultat de l'opération: 15.3/3.0 = 5.1000000000000005
```

Suite à la page 239

RU IV.5. Инструкции “*try*”, “*except*”, “*else*” и “*finally*”

Здесь мы показываем последний элемент имени «***finally***». Код, который находится в этой части, выполняется во всех случаях, независимо от того, есть ошибка или нет. Итак, как ввести команду «***finally***» в предыдущем примере ? Какую информацию или операцию в качестве примера необходимо выполнить в этой части программы? Попробуем определить, сколько времени каждый пользователь затратит для выполнения всего кода. Для этого необходимо внести следующие изменения в предшествующий пример:

Импортируем модуль time для работы с временем:

import time

Определим время запуска программы:

begin = time. time()

Определим время завершения программы:

end = time. time()

Вычисляем разницу и определяем время, потраченное на запуск программы

print ("Затраченное время для выполнения программы было", end-begin, " секунды")

Решение с инструкциями «try», «except», «else» и «finally»

```
import time

begin = time.time()
while True:
    try:
        print("4 операции в калькуляторе с 2 числами: +,/,*,-")
        x = float(input('Введите первое число x = '))
        y = float(input('Введите второе число y = '))
        answer = 'Внимание!!! Нужно выбрать от 1 до 4 \n '
        while True:
            print("1-сложение; 2-деление; 3-умноже*; 4-вычитание")
            operation = input("Выберите тип операции : ")
            sign = operation.strip()
            if sign == "1":
                answer = f"{x}+{y} = {x+y}"
            elif sign == "2":
                answer = f"{x}/{y} = {x/y}"
            elif sign == "3":
                answer = f"{x}*{y} = {x*y}"
            elif sign == "4":
                answer = f"{x}–{y} = {x-y}"
            else:
                print(answer)
                continue
            break
    except TypeError:
        print("Операция с несовместимыми типами \n")
        continue
    except ValueError:
        print("Введенное значение не является верным числом \n")
        continue
                                    ...
```

Продолжение коды программы

```
                                    ...
    except ZeroDivisionError:
        print("Запрещено делить на ноль (0) \n")
        continue
    else:
        print(f" Результат операции: {answer}\n")
    finally:
        end = time.time()
        t_mis = end - begin
        print("Затраченное время для выполнения программы
                                    было %.3f секунды " % t_mis )

    break
```

```
# Результат: (одна из возможностей – то же самое, что и
                                        первый пример)
4 операции в калькуляторе с 2 числами: +,/,*,-
Введите первое число x = 15.3
Введите второе число y = 3
1-сложение; 2-деление; 3-умножение; 4-вычитание
Выберите тип операции : 2
Результат операции: 15.3/3.0 = 5.1000000000000005
Затраченное время для выполнения программы было
17.730 секунды
```

Продолжение на странице 241

EN IV.5. «try», «except», «else» and «finally» statements

Here we show the last element of the name «***finally***». The code that is in this part is executed in all cases, regardless of whether there is an error or not. So, how do I enter the «***finally***» command in the previous example?

What information or example operation should be performed in this part of the program? Let's try to determine how much time each user will spend to execute all the program code. To do this, make the following changes to the previous example:

Importing the ***time*** module for working with time:

import time

Let's determine the start time of the program:

begin = time.time()

Determine the end time of the program:

end = time.time()

We calculate the difference and determine the time spent on running the program.

print ("The elapsed time to execute the program was", end-begin, " seconds")

Solution with «try», «except», «else» and «finally» statements

```
import time

begin = time.time()
while True:
    try:
        print("4 operations in the calculator with 2 numbers: +,/,*,- ")
        x = float(input('Enter the first number x = '))
        y = float(input('Enter the second number y = '))
        answer = 'Attention!!! You need to choose from 1 to 4 \n'
        while True:
            print("1– addition; 2– division; 3– multip*; 4– subtraction ")
            operation = input("Select the type of operation: ")
            sign = operation.strip()
            # To be continued
```

Continuation of the program code

```
        if sign == "1":
            answer = f"{x}+{y} = {x+y}"
        elif sign == "2":
            answer = f"{x}/{y} = {x/y}"
        elif sign == "3":
            answer = f"{x}*{y} = {x*y}"
        elif sign == "4":
            answer = f"{x}–{y} = {x-y}"
        else:
            print(answer)
            continue
        break
    except TypeError:
        print("Operation with incompatible types \n ")
        continue
    except ValueError:
        print("The entered value is not a valid number \n ")
        continue
    except ZeroDivisionError:
        print("It is forbidden to divide by zero (0)\n ")
        continue
    else:
        print(f"The result of the operation: {answer}\n")
    finally:
        end = time.time()
        t_mis = end - begin
        print("The elapsed time to execute the program was %.3f
                                    seconds" % t_mis )
    break
```

```
# Result: (one of the possibilities – same as the first example)
4 operations in the calculator with 2 numbers: +,/,*,-
Enter the first number x = 15.3
Enter the second number y = 3
1 – addition; 2– division; 3– multip*; 4– subtraction
Select the type of operation: 2
The result of the operation 15.3/3.0 = 5.1000000000000005
The elapsed time to execute the program was 17.730 seconds
```

Continued on page 249

FR IV.5. Instructions «try», «except», «else» et «finally»

Ici, nous faisons apparaître un dernier élément du nom de «***finally***». Le code qui se trouve dans cette partie est exécuté dans tous les cas, qu'il y ait erreur ou pas. Alors, comment l'introduire la commande «finally» dans l'exemple précédent ? Quelle information ou opération à titre d'exemple doit t-on obligatoirement effectuer dans cette partie du programme? Essayons de déterminer le temps mis que fera chaque utilisateur pour exécuter tout le code. Pour cela, voila les changements qu'il faudra apporter à l'exemple précédant:

Importons le module ***time*** pour travailler avec les heures:

import time

Déterminons le temps de démarrage du programme:

début = time.time()

Déterminons le temps auquel le programme s'est achevé:

fin = time.time()

Faisons la différence et déterminons le temps mis pour exécuter le programme

print("Durée pour exécuter le programme était de ", fin-début," secondes")

Résolution avec les instructions «try», «except» «else» et «finally»

```
import time

début = time.time()
while True:
    try:
        print("4 opérations dans la calculatrice de 2 nombres: +,/,*,-")
        x = float(input('Introduire un premier nombre x = '))
        y = float(input('Introduire un second  nombre y = '))
        answer = 'Attention!!! Il faut choisir entre 1 à 4 \n'
        while True:
            print("1-addition; 2-division; 3-multiplication; 4-soustract")
            operation = input("Choisissez le type d'opération à faire : ")
            sign = operation.strip()
            if sign == "1":
                answer = f"{x}+{y} = {x+y}"
            elif sign == "2":
                answer = f"{x}/{y} = {x/y}"
            elif sign == "3":
                answer = f"{x}*{y} = {x*y}"
            elif sign == "4":
                answer = f"{x}–{y} = {x-y}"
            else:
                print(answer)
                continue
            break
    except TypeError:
        print("Opération avec des types incompatibles entre eux \n")
        continue
    except ValueError:
        print("La valeur entrée n'est pas un réel \n")
        continue
# À suivre ...
```

Suite du code du programme

```
                                                    ...
    except ZeroDivisionError:
        print("Interdit de diviser par zéro (0) \n")
        continue
    else:
        print(f"Résultat de l'opération: {answer}\n")
    finally:
        fin = time.time()
        t_mis = fin - début
        print("la durée pour exécuter le programme était de %.3f
                                                    secondes" % t_mis )

    break
```

```
# Résultat: (une des possibilités – même chose que le 1ier exemple)
4 opérations dans la calculatrice de 2 nombres: +,/,*,-
Introduire un premier nombre x = 15.3
Introduire un second  nombre y = 3
1-addition; 2-division; 3-multiplication; 4-soustract
Choisissez le type d'opération à faire : 2
Résultat de l'opération: 15.3/3.0 = 5.1000000000000005

La durée pour exécuter le programme était de 17.730 secondes
```

Suite à la page 257

RU IV.6. «Raise» – чем отличается от других исключений

Мы используем «***raise***» для создания исключений, где мы знаем, что могут быть «логические» ошибки, которые другие исключения не смогут автоматически обнаружить.

Например, если мы вернемся к нашему предыдущему примеру, мы можем использовать метод «raise» для перехвата ошибок всех

пользователей, которые вместо того, чтобы нажимать цифры 1, 2, 3 или 4, для выполнения одной из 4-х операций, нажимаем другой символ. Еще один пример об оценке учащихся, которым нужно присвоить уровень знаний: совершенно очевидно, что с различными системами образования, у нас тоже получаются разные шкалы для оценивания. И если нужно присвоить оценку учащемуся, который учится во французской системе, то очевидно, что она может варьироваться только между 0 и 20.

Следующий код показывает нам, как с помощью метода «raise» мы можем предотвратить выполнение последующих вычислений с несовместимыми данными (оценки находящиеся за пределами интервала от 0 до 20). Пользователь, который будет набирать цифру, не входящую в интервал от 0 до 20, увидит, что программа остановится (сразу после этого).

Обработка исключений с помощью инструкции «raise»

```
try:
    g = int(input("Введите оценку 'g':"))
    if (g >= 0) & (g <= 20):
        print("Совместимая оценка с французской системой ")
    else:
        raise ValueError("Внимание!!! g не д. б. > 20 или < 0!!!")
    print("С этой оценкой, Вы можете продолжать свои
                                                операции.")
    # ...
except ValueError as nom_err_raise_si:
    print("Ваша оценка не соответствует правилам.")
    print(nom_err_raise_si)  # Он позволяет печатать параметр в
                                                методе "raise".
```

```
# Результат: (вариант №1: Если совместимые данные)
Введите оценку 'g':12
Совместимая оценка с французской системой
С этой оценкой, Вы можете продолжать свои операции.
```

В случае, если оценка выходит за пределы интервала, программа будет завершаться со следующем сообщением:

```
# Результат: (вариант №2: Если не совместимые данные)
Введите оценку 'g':21
Ваша оценка не соответствует правилам.
Осторожно!!! g не д. б. > 20 или < 0!!!
```

На каком уровне можно применить метод “*raise*” в нашем калькуляторе из четырех операции для двух цифр?

Мы уже ответили на этот вопрос в предыдущем абзаце, когда вводили с клавиатуры целое число от 1 до 4. Давайте рассмотрим этот же пример, сняв возможность повторного введения данных в случае ошибки. Пусть ошибка приведет к немедленному прекращению выполнения кода.

```
import time

begin = time.time()
try:
    print("4 операции в калькуляторе с 2 числами: +,/,*,-")
    x = float(input(' Введите первое число x = '))
    y = float(input(' Введите второе число y = '))
    answer = "Осторожно!!! Нужно выбрать от 1 до 4'
    print("1-сложение; 2-деление; 3-умножение; 4-вычитание")
    operation = input("Выберите тип операции: ")
    sign = operation.strip()
# Продолжение следует...
```

Продолжение кода

```
    if sign == "1":
        answer = f"{x}+{y} = {x+y}"
    elif sign == "2":
        answer = f"{x}/{y} = {x/y}"
    elif sign == "3":
        answer = f"{x}*{y} = {x*y}"
     elif sign == "4":
        answer = f"{x}–{y} = {x-y}"
     else:
        raise ValueError("Введенное значение неверно :
                          соблюдайте заданный интервал.")
except ValueError as nom_erreur_raise:
    print("Введенное значение не является допустимым числом
                                  или ожидаемым.\n")
    print(nom_erreur_raise) # Отображает информацию, которая
  # находится в параметре встроенного исключения ValueError.
except ZeroDivisionError as nom_erreur_de_ZDE:
    print("Запрещено делить на ноль (0) \n") # Передача ошибки,
                                  # которая будет отображаться
    print(nom_erreur_de_ZDE) # Отображает базовой ошибки,
                                  # так как он не имеет "raise".
else:
    print(f"Результат операции: {answer}\n")
finally:
    end = time.time()
    t_mis = end - begin
    print("Время выполнения программы составляло %. 3f
  секунд." % t_mis)
```

Возможные результаты, которые мы можем получить с различными исходными данными

```
# Результат: (вариант №1: случай, когда ошибка вызывает
                                        метод «raise»)
4 операции в калькуляторе с 2 числами: +,/,*,-"
Введите первое число x = 3
Введите второе число y = 2
1-сложение; 2-деление; 3-умножение; 4-вычитание
Выберите тип операции: 5
Введенное значение не является допустимым числом или
                                        ожидаемым.

Введенное значение неверно : соблюдайте заданный интервал.
Время выполнения программы составляло 11,651 секунд.
```

```
# Результат: (Вариант 2: случай, когда ошибка проходит через
                                        метод «except»)
4 операции в калькуляторе с 2 числами: +,/,*,-"
Введите первое число x = 4
Введите второе число y = 0
1-сложение; 2-деление; 3-умножение; 4-вычитание
Выберите тип операции: 2
Запрещено делить на ноль (0)

float division by zero
Время выполнения программы составляло 14,784 секунд.
```

```
# Результат: (Вариант 3: случай, когда ошибка проходит через
                                        метод «except»)
4 операции в калькуляторе с 2 числами: +,/,*,-"
Введите первое число x = c
```

```
…
Введенное значение не является допустимым числом или
                                        ожидаемым.
could not convert string to float: 'c'
Время выполнения программы составляло 4,111 секунд.
```

В предыдущих примерах были ошибки, которые непосредственно приводили к прерыванию выполнения кода с неточными данными. Изменим код так, чтобы он мог дать нам возможность повторно вводить данные без повторного интерпретирования кода.

Для достижения этой цели мы будем использовать следующие методы: «while», «continue» и «break».

```
import time

begin = time.time()
while True:
    try:
        print("4 операции в калькуляторе с 2 числами: +,/,*,–")
        x = float(input('Введите первое число x = '))
        y = float(input('Введите второе число y = '))
        answer = ' Осторожно!!! Нужно выбрать от 1 до 4 \n'
        while True:
            print("1-сложение; 2-деление; 3-умножение; 4-вычит...")
            operation = input("Выберите тип операции: ")
            sign = operation.strip()
            if sign == "1":
                answer = f"{x}+{y} = {x+y}"
            elif sign == "2":
                answer = f"{x}/{y} = {x/y}"
# ... продолжение следует…
```

Продолжение кода программы

```
            elif sign == "3":
                answer = f"{x}*{y} = {x*y}"
            elif sign == "4":
                answer = f"{x}–{y} = {x-y}"
            else:
                print(answer)
                continue
                break
        except TypeError:
            print("Операция с несовместимыми типами \n")
            continue
        except ValueError:
            print("Введенное значение не допустимым числом \n")
            continue
        except ZeroDivisionError:
            print("Запрещено делить на ноль (0) \n")
            continue
        else:
            print(f"Результат операции: {answer}\n")
        finally:
            end = time.time()
            t_mis = end - begin
            print("Время выполнения программы составляло %. 3f
                                                    секунд." % t_mis)
            break
```

Возможные результаты, которые мы можем получить с различными исходными данными

```
# Результат: (вариант №1: случай, когда ошибка вызывает
                                        метод «raise»)
4 операции в калькуляторе с 2 числами: +,/,*,-"
Введите первое число x = 3
Введите второе число y = 2
1-сложение; 2-деление; 3-умножение; 4-вычитание
Выберите тип операции: 5
Осторожно!!! Нужно выбрать от 1 до 4

1-сложение; 2-деление; 3-умножение; 4-вычитание
Выберите тип операции:
```

```
# Результат: (Вариант 2: случай, когда ошибка проходит через
                                        метод «except»)
4 операции в калькуляторе с 2 числами: +,/,*,-"
Введите первое число x = 4
Введите второе число y = 0
1-сложение; 2-деление; 3-умножение; 4-вычитание
Выберите тип операции: 2
Запрещено делить на ноль (0)

Время выполнения программы составляло 7.861 секунд.
4 операции в калькуляторе 2 числа: +,/,*,-"
Введите первое число x =
```

```
# Результат: (Вариант 3: случай, когда ошибка проходит через
                                        метод «except»)
4 операции в калькуляторе с 2 числами: +,/,*,-"
Введите первое число x = c
Введенное значение не является допустимым числом

Время выполнения программы составляло 4,373 секунд.
4 операции в калькуляторе 2 числа: +,/,*,-"
Введите первое число x =
```

```
# Результат: (Вариант 4: случай, когда нет ошибки)
4 операции в калькуляторе с 2 числами: +,/,*,-"
Введите первое число x = 12
Введите второе число y = 3
1-сложение; 2-деление; 3-умножение; 4-вычитание
Выберите тип операции: 2
Результат операции: 12,0/3,0 = 4,0

Время выполнения программы составляло 17,373 секунд.
```

Упражнение 4.1

Решите все предыдущие упражнения снова, добавив, если это возможно, инструкции “*try*”, “*except*”, “else”, “*finally*”или/и “*raise*”. Цель обновления заключает в том, чтобы сделать предыдущие коды программ более эффективными.

Продолжение на страницы 266 и 267

EN IV.6. «Raise» – how does it differ from other exceptions

We use «***raise***» to create exceptions, where we know that there may be «logical» errors that other exceptions will not be able to detect automatically.

For example, if we go back to our previous example, we can use the «***raise***» method to catch the errors of all users who, instead of pressing the

numbers “1”, “2”, “3” or “4”, to perform one of the 4 operations, press another character. Another example about the assessment of students who need to be assigned a level of knowledge: it is quite obvious that with different educational systems, we also get different scales for assessment. And if you want to assign a grade to a student who is studying in the French system, then it is obvious that it can only vary between 0 and 20.

The following code shows us how we can use the «***raise***» method to prevent subsequent calculations with incompatible data (estimates that are outside the range from 0 to 20). A user who will type a number that is not in the range from 0 to 20 will see that the program will stop (immediately after that).

Resolving exceptions with the «raise» statement

```
try:
    g = int(input("Introduce grade 'g': ")
    if (g >= 0) & (g <= 20):
        print("The grade compatible with the French system ")
    else:
        raise ValueError("Attention!!! 'g' must not be > 20 or < 0!!!")
    print("You could continue your operations with this mark.")
    # ...
except ValueError as nom_err_raise_si:
    print("Your grade is not compatible with the regulations.")
    print(nom_err_raise_si)  # Print the parameter in the "raise"
                                                    method
```

```
# The result: (Variant 1: if compatible data)
Introduce grade 'g': 12
The grade compatible with the French system
You could continue your operations with this mark.
```

In the event that the grade goes outside the interval, we will notice the termination of the program with the following information:

```
# The result: (Variant 2: if data not compatible)
Introduce grade 'g': 21
Your grade is not compatible with the regulations.
Attention!!! 'g' must not be > 20 or < 0!!!
```

At what place can we introduce the «raise» method in our four-operation calculator for two numbers?

We have already answered this in the previous paragraph, when typing an integer between *1* and *4*. Let's look at this example with the exception of the possibility to resume the input of the data in case of an error; the error must cause the execution of the code to stop immediately.

```
import time

begin = time.time()
try:
    print("4 operations in the calculator with 2 numbers: +,/,*,–)
    x = float(input('Enter the first number x = '))
    y = float(input('Enter the second number y = '))
    answer = ' Attention!!! You need to choose from 1 to 4 \n '
    print("1– addition; 2– division; 3– multip*; 4– subtraction")
    operation = input("Select the type of operation: ")
    sign = operation.strip()
    if sign == "1":
        answer = f"{x}+{y} = {x+y}"
    elif sign == "2":
        answer = f"{x}/{y} = {x/y}"
    elif sign == "3":
        answer = f"{x}*{y} = {x*y}"
     elif sign == "4":
        answer = f"{x}–{y} = {x-y}"
```

Continuation of the program code

```
    else:
        raise ValueError("The value entered is incorrect: respect the
                                                    given interval.")
except ValueError as nam_error_raise:
    print("The entered value is not a valid or expected value. \n")
    print(nam_error_raise) # Displays the information that is in the
                                # exception parameter ValueError
except ZeroDivisionError as nom_erreur_of_ZDE:
    print("It is forbidden to divide by zero (0)\n") # Translation of
                                        # the error that will be displayed
    print(nam_error_of_ZDE) # Displays the initial error type,
                            # because it does not have a "raise"method
else:
    print(f"The result of operation: {answer}\n")
finally:
    end = time.time()
    t_mis = end – begin
    print("The elapsed time to execute the program was %.3f
                                                    seconds" % t_mis)
```

The possible results we can get with different data

```
# The result: (Variant 1: case the error calls the “raise” method)
4 operations in the calculator with 2 numbers: +,/,*,-
Enter the first number x = 3
Enter the second number y = 2
1 – addition; 2– division; 3– multip*; 4– subtraction
Select the type of operation: 5
The entered value is not a valid or expected value.

The value entered is incorrect: respect the given interval.
The elapsed time to execute the program was 11.651 seconds
```

```
# The result: (Variant 2: case the error goes through the «except»
                                                    method)
4 operations in the calculator with 2 numbers: +,/,*,-
Enter the first number x = 4
Enter the second number y = 0
1 – addition; 2– division; 3– multip*; 4– subtraction
Select the type of operation: 2
It is forbidden to divide by zero (0)

float division by zero
The elapsed time to execute the program was 14.784 seconds
```

```
# The result: (Variant 3: case the error goes through the «except»
                                                    method)
4 operations in the calculator with 2 numbers: +,/,*,-
Enter the first number x = c
The entered value is not a valid or expected value.

could not convert string to float: 'c'
The elapsed time to execute the program was 4.111 seconds
```

In the previous examples, we had errors that directly led to the interruption of the code execution with incorrect data. Let's change the code so that it can give us the ability to re-enter new data without restarting the code interpretation.

To achieve this, we will use the following methods: «while», «continue» and «break».

```
import time

begin = time.time()
while True:
    try:
        print("4 operations in the calculator with 2 numbers: +,/,*,-")
        x = float(input('Enter the first number x = '))
        y = float(input('Enter the second number y = '))
        answer = 'Attention!!! You need to choose from 1 to 4 \n '
        while True:
            print("1– addition; 2– division; 3– multip*; 4– subtraction ")
            operation = input("Select the type of operation:")
            sign = operation.strip()
            if sign == "1":
                answer = f"{x}+{y} = {x+y}"
            elif sign == "2":
                answer = f"{x}/{y} = {x/y}"
            elif sign == "3":
                answer = f"{x}*{y} = {x*y}"
            elif sign == "4":
                answer = f"{x}–{y} = {x-y}"
            else:
                print(answer)
                continue
                break
    except TypeError:
        print("Operation with incompatible types \n")
        continue
    except ValueError:
        print("The entered value is not a valid number \n")
        continue                                    # ... to be continued…
```

Continuation of the program code

```
    except ZeroDivisionError:
        print("It is forbidden to divide by zero (0) \n")
        continue
    else:
        print(f"The result of the operation: {answer}\n")
    finally:
        end = time.time()
        t_mis = end – begin
        print("The elapsed time to execute the program was %.3f
                    seconds " % t_mis)
    break
```

The possible results we can get with different data

```
# The result: (Variant 1: case the error calls the “raise” method)
4 operations in the calculator with 2 numbers: +,/,*,-
Enter the first number x = 3
Enter the second number y = 2
1 – addition; 2– division; 3– multip*; 4– subtraction
Select the type of operation: 5
Attention!!! You need to choose from 1 to 4

1 – addition; 2– division; 3– multip*; 4– subtraction
Select the type of operation:
```

```
# The result: (Variant 2: case the error goes through the «except»
                                                    method)
4 operations in the calculator with 2 numbers: +,/,*,-
Enter the first number x = 4
Enter the second number y = 0
1 – addition; 2– division; 3– multip*; 4– subtraction
Select the type of operation: 2
It is forbidden to divide by zero (0)

The elapsed time to execute the program was 7.861 seconds
4 operations in the calculator with 2 numbers: +,/,*,-
Enter the first number x =
```

```
# The result: (Variant 3: case the error goes through the «except»
                                                    method)
4 operations in the calculator with 2 numbers: +,/,*,-
Enter the first number x = c
The entered value is not a valid number

The elapsed time to execute the program was 4.373 seconds
4 operations in the calculator with 2 numbers: +,/,*,-
Enter the first number x =
```

```
# The result: (Variant 4: case where no error)
4 operations in the calculator with 2 numbers: +,/,*,-
Enter the first number x = 12
Enter the second number y = 3
1 – addition; 2– division; 3– multip*; 4– subtraction
Select the type of operation: 2
The result of the operation: 12.0/3.0 = 4.0

The elapsed time to execute the program was 17.373 seconds
```

Exercice 4.1

Repeat all the previous exercises, adding, if possible, the statements: "*try*", "*except*", "*else*", "*finally*" or/and "*raise*". The purpose of the update is to make these program codes more efficient.

Continued on pages 266 and 270

FR IV.6. «Raise» – Quelle différence avec les autres exceptions

On utilise «***raise***» pour créer des exceptions là où nous savons qu'il peut avoir des erreurs de type «logique» que le autres exceptions ne pourront pas détecter automatiquement.

Par exemple, si nous revenons sur notre exemple précédant, nous pouvons utiliser cette méthode «raise» pour intercepter les erreurs de tous les utilisateurs qui au lieu d'appuyer les chiffres “1”, “2”, “3” ou “4” pour faire l'une des 4 opérations, saisirons un autre symbole. Un autre exemple sur la note d’un étudiant que nous devons attribuer: il est tout à fait évidant qu’ avec les systèmes éducatifs différents, nous avons aussi des diapasons différents de notes. Et s’il faudrait attribuer une note à un étudiant qui travaille dans un système français, il est évidant que celle-ci puisse se rependre uniquement entre 0 et 20.

Le code ci-après nous montre comment avec la méthode «raise», nous pouvons empêcher d’effectuer la suite des calculs avec des données incompatibles (notes hors de l’intervalle de 0 à 20). Celui qui va introduire un chiffre qui ne se trouve pas dans l'intervalle de 0 à 20 verra le programme s'arrêter à mis chemin (juste après).

Résolution des exceptions avec l’ instruction «raise»

```
try:
    g = int(input("Introduire la note 'g':"))
    if (g >= 0) & (g <= 20):
        print("Note compatible au système français")
    else:
# À suivre
```

Suite du code du programme

```
        raise ValueError("Attention !!! g ne doit pas être > à 20 ni < à
                                                        0!!!")
    print("Vous pouviez poursuivre vos opérations avec cette note.")
    # ...
except ValueError as nom_err_raise_si:
    print("Votre note n'est pas compatible à la réglementation.")
    print(nom_err_raise_si)  # Permet d'imprimer le paramètre dans
                                        la méthode "raise"
```

```
# Résultat: (Variante 1 : si données compatibles)
Introduire la note 'g':12
Note compatible au système français
Vous pouviez poursuivre vos opérations avec cette note.
```

Dans le cas où la note va en dehors de l'intervalle, nous aurons arrêt du programme avec les informations suivantes:

```
# Résultat: (Variante 2 : si données non compatibles)
Introduire la note 'g':21
Votre note n'est pas compatible au règlement.
Attention 1 !!! g ne doit pas être > à 20 ni < à 0!!!
```

A quel niveau peut-on introduire la méthode «raise» dans notre calculatrice à quatre opérations de deux chiffres?

Nous l'avons déjà répondu au paragraphe précédent, lors du saisissement au clavier d'un nombre entier compris entre 1 à 4. Regardons cet exemple en exceptant la possibilité de reprendre l'introduction des données en cas d'erreur; l'erreur doit entraîner l'arrêt immédiat de l'exécution du code.

```
import time

début = time.time()
try:
    print("4 opérations dans la calculatrice de 2 nombres: +,/,*,-")
    x = float(input('Introduire un premier nombre x = '))
    y = float(input('Introduire un second  nombre y = '))
    answer = 'Attention!!! Il faut choisir entre 1 à 4 \n'
    print("1-addition; 2-division; 3-multiplication; 4-soustract")
    operation = input("Choisissez le type d'opération à faire : ")
    sign = operation.strip()
    if sign == "1":
        answer = f"{x}+{y} = {x+y}"
    elif sign == "2":
        answer = f"{x}/{y} = {x/y}"
    elif sign == "3":
        answer = f"{x}*{y} = {x*y}"
     elif sign == "4":
        answer = f"{x}–{y} = {x-y}"
     else:
        raise ValueError("La valeur introduite est incorrecte :
                                        respectez le diapason donné.")
except ValueError as nom_erreur_raise:
    print("La valeur entrée n'est pas valide ou celle qu'on attend.\n")
    print(nom_erreur_raise) # Affiche l'information qui est dans le
                            # paramètre d'exception ValueError
except ZeroDivisionError as nom_erreur_de_ZDE:
    print("Interdit de diviser par zéro (0) \n") # Traduction de l'erreur
                                                 # qui sera affichée
    print(nom_erreur_de_ZDE) # Affiche le type d'erreur de la base,
                             # car il n'a pas de "raise"
# À suivre
```

Suite du code du proramme

```
else:
    print(f"Résultat de l'opération: {answer}\n")
finally:
    fin = time.time()
    t_mis = fin - début
    print("La durée pour exécuter le programme était de %.3f
          secondes." % t_mis)
```

Les résultats possibles que nous pouvons obtenir avec différentes données

```
# Résultat: (Variante 1: cas où l'erreur fait appel à la méthode
                                                «raise»)
4 opérations possible dans la calculatrice de 2 nombres: +,/,*,-
Introduire un premier nombre x = 3
Introduire un second  nombre y = 2
1-addition; 2-division; 3-multiplication; 4-soustraction
Choisissez le type d'opération à faire : 5
La valeur entrée n'est pas valide ou celle qu'on attend.

La valeur introduite est incorrecte : respectez le diapason donné.
La durée pour exécuter le programme était de 11.651 secondes.
```

```
# Résultat: (Variante 2: cas où l'erreur passe par la méthode
                                                «except»)
4 opérations possible dans la calculatrice de 2 nombres: +,/,*,-
Introduire un premier nombre x = 4
Introduire un second  nombre y = 0
# À suivre
```

La suite du résultat

```
# Résultat: (Variante 2: cas où l'erreur passe par la méthode
                                                «except»)
1-addition; 2-division; 3-multiplication; 4-soustraction
Choisissez le type d'opération à faire : 2
Interdit de diviser par zéro (0)

float division by zero
La durée pour exécuter le programme était de 14.784 secondes.
```

```
# Résultat: (Variante 3: cas où l'erreur passe par la méthode
                                                «except»)
4 opérations possible dans la calculatrice de 2 nombres: +,/,*,-
Introduire un premier nombre x = c
La valeur entrée n'est pas valide ou celle qu'on attend.

could not convert string to float: 'c'
La durée pour exécuter le programme était de 4.111 secondes.
```

Dans les exemples précédents, nous avons eu des erreurs qui entraînaient directement l'interruption de l'exécution du code avec des données inexactes. Modifions le code de manière à ce qu'il puisse nous donner la possibilité de réintroduire les données sans relancer l'interpretation du code.

Pour y parvenir, nous allons utiliser les méthodes suivantes: «while», «continue» et «break».

```
import time

début = time.time()
while True:
    try:
        print("4 opérations dans la calculatrice de 2 nombres: +,/,*,–")
        x = float(input('Introduire un premier nombre x = '))
        y = float(input('Introduire un second  nombre y = '))
        answer = 'Attention!!! Il faut choisir entre 1 à 4 \n'
        while True:
            print("1-addition; 2-division; 3-multiplication; 4-soustract")
            operation = input("Choisissez le type d'opération à faire : ")
            sign = operation.strip()
            if sign == "1":
                answer = f"{x}+{y} = {x+y}"
            elif sign == "2":
                answer = f"{x}/{y} = {x/y}"
            elif sign == "3":
                answer = f"{x}*{y} = {x*y}"
            elif sign == "4":
                answer = f"{x}–{y} = {x-y}"
            else:
                print(answer)
            continue
            break
    except TypeError:
        print("Opération avec des types incompatibles entre eux \n")
    continue
    except ValueError:
        print("La valeur entrée n'est pas valide \n")
    continue                                    # ... À suivre…
```

Suite du code du programme

```
    except ZeroDivisionError:
        print("Interdit de diviser par zéro (0) \n")
        continue
    else:
        print(f"Résultat de l'opération: {answer}\n")
    finally:
        fin = time.time()
        t_mis = fin - début
        print("Durée pour exécuter le programme était de %.3f
              secondes" % t_mis)
    break
```

Les résultats possibles que nous pouvons obtenir avec différentes données

```
# Résultat: (Variante 1: cas où l'erreur fait appel à la méthode
                                        «raise»)
4 opérations dans la calculatrice de 2 nombres: +,/,*,-
Introduire un premier nombre x = 3
Introduire un second  nombre y = 2
1-addition; 2-division; 3-multiplication; 4-soustract
Choisissez le type d'opération à faire : 5
Attention!!! Il faut choisir entre 1 à 4

1-addition; 2-division; 3-multiplication; 4-soustract
Choisissez le type d'opération à faire :
```

```
# Résultat: (Variante 2: cas où l'erreur passe par la méthode
                                        «except»)
4 opérations dans la calculatrice de 2 nombres: +,/,*,-
Introduire un premier nombre x = 4
Introduire un second  nombre y = 0
1-addition; 2-division; 3-multiplication; 4-soustract
Choisissez le type d'opération à faire : 2
Interdit de diviser par zéro (0)

Durée pour exécuter le programme était de 7.861 secondes
4 opérations dans la calculatrice de 2 nombres: +,/,*,-
Introduire un premier nombre x =
```

```
# Résultat: (Variante 3: cas où l'erreur passe par la méthode
                                        «except»)
4 opérations dans la calculatrice de 2 nombres: +,/,*,-
Introduire un premier nombre x = c
La valeur entrée n'est pas valide

Durée pour exécuter le programme était de 4.373 secondes
4 opérations dans la calculatrice de 2 nombres: +,/,*,-
Introduire un premier nombre x =
```

```
# Résultat: (Variante 4: cas où pas d'erreur)
4 opérations dans la calculatrice de 2 nombres: +,/,*,-
Introduire un premier nombre x = 12
Introduire un second  nombre y = 3
1-addition; 2-division; 3-multiplication; 4-soustract
Choisissez le type d'opération à faire : 2
Résultat de l'opération: 12.0/3.0 = 4.0

Durée pour exécuter le programme était de 17.373 secondes
```

Exercice 4.1

Résoudre à nouveau tous les exercices précédents en ajoutant si possible, les instructions “*try*”, “*except*”, “*else*”, “*finally*” ou/et “*raise*”. Le but de la mise à jour consiste à rendre plus efficaces les différents programmes codes.

Suite aux pages 266 et 273

RU Глава V. ФУНКЦИИ
EN Chapter V. FUNCTIONS
FR Chapitre V. LES FONCTIONS

RU Введение

Отметим, что существует именно два вида функций: предопределенные функции и функции, созданные программистом. Чтобы лучше понять что это значит, определим, что такое функция в программировании.

Функция – это набор созданных операторов, которые объединяются в блок для выполнения определенной задачи столько раз, сколько требуется в программе. Поэтому, когда задача должна выполняться программой несколько раз, лучше ввести ее в функцию, чтобы программа была удобна в чтении, более систематична и более оптимизирована.

Продолжение на странице 267

EN Introduction

Note that there are precisely 2 types of functions: predefined functions and functions created by the programmer. To better understand this concept, let’s try to define the term “*function*” in programming.

A function is a set of instructions created in a block to perform a specific task, as many times as desired in a program. So, when a task needs to be performed several times by a program, it is better to introduce it into a function so that it (the program) can be convenient in its reading, more systematic and more optimized.

Continued on page 270

FR Introduction

Notons qu'il existe précisément 2 types de fonctions: les fonctions prédéfinies et les fonctions créées par le programmiste (développeur). Pour mieux comprendre, essayons de définir ce que c'est qu'une fonction en programmation.

Une fonction est un ensemble d'instructions créées constituées dans un bloc pour effectuer une tâche précise, autant de fois qu'on le souhaite dans un programme. Ainsi, lorsqu'une tâche doit être réalisée plusieurs fois par un programme, il est préférable de l'introduire dans une fonction pour qu'il (le programme) puisse être commode dans sa lecture, plus systématique et plus optimisé.

Suite à la page 273

RU V.1. Функции, созданные программистом

Синтаксис функции:

Функция объявляется от ключевого слова «***def***». После этого ключевого слова, можно ставить имя функции, с некоторыми параметрами.

```
def nom_fonction (параметры):
        Инструкция...
```

Пример

```
def add2number (number1, number2):
        suma = number1+number2
        return suma

или ещё
def add2number (number1, number2):
        return number1+number2
```

Это функция, которая суммирует два числа

Новая функция, созданная "add2number" имеет параметры "*number1*" и "*number2*", которые будут сложены. Результат будет зарегистрирован в переменной "*suma*" и возвращен через «return» туда, откуда функция была вызвана.

Как использовать эту функцию в программном коде?

В один файл добавим следующие инструкции:

```
x1 = float(input('Введите первое значение = '))
x2 = float(input(' Введите второе значение = '))

answer=add2number(x1,x2)
print('Результат = ', answer)
```

- Если мы не используем «return», чтобы вернуть любое значение, значение возвращаемой функции, несмотря на отсутствие ключевого слова «return», будет равно «None».
- Количество параметров в функции не ограничено.
- Можно использовать произвольное количество параметров, используя символ "*****" и слово, например: **argtupe* (можно заменить слово "*argtupe*" другим по выбору). Значения этих параметров вводятся "*tuple*". Аналогично с символом "******" и слово, например: ***argdico* (можно тоже заменить слово "*argdico*" другим по выбору), разница с предыдущим заключается в том, что здесь, необходимо использовать «*dictionary*» для сохранения значений параметров.

В качестве примера с «*tuple*» перепишем предыдущий пример:

```
def addnnumber (*argtuple):
    suma = 0
    for i in argtuple:
        suma +=i
    return suma

x1 = float(input('Введите первое значение = '))
x2 = float(input(' Введите второе значение = '))
answer=addnnumber(x1, x2)
print('Результат = ', answer)
```

При создании новой функции (“addnnumber”) количество параметров не ограничено. Таким образом, мы можем добавить третье значение без изменения кодов функции: addnnumber (x1, x2, x3) и т. д.

В качестве примера с “*dictionary*” перепишем предыдущий пример:

```
def addnnumberd (**argdico):
    suma = 0
    for i, j in argdico.items():
        suma +=j
        # print ('i= ', i)
    return suma

x1 = float(input('Введите первое значение = '))
x2 = float(input('Введите второе значение = '))

answer=addnnumberd(value1=x1, value2=x2)
print('Результат = ', answer)
```

С новой созданной функцией (“addnnumberd”) и вместе с размером параметров, который не ограничен, мы также можем манипулировать значениями ***i***, которые находятся в «*dictionary*».

Синтаксис «слово*» может использоваться иначе:**

```
def add3number (number1, number2, number3):
            suma = number1+number2+number3
return suma
# Продолжение следует
```

Продолжение кода программы

```
x1 = float(input('Введите первое значение = '))
x2 = float(input('Введите второе значение = '))
x3 = float(input('Введите третье значение = '))
les_valeurs = [x1, x2, x3]
answer=addnnumber(*les_valeurs)
print('Результат = ', answer)
```

Продолжение на странице 276

EN V.1. Functions created by the programmer

The syntax of a function:

The function is declared from the keyword "***def***". After this keyword, we find the name of the function with maybe some parameters.

```
def nom_fonction (parameters):
        Instructions...
```

Example

```
def add2number (number1, number2):
        suma = number1+number2
        return suma

or

def add2number (number1, number2):
        return number1+number2
```

A function created that adds 2 numbers

The new function "add2number" created has for parameters "number1" and "number2" that will be added. The result will be saved in "*suma*", and returned where the function was called by «return».

How to use this function in a code program?

In the same file, let's add the following instructions:

```
x1 = float(input(' Enter the first value = '))
x2 = float(input(' Enter the second value = '))

answer=add2number(x1,x2)
print('The result = ', answer)
```

- If we do not use «return», to return any value, the value of the function returned despite the absence of the keyword «return» will be equal to «*None*».
- The number of parameters in the function is not limited.
- One can use an arbitrary number of parameters using the symbol * and a word, for example: **argtupe* (you could replace the word "*argtupe*" with another of your choice). The values of these parameters are fed into a tuple. It is the same with the symbol ** and a word, for example: ***argdico* (you could replace the word "*argdico*" with another of your choice); the difference with the previous one is that we will use a "*Dictionary*" to keep the values in the parameters.

As an example with the «*tuple*», let's rewrite the previous example:

```
def addnnumber (*argtuple):
    suma = 0
    for i in argtuple:
        suma +=i
    return suma

x1 = float(input('Enter the first value = '))
x2 = float(input('Enter the second value '))
answer=addnnumber(x1, x2)
print('The result = ', answer)
```

With the newly created function "*addnnumber*", the number of parameters is not limited. So we can add a third value without changing the codes of the function: addnnumber (x1, x2, x3) etc.

As an example with the "dictionary", let's rewrite the previous example:

```
def addnnumberd (**argdico):
    suma = 0
    for i, j in argdico.items():
        suma +=j
        # print ('i= ', i)
    return suma

x1 = float(input('Enter the first value = '))
x2 = float(input('Enter the second value = '))

answer=addnnumberd(value1=x1, value2=x2)
print('The result = ', answer)
```

With the new function "addnnumberd" created and in addition to the number of parameters that is not limited, we can also manipulate the values of "*i*" that are in the dictionary.

The syntax «*word» can be used otherwise:

```
def add3number (number1, number2, number3):
            suma = number1+number2+number3
return suma

x1 = float(input('Enter the first value = '))
x2 = float(input('Enter the second value = '))
# To be continued
```

Continuation of the program code

```
x3 = float(input('Enter the third value = '))
les_valeurs = [x1, x2, x3]
answer=addnnumber(*les_valeurs)
print('The result = ', answer)
```

Continued on page 277

FR V.1. les fonctions créées par le développeur

La syntaxe d'une function:

La fonction se déclare à partir du mot clé «***def***». Après ce mot clé, nous retrouvons le nom de la fonction avec peut-être quelques paramètres.

```
def nom_fonction (paramètres):
        Instructions...
```

Exemple

```
def add2number (number1, number2):
        suma = number1+number2
        return suma

ou bien
def add2number (number1, number2):
        return number1+number2
```

C'est une fonction qui additionne deux nombres

La nouvelle fonction "add2number" créée a pour paramètres "number1" et "number2" qui seront additionnés. Le résultat va s'enregistrer dans "suma" et renvoyé là où la fonction a été appelée par «return».

Comment utiliser cette fonction dans un programme code ?

Dans un même fichier, ajoutons les instructions suivantes:

```
x1 = float(input('Introduire la première valeur = '))
x2 = float(input('Introduire la deuxième valeur = '))

answer=add2number(x1,x2)
print('Résultat = ', answer)
```

- Si nous n'utilisons pas «return», pour retourner une valeur quelconque, la valeur de la fonction retournée malgré l'absence du mot clé «return» sera égale à «None».
- Le nombre de paramètres dans la fonction n'est pas limité.
- On peut utiliser un nombre arbitraire de paramètres en utilisant le symbole * plus un mot, par exemple: **argtupe* (vous pouviez remplacer le mot "*argtupe*" par un autre de votre choix). Les valeurs de ces paramètres sont introduites dans un tuple. Il en est de même avec le symbole ** plus un mot, par exemple: ***argdico* (vous pouviez remplacer le mot "*argdico*" par un autre de votre choix); la différence avec le précédent est qu'on utilisera un «*dictionary*» pour garder les valeurs aux paramètres.

Comme exemple avec le «*tuple*», réécrivons l'exercice précédent:

```
def addnnumber (*argtuple):
    suma = 0
    for i in argtuple:
        suma +=i
    return suma

x1 = float(input('Introduire la première valeur = '))
# À suivre
```

Suite du code du programme

```
x2 = float(input('Introduire la deuxième valeur = '))
answer=addnnumber(x1, x2)
print('Résultat = ', answer)
```

Avec la nouvelle fonction “addnnumberd” créée, le nombre de paramètres n’est pas limité. Ainsi nous pouvons ajouter une troisième valeur sans changer les codes de la fonction: addnnumber(x1, x2, x3) etc.

Comme exemple avec le dictionnaire, réécrivons l’exercice précédent:

```
def addnnumberd (**argdico):
    suma = 0
    for i, j in argdico.items():
        suma +=j
        # print ('i= ', i)
    return suma

x1 = float(input('Introduire la première valeur = '))
x2 = float(input('Introduire la deuxième valeur = '))

answer=addnnumberd(value1=x1, value2=x2)
print('Résultat = ', answer)
```

Avec la nouvelle fonction “addnnumberd” créée et en plus du nombre de paramètres qui n’est pas limité, nous pouvons également manipuler les valeurs de ***i*** qui se trouvent dans le dictionnaire.

La syntaxe «*mot» peut être utilisée autrement:

```
def add3number (number1, number2, number3):
                suma = number1+number2+number3
return suma

x1 = float(input('Introduire la première valeur = '))
x2 = float(input('Introduire la deuxième valeur = '))
x3 = float(input('Introduire la troisième valeur = '))
les_valeurs = [x1, x2, x3]
answer=addnnumber(*les_valeurs)
print('Résultat = ', answer)
```

Suite à la page 278

RU V.2. Как вернуть несколько значений с помощью «*return*»

Это возможно только через «*list*» или «*tuple*». Используя «*return*», возвращаемые значения разделяются запятой.

Рассмотрим следующий пример: предположим, что наша функция должна вернуть нам сумму и умножение двух введенных чисел, следовательно мы должны иметь следующий код:

```
def add_multi(number1, number2):
                suma = number1+number2
                multi= number1*number2
return suma, multi

x1 = float(input('Введите первое значение = '))
x2 = float(input('Введите второе значение = '))

answer=add_multi(x1, x2)
print('Результат = ', answer)
```

```
# Результат выполнения функции add_multi
Введите первое значение = 4
Введите второе значение = 5
Результат =  (9.0, 20.0)
```

Функция также может вызываться сама во время её выполнения. Это **рекурсия**.

Одна функция может выполнять другую функцию.

Упражнение №4 с решением

Напишите функцию, которая вычисляет факториал заданного числа.

Решение находится в главе VI

Продолжение на странице 279

EN V.2. How to return multiple values with «return»

It is possible only through lists or tuples. Using «return», the values to be returned are separated by a comma.

Let's look at this example Consider that our function must return us the sum and multiplication of the 2 numbers entered, so we will have the following code:

```
def add_multi (number1, number2):
                suma = number1+number2
                multi= number1*number2
return suma, multi

x1 = float(input('Enter the first value = '))
x2 = float(input('Enter the second value = '))

answer=add_multi(x1, x2)
print('The result = ', answer)
```

```
# Result of execution of add_multi function
Enter the first value = 4
Enter the second value = 5
The result =  (9.0, 20.0)
```

A function can also be called itself in its execution: it is **recursion**.

One function can perform another function.

Exercise 5.1 with a solution

Write a function that calculates the factorial of any given number.

The resolution can be found in Chapter VI

Continued on page 280

FR V.2. Comment retourner plusieurs valeurs avec «return»

Il est possible uniquement à travers les listes ou tuples. En utilisant «return», les valeurs à retourner sont séparées par une virgule.

Regardons cet exemple

Considérons que notre fonction doit nous renvoyer la somme et la multiplication des 2 nombres entrés, ainsi nous aurons le code suivant:

```
def add_multi(number1, number2):
                suma = number1+number2
                multi= number1*number2
return suma, multi

x1 = float(input('Introduire la première valeur = '))
x2 = float(input('Introduire la deuxième valeur = '))

answer=add_multi(x1, x2)
print('Résultat = ', answer)
```

```
# Résultat de l'exécution de la fonction add_multi
Introduire la première valeur = 4
Introduire la deuxième valeur = 5
Résultat =  (9.0, 20.0)
```

Une fonction peut également s'appeler elle même dans son exécution : c'est **la récursivité.**

Une fonction peut exécuter une autre fonction.

Exercice 5.1 corrigé

Écrire une fonction qui calcule la factorielle d'un nombre donné.

Voir la résolution au chapitre VI

Suite à la page 281

RU V.3. Иллюстративный пример роли функции

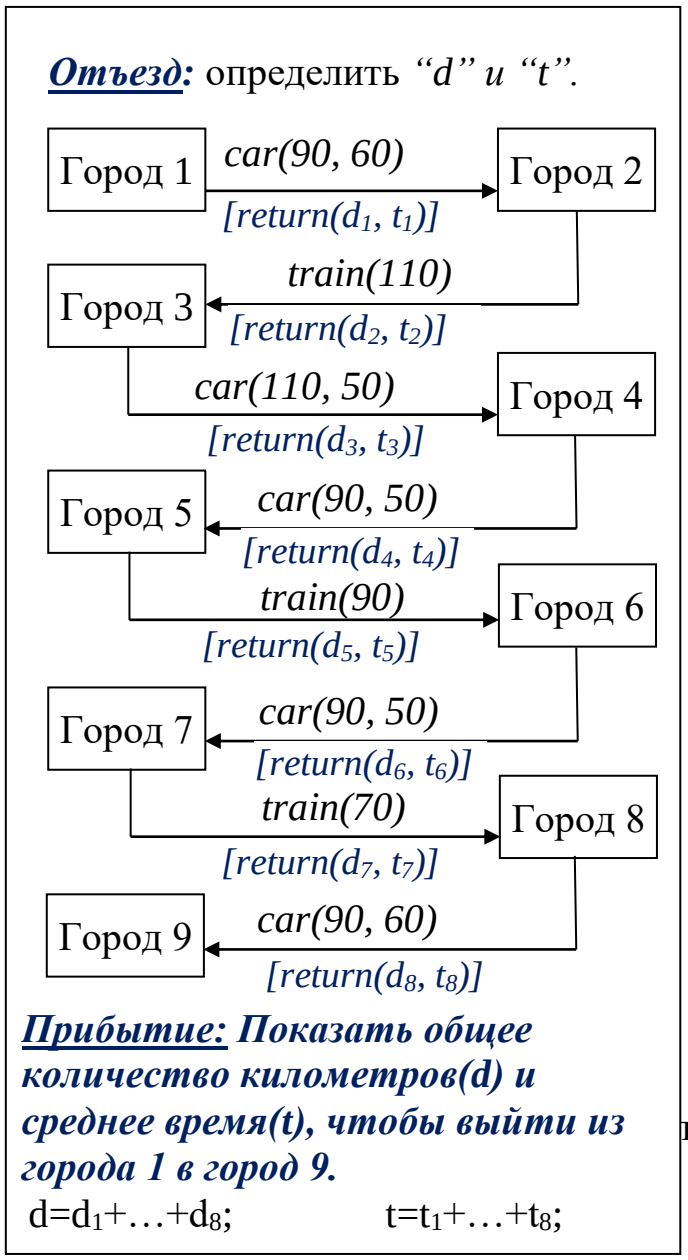
Рассмотрим конкретный пример, где задача (программа) для человека состоит в том, чтобы посетить 9 городов в одной стране с использованием различных видов транспорта. Эти средства передвижения имеют разные скорости функционирования. Таким образом, рассмотрим средства передвижения как функции, и каждая функция может иметь один или два параметра.

Имеются следующие синтаксисы:

автомобиль(параметры) и поезд(параметр)

car(p_i, p_i) и train(p_i)

Параметр здесь представляется собой скорость движения, которая варьируется в зависимости от региона. В виде схемы мы можем проиллюстрировать её таким образом (см. рис. 5.1):



car(p_1, p_2)
{
– *определить d;*
– *определить t;*
– *возвращать d и t в программу.*
}

train(p_3)
{
– *определить d;*
– *определить t;*
– *возвращать d и t в программу.*
}

return(d_i, t_i) – *это оператор, который в основной программе возвращает значения параметров* d_i *и* t_i. *i – являющийся индексом, представляющим любые 2 города.*

Рис. 5.1 Иллюстрация примера локомоции

Продолжение на странице 283

EN V.3. Illustrative example of the role of a function

Consider this concrete example where the exercise (program) for a person is to visit 9 cities in a country with different means of transport. These means of locomotion use different speeds to operate. Thus, consider

the means of locomotion as functions and each function can have one or two parameters.

We have the following syntaxes:

car(parameters) and ***train(parameter)***.

The parameter here is the speed of circulation that varies depending on the regions. In the form of a diagram, we can illustrate it in this way (see figure 5.1):

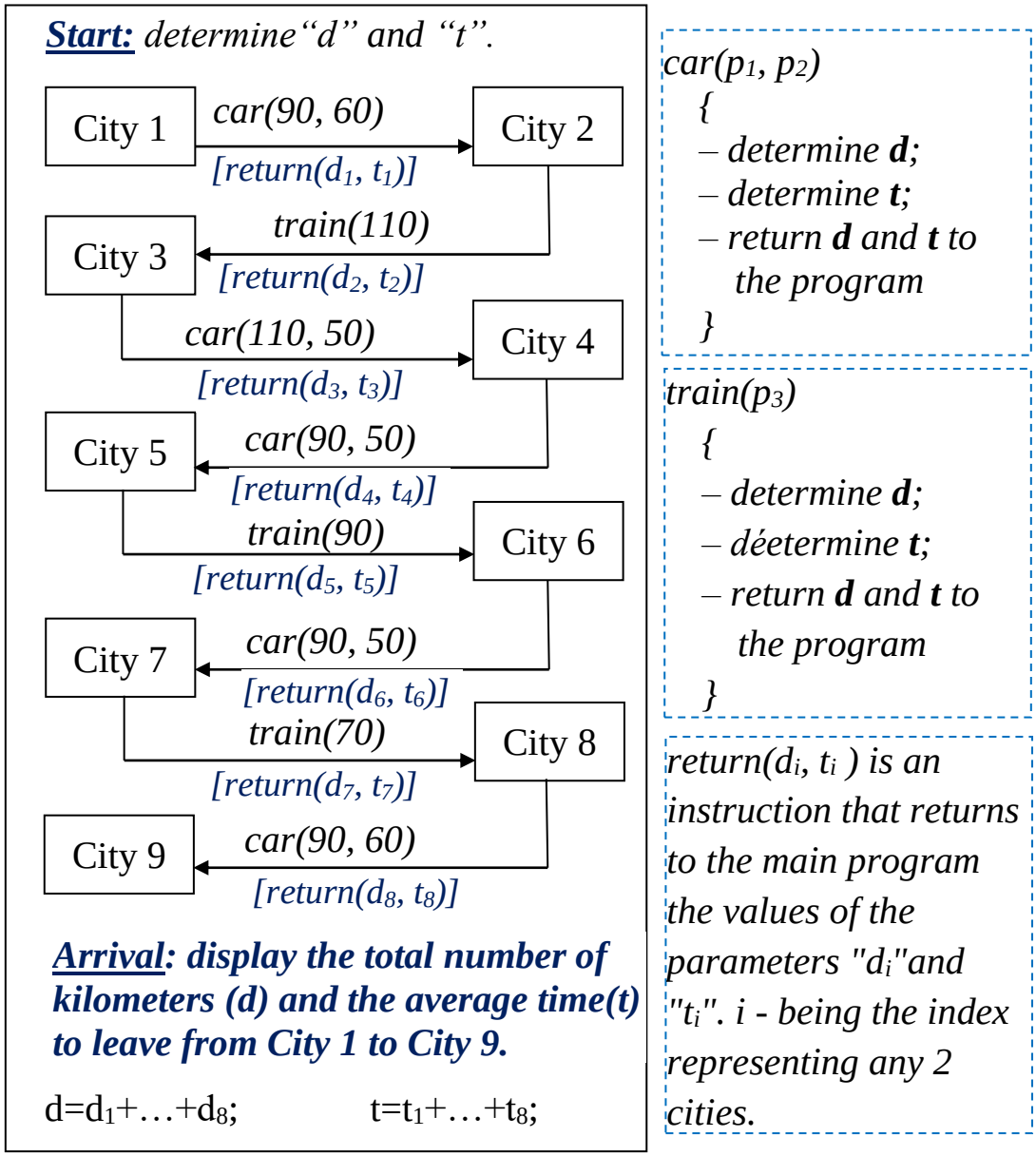


Fig. 5.1 – Illustration of the example on locomotion

Continued on page 284

FR V.3. Exemple illustratif du rôle d'une fonction

Considérons cet exemple concret où l'exercice (programme) pour l'Homme consiste à visiter 9 villes dans un pays avec des moyens de transports différents. Dans la figure 5.1, nous avons les différents parcours possibles.

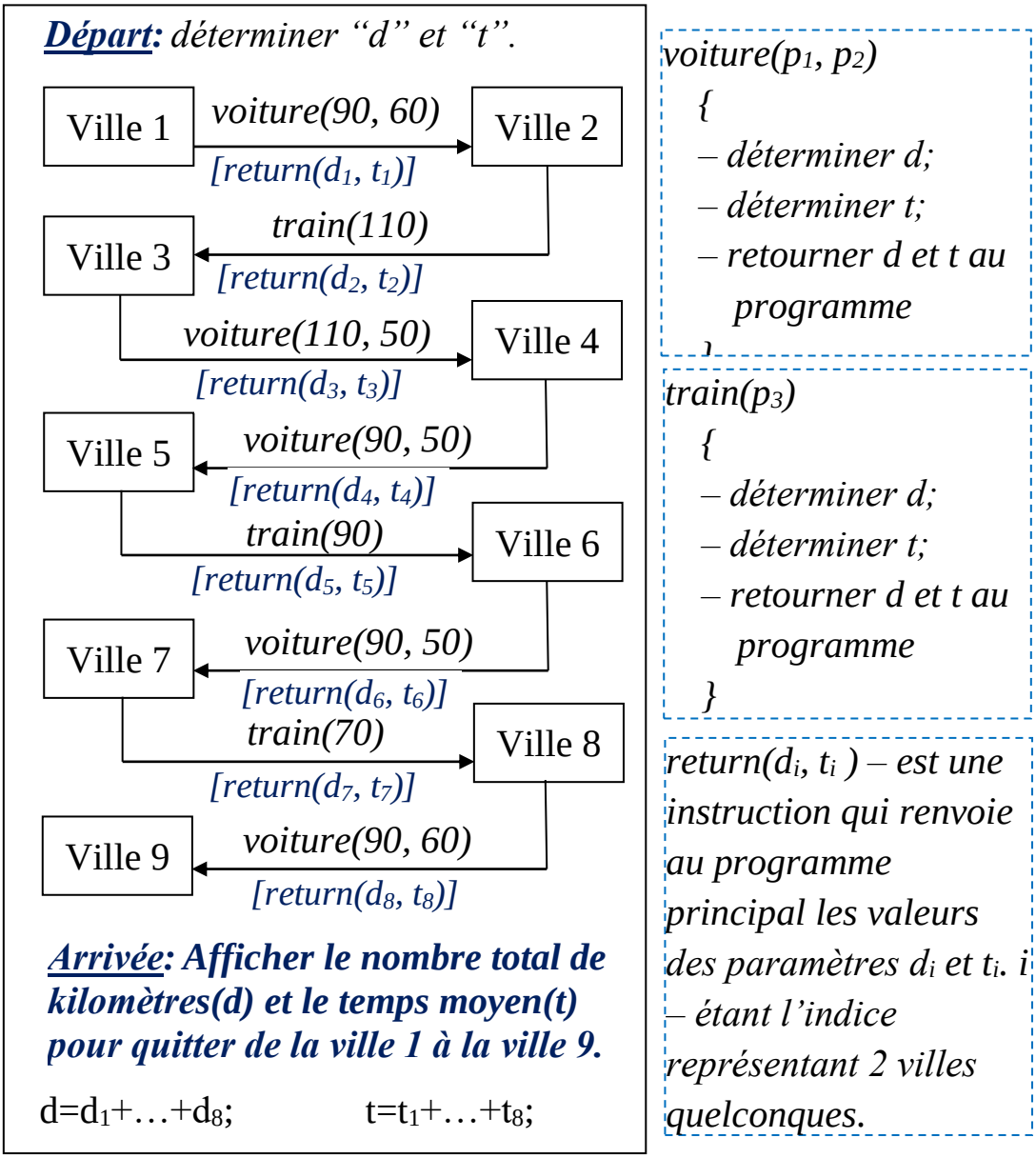


Fig. 5.1 – Illustration de l'exemple sur la locomotion

Ces moyens de locomotion (voiture et train) utilisent des vitesses différentes pour fonctionner. Ainsi, considérons les moyens de locomotion comme des fonctions et chaque fonction peut avoir un ou deux paramètres.

Nous avons les syntaxes suivantes:

voiture(paramètres) et ***train(paramètre)***.

voiture (p_i, p_j) ***u train*** (p_j)

Notons que le paramètre ici est la vitesse de circulation qui varie en fonction des régions.

Suite à la page 284

RU V.4. Дополнительная информация о функциях и процедурах

Функция представляет собой набор операторов, которые через имя переходят на выполнение инструкций, которые там находятся.

В сценарии сначала нужно написать функцию или функции, а затем основную программу.

Значения или аргументы, заданные при вызове функции, должны редактироваться в том же порядке, что и параметры, соответствующие им в определении функции.

Функция Python не обязательно возвращает результат основной программе с помощью оператора «return», в этом случае она будет просто манипулировать входными данными.

Функция, которая не возвращает данные, называется в программировании ***процедурой***. В Python, даже если нет данных для возврата в функцию, она все равно называется функцией, поскольку она все равно возвращает значение "None", которое означает, что ничего не было возвращено.

Код, следующий за оператором «return» в функции, никогда не будет прочитан или выполнен, поэтому этот оператор всегда помещается в конец функции.

Продолжение на странице 285

EN V.4. Other information on functions and procedures

A function is a sequence of statements that is called with a name to execute the statements that are found.

In a script, one must first write the function(s) and after the main program.

The values or arguments provided when calling a function must be edited in exactly the same order as the corresponding parameters in the function definition.

A Python function does not necessarily return a result to the main program by the «*return*» statement, in which case it will just manipulate the input data.

A function that does not return data is called in programming *a procedure*. In Python, even when there is no data to return in a function, it is still called *a function* since it still returns the value «*None*» which means that nothing has been returned.

Code following a «*return*» statement in a function will never be read or executed, which is why this statement is always placed at the end of the function.

Continued on page 289

FR V.4. Autres informations sur les fonctions et procédures

Une fonction est une suite d'instructions que l'on appelle avec un nom pour exécuter les instructions qui se trouvent.

Dans un script, il faut d'abord écrire la ou les fonctions et après le ou les programmes principaux.

Les valeurs ou arguments qu'on fournit lors de l'appel d'une fonction doivent être édités exactement dans le même ordre que celui des paramètres qui leur correspondent dans la définition de la fonction.

Une fonction Python ne renvoie pas obligatoirement de résultat au programme principal par l'instruction "return", dans ce cas, elle manipulera juste les données en entrée.

Une fonction qui ne renvoie pas de données est appelée en programmation une procédure. En Python, même quand il n'y a pas de données à renvoyer dans une fonction, celle-ci est tout de même appelée fonction dans la mesure où elle renvoie tout de même la valeur «None» qui veut dire que rien n'a été renvoyé.

Le code suivant une instruction «return» dans une fonction ne sera jamais lu ni exécuté, c'est pourquoi cette instruction est toujours placée à la fin de la fonction.

Suite à la page 292

RU V.5. Встроенные или предопределенные функции в Python

Можно сказать, что встроенные функции – это функции, которые уже созданы Python и доступны программистам. Они всегда доступны и используются в различных сценариях (см. таб. 5.1).

Таблица 5.1 – Встроенные функции

Встроенные или предопределенные функции				
abs()	delattr()	hash()	memoryview()	set()
all()	dict()	help()	min()	setattr()
any()	dir()	hex()	next()	slice()
ascii()	divmod()	id()	object()	sorted()
bin()	enumerate()	input()	oct()	staticmethod()
bool()	eval()	int()	open()	str()
breakpoint()	exec()	isinstance()	ord()	sum()
bytearray()	filter()	issubclass()	pow()	super()
bytes()	float()	iter()	print()	tuple()
callable()	format()	len()	property()	type()
chr()	frozenset()	list()	range()	vars()
classmethod()	getattr()	locals()	repr()	zip()
compile()	globals()	map()	reversed()	__import__()

Продолжение таблицы 5.1

Встроенные или предопределенные функции				
complex()	hasattr()	max()	round()	

Дополнительные сведения о ролях каждой функции смотри по следующей ссылке URL:

https://docs.python.org/3/library/functions.html.

Следует отметить, что некоторые из этих функций были определены выше.

Для дополнения раздела, рассмотрим некоторые из этих функций.

Функция locals() – возвращает различные переменные (необходимую информацию) из текущего файла в виде словаря (Dictionary).

Пример

Создадим файл «***prog_1.py***» и оставим его без кода (т.е. пустым), затем используем различные переменные (локальная информация), которые находятся в таблице 5.2.

Таблица 5.2 – Переменные и значения функции *locals()*

Имя переменных	***Соответствующие значения переменных***
'__name__'	*'__main__'*
'__doc__'	*None*
'__package__'	*None*
'__loader__'	*<class '_frozen_importlib.BuiltinImporter'>*
'__spec__'	*None*
'__annotations__'	*{}*
'__builtins__'	*<module 'builtins' (built-in)>*
'__file__'	*'Put_k_faylu/prog_1.py'*
...	...

Итак, если мы добавим данные в файл, то они будут добавлены в эту таблицу. Вся эта информация устанавливается в функции «*locals()*».

В «***prog_1.py***» добавим следующий код:

```
var_locale_file = locals()
print(var_locale_file)
```

Результат выполнения этого кода выглядит следующим образом:

```
{'__name__': '__main__', '__doc__': None, '__package__': None,
'__loader__': <_frozen_importlib_external.SourceFileLoader object at
0x00E4AF58>, '__spec__': None, '__annotations__': {},
'__builtins__': <module 'builtins' (built-in)>, '__file__':
'Put_k_faylu/prog_1.py', '__cached__': None, 'var_locale_file': {...}}
```

Эта информация может быть полезна в продолжении программы кода. Например, мы можем найти имя папки, в которой находится файл; эта информация находится в переменной «__file__».:

```
File_path=__file__
print('Значение переменной __file__ является: ', File_path)
```

Результат выполнения этого кода выглядит следующим образом:

```
Значение переменной __file__ является: Put_k_faylu \prog_1.py
```

Функция *globals()* – возвращает необходимую глобальную информацию (переменную) из текущего файла в виде “Dictionary”.

Обратите внимание, что на уровне модуля, *locals()* и *globals()* являются одним и тем же словарем “Dictionary”. То есть информация из предшествующей таблицы такая же, как и в функции «*globals()*».

Функция *dir*

эта функция может использоваться в двух случаях:

▪ для отображения списка всех локальных переменных без значений. Здесь мы используем её без аргументов, это также эквивалентно функции *locals()*. Чтобы увидеть этот список, напишем, например, в файл « **.py* » без кода это:

print(*dir*())

Мы будем иметь следующий ответ:

```
['__annotations__', '__builtins__', '__doc__', '__file__',
'__loader__', '__name__', '__package__', '__spec__']
```

▪ мы также можем отобразить список всех атрибутов определенного объекта (например, объект может быть модулем, типом или классом). Синтаксис следующий: *dir*([*objet*]).

Если имеем следующие значения *«x=3.5»* et *«y= 'слово'»*, то получим результат функции dir(x), который отличается от результата функции dir(y). Ибо *x* – вещественное число, а *y* – строковое значение.

Функция *vars*

Эта функция возвращает различные переменные (необходимую информацию) из текущего файла в виде словаря «Dictionary» с их значениями, такими как функция *locals()*. С аргументом «*vars*([*object*])», она возвращает атрибут ***__dict__*** модуля, класса, экземпляра или любого объекта с атрибутом ***__dict__***.

Функция ***__import__()*** – не путать с «*import*». Эта функция вызывается с помощью инструкции ***import***. Функция импортирует модуль *name*, потенциально используя *globals* и *locals*, чтобы

определить, как интерпретировать имя в контексте пакета. Отсюда следующий синтаксис:

__import__(name, globals=None, locals=None, fromlist=(), level=0).

Продолжение на странице 296

EN V.5. Built-in or predefined functions in Python

We can say that built-in functions are functions already created and made available to programmers by Python. They are always available and are used in different scripts (see table 5.1).

Table 5.1 – Built-in Fonctions

Built-in or predefined functions				
abs()	delattr()	hash()	memoryview()	set()
all()	dict()	help()	min()	setattr()
any()	dir()	hex()	next()	slice()
ascii()	divmod()	id()	object()	sorted()
bin()	enumerate()	input()	oct()	staticmethod()
bool()	eval()	int()	open()	str()
breakpoint()	exec()	isinstance()	ord()	sum()
bytearray()	filter()	issubclass()	pow()	super()
bytes()	float()	iter()	print()	tuple()
callable()	format()	len()	property()	type()
chr()	frozenset()	list()	range()	vars()
classmethod()	getattr()	locals()	repr()	zip()
compile()	globals()	map()	reversed()	__import__()
complex()	hasattr()	max()	round()	

For more information on the roles of each function, see the following links URL: *https://docs.python.org/3/library/functions.html*. It should be noted that some of these functions have been defined somewhat earlier in other parts.

To complete the section, let's try to study some of these functions:

Function locals() – it returns the various variables (necessary information) of the current file as a Dictionary.

Example

Let's create a file «***prog_1.py***» and leave it without code, then the different variables (local information) that are in it are in this table 5.2.

Table 5.2 – Variables and values of the function *locals()*

Variable name	***The respective values of the variables***
'__name__'	*'__main__'*
'__doc__'	*None*
'__package__'	*None*
'__loader__'	*<class '_frozen_importlib.BuiltinImporter'>*
'__spec__'	*None*
'__annotations__'	*{}*
'__builtins__'	*<module 'builtins' (built-in)>*
'__file__'	*'The_path_to_the_file/prog_1.py'*
...	...

So if we add one or more information to the file, then that one or more will be added to this table. All this information is fixed in the function «*locals()*».

In «***prog_1.py***» let's add the following code:

```
var_locale_file = locals()
print(var_locale_file)
```

The result of running this code is as follows:

```
{'__name__': '__main__', '__doc__': None, '__package__': None,
'__loader__': <_frozen_importlib_external.SourceFileLoader object at
0x00E4AF58>, '__spec__': None, '__annotations__': {},
'__builtins__': <module 'builtins' (built-in)>, '__file__':
'The_path_to_the_file/prog_1.py', '__cached__': None,
'var_locale_file': {...}}
```

This information can be useful in the continuation of the program. For example, we can look for the name of the folder where the file is located; this information is in the variable « *__file__* ». Thus, we will write:

```
File_path=__file__
print('La valeur de __file__ est:',File_path)
```

The result of running this code is as follows:

```
The value of __file__ is: The_path_to_the_file\prog_1.py
```

Globals() function – it returns the various necessary global information (variables) of the current file as a *Dictionary*.

Note that at a module level, *locals()* and *globals()* are the same *Dictionary*. That is, the information in the preceding table is the same as in the «*globals()*» function.

Function dir

This function can be used in two cases:

▪ we can use it to display the list of all local variables without values; here we use it without arguments, it is also the equivalent of the *locals()* function. To see this list, let's write for example in a file « ****.py*** » without code this:

print(*dir*())

We will have as answer:

```
['__annotations__', '__builtins__', '__doc__', '__file__',
'__loader__', '__name__', '__package__', '__spec__']
```

- we can also display the list of all attributes of a specific object (the object can be a module, a type or a class for example). The syntax is: *dir*([*objet*]).

If we have *«x=3.5 »* et *« y= 'un mot' »*, then we have the result of the ***dir(x)*** function which is different from the result of the ***dir****(y)* function. Because ***x*** is a real number, and ***y*** is a string.

Function vars

This function returns the various variables (necessary information) of the current file as a dictionary with their values as the function ***locals()***. With « *vars*([*object*]) » argument, it returns the ***__dict__*** attribute of a module, class, instance, or any object with a ***__dict__*** attribute__.

Fonction __import__() – do not confuse with import. This function is invoked via the import statement. The function imports the name module, potentially using globals and locals to determine how to interpret the name in the context of a package. Hence the following syntax:

__import__(name, globals=None, locals=None, fromlist=(), level=0).

Continued on page 297

FR V.5. Les fonctions natives ou prédéfinies dans Python

Nous pouvons dire que les fonctions natives sont des fonctions déjà créées et mises à la disposition des programmistes par Python. Elles sont toujours disponibles et sont utilisées dans différents scripts (voir table 5.1).

Table 5.1 – Les fonctions natives

Les fonctions natives ou prédéfinies				
abs()	delattr()	hash()	memoryview()	set()
all()	dict()	help()	min()	setattr()
any()	dir()	hex()	next()	slice()
ascii()	divmod()	id()	object()	sorted()
bin()	enumerate()	input()	oct()	staticmethod()
bool()	eval()	int()	open()	str()
breakpoint()	exec()	isinstance()	ord()	sum()
bytearray()	filter()	issubclass()	pow()	super()
bytes()	float()	iter()	print()	tuple()
callable()	format()	len()	property()	type()
chr()	frozenset()	list()	range()	vars()
classmethod()	getattr()	locals()	repr()	zip()
compile()	globals()	map()	reversed()	__import__()
complex()	hasattr()	max()	round()	

Pour plus d'informations sur les rôles de chaque fonction, voir les liens suivants *[URL: https://docs.python.org/fr/3/library/functions.html]*. Il faut retenir que certaines de ces fonctions ont été définies un peu plus haut dans d'autres parties.

Pour compléter la section, essayons d'étudier quelques une de ces fonctions:

Fonction locals() – elle renvoie les différentes variables (informations nécessaires) du fichier en cours sous forme de dictionnaire.

Exemple.

Créons un fichier «***prog_1.py***» et laissons le sans code, alors les différentes variables (informations locales) qui s'y trouvent sont dans ce tableau 5.2.

Table 5.2 – Variables et valeurs de la fonction *locals()*

Nom des variables	***Les valeurs respectives des variables***
'__name__'	*'__main__'*
'__doc__'	*None*
'__package__'	*None*
'__loader__'	*<class '_frozen_importlib.BuiltinImporter'>*
'__spec__'	*None*
'__annotations__'	*{}*
'__builtins__'	*<module 'builtins' (built-in)>*
'__file__'	*'Le_chemin_d_accès_au_fichier/prog_1.py'*
...	...

Alors si nous ajoutons une ou des informations dans le fichier, alors celle-ci ou celles-ci vont s'ajouter dans ce tableau. L'ensemble de ces informations sont fixées dans la fonction « *locals()* ».

Dans «***prog_1.py***» ajoutons le code suivant:

```
var_locale_file = locals()
print(var_locale_file)
```

Le résultat de l'exécution de ce code est le suivant:

```
{'__name__': '__main__', '__doc__': None, '__package__': None,
'__loader__': <_frozen_importlib_external.SourceFileLoader object at
0x00E4AF58>, '__spec__': None, '__annotations__': {},
'__builtins__': <module 'builtins' (built-in)>, '__file__':
'Le_chemin_d_accès_au_fichier/prog_1.py', '__cached__': None,
'var_locale_file': {...}}
```

Ces informations peuvent être utiles dans la suite du programme. Nous pouvons par exemple chercher à savoir le nom du dossier où se

trouve le fichier; cette information se trouve dans la variable « *__file__* ». Ainsi, nous ecrirons:

```
File_path=__file__
print('La valeur de __file__ est:',File_path)
```

Le résultat de l'exécution de ce code est le suivant:

```
La valeur de __file__ est: Le_chemin_d_accès_au_fichier \prog_1.py
```

Fonction globals() – elle renvoie les différentes informations globales (variables) nécessaires du fichier en cours sous forme de dictionnaire.

Notez qu'au niveau d'un module, locals() et globals() sont le même dictionnaire. C'est-à-dire que les informations du tableau précédant sont les mêmes que dans la fonction «*globals()*».

Fonction dir

Cette fonction peut être utilisée dans deux cas:

- nous pouvons l'utiliser pour afficher la liste de toutes les variables locales sans valeurs; Ici nous l'utilisons sans argument, c'est également l'équivalent de la fonction *locals()*. Pour voir cette liste, écrivons par exemple dans un fichier « ***.py*** » sans code ceci:

print(*dir*())

Nous aurons comme réponse:

```
['__annotations__', '__builtins__', '__doc__', '__file__',
'__loader__', '__name__', '__package__', '__spec__']
```

- nous pouvons également afficher la liste de tous les attributs d'un objet spécifique (L'objet peut être un module, un type ou une classe par exemple). La syntaxe est la suivante: *dir*([*objet*]).

Si nous avons *«x=3.5 »* et *« y= 'un mot' »*, alors nous avons le

résultat de la fonction ***dir(x)*** qui est différent du résultat de la fonction ***dir(y)***. Car ***x*** est un nombre réel et ***y*** – une chaîne.

Fonction vars

Cette fonction elle renvoie les différentes variables (informations nécessaires) du fichier en cours sous forme de dictionnaire avec leurs valeurs comme la fonction ***locals()***. Avec argument « *vars*([*object*]) », elle renvoie l'attribut ***__dict__*** d'un module, d'une classe, d'une instance ou de n'importe quel objet avec un attribut ***__dict__***.

Fonction __import__() – Ne pas confondre à avec ***import***. Cette fonction est invoquée via l'instruction ***import***. La fonction importe le module name, utilisant potentiellement globals et locals pour déterminer comment interpréter le nom dans le contexte d'un paquet. D'où la syntaxe suivante:

__import__(name, globals=None, locals=None, fromlist=(), level=0).

Suite à la page 298

RU V.6. Как работает конструкция «*import*»

Процесс функционирования инструкции "*import*" в схематическом виде (см. рис. 5.2).

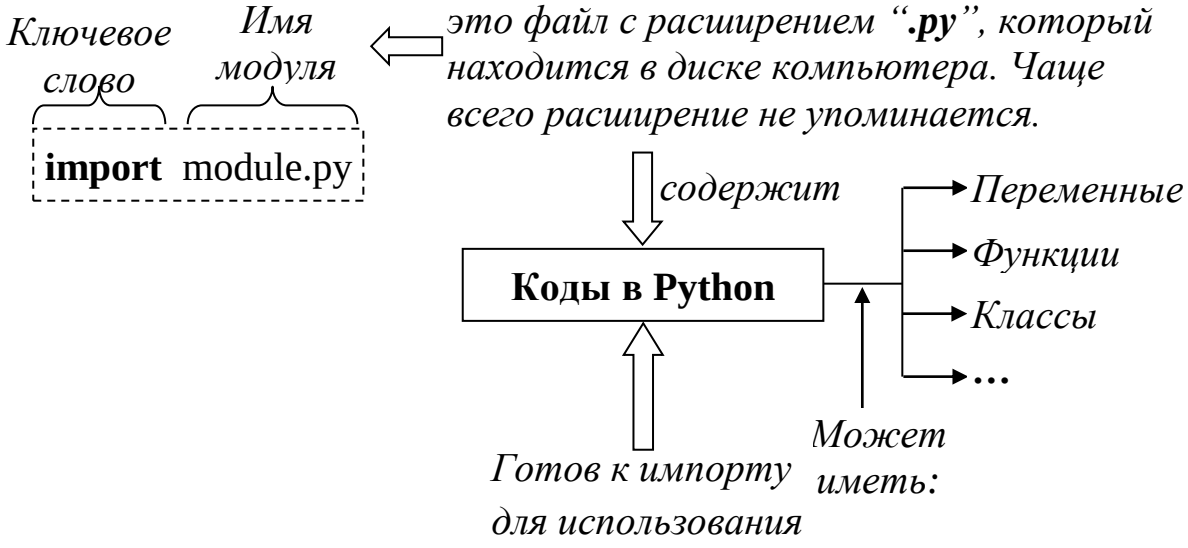


Рис. 5.2 Схематическая операция инструкции "*import*"

Модуль в Python – это файл, состоящий из кода Python, который можно вызвать и использовать без необходимости переписывать его в новый код.

Дополнительная информация, которая необходимо знать:

Прямой вызов функции "*__import__()*" выполняет только поиск модуля. Механизм импорта, который выполняет "*import*", также может выполняться функциями:

importlib.import_module() и *__import__()*

При выполнении инструкции "*import*" мы становимся свидетелями вызова встроенной функции "__import__()".

Модуль "*importlib*" предоставляет богатый API (*интерфейс прикладного программирования (Application Programming Interface) – это описание того, как компьютерная программа может взаимодействовать с другой программой*) для взаимодействия с системой импорта.

Продолжение на странице 299

EN V.6. How the "*import*" instruction works

The process of functioning instruction "*import*" in a schematic form (see Fig. 5.2).

A module in Python is a file consisting of Python code that can be called and used without having to rewrite it into new code.

Additional information to know:

A direct call to the "*__import__()*" function only performs the module search.

The import mechanism that «*import*» performs can also be performed by the functions: *importlib.import_module()* and *__ import__()*

When an import statement is executed, we witness a call to the built-in function "*__import__()*".

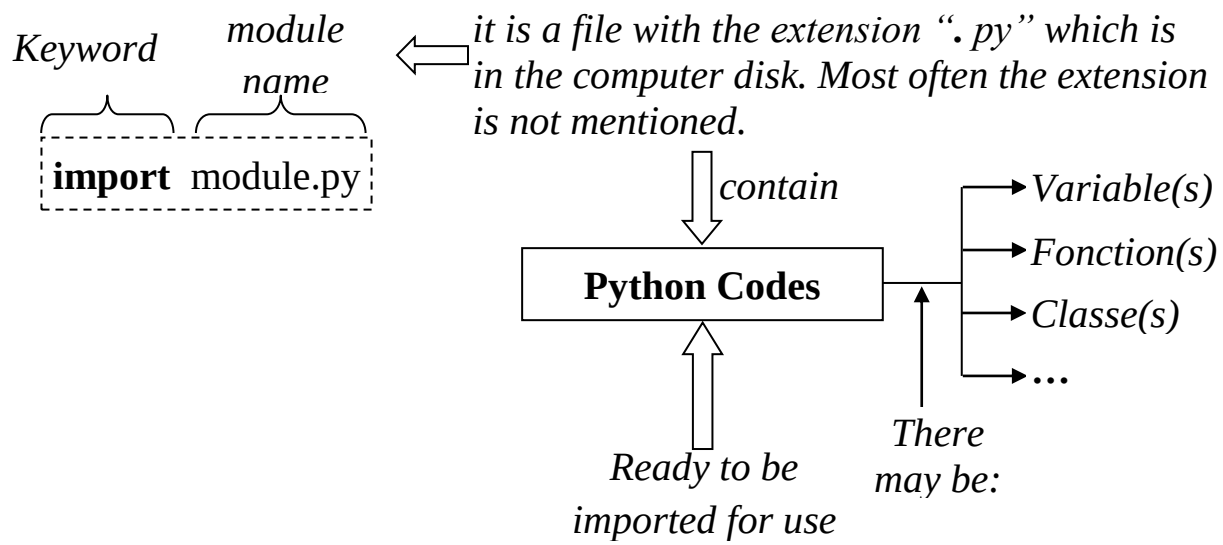


Fig. 5.2 Schematic form of the operation of “*import*”

The “*importlib*” module provides a rich API (*Application Programming Interface – this is a description of how a computer program can interact with another program*) to interact with the import system.

Continued on page 302

FR V.6. Fonctionnement de l’instruction “*import*”

Le fonctionnement le l’instruction «import» en forme schématique (voir fig. 5.2).

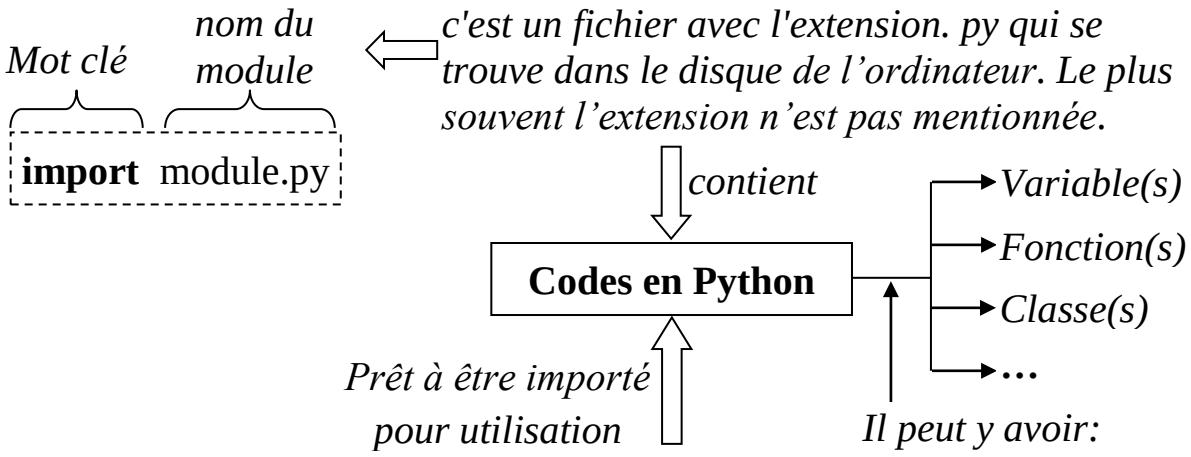


Fig. 5.2 Forme schématique du fonctionnement de «*import*»

Un module en Python est un fichier constitué de code Python qu'on peut appeler et utiliser sans avoir besoin de le recopier dans un nouveau code.

Autres informations à savoir:

Un appel direct à la fonction *__import__()* effectue seulement la recherche du module.

Le mécanisme d'importation qu'effectue *import* peut également être effectué par les fonctions: ***importlib.import_module()*** et ***__import__()***

Lorsqu'une instruction *import* est exécutée, nous assistons à un appel de la fonction native *__import__()*.

Le module ***importlib*** fournit une **API** (*Application Programming Interface - il s'agit d'une description de la manière par laquelle un programme informatique peut interagir avec un autre programme*) riche pour interagir avec le système d'import.

Suite à la page 304

RU V.7. Как создать модуль в Python

Начнем с этого упражнения, чтобы понять как это происходит: создайте модуль, который позволит определить, является ли число четным (или отрицательным).

***Шаг 1*:** создайте файл с расширением "*py*" для импорта, необходимо назать его так: «my_module.py"

***Шаг 2*:** ввести в этот файл коды, которые выполняют одну или несколько четко определенных задач. Эти коды предпочтений могут быть в разных функциях. В нашем случае мы будем иметь две функции:

 # С первой функцией мы определяем, является ли число четным или нет.

```
def parite(y):
    while y >= 2:
        y = y-2
    if y == 1:
        res = 'не четное'
    else:
        res = 'четное'
    return res
```

Новая функция *parite()* имеет входной параметр, который *y*

Возвращает значение пер.“res” по «return» там, где оно было вызвано

 # Мы можем добавить в модуль еще одну функцию, которая будет определять максимум между двумя числами.

```
def my_max(x1, x2):
    if x1>x2:
        res = x1
    else:
        res = x2
return res
```

Новая функция *my_max()* имеет два входных параметра: *x1 и x2*.

Возвращает значение пер.“res” по «return» там, где оно было вызвано

Шаг 3: Написать небольшую программу в коде Python для тестирования этого модуля. Необходимо дать этому файлу любое имя: «*my_code.py*». Оба файла могут находиться в одном каталоге или в разных каталогах при условии указания в файле «*my_code.py*», путь к файлу «*my_module.py*» при его импорте.

Решение № 1 Импортировать всего модуля

```
# Теперь можно использовать функции модуля (parite и
my_max) :
from my_module import *

x1 = float(input('Введите любое число = '))
# Продолжение следует
```

Продолжение кода программы

```
x2 = float(input('Введите второе число = '))
print(f"Число {x1} : ", parite(x1))
print(f"Число {x2} : ", parite(x2))
print(f"Максимальное число между {x1} и {x2} : ",
                                        my_max(x1, x2))
```

```
# Результат выполнения:
Введите любое число = 4
Введите второе число = 61
Число 4,0 :  pair
Число 61,0 :  impair
Максимальное число между 4,0 и 61,0 :  61,0
```

Решение № 2 Импортировать части модуля

```
# Теперь можно использовать часть модуля (my_max) :
from my_module import my_max

x1 = float(input('Введите любое число = '))
x2 = float(input('Введите второе число = '))
print(f"Максимальное число между {x1} и {x2} : ",
                                        my_max(x1, x2))
```

```
# Результат выполнения:
Введите любое число = 10
Введите второе число = 5
Максимальное число между 10,0 и 5,0 : 10,0
```

Если модуль находится в другом каталоге, чем код, например, он доступен по пути: «/MyCodes/my_Module», тогда точность пути к модулю в коде будет следующий:

*from MyCodes.my_Module import **

Упражнение 5.2

Обновите решения в упражнении 4.1 предыдущей главы, добавив по крайней мере одну функцию «*def*» в каждое разрешенное упражнение. Цель обновления заключает в том, чтобы сделать эти программные коды более эффективными.

Продолжение на странице 308

EN V.7. How to create a module in Python

Let's start with this exercise to understand it. Create a module that determins if a number is even or if it is negative.

Step 1: create a file with extension "*py*" to import, let's name it like this: «*my_module.py*»

Step 2: enter in this file codes that perform one or more specific tasks. These preference codes can be in different functions. In our case, we will have 2 functions:

 # With the first function, we determine if the digit is even or not.

```
def parite(y):
    while y >= 2:
        y = y-2
    if y == 1:
        res = 'odd'
    else:
        res = 'even'
    return res
```

The new *parite()* function has an input parameter that is *y*

Returns the value of "res" by «return» where it was called

 # We can add another function in the module that will have the role of determining the maximum between two digits.

```
def my_max(x1, x2):
    if x1>x2:
        res = x1
    else:
        res = x2
return res
```

The new function *my_max()* has two input parameters which are *x1 and x2*.

Returns the value of “res” by «return» where it was called

Step 3: let's write a small program in Python code to test this module. Let's give this file any name: «*my_code.py*». The two files can be at the same time in a single directory or in different directories provided that you specify in the file «my_code.py», the file path «my_module.py» when it is imported.

Resolution 1 Import the entire module

```
# We can now use the functions of the module (parite and
my_max) :
from my_module import *

x1 = float(input('Enter any number = '))
x2 = float(input('Enter a second number = '))
print(f"The number {x1} is : ", parite(x1))
print(f"The number {x2} is : ", parite(x2))
print(f"The max. between {x1} and {x2} is :", my_max(x1, x2))
```

```
# The result of execution:
Enter any number = 4
Enter a second number = 61
The number 4.0 is : even
The number 61.0 is : odd
The max. between 4.0 and 61.0 is: 61.0
```

Resolution 2 Import part of the module

```
# We can now use part of the module (my_max) :
from my_module import my_max

x1 = float(input('Enter any number = '))
x2 = float(input('Enter a second number = '))

print(f"The max. between {x1} et {x2} est : ", my_max(x1, x2))
```

```
# The result of execution:
Enter any number = 10
Enter a second number = 5
The max. between 10.0 and 5.0 is: 10.0
```

If the module is located on a different directory of the code, for example, it accessible via the path: «*/MyCodes/my_Module*», thus the precision of the path to the module in the code will be:

*from MyCodes.my_Module import **

Exercise 5.2

Modernize the resolutions of exercise 4.1 of the preceding chapter by adding at least one "*def*" function in each resolved exercise. The purpose of the update is also to make these program codes more efficient.

Continued on page 310

FR V.7. Comment créer un module dans Python

Commençons par cet exercice pour le comprendre: réaliser un module qui permet de déterminer si un chiffre est paire ou alors s'il est négatif.

Étape 1: créer un fichier avec extension "*py*" à importer, nommons le ainsi: «my_module.py»

Étape 2: introduire dans ce fichier des codes qui réalisent une ou des tâches bien déterminées. Ces codes de préférence peuvent être dans des fonctions différentes. Dans notre cas, nous aurons 2 fonctions:

 # Avec la première fonction, nous déterminons si le chiffre est pair ou pas

```
def parite(y):
    while y >= 2:
        y = y-2
    if y == 1:
        res = 'impair'
    else:
        res = 'pair'
    return res
```

La nouvelle fonction *parite()* a un paramètre d'entrée qui est *y*

Retourne la valeur de "res" par «return» là où elle a été appelée

 # Nous pouvons ajouter une autre fonction dans le module qui aura pour rôle de déterminer le maximum entre deux chiffres

```
def my_max(x1, x2):
    if x1>x2:
        res = x1
    else:
        res = x2
return res
```

La nouvelle fonction *my_max()* a deux paramètres d'entrée qui sont *x1 et x2.*

Retourne la valeur de "res" par «return» là où elle a été appelée

Étape 3: Écrivons un petit programme en code Python pour tester ce module. Donnons un nom quelconque à ce fichier: «my_code.py». Les deux fichiers peuvent être en même dans un répertoire unique ou dans des répertoires différents à condition de préciser dans le fichier «my_code.py», le chemin du fichier «my_module.py» lors de son importation.

Résolution 1 Importer tout le module

```
# On peut maintenant utiliser les fonctions du module (parite et my_max) :
from my_module import *

x1 = float(input('Introduire un nombre quelconque = '))
x2 = float(input('Introduire un second nombre = '))
print(f"le nombre {x1} est d'une parité : ", parite(x1))
print(f"le nombre {x2} est d'une parité : ", parite(x2))
print(f"le maximum entre {x1} et {x2} est : ", my_max(x1, x2))
```

```
# Résultat de l'exécution:
Introduire un nombre quelconque = 4
Introduire un second nombre = 61
le nombre 4.0 est d'une parité :  pair
le nombre 61.0 est d'une parité :  impair
le maximum entre 4.0 et 61.0 est :  61.0
```

Résolution 2 Importer une partie du module

```
# On peut maintenant utiliser une partie du module (my_max) :
from my_module import my_max

x1 = float(input('Introduire un nombre quelconque = '))
x2 = float(input('Introduire un second nombre = '))

print(f"le maximum entre {x1} et {x2} est : ", my_max(x1, x2))
```

```
# Résultat de l'exécution:
Introduire un nombre quelconque = 10
Introduire un second nombre = 5
le maximum entre 10.0 et 5.0 est :  10.0
```

Si le module se trouve sur un répertoire différent du code, et accessible par exemple via le chemin: «*/MesCodes/my_Module*», alors la précision du chemin vers le module dans le code sera:

*from MesCodes.my_Module import **

Exercice 5.2

Moderniser les résolutions de l'exercice 4.1 du chapitre précédant en ajoutant au moins une fonction «***def***» dans chaque exercice résolu. Le but également de la mise à jour est de rendre ces codes de programme plus efficaces.

Suite à la page 313

RU Глава VI. УПРАЖНЕНИЯ И НЕКОТОРЫЕ РЕШЕНИЯ
EN Chapter VI. EXERCISES AND SOME RESOLUTIONS
FR Chapitre VI. EXERCICES ET QUELQUES RÉSOLUTIONS

RU VI.1. Упражнения без решения

Упражнение №6.1 Напишите программу на Python, которая позволит находить и отображать все делители любого целого числа N.

Упражнение №6.2 Напишите программу на Python, которая позволит преобразовать целое число из любого основания в десятичное.

Упражнение № 6.3 Напишите программу на Python, которая позволит конвертировать правильные дроби из любой базы в десятичные.

Упражнение №6.4 Дан двумерный массив A[i, j] размера $n \times m$ (где n – количество строк, а m – количество столбцов). Напишите программу на Python, которая позволит определить количество четных элементов в каждой строке.

Упражнение №6.5 Дан двумерный массив B[i, j] размером $n \times n$ (где n в то же время количество строк и количество столбцов). Напишите программу на Python, которая позволит определить, является ли данный массив латинским квадратом (это квадратный массив отдельных элементов, каждая строка и столбец которого содержат только один экземпляр, и, кроме того, сумма элементов в каждой строке и в каждом столбце равна) или нет.

Упражнение №6.6 Напишите программу на Python, позволяющую создать по определенному правилу двумерный массив порядка 5 или размера 5×5, в котором суммы элементов не только на

каждой строке и на каждом столбце одинаковы или равны, но и на двух диагоналях.

Упражнение №6.7 Дан двумерный массив размером $n \times n$, где n – число которое делиться на 4. Запишите коды на языке Python, которые позволят выполнить следующие действия:

1) разделите исходный массив на четыре блока, где каждый блок (новый двумерный массив) имеет размер $\frac{n}{2} \times \frac{n}{2}$; элементы блоков суммируются и записываются в новый массив размером $\frac{n}{2} \times \frac{n}{2}$;

2) запишите все элементы в одномерном массиве и сделайте сортировку по убыванию;

3) найдите разность между максимальным и минимальным числом нового массива и определите его тип: четный или нечетный.

Например n = 6, получим:

- количество блоков будет равно 9 (см. рис. 6.2);
- двумерный массив находится на рисунке 6.1;
- разделив таблицу на рисунке 6.1, получаем таблицу на рисунке 6.2.

Обозначение двумерного массива: $B_{[i][j]}$

$B_{[1][1]}$	$B_{[1][2]}$	$B_{[1][3]}$	$B_{[1][4]}$	$B_{[1][5]}$	$B_{[1][6]}$
$B_{[2][1]}$	$B_{[2][2]}$	$B_{[2][3]}$	$B_{[2][4]}$	$B_{[2][5]}$	$B_{[2][6]}$
$B_{[3][1]}$	$B_{[3][2]}$	$B_{[3][3]}$	$B_{[3][4]}$	$B_{[3][5]}$	$B_{[3][6]}$
$B_{[4][1]}$	$B_{[4][2]}$	$B_{[4][3]}$	$B_{[4][4]}$	$B_{[4][5]}$	$B_{[4][6]}$
$B_{[5][1]}$	$B_{[5][2]}$	$B_{[5][3]}$	$B_{[5][4]}$	$B_{[5][5]}$	$B_{[5][6]}$
*$B_{[6][1]}$	$B_{[6][2]}$	$B_{[6][3]}$	$B_{[6][4]}$	$B_{[6][5]}$	$B_{[6][6]}$

i - номер строки; *j* - номер столбца ;

количество строк и столбцов составляет 6 каждый.

Рис. 6.1 Пример двумерного массива размером 6×6 без элементов

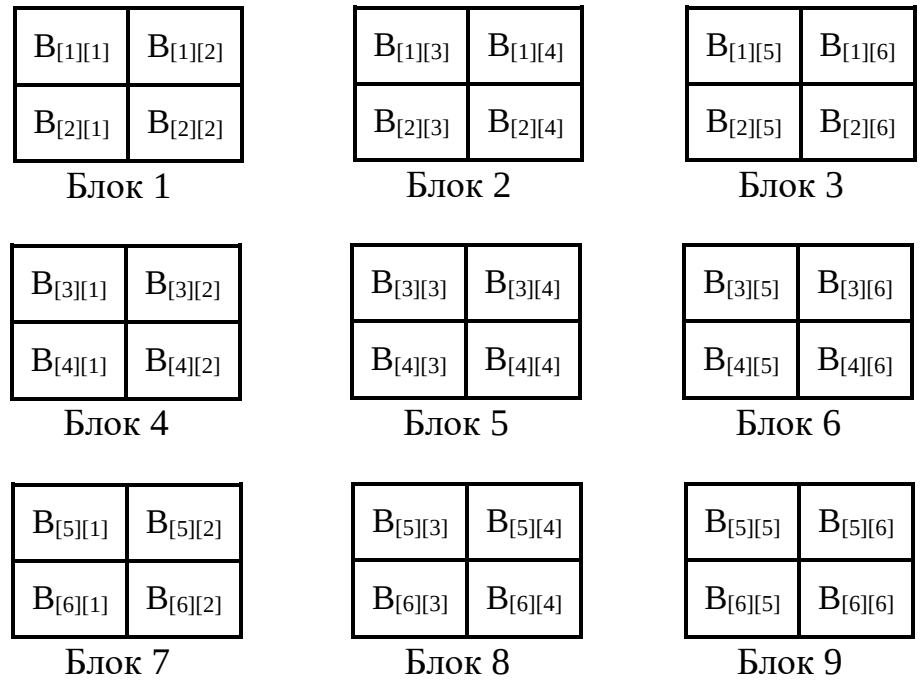


Рис. 6.2 Различные блоки двумерного массива

Как сложить элементы блока №1?

$$R_1=B_{[1][1]}+ B_{[1][2]}+ B_{[2][1]}+ B_{[2][2]}$$

Итак далее, необходимо найти R_2, R_3, R_4, R_5, R_6, R_7, R_8 и R_9. Сохраните эти результаты в новом одномерном массиве:

R_1	R_2	R_3	R_4	R_5	R_6	R_7	R_8	R_9

Сортировка элементов массива в порядке убывания:

R_9	R_8	R_7	R_6	R_5	R_4	R_3	R_2	R_1

Найдите разницу между максимальным и минимальным элементом этого нового массива: $D = R_9 – R_1$ и определите, является ли он четным или нечетным.

Продолжение на странице 316

EN VI.1. Some exercises

Exercise 6.1 Write a program code in Python that allows you to find and display all divisors of any integer N.

Exercise 6.2 Write a program code in Python that allows you to convert an integer from any base to decimal.

Exercise 6.3 Write a program code in Python that allows you to convert the correct fractions from any base to decimal base.

Exercise 6.4 You are given a two-dimensional array of numbers A [i, j] of size n×m (where n is the number of rows and m is the number of columns), write a program code in Python that allows you to determine the number of even elements on each row.

Exercise 6.5 You are given a two-dimensional array of numbers C [i, j] of size n×n (where n is at the same time the number of rows and the number of columns), write a program code in Python that allows you to determine whether a given array is a latin square (it is a square array of distinct elements of which each row and each column contains only one copy and in addition the sum of the elements on each row and on each column are equal) or not.

Exercise 6.6 Write a program code in Python that allows you to create a two-dimensional array of order 5 or size 5×5, in which the sums of the elements not only on each row and on each column are the same or equal, but also on the 2 diagonals.

Exercise 6.7 Given a two-dimensional array of size n×n, where n is a number that is divisible by 4. Write program codes in Python, which allow you to perform the following actions:

1) it is necessary to divide the original array into four blocks, where each block (a new two-dimensional array) has the size $k = \frac{n}{2} \times \frac{n}{2}$; the elements of the blocks are summed and written to a new array of the size $\frac{n}{2} \times \frac{n}{2}$;

2) write all the elements in a one-dimensional array and do a descending sort;

3) find the difference between the maximum and minimum number of the new array and determine its type: even or odd.

For example, n = 6, we get:

- the number of blocks will be 9 (see Fig. 6.2);
- the two-dimensional array is shown in figure 6.1;
- dividing the table in figure 6.1, we get the table in figure 6.2;

Designation of a two-dimensional array: $B_{[i][j]}$

$B_{[1][1]}$	$B_{[1][2]}$	$B_{[1][3]}$	$B_{[1][4]}$	$B_{[1][5]}$	$B_{[1][6]}$
$B_{[2][1]}$	$B_{[2][2]}$	$B_{[2][3]}$	$B_{[2][4]}$	$B_{[2][5]}$	$B_{[2][6]}$
$B_{[3][1]}$	$B_{[3][2]}$	$B_{[3][3]}$	$B_{[3][4]}$	$B_{[3][5]}$	$B_{[3][6]}$
$B_{[4][1]}$	$B_{[4][2]}$	$B_{[4][3]}$	$B_{[4][4]}$	$B_{[4][5]}$	$B_{[4][6]}$
$B_{[5][1]}$	$B_{[5][2]}$	$B_{[5][3]}$	$B_{[5][4]}$	$B_{[5][5]}$	$B_{[5][6]}$
$B_{[6][1]}$	$B_{[6][2]}$	$B_{[6][3]}$	$B_{[6][4]}$	$B_{[6][5]}$	$B_{[6][6]}$

i – row number; *j* – column number ;
the number of rows and columns is 6 each.

Fig. 6.1 Example of a two-dimensional 6×6 array without elements

$B_{[1][1]}$	$B_{[1][2]}$	$B_{[1][3]}$	$B_{[1][4]}$	$B_{[1][5]}$	$B_{[1][6]}$
$B_{[2][1]}$	$B_{[2][2]}$	$B_{[2][3]}$	$B_{[2][4]}$	$B_{[2][5]}$	$B_{[2][6]}$

Bloc1 Bloc2 Bloc3

$B_{[3][1]}$	$B_{[3][2]}$	$B_{[3][3]}$	$B_{[3][4]}$	$B_{[3][5]}$	$B_{[3][6]}$
$B_{[4][1]}$	$B_{[4][2]}$	$B_{[4][3]}$	$B_{[4][4]}$	$B_{[4][5]}$	$B_{[4][6]}$

Bloc4 Bloc5 Bloc6

Fig. 6.2 Various blocks of a two-dimensional array (part 1)

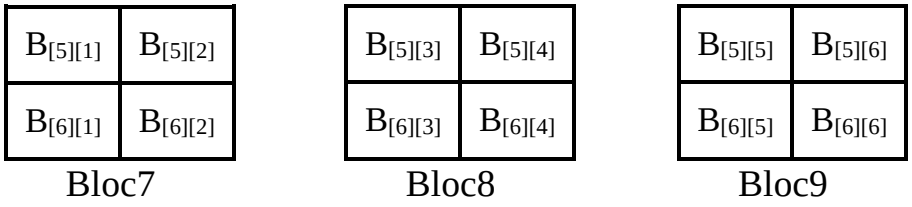


Fig. 6.2 Various blocks of a two-dimensional array (part 2)

How to sum the elements of Block №1?

$$R_1=B_{[1][1]}+ B_{[1][2]}+ B_{[2][1]}+ B_{[2][2]}$$

So on, it will be necessary to find R_2, R_3, R_4, R_5, R_6, R_7, R_8 et R_9.

Save these results in a new one-dimensional array:

R_1	R_2	R_3	R_4	R_5	R_6	R_7	R_8	R_9

Sort array items in descending order:

R_9	R_8	R_7	R_6	R_5	R_4	R_3	R_2	R_1

Find the difference between the maximum element and the minimum element of this new array: $D = R_9 – R_1$ and determine whether it is even or odd.

Continued on page 321

FR VI.1. Quelques exercices

Exercice 6.1 Écrire un programme en Python qui vous permet de trouver et d'afficher tous les diviseurs d’un nombre entier N quelconque.

Exercice 6.2 Écrire un programme en Python qui vous permet de convertir un entier de n'importe quelle base en décimal.

Exercice 6.3 Écrire un programme en Python qui vous permet de convertir les fractions correctes de n'importe quelle base en base décimale.

Exercice 6.4 Il vous est donné un tableau d'effectifs bidimensionnel A[i, j] de taille n×m (où n est le nombre de lignes et m est le nombre de colonnes), écrire un programme en Python qui vous permet de déterminer le nombre d'éléments pairs sur chaque ligne.

Exercice 6.5 Il vous est donné un tableau d'effectifs bidimensionnel C[i, j] de taille n×n (où n est en même temps le nombre de rangées et le nombre de colonnes), écrire un programme en Python qui vous permet de déterminer si un tableau donné est un carré latin (c'est un tableau carré d'éléments distincts dont chaque rangée et chaque colonne ne contient qu'un seul exemplaire et en plus la somme des éléments sur chaque rangée et sur chaque colonne sont égales) ou pas.

Exercice 6.6 Écrire un programme en Python qui vous permet de créer un tableau d'effectifs bidimensionnel d'ordre 5 ou de taille 5×5, dans lequel les sommes des éléments non seulement sur chaque rangée et sur chaque colonne sont les mêmes ou égales, mais aussi sur les 2 diagonales.

Exercice 6.7

Un tableau d'effectifs bidimensionnel de taille n×n est donné, où n est un nombre divisible par 2. Écrire les codes en Python qui vous permettent d'effectuer les opérations suivantes:

1) il faudra diviser le tableau original en blocs égale au nombre $k = \frac{n}{2} \times \frac{n}{2}$, où chaque bloc (représente un nouveau tableau d'effectifs bidimensionnel) est d'ordre 2 (taille 2×2); les éléments des blocs sont tous additionnés et écrits dans un nouveau tableau de taille $\frac{n}{2} \times \frac{n}{2}$;

2) écrire tous les éléments(résultats des additions) dans un tableau d'effectifs unidimensionnel et faire le tri par ordre décroissant;

3) trouver la différence entre l'élément maximal et l'élément minimal de ce nouveau tableau et déterminer le type de chiffre qu'est: pair ou impair.

Par exemple n = 6, on aura:

- le nombre de bloc sera égale à 9 (voir figure 6.2);
- le tableau d'effectifs bidimensionnel est à la figure №6.1;

- en divisant le tableau de la figure №6.1, on obtient celui de la figure №6.2.

Notation du tableau bidimensionnel: $B_{[i][j]}$

$B_{[1][1]}$	$B_{[1][2]}$	$B_{[1][3]}$	$B_{[1][4]}$	$B_{[1][5]}$	$B_{[1][6]}$
$B_{[2][1]}$	$B_{[2][2]}$	$B_{[2][3]}$	$B_{[2][4]}$	$B_{[2][5]}$	$B_{[2][6]}$
$B_{[3][1]}$	$B_{[3][2]}$	$B_{[3][3]}$	$B_{[3][4]}$	$B_{[3][5]}$	$B_{[3][6]}$
$B_{[4][1]}$	$B_{[4][2]}$	$B_{[4][3]}$	$B_{[4][4]}$	$B_{[4][5]}$	$B_{[4][6]}$
$B_{[5][1]}$	$B_{[5][2]}$	$B_{[5][3]}$	$B_{[5][4]}$	$B_{[5][5]}$	$B_{[5][6]}$
$B_{[6][1]}$	$B_{[6][2]}$	$B_{[6][3]}$	$B_{[6][4]}$	$B_{[6][5]}$	$B_{[6][6]}$

i - numéro de rangée; *j* - numéro de colonne ;
le nombre de rangée et le nombre de colonne est chacun égal à 6.

Fig. 6.1 – Exemple du tableau d'effectifs bidimensionnel de dimension 6×6 sans les éléments

$B_{[1][1]}$	$B_{[1][2]}$	Bloc1	$B_{[1][3]}$	$B_{[1][4]}$	Bloc2	$B_{[1][5]}$	$B_{[1][6]}$	Bloc3
$B_{[2][1]}$	$B_{[2][2]}$		$B_{[2][3]}$	$B_{[2][4]}$		$B_{[2][5]}$	$B_{[2][6]}$	
$B_{[3][1]}$	$B_{[3][2]}$	Bloc4	$B_{[3][3]}$	$B_{[3][4]}$	Bloc5	$B_{[3][5]}$	$B_{[3][6]}$	Bloc6
$B_{[4][1]}$	$B_{[4][2]}$		$B_{[4][3]}$	$B_{[4][4]}$		$B_{[4][5]}$	$B_{[4][6]}$	
$B_{[5][1]}$	$B_{[5][2]}$	Bloc7	$B_{[5][3]}$	$B_{[5][4]}$	Bloc8	$B_{[5][5]}$	$B_{[5][6]}$	Bloc9
$B_{[6][1]}$	$B_{[6][2]}$		$B_{[6][3]}$	$B_{[6][4]}$		$B_{[6][5]}$	$B_{[6][6]}$	

Fig. 6.2 – Les différents blocs du tableau d'effectifs bidimensionnel

Comment additionner les éléments du bloc №1?

$$R_1=B_{[1][1]}+ B_{[1][2]}+ B_{[2][1]}+ B_{[2][2]}$$

Ainsi de suite, il faudra trouver R_2, R_3, R_4, R_5, R_6, R_7, R_8 et R_9.

Enregistrer ces résultats dans une nouveau tableau d'effectifs unidimensionnel:

R_1	R_2	R_3	R_4	R_5	R_6	R_7	R_8	R_9

Faire le tri des éléments du tableau par ordre décroissant:

R_9	R_8	R_7	R_6	R_5	R_4	R_3	R_2	R_1

Trouver la différence entre l'élément maximal et l'élément minimal de ce nouveau tableau: $D = R_9 – R_1$ et déterminer s'il est paire ou impaire.

Suite à la page 327

RU VI.2. Решение некоторых упражнений

Решение упражнения №1.1

Сначала представим блок-схему для решения этого уравнения второй степени (см. рис. 6.3). Где *a*, *b* и *c* обозначают действительные числа, D – дискриминант уравнения, x_1 и x_2 – корни уравнения.

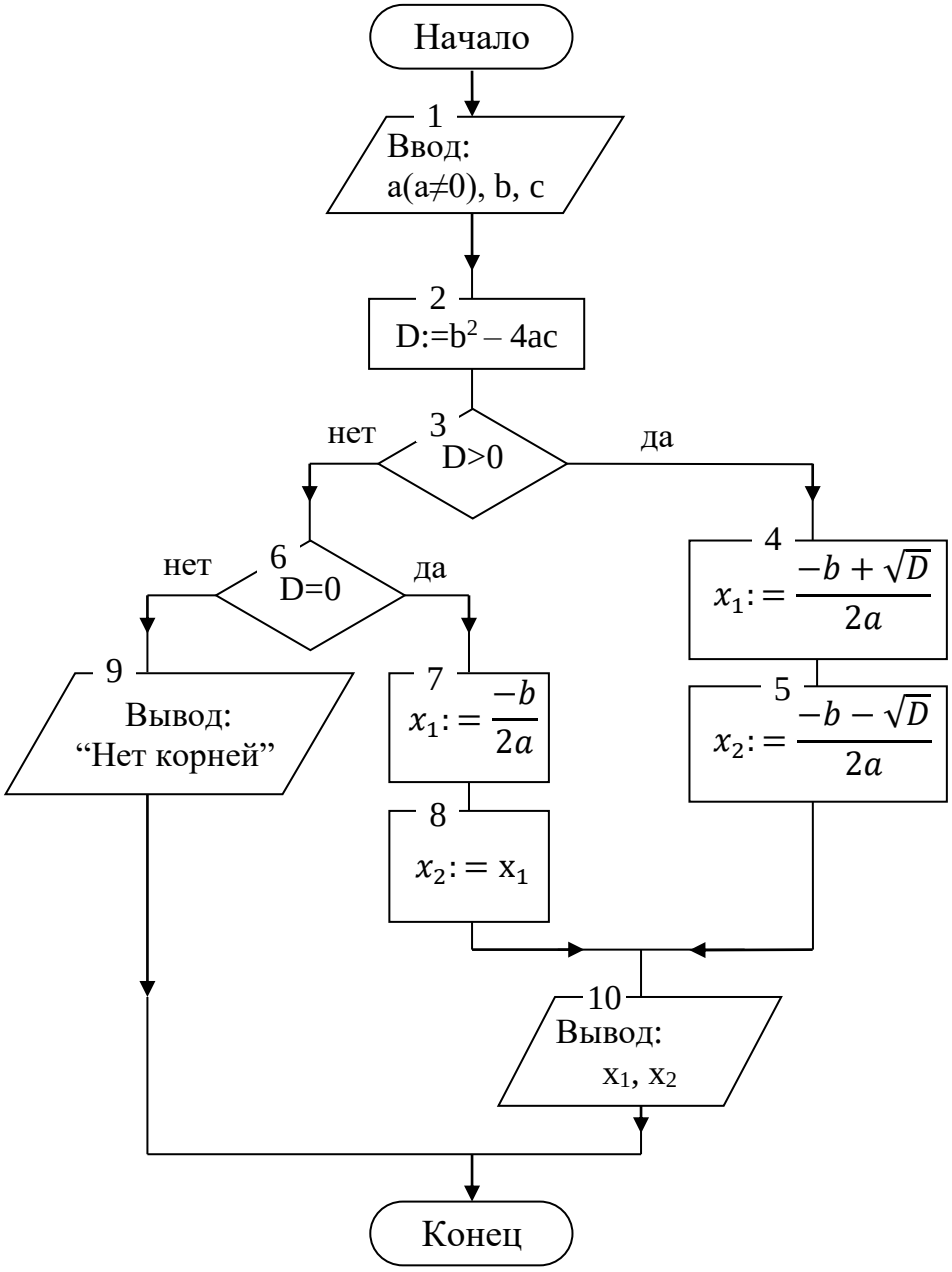


Рис. 6.3 Блок-схема решения уравнения второй степени

Программный код:

Рисунок 6.3

```
a = float(input ('Введите значение a = '))                    → Bloc 1
b = float(input ('Введите значение b = '))                    → Bloc 1
c = float(input ('Введите значение c = '))                    → Bloc 1
# Определим дискриминант
D=float(b*b - 4*a*c)                                          → Bloc 2
if (D>0):                                                     → Bloc 3
    # Вычисляем первый корень
    x1=(-b+D**0.5)/(2*a)                                      → Bloc 4
    # Вычисляем второй корень
    x2=(-b-D**0.5)/(2*a)                                      → Bloc 5
    print(f"Корни такие: x1={x1}, x2={x2}")                   → Bloc 10
elif (D==0):                                                  → Bloc 6
    # Вычисляем первый корень
    x1=(-b)/2*a                                               → Bloc 7
    # Вычисляем второй корень
    x2=x1                                                     → Bloc 8
    print(f"Корни такие: x1={x1}, x2=x1")                     → Bloc 10
else:
    print(f"Нет корней для уравнения")                        → Bloc 9
```

Обновление кода

1) Проверяем значение a, если оно равно 0, то мы показываем сообщение а том, что не должен быть 0 и повторяем процедуру ввода. Здесь необходимо применить инструкцию «While».

```
n = bool(True)
while n==True:
    a = float(input ('Введите значение a = '))
```

```
    if (a==0):
        print(' a не должно быть равно 0')
        n = True
        continue
    n = False
    b = float(input ('Введите значение b = '))
    c = float(input ('Введите значение c = '))
    # Определим дискриминант
    D=float(b*b - 4*a*c)
    print(f"Дискриминант = {D}")
    if (D>0):
        # Вычисляем первый корень
        x1=(-b-D**0.5)/(2*a)
        # Вычисляем второй корень
        x2=(-b+D**0.5)/(2*a)
        print(f"Корни такие: x1={x1}, x2={x2}")
    elif (D==0):
        # Вычисляем первый корень
        x1=(-b)/2*a
        # Вычисляем второй корень
        x2=x1
        print(f" Корни такие: x1={x1}, x2=x1")
    else:
        print(f" Нет корней для уравнения ")

print("Конец!")
```

Самостоятельное задание №6.1 Обновите вновь код, используя конструкций «*try*», «*except*», «*else*», «*finally*» или/и «*raise*».

Решение упражнения №3.1

```
import numpy as np
# Найдите индексы, которые имеют значения меньше 3
noms_stud = np.array(['Ivanov', 'Andreeva', 'Petrov', 'Antonov',
'Misonov', 'Sidorov', 'Davidov', 'Shelkova', 'Timkov', 'Belova'])
laliste = np.array([3, 4, 5, 2, 0, 3, 5, 4, 2, 3])
filt_i = np.where(laliste<3) # Эти индексы сохраняются здесь


# Создайте новый список на основе этих индексов
# Создать пустой список для оценок людей в “долгах”
Ld_notes = []
# создать пустой список людей (фамилии) в “долгах”
Ld_noms = []
# создать пустой список людей без “долгов”
Lsd_noms = noms_stud

for x in filt_i[0]:
    Ld_notes.append(laliste[x])
    Ld_noms.append(noms_stud[x])

print(f"Список для оценок людей в \“долгах\”: {Ld_notes}")
print(f"Список людей (фамилии) в \“долгах\”: {Ld_noms}")

Lsd_noms = np.delete(noms_stud, filt_i[0])
print(f"Список людей (фамилии) без \“долгов\”: {Lsd_noms}")
```

```
# Результат:
Список для оценок людей в “долгах”: [2, 0, 2]
Список людей (фамилии) в “долгах”:
                    ['Antonov', 'Misonov', 'Timkov']
Список людей (фамилии) без “долгов”:
['Ivanov' 'Andreeva' 'Petrov' 'Sidorov' 'Davidov' 'Shelkova' 'Belova']
```

Решение упражнения №5.1

Напомним, что факториал является произведением строго положительных целых чисел, меньших или равных *n*. В математике факториал натурального целого числа *n* обозначается так: *n*!. Что значит: (n+1)! = n! × (n+1) для всех *n*.

Функция в Python будет выглядеть следующим образом:

```
def factor(n):
   if n<=1:
        return 1
   else:
        return n * factor(n-1)
```

Как вызвать функцию ?

```
x = int(input('Введите целое число = '))
fac = factor(x)
print(f"{x}!= ", fac)
```

```
# Результат выполнения:
Введите целое число = 5
5!=  120
```

Продолжение на странице 332

EN VI.2. Resolution of exercises

Solving exercise 1.1

First represent an algorithm (flowchart) to solve this equation of the second degree (See figure 6.3). Where a, b and c denote real numbers, D – discriminant of the equation, x_1 and x_2 – the roots of the equation.

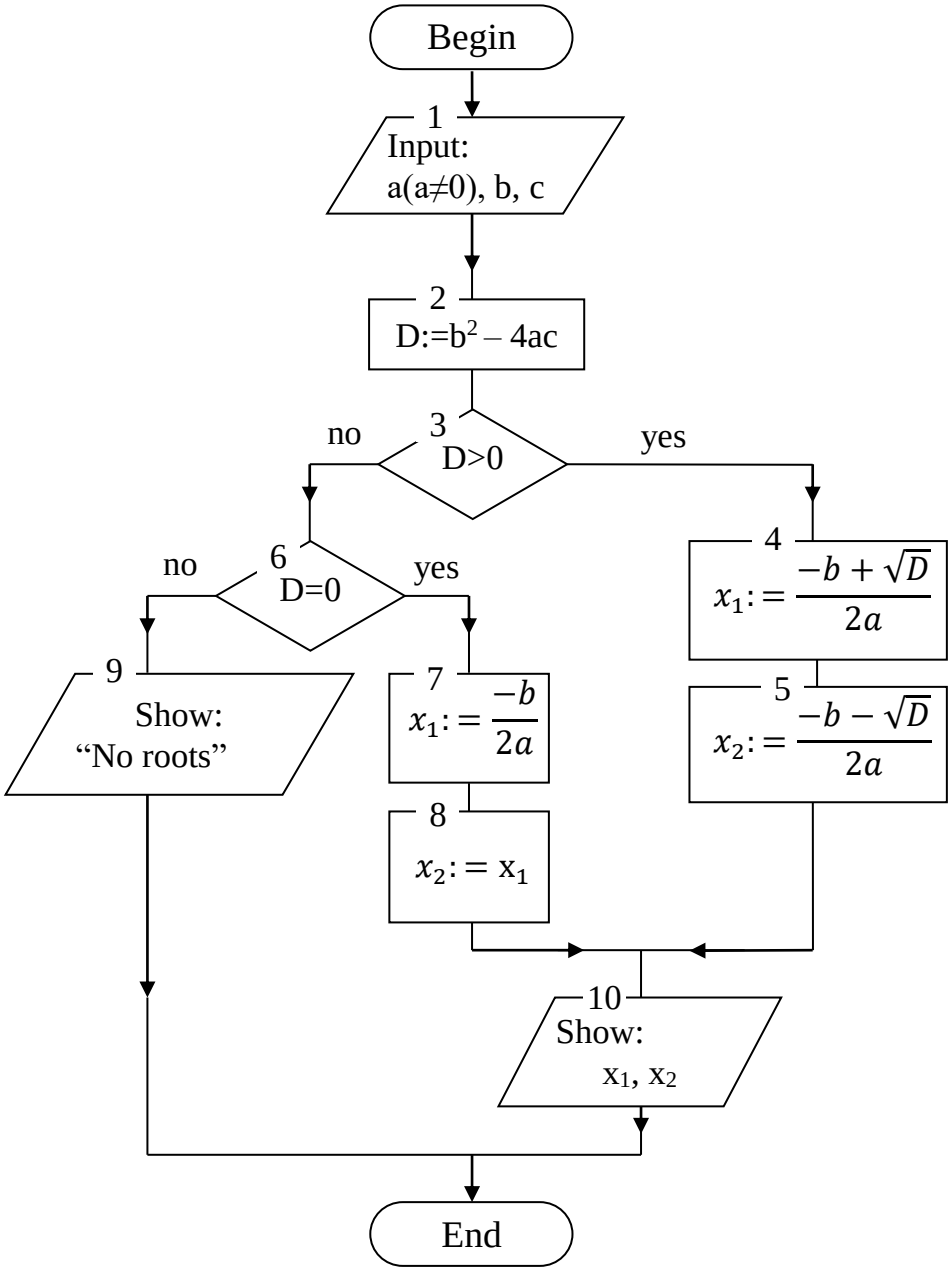


Fig. 6.3 Flowchart for solving the equation of the second degree

Program code:

Figure 6.3

```
a = float(input ('Enter the value of a = '))                    → Block 1
b = float(input ('Enter the value of b = '))                    → Block 1
c = float(input ('Enter the value of c = '))                    → Block 1
# Let's determine the discriminant
D=float(b*b - 4*a*c)                                            → Block 2
if (D>0):                                                       → Block 3
    # Let's determine the first root
    x1=(-b+D**0.5)/(2*a)                                        → Block 4
    # Let's determine the second root
    x2=(-b-D**0.5)/(2*a)                                        → Block 5
    print(f"Roots are : x1={x1}, x2={x2}")                      → Block 10
elif (D==0):                                                    → Block 6
    # Let's determine the first root
    x1=(-b)/2*a                                                 → Block 7
    # Let's determine the second root
    x2=x1                                                       → Block 8
    print(f" Roots are : x1={x1}, x2=x1")                       → Block 10
else:
    print(f"No root for the equation")                          → Block 9
```

Code update

1) we check the value of ***a***, if equal to 0, then we display a message and go out to start again. Here comes the «While» statement.

```
n = bool(True)
while n==True:
    a = float(input ('Enter the value of a = '))
    if (a==0):
```

```
            print('«a» must not be equal to 0')
            n = True
            continue
        n = False
        b = float(input ('Enter the value of b = '))
        c = float(input ('Enter the value of c = '))
        # Let's determine the discriminant
        D=float(b*b - 4*a*c)
        print(f"The discriminant is equal to {D}")
        if (D>0):
            # Let's determine the first root
            x1=(-b-D**0.5)/(2*a)
            # Let's determine the second root
            x2=(-b+D**0.5)/(2*a)
            print(f"Root are: x1={x1}, x2={x2}")
        elif (D==0):
            # Let's determine the first root
            x1=(-b)/2*a
            # Let's determine the second root
            x2=x1
            print(f"Root are: x1={x1}, x2=x1")
        else:
            print(f"No root for the equation")

    print("End!")
```

Individual exercise 6.1 Update the code again using the «*try*», «*except*», «*else*», « *finally* » or/and «*raise*» statements.

Solving exercise 3.1

```
import numpy as np
# Find the indexes so the values are < 3
names_stud = np.array(['Ivanov', 'Andreeva', 'Petrov', 'Antonov',
'Misonov', 'Sidorov', 'Davidov', 'Shelkova', 'Timkov', 'Belova'])
thelist = np.array([3, 4, 5, 2, 0, 3, 5, 4, 2, 3])
filt_i = np.where(thelist<3) # These indexes are saved here

# Build a new list from these indexes
# create an empty list for grades of people who are going to retake
                                                    # the exam
Ld_notes = []
# create an empty list of people (surnames) to retake the exam
Ld_names = []
# create an empty list of people who are not going to retake the
                                                    # exam.
Lsd_names = names_stud

for x in filt_i[0]:
    Ld_notes.append(thelist[x])
    Ld_names.append(names_stud[x])

print(f"The list of grades of people to retake the exam:
{Ld_notes}")
print(f"The list of people(surnames) to retake the exam:
{Ld_names}")
Lsd_names = np.delete(names_stud, filt_i[0])
print(f"The list of people(surnames) who should not retake the
exam: {Lsd_names}")
```

```
# The result:
The list of grades of people to retake the exam: [2, 0, 2]
The list of people(surnames) to retake the exam.: ['Antonov',
          'Misonov', 'Timkov']
The list of people(surnames) who should not retake the exam:
['Ivanov' 'Andreeva' 'Petrov' 'Sidorov' 'Davidov' 'Shelkova' 'Belova']
```

Solving exercise 5.1

Recall that a factorial is the product of strictly positive integers less than or equal to ***n***. In Mathematics, the factorial of a natural integer ***n*** is denoted thus: ***n***!, which means: $(n+1)! = n! \times (n+1)$ for any ***n***.

The function in Python will be as follows:

```
def factor(n):
   if n<=1:
        return 1
   else:
        return n * factor(n-1)
```

How to call the function ?

```
x = int(input('Enter an integer = '))
fac = factor(x)
print(f"{x}!= ", fac)
```

```
# The result of execution:
Enter an integer = 5
5!=  120
```

Continued on page 333

FR VI.2. Résolution des exercices

Résolution de l'exercice 1.1

Représenter d'abord un algorithme (organigramme) pour résoudre cette équation du second degré (voir figure 6.1). Où a, b et c désignent des nombres réels, D – discriminant de l'équation, x_1 et x_2 – les racines de l'équation.

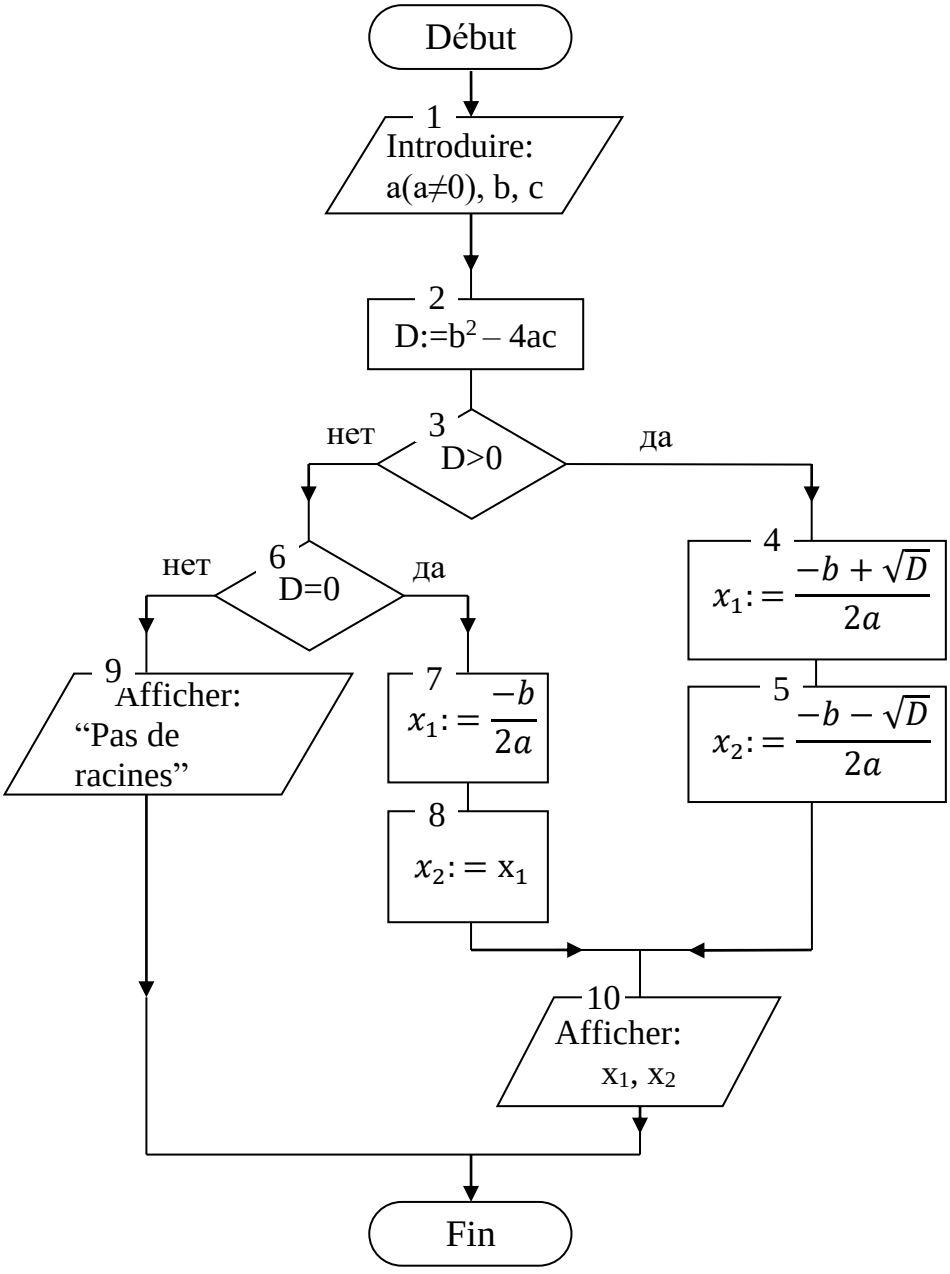


Fig. 6.3 – Organigramme pour résoudre l'équation du second degré

Le code:

Schéma №6.3

```
a = float(input ('Introduire la valeur de a = '))                    → Bloc 1
b = float(input ('Introduire la valeur de b = '))                    → Bloc 1
c = float(input ('Introduire la valeur de c = '))                    → Bloc 1
# Déterminons le discriminant
D=float(b*b - 4*a*c)                                                 → Bloc 2
if (D>0):                                                            → Bloc 3
   # Déterminons la première racine
   x1=(-b+D**0.5)/(2*a)                                              → Bloc 4
   # Déterminons la deuxième racine
   x2=(-b-D**0.5)/(2*a)                                              → Bloc 5
   print(f"Les racines sont: x1={x1}, x2={x2}")                      → Bloc 10
elif (D==0):                                                         → Bloc 6
   # Déterminons la première racine
   x1=(-b)/2*a                                                       → Bloc 7
   # Déterminons la deuxième racine
   x2=x1                                                             → Bloc 8
   print(f"Les racines sont: x1={x1}, x2=x1")                        → Bloc 10
else:
   print(f"Pas de racine pour l'equation")                          → Bloc 9
```

Modernisation du code

1) Nous vérifions la valeur de a, si égale à 0, alors nous affichons un message et sortons pour recommencer. Ici, intervient l'instruction «While».

```
n = bool(True)
while n==True:
   a = float(input ('Introduire la valeur de a = '))
   if (a==0):
```

```
            print('a ne doit pas être égale à 0')
            n = True
            continue
        n = False
        b = float(input ('Introduire la valeur de b = '))
        c = float(input ('Introduire la valeur de c = '))
        # Déterminons le discriminant
        D=float(b*b - 4*a*c)
        print(f"Le discriminant est égale à {D}")
        if (D>0):
            # Déterminons la première racine
            x1=(-b-D**0.5)/(2*a)
            # Déterminons la deuxième racine
            x2=(-b+D**0.5)/(2*a)
            print(f"Les racines sont: x1={x1}, x2={x2}")
        elif (D==0):
            # Déterminons la première racine
            x1=(-b)/2*a
            # Déterminons la deuxième racine
            x2=x1
            print(f"Les racines sont: x1={x1}, x2=x1")
        else:
            print(f"Pas de racine pour l'equation")

    print("FIN!")
```

Devoir individuel 6.1

Mettre de nouveau à jour le code en utilisant les instructions «*try*», «*except*», «*else*», «*finally*» ou/et «*raise*».

Résolution de l'exercice 3.1

```
import numpy as np
# Trouvez les index donc les valeurs sont < 3
noms_stud = np.array(['Ivanov', 'Andreeva', 'Petrov', 'Antonov',
'Misonov', 'Sidorov', 'Davidov', 'Shelkova', 'Timkov', 'Belova'])
laliste = np.array([3, 4, 5, 2, 0, 3, 5, 4, 2, 3])
filt_i = np.where(laliste<3) # Ces index sont sauvegardés ici

# Formez une nouvelle liste à base de ces index
# créer une liste vide pour les notes des personnes en rattrapage
Ld_notes = []
# créer une liste vide des personnes(noms) en rattrapage
Ld_noms = []
# créer une liste vide des personnes sans rattrapage

Lsd_noms = noms_stud

for x in filt_i[0]:
    Ld_notes.append(laliste[x])
    Ld_noms.append(noms_stud[x])

print(f"La liste des notes des personnes en rattrapage: {Ld_notes}")
print(f"La liste des personnes(noms) en rattrapage: {Ld_noms}")

Lsd_noms = np.delete(noms_stud, filt_i[0])
print(f"La liste des personnes(noms) sans rattrapage: {Lsd_noms}")
```

```
# Résultat:
La liste des notes des personnes en rattrapage: [2, 0, 2]
La liste des personnes(noms) en rattrapage:
                    ['Antonov', 'Misonov', 'Timkov']
La liste des personnes(noms) sans rattrapage:
['Ivanov' 'Andreeva' 'Petrov' 'Sidorov' 'Davidov' 'Shelkova' 'Belova']
```

Résolution de l'exercice 5.1

Rappelons qu'une factorielle est le produit des nombres entiers strictement positifs inférieurs ou égaux à n. En mathématiques, la factorielle d'un entier naturel n est notée ainsi: n!. Ce qui veut dire que: (n+1)! = n! × (n+1) pour tout n.

La fonction en Python sera la suivante:

```
def factor(n):
   if n<=1:
        return 1
   else:
        return n * factor(n-1)
```

Comment appeler la fonction ?

```
x = int(input('Introduire un nombre entier = '))
fac = factor(x)
print(f"{x}!= ", fac)
```

```
# Résultat de l'exécution:
Introduire un nombre entier = 5
5!=  120
```

Suite à la page 335

ЗАКЛЮЧЕНИЕ

Завершая изложение первого тома, сначала ответим на вопросы, поставленные во введении. Чтобы лучше освоить любой язык, вам нужно знать его начальные основы, чтобы вы могли лучше общаться на нём и чувствовать себя комфортно в использовании. Конечно, это не могут быть все учащиеся, которые начинают таким образом с базового понятия, по следующим причинам: либо из-за незнания того, как лучше начать, либо из-за своей цели, которая просто может быть вписанной в очень определенную область или ограниченные рамки.

Если вы желаете стать профессиональным разработчиком, то вам необходимо сперва овладеть основами языка, что будет верным началом в процессе обучения программированию на нем. Важно знать различные возможности этого языка, чтобы грамматно пользоваться им.

Таким образом, для того чтобы стать профессиональным разработчиком, еще прежде, чем заняться программированием, необходимо попытаться понять, с какой целью вы хотите учиться программировать и какой именно вид программирования сейчас вы хотите освоить: искусственный интеллект, веб-разработка (“*Front-end*” или “*Back-end*”), драйверы устройств, приложения (программное обеспечение).

Python и *Java* являются самыми универсальными и распространенными языками в мире. Это означает, что независимо от вашего выбора области программирования, знания, изложенные в учебном пособии, будут для вас необходимы.

Помните, что в программировании вы не можете обойтись без использования функций (собственных или нет) и исключений. На данном этапе это две важные области, на которых обучающимся будет нужно акцентировать больше внимания в этой книге, чтобы помочь себе стать разработчиком.

Также следует помнить, что низкоуровневые языки ближе к компьютеру, поскольку кодируются в двоичном формате. Они очень трудны людям для чтения или написания, но им требуется гораздо меньше места в памяти, что делает их быстрее. Что касается языков программирования высокого уровня, то они независимы от того, какая машина будет использоваться, и не имеют никаких “обязательств” перед компьютерами. Эти языки легче читаются людьми и очень далеки от машинных языков, поэтому им требуются интерпретаторы и/или компиляторы. Оба эти способа (компиляция и интерпретирование) имеют преимущества и недостатки. Компиляция делает выполнение программы намного быстрее, а сам исходный текст программы при этом остается закодированным, то есть закрытым. В то время как интерпретация имеет менее быстрое выполнение, а исходный текст (код) не является закрытым, поэтому он имеется в публичном доступе. Такой код быстрее развивается и исправляется.

CONCLUSION

To conclude this first volume, we will first answer the questions posed in the introduction. To better master any language, you need to know its initial basics so that you can better communicate in it and feel comfortable using it. Of course, it is not all students who start this way, with a basic concept, for the following reasons: either because of ignorance of “how to start best,” or because of their goal, which may simply be written in a very specific area or limited framework.

If you hope to become a professional developer, then you need to first master the basics of the language, which will be the right start in the process of learning to program in it. But it is important to know the various features of this language in order to use it skillfully.

Thus, in order to become a professional developer, even before you start programming, you need to try to understand for what purpose you want to learn programming and what kind of programming you want to master now: Artificial Intelligence, Web development ("*Front-end*" or "*Back-end*"), device drivers, applications (software).

Python and *Java* are the most versatile and widely used languages in the world. This means that regardless of your choice in the programming field, the knowledge outlined in this course book will be essential for you.

Remember that in programming, you can't do without using functions (native or not) and exceptions. At this stage, these are the two important areas that you will need to focus more on in this book to help you become developers.

Also keep in mind that low-level languages are closer to the computer because they are encoded in binary format. They are very difficult for people to read or write, but they require much less memory space, which makes them faster. As for high-level programming languages, they are independent of which machine will be used, and have no "obligations" to computers. These languages are easier to read by humans and are very far from machine languages, so they require interpreters and/or compilers. Both of these methods (compilation and interpretation) have advantages and disadvantages. Compilation makes program execution much faster, and the source code of the program itself remains encoded, that is, closed. The interpretation has a slower execution and the source code is not private and, therefore, available in the public domain. This code is faster to develop and fix.

CONCLUSION

Pour achever ce premier volume, nous allons conclure en répondant tout d'abord aux questions posées à l'introduction. Pour mieux maîtriser une langue quelconque donnée, il va falloir connaître les premières notions de base afin de pouvoir mieux communiquer et se sentir à l'aise dans son utilisation. Bien sûr que tout apprenant ne commence pas de cette manière, par la notion de base pour des raisons suivantes: soit à cause de son ignorance, soit par son but qui est juste limité dans un domaine ou cadre très réduit.

Pour espérer devenir développeur, il faudrait au minimum maîtriser ses notions de base qui constituent une bonne lancée dans le processus d'apprentissage de la programmation. Il est important de connaître les différentes possibilités qu'offre ce langage afin de pouvoir l'utiliser.

Ainsi avant toute programmation ou alors pour devenir un développeur de profession, il faudra tout d'abord chercher à comprendre pourquoi vous voulez programmer et quel genre de programmation vous tenez à maîtriser: (Intelligence artificielle, spécialiste en web (“*Front-end*” ou “*Back-end*”), pilote de périphérique, application (logiciel)).

Le *Python* et le *Java* sont des langages les plus polyvalents et les plus répandus dans le monde. Ce qui revient à dire que quelque soit votre choix, la maîtrise du contenu de ce manuel est très indispensable.

Retenons que vous ne pouvez pas vous passer de l'utilisation des fonctions (natives ou pas) et des exceptions en programmation. Pour le moment se sont deux parties importantes qu'il faudra minutieusement prêter plus attention afin de pouvoir espérer devenir un développeur professionnel.

Retenons également que les langages de bas niveau sont plus proches de l’ordinateur, car codé en binaire. Ils sont très difficile à lire ou écrire pour les êtres humains et nécessitent beaucoup moins d’espace mémoire, ce qui fait à ce qu'ils sont plus rapides. Quant aux langages de

programmation de haut niveau, ils sont indépendant de la machine et n'ont pas d'obligation vis-à-vis des ordinateurs. Ces langages sont lisibles par les êtres humains et très éloignés du langage machine, raison pour laquelle, ils nécessitent des interpréteurs et/ou compilateurs. Les deux processus ont des avantages et des inconvénients. La compilation rend l’exécution du programme beaucoup plus rapide et le code source reste codé c’est-à-dire privé pour l'Homme. Alors que l’interprétation a une exécution moins rapide et le code source n'est pas privé – donc public. Elle est plus rapide à développer et à être corrigée.

СПИСОК ИСПОЛЬЗОВАННОЙ И РЕКОМЕНДУЕМОЙ ЛИТЕРАТУРЫ
LIST OF LITERATURES USED AND RECOMMENDED
LISTE DES MANUELS (SOURCES) UTILISÉS ET RECOMMANDÉS

Информатика и информационные технологии: три в одном : учеб. пособие / Ф. Ж. Таннинг ; Владим. гос. ун-т им. А. Г. и Н. Г. Столетовых. – Владимир : Изд-во ВлГУ, 2020. – 268 с. ISBN 978-5-9984-1101-4

RU Интернет-источники EN URL Online FR Les sources

1) https://docs.python.org/fr/
2) https://docs.python.org/3/
3) https://www.w3schools.com/python/
4) https://github.com/numpy/numpy (The fundamental package for scientific computing with Python.)
5) https://www.pierre-giraud.com/
6) https://www.lfd.uci.edu/~gohlke/pythonlibs/
7) https://www.anaconda.com/products/individual
8) https://foxford.ru/wiki/informatika/dvumernye-massivy-v-python
9) https://labs-org.ru/python-8/
10) http://xymaths.free.fr/Informatique-Programmation/python/exercices.php
11) https://conda.io/en/latest/
12) https://riptutorial.com/numpy
13) https://younglinux.info/python/
14) https://pythonru.com/osnovy/znachenija-iskljuchenij-i-oshibok-v-python
15) https://www.tresfacile.net/

16) https://courspython.com/
17) https://python.hotexamples.com/ru/
18) https://www.tiobe.com/tiobe-index/
19) https://inforef.be/swi/download/apprendre_python3_5.pdf
20) https://www.commentcamarche.net/contents/617-les-langages-informatiques
21) http://www.maxime-ancelin.com/files/carreMagique.pdf

Учебное издание (Educational edition)

ТАННИНГ Жиогап Фирмэн

TANGNING Jiogap Firmin

ОСНОВЫ ПРОГРАММИРОВАНИЯ В PYTHON: ТРИ В ОДНОМ

THE BASICS OF PYTHON PROGRAMMING: THREE IN ONE

Учебное пособие (The course book)

Том I (Volume I)

Издается в авторской редакции (Published in the author's edition)

Подписано в печать 06.04.21.
Формат 60x84/16. Усл. печ. л. 19,76. Тираж 50 экз.
Заказ
Издательство
Владимирского государственного университета
имени Александра Григорьевича и Николая Григорьевича Столетовых.
600000, Владимир, ул. Горького, 87.