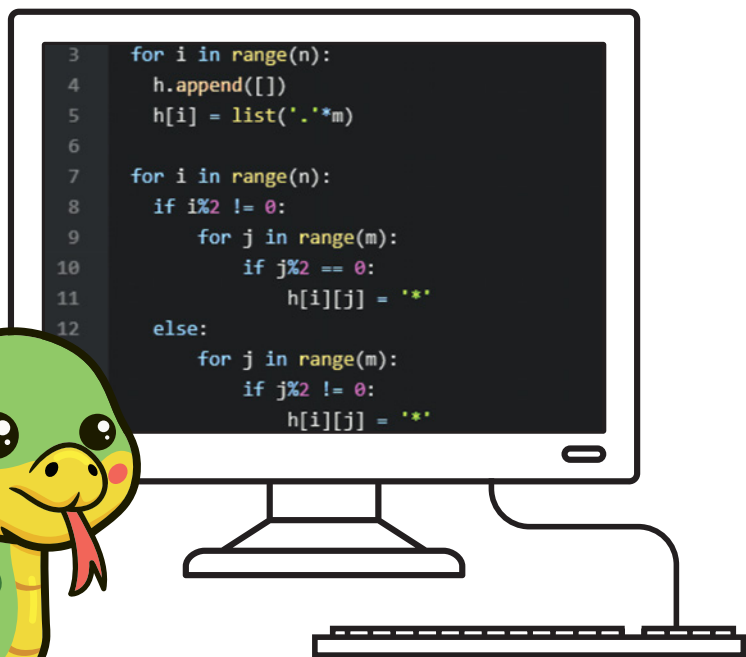
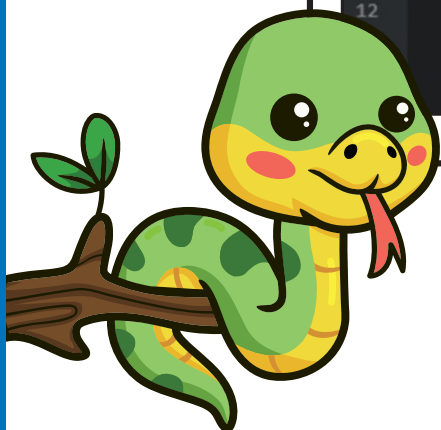




А. В. Щерба

Программирование на Python®

```
3 for i in range(n):
4     h.append([])
5     h[i] = list('.*m')
6
7 for i in range(n):
8     if i%2 != 0:
9         for j in range(m):
10            if j%2 == 0:
11                h[i][j] = '**'
12
13 else:
14     for j in range(m):
15         if j%2 != 0:
16             h[i][j] = '**'
```



Первые
шаги

А. В. Щерба

Программирование на Python®

Первые шаги

Электронное издание



Москва
Лаборатория знаний
2022

УДК 004.9
ББК 32.973
Щ61

Серия основана в 2018 г.

Щерба А. В.

Щ61 Программирование на Python® : Первые шаги / А. В. Щерба. — Электрон. изд. — М. : Лаборатория знаний, 2022. — 253 с. — (Школа юного программиста). — Систем. требования: Adobe Reader XI ; экран 10". — Загл. с титул. экрана. — Текст : электронный.

ISBN 978-5-93208-578-3

В книге описаны базовые конструкции программирования на языке Python: от именования переменных до многострочных программ с несколькими вложенными циклами и условными конструкциями.

Материал содержит подробные пошаговые инструкции, множество примеров. В помощь читателю приведены иллюстрации и примеры интерактивных оболочек программных кодов, а также ссылки на источники и интернет-ресурсы. В каждой главе разбор возможных ошибок и задания с ответами в конце книги.

Книга подойдет для дополнительного образования в школе и дома. Будет полезна учащимся средней и старшей школы, учителям, руководителям кружков и всем, кто интересуется программированием.

УДК 004.9
ББК 32.973

Деривативное издание на основе печатного аналога: Программирование на Python® : Первые шаги / А. В. Щерба. — М. : Лаборатория знаний, 2022. — 250 с. : ил. — (Школа юного программиста). — ISBN 978-5-93208-235-5.

6+

В соответствии со ст. 1299 и 1301 ГК РФ при устранении ограничений, установленных техническими средствами защиты авторских прав, правообладатель вправе требовать от нарушителя возмещения убытков или выплаты компенсации

ISBN 978-5-93208-578-3

© Лаборатория знаний, 2022

Введение

Программист — профессия, о которой многие наверняка что-то слышали. На ум приходит образ человека, постоянно сидящего за компьютером и «клацающего» по клавиатуре. И если не каждый учащийся после школы собирается стать программистом, то зачем учиться этому в школе? Попробуем ответить на этот вопрос.

Программа и программирование

Если говорить простыми словами, то *программа* — это инструкции для машины, *язык программирования* — способ их передачи, а *программирование* — сам процесс написания подобных инструкций на выбранном языке.

Как и для людей, для машины одна и та же инструкция может быть записана на различных языках (C++, Python, JavaScript, Ruby и др.).

В данной книге мы рассмотрим язык программирования Python. Этот язык характеризуется огромным количеством решаемых задач, простотой изучения и удобством работы с кодом, а также пользуется большим спросом среди работодателей и в IT-сообществе. Например, язык Python применяют для создания таких web-приложений, как Gmail, Google Maps и YouTube.

В чем ценность умения программировать?

Стоит отметить, что любой навык не обязательно используется именно в профессиональной сфере. Так, умение быстро считать, приобретаемое на уроках математики, способствует развитию быстрой реакции на поставленную задачу, составление таблиц по биологии или истории учит навыку выбирать главное и проследивать взаимосвязи, изучение языков показывает наличие совершенно другого типа мышления.

Тогда вопрос о важности приобретения этих навыков в школе становится практически риторическим и сводится к вопросу, важно ли научиться быстро реагировать, находить главное из

большого потока информации и смотреть на вещи под разным углом. Можете ответить на этот вопрос самостоятельно и поразмышлять о том, какие еще навыки развиваются в школе.

Умение структурировать и оптимизировать информацию/процессы, создавать понятный интерфейс приложения, доступный и удобный онлайн-сервис, организовывать диалог между пользователем и системой — все это находится на расстоянии вытянутой руки в самой распространенной профессиональной области XXI в.

Подобным навыкам начинают обучаться уже в школьные годы. Программирование как процесс помогает научиться выделять главное, раскладывать сложное на простое, развивает дальновидность и креативность. Что вовсе может не касаться компьютерной науки.

Как оптимизировать свои действия? С чего начать написание проекта или доклада?

Программирование — зерно, которое в скором времени обязательно даст свои плоды в виде уменьшения времени, которое затрачивается, например, на выполнение домашнего задания или создание проекта. Глобально же навык программирования можно рассматривать как инструмент развития личности, который пригодится во всех сферах жизни.

Предлагаем смело начать изучение основ программирования на языке Python, а в процессе самостоятельно решить, хотите вы начать заниматься этим на более глубоком и профессиональном уровне или применять освоенные навыки в повседневной жизни.

Глава 1. Знакомство со средой программирования IDLE и первая программа

IDLE — это среда, которая позволяет просматривать, редактировать, запускать и производить отладку программ на языке Python.

Данная среда программирования является свободно распространяемым программным обеспечением, доступным для скачивания с сайта www.python.org, поэтому ею может воспользоваться любой пользователь сети Интернет.

Установка среды IDLE

Пользователям **Windows** необходимо скачать установочный файл на официальном сайте <https://www.python.org/downloads/windows/>, кликнуть дважды по загруженному файлу и следовать инструкциям установщика (рис. 1).

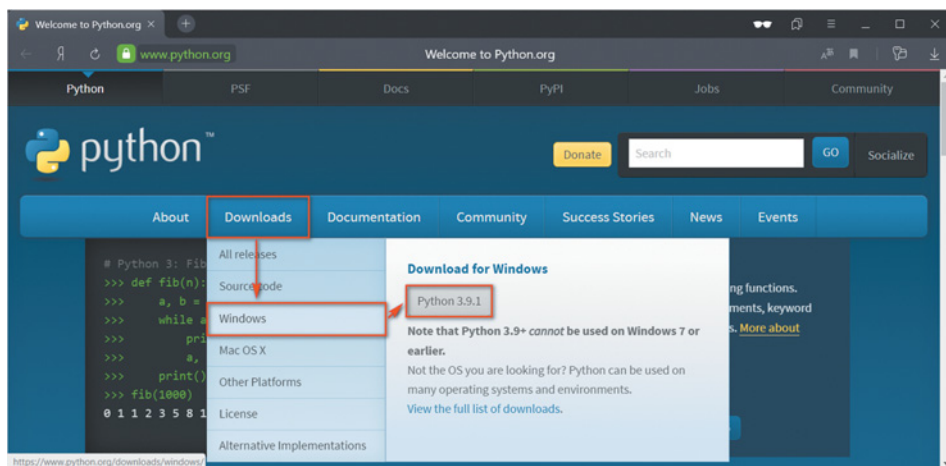


Рис. 1. Загрузка установочного файла

На **Linux** или **Mac OS** язык Python может быть уже установлен и готов к использованию, поскольку является стандартным компонентом этих операционных систем. Если его все же нет,

его можно скачать на официальном сайте, в том же разделе, что и для Windows.

В **Linux** для установки также достаточно двух команд в терминале:

```
$ sudo apt-get update
$ sudo apt-get install idle3
```

Интерфейс среды IDLE

После загрузки и установки Python откройте IDLE. На экране появится следующее окно (рис. 2):

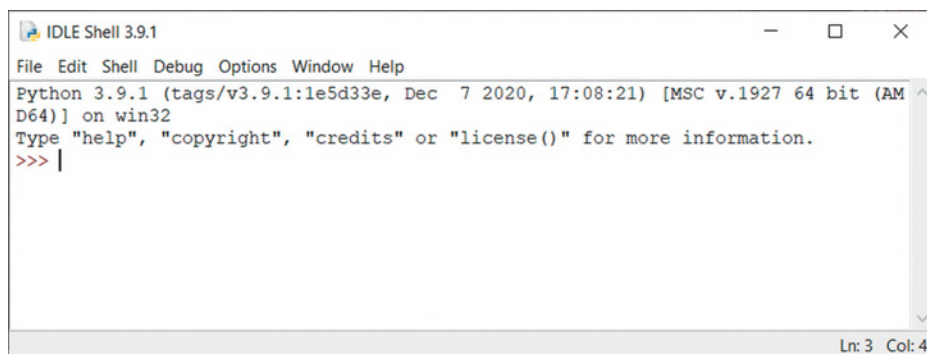


Рис. 2. Интерактивный режим IDLE

Перед нами *интерпретатор Python* — специальный модуль, который позволяет процессору считывать команды, записанные на языке программирования, и исполнять их. Другими словами, когда мы пишем код на языке Python, интерпретатор «читает» нашу программу и покомандно выполняет написанные в ней инструкции, опуская промежуточный этап сборки (компиляцию единого файла), в отличие от многих других языков программирования.

Существует два вида работы в IDLE: *интерактивный* и *с помощью создания отдельного файла*.

Интерактивный сеанс в IDLE начинается с вывода двух строк информационного текста о дате, времени и разрядности операционной системы, которые можно видеть на рис. 2, затем выводится приглашение к вводу команды >>>.

Ввод каждой инструкции завершается нажатием клавиши **Enter**, после чего интерпретатор Python выполняет эту операцию и выдает результат или сообщение об ошибке.

В интерактивном режиме можно вводить любое число команд, и каждая из них будет выполняться сразу же после ввода. Такой тип работы также называют работой в *интерактивной оболочке*.

Второй тип работы в IDLE мы рассмотрим позже в данной главе.

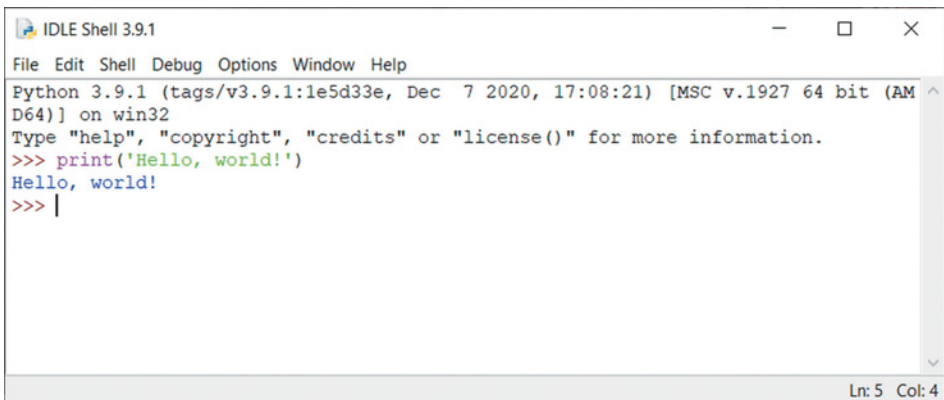
Первая программа на языке Python

Традиционно первой программой при изучении нового языка программирования является вывод строки «Hello, world!», символизирующей начало диалога между машиной и пользователем.

В строке приглашения к вводу введем первую инструкцию:

```
>>> print ('Hello, world!')
```

Теперь нажмем **Enter** и увидим, как интерпретатор Python моментально выполнит указание напечатать строку «Hello, world!» (рис. 3). Наша первая программа (инструкция для Python) готова!

A screenshot of the IDLE Shell 3.9.1 window. The window title is "IDLE Shell 3.9.1". The menu bar includes "File", "Edit", "Shell", "Debug", "Options", "Window", and "Help". The main text area shows the following text: "Python 3.9.1 (tags/v3.9.1:1e5d33e, Dec 7 2020, 17:08:21) [MSC v.1927 64 bit (AMD64)] on win32", "Type 'help', 'copyright', 'credits' or 'license()' for more information.", ">>> print('Hello, world!)", "Hello, world!", and ">>> |". The status bar at the bottom right shows "Ln: 5 Col: 4".

```
IDLE Shell 3.9.1
File Edit Shell Debug Options Window Help
Python 3.9.1 (tags/v3.9.1:1e5d33e, Dec 7 2020, 17:08:21) [MSC v.1927 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>> print('Hello, world!')
Hello, world!
>>> |
```

Рис. 3. Запуск первой программы на Python

Оператор print() — вывод данных на экран

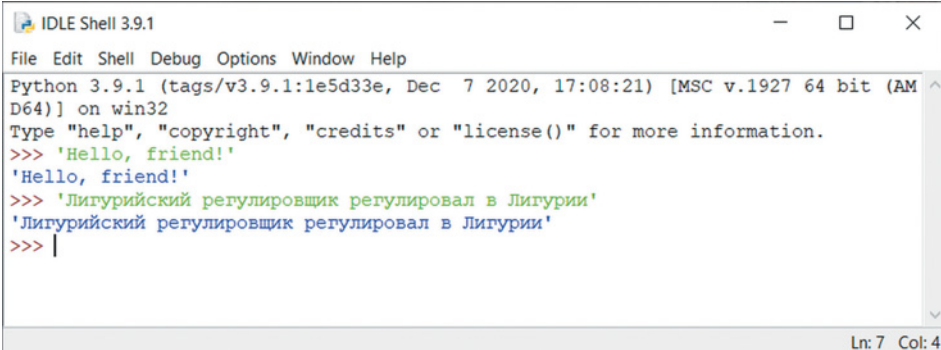
Вывод на экран, пожалуй, одна из самых важных функций в программировании. С ее помощью осуществляется передача информации пользователю или программисту.

Обратите внимание, что 'Hello, world!' в нашей первой программе записано в кавычках. Так мы показываем Python, что необходимо вывести данную последовательность символов (включая пробелы) без изменений. Это также означает, что последовательность символов может быть любой и даже не иметь смысловой нагрузки. Например, '!', 'Error', ' ' или 'ytrewq'.

Последовательность символов, заключенная в кавычки, называется *строкой* (не путать со строчкой в тексте) и является *неизменяемым типом данных*. Поэтому команда `print('1')` выведет на экран не число 1, а символ «1», с которым, например, нельзя будет производить арифметические операции.

Важно!

Интерактивная оболочка позволяет не использовать `print()`, поскольку в данном режиме нажатие клавиши **Enter** подразумевает автоматический вывод результата команды (рис. 4).



```
Python 3.9.1 (tags/v3.9.1:1e5d33e, Dec 7 2020, 17:08:21) [MSC v.1927 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>> 'hello, friend!'
'Hello, friend!'
>>> 'Лигурийский регулировщик регулировал в Лигурии'
'Лигурийский регулировщик регулировал в Лигурии'
>>> |
```

Рис. 4. Возможность печатать строки без команды `print()`

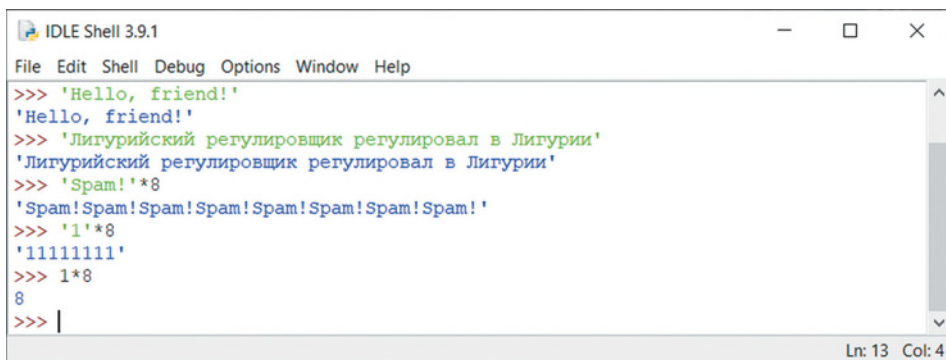
Эксперименты в интерактивной оболочке

Благодаря тому что программный код выполняется немедленно, интерактивный режим превращается в замечательный инструмент для экспериментов.

Предположим, что мы изучаем некоторый фрагмент программы на языке Python и наталкиваемся на выражения:

```
'Spam!'*8
'1'*8
1*8
```

Можно потратить с десяток минут в попытках выяснить, что же делают эти инструкции, а можно выполнить их в интерактивной оболочке — так будет намного быстрее и проще (рис. 5).

The image shows a screenshot of the IDLE Shell 3.9.1 window. The window title is "IDLE Shell 3.9.1" and it has a menu bar with "File", "Edit", "Shell", "Debug", "Options", "Window", and "Help". The main text area contains the following Python code and its output:

```
>>> 'Hello, friend!'
'Hello, friend!'
>>> 'Лигурийский регулировщик регулировал в Лигурии'
'Лигурийский регулировщик регулировал в Лигурии'
>>> 'Spam!'*8
'Spam!Spam!Spam!Spam!Spam!Spam!Spam!Spam!'
>>> '1'*8
'11111111'
>>> 1*8
8
>>> |
```

The status bar at the bottom right of the window shows "Ln: 13 Col: 4".

Рис. 5. Инструкция для Python вывести на экран
'Spam!'*8, '1'*8, 1*8

Первый эксперимент наглядно показывает, что произошло умножение строки 'Spam!' на число 8: в языке Python оператор `*` выполняет операцию умножения над числами, но, если левый операнд является строкой, он действует как оператор *многократной конкатенации* строки с самой собой.

Не совсем понятно, правда? Разберем несколько новых понятий: «переменная», «операция», «оператор», «операнд», «выражение», «конкатенация».

Переменная — это *объект* (реализуемый как именованная область памяти), который может принимать различные значения. Название переменной начинается с одной или нескольких латинских букв (например, *b*, *sum*), может содержать цифры и знаки подчеркивания (например, *num1*, *num_2*, *num_3_1* и т. д.). При этом имена переменных в Python чувствительны к регистру (например, *Number*, *NUMBER*, *number* — это три различные переменные).

Операция — это некоторое *действие*, которое необходимо совершить над числами и/или переменными (например, *сложение*, *вычитание*, *умножение*, *деление* и т. д.).

Оператор — это объект (символ), который *выполняет операцию* и имеет привычную символьную запись (например, +, -, *, /).

Операнд — это объект (число, символ, строка или переменная), *над которым* оператор выполняет операцию.

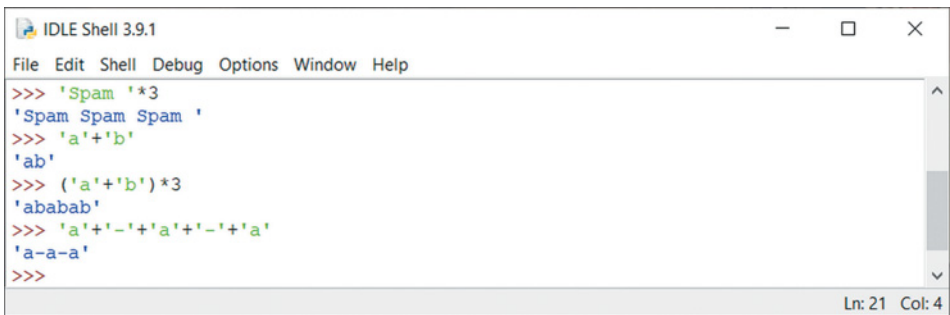
Таким образом, **выражение** — совокупность *операций*, которые выполняются *операторами* над *операндами*.

Например:

$b + 5$ — выражение;
сложение — операция;
 + — оператор;
 $b, 5$ — операнды.

Конкатенация строк — операция *присоединения*, «склеивания» символов или их наборов.

Операция конкатенации строк возможна не только с помощью оператора *, но и с помощью оператора +. Также их можно использовать вместе (рис. 6).



```

IDLE Shell 3.9.1
File Edit Shell Debug Options Window Help
>>> 'Spam '*3
'Spam Spam Spam '
>>> 'a'+ 'b'
'ab'
>>> ('a'+ 'b') *3
'ababab'
>>> 'a'+ '-' + 'a'+ '-' + 'a'
'a-a-a'
>>>
Ln: 21 Col: 4
  
```

Рис. 6. Пример конкатенации строк с помощью операторов * и +

Эксперименты с кавычками

Уделим особое внимание одинарным и двойным кавычкам, которые используются при выводе строк, через эксперимент в интерактивном режиме:

```

>>> 'Одинарные кавычки уже были рассмотрены нами'
'Одинарные кавычки уже были рассмотрены нами'
>>> "Как насчет двойных? - Ого! Так тоже работает!"
'Как насчет двойных? - Ого! Так тоже работает!'
>>> "Python считал наши двойные кавычки, но вывел одинарные...xm"
'Python считал наши двойные кавычки, но вывел одинарные...xm'
  
```

```
>>> "Хотим двойные кавычки! Python, выведи "Москва". По-  
жалуйста!"
```

```
SyntaxError: invalid syntax
```

```
>>> 'Москва не подсвечивается зеленым цветом. А это зна-  
чит, что Python видит только строки между одинарными ка-  
вычками или между двойными. "Попробуем совместить"'  
'Москва не подсвечивается зеленым цветом. А это значит,  
что Python видит только строки между одинарными кавычка-  
ми или между двойными. "Попробуем совместить"'
```

```
>>> "Получилось! А если 'внутри' предложения взять оди-  
нарные кавычки, а снаружи - двойные?"  
"Получилось! А если 'внутри' предложения взять одинарные  
кавычки, а снаружи - двойные?"
```

```
>>> 'А если все одинарные'?':'  
SyntaxError: invalid syntax
```

```
>>> "No : ("  
'No : ('
```

```
>>> 'ВЫВОД'  
'ВЫВОД'
```

```
>>> "ВЫВОД"  
'ВЫВОД'
```

```
>>> "ВЫВОД'вывод'ВЫВОД"  
"ВЫВОД'вывод'ВЫВОД"
```

```
>>> 'вывод"ВЫВОД"вывод'  
'вывод"ВЫВОД"вывод'
```

Вывод:

- Чтобы вывести строку, не содержащую текста с кавычками, можно использовать как одинарные, так и двойные кавычки.
- Если необходимо вывести строку, содержащую текст с кавычками, внешние и внутренние кавычки *не* должны совпадать.

В некоторых языках программирования одинарные и двойные кавычки предназначены для разных целей. Python же позволяет использовать оба варианта, но строка обязательно должна начинаться и заканчиваться одним и тем же типом кавычек.

Говорят, что бывают тройные одинарные и тройные двойные кавычки. Попробуем! Эксперимент продолжается:

```
>>> """Попробуем просто написать что-нибудь в тройных
двойных кавычках"""
```

```
'Попробуем просто написать что-нибудь в тройных двойных
кавычках'
```

```
>>> '''А теперь в тройных одинарных'''
```

```
'А теперь в тройных одинарных'
```

```
>>> """Говорят, что у тройных кавычек есть своя особен-
ность...
```

```
Кстати, почему мысли растекаются на такие большие пред-
ложения, что даже в одну строку не вмещаются?"""
```

```
'Говорят, что у тройных кавычек есть своя особенность...
\nКстати, почему мысли растекаются на такие большие
предложения, что даже в одну строку не вмещаются?'
```

```
>>> "Ребята, вы видели?! Перед словом 'Кстати' была на-
жата кнопка Enter, а Python записал все в одну строку"
```

```
"Ребята, вы видели?! Перед словом 'Кстати' была нажата
кнопка Enter, а Python записал все в одну строку"
```

```
>>> "Может, каждый перенос строки обозначается \n? Про-
верим:
```

```
SyntaxError: EOL while scanning string literal
```

```
>>> 'Ах да! Нужно не забывать закрывать кавычки. Итак,
проверяем:'
```

```
'Ах да! Нужно не забывать закрывать кавычки. Итак, про-
веряем:'
```

```
>>> """abra
```

```
kadabra
```

```
dabra
```

```
badra
```

```
bodra
```

```
bobra
```

```
dobra!"""
```

```
'abra\nkadabra\ndabra\nbadra\nbodra\nbobra\ndobra!'
```

```
>>> 'Значит, так и есть. С тройными одинарными так же?'
```

```
'Значит, так и есть. С тройными одинарными так же?'
```

```
>>> '''  
poly  
moly  
usy  
pusy'''  
'poly\nmoly\nusy\npusy'
```

```
>>> 'Хорошо, тут то же самое. Так чем же отличаются оди-  
нарные и двойные от тройных одинарных и тройных двойных  
кавычек?'
```

```
'Хорошо, тут то же самое. Так чем же отличаются одинар-  
ные и двойные от тройных одинарных и тройных двойных  
кавычек?'
```

```
>>> 'Между прочим, хотелось сейчас продолжить писать на  
другой строке, но после нажатия Enter строка с одинар-  
ными кавычками была моментально выполнена Python. Это  
значит, что только с тройными кавычками можно переходить  
на другую строку при нажатии клавиши Enter. Вот и разо-  
брались.'
```

```
'Между прочим, хотелось сейчас продолжить писать на дру-  
гой строке, но после нажатия Enter строка с одинарными  
кавычками была моментально выполнена Python. Это зна-  
чит, что только с тройными кавычками можно переходить на  
другую строку при нажатии клавиши Enter. Вот и разобр-  
лись.'
```

Вывод:

Тройные одинарные и тройные двойные кавычки позволяют де-
лать перенос строки с помощью клавиши **Enter**, который после
выполнения инструкции обозначается *специальным символом* `\n`.

Арифметические операторы

История первого компьютера рассказывает о том, что изначально компьютер был создан как большая вычислительная машина и только спустя некоторое время появились дополнительные устройства ввода и вывода информации (монитор, клавиатура, мышь и т. д.).

Поэтому использование Python в качестве калькулятора — базовая возможность, и реализуется она с помощью арифметических операторов:

| | | | | | | |
|---|---|---|---|----|----|---|
| + | - | * | / | ** | // | % |
|---|---|---|---|----|----|---|

Ниже представлены некоторые варианты вычислений с различными видами чисел (натуральные, целые, дробные) в интерактивном режиме:

| + | - | * | / |
|---------------------|-----------------------|-----------------------|--------------------|
| >>>2+5 7 | >>>2-5 -3 | >>>2*5 10 | >>>2/5 0.4 |
| >>>-2+5 3 | >>>-2-5 -7 | >>>-2*5 -10 | >>>-2/5 -0.4 |
| >>>2+(-5.0) -3.0 | >>>2-(-5.0) 7.0 | >>>2*(-7.2) -14.4 | >>>20/-4 -5.0 |
| >>>2+-5 -3 | >>>2--5 7 | >>>2*-5 -10 | >>>4.6/2.3 2.0 |
| >>>2+2.3 4.3 | >>>2.3-2 0.3 | >>>-2*2.3 -4.6 | >>>-5/25 -0.2 |
| >>>5.2+3.03 8.23 | >>>-5.2-3.03 -8.23 | >>>5.2*3.03 15.756 | >>>-24/-25 0.96 |

Важно!

- Все дробные числа записаны с точкой, а не с запятой.
- В отличие от привычной математики, в Python могут ставиться два арифметических знака подряд. Например, $2+-5$. Первый знак будет считываться как операция, второй — как знак числа.

В общем случае Python различает приоритет выполнения операций:

- 1) унарные операции (унарный минус);
- 2) операции умножения и деления;
- 3) операции сложения и вычитания.

Важно!

Для изменения порядка выполнения используются круглые скобки.

Рассмотрим следующую таблицу с менее знакомыми операторами:

| ** | // | % |
|----------------|-----------------|---------------|
| >>>2**5 32 | >>>2//5 0 | >>>2%5 2 |
| >>>-3**2 9 | >>>16//13 1 | >>>16%13 3 |
| >>>-1**9 -1 | >>> -5//2 -3 | >>>-20%7 1 |

Оператор ****** отвечает за операцию возведения в степень. Таким образом, число слева от оператора ****** — это *основание степени*, а число справа — *показатель степени*:

$$\begin{aligned}
 2 ** 5 &= 32, & \text{ так как } 2^5 &= 32; \\
 -3 ** 2 &= 9, & \text{ так как } (-3)^2 &= 9; \\
 -1 ** 9 &= -1, & \text{ так как } (-1)^9 &= -1.
 \end{aligned}$$

Оператор **//** отвечает за целую часть при делении первого числа на второе и округляет результат до ближайшего наименьшего целого числа¹:

$$\begin{aligned}
 2 // 5 &= 0, & \text{ так как } \frac{2}{5} &= 0\frac{2}{5} \text{ (ближайшие целые числа } \\
 & & & \text{ 0 и 1, 0 — наименьшее);} \\
 16 // 13 &= 1, & \text{ так как } \frac{16}{13} &= 1\frac{3}{13} \text{ (ближайшие целые числа } \\
 & & & \text{ 1 и 2, 1 — наименьшее);} \\
 -5 // 2 &= -3, & \text{ так как } -\frac{5}{2} &= -2\frac{1}{2} \text{ (ближайшие целые числа } \\
 & & & \text{ -2 и -3, -3 — наименьшее).}
 \end{aligned}$$

¹ До 3-й версии языка при делении целых чисел с помощью оператора / тоже отбрасывалась дробная часть ($2/4 = 0$, $2//4 = 0$). Разницу можно было увидеть только при делении целых и дробных чисел ($2.0/4 = 0.5$, $2.0//4 = 0.0$). Затем разработчики перераспределили роли данных операторов, и за отбрасывание дробной части стал отвечать только оператор //.

Оператор `%` отвечает за остаток при делении первого числа на второе и выводит количество единиц (остаток) до ближайшего наименьшего целого числа:

$2 \% 5 = 2$, так как $2 : 5 = 0$ (ост. 2) (2 единицы до наименьшего целого числа 0);
 $16 \% 13 = 3$, так как $16 : 13 = 1$ (ост. 3) (3 единицы до наименьшего целого числа 1);
 $-20 \% 3 = 1$, так как $-20 : 3 = (-21+1) : 3 = -7$ (ост. 1) (1 единица до наименьшего целого числа -7).

Экспериментируйте с числами, чтобы проверить, правильно ли вы поняли ту или иную новую операцию.

Разработка программ в IDLE

Обратной стороной интерактивного режима является неудобство в создании многострочных программ. Поэтому пришло время поговорить о втором, самом используемом способе работы с IDLE — создании отдельных файлов, что позволяет многократно обращаться, исправлять, дополнять и снова запускать код для получения и проверки результата.

Итак, первой программой в интерактивном режиме был вывод строки «Hello, world!» с помощью оператора `print()`. Создадим файл для нашей первой программы.

1. В верхнем поле окна IDLE выберем **File** → **New File** (рис. 7).

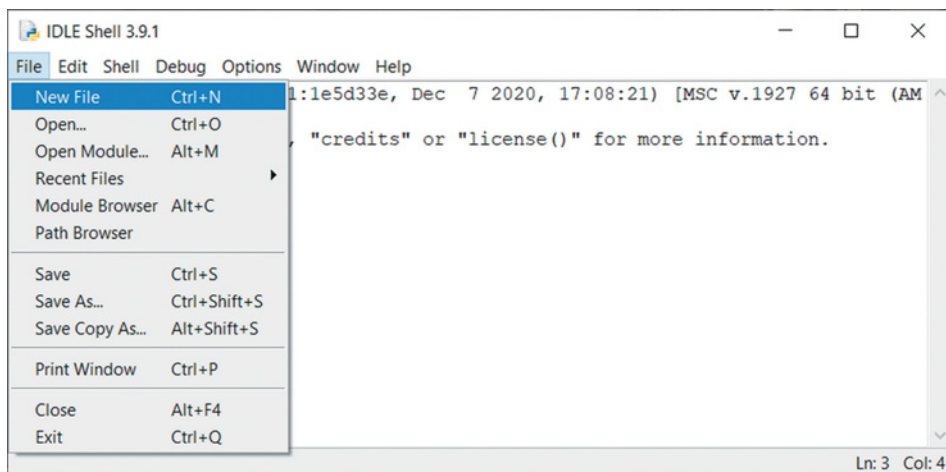


Рис. 7. Создание нового файла в IDLE

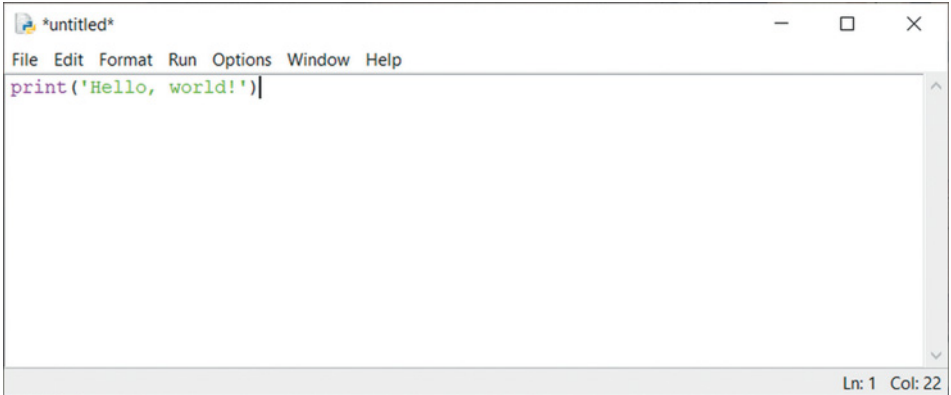


Рис. 8. Ввод первой программы «Hello, world!» в новом файле

2. Введем текст первой программы в открывшемся окне (рис. 8):

```
print('Hello, world!')
```

3. Сохраним файл в выбранной нами папке с названием «HelloWorld». Для этого нажмем **File** → **Save As** (рис. 9), в открывшемся окне выберем необходимую для сохранения наших программ папку, впишем название «HelloWorld» и кликнем на кнопку **Сохранить** (рис. 10).

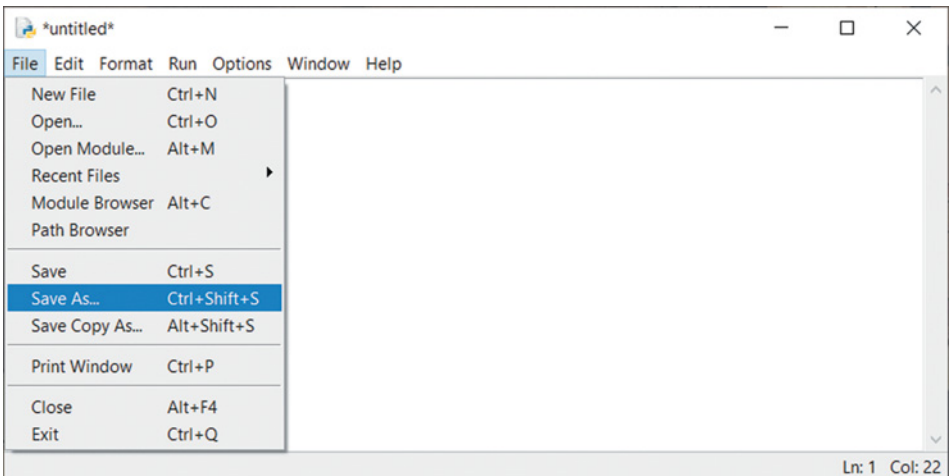


Рис. 9. Сохранение файла в IDLE

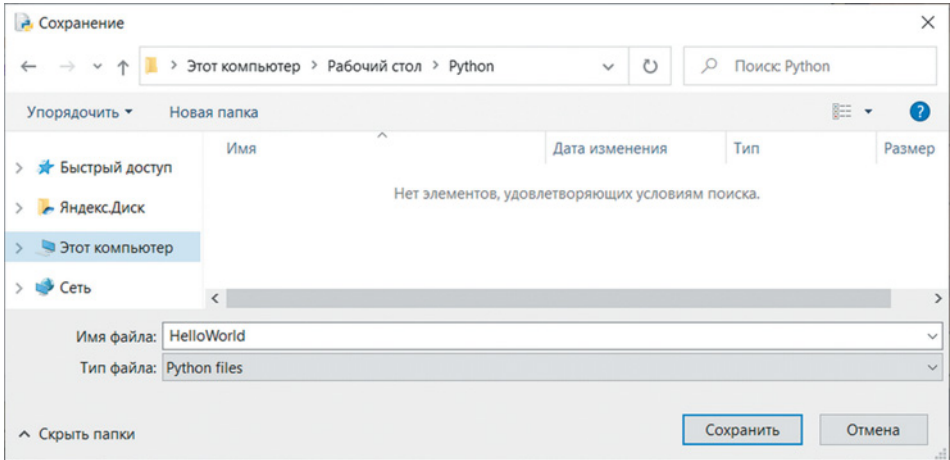


Рис. 10. Сохранение файла в IDLE

Название файла «HelloWorld.py» и путь до него можно видеть в заголовке окна IDLE после сохранения.

Чтобы запустить программу, необходимо выбрать **Run** → **Run Module** или нажать клавишу **F5** (рис. 11).

После запуска программы результат ее выполнения отображается в интерактивной оболочке IDLE (рис. 12).

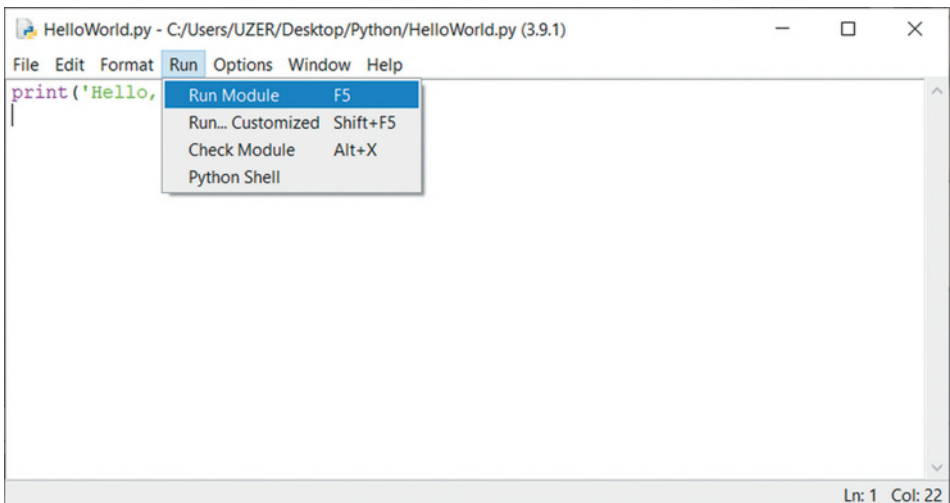
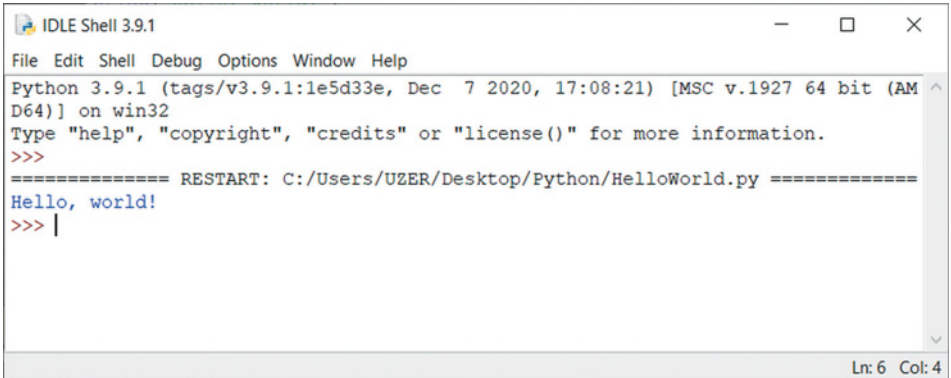


Рис. 11. Запуск программы из файла в IDLE



```
IDLE Shell 3.9.1
File Edit Shell Debug Options Window Help
Python 3.9.1 (tags/v3.9.1:1e5d33e, Dec 7 2020, 17:08:21) [MSC v.1927 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
===== RESTART: C:/Users/UZER/Desktop/Python/HelloWorld.py =====
Hello, world!
>>> |
Ln: 6 Col: 4
```

Рис. 12. Результат выполнения файла HelloWorld.py

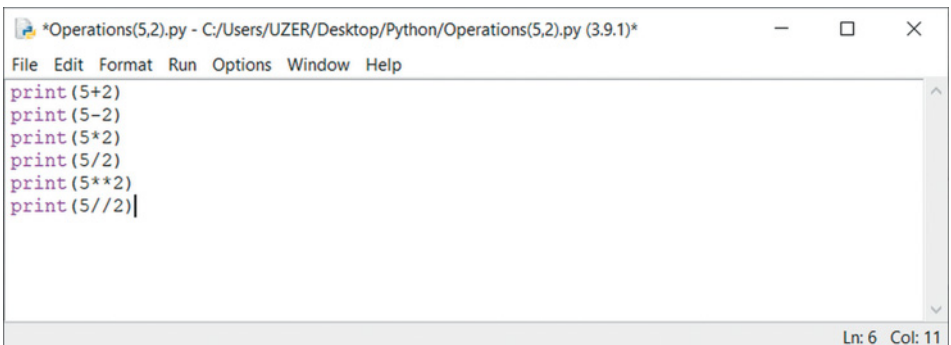
Важно!

В файле инструкция для вывода на экран без оператора `print()` не выполняется.

Первая многострочная программа

Напишем программу проверки изученных арифметических операций для чисел 2 и 5.

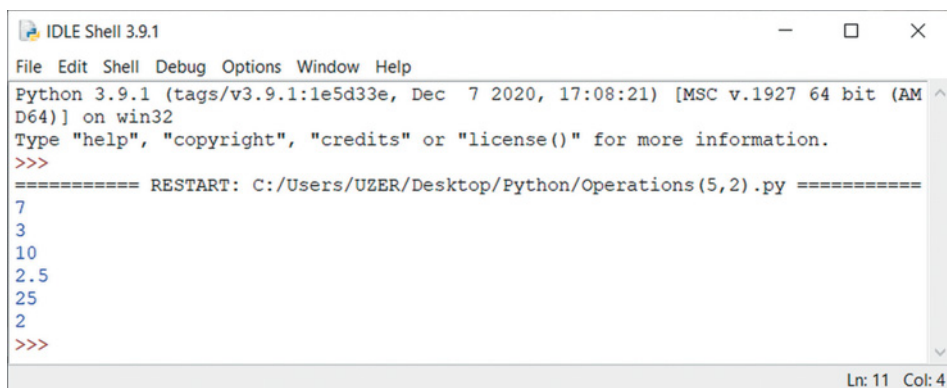
Для этого создадим файл (**File** → **New File**) и сохраним его в необходимую папку с названием «Operations(5,2)». С помощью оператора `print()` выведем на экран результаты применения операторов `+`, `-`, `*`, `/`, `**`, `//` для чисел 5 и 2 (рис. 13).



```
*Operations(5,2).py - C:/Users/UZER/Desktop/Python/Operations(5,2).py (3.9.1)*
File Edit Format Run Options Window Help
print(5+2)
print(5-2)
print(5*2)
print(5/2)
print(5**2)
print(5//2)
Ln: 6 Col: 11
```

Рис. 13. Текст программного файла Operations(5,2).py

Запускаем программу (**Run** → **Run Module** или **F5**) (рис. 14).



```
IDLE Shell 3.9.1
File Edit Shell Debug Options Window Help
Python 3.9.1 (tags/v3.9.1:1e5d33e, Dec 7 2020, 17:08:21) [MSC v.1927 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
===== RESTART: C:/Users/UZER/Desktop/Python/Operations(5,2).py =====
7
3
10
2.5
25
2
>>>
```

Рис. 14. Результат выполнения Operations(5,2).py

Ключи оптимизации

Для начала определим, что под шагами оптимизации программы мы будем понимать:

- увеличение спектра решаемых ею задач (усиление универсальности);
- понятный диалог с пользователем;
- предусмотрение различного рода ошибок пользователя;
- выбор более компактного варианта кода;
- увеличение скорости работы с кодом;
- увеличение скорости выполнения программы.

На данный момент программа Operations(5,2).py может производить арифметические операции только с числами 2 и 5. Поэтому *первый шаг* оптимизации нашей программы — объявление переменных a и b , которым можно будет приписывать другие значения. Таким образом, программа решает одну и ту же задачу, но уже дает возможность программисту вводить новые данные не в шести строках, а только в первых двух и тем самым увеличивает скорость получения результатов.

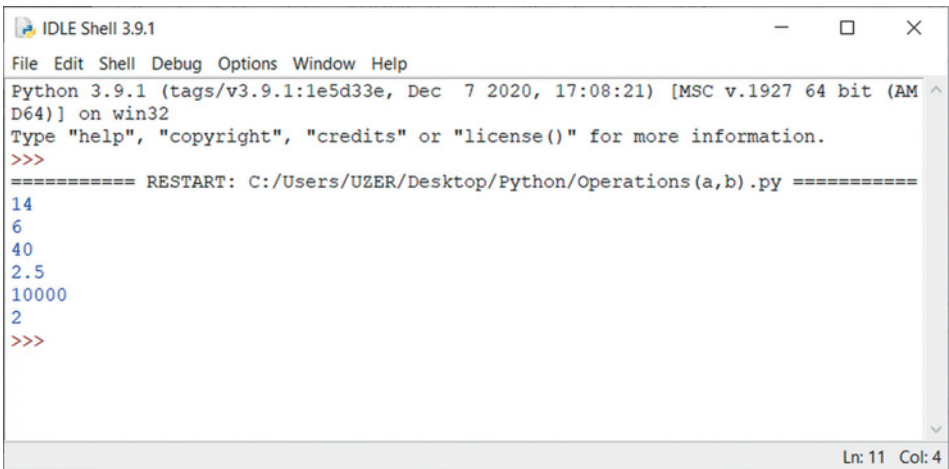
Создадим новый файл Operations(a,b).py и запишем инструкцию:

```
a = 5
b = 2
```

```
print(a+b)
print(a-b)
print(a*b)
print(a/b)
print(a**b)
print(a//b)
```

После запуска программы результат оказывается тем же, что и при запуске `Operations(5,2)` (см. рис. 14).

Пробуем ввести другие числа: меняем в коде значения a и b на 10 и 4 соответственно, сохраняем и запускаем программу (рис. 15).



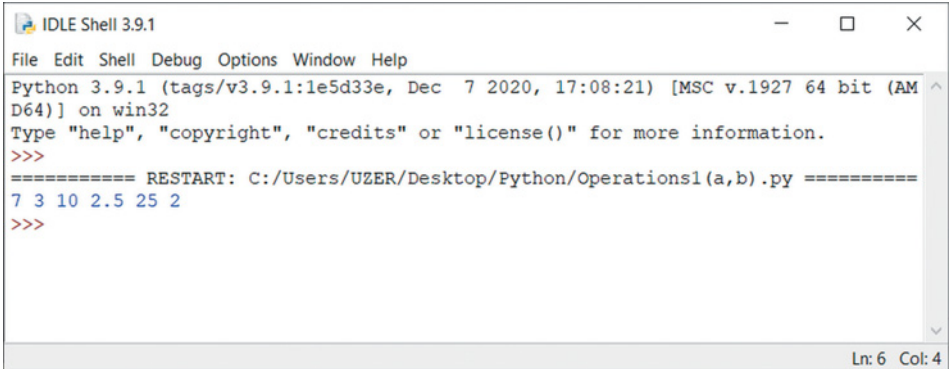
```
IDLE Shell 3.9.1
File Edit Shell Debug Options Window Help
Python 3.9.1 (tags/v3.9.1:1e5d33e, Dec 7 2020, 17:08:21) [MSC v.1927 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
===== RESTART: C:/Users/UZER/Desktop/Python/Operations(a,b).py =====
14
6
40
2.5
10000
2
>>>
```

Рис. 15. Результат выполнения `Operations(a,b).py` с новыми значениями

Приступаем ко *второму шагу* оптимизации программы. Чтобы вывести подряд результаты арифметических операций, вовсе не обязательно каждый раз прописывать оператор `print()`. Для этого достаточно написать его единожды, а все то, что мы хотим вывести, прописать внутри скобок через запятую. Таким образом наш код становится более компактным.

Меняем текст инструкции, сохраняем файл с новым названием `Operations1(a,b).py` и запускаем (рис. 16).

```
a=5
b=2
print(a+b,a-b,a*b,a/b,a**b,a//b)
```



```

IDLE Shell 3.9.1
File Edit Shell Debug Options Window Help
Python 3.9.1 (tags/v3.9.1:1e5d33e, Dec 7 2020, 17:08:21) [MSC v.1927 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
===== RESTART: C:/Users/UZER/Desktop/Python/Operations1(a,b).py =====
7 3 10 2.5 25 2
>>>
Ln: 6 Col: 4

```

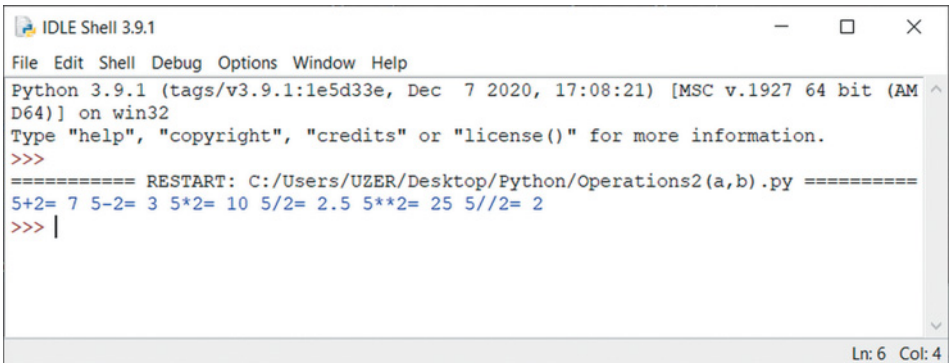
Рис. 16. Результат выполнения Operations1(a,b).py

Сделаем *третий шаг* оптимизации программы. Важно, чтобы пользователь понимал, за что отвечают выведенные на экран числа. Поэтому добавим через запятую строки с выражениями. Сохраним файл с названием Operations2(a,b).py и запустим программу (рис. 17):

```

a=5
b=2
print('5+2=',a+b,'5-2=',a-b,'5*2=',a*b,'5/2=',
a/b,'5**2=',a**b,'5//2=',a//b)

```



```

IDLE Shell 3.9.1
File Edit Shell Debug Options Window Help
Python 3.9.1 (tags/v3.9.1:1e5d33e, Dec 7 2020, 17:08:21) [MSC v.1927 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
===== RESTART: C:/Users/UZER/Desktop/Python/Operations2(a,b).py =====
5+2= 7 5-2= 3 5*2= 10 5/2= 2.5 5**2= 25 5//2= 2
>>> |
Ln: 6 Col: 4

```

Рис. 17. Результат выполнения Operations2(a,b).py

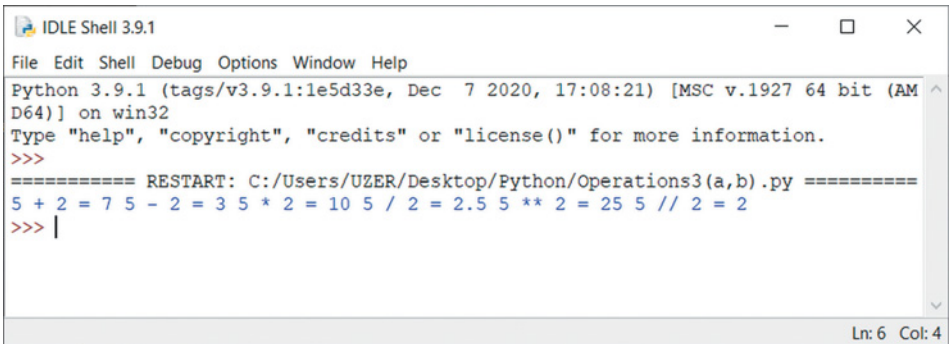
Для конкретных чисел такой вариант, может, и подходит, но в нашем случае a и b мы можем менять. Поэтому выполним вывод на экран чисел, которые будут записаны в переменные a и b . Меняем инструкцию, сохраняем файл Operations3(a,b).py и запускаем (рис. 18):

```

a=5
b=2

```

```
print(a, '+', b, '=', a+b, a, '-', b, '=', a-b,
      a, '*', b, '=', a*b, a, '/', b, '=', a/b, a, '**', b, '=', a**b,
      a, '//', b, '=', a//b)
```



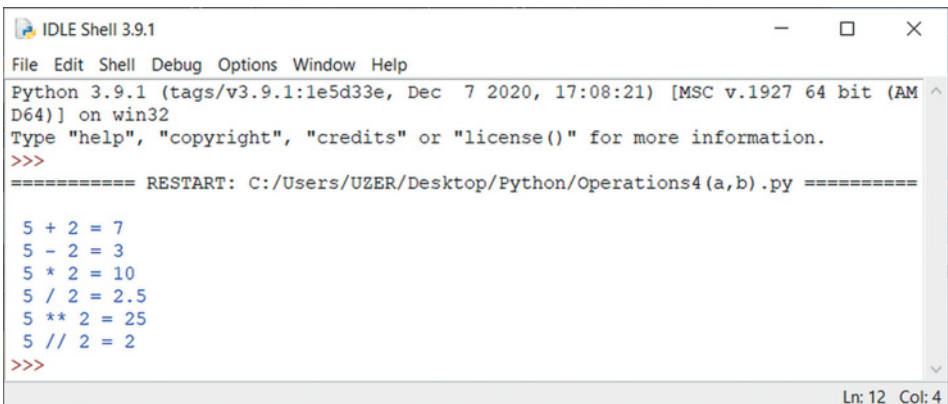
```
IDLE Shell 3.9.1
File Edit Shell Debug Options Window Help
Python 3.9.1 (tags/v3.9.1:1e5d33e, Dec 7 2020, 17:08:21) [MSC v.1927 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
===== RESTART: C:/Users/UZER/Desktop/Python/Operations3(a,b).py =====
5 + 2 = 7 5 - 2 = 3 5 * 2 = 10 5 / 2 = 2.5 5 ** 2 = 25 5 // 2 = 2
>>> |
Ln: 6 Col: 4
```

Рис. 18. Результат выполнения Operations3(a,b).py

Выглядит неплохо, но еще лучше, если результат каждой операции будет виден с новой строки. Мы уже знаем, что переход на новую строку в Python обозначается `\n`. Добавим перед каждым арифметическим выражением эту пару символов, сохраним файл Operations4(a,b).py и запустим его (рис. 19):

```
a=5
b=2
print('\n', a, '+', b, '=', a+b, '\n', a, '-', b, '=', a-b,
      '\n', a, '*', b, '=', a*b, '\n', a, '/', b, '=', a/b,
      '\n', a, '**', b, '=', a**b, '\n', a, '//', b, '=', a//b)
```

Теперь можно менять в коде значения *a* и *b*, запускать программу многократно и видеть результат выполнения операций.



```
IDLE Shell 3.9.1
File Edit Shell Debug Options Window Help
Python 3.9.1 (tags/v3.9.1:1e5d33e, Dec 7 2020, 17:08:21) [MSC v.1927 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
===== RESTART: C:/Users/UZER/Desktop/Python/Operations4(a,b).py =====

5 + 2 = 7
5 - 2 = 3
5 * 2 = 10
5 / 2 = 2.5
5 ** 2 = 25
5 // 2 = 2
>>>
Ln: 12 Col: 4
```

Рис. 19. Результат выполнения Operations4(a,b).py

Что нового мы узнали в первой главе?

1. **Программа** — это инструкция или последовательность действий для машины, а процесс ее написания — *программирование*.

2. **IDLE** — это среда, которая позволяет просматривать, редактировать, запускать, производить отладку программ на языке Python.

3. **Интерпретатор Python** — специальный модуль, который позволяет процессору считывать команды, записанные на языке программирования, и исполнять их, опуская промежуточный этап целостной сборки, в отличие от других языков программирования.

4. Существует два вида работы в IDLE.

- **Интерактивный**

Сеанс в IDLE начинается с вывода двух строк информационного текста, затем выводится приглашение к вводу команды `>>>`.

Ввод каждой инструкции завершается нажатием клавиши **Enter**, после чего Python выполняет введенную операцию и выдает результат или сообщение об ошибке.

В интерактивном режиме можно вводить любое число команд, и каждая из них будет выполняться сразу же после ввода. Такой тип работы также называют *работой в интерактивной оболочке*.

- **С помощью отдельного файла**

Создание **File** → **New File**, сохранение **File** → **Save As**, запуск **Run** → **Run Module** или **F5**.

5. С помощью функции **print()** осуществляется вывод информации на используемый монитор (в частном случае на дисплей).

Интерактивная оболочка позволяет не использовать функцию `print()`.

6. Последовательность символов, заключенная в кавычки, называется *строкой* (не путать со строчкой в тексте) и является *неизменяемым типом данных*.

7. **Выражение** — совокупность *операций*, которые выполняются *операторами* над *операндами*.

- **Переменная** — это *объект*, который может принимать различные значения. Название переменной начинается с од-

ной или нескольких латинских букв (например, *b*, *sum*), может содержать цифры и знаки подчеркивания (например, *num1*, *num_2*, *num_3_1* и т. д.). При этом имена переменных в Python чувствительны к регистру (например, *Number*, *NUMBER*, *number* — это три различные переменные).

- **Операция** — это некоторое *действие*, которое необходимо совершить над числами и/или переменными (например, *сложение*, *вычитание*, *умножение*, *деление* и т. д.).

- **Оператор** — это объект (символ), который *выполняет операцию* и имеет привычную символьную запись (например, *+*, *-*, ***, */*).

- **Операнд** — это объект (число, символ, строка или переменная), *над которым* оператор выполняет операцию.

8. Конкатенация строк — операция присоединения, «склеивания» символов или их наборов с помощью операторов *** и *+*.

9. Использование одинарных, двойных и тройных кавычек

- Чтобы вывести строку, не содержащую текста с кавычками, можно использовать как одинарные, так и двойные кавычки.

- Если необходимо вывести строку, содержащую текст с кавычками, внешние и внутренние кавычки *не* должны совпадать.

- Тройные одинарные и тройные двойные кавычки позволяют делать перенос строки с помощью клавиши **Enter**, который после выполнения инструкции обозначается специальным символом `\n`.

10. Арифметические операторы и операции

- Обозначения:

- +* — сложение;
- — унарный минус, вычитание;
- ** — умножение;
- /* — деление;
- *** — возведение в степень;
- //* — целая часть от деления;
- %* — остаток от деления.

- Все дробные числа записываются с точкой, а не с запятой.
- В отличие от привычной математики, в Python могут ставиться два арифметических знака подряд. Например, `2+-5`.

- Python различает приоритет выполнения операций:
 - 1) унарные операции (унарный минус);
 - 2) операции умножения и деления;
 - 3) операции сложения и вычитания.

Для изменения порядка выполнения используются круглые скобки.

11. Под шагами оптимизации программы в данной книге будем понимать:

- увеличение спектра решаемых ею задач (усиление универсальности);
- понятный диалог с пользователем;
- предусмотрение различного рода ошибок пользователя;
- выбор более компактного варианта кода;
- увеличение скорости работы с кодом.

Практикум

Основные понятия главы

1. Сопоставьте элементы соответствующим им значениям:

| | |
|-----------------|-----------|
| $7 + c$ | оператор |
| $c, 7$ | выражение |
| $+$ | операнды |
| <i>сложение</i> | операция |

2. Заполните пропуски в таблице, используя элементы «*выражение*», «*операции*», «*операторы*», «*операнды*»:

| | |
|--|--|
| $(p * 2) \% 3$ | |
| $p, 2, 3$ | |
| $*, \%$ | |
| <i>умножение, остаток от деления</i> | |

3. Заполните пропуски в таблице, используя элементы «выражение», «операторы», «возведение в степень, целая часть от деления», «a, 13//2, 13, 2»:

| | |
|-----------------|-----------------|
| $a^{**}(13//2)$ | |
| | <i>операнды</i> |
| $**$, $//$ | |
| | <i>операции</i> |

4. Составьте инструкцию для Python, выводящую на экран строки:

4.1) "Корабль 'Лавр' лавировал"

4.2) 'Она сказала:"Топ-топ"Она сказала:"Топ-топ"'

4.3) '\nПривет!\nКак дела?\n\n:)'

5. Составьте инструкцию для Python, содержащую оба оператора конкатенации и выводящую на экран строку 'ababababab'.

6. Перечислите возможные варианты инструкций для интерпретатора Python, при которых конкатенация выполняется одним оператором *, а результат получается следующим: '1111111111'.

Чтение кода

7. Вычислите без среды IDLE выражения:

| № | Выражение | Результат |
|-----|-------------------------------|-----------|
| 7.1 | $((2**5)+5)//5$ | |
| 7.2 | $((13\%2)-1.0)/5$ | |
| 7.3 | $((8//5)**2)-10.2$ | |
| 7.4 | $((((2+10.2)-0.2)**2)//4)\%2$ | |

8. Выберите правильный порядок операций в выражении

$$2*a+b*c:$$

- *, +, *
- +, *, *
- *, *, +

9. Выберите правильный порядок операций в выражении

$(d+2*30)-6//2$:

- //, -, *, +
- *, +, //, -
- //, *, +, -

10. Выберите правильный порядок операций в выражении

$a\%2+(1-6//6)$:

- %, +, -, //
- -, %, //, +
- //, -, %, +

Поиск ошибок

11. При вводе каких инструкций интерпретатор Python выдаст ошибку?

- $(a * 5)$
- $(3 * 9)$
- 'hi'
- "Hello"

12. При вводе какой инструкции интерпретатор Python выдаст ошибку?

- "Привет"Лена"
- "55" + "11"
- "'Little'"
- ""Привет,
Как дела?""

13. При вводе каких инструкций интерпретатор Python выдаст ошибку?

- $2 ** 5$
- $6 = a$
- 'hi' - 'i'
- "Hello" * 2

14. При вводе каких инструкций интерпретатор Python выдаст ошибку?

- "Привет" + ",Лена"
- "55" * '11'
- "'Little'"
- $5 / 0$

15. При вводе какой инструкции интерпретатор Python выдаст ошибку?

- `((a * 5))`
- `12,5 * 2`
- `'hi' / 2`
- `1.2 + 13.0`

16. При вводе каких инструкций интерпретатор Python *не* выдаст ошибку?

- `"Сегодня в 'Москве' прекрасная погода"`
- `"Сегодня в "Москве" прекрасная погода"`
- `'Сегодня в "Москве" прекрасная погода'`
- `'Сегодня в 'Москве' прекрасная погода'`

17. При вводе каких инструкций интерпретатор Python *не* выдаст ошибку?

- `""Как дела?" - спросила Оля."`
- `""Как дела?" - спросила Оля."`
- `""Как дела?" - спросила Оля." * 2`
- `""Как дела?" - спросила Оля." * 2`

18. При вводе какой инструкции интерпретатор Python выдаст ошибку?

- `"Пока"`
- `X = 'n' * 2`
- `2 + 6x = 7`
- `5.0`

Оптимизация

19. Приведенный ниже код решает задачу нахождения периметра треугольника со сторонами 3, 4 и 5:

```
a = 3
b = 4
c = 5
P = a + b + c
print(P)
```

19.1) Произведите замену первых трех строк программы на:

```
a = int(input('Введите сторону a: '))
b = int(input('Введите сторону b: '))
c = int(input('Введите сторону c: '))
```

Сохраните программу в файле и протестируйте ее для чисел 7, 9 и 13.

Важно!

Оператор `input()` является одним из вариантов увеличения спектра решаемых задач и будет подробно рассмотрен в следующей главе.

19.2) Предложите свой вариант более компактного кода.

Разработка

20. Напишите программу, которая выводит на экран дату вашего рождения.

Пример:

Дата моего рождения 18 апреля.

21. Напишите программу, которая выводит на экран не только результаты вычисления следующих выражений, но и сами выражения.

| № | Выражение | Результат |
|------|-----------------------------|-----------|
| 21.1 | $-365 \cdot 20$ | |
| 21.2 | $0.5 \cdot 650 - 25$ | |
| 21.3 | $(132 : 2 + 1) \cdot 6$ | |
| 21.4 | $-576 + (23 \cdot 45 - 13)$ | |

22. Напишите программу, которая выводит ваше имя, где каждая согласная написана с помощью символов «0», а каждая гласная — с помощью символов «*». Если ваше имя содержит мягкий или твёрдый знак, выведите его с помощью произвольного символа.

Пример:

```
0000  ****  0  0  ****
0      *  *  0 0 0  *  *
0      ****  0 0 0  ****
0000  *  *  00000  *  *
```

23. Напишите программу, которая выводит домик на фоне звезд.

Пример:

```

*****
*           +           *
*         / \         + *
*       /   \       + *
*     +-----+     *
*     | +---+ |     *
*     | | | | |     +
*   ___|___|___|___ *
*****

```

24. Напишите программу, вычисляющую площадь квадрата, сторону которого вводит пользователь с клавиатуры. Для запроса чисел используйте конструкцию задания 19.1.

25. Напишите программу, которая запрашивает целое число и выводит следующее за ним. Для запроса чисел используйте конструкцию задания 19.1.

Пример:

```

Введите число: 15
Следующее число: 16

```

26. Напишите программу, которая запрашивает два целых числа и выводит их сумму. Для запроса чисел используйте конструкцию задания 19.1.

Пример:

```

Первое число: 15
Второе число: 13
Сумма чисел: 28

```

27. Напишите программу, которая запрашивает два целых числа и выводит их произведение. Для запроса чисел используйте конструкцию задания 19.1.

Пример:

```

Первое число: 90
Второе число: 8
90 * 8 = 720

```

28. Напишите программу, которая запрашивает три целых числа и выводит сумму первого и второго чисел, разность второго

и третьего чисел, произведение первого и третьего чисел. Для запроса чисел используйте конструкцию задания 19.1.

Пример:

Первое число: -5

Второе число: 8

Третье число: 0

$-5 + 8 = 3$

$8 - 0 = 8$

$-5 * 0 = 0$

29. Напишите программу, которая запрашивает два целых числа, где:

- первое число будет количеством символов «о» в результате выполнения программы;
- второе число будет количеством символов «м» в результате выполнения программы;
- последняя строка программы подсчитывает суммарное количество символов.

Для запроса чисел используйте конструкцию задания 19.1.

Пример:

Количество 'о' : 5

Количество 'м' : 5

оооооммммм

Общее количество символов: 10

Глава 2. Переменные, типы данных. Функции `input()` и `eval()`

Информация, которая необходима машине для вычислений, преобразований и диалога с пользователем, хранится в памяти компьютера. Память компьютера представляет собой схожую с фотоленкой структуру, т. е. состоит из отдельных фрагментов (ячеек памяти).

Python лишь наполняет ячейки данными, вводимыми программистом, а после обращается к ним по ссылке. (Совсем как человек обращается к кадру фотоленки, чтобы открыть «коробочку» с воспоминаниями.) Программисту же дается право не только вводить данные в память компьютера, но и называть «коробочки» уникальным образом, чтобы обращаться к ним наиболее удобным для него способом.

Будем называть «коробочку», фрагмент памяти компьютера, который имеет уникальное имя, переменной.

Понятие переменной

Итак, *переменная* — это *именованная область памяти компьютера*, а также «коробочка», которая хранит *данные определенного типа*. Такие данные называются *значением переменной*.

Стоит отметить, что при необходимости можно менять как само значение переменной, так и его тип. Подобные преобразования будут рассмотрены позже.

При объявлении переменной в Python указываются всего две части:

- имя переменной (*как* назвать область памяти компьютера);
- значение переменной (*чем* заполнить область памяти компьютера).

Имя переменной не содержит пробелов, всегда начинается с латинской буквы, а остальные символы могут быть буквами, цифрами или нижним подчеркиванием «`_`». При этом имена переменных в Python чувствительны к регистру, как упоминалось ранее (например, `Number`, `NUMBER`, `number` — это три различные переменные).

Для того чтобы записать в переменную значение, необходим *оператор присваивания*, который обозначается символом «`=`»

и выполняет *операцию присваивания*, т. е. выступает в роли «моста» перемещения данных в отведенную область памяти.

| | |
|----------|---------------------|
| оператор | = |
| операция | <i>присваивание</i> |

При выполнении инструкции « $a = 5$ » целое число 5 помещается в переменную с именем « a ». Другими словами, выражение « $a = 5$ » всегда читается компьютером справа налево: «Число 5 кладем в переменную a ».

Внимание!

Переменную a положить в число 5 невозможно.

Приведем простейшие примеры работы с присваиванием значений переменным в интерактивном режиме (рис. 20).

```

IDLE Shell 3.9.1
File Edit Shell Debug Options Window Help
Python 3.9.1 (tags/v3.9.1:1e5d33e, Dec 7 2020, 17:08:21) [MSC v.1927 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>> a = 5
>>> b = 2.1
>>> c = '103a'
>>> a-b
2.9
>>> b+a
7.1
>>> a*c
'103a103a103a103a103a'
>>> a = 10
>>> a*c
'103a103a103a103a103a103a103a103a103a'
>>> a-b
7.9
>>> |
  
```

Рис. 20. Присваивание значений переменным

Прокомментируем каждое действие интерактивного сеанса.

| | |
|------------------|---|
| >>> $a = 5$ | Положить число 5 в переменную a |
| >>> $b = 2.1$ | Положить число 2.1 в переменную b |
| >>> $c = '103a'$ | Положить строку '103a' в переменную c |

| | |
|---|--|
| >>> a-b 2.9 | Вывести результат выполнения операции $(a-b)$ (вычитания числа 2.1 из 5) |
| >>> b+a 7.1 | Вывести результат выполнения операции $(b+a)$ (сложения чисел 2.1 и 5) |
| >>> a*c '103a103a103a103a103a' | Вывести результат выполнения операции $(a*c)$ (дублирование строки '103a' 5 раз) |
| >>> a = 10 | Положить число 10 в переменную <i>a</i> (перезаписать значение переменной <i>a</i>) |
| >>> a*c '103a103a103a103a103a103a103a103a103a' | Вывести результат выполнения операции $(a*c)$ (конкатенации строки '103a' 10 раз) |
| >>> a-b 7.9 | Вывести результат выполнения операции $(a-b)$ (вычитания числа 2.1 из 10) |
| >>> | Готовность Python к дальнейшим инструкциям |

Из примеров видно, что переменную можно перезаписывать и класть в нее различные типы данных. Они по-разному записываются в памяти компьютера, а значит, и действия, выполняемые с ними, будут различны.

Несмотря на то что для правильной работы программ очень важно понимать, какие типы данных используются в создаваемых переменных, в Python *отсутствует строгая типизация данных*. Это означает, что Python самостоятельно определяет широкий спектр различных типов и при необходимости переопределяет его. Например, при сложении дроби 2.5 и целого числа 5 сначала число 5 будет переопределено в 5.0 и только после этого выполнится операция сложения ($2.5 + 5 = 2.5 + 5.0 = 7.5$).

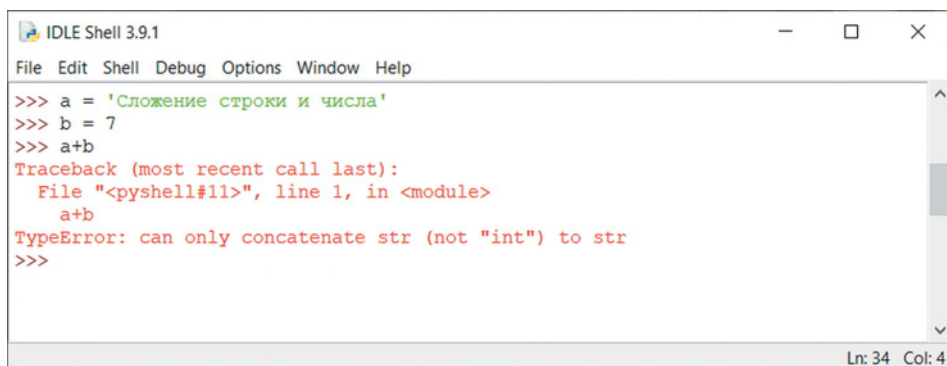
Примером самостоятельности Python является и оператор `*`, который в зависимости от типов операндов может выполнять операцию умножения или операцию дублирования (конкатенации), как мы узнали из экспериментов в первой главе.

Другими словами, Python — динамичный (гибкий, подвижный) язык программирования, который самостоятельно подстраивается под предлагаемые программистом условия. Его также называют языком программирования с *динамической типиза-*

цией данных. Ведь в языках программирования со *строгой типизацией* объявляемая переменная записывается вместе с ее типом данных, становится однозначно определенной и далее может переопределяться только в случае вмешательства программиста в программный код.

Могут возникнуть сомнения в эффективности этого уменьшения контроля со стороны программиста, однако об этом не стоит беспокоиться: создатели языка позаботились о том, чтобы, несмотря на вариативность, однозначность интерпретации языком сохранялась.

Если выполнить операцию, которую Python однозначно не интерпретирует, то в результате выведется ошибка определенного типа, что даст подсказку, как необходимо поменять значение для конкретной инструкции. Примером может служить сложение числового и строкового типов данных (рис. 21).



```

IDLE Shell 3.9.1
File Edit Shell Debug Options Window Help
>>> a = 'Сложение строки и числа'
>>> b = 7
>>> a+b
Traceback (most recent call last):
  File "<pyshell#11>", line 1, in <module>
    a+b
TypeError: can only concatenate str (not "int") to str
>>>
Ln: 34 Col: 4
```

Рис. 21. Ошибка при сложении числа и строки

Во избежание подобного рода ошибок и невозможности работы программы иногда программист может самостоятельно менять тип данных. Пример такого преобразования уже встретился в практикуме к первой главе, где в одной из задач было предложено использовать готовый код для ввода значений пользователем с клавиатуры:

```
a = int(input('Введите число a: '))
b = int(input('Введите число b: '))
```

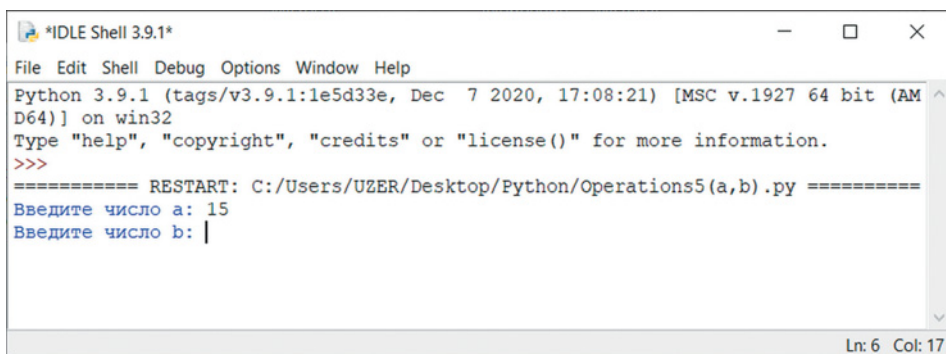
Разберем подробнее механизм, заложенный в конструкции, познакомимся с оператором `input()` и принудительным изменением типа данных.

Оператор input() — ввод данных в программу

Заменяем первые две строчки файла `Operations4(a,b).py` из первой главы на приведенные ниже и получившийся код сохраним в файле `Operations5(a,b).py`:

```
a = int(input('Введите число a: '))
b = int(input('Введите число b: '))
print('\n',a,'+',b,'=',a+b,'\n',a,'-',b,'=',a-b,
      '\n',a,'*',b,'=',a*b,'\n',a,'/',b,'=',a/b,
      '\n',a,'**',b,'=',a**b,'\n',a,'//',b,'=',a//b)
```

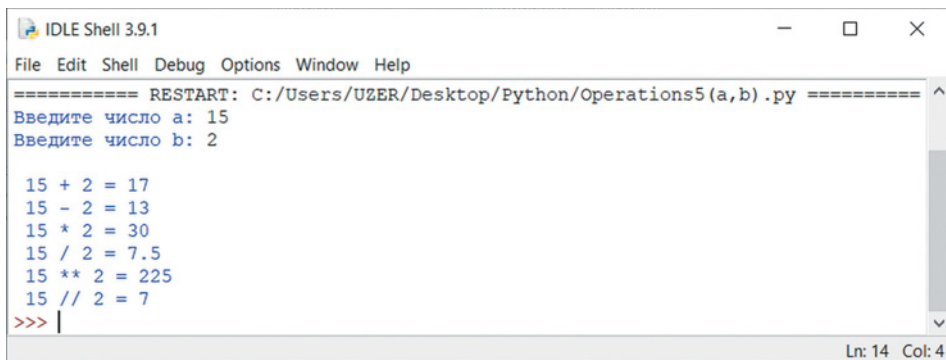
Запустив программу (**Run** → **Run Module** или **F5**), видим, что Python ожидает ввод числа с клавиатуры (рис. 22), а после ввода чисел выполняет все указанные в программе арифметические операции над ними (рис. 23).



```
*IDLE Shell 3.9.1*
File Edit Shell Debug Options Window Help
Python 3.9.1 (tags/v3.9.1:1e5d33e, Dec 7 2020, 17:08:21) [MSC v.1927 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
===== RESTART: C:/Users/UZER/Desktop/Python/Operations5(a,b).py =====
Введите число a: 15
Введите число b: |
```

Ln: 6 Col: 17

Рис. 22. Ожидание ввода чисел при запуске `Operations5(a,b).py`



```
IDLE Shell 3.9.1
File Edit Shell Debug Options Window Help
===== RESTART: C:/Users/UZER/Desktop/Python/Operations5(a,b).py =====
Введите число a: 15
Введите число b: 2

15 + 2 = 17
15 - 2 = 13
15 * 2 = 30
15 / 2 = 7.5
15 ** 2 = 225
15 // 2 = 7
>>> |
```

Ln: 14 Col: 4

Рис. 23. Результат выполнения `Operations5(a,b).py`

Разберем конструкцию

```
a = int(input('Введите число a: '))
```

- 'Введите число a: ' — строка-аргумент функции `input()`, некий комментарий для пользователя, который выводится для удобства использования программы;
- функция `input()` ожидает ввода данных с клавиатуры после комментария для пользователя до нажатия клавиши **Enter**, причем все данные записываются как *строка символов* (это очень важно!);
- функция `int()` преобразует строку, введенную с клавиатуры, в *целое число*;
- `a` — переменная, в которую записывается число, полученное путем ввода с клавиатуры.

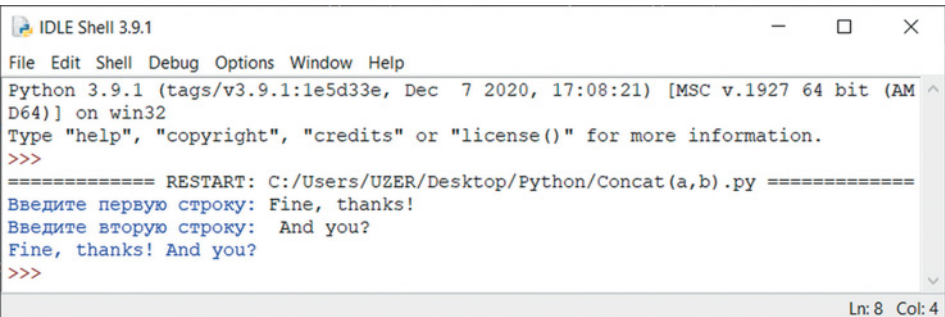
Другими словами, если программисту необходимо проверить, как работает конкатенация строк, то в функции `int()` не будет необходимости.

Для примера напишем код, который выполняет задачу склеивания двух строк:

```
a = input('Введите первую строку: ')
b = input('Введите вторую строку: ')
print(a+b)
```

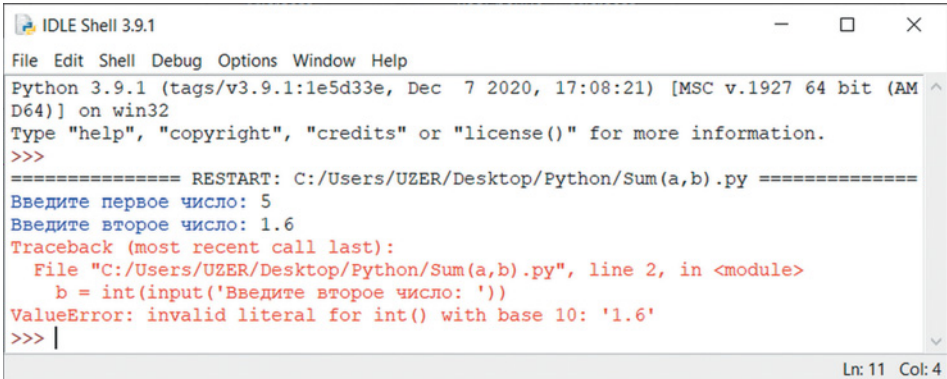
Сохраним код в файле `Concat(a,b).py` и запустим программу. Видим, что конкатенация строк прошла успешно (рис. 24).

Далее напишем программу, запрашивающую два любых числа и выводящую их сумму. Создадим новый файл с названием



```
IDLE Shell 3.9.1
File Edit Shell Debug Options Window Help
Python 3.9.1 (tags/v3.9.1:1e5d33e, Dec 7 2020, 17:08:21) [MSC v.1927 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
===== RESTART: C:/Users/UZER/Desktop/Python/Concat(a,b).py =====
Введите первую строку: Fine, thanks!
Введите вторую строку: And you?
Fine, thanks! And you?
>>>
```

Рис. 24. Результат выполнения файла `Concat(a,b).py`



```
Python 3.9.1 (tags/v3.9.1:1e5d33e, Dec 7 2020, 17:08:21) [MSC v.1927 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
===== RESTART: C:/Users/UZER/Desktop/Python/Sum(a,b).py =====
Введите первое число: 5
Введите второе число: 1.6
Traceback (most recent call last):
  File "C:/Users/UZER/Desktop/Python/Sum(a,b).py", line 2, in <module>
    b = int(input('Введите второе число: '))
ValueError: invalid literal for int() with base 10: '1.6'
>>> |
```

Рис. 25. Результат выполнения Sum(a,b).py с использованием int()

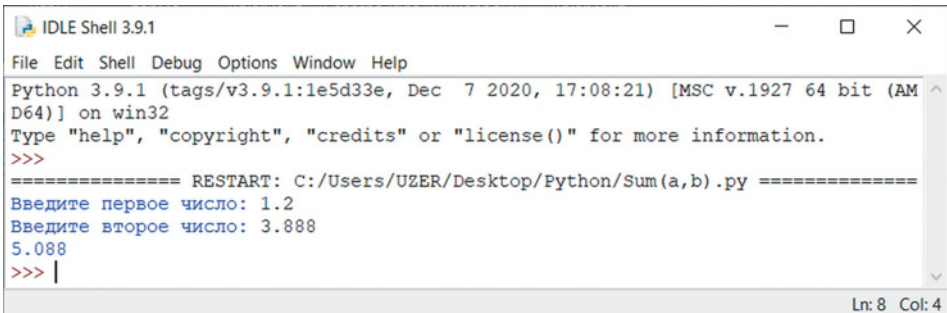
Sum(a,b). Вроде бы с такой задачей мы уже сталкивались и программный код на первый взгляд должен быть следующим:

```
a = int(input('Введите первое число: '))
b = int(input('Введите второе число: '))
print(a+b)
```

Казалось бы, строки в числа перевели функцией int(), но при запуске программы Python «ругается» и сообщает об ошибке (рис. 25).

Конечно же! Функция int() преобразует строку в целое число, а числа могут быть еще и дробными. Заменим int на float и запустим программу снова (рис. 26).

Написание комментариев для пользователя на конкретном языке не всегда удобно и подходит не всем, поэтому часто ком-



```
Python 3.9.1 (tags/v3.9.1:1e5d33e, Dec 7 2020, 17:08:21) [MSC v.1927 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
===== RESTART: C:/Users/UZER/Desktop/Python/Sum(a,b).py =====
Введите первое число: 1.2
Введите второе число: 3.888
5.088
>>> |
```

Рис. 26. Результат выполнения Sum(a,b).py с использованием float()

ментарии заменяются математической записью. Например, код программы `Sum(a,b).py` можно изменить следующим образом:

```
a = float(input('a = '))
b = float(input('b = '))
print(a + b)
```

Важно отметить, что часто программа содержит больше одной операции. В таком случае удобно создать некоторую переменную для всех вычислений и преобразований и только потом обратиться к ней при выводе. Например, в нашей программе можно было объявить переменную `sum`:

```
a = float(input('a = '))
b = float(input('b = '))
sum = a + b
print(sum)
```

Резюмируем преобразования задачи `Operations4(a,b).py`:

- функция `input()` используется для ввода информации с клавиатуры;
- аргументом функции `input()` может выступить комментарий для пользователя, который укажет на то, что ожидает от него получить машина;
- часто комментарий для пользователя выводится в унифицированном математическом виде;
- любые данные, вводимые с клавиатуры с помощью функции `input()`, считываются как строка;
- при необходимости можно «навесить» на функцию `input()` преобразователи `int()` или `float()`, чтобы перевести строку в целое или дробное число (с плавающей точкой) соответственно;
- при большом количестве действий удобно создать отдельную переменную для выполнения всех операций и обратиться к ней при выводе в аргументе функции `print()`.

Итак, мы не только расширили количество решаемых задач с помощью изменения преобразователя `int()` на `float()`, но и позаботились о том, чтобы язык комментария для пользователя был универсальным.

Ключи оптимизации

Существует множество способов решить одну и ту же задачу, но если этим решением сможет пользоваться большее количество человек или она будет решать не одну конкретную задачу, а множество типовых, то подобного рода решение будет более ценным в сравнении с другими.

Возьмем за основу программу, которая решает задачу вычисления площади квадрата со стороной 2, и проследим, как можно увеличить ее возможности.

| | | |
|---|---|--|
| 1 | Программа, вычисляющая площадь квадрата со стороной 2 | Решает всего одну типовую задачу |
| 2 | Программа, вычисляющая площадь квадрата со стороной a , значение которой вводится программистом при написании кода | Решает целый ряд типовых задач, однако используется только программистом, так как менять значение переменной a можно только при работе с кодом программы |
| 3 | Программа, вычисляющая площадь квадрата со стороной a , значение которой вводится пользователем с клавиатуры | Решает такой же ряд задач, однако может использоваться неограниченным количеством пользователей |
| 4 | Программа, вычисляющая площадь прямоугольника со сторонами a и b , значения которых вводятся пользователем с клавиатуры | Поскольку квадрат является частным случаем прямоугольника, ряд решаемых задач сильно расширяется. Более того, программа будет доступна к использованию большим количеством пользователей |

Рассмотренные программы увеличивают количество решаемых типовых задач и/или аудиторию пользователей, однако существуют и другие способы оптимизации работы программиста и расширения возможностей самого кода. Примером может быть использование функции `eval()`, которая упрощает обработку ввода с клавиатуры.

Функция преобразования eval()

Функция `eval()` преобразует строку символов, вводимых с клавиатуры, в числа и даже в арифметические выражения.

Вспомним наше знакомство с интерактивным режимом Python в качестве калькулятора. Оболочка IDLE самостоятельно определяла тип данных и производила вычисления различных арифметических выражений.

Например, в переменную `a` будем записывать математическое выражение с клавиатуры, в переменной `result` его вычислять, а результат выводить на экран с помощью функции `print`:

```
a = input('a = ')
result = eval(a)
print(a, '=', result)
```

или использовать только ввод и вывод без дополнительных переменных:

```
a = input('a = ')
print(a, '=', eval(a))
```

Другими словами, данная функция делает возможным использование интерактивного калькулятора в режиме написания многострочных программ. Просто представьте, что вместо функции `eval()` стоит приглашение `>>>` интерпретатора Python.

Обратите внимание, что функция обрабатывает одну строку:

| | |
|--|---|
| <pre>a = input('a = ') #5 b = input('b = ') #6 print(eval(a+b)) #56</pre> | <p>Как интерпретатор считал <code>a</code> и <code>b</code>? — Строки <code>'5'</code> и <code>'6'</code>, записанные ранее в переменные <code>a</code> и <code>b</code>.</p> <p>Какая операция выполнялась? — Конкатенация строк (<code>'5' + '6' = '56'</code>)</p> |
| <pre>a = input('a = ') #5 b = input('b = ') #6 print(eval(a)+ eval(b)) #11</pre> | <p>Как интерпретатор считал <code>a</code>? — Целое число 5.</p> <p>Как интерпретатор считал <code>b</code>? — Целое число 6.</p> <p>Какая операция выполнялась? — Сложение целых чисел (<code>5 + 6 = 11</code>)</p> |

| | |
|--|---|
| <pre>a = input('a = ') #5 b = input('b = ') #6 c = eval(input('c = ')) #0.0 d = eval(input('d = ')) #3 print((a+b)*int(d-c)) #565656</pre> | <p>Какие данные содержат a и b после ввода с клавиатуры? — Строки '5' и '6'.</p> <p>Как интерпретатор считал c? — Вещественное число 0.0.</p> <p>Как интерпретатор считал d? — Целое число 3.</p> |
| | <p>Какие операции выполнились? — Конкатенация строк ('5' + '6' = '56'), вычитание вещественного числа из целого (3 - 0.0 = 3.0), преобразование в целый тип данных (int(3.0) = 3) и дублирование строки ('56' * 3 = '565656')</p> |
| <pre>a = input('a = ') #5*6.0+1 print(a, '=', eval(a)) #31.0</pre> | <p>Как интерпретатор считал a? — Арифметическое выражение 5*6.0+1.</p> <p>Какие операции выполнились? — Умножение целого числа на вещественное (5 * 6.0 = 30.0) и сложение вещественного числа с целым (30.0 + 1 = 31.1)</p> |
| <pre>a = input('a = ') #5-'4' print(a, '=', eval(a)) #TypeError: unsupported operand type(s) for -: 'int' and 'str'</pre> | <p>Как интерпретатор считал a? — Попытку совершить операцию между целым числом и строкой.</p> <p>Какой результат? — Ошибка: оператор - не поддерживает целый и строковый тип operandов</p> |

Из последнего примера видно, что для работы с Python по-прежнему важно, чтобы определенным типам данных соответствовали возможные для них операции. Поэтому необходимо знать, какие типы данных существуют и какие взаимодействия с ними предусмотрены.

Для начала рассмотрим самые распространенные типы данных, которые встречаются практически в каждой программе: **числовой**, **логический** и **строковый**, а далее в книге познакомимся и с другими.

Числовой тип данных

Основными видами чисел, используемых в Python, являются *целые* (**int**) и *дробные* (**float**, вещественные числа или числа с плавающей точкой).

Примеры целых чисел: 0; -10; 99900.

Примеры чисел с плавающей точкой: 5.0; -8.999; 9.7531.

Результат арифметических операций с числами в Python

1. Сложение, вычитание, умножение целых чисел приводят к целочисленному результату¹.

2. Сложение, вычитание, умножение целого и вещественного или нескольких вещественных чисел приводят к вещественному результату.

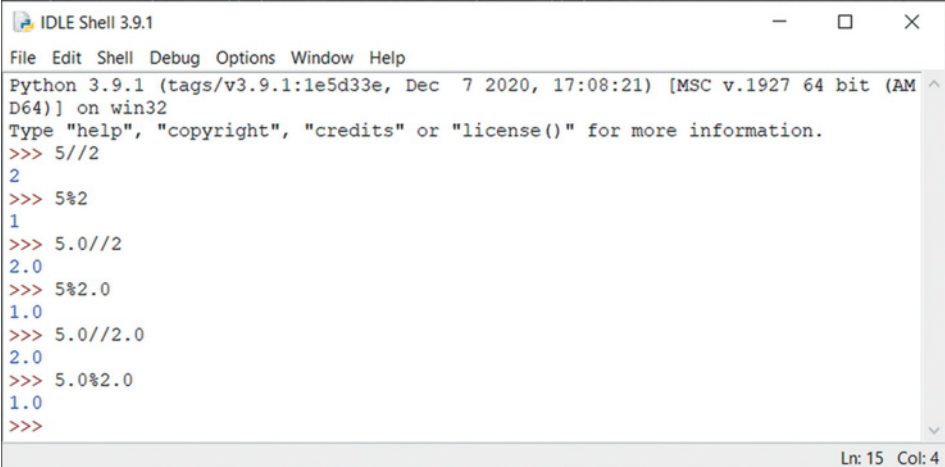
3. Деление чисел *всегда* приводит к вещественному результату.

4. При работе с операторами `//` и `%` («целая часть от деления» и «остаток от деления») Python будет возвращать целочисленный результат, если оба операнда будут целыми, и вещественный результат, если хотя бы один из операндов будет вещественным (рис. 27).

| | |
|------------------------------------|--|
| <pre>>>> 5//2 2</pre> | <p>Какое количество раз число 2 входит в число 5? — Дважды. Какие типы операндов используются? — Только целые. Тогда какой результат получается? — Целое число 2.</p> |
| <pre>>>> 5%2 1</pre> | <p>Какой остаток от деления числа 5 на 2? — 1. Какие типы операндов используются? — Только целые. Тогда какой результат получается? — Целое число 1.</p> |
| <pre>>>> 5.0//2 2.0</pre> | <p>Какое количество раз число 2 входит в число 5? — Дважды. Какие типы операндов используются? — Вещественный и целый. Тогда какой результат получается? — Вещественное число 2.0.</p> |

¹ В Python 2 при делении целых чисел результат был тоже целым (отбрасывалась дробная часть подобно оператору `//`; разницу в результатах использования операторов `/` и `//` можно было увидеть только при делении целых и вещественных чисел).

| | |
|--------------------------------------|---|
| <pre>>>> 5%2.0 1.0</pre> | Какой остаток от деления числа 5 на 2? — 1. Какие типы операндов используются? — Целый и вещественный. Тогда какой результат получается? — Вещественное число 1.0. |
| <pre>>>> 5.0//2.0 2.0</pre> | Какое количество раз число 2 входит в число 5? — Дважды. Какие типы операндов используются? — Оба вещественные. Тогда какой результат получается? — Вещественное число 2.0. |
| <pre>>>> 5.0%2.0 1.0</pre> | Какой остаток от деления числа 5 на 2? — 1. Какие типы операндов используются? — Оба вещественные. Тогда какой результат получается? — Вещественное число 1.0. |



```
IDLE Shell 3.9.1
File Edit Shell Debug Options Window Help
Python 3.9.1 (tags/v3.9.1:1e5d33e, Dec 7 2020, 17:08:21) [MSC v.1927 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>> 5//2
2
>>> 5%2
1
>>> 5.0//2
2.0
>>> 5%2.0
1.0
>>> 5.0//2.0
2.0
>>> 5.0%2.0
1.0
>>>
```

Ln: 15 Col: 4

Рис. 27. Работа с целыми и вещественными числами в Python

Логический тип данных

Логический тип данных (**bool**) используется, когда можно задать вопрос, за которым следует однозначный ответ «да» или «нет». Например, «Зеленый свет светофора включен?», «За окном идет дождь?», «Пять больше пяти?». Для компьютера ответ «да» соответствует значению **True** (истина), а ответ «нет» — значению

False (ложь). Именно эти два значения содержит логический тип данных, который носит название **bool** или булев тип и относится к разделу математической логики под названием «булева алгебра», «алгебра логики» или «алгебра высказываний».

Тип **bool** необходим при проверке каких-либо условий или ограничений¹. Например, если при вводе с клавиатуры данных для подсчета периметра фигуры пользователь ввел отрицательное число, то программа должна выводить сообщение об ошибке, ведь длина не может быть отрицательной. Поэтому в код добавляют условие неотрицательности, чтобы избежать некорректных результатов при работе программы. Как следствие, программисту очень важно вводить отрицательные, положительные значения и нуль при тестировании работы кода, что также будет являться проверкой оптимальности и эффективности.

Строковый тип данных

Строковый тип данных (**str**) встречался в примерах так же часто, как и числа. *Строка* — это последовательность символов, включая пробел, знаки препинания, цифры и т. д. Например, сообщения в социальных сетях имеют строковый тип данных.

Примеры: '1', 'Имя2000', '___'.

Единственное отличие строкового типа данных от числового в том, что с такими данными нельзя производить арифметические вычисления. Чаще всего его используют для вывода сообщений на экран, включая диалог с пользователем.

Еще раз вспомним операции склеивания и дублирования с помощью операторов + и *.

1. *Конкатенация* выполняется с помощью оператора + и только при условии, что оба операнда строкового типа.

2. *Дублирование* выполняется с помощью оператора * и только при условии, что один операнд строкового типа, а второй не просто числового, а целого типа.

¹ Интересно, что у переменной логического типа, отвечающей за некоторую развилку событий (например, условие выполняется и далее следуют одни действия или условие не выполняется и следуют другие действия), существует название «флаг», которое, подобно поднятому флагу на поле боя, сигнализирует о принятом решении и дальнейших действиях.

Что нового мы узнали во второй главе?

1. Память компьютера состоит из отдельных фрагментов (ячеек памяти), которые могут быть заполнены различными данными. Python лишь наполняет ячейки данными, вводимыми программистом, а после обращается к ним только по ссылке (переменной).

2. *Переменная* является именованной областью памяти компьютера.

- *Имя переменной* не содержит пробелов, всегда начинается с латинской буквы, а остальные символы могут быть буквами, цифрами или нижним подчеркиванием «_». При этом имена переменных в Python чувствительны к регистру.
- Присваивание *значения переменной* происходит с помощью оператора « = » справа налево. При необходимости значение переменной можно переопределять.

3. *В Python отсутствует строгая типизация данных.* Это означает, что Python самостоятельно определяет тип данных, при необходимости переопределяет его автоматически и производит возможные для этого типа данных операции.

4. Функция `input()` считывает данные, вводимые с клавиатуры до нажатия клавиши **Enter**, и присваивает им строковый тип.

5. Самыми распространенными типами данных, используемыми при написании программ, являются числа (*целые (int), дробные*, или *вещественные (float)*), *строки (str)* и *булевы* (логические) значения (**bool**). Каждый тип данных имеет свои свойства совместимости с другими типами данных.

6. Существуют функции, которые помогают программисту самостоятельно переопределять тип данных. Мы познакомились с функциями `int()`, `float()` и `eval()`.

- Функция `int()` преобразует то, что находится в скобках, в целое число, отбрасывая дробную часть при ее наличии.
- Функция `float()` преобразует то, что находится в скобках, в дробное число.
- Функция `eval()` преобразует строку символов, вводимых с клавиатуры, в числа и арифметические выражения.

Заметьте, что функции имеют название типов данных. Поэтому нетрудно предположить, что существуют функции преобразования `str()` и `bool()`. А это значит, что можно снова воспользоваться интерактивным режимом Python для наших экспериментов.

Практикум

Основные понятия главы

1. Заполните таблицу, определив, какие имена переменных можно использовать:

| Имя переменной | Можно (+) / нельзя (-) использовать |
|----------------|-------------------------------------|
| 5a | |
| d | |
| pol_10 | |
| abs | |
| mod | |
| сумма 1 | |
| num-5 | |

2. Заполните таблицу, определив, к какому типу данных относится объект:

| Объект | int | float | str |
|---------------|-----|-------|-----|
| -5 | + | - | - |
| 46.0 | | | |
| -100000000000 | | | |
| '2+5' | | | |
| -9.5 | | | |
| 1 | | | |
| '900' | | | |

3. Заполните таблицу, используя типы данных из задания 2:

| | b=input(a) | b=int(input(a)) | b=float(input(a)) |
|-------------------------------|------------|-----------------|-------------------|
| Тип аргумента функции input() | | | |
| Тип аргумента функции int() | - | | |
| Тип аргумента функции float() | - | - | |
| Тип переменной b | | | |

Чтение кода

4. Заполните таблицу, вычисляя результат следующих фрагментов кода без среды IDLE:

| № | Код | Результат |
|-----|--|-----------|
| 4.1 | <pre>print('Вспомни 'эксперимент', который был в первой главе')</pre> | |
| 4.2 | <pre>a = '0' b = '1' print('\n',a+a, '\n',a+b, '\n',b+a, '\n',b+b)</pre> | |
| 4.3 | <pre>a = float('6') b = int('5') c = str(int(a%b))+ 'C' print(c)</pre> | |
| 4.4 | <pre>a = float('6') b = int('5') print(a//b)</pre> | |

Поиск ошибок

5. Объясните, почему в следующих частях программы Python выведет ошибку, используя варианты ответов:

- А. Переменная не определена.
- В. Невозможное присваивание.
- С. Аргумент не строкового типа.
- Д. Количество открытых скобок не равно количеству закрытых.
- Е. Невозможная операция сложения/конкатенации.

| № | Фрагмент кода | Ошибка |
|-----|--|--------|
| 5.1 | <pre>a = float(input('a = ')) razn = a - b print(razn)</pre> | |
| 5.2 | <pre>a = int(input('a = ')) b = 2*a print(b)</pre> | |
| 5.3 | <pre>a = int(input('a = ')) b = input('b = ') sum = a + b print(sum)</pre> | |

| | | |
|-----|--|--|
| 5.4 | <pre>a = input(a =) b = input(b =) print(eval(a) + eval(b))</pre> | |
| 5.5 | <pre>int(input("a = ")) = a input("b = ") = b print(eval(a) * eval(b))</pre> | |

Оптимизация

6. Дана программа, которая запрашивает сначала возраст старшего брата, затем возраст младшего брата и выводит разницу в возрасте между братьями на экран:

```
a = int(input())
b = int(input())
print(a-b)
```

Предложите возможный вариант написания кода, чтобы был реализован понятный диалог с пользователем при вводе и выводе данных.

Разработка

7. Напишите программу, которая:

- 7.1) просит пользователя ввести число и выводит его на экран;
- 7.2) просит пользователя ввести целое число и выводит число, следующее за ним;
- 7.3) просит пользователя ввести дробное число и выводит три числа, следующих за ним с интервалом 1;
- 7.4) просит пользователя ввести целое число и выводит это число десять раз без пробелов и знаков препинания;
- 7.5) просит пользователя ввести любой символ и выводит этот символ пять раз через пробел;
- 7.6) просит пользователя ввести два целых числа и выводит сумму, разность и произведение этих чисел без использования функции `eval()`;
- 7.7) просит пользователя ввести два числа и выводит сумму, разность и произведение этих чисел с помощью функции `eval()`.

8. Напишите программу, которая вычисляет произведение двух целых чисел, введенных с клавиатуры, но выводит его три раза. Используйте в задаче функции преобразования типов `int()` и `str()`.

Пример:

```
Введите первое число: 5
Введите второе число: 10
505050
```

9. Напишите программу, которая выводит строку, введенную пользователем с клавиатуры, столько раз, сколько запросил пользователь.

Пример:

```
Введите строку: ма
Введите количество повторений: 2
мама
```

10. Напишите программу, которая просит пользователя ввести сообщение и выводит его с комментарием для другого человека.

Пример:

```
Введите сообщение: Это Ольга, ваш новый менеджер.
Для вас одно новое сообщение: "Это Ольга, ваш новый
менеджер."
```

11. Напишите программу, которая запрашивает количество покупаемых продуктов и выводит общую стоимость покупки. Используйте следующую таблицу стоимости продуктов:

| | | |
|------------------|------------------|------------------|
| хлеб — 35.5 руб. | молоко — 74 руб. | масло — 120 руб. |
|------------------|------------------|------------------|

Пример:

```
Количество покупаемых единиц хлеба: 1
Количество покупаемых единиц молока: 2
Количество покупаемых единиц масла: 0
```

Общая сумма к оплате - 183.5 руб.

Глава 3. Условия

Программы, рассмотренные в первых двух главах, представляют прямую последовательность действий, где каждая операция совершается строго одна за другой. Однако также было упомянуто, что существует логический тип **bool**, который позволяет использовать *условные конструкции*.

Появление условия в программе указывает на *ветвление* — форму организации действий, при которой в зависимости от выполнения или невыполнения некоторого условия совершается определенная последовательность действий, что создает больше возможностей и форм для решения той или иной задачи.

Неполная форма ветвления: конструкция «если..., то...»

Самая примитивная форма ветвления — *неполная*. Она имеет конструкцию «*если...*, *то...*». Не задумываясь, мы часто используем ее

- в повседневной жизни: «**Если** температура на улице низкая, **то** лучше одеться теплее»;
- в математике: «**Если** теплоход плывет против течения реки, **то** его скорость в данный момент меньше его собственной»; «**Если** $x + 2 = 5$, **то** $x = 0$ ».

Заметьте, последнее высказывание не является верным с точки зрения математики. Этот пример демонстрирует *ложность высказывания*, в то время как выражение «**если** $x + 2 = 5$, **то** $x = 3$ » показывает *истинность высказывания*. Другими словами, конструкция как бы разбивается на две части: «условие» и «следствие» или, в терминах алгебры логики, «посылка» и «следствие». При этом

- посылка является условием, достаточным для выполнения следствия;
- следствие является условием, необходимым для истинности посылки.

Данная логическая конструкция играет важную роль при формулировании определений различных понятий, теорем и научных законов.

Итак, определение истинности или ложности высказывания «если $x + 2 = 5$, то $x = 0$ » сводится к вопросу «2 равно 5?». Аналогично можно поставить в соответствие вопросам знакам неравенств $>$, \geq , $<$, \leq :

| Условие в математическом выражении | Соответствующий вопрос | Ответ на вопрос | Истинность / ложность условия |
|------------------------------------|------------------------|-----------------|-------------------------------|
| $5 = 2 + 3$ | 5 равно 5? | Да | Условие истинно |
| $5 > 2 + 3$ | 5 больше 5? | Нет | Условие ложно |
| $5 \geq 2 + 3$ | 5 больше или равно 5? | Да | Условие истинно |
| $5 < 2 + 3$ | 5 меньше 5? | Нет | Условие ложно |
| $5 \leq 2 + 3$ | 5 меньше или равно 5? | Да | Условие истинно |

На языке Python запись подобных математических условий производится с помощью *операторов сравнения* $<$, $>$, $<=$, $>=$, $==$, $!=$.

| Знак сравнения в арифметике | Оператор сравнения на языке Python |
|-----------------------------|------------------------------------|
| = | == |
| \neq | != |
| > | > |
| \geq | >= |
| < | < |
| \leq | <= |

Важно!

В языке Python одиночным знаком «=» обозначают операцию присваивания, а знаком «==» — операцию сравнения.

| | | |
|---------------------|-----------------------|--------------------------------|
| <code>a = 5</code> | Операция присваивания | Число 5 кладем в переменную a |
| <code>a == 5</code> | Операция сравнения | Значение переменной a равно 5? |

Итого имеем всего шесть типов сравнения, однако результат каждого из них имеет один тип — логический (**bool**), с которым мы встречались во второй главе.

Условная конструкция «если..., то...» на языке Python записывается с помощью условного оператора **if**.

Полная форма ветвления: конструкция «если..., то..., иначе...»

Полная форма ветвления отличается от неполной наличием в ней альтернативного варианта развития событий. Условная конструкция такой формы ветвления имеет вид «если..., то..., иначе...» и на языке Python записывается с помощью парного условного оператора **if-else**.

Важно!

В конце первой строки условной конструкции ставится *двоеточие*, которое показывает, что после нажатия клавиши **Enter** будут перечислены действия, выполняющиеся только при истинности указанного выше условия.

Python автоматически делает небольшой отступ вправо, называемый *табуляцией*, и при переходе на следующую строку нажатием клавиши **Enter** отступ сохраняется (данный отступ можно поставить самостоятельно с помощью клавиши **Tab**¹, а убрать с помощью **Backspace**).

```
if < условие >:  
    < действие 1 >  
    < действие 2 >  
    ...
```

```
if < условие >:  
    < действие 1.1 >  
    < действие 1.2 >  
    ...  
else:  
    < действие 2.1 >  
    < действие 2.2 >  
    ...
```

¹ Среды разработки для удобства делают автоматический отступ табуляцией, однако изначально в языке Python предполагается четыре пробела.

Чтобы запустить условную конструкцию в интерактивной оболочке IDLE, после написания последнего действия необходимо нажать клавишу **Enter** дважды. Первое нажатие переводит строку для написания следующего действия, а второе выходит из условной конструкции и запускает код.

Примеры неполной и полной форм ветвления

Допустим, задача состоит в том, чтобы программа выводила приветствие только при условии, что пользователь введет с клавиатуры пароль 999, как в фильмах про шпионов.

Вариант неполной формы ветвления мог бы выглядеть следующим образом:

```
X = input('Добрый день. Чем могу Вам помочь? ')
if X == '999':
    print('Проходите. Вас ожидают.')
```

Для полной же формы ветвления достаточно добавить альтернативное действие с помощью оператора **else**, т. е. то, что выведет компьютер, если введенный пользователем код окажется неверным. Например:

```
X = input('Добрый день. Чем могу Вам помочь? ')
if X == '999':
    print('Проходите. Вас ожидают.')
```

else:

```
    print('Извините. Перерыв по техническим причинам.')
```

Условный оператор elif

Количество альтернативных действий может быть неограниченным. Для подобных случаев Python предлагает еще один условный оператор **elif**, представляющий слияние комбинации else-if.

Представьте, что к вам подошли с фразой, которая начинается со слов: «Иначе вы должны будете...» Мало того, что это вводит в недоумение, так еще и остается открытым вопрос, что же было до этого.

Поэтому условный оператор **elif** не является самостоятельным, поскольку начинается с альтернативы («иначе»), ведь она подразумевает наличие какого-то условия до его использования.

Приведем пример:

```
print('Есть ли у Вас дисконтная карта нашего магазина?')
X = input("1 - 'Да'; 0 - 'Не помню'; -1 - 'Нет': ")
if X == '1':
    print('Приложите карту к терминалу')
elif X == '0':
    print('Проверить наличие Вас в базе данных?')
elif X == '-1':
    print('Вы хотели бы приобрести дисконтную карту?')
```

Последний пример, конечно, демонстрирует использование условного оператора **elif**, но после вывода вопроса о поиске клиента в базе данных и предложения приобрести дисконтную карту можно предложить клиенту выбрать ответ «Да» или «Нет» с помощью *вложенных условий* и довести алгоритм до логичного завершения. Тогда обновленная версия примера может быть следующей:

```
print('Есть ли у Вас дисконтная карта нашего магазина?')
X = input("1 - 'Да'; 0 - 'Не помню'; -1 - 'Нет': ")
if X == '1':
    print('Приложите карту к терминалу')
elif X == '0':
    print('Проверить наличие Вас в базе данных?')
    X = input("1 - 'Да'; -1 - 'Нет': ")
    if X == '1':
        print('Обратиться по поводу восстановления номера карты можно к администратору магазина')
    elif X == '-1':
        print('Можете продолжить покупку')
elif X == '-1':
    print('Вы хотели бы приобрести дисконтную карту?')
    X = input("1 - 'Да'; -1 - 'Нет': ")
    if X == '1':
        print('Обратиться по поводу приобретения дисконтной карты можно к администратору магазина')
    elif X == '-1':
        print('Можете продолжить покупку')
```

В примере рассмотрены все случаи с предложенными вариантами ответа для покупателя, но давайте будем реалистами. Покупатель может оказаться с годовалым ребенком, который «пытается» на клавиатуре все что можно и нельзя, а значит, мы не попадем ни в одну условную конструкцию и обратная связь от машины не будет осуществлена.

Поэтому в окончательный вариант кода следует добавить альтернативу **else** после других условных операторов во всех ветвлениях:

```
print('Есть ли у Вас дисконтная карта нашего магазина?')
X = input("1 - 'Да'; 0 - 'Не помню'; -1 - 'Нет': ")
if X == '1':
    print('Приложите карту к терминалу')
elif X == '0':
    print('Проверить наличие Вас в базе данных?')
    X = input("1 - 'Да'; -1 - 'Нет': ")
    if X == '1':
        print('Обратиться по поводу восстановления номера карты можно к администратору магазина')
    elif X == '-1':
        print('Можете продолжить покупку')
    else:
        print('Неправильный ввод')
elif X == '-1':
    print('Вы хотели бы приобрести дисконтную карту?')
    X = input("1 - 'Да'; -1 - 'Нет': ")
    if X == '1':
        print('Обратиться по поводу приобретения дисконтной карты можно к администратору магазина')
    elif X == '-1':
        print('Можете продолжить покупку')
    else:
        print('Неправильный ввод')
else:
    print('Неправильный ввод')
```

Ура! Даже такой вариант мы предусмотрели.

Операторы `and` и `or`

Комбинации условий также возможны с помощью вспомогательных операторов `and` и `or`.

При использовании оператора `and` (логическое И) *каждое из условий* должно быть истинным. Только в этом случае будет выполняться действие, заложенное в результат условной конструкции.

Например, чтобы установить существование треугольника с заданными длинами сторон, необходимо проверить истинность условия «любая сторона треугольника меньше суммы двух других сторон»:

```
print('A, B, C - стороны треугольника')
X = int(input('Введите A: '))
Y = int(input('Введите B: '))
Z = int(input('Введите C: '))
print('Каждая сторона треугольника должна быть меньше
суммы двух других')
if X < Y + Z and Y < X + Z and Z < X + Y:
    print('Треугольник с введенными сторонами суще-
ствует')
else:
    print('Треугольник с введенными сторонами не су-
ществует')
```

При использовании оператора `or` (логическое ИЛИ) необходимо, чтобы выполнялось *хотя бы одно из условий*. Например, для равнобедренности треугольника достаточно равенства двух любых сторон, поэтому все возможные случаи равенств связываем в условной конструкции с помощью оператора `or`:

```
print('A, B, C - стороны треугольника')
X = int(input('Введите A: '))
Y = int(input('Введите B: '))
Z = int(input('Введите C: '))
print('В равнобедренном треугольнике есть две равные
стороны')
if X == Y or Y == Z or Z == X:
    print('Треугольник с введенными сторонами равно-
бедренный')
else:
    print('Треугольник с введенными сторонами не рав-
нобедренный')
```

Что нового мы узнали в третьей главе?

1. **Ветвление** — форма организации действий, при которой в зависимости от выполнения или невыполнения некоторого условия совершается определенная последовательность действий.

2. Существует две формы условной конструкции: *неполная* («если..., то...») и *полная* («если..., то..., иначе...») и три типа условных операторов (**if**, **if-else**, **elif**), позволяющих использовать условия в языке Python.

3. При необходимости можно создавать вложенные условия и комбинировать условные конструкции с помощью вспомогательных операторов **and** и **or**:

- при использовании оператора **and** *каждое из условий* должно быть истинным, только в этом случае будет выполняться действие, заложенное в результат условной конструкции;
- при использовании оператора **or** необходимо, чтобы выполнялось *хотя бы одно из условий*.

4. Существует всего шесть типов сравнения, но всего один тип результата сравнения — логический (**bool**).

5. Запись операторов сравнения на языке Python имеет отличия от привычных арифметических знаков:

| Знак сравнения в арифметике | Оператор сравнения на языке Python |
|-----------------------------|------------------------------------|
| = | == |
| ≠ | != |
| > | > |
| ≥ | >= |
| < | < |
| ≤ | <= |

6. В конце первой строки условной конструкции всегда ставится двоеточие, которое показывает, что после нажатия клавиши **Enter** будут перечислены действия, выполняющиеся только при истинности указанного выше условия. Python автоматически делает небольшой отступ вправо, называемый *табуля-*

цией. При этом с переходом на следующую строку нажатием клавиши **Enter** отступ сохраняется. Данный отступ также можно поставить с помощью клавиши **Tab**, а убрать с помощью клавиши **Backspace**.

7. Чтобы запустить условную конструкцию в интерактивной оболочке **IDLE**, после написания последнего действия необходимо нажать клавишу **Enter** дважды. Первое нажатие переводит строку для написания следующего действия, а второе выводит из условной конструкции и запускает код.

Практикум

Основные понятия главы

1. Приведите примеры условных конструкций:

1.1) из школьной жизни;

1.2) в походе;

1.3) при использовании мобильного телефона.

2. Приведите примеры истинных и ложных высказываний:

2.1) в математике;

2.2) в русском языке.

3. Определите, какие из высказываний демонстрируют неполную форму ветвления:

- Если собака лает, значит, по забору идет кошка;
- Если собака лает, значит, по забору идет кошка или пришел сосед;
- Если собака лает, значит, по забору идет кошка или пришел сосед, иначе что-то другое.

4. Заполните таблицу:

| Условие | Соответствующий вопрос | Ответ на вопрос | Истинность (1) / ложность (0) |
|----------------|------------------------|-----------------|-------------------------------|
| $12 >= 12 - 0$ | | | |
| $5 == 2 + 3.0$ | | | |

| Условие | Соответствующий вопрос | Ответ на вопрос | Истинность (1) / ложность (0) |
|------------------|------------------------|-----------------|-------------------------------|
| $7 < 2 * 3$ | | | |
| $0 > -23$ | | | |
| $5 \leq 0.2 + 3$ | | | |

5. Определите истинность или ложность условий:

| Условие | Истинность (1) / ложность (0) |
|-------------------------------------|-------------------------------|
| $(23 - 23) == (70 - 70)$ | |
| $7 \neq 2 * 3$ | |
| $(1 + 2 * 3 - 4) > (1 - 2 * 3 + 4)$ | |
| $2 ** 10 \geq 32 * 32$ | |
| $5 \% 2 < 5 // 2$ | |
| $10 / 2 \leq (2 + 100 \% 3)$ | |

Чтение кода

6. Определите результат работы следующих фрагментов программ, заполняя правый столбец таблицы:

| № | Фрагмент программы | Результат |
|-----|---|-----------|
| 6.1 | <pre>i = 10 if i > 1: i = i - 1 print(i)</pre> | |
| 6.2 | <pre>i = 1 if i > 1: i = i - 1 print(i) elif i == 1: i = (i + 1) * 3 + 1 print(i)</pre> | |

| № | Фрагмент программы | Результат |
|-----|---|-----------|
| 6.3 | <pre> i = int(-3) if i > 1: i = i - 1 print(i) elif i == 1: i = i + 1 print(i*2) else: print(i**2) </pre> | |
| 6.4 | <pre> i = 25 if i <= 0: print(i) elif i < 0: i = i-10 if i <= 0: print(i) elif i > 0: i = i-10 if i <= 0: print(i) else: print('error') </pre> | |
| 6.5 | <pre> i = 10 if i > 1 and i < 10: i = i - 1 print(i) elif i == 1 or i == 10: i = i + 1 print(i) print(i + 1) </pre> | |
| 6.6 | <pre> i = 4 if i > 1 and i < 10: i = i * 2 print (i) if i > 1 and i < 10: i = i * 2 print (i) </pre> | |

| № | Фрагмент программы | Результат |
|---|--|-----------|
| | <pre> if i > 1 and i < 10: print (i * 2) else: print (i + 2) else: print (i + 2) else: print(i) </pre> | |

Поиск ошибок

7. Объясните, почему в следующих частях программы Python выведет ошибку, используя варианты ответов:

- A. Переменная не определена.
- B. Невозможное присваивание.
- C. Аргумент не строкового типа.
- D. Количество открытых скобок не равно количеству закрытых.
- E. Невозможная операция сложения/конкатенации.
- F. Невозможная операция сравнения.
- G. Отсутствие отступа.
- H. Неверный оператор сравнения.

| № | Код | Ошибка |
|-----|--|--------|
| 7.1 | <pre> i = int(input()) if i > 1: i = i - 1 print(i) elif i == 1: i = (i + 1)*3) + 1 print(i) </pre> | |
| 7.2 | <pre> a = float(input("a = ")) sum = a + b if sum > 10: i = 'Try again' print(i) </pre> | |

| № | Код | Ошибка |
|-----|--|--------|
| 7.3 | <pre>i = 10 if i > 1 and i < 10: i = i - 1 print(i) elif i = 1 or i = 10: i = i + 1 print(i) print(i+1)</pre> | |
| 7.4 | <pre>i = '5' if i > 1: a = input('a = ') i = i + a print(i) elif i == 1: b = input('a = ') i = i + b print(i) else: print(i*2)</pre> | |
| 7.5 | <pre>i = int(input(Введите число)) if i <= 0: print(i) elif i > 0: i = i - 10 if i <= 0: print(i) elif i > 0: i = i - 10 if i <= 0: print(i) else: print('error')</pre> | |

| № | Код | Ошибка |
|-----|--|--------|
| 7.6 | <pre>c = input("a = ") i = c + 3 if c > 1: i = i - 1 print(i) elif c == 0: i = i + 1 print(i)</pre> | |
| 7.7 | <pre>i = 25 if i <= 0: print(i) elif i > 0: i = i - 10 if i <= 0: print(i) elif i > 0: i = i - 10 if i <= 0: print(i) else: print('error')</pre> | |
| 7.8 | <pre>i = 4 if i > 1 and i < 10: i * 2 = i print (i) if i > 1 and i < 10: i * 2 = i print (i) if i > 1 and i < 10: print (i*2) else: print (i+2) else: print (i+2) else: print(i)</pre> | |

8. Добавьте в необходимых строках отступ, чтобы результат работы программы совпал со значением в правом столбце:

| № | Фрагмент программы | Результат |
|-----|--|-----------|
| 8.1 | <pre>i = 100 if i == 100: i = i - int('96') print(i)</pre> | 4 |
| 8.2 | <pre>i = 1 if i > 1: i = 1 elif i == 1: print(i) print(i)</pre> | 1 1 |

Оптимизация

9. Программа решает задачу нахождения площади ромба со стороной 10 и высотой 8:

```
a = 10
h = 8
s = 0.5*a*h
print(s)
```

Опишите возможный вариант измененного кода, используя следующие пункты оптимизации:

- увеличение спектра решаемых задач (усиление универсальности);
- понятный диалог с пользователем;
- предусмотрение различного рода ошибок пользователя.

Разработка

10. Напишите программу, которая выводит «ENG» при условии, что пользователь вводит с клавиатуры пароль «qwerty», и выводит «РУС» при условии, что пользователь вводит с клавиатуры «йцукен».

11. Напишите программу, которая просит пользователя ввести целое число. Далее:

- если число оказалось отрицательным, то выводит предыдущее число;
- если число оказалось положительным, то выводит следующее число;
- в любом другом случае программа выводит «0».

12. Напишите программу, которая определяет четность или нечетность целого числа.

Пример:

Введите число: 5
Число 5 нечетное.

13. Напишите программу, которая запрашивает два числа a и b и выводит результаты $a + b$, $a - b$, $a \cdot b$, a / b . При этом если $b = 0$, то выводится сообщение о несуществовании результата a / b .

Пример:

Введите a: -6
Введите b: 0
 $a + b = -6.0 + 0.0 = -6.0$
 $a - b = -6.0 - 0.0 = -6.0$
 $a * b = -6.0 * 0.0 = 0.0$
 a / b - не существует (делить на 0 нельзя!)

14. Напишите программу, которая определяет, входит ли число, введенное пользователем, в отрезок $[-50;50]$.

Пример:

Введите число: -50
Число -50.0 принадлежит диапазону $[-50;50]$.

15. Напишите программу, которая определяет, входит ли число, введенное пользователем, в полуинтервал $(-50;50]$.

Пример:

Введите число: -50
Число -50.0 не принадлежит диапазону $(-50;50]$.

16. Напишите программу, которая запрашивает левую (a) и правую (b) границы числового промежутка и спрашивает, какой тип числового промежутка вывести на экран. При этом:

- если пользователь вводит строку «интервал», то на экран выводится числовой промежуток в формате ($a; b$);
- если пользователь вводит строку «полуинтервалы», то на экран выводятся два промежутка в формате ($a; b$) и [$a; b$];
- если пользователь вводит строку «отрезок», то на экран выводится числовой промежуток в формате [$a; b$].

Пример:

Введите левую границу промежутка: -200.3

Введите правую границу промежутка: 0

Вывести интервал, полуинтервалы или отрезок с заданными границами? - Полуинтервалы

Полуинтервалы: (-200.3; 0] и [-200.3; 0)

17. Напишите программу, которая упорядочивает цифры двузначного положительного числа от меньшего к большему и выводит на экран полученный результат. При этом если введено не двузначное число, на экран выводится ошибка.

Пример:

Введите двузначное положительное число: 71

17

18. Напишите программу, которая выводит «зеркально отраженное» четырехзначное положительное число. При этом если введено не четырехзначное число или получившееся число не является четырехзначным, на экран выводится ошибка.

Пример:

Введите четырехзначное положительное число: 7198

8917

19. Напишите программу, которая проверяет, является ли билет с четырехзначным номером «счастливым по-питерски» (сумма первых двух цифр равна сумме последних двух). При этом если введено не четырехзначное число, на экран выводится ошибка.

Пример:

Введите номер билета: 1762

Билет № 1762 – счастливый по-питерски!

20. Напишите программу, которая упорядочивает цифры трехзначного положительного числа от большего к меньшему и выводит на экран полученный результат. При этом если введено не трехзначное число, на экран выводится ошибка.

Пример:

Введите трехзначное положительное число: 691

961

21. Напишите программу, показывающую пользователю символы, которые можно использовать для сложения, умножения, вычитания и деления, затем предлагает ввести арифметическое выражение и вычисляет его. В задаче используйте: функцию `eval()`, переход на следующую строку `\n`, тройные кавычки.

Пример:

Используемые операции и их обозначения:

Сложение – '+'

Вычитание или унарный минус – '-'

Умножение – '*'

Введите арифметическое выражение, используя данные операции: $-3*3+3$

Ответ: 6

Глава 4. Циклы

Представим задачу, в которой необходимо вывести на экран компьютера все натуральные числа от 1 до 20. Неужели придется прописать все числа для вывода или задать первоначальное число $n=1$, а после выводить через `print(n, n+1, n+2, ..., n+19)`? С таким положением вещей никак нельзя мириться. Для повторяющихся команд (в нашем случае прибавление единицы к предыдущему числу) существуют циклы. Они позволяют не прописывать одинаковые команды, экономя время на написание кода и, в случае необходимости, на исправление в нем ошибок.

Таким образом, многократно повторяющаяся последовательность действий в программировании называется *циклом*. При этом сама последовательность, предназначенная для повторения, называется *телом цикла*, а единоразовое прохождение тела цикла — *итерацией*.

Предложенную задачу можно описать на Python с помощью двух типов циклической конструкции: *с заданным числом повторений* и *с предусловием*. Рассмотрим оба варианта¹.

Цикл с заданным числом повторений — `for`

Рассмотрим первый вариант цикла (с заданным числом повторений). Для него используется оператор цикла `for` (англ. «для»):

```
for n in range(1, 21):  
    print(n)
```

Тип цикла: с заданным числом повторений.

Количество повторений: 20, заданное с помощью функции `range(1, 21)`.

Характеристика итерации: Вывод на экран содержимого переменной n , которое меняется после каждой итерации на следующее число диапазона, созданного с помощью функции `range()`.

Количество итераций: 20.

¹ Если ты изучал другие языки программирования, то ожидал увидеть цикл с постусловием. В Python такой тип цикла отсутствует.

Функция `range(1, 21)` в данном коде представляет собой список целых чисел от 1 до 21, включая 1 и не включая 21, т. е. следующий набор чисел¹:

[1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20].

Другими словами, каждую итерацию цикла `for` можно представить следующим образом:

| | | | | | |
|-----------------------|-----------------------|-----------------------|-----------------------|-----|------------------------|
| n | 1 | 2 | 3 | ... | 20 |
| <code>print(n)</code> | <code>print(1)</code> | <code>print(2)</code> | <code>print(3)</code> | ... | <code>print(20)</code> |

Стоит также отметить, что `range()` — самостоятельная функция и используется не только в циклах. Поэтому рассмотрим ее подробнее.

Функция range()

В общем случае функция `range()` имеет три аргумента:

`range(от ..., до ... не включая, с шагом ...)`.

- Если предъявлен только один аргумент функции, то первый аргумент принимается за ноль, второй — за введенный, а третий — за единицу, т. е. `range(A)` то же самое, что `range(0, A, 1)`.
- Если последний аргумент из трех не прописывается, то он автоматически принимается за единицу, т. е. `range(A, B)` то же самое, что `range(A, B, 1)`.

Примеры:

| | | |
|------------------------------|-----------------------------------|---------------------|
| <code>range(5)</code> | от 0, до 5 не включая, с шагом 1 | [0, 1, 2, 3, 4] |
| <code>range(1, 4)</code> | от 1, до 4 не включая, с шагом 1 | [1, 2, 3] |
| <code>range(-1, 5, 1)</code> | от -1, до 5 не включая, с шагом 1 | [-1, 0, 1, 2, 3, 4] |

¹ В программировании принята запись, которой мы будем пользоваться в дальнейшем: «от 1 включая, до 21 не включая».

| | | |
|------------------------------|-----------------------------------|-------------------------------|
| <code>range(-2, 8, 2)</code> | от -2, до 8 не включая, с шагом 2 | <code>[-2, 0, 2, 4, 6]</code> |
| <code>range(8, 0, -3)</code> | от 8, до 0 не включая, с шагом -3 | <code>[8, 5, 2]</code> |

Переменная-счетчик

Переменная-счетчик — это специальная переменная, которая обычно используется для подсчета количества итераций цикла.

Впервые с такой переменной мы встречаемся именно в цикле `for`, без нее он просто невозможен. В нашем примере эта переменная носит имя `i`. Инструкция `for i in range(1, 21)` читается следующим образом: «для `i` в диапазоне от 1 до 21 не включая». Механизм цикла таков, что переменной `i` присваивается начальное значение (в данном случае 1) и далее к ней прибавляется заданный шаг (у нас по умолчанию шаг равен единице) до тех пор, пока она не достигнет последнего значения (в нашем случае 20). Как только `i` становится равна 20, цикл завершается.

В Python возможности переменной-счетчика несколько шире, чем в других языках программирования, благодаря оператору `in`, который фактически может «положить» в нее любые значения и объекты. Для примера рассмотрим программу, которая определяет, содержит ли введенное слово букву «к»:

```
a = input('Введите слово: ')
for i in a:
    if i == 'к':
        print('В этом слове есть буква "к"')
```

Такой алгоритм будет перебирать все символы строки `a`, проверяя, является ли он искомым, т.е. «к». При этом каждый символ строки будет помещаться в переменную-счетчик `i`.

В дальнейшем мы будем применять этот прием при работе со многими структурами данных.

Цикл с предусловием — while

Второй тип цикла — с предусловием, для него используется оператор `while`. Он применяется тогда, когда число повторений тела цикла заранее *неизвестно*, но может определяться каким-либо

условием вида «*пока...*». Например, пока пользователь не введет верный пароль, будем запрашивать его снова и снова.

Важно!

Цикл начинает итерацию выполнения тела только в случае *истинности* проверяемого условия. То есть такой цикл может не выполниться ни разу, если условие изначально будет *ложным*.

Рассмотрим вариант решения задачи вывода на экран всех натуральных чисел до 20 с помощью данного цикла.

Зададим в начале программы самое первое натуральное число, а далее будем увеличивать его на единицу при каждой итерации цикла и выводить число на экран, пока не дойдем до 20:

```
n = 1
while n <= 20:
    print(n)
    n = n + 1
```

Тип цикла: с предусловием.

Предусловие: $n \leq 20$.

Характеристика итерации: Каждая итерация выводит на экран число, содержащееся в переменной n , и увеличивает ее на единицу.

Количество итераций: 20.

На 20-й итерации цикла сначала выводится «20», затем значение переменной n становится равным 21, что перестает удовлетворять условию $n \leq 20$, и цикл завершается.

Порядок действий в теле цикла крайне важен. Если в программе с циклом поменять местами действие печати и увеличения на единицу, то первым числом, выведенным на экран, будет число 2, поскольку первоначальное значение переменной n уже было задано до начала цикла.

Итак, отличительные особенности рассмотренных циклов таковы:

- цикл **for** выполняется заданное количество раз;
- цикл **while** выполняется до того момента, пока истинно проверяемое перед телом цикла условие.

Операторы `break` и `continue`

В Python и других языках программирования существуют специальные операторы на случай «непредвиденных обстоятельств» внутри цикла. Это может быть пропуск одного и более шагов (оператор `continue`) или досрочное его завершение (оператор `break`). Иначе их называют директивами, т. е. указаниями для компьютера.

Вывод натуральных чисел, прерывающийся на числе 5:

```
a = 0
while True:
    a = a + 1
    if a == 6:
        break
    print(a)
```

Внимание!

Приведен пример организации бесконечного цикла: условие всегда истинно (`True`).

Вывод натуральных чисел от 1 до 10, с пропуском чисел 5 и 8:

```
for a in range(11):
    if a == 5 or a == 8:
        continue
    print(a)
```

Вложенные циклы

Циклические конструкции могут быть вложены друг в друга точно так же, как и условные.

Рассмотрим пример, где необходимо вычислить количество элементов матрицы¹ (ячеек таблицы), состоящей из 3 строк и 4 столбцов, и вывести результат на экран. Для удобства обозначим эле-

¹ Математический объект, представляющий собой набор упорядоченных элементов, который обычно записывается в виде прямоугольной таблицы, состоящей из строк и столбцов.

менты матрицы символами a_{ij} , где индекс i — номер строки, а индекс j — номер столбца:

| | | | |
|----------|----------|----------|----------|
| a_{11} | a_{12} | a_{13} | a_{14} |
| a_{21} | a_{22} | a_{23} | a_{24} |
| a_{31} | a_{32} | a_{33} | a_{34} |

Из таблицы видно, что значение i меняется от 1 до 3, а значение j — от 1 до 4. Можно предложить как минимум два варианта прохода по всем ячейкам таблицы:

- 1) проходить все ячейки по строкам;
- 2) проходить все ячейки по столбцам.

Приведем для примера код вложенной циклической конструкции **for** для первого варианта:

```
count = 0
for i in range(1, 4):
    for j in range(1, 5):
        count = count + 1
print(count)
```

Составим таблицу итераций для данной программы. Для удобства обозначим начальное значение переменной `count` нулевым номером итерации.

| Номер итерации | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 |
|----------------|---|---|---|---|---|---|---|---|---|---|----|----|----|
| i | | 1 | 1 | 1 | 1 | 2 | 2 | 2 | 2 | 3 | 3 | 3 | 3 |
| j | | 1 | 2 | 3 | 4 | 1 | 2 | 3 | 4 | 1 | 2 | 3 | 4 |
| $count$ | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 |

Результат: 12

Если выбрать вариант прохода ячеек по столбцам, то поменяются местами только вторая и третья строки программы.

Что нового мы узнали в четвертой главе?

1. Многократно повторяющаяся последовательность действий в программировании называется **циклом**. При этом сама последовательность, предназначенная для повторения, называется **телом цикла**, а единоразовое прохождение тела цикла — **итерацией**.

2. Порядок действий в теле цикла крайне важен в Python.

3. Существуют два вида циклов: *с заданным числом повторений* (цикл **for**) и *с условием* (цикл **while**);

4. Стандартная функция `range()` организует список чисел и в общем виде имеет три аргумента: `range(от ..., до ... не включая, с шагом ...)`.

При этом могут быть предъявлены один или два аргумента из трех:

- `range(A) = range(0, A, 1)`;
- `range(A, B) = range(A, B, 1)`.

5. Задача переменной-счетчика состоит в том, чтобы определять количество итераций цикла и момент, когда цикл должен быть завершен.

6. Возможность досрочного завершения цикла обеспечивает директива **break**, а возможность пропуска итерации — директива **continue**.

7. Циклические конструкции могут быть вложены друг в друга.

Важно!

Часто в коде можно встретить операторы `+=` и `-=`, которые соответственно увеличивают и уменьшают операнд слева на числовое значения операнда справа. Об их существовании нужно и важно знать, поэтому в практике вы также поработаете с ними.

| Привычная запись | Новая запись |
|------------------------|---------------------|
| <code>a = a + 1</code> | <code>a += 1</code> |
| <code>c = c - 1</code> | <code>c -= 1</code> |

Практикум

Основные понятия главы

1. Заполните пустые ячейки таблицы, используя знания о функции `range()`:

| | | |
|------------------------------|---|-------------------------|
| | от __, до __ не включая, с шагом __ | $[-6, -5, -4]$ |
| <code>range(9)</code> | от __, до __ не включая, с шагом __ | |
| | от 7, до -3 не включая, с шагом -5 | |
| <code>range(1, 4)</code> | от __, до __ не включая, с шагом __ | |
| | от 0, до 4 не включая, с шагом 4 | |
| <code>range(-6, 2, 3)</code> | от __, до __ не включая, с шагом __ | |
| | от 5, до 10 не включая, с шагом 1 | |
| | от __, до __ не включая, с шагом __ | $[0, 1, 2, 3, 4, 5, 6]$ |
| <i>Вариант 1:</i> | от __, до __ не включая, с шагом __ | $[-10, -5, 0]$ |
| <i>Вариант 2:</i> | от __, до __ не включая, с шагом __ | |
| <i>Вариант 3:</i> | от __, до __ не включая, с шагом __ | |
| <i>Вариант 4:</i> | от __, до __ не включая, с шагом __ | |
| <i>Вариант 5:</i> | от __, до __ не включая, с шагом __ | |

Чтение кода

2. Опишите характеристики следующих циклических конструкций:

| | |
|-----|---|
| 2.1 | <pre>s = 0 for k in range(3,10): s = s + 6 print(s)</pre> <p>Тип цикла:</p> <p>Количество повторений/предусловие:</p> <p>Характеристика итерации:</p> <p>Количество итераций:</p> |
| 2.2 | <pre>k = 0 for m in range(1,10,2): k = k + 6 print(k)</pre> <p>Тип цикла:</p> <p>Количество повторений/предусловие:</p> <p>Характеристика итерации:</p> <p>Количество итераций:</p> |
| 2.3 | <pre>s = 10 for k in range(5): s -= k print(s)</pre> <p>Тип цикла:</p> <p>Количество повторений/предусловие:</p> <p>Характеристика итерации:</p> <p>Количество итераций:</p> |

3. Опишите характеристики следующих циклических конструкций:

| | |
|------------|--|
| 3.1 | <pre>s = -30 while s < 0: s += 6 print(s)</pre> |
| | Тип цикла: |
| | Количество повторений/предусловие: |
| | Характеристика итерации: |
| | Количество итераций: |
| 3.2 | <pre>s = 1 n = 15 while n > 0: n = n - s s = s * 2 print(n)</pre> |
| | Тип цикла: |
| | Количество повторений/предусловие: |
| | Характеристика итерации: |
| | Количество итераций: |
| 3.3 | <pre>k = 1 while k <= 64: k = k*2 print(k)</pre> |
| | Тип цикла: |
| | Количество повторений/предусловие: |
| | Характеристика итерации: |
| | Количество итераций: |

4. Составьте таблицы итераций для программ из заданий 2 и 3.
5. Измените с помощью оператора **break** первый программный код из задания 2 таким образом, чтобы конструкция завершилась при $s = 30$.
6. Измените с помощью оператора **continue** последний программный код из задания 3 таким образом, чтобы не были напечатаны числа 8 и 64.
7. Последний программный код из задания 2 измените таким образом, чтобы был использован цикл **while**.
8. Первый программный код из задания 3 измените таким образом, чтобы был использован цикл **for**.
9. Определите результат работы следующих фрагментов программ, заполняя правый столбец таблицы:

| № | Фрагмент программы | Результат |
|-----|--|-----------|
| 9.1 | <pre>for i in range(25,30,2): print(i)</pre> | |
| 9.2 | <pre>i = 7 while i > 0: i = 2*i-10 print(i)</pre> | |
| 9.3 | <pre>for i in range(3): while i != 0: print('Ура!') i = i - 1</pre> | |
| 9.4 | <pre>a = 35 while a <= 50: if a == 43 or a == 47: continue if a % 2 == 1: print(a) a += 2</pre> | |
| 9.5 | <pre>a = 'Key' b = 2021 for i in a: print(b % 10) b = b // 10</pre> | |

Поиск ошибок

10. Учащийся написал программу, которая подсчитывает количество ячеек в таблице, содержащей 3 строки и 4 столбца, с помощью циклического оператора **while**, но после запуска программы на экран выводится число 4. Найдите ошибку в коде программы и исправьте ее.

```
count = 0
i = 1
j = 1
while i <= 3:
    while j <= 4:
        count = count + 1
        j = j + 1
    i = i + 1
print(count)
```

11. Объясните, почему в следующих частях программы Python выведет ошибку, используя варианты ответов:

- А. Переменная не определена.
- В. Невозможное присваивание.
- С. Аргумент не строкового типа.
- Д. Количество открытых скобок не равно количеству закрытых.
- Е. Невозможная операция сложения/конкатенации.
- Ф. Невозможная операция сравнения.
- Г. Отсутствие отступа.
- Н. Неверный оператор сравнения.

| № | Код | Ошибка |
|------|---|--------|
| 11.1 | <pre>4 = i while i > 0: if i%2 == 0: print(i, "- четное") else: print(i, "- нечетное")</pre> | |
| 11.2 | <pre>for i in range(5): print(a)</pre> | |
| 11.3 | <pre>for i in range(3): a = int(input(Введите число)) print(a*5)</pre> | |

| № | Код | Ошибка |
|------|---|--------|
| 11.4 | <pre>count1 = 0 count2 = 0 for t in range(7): while t > 0 and t < 4: count1 += 1 print('1'*count1) while t >= 4 and t < 7: count2 += 1 print('2'*count2) if (count1 + count2) = '11222': print('Yes') else: print('No')</pre> | |
| 11.5 | <pre>i = int(input()) while i >= 0: i = i - 1 print(i)</pre> | |
| 11.6 | <pre>for i in range(7,100,6): i = str(i) + '0' print(i)</pre> | |
| 11.7 | <pre>a = int(input("a = ")) b = input("b = ") while a + b > 100: print("True")</pre> | |
| 11.8 | <pre>i = int(input()) while i < 20 and i >= 0: if i = 5: continue print(i) i += 1</pre> | |

Оптимизация

12. Программа решает задачу проверки делимости пятизначного натурального числа на 3.

```
a = 52344
s = 0
```

```
while a != 0:
    s += a%10
    a = (a - a%10)//10
if s/3 == int(s/3):
    print(True)
else:
    print(False)
```

Опишите возможный вариант измененного кода, используя следующие пункты оптимизации:

- увеличение спектра решаемых задач (усиление универсальности);
- ведение понятного диалога с пользователем;
- предусмотрение различного рода ошибок пользователя.

Разработка

13. Напишите программу, которая подсчитывает количество ячеек в таблице, содержащей 5 строк и 2 столбца. При этом:

- 13.1)** внешний цикл проходит все ячейки таблицы по столбцам;
- 13.2)** внешний цикл проходит все ячейки таблицы по строкам.

14. С помощью цикла **for**:

- 14.1)** выведите на экран числа от 25 до 50;
- 14.2)** выведите на экран только четные числа из диапазона от 25 до 50;
- 14.3)** выведите на экран все нечетные числа от 25 до 50, кроме тех, что больше 45, с помощью директивы **continue**.

15. С помощью цикла **while**:

- 15.1)** выведите на экран числа от 50 до 25;
- 15.2)** выведите на экран только нечетные числа от 25 до 50;
- 15.3)** выведите на экран все четные числа от 25 до 50, кроме числа 46, без директивы **continue**.

16. Напишите программу, которая запрашивает у пользователя натуральное число, а затем печатает каждую его цифру, начиная с конца, на отдельной строке.

17. Напишите программу, которая вычисляет факториал не-отрицательного числа, введенного с клавиатуры, и выводит ошибку, если пользователь ввел отрицательное число.

Факториал числа n ($n!$) — это произведение натуральных чисел от 1 до n . Например, $5! = 1 \cdot 2 \cdot 3 \cdot 4 \cdot 5 = 5 \cdot 4 \cdot 3 \cdot 2 \cdot 1$. При этом $0! = 1$.

18. Напишите программу, которая определяет, является ли введенное натуральное число палиндромом.

Палиндром — число, буквосочетание, слово или текст, одинаково читающееся в обоих направлениях.

19. Напишите программу, находящую наибольший общий делитель (НОД) двух целых положительных чисел a и b , вводимых с клавиатуры.

Глава 5. Псевдослучайные числа и математика

Подключение модулей и библиотек

Помимо встроенных возможностей языка, таких как автоматическое определение типа данных, операций, которые можно с ними производить, функций `print()`, `range()`, в Python реализована возможность подключения дополнительных модулей.

Модуль — это своеобразная библиотека с готовыми функциями (алгоритмами) и операторами, которые позволяют сократить программный код за счет реализованных в них операций.

Информацию о любом модуле и его содержимом можно найти в сети Интернет. Более того, одним из самых важных навыков для программиста является умение находить готовые модули и подключать их в своей программе для упрощения решения задач, а также, на более продвинутом уровне, создавать свои собственные модули и делиться ими с другими программистами¹.

Подключаются модули с помощью оператора **import** до начала использования их содержимого. Обычно это делают в самом начале программы — удобнее, когда все внешние модули собраны в одном месте и их содержимое доступно в любой части основной программы.

Псевдослучайные числа

Использование случайных величин не является встроенной возможностью языка, хотя разработчики игр часто используют их для возможности выбора случайной карты, цвета, количества игроков и т. д. Поэтому возникает необходимость подключения такого модуля.

Рассмотрим использование в программе случайных, а если быть точнее, *псевдослучайных чисел*. Естественно, машина не

¹ Хотя модули занимают много строк кода и могут показаться неоптимальными по требуемой памяти, всё же они делают код более универсальным, чем функции, заново «изобретенные» программистом (в большинстве случаев), так как над модулями работает сообщество.

может взять произвольное число «из воздуха». Ей нужны некоторые границы для выборки, а также понимание того, какое число необходимо: целое или вещественное, положительное или отрицательное. Ведь из бесконечной прямой с бесконечным количеством чисел для компьютера просто невозможно выбрать какое-то одно, как минимум потому, что его память ограничена и, по сути, понятие бесконечности для компьютера не существует.

Случайные числа, моделируемые компьютером, называются *псевдослучайными*, поскольку компьютер использует для их генерации особые алгоритмы, основанные на вероятностном выборе одного числа из ограниченного диапазона. В то же время сами эти алгоритмы были тоже разработаны программистами, использовавшими сложные математические вычисления, и для простого пользователя угадывание псевдослучайного числа оказывается трудновыполнимой задачей.

Модуль `random`

Итак, чтобы использовать различные функции, генерирующие псевдослучайные числа, необходимо подключить модуль **`random`**.

Напишем в начале программы

```
import random
```

Модуль **`random`** содержит множество методов или функций, генерирующих псевдослучайные числа, которые можно выбрать в зависимости от целей задачи:

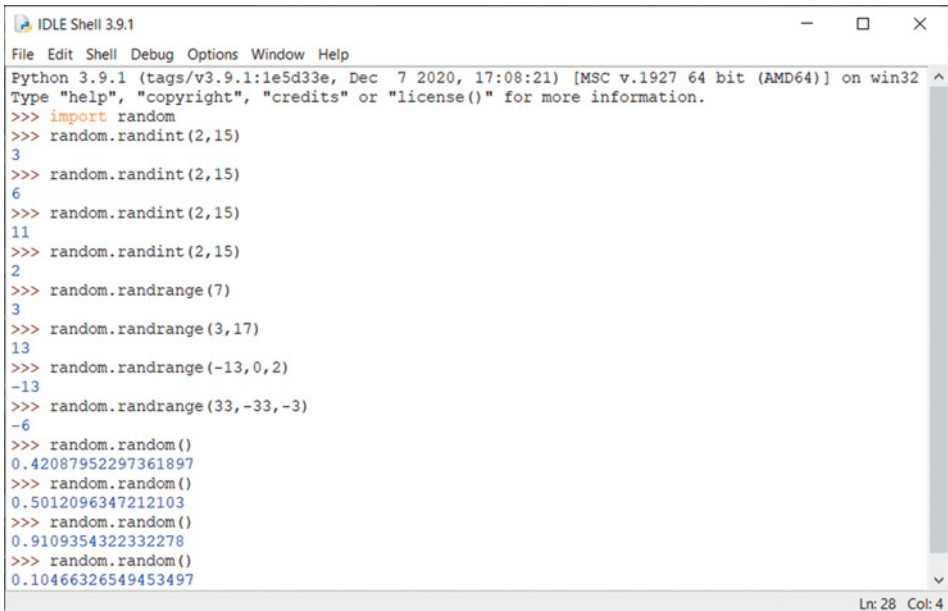
| Функция | Назначение функции |
|-------------------------------------|--|
| <code>random.randint(A, B)</code> | Возвращает псевдослучайное целое число из луча от A до B (не включая B) |
| <code>random.randrange(B)</code> | Возвращает псевдослучайное число из последовательности целых чисел от 1 до B (не включая B) |
| <code>random.randrange(A, B)</code> | Возвращает псевдослучайное число из последовательности целых чисел от A до B (не включая B) |

| Функция | Назначение функции |
|---|--|
| <code>random.randrange (A, B, C)</code> | Возвращает псевдослучайное число из последовательности целых чисел от <i>A</i> до <i>B</i> (не включая <i>B</i>) с шагом <i>C</i> |
| <code>random.random ()</code> | Возвращает псевдослучайное вещественное число от 0.0 до 1.0 |

Заметьте, функция `randint()` содержит в своем названии сочетание букв «int», `randrange()` — «range», поэтому несложно запомнить, за что отвечают эти функции.

Ознакомиться с полным списком функций модуля **random** можно здесь: <https://pythonworld.ru/moduli/modul-random.html>.

Возвращаемся к интерактивному режиму IDLE, чтобы поэкспериментировать с данным модулем и вышеуказанными функциями (рис. 28).



```

IDLE Shell 3.9.1
File Edit Shell Debug Options Window Help
Python 3.9.1 (tags/v3.9.1:1e5d33e, Dec 7 2020, 17:08:21) [MSC v.1927 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>> import random
>>> random.randint(2,15)
3
>>> random.randint(2,15)
6
>>> random.randint(2,15)
11
>>> random.randint(2,15)
2
>>> random.randrange(7)
3
>>> random.randrange(3,17)
13
>>> random.randrange(-13,0,2)
-13
>>> random.randrange(33,-33,-3)
-6
>>> random.random()
0.42087952297361897
>>> random.random()
0.5012096347212103
>>> random.random()
0.9109354322332278
>>> random.random()
0.10466326549453497
Ln: 28 Col: 4

```

Рис. 28. Демонстрация работы функций `randint()`, `randrange()`, `random()` в интерактивном режиме оболочки IDLE

Игра «Угадай-ка!»

Попробуем поиграть с компьютером, написав программу для игры «Угадай-ка!», которая заключается в том, что компьютер единожды генерирует целое число из предложенного диапазона, а пользователь пытается угадать его, предлагая свои варианты, за определенное количество попыток. При этом если пользователь не угадывает число, то программа выводит сообщение о том, больше сгенерированное число компьютером или меньше того, что ввел пользователь. Если пользователь угадал число, то выводится сообщение о выигрыше, если истратил все попытки, но не угадал — сообщение о проигрыше.

Первый этап. Опишем «скелет» программы:

- подключение модуля;
- создание переменных для чисел компьютера и пользователя;
- написание основной условной конструкции игры.

```
#Подключение модуля генератора чисел
import random
#Кладем в a псевдослучайное число от 0 до 10:
a = random.randint(0,10)
#Кладем в b целое число, введенное пользователем
#с клавиатуры:
b = int(input('Введите число: '))
#Записываем основную конструкцию условия:
if a == b:
    print('Поздравляем! Вы выиграли!')
elif a > b:
    print('Загаданное число больше введенного')
else:
    print('Загаданное число меньше введенного')
```

На следующем этапе добавим переменную, отвечающую за количество попыток, и цикл, в котором условия проверяются только заданное количество раз и, соответственно, пользователь может вводить число только до тех пор, пока попытки не будут исчерпаны.

Второй этап. Ввод ограничения на количество попыток с помощью объявления новой переменной и цикла **while**:

```
import random
a = random.randint(0,10)
```

```
#Объявляем переменную, отвечающую за количество
#попыток, присваиваем ей начальное значение 0
p = 0
#Объявляем цикл с условием "p строго меньше 5"
while p < 5:
    b = int(input('Введите число: '))
#Все условия, предназначенные для игры, сдвигаются
#вправо
    if a == b:
        print('Поздравляем! Вы выиграли!')
    elif a > b:
        print('Загаданное число больше введен-
ного')
    else:
        print('Загаданное число меньше введен-
ного')
#Не забываем увеличивать на единицу количество
#израсходованных попыток, иначе смысла в p < 5 нет
#(0 всегда меньше 5)
    p = p + 1
```

На данном этапе решен лишь вопрос с количеством попыток, но если пользователь угадал число в начале игры, компьютер все равно снова и снова выводит запрос на ввод числа. Необходимо добавить уже знакомый оператор **break**, который остановит работу цикла, как только пользователь угадает число, после условия на проверку равенства.

Третий этап. Ввод инструкции об условии досрочного завершения игры с помощью оператора **break**:

```
import random
a = random.randint(0,10)
p = 0
while p < 5:
    b = int(input('Введите число: '))
    if a == b:
        print('Поздравляем! Вы выиграли!')
        break
    elif a > b:
        print('Загаданное число больше введенного')
    else:
        print('Загаданное число меньше введенного')
    p = p + 1
```

Эта проблема теперь тоже исправлена, однако еще не учтен тот вариант, когда пользователю не хватает попыток, чтобы угадать сгенерированное компьютером число. Поэтому в самом конце программы добавим условие, чтобы после прохождения пятой итерации цикла (0, 1, 2, 3, 4) программа выводила на экран сообщение о проигрыше.

Четвертый этап. Ввод инструкции для компьютера на случай неугаданного числа за предусмотренное количество ходов пользователя:

```
import random
a = random.randint(0,10)
p = 0
while p < 5:
    b = int(input('Введите число: '))
    if a == b:
        print('Поздравляем! Вы выиграли!')
        break
    elif a > b:
        print('Загаданное число больше введенного')
    else:
        print('Загаданное число меньше введенного')
    p = p + 1
if p == 5:
    print('Вы проиграли. Загаданное число:', a)
```

Игра готова! Теперь можно продемонстрировать ее друзьям и посмотреть, кто окажется умнее компьютера!

Модуль math

Для работы с числами часто пригождается библиотека математических функций **math**.

Напомним, что перед тем как использовать функции этой библиотеки, необходимо подключить модуль с помощью оператора **import** в программе до начала его использования (например, в начале программы):

```
import math
```

Чтобы использовать в программе функции данного модуля, необходимо указывать его название, а после точки необходимую функцию — это правило действует в Python всегда. Например:

- `math.ceil(X)` — округление X до ближайшего большего числа;
- `math.sqrt(X)` — квадратный корень из X ;
- `math.pi` — число «пи»¹: $\pi = 3,1415926... .$

Посмотреть полный список функций модуля **math** можно здесь: <https://pythonworld.ru/moduli/modul-math.html>. На сайте pythonworld.ru можно найти другие модули и использовать их при написании программ.

Библиотека **math** входит в стандартный набор библиотек, устанавливаемых с Python. Если какая-либо библиотека отсутствует на компьютере, ее можно найти в сети Интернет.

Использование псевдонимов, оператор as

В общем случае модуль в Python — это просто файл, содержащий код на данном языке. Поэтому можно использовать любой созданный вами файл с программным кодом на языке Python как модуль в другом файле, выполнив в нем команду **import**. Другими словами, сослаться на другую программу или библиотеку, не описывая снова функции, которые в них уже есть.

Помимо подключения файлов с помощью команды **import**, Python дает возможность использовать псевдонимы для модулей — более короткие названия, устанавливаемые с помощью оператора **as** (от англ. «как», «в качестве»).

В качестве примера рассмотрим подключение уже знакомых библиотек **random** и **math**:

| Вызов модулей без использования псевдонима | Вызов модулей с использованием псевдонима |
|--|--|
| <pre>import math import random math.sqrt(X) random.randint(0,10)</pre> | <pre>import math as m import random as rnd m.sqrt(X) rnd.randint(0,10)</pre> |

¹ Число «пи» (π) — математическая константа, которая выражает отношение длины окружности к её диаметру.

Что нового мы узнали в пятой главе?

1. *Модуль* — это библиотека с готовыми функциями и операторами, которые позволяют сократить программный код за счет реализованных в них операций. Подключаются с помощью оператора **import** до начала использования их содержимого (чаще всего в начале программы).

2. *Модуль random* — библиотека функций, генерирующих псевдослучайные числа. Полный список функций модуля **random**: <https://pythonworld.ru/moduli/modul-random.html>.

3. *Модуль math* — библиотека математических функций для работы с числами. Полный список функций модуля **math**: <https://pythonworld.ru/moduli/modul-math.html>.

4. Если какая-либо библиотека отсутствует на компьютере, ее можно найти в сети Интернет.

5. *Оператор as* позволяет использовать псевдоним для модуля, т. е. удобное для программиста название для применения его в конкретной программе.

Практикум

Основные понятия главы

1. Определите возможные варианты подключения модуля **math**:

| № | Код | Возможно (+) / невозможно (-) |
|-----|--|-------------------------------|
| 1.1 | <pre>import math r = int(input()) c = 2*math.pi*r print(c)</pre> | |
| 1.2 | <pre>r = int(input()) import math c = 2*math.pi*r print(c)</pre> | |
| 1.3 | <pre>r = int(input()) c = 2*math.pi*r import math print(c)</pre> | |

| № | Код | Возможно (+) / невозможно (-) |
|-----|---|-------------------------------|
| 1.4 | <pre>r = int(input()) import math as m c = 2*math.pi*r print(c)</pre> | |
| 1.5 | <pre>r = int(input()) import math as m c = 2*m.pi*r print(c)</pre> | |
| 1.6 | <pre>r = int(input()) import math as pi c = 2*pi.pi*r print(c)</pre> | |

2. Заполните пустые ячейки таблицы, используя знания о функции `range()`:

| Команда | Результат |
|---|--|
| <code>random.randrange(5)</code> | Случайное число от 0, до 5 не включая, с шагом 1 |
| <code>random.randrange(1, 4)</code> | |
| <code>random.randrange(-1, 5, 1)</code> | |
| <code>random.randrange(-2, 8, 2)</code> | |
| <code>random.randrange(8, 0, -3)</code> | |

3. Запишите и вычислите в IDLE математические операции через обращение к библиотеке `math`, используя информацию из сети Интернет:

| Операция | Запись | Результат |
|---|----------------------------|-----------|
| Модуль числа -5 | <code>math.fabs(-5)</code> | 5 |
| Округление 2,404 до ближайшего целого числа сверху | | |
| Округление $-2,404$ до ближайшего целого числа сверху | | |

| Операция | Запись | Результат |
|--|--------|-----------|
| Округление 2,404 до ближайшего целого числа снизу | | |
| Округление -2,404 до ближайшего целого числа снизу | | |
| Остаток от деления 54 на 3 | | |
| Квадратный корень из 4096 | | |

Чтение кода

4. Определите результат работы следующих фрагментов программ, заполняя правый столбец таблицы:

| № | Код | Результат |
|-----|---|-----------|
| 4.1 | <pre>import math a = math.pow(math.pow(2,3),2) print(a)</pre> | |
| 4.2 | <pre>import math a = int(math.pow(math.pow(8,3),0)) print(a*2)</pre> | |
| 4.3 | <pre>import math a = int(math.pow(math.pow(8,3),0)) print(str(a)*2)</pre> | |
| 4.4 | <pre>import math as m a = m.pow(-5,2)//m.pow(2,5) print('<',a,'>\n__^__')</pre> | |

Поиск ошибок

5. Объясните, почему в следующих частях программы Python выведет ошибку, используя варианты ответов:

- A. Переменная не определена.
- B. Интерпретация строки кода невозможна.
- C. Аргумент не строкового типа.
- D. Неизвестная функция.

- Е. Невозможная операция сложения/конкатенации.
 Ф. Невозможная операция сравнения.
 Г. Отсутствие отступа.
 Н. Используемая библиотека не объявлена.

| № | Код | Ошибка |
|-----|--|--------|
| 5.1 | <pre>i = int(input()) while i >= 0: i = i - 1 print(i)</pre> | |
| 5.2 | <pre>for i in range(a): print(a)</pre> | |
| 5.3 | <pre>r = int(input(Разрядность числа:)) b = "" for i in range(r): a = input(Введите ,i, разряд числа:) b += a print(b)</pre> | |
| 5.4 | <pre>a = input("a = ") b = 1000 while a + b > a * b: print("False")</pre> | |
| 5.5 | <pre>a = int(input("Сколько вам лет? ")) if a = 26: print('О! Мне тоже!') else: print('Понятно...')</pre> | |
| 5.6 | <pre>i = 4 while i > 0: if i%2 == 0: "четное" else: "нечетное"</pre> | |
| 5.7 | <pre>for i in range(1,50): print(i)</pre> | |
| 5.8 | <pre>r = int(input()) c = 2*math.pi*r print(c)</pre> | |

Оптимизация

6. Оптимизируйте программу, написанную для игры «Угадай-ка!», таким образом, чтобы пользователь сам задавал число попыток.

7. Написана программа, вычисляющая объем куба, ребро которого вводится с клавиатуры пользователем:

```
import math
a = int(input())
S = math.pow(a, 3)
print(S)
```

Оптимизируйте программу таким образом, чтобы:

- осуществлялся понятный диалог с пользователем;
- пользователь мог получить результат вычисления объема куба с дробным значением ребра;
- выводилось сообщение об ошибке, если пользователь ввел отрицательную длину ребра или 0.

8. (*Задача повышенной сложности*) Ниже приведен промежуточный этап написания программы, вычисляющей дискриминант квадратного уравнения и его корни (при их наличии) по заданным коэффициентам a , b , c .

«Скелет» программы:

- подключение модуля **math**;
- добавление переменных для коэффициентов a , b , c ;
- вычисление дискриминанта D и корней x_1 и x_2 по формулам, приведенным ниже, где условие $D \geq 0$ является необходимым и достаточным для существования корней:

$$D = b^2 - 4ac;$$

$$x_1 = \frac{-b - \sqrt{D}}{2a}; \quad x_2 = \frac{-b + \sqrt{D}}{2a}.$$

```
import math
```

```
a = input('Введите коэффициент a: ')
b = input('Введите коэффициент b: ')
c = input('Введите коэффициент c: ')

```

```
D = b*b-4*a*c
print('D =', D)
```

```
if D >= 0:
    x1 = (-b - math.sqrt(D))/(2*a)
    x2 = (-b + math.sqrt(D))/(2*a)
    print('x1 =', x1, 'x2 =', x2)
else:
    print('Корней нет')
```

Измените программу для нахождения корней квадратного уравнения таким образом, чтобы:

- была исправлена ошибка, не позволяющая программе производить арифметические операции с введенными коэффициентами;
- при $D = 0$ выводился только единственный корень;
- при $D > 0$ второй корень выводился на следующей строке, без повторного использования функции `print()`;
- использовалась функция возведения числа в степень с помощью библиотеки **math**;
- с помощью оператора **as** использовался псевдоним для модуля **math**.

Разработка

9. Используя библиотеку **math**, напишите программу, вычисляющую следующее числовое выражение:

$$\frac{\sqrt{11^3 - 8^3}}{6!} \cdot \pi.$$

10. Используя библиотеку **math**, напишите программу, вычисляющую длину окружности и площадь круга при вводимом с клавиатуры:

10.1) радиусе;

10.2) диаметре.

При написании программы учтите, что вводимое число может быть не целым, но обязательно должно быть положительным (длина отрезка не может быть отрицательной).

11. Напишите программу, генерирующую для преподавателя типовую задачу о нахождении длины окружности со случай-

ным значением радиуса. При этом преподаватель должен иметь возможность самостоятельно задавать диапазон для функции `random.randint(a, b)`.

За основу текста типовой задачи возьмите следующий текст: «Вычислите длину окружности радиусом ___. Значение числа «пи» округлите до сотых».

12. Усовершенствуйте задачу 11 и добавьте проверку ответа к сгенерированной задаче.

Пример:

Введите левую границу диапазона: 2

Введите правую границу диапазона: 10

Задача:

Вычислите длину окружности радиусом 7. Значение числа пи округлите до сотых.

Проверка ответа: 12.5

Результат: Неверно

13. Добавьте в программный код задачи 12 цикл, который предлагает проверять правильность ответа до тех пор, пока результат не окажется верным.

Пример:

Введите левую границу диапазона: 2

Введите правую границу диапазона: 10

Задача:

Вычислите длину окружности радиусом 8. Значение числа пи округлите до сотых.

Проверка ответа: 45

Неверно. Попробуй еще раз

Новый ответ: 87

Неверно. Попробуй еще раз

Новый ответ: 50.24

Верно. Ты нашел правильный ответ!

14. На основе решения задачи 13 создайте программы для генерирования следующих типовых задач с последующей проверкой ответа:

14.1) *«Вычислите периметр квадрата со стороной __».*

14.2) *«Вычислите площадь прямоугольника со сторонами __ и __».*

Глава 6. Исполнитель Черепашка

Помимо модулей, содержащих в себе вспомогательные функции для математических расчетов, существуют те, что не ориентированы на вычисления. Одним из таких является модуль **turtle**, который позволяет рисовать простые или замысловатые графические изображения с помощью объекта Черепашка, который перемещается по полотну пикселей согласно командам, написанным в коде.

Команды перемещения и рисования Черепашки

Приведем список основных команд:

- `forward(L)` — вперед на L единиц;
- `backward(L)` — назад на L единиц;
- `left(D)` — повернуть влево на D градусов;
- `right(D)` — повернуть вправо на D градусов;
- `goto(X, Y)` — переместиться на X по горизонтали и на Y по вертикали (исходная позиция Черепашки — центр холста);
- `home()` — переместиться в центр холста;
- `circle(R)` — нарисовать окружность радиусом R ;
- `color('COLOR')` — использовать цвет $COLOR$ для рисования;
- `bgcolor('COLOR')` — использовать цвет $COLOR$ в качестве фона;
- `begin_fill()` — начало операции заливки фигуры;
- `end_fill()` — конец операции заливки фигуры;
- `color('COLOR_1', 'COLOR_2')` — использовать цвет $COLOR_1$ для пера, цвет $COLOR_2$ для заливки фигуры;
- `down()` — опустить перо. После этой команды Черепашка начнет оставлять след при любом своем передвижении;
- `up()` — поднять перо;
- `width(W)` — установить ширину следа Черепашки в W пикселей;
- `write(T)` — вывести текстовую строку T в точке нахождения Черепашки.

Приведем таблицу основных цветов. (В сети Интернет можно найти намного больше оттенков, для этого достаточно ввести в поисковую систему «список цветов».)

| | | | |
|---------------|----------------|---------------|-----------|
| yellow | желтый | red | красный |
| green | зеленый | orange | оранжевый |
| blue | голубой, синий | pink | розовый |
| brown | коричневый | gray | серый |
| white | белый | black | черный |

Внимание!

Первоначально объект Черепашка направлен вправо.

Также не забываем, что любой модуль подключается с помощью команды **import** и его названия.

Рисование элементарных фигур

Отрезок

Чтобы объект Черепашка начертил отрезок, достаточно воспользоваться функциями `forward(L)` или `backward(L)`. Например, чтобы нарисовать отрезок длиной в 50 единиц (рис. 29) необходим следующий код:

```
import turtle
turtle.forward(50)
```

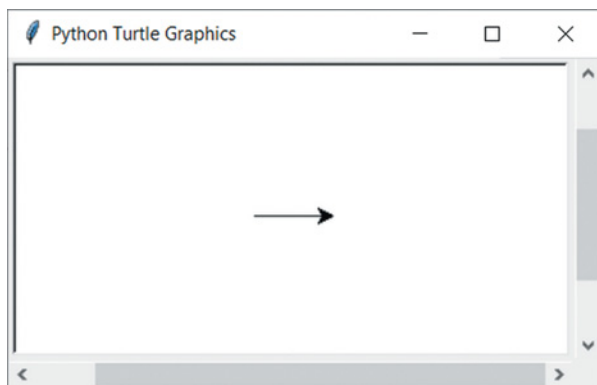


Рис. 29. Отрезок с помощью команды `turtle.forward(50)`

Однако, используя только эти функции, можно получить лишь горизонтальные отрезки заданной длины. Для того чтобы задать наклон отрезку, сначала вводится функция `left(D)` или `right(D)`. Очень важно понимать, как отклоняется от исходного положения курс Черепашки. Нарисуем второй отрезок под 45° (рис. 30) и для сравнения его же под углом 135° (рис. 31):

```
import turtle
turtle.forward(50)
turtle.left(45)
turtle.forward(50)
```

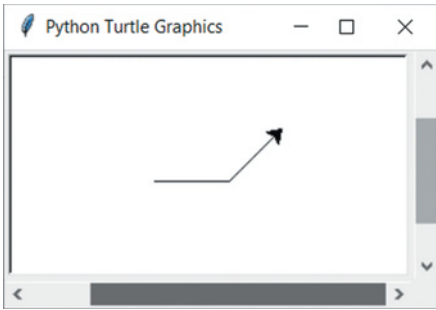


Рис. 30. Поворот влево на 45°

```
import turtle
turtle.forward(50)
turtle.left(135)
turtle.forward(50)
```

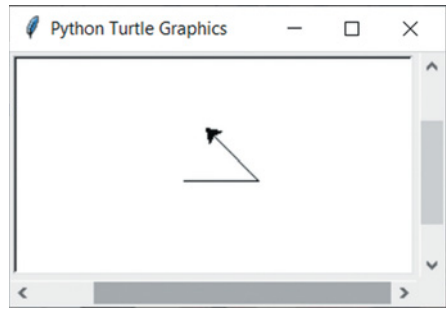


Рис. 31. Поворот влево на 135°

Потренируемся использовать функции из списка команд, рисуя правильные многоугольники.

Внимание!

Далее будем использовать псевдоним **t** для модуля **turtle**.

Равносторонний треугольник

1. Подключаем модуль **turtle** с использованием псевдонима **t**.
2. Рисуем нижнюю сторону треугольника (например, длиной 150 единиц) с помощью команды `t.forward(150)`.
3. Организуем правый угол в 60° : в данный момент Черепашка направлена вправо, поэтому ей необходимо повернуть на угол, равный внешнему углу треугольника, т. е. на 120° . Выполняем это действие с помощью команды `t.left(120)`.
4. Повторяем пункты 2, 3, 2, чтобы нарисовать вторую и третью стороны треугольника.
5. Поднимаем перо с помощью команды `t.up()` и отодвигаем Черепашку на 50 единиц с помощью команды `t.forward(50)`.

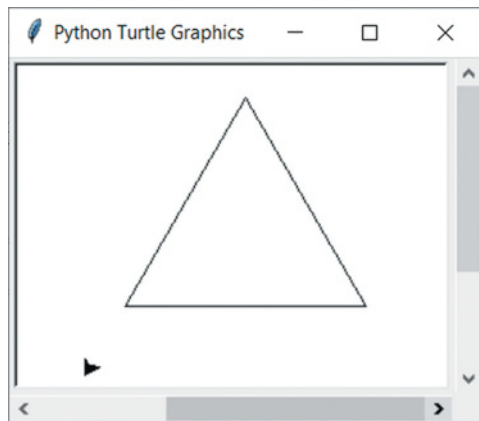


Рис. 32. Равносторонний треугольник со стороной 150

Запишите код самостоятельно и сравните результат выполнения программы с рис. 32.

Квадрат

Нарисуем квадрат со стороной 100 единиц и попробуем добавить другой цвет, например красный (рис. 33). Как и с равносторонним треугольником, будем поочередно рисовать отрезки и менять курс Черепашки на внешний угол стороны фигуры:

```
#Подключаем модуль turtle и используем псевдоним t
import turtle as t
#Подключаем красный цвет для рисования
t.color('red')
#Рисуем нижнюю сторону квадрата
t.forward(100)
#Рисуем правую сторону квадрата
t.left(90)
t.forward(100)
#Рисуем левую сторону квадрата
t.left(90)
t.forward(100)
#Рисуем верх
t.left(90)
t.forward(100)
#Поднимаем перо
t.up()
#Отводим Черепашку на 50 единиц
t.forward(50)
```

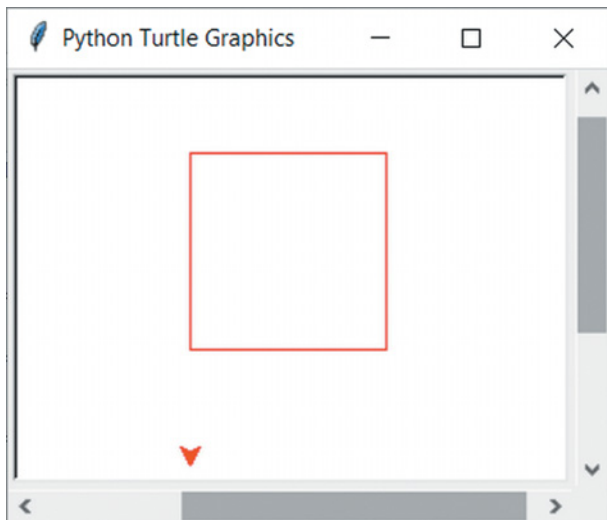



Рис. 33. Красный квадрат со стороной 100

В примере был использован поворот влево, поэтому треугольник и квадрат рисуются снизу вверх, если же использовать поворот вправо, то рисование будет направлено сверху вниз.

Окружность

Для рисования окружности необходима всего лишь одна команда — `circle(R)`. Добавим команду нарисовать окружность радиусом 50 в последний код до поднятия пера и посмотрим, что получится (рис. 34):

```
import turtle as t
t.color('red')
t.forward(100)
t.left(90)
t.forward(100)
t.left(90)
t.forward(100)
t.left(90)
t.forward(100)
t.circle(50)
t.up()
t.forward(50)
```

Обратите внимание, что левая нижняя вершина квадрата не является центром окружности. Необходимо это учитывать при

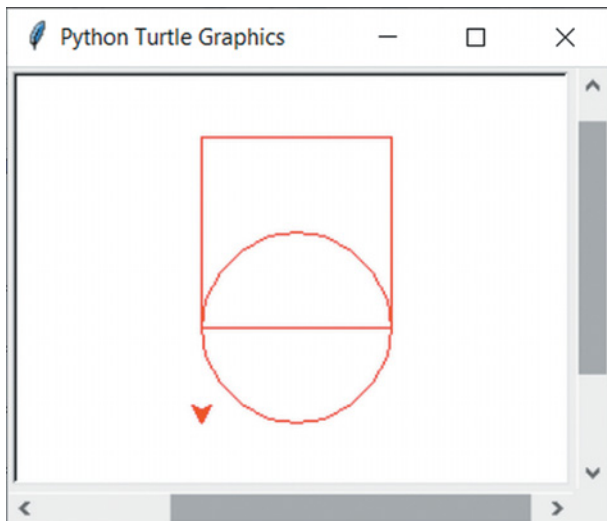


Рис. 34. Рисование окружности из левой нижней вершины квадрата

вписывании фигуры в другие графические объекты. Диаметр окружности равен двум радиусам, поэтому окружность также коснулась правой нижней вершины квадрата.

Работа с цветом

Простые команды, использующие только один цвет, такие как `color('COLOR')` и `bgcolor('COLOR')`, могут работать в программе самостоятельно.

С добавлением цвета при заливке фигуры возникает необходимость в следующей конструкции:

```
begin_fill()
...
color('COLOR_1', 'COLOR_2')
...
end_fill()
```

Здесь

- `begin_fill()` — начало операции заливки фигуры;
- `'COLOR_1'` — выбор цвета пера;
- `'COLOR_2'` — выбор цвета заливки фигуры;
- `end_fill()` — конец операции заливки фигуры.

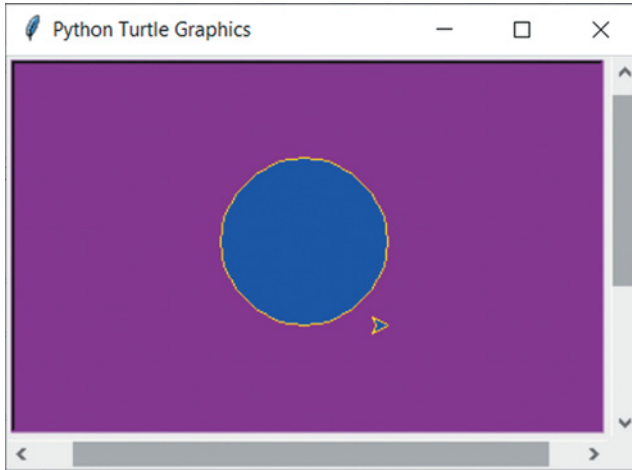


Рис. 35. Желтая окружность радиусом 50 с синей заливкой на фиолетовом фоне

Напишем программу, которая выводит желтую окружность радиусом 50 с синей заливкой на фиолетовом фоне (рис. 35):

```
import turtle as t
#Используем фиолетовый фон
t.bgcolor('purple')
#Рисуем желтую окружностью радиусом 50 с синей заливкой
#Создаем конструкцию begin_fill()...end_fill() для заливки
#фигуры
t.begin_fill()
#Определяем желтый цвет для пера, синий - в качестве
#заливки
t.color('yellow','blue')
#Рисуем окружность радиусом 50
t.circle(50)
#Заканчиваем конструкцию begin_fill()...end_fill()
#для заливки фигуры
t.end_fill()
#Поднимаем перо
t.up()
#Отводим Черепашку на 50 единиц
t.forward(50)
```

Работа с полем

Можно было заметить, что, если не уменьшать размер окна, рисунок появляется по центру (рис. 36).

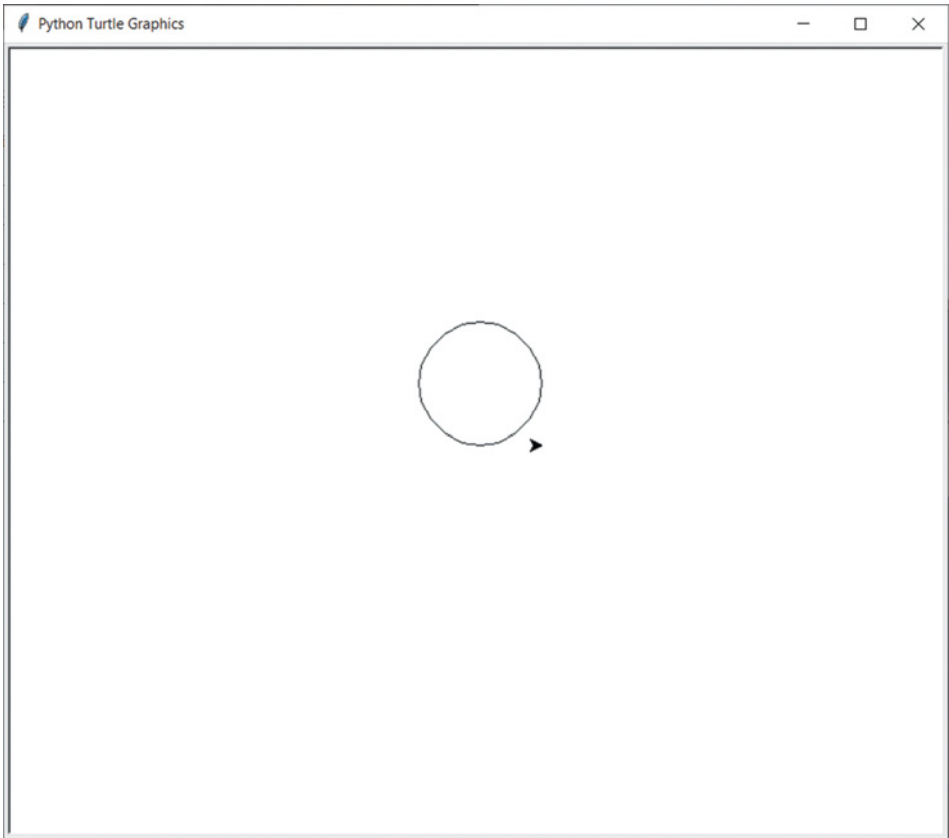


Рис. 36. Развернутое окно с изображением

Однако местоположением рисунка можно управлять с помощью функции `goto(X, Y)`.

Попробуем добавить команду `goto(20,100)`, чтобы изменить местоположение окружности (рис. 37):

```
import turtle as t
t.goto(20,100)
t.circle(50)
t.up()
t.forward(50)
```

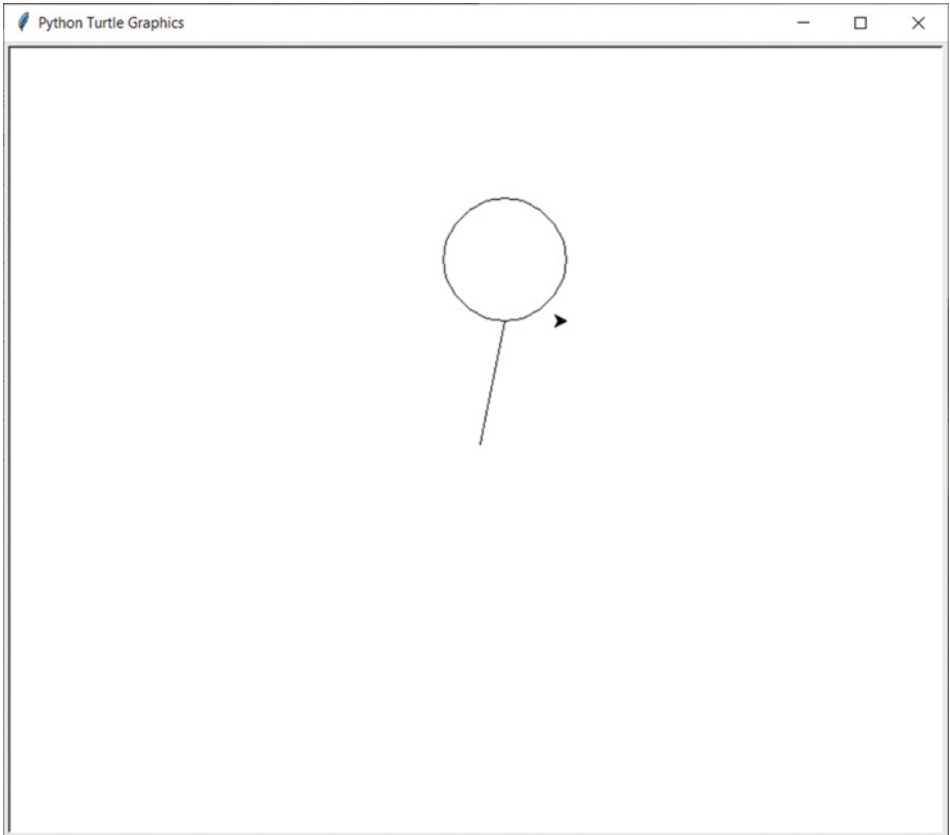


Рис. 37. Использование команды `goto(20,100)`

Во-первых, после запуска кода видим воздушный шарик или леденец, рисовать которые в наши планы не входило, однако теперь мы знаем, как это сделать. Во-вторых, сдвиг произошел от центра на 20 единиц вправо и на 100 единиц вверх.

Поднимем перо перед тем как переместиться и снова опустим перед рисованием окружности (рис. 38):

```
import turtle as t
t.up()
t.goto(20,100)
t.down()
t.circle(50)
t.up()
t.forward(50)
```

Для возвращения в центр достаточно воспользоваться функцией `home()`, хотя вариант `goto(0,0)` также будет работать.

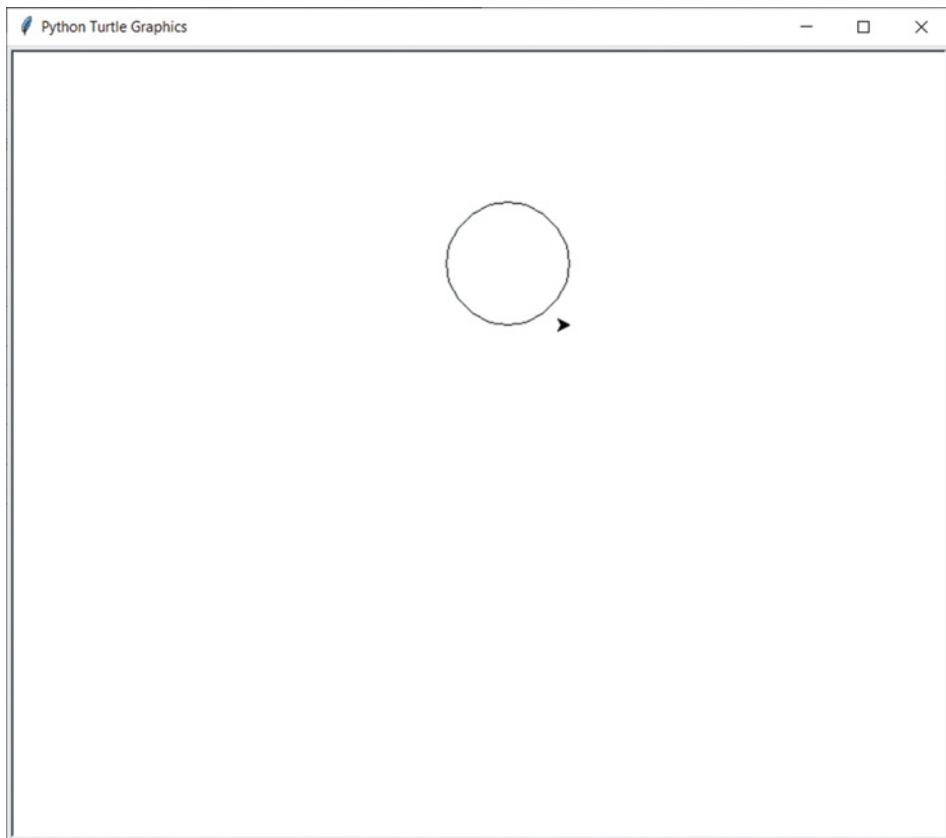


Рис. 38. Перемещение окружности на 20 единиц вправо и на 100 вверх

Реализация условных и циклических конструкций

Рисование правильных многоугольников (в нашем случае равностороннего треугольника и квадрата) может быть реализовано с помощью цикла **for**:

| Треугольник | Квадрат |
|---|--|
| <pre>import turtle as t for i in range(3): t.forward(150) t.left(120)</pre> | <pre>import turtle as t for i in range(4): t.forward(100) t.left(90)</pre> |

Неправильные многоугольники таким образом быстро не рисуешь, однако можно нарисовать правильные пяти-, шести-, десяти-, ..., n -угольники, поскольку меняется только угол между любыми смежными сторонами. Существует даже формула для вычисления угла правильного многоугольника:

$$\frac{(n - 2) \cdot 180^\circ}{n},$$

где n — количество вершин многоугольника.

Напишем программу, позволяющую пользователю самому решать, какой правильный многоугольник нарисовать с помощью Черепашки:

```
import turtle as t
n = int(input('Количество вершин многоугольника: '))
g = (n-2)*180/n
for i in range(n):
    t.forward(100)
    t.left(180-g)
```

Усложним задачу: пусть правильные многоугольники с четным количеством вершин имеют один цвет, а с нечетным — другой. Для этого добавим две условные конструкции:

```
import turtle as t
n = int(input('Количество вершин многоугольника: '))
g = (n-2)*180/n
#Рисование правильного многоугольника с четным
#количеством вершин красным цветом на желтом фоне
if n%2==0:
    t.bgcolor('yellow')
    t.color('red')
#Рисование правильного многоугольника с нечетным
#количеством вершин желтым цветом на красном фоне
else:
    t.bgcolor('red')
    t.color('yellow')
for i in range(n):
    t.forward(100)
    t.left(180-g)
```

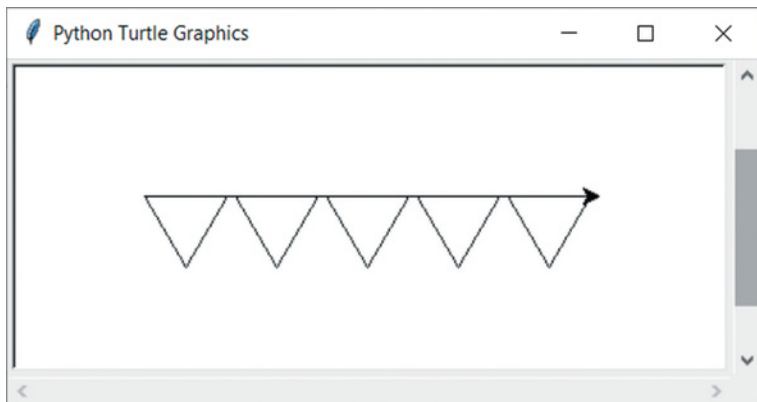


Рис. 39. Гирлянда из треугольников, направленных вниз

С помощью данного кода Черепашка рисует всего одну фигуру. Как насчет того, чтобы фигура повторялась цепочкой в виде гирлянды или напоминала лепестки цветка?

Чтобы реализовать первый вариант, необходимо рисовать фигуру, а после отодвигаться на некоторое расстояние. Нарисуем гирлянду из пяти треугольных флажков (рис. 39):

```
import turtle as t
for j in range (5):
    for i in range(3):
        t.forward(50)
        t.right(120)
    if i == 2:
        t.forward(55)
```

Чтобы треугольник был направлен вниз, использовалась команда `right()`.

После рисования каждой третьей стороны ($i = 2$) был добавлен отрезок в 55 единиц, что позволяет пройти 50 единиц по верхнему основанию треугольника и 5 единиц для соединения со следующим.

Чтобы реализовать второй вариант (несколько одинаковых фигур сходятся в одной точке), необходимо рисовать фигуру, менять наклон Черепашки и повторять эти два действия необходимое количество раз.

Нарисуем цветок из пяти кругов радиуса 50 (рис. 40). Разделим полное вращение направления Черепашки (360°) на количе-

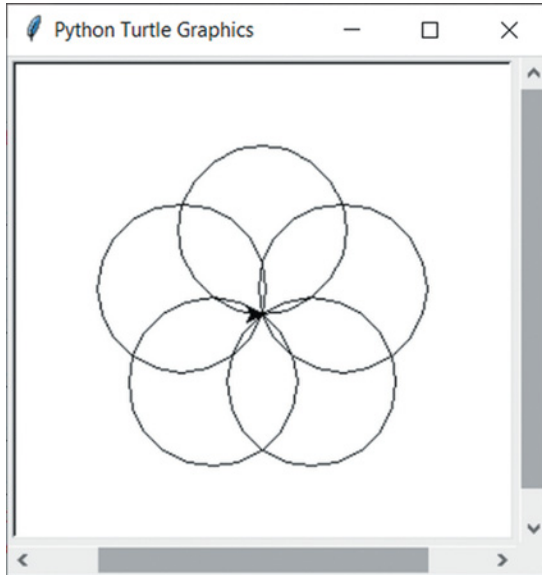


Рис. 40. Цветок из пяти окружностей

ство лепестков (5 окружностей), чтобы получить угол поворота внутри цикла:

```
import turtle as t
for i in range(5):
    t.circle(50)
    t.left(72)
```

В коде есть три аргумента, с которыми можно экспериментировать, меняя их:

- аргумент при функции `range()` показывает количество повторяемых фигур;
- аргумент при функции `circle()` влияет на размер круга;
- аргумент при функции `left()` показывает, на сколько градусов изменяется направление Черепашки перед рисованием следующего круга.

Итак, мы рассмотрели возможности использования еще одного модуля (**turtle**), который позволяет создавать графические изображения с помощью объекта Черепашка, а заодно еще раз попрактиковались в использовании условных и циклических конструкций перед тем, как познакомиться со следующей группой объектов в Python.

Что нового мы узнали в шестой главе?

1. *Turtle* — модуль, который позволяет с помощью программного кода рисовать изображения, основанные на графических примитивах (отрезок, окружность).

2. Исходное положение Черепашки находится в центре холста и направлено вправо.

3. Основные команды объекта Черепашка:

- `forward(L)` — вперед на L единиц;
- `backward(L)` — назад на L единиц;
- `left(D)` — повернуть влево на D градусов;
- `right(D)` — повернуть вправо на D градусов;
- `goto(X, Y)` — переместиться на X по горизонтали и на Y по вертикали (исходная позиция Черепашки — центр холста);
- `home()` — переместиться в центр холста;
- `circle(R)` — нарисовать окружность радиусом R ;
- `color('COLOR')` — использовать цвет $COLOR$ для рисования;
- `bgcolor('COLOR')` — использовать цвет $COLOR$ в качестве фона;
- `begin_fill()` — начало заливки фигуры;
- `end_fill()` — конец заливки фигуры;
- `color('COLOR_1', 'COLOR_2')` — использовать цвет $COLOR_1$ для пера, цвет $COLOR_2$ для заливки фигуры;
- `down()` — опустить перо. После этой команды Черепашка начнет оставлять след при любом своем передвижении;
- `up()` — поднять перо;
- `width(W)` — установить ширину следа Черепашки в W пикселей;
- `write(T)` — вывести текстовую строку T в точке нахождения Черепашки.

Внимание!

Заливка фигуры с помощью команды `color('COLOR_1', 'COLOR_2')` может быть выполнена только в рамках конструкции `begin_fill() ... end_fill()`.

4. С помощью условных и циклических конструкций можно рисовать более сложные конструкции.

Практикум

Основные понятия главы

1. Это интересно! Для движения Черепашки могут быть использованы сокращения. Сопоставьте сокращения их названиям:

| | |
|------|------------|
| fd() | left() |
| bk() | forward() |
| lt() | right() |
| rt() | backward() |

2. Проверьте работоспособность сокращений на практике для любой решенной задачи из главы 6.

Чтение кода

3. Определите, какие геометрические фигуры нарисует Черепашка, выполняя следующие фрагменты программного кода:

| № | Фрагмент кода | Название фигуры |
|-----|---|-----------------|
| 3.1 | <pre>turtle.color('aqua') turtle.circle(100)</pre> | |
| 3.2 | <pre>a = 3 while a != 0: turtle.forward(100) turtle.right(120) a -= 1</pre> | |
| 3.3 | <pre>turtle.goto(200,0) turtle.goto(200,80) turtle.goto(0,80) turtle.goto(0,0)</pre> | |
| 3.4 | <pre>a = 0 b = 2 while a != 2: turtle.color('black') turtle.left(90) turtle.forward(100) a += 1</pre> | |

| № | Фрагмент кода | Название фигуры |
|-----|---|-----------------|
| | <pre>while b != 0: turtle.color('red') turtle.left(90) turtle.forward(100) b -= 1</pre> | |
| 3.5 | <pre>turtle.begin_fill() turtle.color('Grey','Grey') turtle.circle(30.75) turtle.end_fill()</pre> | |

Поиск ошибок

4. Объясните, почему в следующих частях программы Python выведет ошибку, используя варианты ответов:

- A. Переменная не определена.
- B. Отсутствие псевдонима, невозможное обращение к модулю.
- C. Аргумент не строкового типа.
- D. Модуль не подключен.
- E. Введен несуществующий цвет.
- F. Неверный оператор сравнения.
- G. Отсутствие отступа.

| № | Код | Ошибка |
|-----|--|--------|
| 4.1 | <pre>import turtle as t t.up() t.goto(0,-70) t.color(black) t.down() t.forward(180)</pre> | |
| 4.2 | <pre>import turtle as t t.width(3) t.color('rad') t.up() t.goto(-20,-90) t.down() t.forward(161) t.left(90) t.forward(110) t.left(90) t.forward(161) t.left(90) t.forward(110)</pre> | |

| № | Код | Ошибка |
|-----|---|--------|
| 4.3 | <pre>import turtle as t for j in range (5): t.begin_fill() t.color('red','red') for i in range(3): t.forward(50) t.right(120) if r == 2: t.forward(55) t.end_fill()</pre> | |
| 4.4 | <pre>import turtle for j in range (5): if j%2 == 0: turtle.color('green') else: turtle.color('pink') for i in range(4): turtle.forward(50) turtle.right(90) if i == 3: turtle.color('black') turtle.forward(60)</pre> | |
| 4.5 | <pre>for j in range(3): turtle.begin_fill() turtle.color('yellow','blue') for i in range(6): turtle.forward(28) turtle.left(60) turtle.end_fill() turtle.right(120)</pre> | |
| 4.6 | <pre>import turtle t.color('red') t.forward(10) t.left(90) t.forward(10) t.left(90) t.forward(10) t.left(90) t.forward(10) t.up() t.forward(50)</pre> | |

| № | Код | Ошибка |
|-----|---|--------|
| 4.7 | <pre>import turtle as t for j in range (5): if j%2 = 0: t.color('purple') else: t.color('orange') for i in range(4): t.forward(50) t.right(90) if i = 3: t.color('black') t.forward(60)</pre> | |

Оптимизация

5. Дан программный код для рисования оптической иллюзии:

```
import turtle
turtle.forward(50)
turtle.left(90)
turtle.forward(50)
turtle.left(90)
turtle.forward(50)
turtle.left(90)
turtle.forward(50)
turtle.left(90)
turtle.left(30)
turtle.backward(20)
turtle.forward(20)
turtle.left(60)
turtle.forward(50)
turtle.right(60)
turtle.backward(20)
turtle.left(60)
turtle.backward(50)
turtle.right(90)
turtle.forward(50)
turtle.left(30)
turtle.forward(20)
```

- Замените возможную часть кода циклом;
- примените псевдоним;
- предложите пользователю самостоятельно выбрать цвет пера из предложенных (*blue, green, orange, pink*);
- позаботьтесь о том, чтобы пользователь не мог ошибиться в написании цвета.

Разработка

6. Измените программный код для рисования равностороннего треугольника таким образом, чтобы были использованы команды:

6.1) `forward(L)`, `right(L)` и `up()`;

6.2) `backward(L)`, `left(L)` и `up()`;

6.3) `backward(L)` и `right(L)`.

7. Измените программный код для рисования окружности в квадрате таким образом, чтобы:

7.1) окружность была полностью вписана в квадрат;

7.2) окружность заменилась на зеленый круг;

7.3) окружность имела радиус -50 (минус пятьдесят). Как это изменение отразилось на рисунке?

8. Напишите программный код, который с помощью модуля `turtle` выводит слово «имя» (рис. 41).

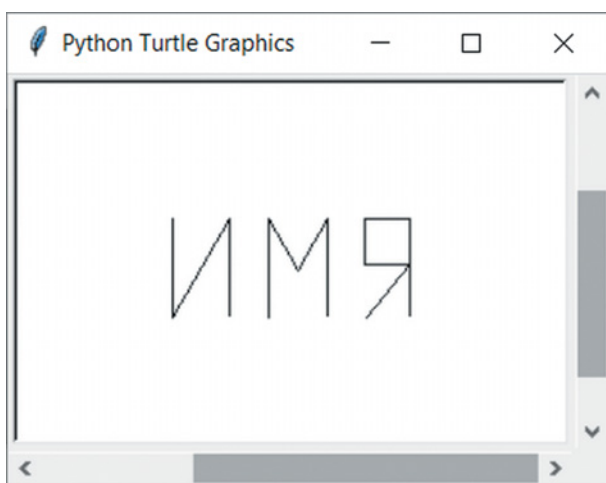


Рис. 41. Программа, выводящая слово «имя»

9. Измените программный код задачи 8: выполните подчеркивание, измените цвет букв и заключите слово «имя» в рамку, имеющую толщину начертания 3 единицы (рис. 42).

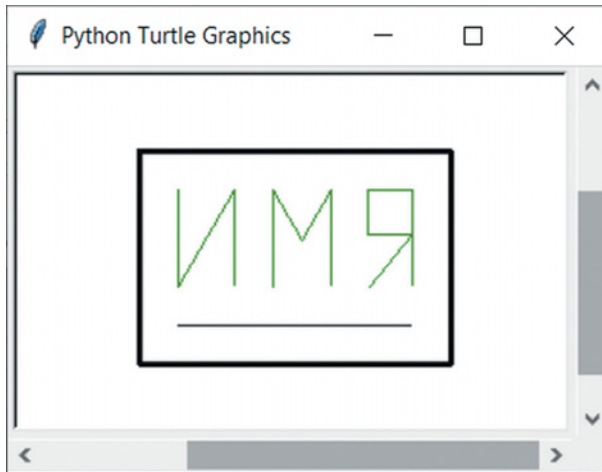


Рис. 42. Программа, выводящая «имя» в рамке

10. Напишите программный код, который с помощью модуля `turtle` выводит изображение игры «Крестики-нолики» (рис. 43).

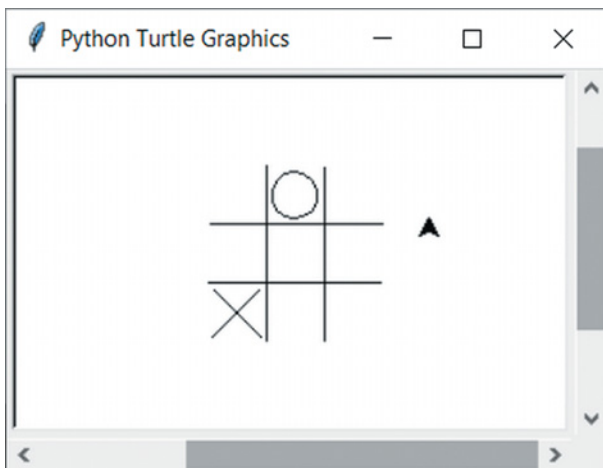


Рис. 43. Вывод «крестики-нолики»

11. Измените код программы для рисования гирлянды с пятью треугольниками, чтобы результат выглядел так, как на рис. 44–46.

11.1)

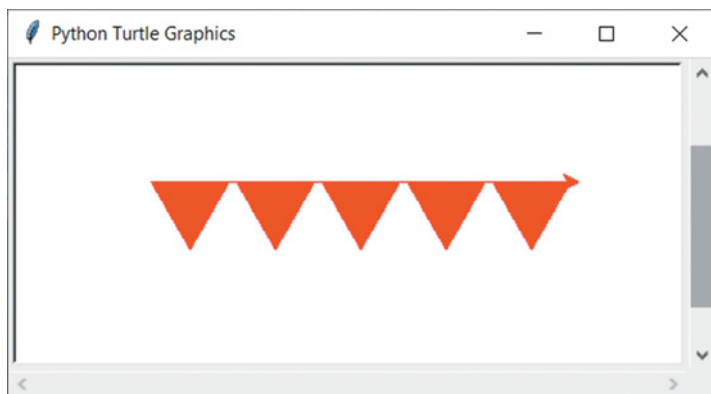


Рис. 44. Вывод красной гирлянды

11.2)



Рис. 45. Вывод цветной гирлянды

11.3)

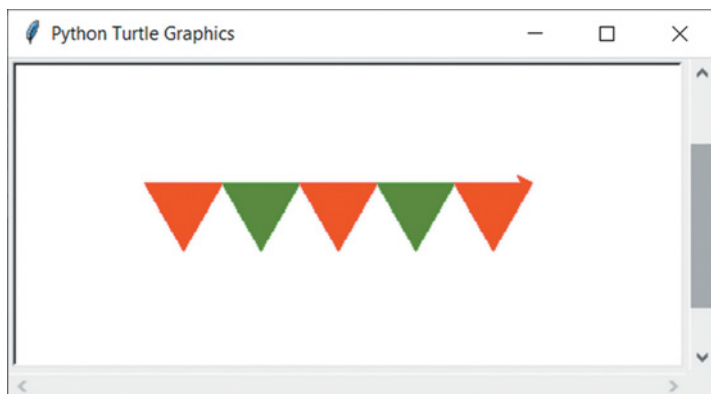


Рис. 46. Вывод цветной гирлянды с заливкой

12. Напишите программу для рисования гирлянды с пятью флажками, чтобы результат выглядел так, как на рис. 47, 48.

12.1)

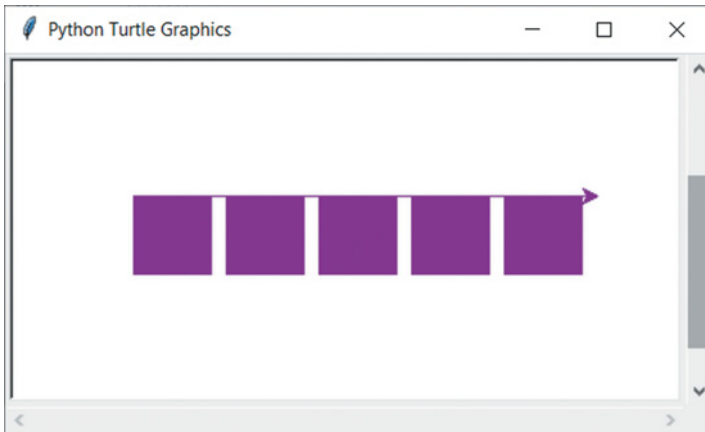


Рис. 47. Фиолетовая гирлянда

12.2) Обратите внимание, что «веревочка гирлянды» черного цвета.

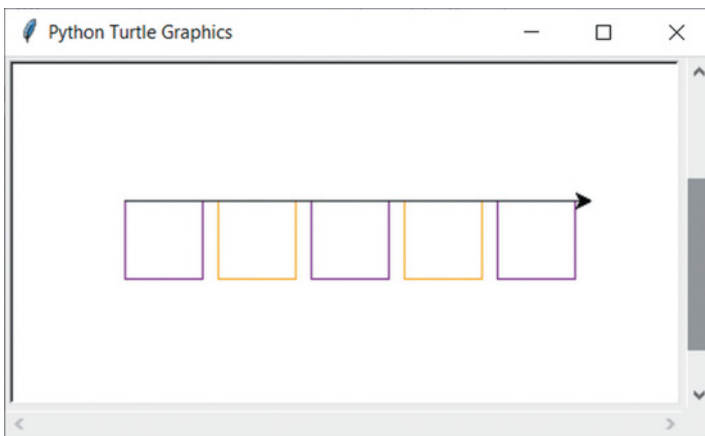


Рис. 48. Гирлянда без заливки

13. Напишите программный код, который с помощью модуля **turtle** выводит следующее изображение (рис. 49). По своему усмотрению добавьте красок, фон, измените ширину пера.

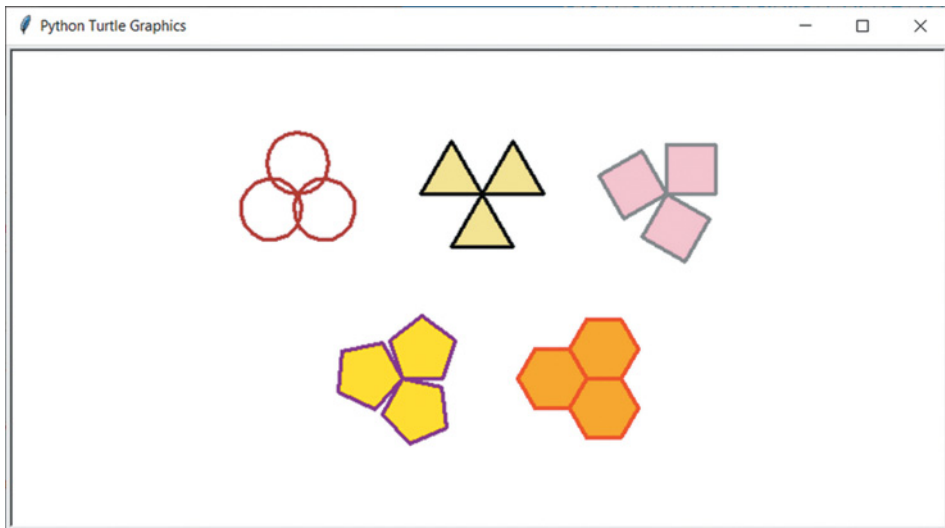


Рис. 49. Дополнительные варианты рисунка

14. После импорта модуля **turtle** в любом своем коде попробуйте написать команду `speed(10)`. Меняя аргумент данной функции, проследите, как меняется поведение Черепашки.

Глава 7. Массивы

В компьютерной науке массив — это пронумерованная последовательность объектов одного типа, обозначаемая одним именем. Python предлагает усовершенствованный аналог массива — *список* (**list**) — пронумерованную последовательность объектов *любой* типов, включая собственный¹.

Списки

Список в Python представляет собой упорядоченный набор значений любого типа, разделенных запятыми и заключенных в квадратные скобки []. Например:

- [1, 0, 4, -3] — список объектов числового типа;
- ['1', 'ab', '#', ' '] — список объектов строкового типа;
- [0, '!', -0.3] — список объектов строкового и числового типов.

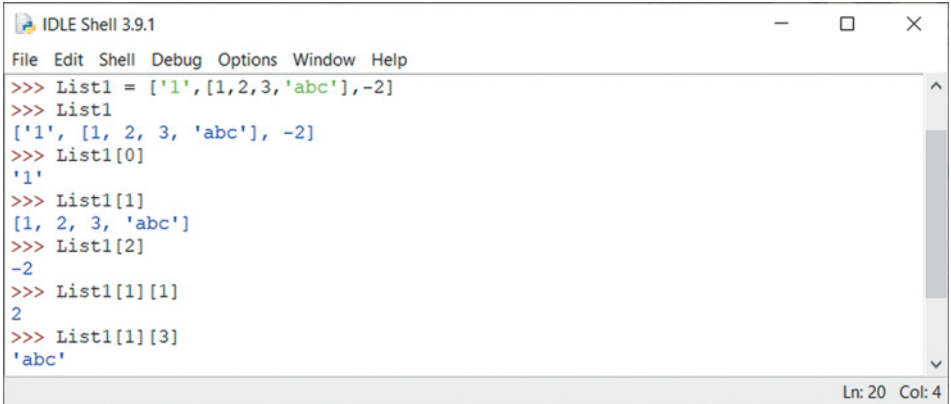
Список также может содержать и свой собственный тип. Даже если это будет список списков, состоящий из списков любого уровня вложенности. Например:

- [[], []] — список пустых списков;
- [[1, 0, 4, -3], ['1', 'ab', '#', ' '], [0, '!', -0.3], [[], []], 'four', [2], 0] — список, содержащий объекты различных типов, включая свой собственный.

Один из способов создания списка в коде — прямое перечисление элементов списка и присвоение его конкретной переменной.

Организуем таким образом (в интерактивном режиме) список с именем `List1`, который будет содержать числовой, строковый и собственный тип данных, и посмотрим, как можно обращаться к любому объекту внутри созданного списка (рис. 50).

¹ Исторически отсутствие ограничения на одинаковый тип данных в Python связан с понятием NLP (Natural Language Processing), т. е. обработкой естественного языка при помощи разбиения потока информации на осмысленные фрагменты.



```

IDLE Shell 3.9.1
File Edit Shell Debug Options Window Help
>>> List1 = ['1',[1,2,3,'abc'],-2]
>>> List1
['1', [1, 2, 3, 'abc'], -2]
>>> List1[0]
'1'
>>> List1[1]
[1, 2, 3, 'abc']
>>> List1[2]
-2
>>> List1[1][1]
2
>>> List1[1][3]
'abc'
Ln: 20 Col: 4

```

Рис. 50. Демонстрация работы со списком в интерактивном режиме

Отметим наши наблюдения:

- создание списка может происходить через перечисление элементов в квадратных скобках через запятую;
- в IDLE вывод списка на экран по-прежнему производится с помощью объявления имени;
- обращение к конкретным элементам списка происходит по указанию в квадратных скобках индекса элемента; именно упорядоченность списка позволяет обращаться к его объектам (элементам) по их порядковому номеру (индексу);

- индексы элементов списка начинаются с 0; так, в списке `List1 = ['1', [1, 2, 3, 'a, b, c'], -2]` имеем

```

List1[0] = '1'
List1[1] = [1, 2, 3, 'a, b, c']
List1[2] = -2

```

- обращение к элементу списка, вложенного в основной список, происходит через указание сначала индекса вложенного списка, а затем индекса элемента в нем; так, в списке `List1 = ['1', [1, 2, 3, 'a, b, c'], -2]` имеем

```

List1[1][0] = 1
List1[1][1] = 2
List1[1][2] = 3
List1[1][3] = 'a, b, c'

```

Для создания списка можно использовать и более автоматизированный подход. Например, функция `list('spam')` создает список из отдельных символов строки `['s', 'p', 'a', 'm']`, а функция `list(range(5))` — список целых чисел `[0, 1, 2,`

3, 4]. Напомним, что в общем виде последняя функция имеет 3 аргумента: `range(a,b,c)`. Например:

```
#Создание списка с помощью функции list()
>>> List2 = list('spam')
>>> List2
['s', 'p', 'a', 'm']
#Создание списка с помощью функций list() и range(a)
>>> List3 = list(range(5))
>>> List3
[0, 1, 2, 3, 4]
#Создание списка с помощью функций list()
#и range(a,b)
>>> List4 = list(range(-2,3))
>>> List4
[-2, -1, 0, 1, 2]
#Создание списка с помощью функций list()
#и range(a,b,c)
>>> List5 = list(range(-20,50,15))
>>> List5
[-20, -5, 10, 25, 40]
```

Операции со списками

Конкатенация и дублирование

Для списков, как и для строк, выполнимы операции конкатенации и дублирования с помощью операторов `+` и `*` соответственно. Например:

```
#Создание списков различными способами
>>> List6 = [A, B, C]
>>> List7 = list('abc')
>>> List8 = list(range(-4,2,2))
#Печать списков для наглядности
>>> List6
[A, B, C]
>>> List7
[a, b, c]
>>> List8
[-4, -2, 0]
#Конкатенация всех списков
>>> List6 + List7 + List8
[A, B, C, a, b, c, -4, -2, 0]
```

```
#Дублирование списка
>>> List6*2
[A, B, C, A, B, C]
#Использование обеих операций одновременно
>>> List7*2 + List8*3
[a, b, c, a, b, c, -4, -2, 0, -4, -2, 0, -4, -2, 0]
```

Добавление и удаление элементов

Любой список может быть увеличен или уменьшен путем добавления или удаления элементов в нем.

Добавить элемент в список можно с помощью функции (метода списка) `list.append()`, где «list» — название списка, к которому хотим добавить элемент, «append()» — функция добавления нового элемента в конец списка («app» + «end»):

```
>>> List9 = [1, 2, 3, 4]
>>> List9.append(5)
>>> List9
[1, 2, 3, 4, 5]
```

Удалить элемент из списка можно несколькими способами:

- написать в начале инструкции `del`, а после указать имя списка и индекс удаляемого элемента из списка в квадратных скобках;
- использовать для удаления метод `pop()`. Если аргумент отсутствует, то автоматически удаляется последний элемент списка, а если аргумент задан, то удаляется элемент указанного в аргументе индекса.

Метод `pop()` отличается от `del` тем, что метод `pop()` удаляет элемент и выводит его на экран, а `del` удаляет и никак об этом пользователю не сообщает.

```
#Удаление элемента с индексом 2 списка List9
>>> del List9[2]
>>> List9
[1, 2, 4, 5]
#Удаление элемента с индексом 0 списка List9
>>> List9.pop(0)
1
>>> List9
[2, 4, 5]
```

```
#Удаление последнего элемента списка List9
>>> List9.pop()
5
>>> List9
[2, 4]
```

После удаления элементов списки обновляются, а следовательно, обновляются и порядковые номера (индексы) «подвинувшихся» элементов.

Очистка списка

Иногда требуется очистка всего списка. Например, если необходимо сбросить содержимое корзины после каждого покупателя. Тогда проще воспользоваться методом `clear()`, который легко запомнить по его названию.

```
>>> List10 = [1, 1, 'uu', 0, [26, 1]]
>>> List10.clear()
>>> List10
[]
```

Итак, мы рассмотрели самые важные операции, которые пригодятся при решении задач: создание списка, добавление и удаление элементов списка и его полная очистка.

Еще несколько методов для списков

- `list.insert(i, x)` — вставляет на место с индексом `i` значение `x`;

```
>>> List10 = [1, 1, 'uu', 0, [26, 1]]
>>> List10.insert(2, [5])
>>> List10
[1, 1, [5], 'uu', 0, [26, 1]]
```

- `list.remove(x)` — удаляет первый элемент в списке со значением `x`. Выводит `ValueError`, если такого элемента не существует.

```
>>> List10 = [1, 1, [5], 'uu', 0, [26, 1]]
>>> List10.remove(1)
>>> List10
[1, [5], 'uu', 0, [26, 1]]
```

С другими методами можно ознакомиться здесь: <https://pythonworld.ru/typy-dannyx-v-python/spiski-list-funkcii-i-metody-spiskov.html>.

Сортировка числовых списков

Огромное количество технических задач приходится на упорядочивание (сортировку) элементов для упрощения визуального поиска, анализа данных и прочей работы.

Например, в справочнике проще найти необходимое слово по алфавитному указателю, в отсортированном списке отметок учеников проще вычислить количество отличников, а в онлайн-магазине найти нужную вещь, отсортировав цены или количество положительных отзывов.

Так или иначе, данная функция пользуется спросом, поэтому важно иметь представление о том, каким образом сортировка происходит на уровне алгоритма. Одним из таких методов является *алгоритм сортировки «пузырьком»*. В его основе лежит цикл, который «поднимает», подобно пузырьку в воде, больший элемент, если сортировка осуществляется по возрастанию, или меньший элемент, если сортировка происходит по убыванию, в конец списка.

Алгоритм сортировки «пузырьком» на Python

Реализуем на Python данный алгоритм, но перед тем как приступить к написанию программы, рассмотрим его работу на примере сортировки по возрастанию небольшой последовательности чисел (5, 4, 3, 2, 1).

Будем последовательно «поднимать» к концу списка больший элемент, поочередно сравнивая два соседних элемента и меняя их местами в случае необходимости.

Тогда согласно методу сортировки «пузырьком»:

- в результате первой итерации цикла число 5 окажется в конце последовательности. Закрепим его и более это число сравнивать с предыдущими не будем, поскольку оно является максимальным среди имеющихся;
- в результате второй итерации «поднимется» число 4 и более не будет рассматриваться и т. д.

Для наглядности в приведенной ниже таблице сравниваемые элементы обозначены красным цветом, а закрепленные подчеркнуты.

Сортировка «пузырьком» последовательности 5, 4, 3, 2, 1:

| | | | | | | |
|---|---|---|---|---|---|------------------------|
| 1-я итерация 4 прохода по всем элементам списка | 5 | 4 | 3 | 2 | 1 | 5 > 4 (меняем местами) |
| | 4 | 5 | 3 | 2 | 1 | 5 > 3 (меняем местами) |
| | 4 | 3 | 5 | 2 | 1 | 5 > 2 (меняем местами) |
| | 4 | 3 | 2 | 5 | 1 | 5 > 1 (меняем местами) |
| | 4 | 3 | 2 | 1 | 5 | 5 — закреплено |
| 2-я итерация 3 прохода по всем элементам списка | 4 | 3 | 2 | 1 | 5 | 4 > 3 (меняем местами) |
| | 3 | 4 | 2 | 1 | 5 | 4 > 2 (меняем местами) |
| | 3 | 2 | 4 | 1 | 5 | 4 > 1 (меняем местами) |
| | 3 | 2 | 1 | 4 | 5 | 4 — закреплено |
| 3-я итерация 2 прохода по всем элементам списка | 3 | 2 | 1 | 4 | 5 | 3 > 2 (меняем местами) |
| | 2 | 3 | 1 | 4 | 5 | 3 > 1 (меняем местами) |
| | 2 | 1 | 3 | 4 | 5 | 3 — закреплено |
| 4-я итерация 1 проход по всем элементам списка | 2 | 1 | 3 | 4 | 5 | 2 > 1 (меняем местами) |
| | 1 | 2 | 3 | 4 | 5 | 2 — закреплено |

Теперь приступим к написанию программы, упорядочивающей элементы списка с помощью цикла **for**:

Создаем основу: первоначальный список + сортирующий цикл + печать отсортированного списка.

#Объявляем список из 5 элементов

```
my_list = [5, 4, 3, 2, 1]
```

#Объявляем цикл, который проходит по всем элементам #списка, сравнивая соседние элементы и меняя их при #необходимости

```
for i in range(4):
    if my_list[i] > my_list[i+1]:
        #Создаем дополнительную переменную a, чтобы
        #временно записать значение большего элемента
        a = my_list[i]
        #Записываем значение следующего элемента на
        #место предыдущего
        my_list[i] = my_list[i+1]
        #Записываем в следующий элемент временно
        #помещенное в a значение
        my_list[i+1] = a
print(my_list)
```

Именно такой способ перестановки двух элементов массива обычно используют в других языках программирования, однако Python предлагает очень простую форму записи:

```
list[i],list[i+1] = list[i+1],list[i]
```

В результате получаем короткий и удобочитаемый вид программного кода:

```
my_list = [5, 4, 3, 2, 1]
for i in range(4):
    if my_list[i] > my_list[i+1]:
        my_list[i],my_list[i+1] = my_list[i+1],my_list[i]
print(my_list)
```

Обращаем внимание, что в цикле мы уменьшаем размер списка на единицу (`range(4)`), поскольку обращение к последнему элементу находится внутри цикла (`my_list[i+1]`). Иначе говоря, если создать список индексов `range(5)`, то последнее сравнение между числами будет между `my_list[4]` и `my_list[5]`, что невозможно (`my_list[5]` не существует).

Теперь необходимо запускать этот механизм до тех пор, пока все элементы не выстроятся по возрастанию. Для этого добавляем внешний цикл, который отвечает за количество проходов всего алгоритма для смещения наибольшего значения в конец списка.

```
my_list = [5, 4, 3, 2, 1]
for j in range(4):
    for i in range(4):
        if my_list[i] > my_list[i+1]:
            my_list[i],my_list[i+1] =
                my_list[i+1],my_list[i]
print(my_list)
```

Алгоритм сортировки «пузырьком» для последовательности из 5 чисел готов!

Метод `split()` и функция `len()`

Расширим функционал программы, добавив возможность введения любых пяти чисел с клавиатуры.

Для этого будем использовать новый метод `split()`, который позволяет создавать список из последовательности символов, разделенных пробелом.

```
l = input('Введите 5 чисел через пробел: ')
#Пример: 4 7 1 3 0
my_list = l.split()
#my_list = ['4', '7', '1', '3', '0']
for j in range(4):
    for i in range(4):
        if my_list[i] > my_list[i+1]:
            my_list[i],my_list[i+1]=
                my_list[i+1],my_list[i]
print(my_list)
```

Проверяем код. Ура! Все работает!

В общем случае аргументом метода `split()` может быть любой символ в роли разделителя. Например:

```
>>> s = '1,2,34,567'
>>> List_s = s.split(',')
>>> List_s
['1', '2', '34', '567']
```

В ходе проверки программы не забываем задавать себе вопросы: «Что будет, если ввести в список не только однозначные числа?», «Что будет, если ввести отрицательные числа?», «Что будет, если ввести дроби?». Исследуйте возможные варианты, чтобы определить соответствие написанной программы поставленной задаче и предугадать возможные ошибки.

Введем для проверки числа 1, 11, 2, 33, 44 и в результате получим список `['1', '11', '2', '33', '44']`, хотя ожидалось увидеть список `['1', '2', '11', '33', '44']`. Что не так?

С программой «все так», однако пока в списке элементы только строкового типа, Python сортирует их по алфавиту, где сравнение происходит посимвольно. Этот механизм работал с цифрами, но не с числами больше 9.

Теперь задача состоит в том, чтобы добавить в код инструкцию, которая преобразует строковый тип данных в числовой для всех элементов образовавшегося списка.

Для этого «навесим» на сравниваемые элементы функцию `int()`:

```
l = input('Введите 5 чисел через пробел: ')
my_list = l.split()
for j in range(4):
    for i in range(4):
        if int(my_list[i]) > int(my_list[i+1]):
            my_list[i],my_list[i+1]=
            my_list[i+1],my_list[i]
print(my_list)
```

Обратите внимание, что данная версия программы преобразует строковый тип данных в числовой только на этапе сравнения (`int(my_list[i]) > int(my_list[i+1])`). Дальнейшая работа производится по-прежнему с элементами строкового типа (`my_list[i],my_list[i+1] = my_list[i+1],my_list[i]`). Поэтому в результате выполнения данного кода на экране появится список со строковыми элементами.

Если необходима дальнейшая работа с числами, то рекомендуется добавить цикл, преобразующий все элементы списка в числовой формат по всей его длине. Функция `int()` подойдет только для работы с целыми числами, а `float()` добавит возможности работать с дробными.

Например:

```
l = input('Введите 5 чисел через пробел: ')
my_list = l.split()
#Для каждого индекса от 0 до 5 производим операцию
#преобразования в целое (дробное) число
for i in range(5):
    my_list[i] = int(my_list[i])
    #my_list[i] = float(my_list[i])
for j in range(4):
    for i in range(4):
        if my_list[i] > my_list[i+1]:
            my_list[i],my_list[i+1]=
            my_list[i+1],my_list[i]
print(my_list)
```

Для снятия ограничения на количество сортируемых чисел воспользуемся функцией `len()`, вычисляющей длину списка, аргументом которой выступит список `my_list`:

```
l = input('Введите числа через пробел: ')
my_list = l.split()
for i in range(len(my_list)):
    my_list[i] = int(my_list[i])
    #my_list[i] = float(my_list[i])
for j in range(len(my_list)-1):
    for i in range(len(my_list)-1):
        if my_list[i] > my_list[i+1]:
            my_list[i],my_list[i+1]=
            my_list[i+1],my_list[i]
print(my_list)
```

Возможен и первоначальный вариант с выводом элементов строкового типа, если не требуются дальнейшие действия над числами:

```
l = input('Введите числа через пробел: ')
my_list = l.split()
for j in range(len(my_list)-1):
    for i in range(len(my_list)-1):
        if int(my_list[i]) > int(my_list[i+1]):
            my_list[i],my_list[i+1]=
            my_list[i+1],my_list[i]
print(my_list)
```

Метод sort()

Реализовать алгоритм сортировки «пузырьком» было замечательной идеей, но еще более замечательной идеей будет узнать о том, что в языке Python уже существует метод `sort()`, который самостоятельно отсортирует последовательность в порядке возрастания ее элементов.

Приведем несколько примеров с различными входными данными и посмотрим на результат выполнения данной функции:

```
l = input('Введите сортируемые элементы через пробел: ')
#Пример 1: 1 2.4 -5.2 -3.8
#Пример 2: Tom Bella Hugo Anna
#Пример 3: 12.5 Jane 7 Jake -8
my_list = l.split()
my_list.sort()
```

```
print(my_list)
#Результат 1: ['-3.8', '-5.2', '1', '2.4']
#Результат 2:['Anna', 'Bella', 'Hugo', 'Tom']
#Результат 3:['-8', '12.5', '7', 'Jake', 'Jane']
```

Для сортировки чисел по-прежнему потребуется цикл преобразования в числовой формат:

```
l = input('Введите сортируемые элементы через пробел: ')
#1 2.4 -5.2 -3.8
my_list = l.split()
for i in range(len(my_list)):
    my_list[i] = float(my_list[i])
my_list.sort()
print(my_list)
#[-5.2, -3.8, 1.0, 2.4]
```

Если необходимо список отсортировать в порядке убывания или в порядке, обратном алфавитному, то аргументу метода `sort()` дописывается уточнение «`reverse = True`»:

```
sort(reverse = True)
```

По умолчанию аргументу `reverse` (обратное направление) присваивается значение `False`.

Продемонстрируем работу сортировки списка в порядке убывания или порядке, обратном алфавитному, на тех же примерах:

```
l = input('Введите сортируемые элементы через пробел: ')
#Пример 1: 1 2.4 -5.2 -3.8
#Пример 2: Tom Bella Hugo Anna
#Пример 3: 12.5 Jane 7 Jake -8
my_list = l.split()
my_list.sort(reverse = True)
print(my_list)
#Результат 1:['2.4', '1', '-5.2', '-3.8']
#Результат 2:['Tom', 'Hugo', 'Bella', 'Anna']
#Результат 3:['Jane', 'Jake', '7', '12.5', '-8']
```

Для сортировки чисел:

```
l = input('Введите сортируемые элементы через пробел: ')
#1 2.4 -5.2 -3.8
my_list = l.split()
```

```
for i in range(len(my_list)):
    my_list[i] = float(my_list[i])
my_list.sort(reverse = True)
print(my_list)
#[2.4, 1.0, -3.8, -5.2]
```

Однако если список изначально содержит значения числового и строкового типов данных, то сортировка оказывается невозможной. Операция сравнения возможна либо между строками, либо между числами:

```
my_list = [12.5, 'Jane', 7, 'Jake', -8]
my_list.sort()
print(my_list)
```

```
#TypeError: '<' not supported between instances of 'str'
and 'float'
```

Поэтому одним из вариантов решения такой задачи будет создание двух различных списков для строк и чисел с их последующей сортировкой и конкатенацией:

```
ln = input('Введите сортируемые числа через пробел: ')
#1 2.4 -5.2
ls = input('Введите сортируемые строки через пробел: ')
#Bella Hugo Anna
```

```
#Создание и сортировка списка чисел
my_list_num = ln.split()
for i in range(len(my_list_num)):
    my_list_num[i] = float(my_list_num[i])
my_list_num.sort()
#Создание и сортировка списка строк
my_list_str = ls.split()
my_list_str.sort()
#Печать склеенных списков
print(my_list_num + my_list_str)
#[-5.2, 1.0, 2.4, 'Anna', 'Bella', 'Hugo']
print(my_list_str + my_list_num)
#[ 'Anna', 'Bella', 'Hugo', -5.2, 1.0, 2.4]
```

При этом числа можно сортировать по убыванию, а строки — в алфавитном порядке или наоборот.

Усложним задачу. Зададим список, содержащий числа, строки и числовые списки, и отсортируем его по частям. Для этого воспользуемся функцией `type()`, которая определяет тип данных элементов, чтобы разбить первоначальный список на три категории:

```
l = [1, 3, -10, [], 'a', 'abc', 2, 9.0, [1, -2], 'opr', [0, 5, 6]]
#Создание дополнительных пустых списков для чисел, строк
#и числовых списков
l_num = []
l_str = []
l_list = []
#Распределение элементов по трем категориям
for i in range(len(l)):
    if type(l[i]) == int or type(l[i]) == float:
        l_num.append(l[i])
    elif type(l[i]) == str:
        l_str.append(l[i])
    elif type(l[i]) == list:
        l_list.append(l[i])
#Сортировка трех вспомогательных списков
l_num.sort()
l_str.sort()
l_list.sort()
#Печать в необходимом порядке
print(l_num + l_str + l_list)
#[-10, 1, 2, 3, 9.0, 'a', 'abc', 'opr', [], [0,5,6], [1,-2]]
```

Внимание!

При сортировке списка `l_list` сравниваются первые элементы этих списков, а не количество элементов в них.

Подсчет количества элементов списка. Метод `count()`

При работе с большими объемами данных требуется не только их сортировка для удобочитаемости или быстрого поиска, но и подсчет количества конкретных наименований. Поэтому в арсенале

библиотек Python также существует метод, подсчитывающий количество конкретных элементов в списке, — `count()`.

Например, дадим Python команду посчитать, сколько раз последовательность цифр «12» встречается в числе 12612712:

```
l = input('Введите число: ')
#Пример: 12612712
count = l.count('12')
print(count) #3
```

Можно также узнать, сколько «пик» встречается в предложении «Купи лучше кипу пик, лучше пик кипу купи!»:

```
l = input('Введите предложение, в котором необходимо
посчитать количество "пик": ')
#Например: Купи лучше кипу пик, лучше пик кипу купи!
count = l.count('пик')
print(count) #2
```

Предлагаем попробовать самостоятельно составить программу подсчета количества элементов в списке с помощью циклов и условных конструкций аналогично методу сортировки «пузырьком». Также много задач на эту тему можно найти в разделе «Практикум».

Кортежи

Стоит отметить, что список является аналогом *динамического (изменяемого) массива* в программировании, однако также существует понятие так называемого *статического (неизменяемого)* массива. Его аналогом на Python является *кортеж*. Он необходим в тех случаях, когда важно защитить какие-либо данные, всевозможные пароли, когда используются неизменяемые данные (дни недели, месяцы и т. д.) или когда нужно использовать элементы кортежа в качестве ключа в словарях (об этом позже).

Итак, *кортеж (tuple)* представляет собой неизменяемый упорядоченный набор значений, разделенных запятыми и заключенных не в квадратные (как в списке), а в круглые скобки (). Например:

- (1, 0, 4, -3) — кортеж объектов числового типа;
- ('1', 'ab', '#', ' ') — кортеж объектов строкового типа;

- $(0, '!', -0.3)$ — кортеж объектов строкового и числового типов;
- $([], [])$ — кортеж пустых списков.

Кортеж, как и список, может содержать и свой собственный тип, даже если это будет кортеж кортежей любого уровня вложенности. Например:

- $((), ())$ — кортеж пустых кортежей;
- $((1, 0, 4, -3), ('1', 'ab', '#', ' '), [0, '!', -0.3], ([], []), 'four', [2], 0)$ — кортеж, содержащий объекты различных типов, включая свой собственный.

Операции с кортежами

Основным различием между списками и кортежами является то, что объекты кортежей не могут быть изменены, а также удалены или добавлены. Другими словами, кортежи можно рассматривать как списки, доступные только для чтения. Поэтому они имеют преимущество в виде меньшего объема занимаемой памяти, а также большей скорости прохода по элементам.

Несмотря на невозможность изменять кортежи, по-прежнему можно обращаться к их элементам по индексам, выполнять операцию умножения кортежа на число или складывать кортежи между собой.

```
#Создание пустого кортежа
>>> tuple1 = tuple()
>>> tuple1
()

#Создание кортежа объявлением элементов в круглых
#скобках
>>> tuple2 = (1,2,[3,5])

#Вывод элемента кортежа по индексу
>>> tuple2[2]
[3,5]

#Операции с элементами кортежа
>>> tuple2[0]+ tuple2[1]
3

#Умножение кортежа на число (не изменение, а создание
#нового)
>>> tuple2*3
(1,2,[3,5],1,2,[3,5],1,2,[3,5])
```

```
#Конкатенация кортежей (не изменение, а создание
#нового)
>>> ('gone', 'new', 'mom') + tuple2
('gone', 'new', 'mom', 1, 2, [3, 5])
```

Внимание!

При операциях конкатенации или дублирования кортежа создается новый кортеж, а не меняется исходный.

Чтобы посмотреть на вариант применения кортежей в реальной жизни, напишем часть программы, которая помогает создать пароль к личному кабинету на произвольном сайте.

Начнем с основы: создадим переменную, отвечающую за место хранения пароля + вопрос о создании пароля + условную конструкцию, в которой за ответом «Да» последует запись переменной с паролем, вводимым с клавиатуры.

```
#Создание пустого кортежа с именем password
password = tuple()
#Вопрос к пользователю о желании создать пароль
q = input('Задать пароль? Да - 1, Нет - 0: ')
#Вариантов в условной конструкции три: 1, 0
#и неверный ввод команды
if q == '1':
#Запись нового пароля с клавиатуры в новую переменную
    new_password = input('Введите пароль: ')
#Запись пароля с помощью кортежа в переменной
#password
    password = tuple(new_password)
elif q == '0':
    ...
else:
    print('Команда введена неверно')
```

Сразу можно проверить, получится ли создать пароль одной строчкой и сократить код:

```
password = tuple()
q = input('Задать пароль? Да - 1, Нет - 0: ')
if q == '1':
```

```
#Запись пароля с клавиатуры в новую переменную
password = tuple(input('Введите пароль: '))
elif q == '0':
    ...
else:
    print('Команда введена неверно')
```

Допустим, при отрицательном ответе на вопрос о задании пароля программа будет предлагать показать текущий. Поэтому добавим еще одну условную конструкцию:

```
password = tuple()
q = input('Задать пароль? Да - 1, Нет - 0: ')
if q == '1':
    password = tuple(input('Введите пароль: '))
elif q == '0':
    n = input('Показать текущий пароль? Да - 1,
    Нет - 0: ')
    if n == '1':
        print(password)
    elif n == '0':
        print('Спасибо за внимание!')
    else:
        print('Команда введена неверно')
else:
    print('Команда введена неверно')
```

Эту же конструкцию можно добавить к положительному ответу на вопрос о смене пароля на случай, если пользователь решит перепроверить введенный им новый пароль:

```
password = tuple()
q = input('Задать пароль? Да - 1, Нет - 0: ')
if q == '1':
    password = tuple(input('Введите пароль: '))
    n = input('Показать текущий пароль? Да - 1,
    Нет - 0: ')
    if n == '1':
        print(password)
    elif n == '0':
        print('Спасибо за внимание!')
    else:
        print('Команда введена неверно')
```

```
elif q == '0':
    n = input('Показать текущий пароль? Да - 1,
    Нет - 0: ')
    if n == '1':
        print(password)
    elif n == '0':
        print('Спасибо за внимание!')
    else:
        print('Команда введена неверно')
else:
    print('Команда введена неверно')
```

При этом пароль может иметь ограничения на количество символов или обязательно включать в себя определенные символы и знаки препинания. В разделе «Практикум» мы предлагаем самостоятельно составить программу, учитывающую подобные ограничения.

Срезы списков, строк и кортежей

К слову о логинах и паролях: при восстановлении пароля нередко выводят последние 4 цифры номера телефона, на который высылается одноразовый проверочный код, а все остальные цифры скрываются звездочками. Реализовать на Python подобную ситуацию с частичным выводом некой последовательности поможет **срез**.

Вывод четырех последних цифр номера телефона может выглядеть следующим образом:

```
numb = '89161112530'
print('*'*7, numb[-4:])
#***** 2530
```

Попробуйте предположить, что значит '-4:' в квадратных скобках.

Итак, **срез для списков, строк и кортежей** позволяет использовать только некоторую их часть и выводится при помощи обращения [с... : до... не включая : с шагом...], как и для функции range(). При этом:

- если первый аргумент не прописан конкретным числом, то Python автоматически принимает его за начало списка;

- если второй аргумент не прописан конкретным числом, то Python считает его равным последнему индексу плюс 1;
- если третий аргумент не прописан конкретным числом, то Python автоматически принимает его за 1.

Рассмотрим возможные варианты срезов на примере списка

```
List5 = [-50, 0, 50, 100, 150, 200, 250, 300, 350, 400, 450]:
```

| Обращение к диапазону списка List5 | Обращаемые индексы | Результат |
|------------------------------------|---------------------|----------------------|
| List5[:3] | с 0 по 2 | [-50, 0, 50] |
| List5[2:6] | с 2 по 5 | [50, 100, 150, 200] |
| List5[7:] | с 7 по 10 | [300, 350, 400, 450] |
| List5[:8:2] | с 0 по 7 с шагом 2 | [-50, 50, 150, 250] |
| List5[::3] | с 0 по 10 с шагом 3 | [-50, 100, 250, 400] |

В примере о последних четырех цифрах номера телефона использован отрицательный индекс. Отрицательные индексы удобно использовать в случае большого количества элементов в списке. Например, если нужно обратиться к последнему элементу, достаточно обратиться к нему по индексу [-1], к предпоследнему — [-2] и т. д.:

| Обращение к элементу или диапазону списка List5 | Обращаемые индексы | Результат |
|---|-----------------------|-------------------------------|
| List5[-5] | -5 | 250 |
| List5[:-6] | с 0 по -7 | [-50, 0, 50, 100, 150] |
| List5[2:-3] | с 2 по -4 | [50, 100, 150, 200, 250, 300] |
| List5[:-5:-1] | с -1 по -4 с шагом -1 | [450, 400, 350, 300] |

Обратите внимание, если задан отрицательный шаг, то движение по индексам списка производится с конца. Следовательно, за первый рассматриваемый индекс принимается -1.

Строка, как неизменяемая последовательность символов, также может быть выведена некоторым диапазоном. Рассмотрим также несколько диапазонов для строки `str = 'new_target'`, включая положительные и отрицательные индексы:

| Обращение к диапазону строки <code>str</code> | Обращаемые индексы | Результат |
|---|-----------------------|-----------|
| <code>str[:3]</code> | с 0 по 2 | 'new' |
| <code>str[2:5]</code> | с 2 по 4 | 'w_t' |
| <code>str[4:]</code> | с 4 по 9 | 'target' |
| <code>str[:6:2]</code> | с 0 по 5 с шагом 2 | 'nwt' |
| <code>str[::3]</code> | с 0 по 9 с шагом 3 | 'n_rt' |
| <code>str[-5]</code> | -5 | 'a' |
| <code>str[:-6]</code> | с 0 по -7 | 'new_' |
| <code>str[2:-3]</code> | с 2 по -4 | 'w_tar' |
| <code>str[:-5:-1]</code> | с -1 по -4 с шагом -1 | 'tegr' |

Когда может пригодиться задание шага в срезе? Рассмотрим, например, следующую задачу: вывести на экран каждый 5-й код на упаковке хлопьев из 20 представленных, поскольку он является призовым.

Для решения этой задачи сначала воспользуемся генератором случайных чисел и создадим 20 кодов, а после выведем их на экран вместе с призовыми:

```
#Подключаем модуль random
import random
#Для создания кодов используем большой диапазон, чтобы
#свести к минимуму вероятность повторяющихся кодов
for i in range(21):
    code = random.randint(1000,10000)
    print('code ', i, ': ', code)
```

На данный момент коды создаются и выводятся, но работать с ними мы пока не можем. Поэтому в начале программы создадим пустой список, а с каждым проходом цикла `for` будем добавлять в него сгенерированные значения. Также упакуем все

получившиеся коды в кортеж, чтобы нельзя было изменять положение номеров и добавлять новые.

```
import random
#Создание списка для возможности работы с кодами
list = []
for i in range(21):
    code = random.randint(1000,10000)
    add_code = list.append(code)
    print('code ', i, ': ', code)
#Упаковка полученного списка в неизменяемый (кортеж)
all_codes = tuple(list)
```

Остается только вывести на экран призовые номера:

```
import random
list_code = []
for i in range(21):
    code = random.randint(1000,10000)
    add_code = list_code.append(code)
    print('code ', i, ': ', code)
all_codes = tuple(list_code)
#Вывод на экран каждого пятого кода (призового)
#через срез
print('Призовые коды: ', all_codes[::5])
```

Вариант результата данного программного кода:

```
code 0 : 6926
code 1 : 6162
code 2 : 3097
code 3 : 1437
code 4 : 4677
code 5 : 9310
code 6 : 5029
code 7 : 4539
code 8 : 6090
code 9 : 2645
code 10 : 8023
code 11 : 1177
code 12 : 3839
code 13 : 6093
code 14 : 7503
```

```
code 15 : 2123
code 16 : 6608
code 17 : 3162
code 18 : 9621
code 19 : 5357
code 20 : 6923
```

Призовые коды: (4677, 2645, 7503, 5357)

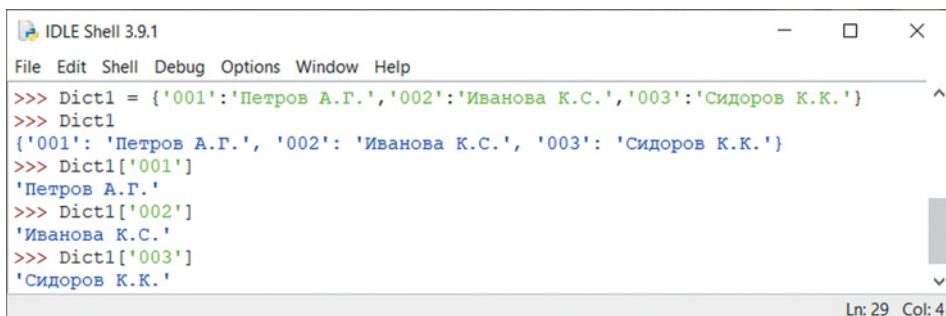
Словари

Когда встает вопрос об уникальных кодах, порядок которых не важен, часто используют еще один тип данных — *словарь*. Прекрасными примерами использования данных этого типа являются:

- база данных для продуктов магазина, где уникальному штрих-коду продукта соответствует его название;
- детская игра «Шифровка», в которой участники придумывают свой секретный язык и каждой букве алфавита дают уникальное обозначение.

В перспективе это может использоваться в более серьезных задачах программирования — шифровании (защите данных) и архивировании данных больших корпораций.

Словарь (**dict**) похож на список, элементы которого берутся не по индексу, а по указанному для него ключу (рис. 51). Чтобы отличить словарь от списка или кортежа, достаточно обратить внимание на фигурные скобки { }, в которых элементы перечислены по шаблону «*Ключ : Значение*». Например: {'001' : 'Петров А.Г.', '002' : 'Иванова К.С.', '003' : 'Сидоров К.К.'}.



```
IDLE Shell 3.9.1
File Edit Shell Debug Options Window Help
>>> Dict1 = {'001':'Петров А.Г.', '002':'Иванова К.С.', '003':'Сидоров К.К.'}
>>> Dict1
{'001': 'Петров А.Г.', '002': 'Иванова К.С.', '003': 'Сидоров К.К.'}
>>> Dict1['001']
'Петров А.Г.'
>>> Dict1['002']
'Иванова К.С.'
>>> Dict1['003']
'Сидоров К.К.'
```

Ln: 29 Col: 4

Рис. 51. Демонстрация создания словаря и извлечения из него элементов

Использование словаря также часто объясняют возможностью самостоятельно задавать индексы (в списке индексы нумеруются автоматически). В таком случае удобно прописывать уникальные номера в зависимости от контекста. Например, номера рейсов, авторов, названия команд для связок «номер рейса : направление», «автор : книга», «название команды : участник команды».

Важно!

Ключ обязан быть уникальным и иметь неизменяемый тип данных, значение может повторяться и иметь любой тип данных. Другими словами:

- числа могут быть ключами и значениями;
- строки могут быть ключами и значениями;
- кортежи могут быть ключами и значениями;
- логические переменные могут быть ключами и значениями;
- *списки не могут быть ключами*, но могут быть значениями.

Подобно функциям `list()` и `tuple()`, позволяющим создавать соответствующие их названиям списки и кортежи, у словаря есть функция `dict()`. Эта функция дает возможность создать словарь несколькими способами. Например:

```
>>> A = dict(d1='one', d2=2, d3=['t','h','r','e','e'])
>>> A
{'d1': 'one', 'd2': 2, 'd3': ['t', 'h', 'r', 'e', 'e']}

>>> B = dict([(1, 'one'), (2, 2), (3, ['t', 'h', 'r', 'e', 'e'])])
>>> B
{1: 'one', 2: 2, 3: ['t', 'h', 'r', 'e', 'e']}
```

- В первом случае ключ всегда должен быть строкового типа и пары словаря задаются перечислением. Общий вид такого способа создания словаря

```
dict(Ключ_1 = Значение_1, Ключ_2 = Значение_2, ...)
```

- во втором случае аргументом функции является список кортежей, что позволяет создавать сам список любым удобным образом и оставаться при этом изменяемым до использования функции `dict()`. Общий вид такого способа создания словаря

```
dict([(Ключ_1, Значение_1), (Ключ_2, Значение_2), ...])
```

или

```
List = [(Ключ_1, Значение_1), (Ключ_2, Значение_2), ...]  
dict(List)
```

Также существует метод **fromkeys**, который позволяет создать словарь, состоящий из уникальных ключей с пустыми или одинаковыми значениями:

```
dict.fromkeys([Ключ_1, Ключ_2, ...])
```

или

```
dict.fromkeys([Ключ_1, Ключ_2, ...], Общее значение)
```

Например:

```
>>> A = dict.fromkeys(['d1', 'd2', 'd3'])
```

```
>>> A
```

```
{'d1': None, 'd2': None, 'd3': None}
```

```
>>> B = dict.fromkeys(['b1', 'b2', 'b3'], 50)
```

```
>>> B
```

```
{'b1': 50, 'b2': 50, 'b3': 50}
```

Операции со словарями

Чтобы дополнить словарь новой парой «*Ключ : Значение*», необходимо присвоить значение новому индексу (даже если это будет выражение):

```
>>> A = {'d1': 'one', 'd2': 2, 'd3': ['t', 'h', 'r', 'e', 'e']}
```

```
>>> A['d4'] = list(range(4))
```

```
>>> A
```

```
{'d1': 'one', 'd2': 2, 'd3': ['t', 'h', 'r', 'e', 'e'],  
'd4': [0, 1, 2, 3]}
```

Присваиванием можно поменять и значение любого ключа:

```
>>> A = {'d1': 'one', 'd2': 2, 'd3': ['t', 'h', 'r', 'e', 'e']}
>>> A['d3'] = 'three'
>>> A
{'d1': 'one', 'd2': 2, 'd3': 'three'}
```

При этом если есть желание или необходимость оставить это значение, то создают дополнительный ключ, присваивают это значение ему, а после обновляют значение предыдущего ключа (такой обмен похож на использованный нами при сортировке «пузырьком»):

```
>>> A = {'d1': 'one', 'd2': 2, 'd3': ['t', 'h', 'r', 'e', 'e']}
#Создание еще одного ключа 'd3_new' для переноса
#значения из ключа 'd3'
>>> A['d3_new'] = A['d3']
#Обновление значения для ключа 'd3'
>>> A['d3'] = 'three'
>>> A
{'d1': 'one', 'd2': 2, 'd3': 'three', 'd3_new': ['t',
'h', 'r', 'e', 'e']}
```

Приведем еще несколько методов словаря:

- `dict.clear()` — очищает словарь;
- `dict.popitem()` — удаляет и возвращает пару «Ключ : Значение». Если словарь пуст, выводит на экран сообщение об ошибке;
- `dict.items()` — возвращает пары «Ключ : Значение»;
- `dict.keys()` — возвращает ключи в словаре;
- `dict.values()` — возвращает значения в словаре.

С полным списком методов `dict` можно ознакомиться здесь: <https://pythonworld.ru/typy-dannyx-v-python/slovari-dict-funkcii-i-metody-slovarej.html>.

Воспользуемся одним из способов создания словаря, чтобы организовать базу данных для книжного магазина, и напишем код, который будет предлагать работнику магазина самостоятельно добавить 3 позиции.

Для начала создадим пустой словарь, зададим цикл для пяти элементов с добавлением пар в пустой словарь и выведем его на экран.

Обратите внимание, что ключи являются уникальными элементами словаря, поэтому запрос на артикул записывается в первую переменную, а название — во вторую.

```
bd_shop = {}
for i in range(3):
    a = input('Введите артикул: ')
    b = input('Введите название: ')
    print('Позиция добавлена')
    bd_shop[a] = b
print(bd_shop)
```

С какими трудностями можно столкнуться? Что, если один артикул введен дважды: работник ошибся, в списке опечатка или действительно нужна замена значения артикула на другую позицию? Для начала запустим код и посмотрим на примере, что Python выведет на экран:

```
Введите штрих-код позиции: 12345
Введите название позиции: Блокнот А5
Позиция добавлена
Введите штрих-код позиции: 12345
Введите название позиции: Блокнот А4
Позиция добавлена
Введите штрих-код позиции: 23456
Введите название позиции: Блокнот А3
Позиция добавлена
{'12345': 'Блокнот А4', '23456': 'Блокнот А3'}
```

Результат показывает, что произошла замена значения для индекса «12345» и на текущий момент словарь содержит не три пары «*Ключ : Значение*», а две. Исправим это!¹

Добавим проверку на существующие ключи словаря. В случае, если артикул совпадает с имеющимся, предложим ввести позицию заново или действительно заменить название позиции.

```
bd_shop = {}
i = 0
while i < 3:
    a = input('Введите артикул: ')
    b = input('Введите название: ')
```

¹ В Python нельзя повлиять на счетчик внутри for.

```
#Если a (артикул) содержится в базе данных магазина
if a in bd_shop:
#Убеждаемся в замене наименования позиции или
#предлагаем ввести артикул и продукт заново
    print('Произвести замену позиции ',a,' - ',
          bd_shop[a],'?')
    c = int(input('Да - 1/Нет - 0: '))
    if c == 1:
#Если все верно, то записываем данную позицию заново
        bd_shop[a] = b
#Выводим отчет для удобства пользователя и проверки
        print('Замена произведена')
    elif c == 0:
#При отрицательном ответе предлагаем ввести артикул
#заново
        c1 = input('Введите артикул заново: ')
        c2 = input('Введите название заново: ')
        bd_shop[c1] = c2
#Если повторных ключей не было, продолжаем добавлять
#позиции:
    else:
        bd_shop[a] = b
        i += 1
        print('Позиция добавлена')
print(bd_shop)
```

Какую инструкцию необходимо заменить в первоначальном шаге построения программы и что добавить, чтобы цикл стал бесконечным, пока пользователь сам не прервет его работу? Попробуйте реализовать эту задачу в разделе «Практикум».

Поиск по словарю

Поиск значений словаря осуществляется через обращение по ключу, как было показано выше. Однако стоит отметить, что при поиске значения по несуществующему ключу программа выдает ошибку `KeyError` и прекратит работу.

Ошибка автоматически исключается в условных конструкциях (заранее подразумеваются варианты `True` и `False`), в другом же случае для избежания вывода ошибки при поиске чаще всего используют метод `dict.get()`. Тогда при отсутствии ключа Python

выведет на экран сообщение «None» или сообщение, указанное программистом в аргументе после ключа:

```
>>> A = {'d1':'one', 'd2':2, 'd3':['t', 'h', 'r', 'e', 'e']}
>>> A.get('d4')
None
>>> A.get('d4', 'Stop out!')
Stop out!
```

Сортировка словаря по ключам

Навести порядок в выводимом на экране списке можно в алфавитном или обратном алфавитному порядке. Для этого производится несколько дополнительных манипуляций, поскольку в словаре нет упорядоченности ключей, как у списка индексов.

В этом случае задача заключается в том, чтобы создать список из ключей словаря, далее отсортировать его (можно с помощью готового метода `sort()`) и затем выводить на экран значения для каждого ключа из получившегося уже упорядоченного списка.

Для примера возьмем список с блоками и неупорядоченными ключами:

```
A = {'234': 'Блокнот А4', '915': 'Блокнот
А3', '37': 'Блокнот А2', '7373': 'Блокнот А1'}
```

```
#Создание списка из ключей
list_keys = list(A.keys())
#['234', '915', '37', '7373']
#Сортировка списка ключей в алфавитном порядке
list_keys.sort()
#['234', '37', '7373', '915']
#Теперь у каждого ключа есть свой индекс
for i in list_keys:
    print(i, ':', A[i])
#A[0] = A['234'] = 'Блокнот А4' и т.д.
```

Результат:

```
234 : Блокнот А4
37 : Блокнот А2
7373 : Блокнот А1
915 : Блокнот А3
```


В результате получаем вывод элементов словаря в алфавитном порядке ключей. Вспомните, что необходимо добавить в аргументе функции `sort()`, чтобы вывод ключей был в порядке, обратном алфавитному.

Что нового мы узнали в седьмой главе?

1. **Массив** — это пронумерованная последовательность объектов одинакового типа, обозначаемая одним именем. Массивы бывают динамические (изменяемые) и статические (неизменяемые).

2. **Список (*list*)** — это пронумерованная последовательность объектов *любых* типов, включая собственный:

- список представляет собой набор значений любого типа, разделенных запятыми и заключенных в квадратные скобки [];
- список часто создают простым перечислением его элементов, с помощью функций `list('строка')` или `list(range())`;
- вывод списка на экран производится с помощью функции `print()`;
- вывод конкретных элементов списка производится по указанию в квадратных скобках индекса элемента;
- индексы элементов списка начинаются с 0;
- `list.append()` — метод добавления элемента в список с названием `list`;
- `del` и `pop()` — методы удаления элемента из списка. Методы `pop()` и `del` различаются тем, что первый удаляет элемент и выводит его на экран, а второй удаляет и никак об этом пользователю не сообщает;
- `list.clear()` — метод удаления всех элементов списка;
- `list.split()` — метод, позволяющий создать список из последовательности символов, разделенных пробелом. В общем случае аргументом `split()` может быть любой символ в роли разделителя (в этом случае он указывается в одинарных кавычках как аргумент функции);
- `len(list)` — функция, вычисляющая длину списка;

- `list.sort()` — метод, сортирующий список в алфавитном порядке (по возрастанию). Для сортировки в порядке, обратном алфавитному (по убыванию), используется `sort(reverse = True)`;
- `list.count()` — метод, возвращающий количество конкретных элементов в списке;
- ознакомиться с полным списком методов `list` можно здесь: <https://pythonworld.ru/typy-dannyx-v-python/spiski-list-funkcii-i-metody-spiskov.html>.

3. Способ перестановки двух элементов списка на языке Python:

```
list[i], list[i+1] = list[i+1], list[i]
```

4. *Кортеж* — статический массив или список, доступный только для чтения, который состоит из набора значений, разделенных запятыми, заключенными не в квадратные, а в круглые скобки ():

- объекты кортежей не могут быть изменены, а также удалены или добавлены;
- обращение к элементам кортежа по индексам, операция умножения кортежа на число и сложение кортежей между собой по-прежнему остаются возможными;
- при операциях конкатенации или дублирования кортежа создается новый кортеж, а не меняется исходный.

5. *Словарь* — неупорядоченный набор элементов, которые имеют вид «*Ключ* : *Значение*» и заключены в фигурные скобки { }:

- словарь похож на список, к элементам которого обращаются не по индексам, а по заданным ключам;
- ключ обязан быть уникальным и иметь неизменяемый тип данных, значение может повторяться и иметь любой тип данных:
 - числа могут быть ключами и значениями;
 - строки могут быть ключами и значениями;
 - кортежи могут быть ключами и значениями;
 - логические переменные могут быть ключами и значениями;
 - списки не могут быть ключами, но могут быть значениями;
- `dict()` — функция, которая позволяет создать словарь несколькими способами:

- о словарь, состоящий из заданных ключей и их значений (ключ обязательно строкового типа):

```
dict(Ключ_1 = Значение_1, Ключ_2 = Значение_2, ...)
```

- о словарь, состоящий из заданных ключей и их значений (ключ не может быть только списком):

```
dict([(Ключ_1, Значение_1), (Ключ_2, Значение_2), ...])
```

- о словарь, состоящий из ключей с пустыми значениями:

```
dict.fromkeys([Ключ_1, Ключ_2, ...])
```

- о словарь, состоящий из ключей с одинаковыми заданными значениями:

```
dict.fromkeys([Ключ_1, Ключ_2, ...], Общее значение)
```

- `dict.clear()` — очищает словарь;
- `dict.popitem()` — удаляет и возвращает пару «*Ключ* : *Значение*» (если словарь пуст, выводит ошибку на экран);
- `dict.items()` — возвращает пары «*Ключ* : *Значение*»;
- `dict.keys()` — возвращает ключи в словаре;
- `dict.values()` — возвращает значения в словаре;
- ознакомиться с полным списком методов `dict` можно здесь: <https://pythonworld.ru/typy-dannyx-v-python/slovari-dict-funkcii-i-metody-slovarej.html>;
- добавление новой пары «*Ключ* : *Значение*» в словарь происходит с помощью присвоения значения новому индексу;
- присваиванием можно изменить значение любого ключа;
- при поиске значения по несуществующему ключу программа выдаст ошибку (`KeyError`) и прекратит работу. Ошибка автоматически исключается в условных конструкциях (заранее подразумеваются варианты `True` и `False`), в другом же случае при поиске чаще всего используют метод `dict.get()`. В этом случае при отсутствии ключа Python выведет на экран сообщений «`None`» или сообщение, указанное программистом в аргументе после ключа;
- сортировка словаря по ключам подразумевает сортировку дополнительно созданного списка его ключей без потери связей.

6. Функция `type()` определяет тип своего аргумента.

Практикум

Основные понятия главы

1. Определите, к какому типу данных относится каждый объект:

| | int | float | str | list | tuple | dict |
|---------------------|-----|-------|-----|------|-------|------|
| -5 | + | - | - | - | - | - |
| [7, -9, 'good'] | | | | | | |
| 46.0 | | | | | | |
| (17, 0, [1,2,3]) | | | | | | |
| -1000000000000000 | | | | | | |
| '2+5' | | | | | | |
| {17: 0, 0: [1,2,3]} | | | | | | |

2. Определите правильность инструкции для создания словаря:

| Пример создания словаря | +/- |
|---|-----|
| <code>D = dict(1 = 'total', 2 = 'money')</code> | |
| <code>D = dict(1, 'total', 2, 'money')</code> | |
| <code>D = dict(a1 = 'total', a2 = 'money')</code> | |
| <code>D = dict(a1 = 'total'; a2 = 'money')</code> | |
| <code>D = dict([(1, 'total'), (2, 'money')])</code> | |
| <code>D = dict([(a1, 'total'), (a2, 'money')])</code> | |
| <code>D = dict([(a1; 'total'), (a2; 'money')])</code> | |
| <code>D = dict([('a1', 'total'), ('a2', 'money')])</code> | |

3. Дан список `L = [0,1,1,1,1,1]`. С помощью обращения к элементам по индексам измените список таким образом, чтобы каждый следующий элемент являлся суммой двух предыдущих, начиная с третьего по индексу.

4. Дан словарь `A = {'int': '_', 'float': '_', 'str': '_', 'list': '_', 'tuple': '_', 'dict': '_'}`. Замените значения элементов словаря на примеры шести типов данных **int**, **float**, **str**, **list**, **tuple**, **dict** с помощью обращения к каждому значению по его ключу.

Чтение кода

5. Какой результат выдаст программа, содержащая следующий код?

```
A = [1,0,15]
B = [26,8,-1]
A[0] = B
print(A)
```

Можно ли изменить элементы *B*? Почему?

6. Какой результат выдаст программа, содержащая следующий код?

```
A = ['10',4,3]
B = [-4,34,'6']
C = {3:A,2:B,1:[A,'8',9]}
A[0] = 0
print(C[3])
```

Можно ли изменить элементы *C*? Почему?

7. Какой результат выдаст программа, содержащая следующий код?

```
A = [1,2,3]
B = [4,5,6]
C = {1:A,2:B,3:[7,8,9]}
D = (A,B,C)
print(D)
```

Можно ли изменить элементы *D*? Почему?

8. Какой результат выдаст программа, содержащая следующий код?

```
A = list('Twitter')
A[3] = 's'
del A[-2:]
print(A)
```

9. Какой результат выдаст программа, содержащая следующий код?

```
A = list(range(10))
del A[0]
A[2] = list('years')
del A[-6:]
print(A)
```

10. Как будет выглядеть список, если известны его части, приведенные ниже?

```
MyList[0] = 1
MyList[1][1] = 0
MyList[1:3] = [['abc', 0, '2'], 36]
MyList[3] = ('hello', ',', 'world', '!')
MyList[-1][-1] = 500
MyList[4] = [100, 500]
```

Поиск ошибок

11. Объясните, почему в следующих частях программы Python выведет ошибку, используя варианты ответов:

- А. Переменная не определена.
- В. Обращение к несуществующему индексу.
- С. Библиотека или модуль не подключен.
- Д. Аргумент не строкового типа.
- Е. Невозможная операция сложения/конкатенации.
- Ф. Обращение к несуществующему ключу.
- Г. Отсутствие отступа.
- Н. Неверный оператор сравнения.

| № | Код | Ошибка |
|------|---|--------|
| 11.1 | <pre>i = random.randint(1,5) if i%2 == 0: print(i,'- четное') else: print(i,'- нечетное')</pre> | |
| 11.2 | <pre>A = int(input('Введите предстоящий год: ')) print('С Новым ' + A + ' годом!')</pre> | |

| № | Код | Ошибка |
|------|--|--------|
| 11.3 | <pre>import random codes = [] while i > 0: code = random.randint(1,20) codes.append(code) i -= 1</pre> | |
| 11.4 | <pre>D = {} D['0'] = '01011' D['1'] = '01111' D['2'] = '01000' D['3'] = '00110' D['4'] = '11011' D['5'] = '11001' print(D[3])</pre> | |
| 11.5 | <pre>List = [] i = 3 while i >= 0: List.append([]) List[i][0] = 0 i = i - 1</pre> | |
| 11.6 | <pre>password = input(Введите пароль) list_password = password.split() tuple_password = tuple(list_password) print("Пароль введен в виде строки:", password) print("Далее преобразован в список:", list_password) print("А также в кортеж:", tuple_password)</pre> | |
| 11.7 | <pre>n = input('Введите числа через пробел ') list_n = n.split() count = 0 for i in range(len(list_n)) if float(list_n[i+1]) < 0: count += 1 print('Количество отрицательных:', count)</pre> | |

| № | Код | Ошибка |
|------|--|--------|
| 11.8 | <pre>n = int(input('Введите положительное целое число: ')) list_n = list(range(-50,n,5)) i = 0 while list_n[i] < n and list_n[i] > -50: if i = 5 or i = 6 or i = 7: continue print(list_n[i]) i += 1</pre> | |

Оптимизация

12. Преобразуйте код программы, задающей пароль, таким образом, чтобы:

12.1) пароль состоял не менее чем из a символов;

12.2) пароль состоял из a символов и содержал хотя бы один символ из следующих: ! @ \$. (Переменная a может быть заменена программистом внутри кода на необходимое количество. Для проверки используйте $a = 5$.)

13. Преобразуйте код программы, которая выводит на экран каждый 5-й код на упаковке хлопьев из 20 представленных таким образом, чтобы происходила проверка на уникальность каждого нового кода (если сгенерированный код совпадает с каким-либо из предыдущих, обнулять и генерировать новое значение). Для проверки используйте диапазон чисел для генерации кода от 1 до 20 (никакие два кода не должны совпадать).

14. Преобразуйте код программы о пополнении базы данных книжного магазина таким образом, чтобы работник мог пополнять базу данных магазина до тех пор, пока самостоятельно не прервет работу программы.

15. Дан код программы, которая записывает 12 фамилий участников некоторой группы (введенных с клавиатуры), сортирует их по алфавиту, распределяет на три равные подгруппы и выводит результат распределения на экран:

```
names = input('Введите фамилии 12 участников через
запятую:\n')
```



```
list_names = names.split(',')
list_names.sort()
print('Подгруппа 1',list_names[:4])
print('Подгруппа 2',list_names[4:8])
print('Подгруппа 3',list_names[8:12])
```

Преобразуйте предложенный код таким образом, чтобы:

- общее количество участников (n) определялось на основании количества введенных фамилий;
- выводился отсортированный список участников с порядковой нумерацией;
- количество участников в подгруппе (k) определялось пользователем;
- если введено отрицательное значение k , то программа выводит сообщение об ошибке и предлагает ввести значение заново до тех пор, пока не будет введено положительное число больше 1;
- если введенное число участников не кратно числу k , то оставшиеся участники попадают в подгруппу «Резерв».

Разработка

16. Составьте программу без использования функции `len()` для подсчета количества:

16.1) вводимых с клавиатуры чисел через пробел;

16.2) неположительных чисел среди вводимых с клавиатуры через запятую;

16.3) положительных чисел, кратных числу 3, среди вводимых с клавиатуры через запятую.

17. Создайте словарь любым известным вам способом и выведите пары словаря, отсортированные в порядке убывания ключей.

18. Напишите код программы, которая распределяет целые числа, введенные пользователем, на два списка:

18.1) четные и нечетные числа;

18.2) положительные и отрицательные числа;

18.3) числа, кратные 5, и все остальные;

18.4) ненулевые числа, делящиеся на 3, но не делящиеся на 9, и числа, делящиеся на 3 и 9 одновременно. Числа,

не вошедшие ни в одну категорию, добавляются в дополнительный список.

19. Напишите код программы, которая формирует список учеников класса по шаблону «Фамилия И.О.», сортирует в алфавитном порядке и выводит его на экран.

Пример:

Добавить нового ученика?(+/-) : +
Введите Фамилию И.О.: Марьяновский И.И.

Добавить нового ученика?(+/-) : +
Введите Фамилию И.О.: Иванченко П.Р.

Добавить нового ученика?(+/-) : -

Список учащихся:
1 – Иванченко П.Р.
2 – Марьяновский И.И.

20. Напишите код программы, которая определяет отличников и хорошистов по введенным оценкам для неограниченного количества классов и выводит их в отсортированном порядке.

Пример:

Класс: 11А
Чтобы закончить ввод в поле 'Фамилия И.О.:' укажите '-'.
_.

Фамилия И.О.: Карасев О.В.
Алгебра: 5
Физика: 5
Русский: 5

Фамилия И.О.: Соболев В.В.
Алгебра: 5
Физика: 4
Русский: 5

Фамилия И.О.: Карасева Л.В.
Алгебра: 3
Физика: 4
Русский: 4

Фамилия И.О.: Дергунин А.С.

Алгебра: 4

Физика: 4

Русский: 4

Фамилия И.О.: -

Отличники:

Карасев О.В.

Хорошисты:

Дергунин А.С.

Соболев В.В.

Ввести данные другого класса?(+/-): +

Класс: 8ю

Чтобы закончить ввод учащихся в поле 'Фамилия И.О.:' укажите '- '.

Фамилия И.О.: Пуговкина А.С.

Алгебра: 4

Физика: 4

Русский: 4

Фамилия И.О.: Семенов П.В.

Алгебра: 4

Физика: 5

Русский: 4

Фамилия И.О.: Никифорова А.А.

Алгебра: 4

Физика: 5

Русский: 5

Фамилия И.О.: -

Отличники:

-

Хорошисты:

Никифорова А.А.

Пуговкина А.С.

Семенов П.В.

Ввести данные другого класса?(+/-): -

21. Два приятеля играют в слова: придумывают по очереди слова, составленные из заданного алфавита. Напишите код программы, которая проверяет, что каждый символ придуманного слова

присутствует в алфавите, проверяет наличие этого слова в «корзинах слов» и выводит победителя в момент, когда как минимум один из приятелей отказывается продолжать игру. После окончания игры «корзины слов» очищаются.

Пример:

Для завершения игры укажите '-' вместо слова.

Алфавит игры: ('к', 'е', 'н', 'т', 'а', 'в', 'р')

Буквы можно использовать неоднократно.

Имя первого участника: Саша

Имя второго участника: Женя

Саша (ход 1): тара

Женя (ход 1): река

Саша (ход 2): варвар

Женя (ход 2): тара

Слово "тара" уже было.

Саша (ход 3): кит

Женя (ход 3): карета

Слово "кит" не удовлетворяет условиям игры.

Саша (ход 4): -

Женя (ход 4): рана

Корзина слов (Саша): ['тара']

Корзина слов (Женя): ['река', 'карета', 'рана']

Женя, поздравляем! Вы выиграли!

22. Напишите код программы, которая генерирует ключи для словаря согласно шаблону для каждого нового слова, введенного пользователем с клавиатуры, затем выводит все значения отсортированного по ключам словаря и общее количество слов.

Шаблон ключа: (количество букв в слове) + (тире) + (порядковый номер слова с таким количеством букв)

Пример:

Чтобы закончить ввод, в поле 'Введите слово:' укажите '_ '.

Введите слово: мама

Слову "мама" присвоен ключ 4-1

Введите слово: кабан

Слову "кабан" присвоен ключ 5-1

Введите слово: тема

Слову "тема" присвоен ключ 4-2

Введите слово: литр

Слову "литр" присвоен ключ 4-3

Введите слово: -

Словарь:

4-1 : мама

4-2 : тема

4-3 : литр

5-1 : кабан

Общее количество слов в словаре: 4

Глава 8. Строки

Любую последовательность символов, включая пробел, знаки препинания и цифры, можно назвать упорядоченным набором или массивом символов. Часто именно строковый тип данных помогает установить понятный диалог между программой и пользователем и осуществляет работу под конкретный запрос пользователя.

Оператор преобразования в строку `str()`

Мы уже не раз пользовались преобразованием строки в целые или вещественные числа после ввода с клавиатуры пользователем с помощью операторов `int()` и `float()`. Так, операторы `+` и `*` меняли операцию конкатенации и дублирования на сложение и умножение. Строковый тип данных также имеет свой оператор `str()`, который преобразует свой аргумент в строку, наделив его своим свойством неизменяемости.

Рассмотрим следующую задачу. Пользователь поочередно вводит шесть цифр, которые являются разрядами шестизначного числа. На выходе программы необходимо получить число, составленное из сумм разрядных цифр, равноудаленных от середины числа. Например, если введенные пользователем разряды образуют число 143562, то результатом программы будет запись 3108 ($1 + 2$, $4 + 6$, $3 + 5$).

Основные блоки написания программного кода состоят из ввода разрядов, сложения равноудаленных от середины разрядных цифр согласно примеру и вывода получившегося числа на экран.

Однако не нужно забывать, что складывать введенные числа можно, только преобразовав их в числовой формат, а соединять результаты в одно число, преобразовав в строковый:

```
a = int(input('1 разряд: '))
b = int(input('2 разряд: '))
c = int(input('3 разряд: '))
d = int(input('4 разряд: '))
e = int(input('5 разряд: '))
f = int(input('6 разряд: '))

af = a + f
be = b + e
cd = c + d
```

```
afbcd = str(af) + str(be) + str(cd)
print(afbcd)
```

Строка как массив

Для строки, как упорядоченного набора символов, существуют специальные функции. Например, такие как обращение по индексу, вычисление длины строки или всевозможные виды срезов. Рассмотрим каждый из них подробнее.

Обращение по индексу

Изменим условие предыдущей задачи: пусть пользователь вводит шестизначное число целиком. Тогда для подсчета необходимых сумм обратимся к каждому разряду введенного числа как к элементу списка, т. е. по индексу:

```
n = input('Введите шестизначное число: ')
a = int(n[0])
b = int(n[1])
c = int(n[2])
d = int(n[3])
e = int(n[4])
f = int(n[5])

sum_1 = a + f
sum_2 = b + e
sum_3 = c + d

n1 = str(sum_1) + str(sum_2) + str(sum_3)
print(n1)
```

Теперь можно подумать над усовершенствованием кода. Каждая переменная — это зарезервированная область памяти, поэтому, чем меньше их будет использовано в коде, тем меньший объем памяти он будет занимать. В нашем случае убрать дополнительные переменные можно несколькими способами. Например, можно вычислять суммы цифр разрядов прямо в итоговой переменной:

```
n = input('Введите шестизначное число: ')
a = int(n[0])
b = int(n[1])
```

```
c = int(n[2])
d = int(n[3])
e = int(n[4])
f = int(n[5])
```

```
n1 = str(a + f) + str(b + e) + str(c + d)
print(n1)
```

Кроме того, можно не создавать отдельных переменных для каждого разряда, а обращаться к ним сразу по индексу:

```
n = input('Введите шестизначное число: ')

n1 = str(int(n[0]) + int(n[5])) + str(int(n[1])+
int(n[4])) + str(int(n[2])+ int(n[3]))

print(n1)
```

Или, раз уж на то пошло, можно использовать всего одну переменную — для ввода:

```
n = input('Введите шестизначное число: ')

print(str(int(n[0]) + int(n[5])) + str(int(n[1])+
int(n[4])) + str(int(n[2]) + int(n[3])))
```

Важно!

Количество открытых скобок должно быть равно количеству закрытых.

Напомним, что обращение к элементам строки может также происходить по отрицательным индексам:

```
n = input('Введите шестизначное число: ')

print(str(int(n[-6]) + int(n[-1])) + str(int(n[-5]) +
int(n[-2])) + str(int(n[-4]) + int(n[-3])))
```


или по тем и другим одновременно:

```
n = input('Введите шестизначное число: ')

print(str(int(n[0]) + int(n[-1])) + str(int(n[1]) +
int(n[-2])) + str(int(n[2]) + int(n[-3])))
```

Такой способ обращения к элементам строки удобен, если строка имеет очень большую длину и необходима работа с символами в конце строки: например, если нужно изменить утвердительное предложение на восклицательное.

Функция *len()*

Напомним, что функция `len()` возвращает количество элементов в некотором массиве объектов (его длину). Будь то длина списка, кортежа, словаря или строки как набора символов.

Вычислим, например, с помощью этой функции количество разрядов у получившегося в результате предыдущей задачи числа:

```
n = input('Введите шестизначное число: ')

n1 = str(int(n[0]) + int(n[-1])) + str(int(n[1]) +
int(n[-2])) + str(int(n[2]) + int(n[-3]))

print(n1)
print('Количество разрядов нового числа:', len(n1))
```

Срезы строк

Повторим уже известные формы среза для строки:

| | |
|-----------------------------------|------------------------|
| <code>str = 'Hello world!'</code> | |
| <code>str[0:3]</code> | <code>'Hel'</code> |
| <code>str[2:7]</code> | <code>'llo w'</code> |
| <code>str[4:-1]</code> | <code>'o world'</code> |
| <code>str[0:-7]</code> | <code>'Hello'</code> |
| <code>str[0:-6]</code> | <code>'Hello '</code> |
| <code>str[:-7]</code> | <code>'Hello'</code> |
| <code>str[:-6]</code> | <code>'Hello '</code> |

| | |
|----------------------|--------------|
| str = 'Hello world!' | |
| str[2:] | 'llo world!' |
| str[-5:] | 'orld!' |
| str[:7:2] | 'Hlow' |
| str[::-3] | 'Hlwl' |

Методы *find*, *replace* и *count*

Помимо общих функций, применяемых к объектам типа **string**, **list**, **tuple** или **dict**, существуют и другие методы именно для строк, которые делают работу с этим типом более гибкой.

Например:

- `str.find(A)` находит в строке с именем `str` подстроку `A`. Функция возвращает индекс первого вхождения искомой подстроки или возвращает значение `-1` в случае, если данная подстрока отсутствует:

```
str1 = 'MamaMia'
print(str1.find('m')) #2
print(str1.find('M')) #0
print(str1.find('maM')) #2
print(str1.find('Mama')) #0
print(str1.find('MAMA')) #-1
```

Обратите внимание, что учитывается регистр символов.

- `str.replace(A, B)` заменяет строку `A` на строку `B` в строке с именем `str`:

```
str2 = 'MAMAMIA'
print(str2.replace('AMA', 'ama')) #MamaMIA
```

- `str.count(A)` подсчитывает количество вхождений в строку с именем `str` подстроки `A`:

```
str3 = 'MAMAMIA'
print(str3.count('aM')) #0
print(str3.count('AM')) #2
```

Ознакомиться с функциями и методами строк можно здесь: <https://pythonworld.ru/typy-dannyx-v-python/stroki-funkcii-i-metody-strok.html>, <https://pythontutor.ru/lessons/str/>.

Что нового мы узнали в восьмой главе?

1. *Строка* — это последовательность символов, включая пробел, знаки препинания, цифры и т. д.

2. `str()` — оператор преобразования в строку;

3. Перечислим основные операции со строками.

- *Конкатенация* — операция присоединения, «склеивания» символов или их наборов, которая выполняется с помощью оператора `+`. «Склейка» возможна только между строками.

- *Дублирование (повторение)* — операция многократной конкатенации строки с самой собой, которая выполняется с помощью оператора `*`. Дублирование возможно только в том случае, если операция выполняется между строкой и целым числом.

- *Доступ к элементу строки по индексу*. Любая строка представима в виде последовательности, у каждого элемента которой есть свой индекс. Чтобы обратиться к элементу строки по индексу, необходимо записать имя переменной, содержащую строку, а после в квадратных скобках указать индекс элемента в строке.

Важно учитывать, что индексы элементов строки слева направо начинаются с 0, справа налево — с -1.

- *Срезы строк*. В общем случае у метода извлечения среза строки три аргумента. По умолчанию первый аргумент равен 0, второй равен длине строки, третий равен 1:

| | |
|-------------------------|--|
| <code>str(A:B:C)</code> | от <i>A</i> , до <i>B</i> не включая, с шагом <i>C</i> |
| <code>str(:B:C)</code> | от 0, до <i>B</i> не включая, с шагом <i>C</i> |
| <code>str(A::C)</code> | от <i>A</i> , до последнего символа не включая, с шагом <i>C</i> |
| <code>str(::~C)</code> | от 0, до последнего символа не включая, с шагом <i>C</i> |
| <code>str(A:B)</code> | от <i>A</i> , до <i>B</i> не включая, с шагом 1 |
| <code>str(:B)</code> | от 0, до <i>B</i> не включая, с шагом 1 |
| <code>str(A:)</code> | от <i>A</i> , до последнего символа не включая, с шагом 1 |

- `len()` — функция, которая позволяет узнать количество объектов в некотором массиве объектов, в том числе количество символов в строке. Другими словами, функция `len()` вычисляет длину строки как массива символов.
- `str.find()` — метод, который находит в строке указанную подстроку и возвращает индекс первого вхождения искомой подстроки (в случае, если данная подстрока отсутствует, возвращает `-1`).
- `str.replace()` — метод, который используется для замены одной подстроки на другую.
- `str.count()` — метод, который подсчитывает количество вхождений в строку подстроки.
- Для методов `find()`, `replace()` и `count()` учитывается регистр символов.
- Ознакомиться с функциями и методами строк можно здесь: <https://pythonworld.ru/typy-dannyx-v-python/stroki-funkcii-i-metody-strok.html>, <https://pythontutor.ru/lessons/str/>.

Практикум

Основные понятия главы

1. Заполните таблицу, используя правила среза строк:

| | |
|--|-----------|
| <code>str = 'Сегодня хорошая погода!'</code> | |
| <code>str[:7]</code> | 'Сегодня' |
| <code>str[8:11]</code> | |
| <code>str[8:14]</code> | |
| <code>str[8:-10]</code> | |
| <code>str[-4:-1]</code> | |
| <code>str[:7:3]</code> | |

2. Используя свойства среза строк, напишите команды, которые меняют слова выражения местами:

| | |
|-------------------------------------|------------------|
| <code>str = 'мама мыла раму'</code> | |
| <code>str[:5] + ... + ...</code> | 'мама раму мыла' |
| | 'мыла мама раму' |
| | 'раму мыла мама' |

Какие варианты перестановки слов могут быть еще?

Чтение кода

3. Определите результат работы следующих фрагментов программ, заполняя правый столбец таблицы:

| № | Код | Результат |
|-----|--|-----------|
| 3.1 | <pre>l = list('35260790') for i in range(len(l)): l[i] = int(l[i]) l.sort() print(l.count(0))</pre> | |
| 3.2 | <pre>import math a = int(math.pow(2.0, 8)) print(str(a)*2)</pre> | |
| 3.3 | <pre>a = 'qwerty' i = 0 while i < len(a): if i%2 == 0: a = a.replace(a[i], '*') i += 1 print(a)</pre> | |
| 3.4 | <pre>s = 'THE BEST' b = '' for i in range(len(s)): b = b + s[i] + ' ' print(b + '!' * (b.count('E') + 1))</pre> | |

Поиск ошибок

4. Объясните, почему в следующих частях программы Python выведет ошибку, используя варианты ответов:

- A. Аргумент не строкового типа.
- B. Невозможное присваивание.
- C. Невозможная операция сложения/конкатенации.
- D. Невозможная операция сравнения.
- E. Выход за пределы диапазона индексов.

| № | Код | Ошибка |
|-----|---|--------|
| 4.1 | <pre>for i in range(-7, 6): i = str(i) + 0 print(i*2)</pre> | |

| № | Код | Ошибка |
|-----|---|--------|
| 4.2 | <pre>float(input("a = ")) = a float(input("b = ")) = b while a + b > 100: print("Error")</pre> | |
| 4.3 | <pre>a = "Белка и стрелка" b = "" for i in range(len(a) + 1): b += (a[i] + " ") print(b)</pre> | |
| 4.4 | <pre>n = input(n =) list_n = list(range(0,n,2)) list_new = [] i = 0 while list_n[i] < n and list_n[i] > 0: if i = 15: continue list_new.append(list_n[i]) i += 1 print(tuple(list.new))</pre> | |
| 4.5 | <pre>n = input('Введите натуральное число: ') while n < 1: n = input('Введите число снова: ') if n >= 1: print(n)</pre> | |

Оптимизация

5. Преобразуйте итоговый код для задачи о шестизначном числе, из которого получается новое число (с помощью сложения цифр разрядов, равноудаленных от середины, и на экран выводится количество разрядов нового числа), таким образом, чтобы были устранены следующие возможные ошибки:

- пользователь может ввести 0 для первого разряда;
- пользователь может ввести число большей или меньшей длины, чем указано в условии.

Разработка

6. Пользователь вводит с клавиатуры пятизначное число. В результате выполнения программы первые две цифры этого числа заменяются его двумя последними цифрами, а остальные остаются неизменными. Учтите возможные ошибки из задания 5.

Пример:

```
Введите число: 38651
Новое число: 51651
```

7. Пользователь вводит с клавиатуры натуральное число. В результате выполнения программы необходимо:

- получить число, равное сумме всех разрядов введенного числа, если длина числа превышает 4 символа;
- получить число, в котором цифры записаны в обратном порядке, если длина числа равна 4;
- получить кортеж, в котором все цифры числа записаны через запятую в порядке возрастания, если длина числа менее 4 символов.

Учтите, что пользователь может ввести число, меньшее 1.

Пример 1:

```
Введите натуральное число: 38651
23
```

Пример 2:

```
Введите натуральное число: 3865
5683
```

Пример 3:

```
Введите натуральное число: 386
('3', '6', '8')
```

8. Найдите методы преобразования строки к верхнему и нижнему регистрам и преобразуйте следующую строку в обоих формах:

«Год 2038-й. Спустя 18 месяцев жизни и работы на поверхности Марса команда из шести исследователей садится обратно в космический аппарат и возвращается на Землю».

9. Используя срезы строк, метод `replace()`, а также методы преобразования к верхнему и нижнему регистрам, получите:

9.1) из строки «ВЕЗДЕХОД» строку «МАРСОХОД»;

9.2) из строки «МОЛОКО» строку «олово»;

9.3) из строки «революционер» строки «эволюция» и «КОЛЛЕКЦИОНЕР».

Глава 9. Функции

От программы к подпрограмме

Ранее в книге были рассмотрены элементарные составляющие (типы данных, условные и циклические конструкции, подключение дополнительных модулей и т. д.), комбинируя которые, можно было решить ту или иную задачу.

В некотором смысле **функция** — это и есть выполнение определенной задачи: «утюг с дополнительной *функцией* управления через Bluetooth», «отключение *функции* управления голосом на смартфоне», «сердце выполняет в системе кровообращения насосную *функцию*».

Представим, что в одной программе можно будет объединить совершенно различные по своему назначению задачи и расширить функционал устройства, для которого пишется программа. Это гораздо приближеннее к реальному миру, где один только смартфон заменяет будильник, записную книжку, фото- и видеоаппаратуру, mp3-плеер и др.

Естественно, каждой подобной функции необходимо свое обозначенное место в программе. Поэтому части программы, которые будут решать отдельные задачи (или подзадачи), будем называть **подпрограммами**.

Типы подпрограмм

Существует два основных типа подпрограмм: **«сделано и забыто»** и **«создано для чего-то большего»**.

Например, процедура досмотра учащегося при входе в пункт проведения экзамена — это последовательность действий охранника, подразумевающая, что результат досмотра повлияет на пропуск учащегося, но в дальнейшем использоваться не будет (**«сделано и забыто»**), в то время как запись на бланках паспортных данных, которые идентифицируют конкретного участника экзамена, будет передана в вышестоящие структуры проверки документов и информирования о результатах (запись паспортных данных на бланках **«создана для чего-то большего»**).

В программировании подпрограмму для первого случая («сделано и забыто») так и называют — **процедура (простая функция)**, а для второго случая («создано для чего-то большего») — **функция**:

- **процедура** — это подпрограмма, которая не возвращает результат в программу для дальнейшего использования, но изменяет состояние объектов;
- **функция** — это подпрограмма, которая возвращает результат в программу для дальнейшего использования.

Также существует еще один тип подпрограммы — **метод**, который привязан к некоторому классу или его объекту.

Например, для списка `my_list` могут быть вызваны:

- **функции** `len(my_list)` и `print(my_list)`, которые работают как для объектов типа **list**, так и для объектов других типов;
- **методы** `my_list.append()` и `my_list.sort()`, которые работают только для объектов типа **list**.

Определение функции в программе, оператор def

Любая подпрограмма начинает свое существование в программе с указания уникального имени, по которому к ней можно будет обращаться. Поэтому в Python существует **оператор def**, который указывает на место создания подпрограммы и ее именования. Можно догадаться, что название **def** происходит от английского слова *definition* («определение»).

Например:

| | |
|-----------------------------|--|
| <code>def Error():</code> | Определение подпрограммы с именем <code>Error</code> |
| <code>def Sum(a,b):</code> | Определение подпрограммы с именем <code>Sum</code> |
| <code>def square(a):</code> | Определение подпрограммы с именем <code>square</code> |

Заметим, что в примерах выше помимо оператора **def** и названия подпрограммы есть скобки, двоеточие, а также наличие и отсутствие переменных в скобках:

- наличие скобок после названия подпрограммы — обязательная часть определения ее «полномочий»: самостоятельна ли подпрограмма (пустые скобки) или требуется некоторый набор «подчиненных» (скобки содержат одну или несколько переменных); переменные внутри скобок называются *параметрами (аргументами)*, о которых поговорим ниже;
- двоеточие — также обязательная часть объявления подпрограммы, так называемый мост между ее названием и содержимым (телом подпрограммы).

Тело подпрограммы — это набор команд, следующий после ее именованя и отделенный дополнительной табуляцией от начала строки. Как в условных и циклических конструкциях, после нажатия клавиши **Enter** двоеточие позволяет автоматически отделять табуляцией описание тела подпрограммы.

Далее мы будем использовать понятие «функция», подразумевающая произвольный тип подпрограммы (и процедуру, и функцию).

Передача параметров (аргументов) в функцию

Наличие или отсутствие аргументов, их количество и место написания при определении функции имеют исключительную роль. Функция `square(2, 6)` никогда не запустится, если изначально была определена как `square(a)`.

- Если в функции отсутствует аргумент, то при ее вызове просто выполняется тело функции.
- Если в функции присутствует аргумент (переменная), то его значение используется в теле функции, что влияет на результат ее вызова.

Пример процедуры (простой функции) без аргументов:

```
#Вычисление площади квадрата со стороной 5

#Определение процедуры (простой функции) :
def square():
    a = 5
    print('Площадь квадрата со стороной 5 =', a*a)

#Вызов процедуры:
square() #Площадь квадрата со стороной 5 = 25
```

Пример процедуры (простой функции) с аргументом:

```
#Вычисление площади квадрата со стороной a

#Определение процедуры (простой функции) :
def square(a):
    print('Площадь квадрата со стороной', a, '=', a*a)
#Вызов процедуры:
square(5) # Площадь квадрата со стороной 5 = 25
square(3) # Площадь квадрата со стороной 3 = 9
square(1.2) # Площадь квадрата со стороной 1.2 = 1.44
```

Важно!

Python не читает команды в теле подпрограммы без ее вызова. Другими словами, без вызова подпрограмма считается просто объявленной, но не используемой.

Возврат значения из функции, оператор return

Напомним, что функция, в отличие от процедуры, возвращает в программу необходимые для дальнейшей работы значения. Мостом для передачи этих значений является оператор **return**.

Если в функции присутствует данный оператор, то значение, которое он передает обратно в программу, может быть снова использовано.

Важно!

Использование функций в программе без оператора **return** невозможно.

Пример функции без аргументов:

```
#Определение площади поверхности куба с ребром 5
#Определение функции, вычисляющей площадь одной грани
#куба:
def square():
    a = 5
    return a*a
```

```
#Определение процедуры, вычисляющей площадь
#поверхности куба:
def surface():
    surface = 6*square()
    print('Площадь поверхности куба с ребром 5 =',
          surface)
#Вызов процедуры surface() с использованием функции
#square():
surface() #Площадь поверхности куба с ребром 5 = 150
```

Пример функции с аргументами:

```
#Определение площади поверхности куба с ребром a
#Определение функции, вычисляющей площадь одной грани
#куба:
def square(a):
    return a*a
#Определение процедуры, вычисляющей площадь
#поверхности куба:
def surface(a):
    surface = 6*square(a)
    print('Площадь поверхности куба с ребром', a, '=',
          surface)
#Вызов процедуры surface() с использованием функции
#square():
surface(5) #Площадь поверхности куба с ребром 5 = 150
surface(3) #Площадь поверхности куба с ребром 3 = 54
surface(1.2) #Площадь поверхности куба с ребром
             #1.2 = 8.64
```

Рассмотрим, как происходит чтение программы с функциями на примере последнего кода.

1. Читая строку `def square(a):`, Python понимает, что в коде может встретиться подпрограмма с названием `square`, но не запускает для выполнения тело этой функции.

2. То же самое происходит со строкой `def surface(a):` — подпрограмма с названием `surface` объявлена, но пока не вызвана для использования.

3. Встретившись с первым вызовом `surface(5)`, Python запускает тело процедуры `surface(a)`, учитывая, что в переменную `a` поместили число 5.

4. В теле `surface(a)` Python встречает функцию `square(a)` и переходит к выполнению этой подпрограммы, учитывая, что в переменную `a` поместили число 5.

5. После выполнения функции `square(a)` оператор `return` возвращает обратно в процедуру `surface(a)` число 25 для дальнейшего умножения на число 6.

6. После выполнения процедуры `surface(5)` на экране появляется результат 'Площадь поверхности куба с ребром 5 = 150'.

7. В следующей строке программа начинает выполнение `surface(3)`, проходя повторно пункты 3–6 при `a = 3` и т. д. до окончания всей программы.

Внимание!

Если описание функции расположить в коде после ее вызова, Python выдаст ошибку, поскольку в таком случае в его памяти не остается записи о наличии данной функции. Другими словами, Python может выполнять только те функции, которые объявлены и описаны до их вызова в программном коде.

Стандартные функции Python

Многократно используемые функции `print()` (для вывода значений и комментариев на экран), `input()` (для ввода данных в программу), `int()`, `double()` (для преобразования типов данных) или `range()` (для организации списков чисел) являются *встроенными* или *стандартными функциями Python*. Как с помощью известных формул в математике могут быть выведены более сложные формулы и теоремы, так и в программировании с помощью стандартных функций можно написать новые программы, решающие индивидуальные или узкоспециализированные задачи.

При этом стандартные функции существуют в Python для всех типов данных: числового, строкового, а также списков, кортежей, словарей и т. д. Естественно, что список стандартных функций Python не заканчивается теми, что были описаны в книге. Предлагаем ознакомиться с ними в сети. Например, на pythonworld.ru.

Напомним, что многие узконаправленные готовые функции объединены в категории, так называемые библиотеки или модули¹ (например, `random`, `math` или `turtle`, подключаемые с помощью инструкции `import`), которые находятся в общем доступе, постоянно дополняются и обновляются разработчиками и пользователями.

Ситуации, в которых целесообразно использование функций

1. **Повторение однотипных участков кода.** Чем больше однотипных участков кода, тем больше объем программы и трудозатраты на исправление ошибок в каждом из них.

2. **Сложность и/или неудобочитаемость длинного кода.** Чем сложнее конструкция программного кода, тем больше возникает необходимость группировки частей кода в функции.

Рекурсивные функции

Выше было упомянуто, что функцию целесообразно использовать, когда многократно повторяется какая-то часть кода. А что, если повторяется не часть, а код полностью? Возможно ли это? Да, это возможно, и у такого механизма даже есть название — *рекурсия* или *рекурсивная функция*, которая вызывает сама себя с другим аргументом.

Тот самый коридор, который получается, если поставить два зеркала напротив друг друга, называется *рекурсией*. При этом рекурсия может быть и не бесконечна (матрешка повторяет себя конечное число раз, лист папоротника состоит из конечного числа похожих листков), но каждый следующий по-прежнему наследует структуру предыдущего.

Один из самых известных примеров математической рекурсии — факториал. Факториал числа n — это произведение всех чисел натурального ряда до числа n включительно. Например, $5!$ («пять факториал») = $5 \cdot 4 \cdot 3 \cdot 2 \cdot 1$.

¹ Модуль — это отдельный файл с расширением `.py`, содержащий инструкции для Python. Библиотека — это совокупность «стандартных», популярных и широко используемых модулей.

При разложении факториала числа на множители можно увидеть, что разложение также содержит факториал и поэтому является примером рекурсивного алгоритма.

$$\begin{aligned} 5! &= 5 \cdot 4! = 5 \cdot 4 \cdot 3! = 5 \cdot 4 \cdot 3 \cdot 2! = 5 \cdot 4 \cdot 3 \cdot 2 \cdot 1! = \\ &= 5 \cdot 4 \cdot 3 \cdot 2 \cdot 1 \cdot 0! = 5 \cdot 4 \cdot 3 \cdot 2 \cdot 1 \cdot 1 = 120. \end{aligned}$$

Итак, чтобы вычислить *факториал числа n* , нужно умножить *число n* на *факториал предыдущего числа в натуральном ряде, т.е. на факториал числа $(n - 1)$* .

Приведем пример вычисления функции факториала на языке Python. При этом для факториала есть исключение: по определению $0! = 1$, поэтому не забудем добавить в программу условную конструкцию.

```
#Определение функции факториала с учетом исключения:
```

```
def factorial(n):  
    if n==0:  
        return 1  
    else:  
        return n*factorial(n-1)
```

```
#Вызов функции factorial(5):
```

```
factorial(5)
```

Попробуйте запустить эту программу.

Выдает ошибку? Да, что-то в этом коде не так.

Напомним, что оператор **return** возвращает результат в программу для дальнейшего использования. Пропишем шаги выполнения программы, чтобы разобраться, в чем дело.

1. Запуск строки `def factorial(n):` лишь уведомляет Python о том, что существует функция с названием `factorial` и пропускает содержимое функции до ее вызова.

2. Запуск строки `factorial(5)` подразумевает чтение тела функции `factorial(n)` с учетом $n = 5$. Условная конструкция в данном случае содержит в себе повторный вызов функции `factorial(5-1)`, а значит, запускает то же самое тело функции, только с учетом $n = 4$. И так далее, пока не запустится функция `factorial(0)`, являющаяся *точкой останова* (преднамеренное завершение программы или подпрограммы).

На языке псевдокода работа цикла может быть записана как последовательное раскрытие функции `factorial(n)` на каждой его итерации:

```
factorial(5) = 5*factorial(4) = 5*4*factorial(3) =  
             = 5*4*3*factorial(2) = 5*4*3*2*factorial(1) =  
             = 5*4*3*2*1*factorial(0) = 5*4*3*2*1*1 = 120
```

А теперь ответим на один вопрос: «*Как используется это число по возвращении в программу?*» В действительности данный оператор помог программе использовать результаты вновь и вновь в течение каждого возвращения в тело функции, но последний результат оказался просто записанным внутри и нигде не указано, как использовать этот результат дальше. Поэтому для исправления ситуации всего лишь добавим инструкцию вывести на экран полученный результат с помощью команды `print()`, а после запустим программу и удостоверимся в правильности ее работы:

```
#Определение функции факториала с учетом исключения:  
def factorial(n):  
    if n==0:  
        return 1  
    else:  
        return n*factorial(n-1)  
  
#Вызов функции factorial(5):  
print(factorial(5)) #120
```

Ура! Заработало!

Анализ рекурсивных алгоритмов является отличным способом усилить свое понимание обработки алгоритмов компьютером. Разберем еще один пример рекурсивного алгоритма и пропишем каждый шаг программы:

```
def F(n):  
    print(n)  
    if n < 5:  
        F(n + 1)  
        F(n + 3)  
F(4)
```

Шаги:

1. `def F(n)` – уведомление Python о существовании функции `F(n)` и пропуск тела функции до ее первого вызова.

2. Запуск функции $F(4)$:

- вывод числа **4** на экран;
- проверка условия $4 < 5$ (истинно):
 - запуск $F(5)$:
 - вывод числа **5** на экран;
 - проверка условия $5 < 5$ (ложно);
 - запуск $F(7)$:
 - вывод числа **7** на экран;
 - проверка условия $7 < 5$ (ложно).

Результат на экране: 4 5 7

Что нового мы узнали в девятой главе?

1. *Подпрограмма* — это часть программы, которая выполняет определенную задачу.

Основные типы подпрограмм:

- *процедура (простая функция)* — это подпрограмма, которая не возвращает результат действия или последовательности действий в программу для дальнейшего использования;
- *функция* — это подпрограмма, которая возвращает результат действия или последовательности действий в программу для дальнейшего использования;
- *метод* — это подпрограмма, которая выполняет задачу для объекта определенного класса.

2. *Оператор def* отвечает за определение подпрограммы в программе.

3. Функции и процедуры могут как содержать *параметры (аргументы)*, так и не содержать их. Они записываются через запятую в скобках после имени подпрограммы при ее определении. При этом:

- если в функции отсутствует аргумент, то при ее вызове выполняется тело функции;
- если в функции присутствует аргумент (входящая переменная), то его значение используется в теле функции, что влияет на результат ее вызова.

4. Мостом для передачи возвращаемых функцией значений является оператор *return*. Если в функции присутствует данный оператор, то значение, которое он передает обратно в программу, может быть снова использовано, при его отсутствии это невозможно.

5. Использование функций целесообразно:

- при повторении однотипных участков кода;
- при необходимости разложения сложного, неудобочитаемого длинного кода на простые составляющие.

6. Функция, которая вызывает сама себя с другим аргументом, называется *рекурсивной функцией*. Одним из примеров рекурсии является математическая функция *факториал натурального числа n* (произведение всех чисел натурального ряда до числа n включительно).

7. *Точка останова* — преднамеренное завершение программы или подпрограммы.

Практикум

Основные понятия главы

1. Определите, к какому типу относятся следующие подпрограммы (нужное подчеркнуть):

| | |
|--|--|
| <pre>def tr_error(a,b,c): sum = a + b + c if sum != 180: print ('Error')</pre> | <p style="text-align: center;">tr_error(a,b,c) процедура/функция</p> |
| <pre>def abc(a,b,c): return a+b+c def abc_new(a,b,c): print (2*abc(a,b,c))</pre> | <p style="text-align: center;">abc(a,b,c) процедура/функция</p> |
| | <p style="text-align: center;">abc_new(a,b,c) процедура/функция</p> |
| <pre>def sum(a,b): return a+b for i in range(5): i = i + sum(i,i) print(i)</pre> | <p style="text-align: center;">sum(a,b) процедура/функция</p> |

Чтение кода

2. Определите результаты вызова подпрограмм, используя код из задачи 1:

| Вызов подпрограмм | Результат |
|--|-----------|
| <code>tr_error(36,72,72)</code> | |
| <code>tr_error(90,45,55)</code> | |
| <code>abc_new(3,7,14)</code> | |
| <code>abc_new('tik', '-', 'tak ')</code> | |
| <code>sum(2,3)</code> | |
| <code>sum('+', '-')</code> | |

3. Определите результат рекурсивных функций:

| № | Программный код | Результат |
|-----|---|-----------|
| 3.1 | <pre>def F(n): if n > 2: return F(n-1)+ F(n-2) else: return 1 print(F(5))</pre> | |
| 3.2 | <pre>def F(n): if n > 2: return F(n-1)+F(n-2)+F(n-3) else: return n print(F(10))</pre> | |
| 3.3 | <pre>def F(n): if n < 5: F(n + 1) F(n + 3) print(F(3))</pre> | |

Поиск ошибок

4. Учащийся написал программу, выводящую символы '*' и '.' в шахматном порядке (количество строк и столбцов матрицы вводятся пользователем самостоятельно). Найдите и исправьте допущенные в коде ошибки.

```

def AB(n,m):
    h = []
    for i in range(n):
        h.append([])
        h[i] = list('.'*m)

    for i in range(n):
        if i%2 != 0:
            for j in range(m):
                if j%2 == 0:
                    h[i][j] = '*'
        else:
            for j in range(m):
                if j%2 != 0:
                    h[i][j] = '*'

    for i in range(n):
        for j in range(m):
            print(h[i][j], end = ' ')
        print()

l = int(input('Введите количество строк: '))
s = int(input('Введите количество столбцов: '))
AB(s,l)

```

Пример правильного вывода:

```

Введите количество строк: 3
Введите количество столбцов: 7
. * . * . * .
* . * . * . *
. * . * . * .

```

5. Учащийся написал программу, которая вычисляет индекс первого вхождения максимального элемента матрицы (количество строк, столбцов, а также элементы матрицы вводятся пользователем самостоятельно). Найдите и исправьте допущенные в коде ошибки.

```

def max(l):
    max = 0
    r = 0
    for i in range(len(l)):
        for j in range(len(l[0])):
            if l[i][j] > max:
                max = l[i][j]

```

```
for i in range(len(l)):
    for j in range(len(l[0])):
        if l[i][j] == max:
            r = '('+i+';' +j+')'
            print('Максимальный элемент:', max)
            return r

n = int(input('Количество строк матрицы: '))
m = int(input('Количество столбцов матрицы: '))
h = []

for i in range(n):
    h.append([])
    c = input()
    h[i] = c.split(' ')

for i in range(n):
    for j in range(m):
        h[i][j] = int(h[i][j])

print(max(l))
```

Пример правильного вывода:

```
Количество строк матрицы: 3
Количество столбцов матрицы:
5 1 4 -9 3 2
6 3 9 1 2
9 0 5 8 -11
Максимальный элемент: 9
(2;3)
```

Оптимизация

6. Дан код программы для учителя, который вписывает 2 числа (максимальное количество баллов за тест и количество набранных за него баллов учащимся), а в результате выполнения программы получает сообщение об отметке учащегося за написанный тест. Отметка «5» ставится от 85% правильных ответов, «4» — от 65%, «3» — от 50% и «2» — менее 50%.

```
m = int(input('MAX количество баллов за тест: '))
k = int(input('Количество набранных баллов: '))
n = (k/m)*100
if n < 50:
    print('Отметка: 2')
elif n >= 50 and n < 65:
    print('Отметка: 3')
```

```
elif n >= 65 and n < 85:  
    print('Отметка: 4')  
elif n >= 85:  
    print('Отметка: 5')
```

- Сформируйте функцию `Test(k,m)` (k — количество набранных за него баллов учащимся, m — максимальное количество баллов за тест), возвращающую в программу отметку учащегося;
- добавьте возможность оценить неограниченное количество учащихся;
- добавьте возможность автоматически создавать словарь с ФИО ученика (ключ) и его оценкой за тест (значение);
- добавьте возможность вывода словаря отсортированным списком учащихся и их отметок.

Разработка

7. Напишите программу, содержащую процедуру, которая определяет четность или нечетность целого числа, вводимого с клавиатуры, и выводит на экран сообщение о результате.

Пример:

```
Введите число: 7  
Число 7 - нечетное
```

8. При покупке на сумму от 3000 рублей в магазине покупатель получает скидку 5%, от 5000 рублей — 7%, от 10 000 — 10%. Напишите программу, содержащую функцию для определения размера скидки (без скидки, 5%, 7% или 10%) и выводящую на экран размер скидки и итоговую сумму покупки.

Пример:

```
Сумма покупки (в руб.): 4300  
Ваша скидка составляет 5%  
Сумма к оплате: 4085.00 руб.
```

9. На карте клиента кафетерия есть некоторое количество бонусных рублей. Напишите программу, содержащую функцию, которая определит сумму к оплате, если клиент решит списать бонусные рубли со своей карты. При этом если бонусные рубли полностью покрывают стоимость заказа, то клиент оплачивает 1 рубль.

Пример:

Количество бонусных рублей на карте: 235

Стоимость заказа (в руб.): 99

Списать бонусные рубли (да/нет): да

Сумма к оплате (в руб.): 1

10. Напишите программу, содержащую функции, которые находят минимальный и максимальный элементы матрицы, индексы их первого вхождения, а также среднее арифметическое всех элементов матрицы (количество строк, столбцов, а также элементы матрицы вводятся пользователем самостоятельно).

Заключение

Итак, в книге мы рассмотрели базовые конструкции программирования на языке Python: от именованния переменных до сборки функций, от простого вывода приветствия 'Hello, world!' до многострочных программ с несколькими вложенными циклами и условными конструкциями.

Чтобы использовать больше возможностей, мы изучали, как подключать дополнительные модули/библиотеки и даже рисовали, а в каждой главе разбирали вопросы возможных ошибок. Мы экспериментировали, оптимизировали готовые программы и разрабатывали свои самостоятельно.

И это только начало! За пределами этой книги еще много нового и интересного.

Поблагодарите себя за этот основательный первый шаг и не забывайте: границы между точными науками (в том числе компьютерными) и повседневностью куда тоньше, чем можно представить.

Так, память компьютера — это воспоминания человека, условные конструкции — его каждодневный выбор, циклы — принцип маленьких шагов на пути к успеху, библиотеки и модули — неограниченные возможности и командная работа, а многофункциональная программа — целый завод, где каждый работник имеет свои место, условия труда и ценность в работе целой системы.

Поэтому, обучаясь программированию, вы осваиваете навык смотреть и действовать глобально во всех сферах вашей жизни.

Успехов!

Ответы

Глава 1

| | | | | | |
|---|----------|---|---|----------|-----------|
| 1 | 7 + c | 1 | 3 | оператор | |
| | c, 7 | 2 | | 1 | выражение |
| | + | 3 | | 2 | операнды |
| | сложение | 4 | | 4 | операция |

| | | | | |
|---|-------------|----------------------------------|-----------|-----------|
| 2 | (p * 2) % 3 | умножение, остаток от деления | выражение | |
| | p, 2, 3 | | | операнды |
| | *, % | | | операторы |
| | | | | операции |

| | | | | |
|---|-----------------|---|-----------|-----------|
| 3 | a**(13//2) | возведение в степень, целая часть от деления | выражение | |
| | a, 13//2, 13, 2 | | | операнды |
| | ** , // | | | операторы |
| | | | | операции |

4.1 >>> "Корабль 'Лавр' лавировал"
"Корабль 'Лавр' лавировал"

4.2 >>> 'Она сказала:"Топ-топ"*2
'Она сказала:"Топ-топ"Она сказала:"Топ-топ"'

4.3 >>> '''
Привет!
Как дела?

:)'''
'\nПривет!\nКак дела?\n\n:)'

5 >>> ('a'+ 'b')*5
'ababababab'

6 >>> '1'*10
'1111111111'
>>> '11'*5
'1111111111'

```

>>> '11111'*2
'1111111111'
>>> '1111111111'*1
'1111111111'
>>> 10*'1'
'1111111111'
>>> 5*'11'
'1111111111'
>>> 2*'11111'
'1111111111'
>>> 1*'1111111111'
'1111111111'

```

- 7.1 $((2**5)+5)//5 = (32+5)//5 = 37//5 = 7$
- 7.2 $((13%2)-1.0)/5 = (1-1.0)/5 = 0.0/5 = 0.0$
- 7.3 $((8//5)**2)-10.2 = (1**2)-10.2 = 1-10.2 = -9.2$
- 7.4 $((((2+10.2)-0.2)**2)//4)%2 = (((12.2-0.2)**2)//4)%2 = ((12.0**2)//4)%2 = (144.0//4)%2 = 36.0%2 = 0.0$
- 8 *, *, +
- 9 *, + , // , -
- 10 // , - , % , +
- 11 (a * 5)) — присутствует незакрывающая скобка
"Hello" — присутствуют незакрывающиеся двойные кавычки
- 12 "Привет"Лена" — после вторых двойных кавычек объект не определен
- 13 6 = a — переменную a в число 6 положить нельзя
'hi' - 'i' — для строк операция вычитания не определена
- 14 "55" * '11' — операция конкатенации возможна только в случае, когда один из двух операндов является целым числом
"Little" — строка должна начинаться и заканчиваться одним типом кавычек
5 / 0 — деление на 0 невозможно
- 15 'hi' / 2 — операция деления для строк не определена
- 16 "Сегодня в 'Москве' прекрасная погода"
'Сегодня в "Москве" прекрасная погода'
- 17 "'Как дела?' - спросила Оля."
"'Как дела?' - спросила Оля." * 2
- 18 2 + 6x = 7 — не определена связь с объектом x

```
19.1 a = int(input('Введите сторону a: '))
      b = int(input('Введите сторону b: '))
      c = int(input('Введите сторону c: '))
      P = a + b + c
      print(P)
```

```
19.2 Введите сторону a: 7
      Введите сторону b: 9
      Введите сторону c: 13
      29
```

```
19.3 a = int(input('Введите сторону a: '))
      b = int(input('Введите сторону b: '))
      c = int(input('Введите сторону c: '))
      print(a + b + c)
```

```
20 print('Дата моего рождения 18 апреля')
```

```
21.1 print(-365*20)
      -7300
```

```
21.2 print(0.5*650-25)
      300.0
```

```
21.3 print((132/2+1)*6)
      402.0
```

```
21.4 print(-576+(23*45-13))
      446
```

```
22 print('0000 **** 0 0 ****')
      print('0 * * 0 0 0 * *')
      print('0 **** 0 0 0 ****')
      print('0000 * * 00000 * *')
```

```
23 print('*'*17)
      print('*      /\      +      *')
      print('*      / \      +      *')
      print('* + /      \      *')
      print('*      +-----+      *')
      print('*      | +--+|      *')
      print('*      | |  ||      *')
      print('*  __|__|__||__      *')
      print('*'*17)
```

```
24 a = int(input('Введите число: '))
      print(a*a)
```

- ```

25 a = int(input('Введите число: '))
 print('Следующее число:', a+1)

26 a = int(input('Первое число: '))
 b = int(input('Второе число: '))
 print('Сумма чисел:', a+b)

27 a = int(input('Первое число: '))
 b = int(input('Второе число: '))
 print(a, '*', b, '=', a*b)

28 a = int(input('Первое число: '))
 b = int(input('Второе число: '))
 c = int(input('Второе число: '))
 print(a, '+', b, '=', a+b)
 print(b, '-', c, '=', b-c, '\n', a, '*', c, '=', a*c)

29 a = int(input("Количество 'о': "))
 b = int(input("Количество 'м': "))
 print('о'*a+'м'*b, '\nОбщее количество символов:', a+b)

```

## Глава 2

| 1 | Имя переменной | Можно (+) / нельзя (-)<br>использовать |
|---|----------------|----------------------------------------|
|   | 5a             | -                                      |
|   | d              | +                                      |
|   | pol_10         | +                                      |
|   | abs            | +                                      |
|   | mod            | +                                      |
|   | сумма 1        | -                                      |
|   | num-5          | -                                      |

| 2 | Объект         | int | float | str |
|---|----------------|-----|-------|-----|
|   | -5             | +   | -     | -   |
|   | 46.0           | -   | +     | -   |
|   | -1000000000000 | +   | -     | -   |
|   | '2+5'          | -   | -     | +   |
|   | -9.5           | -   | +     | -   |
|   | 1              | +   | -     | -   |
|   | '900'          | -   | -     | +   |

|   |                               |            |                 |                   |
|---|-------------------------------|------------|-----------------|-------------------|
| 3 |                               | b=input(a) | b=int(input(a)) | b=float(input(a)) |
|   | Тип аргумента функции input() | str        | str             | str               |
|   | Тип аргумента функции int()   | -          | str             | -                 |
|   | Тип аргумента функции float() | -          | -               | str               |
|   | Тип переменной b              | str        | int             | float             |

4.1 Error

4.2 00  
01  
10  
11

4.3 1C

4.4 1.0

5.1 A

5.2 D

5.3 E

5.4 C

5.5 B

```
6 a = int(input('Введите возраст старшего брата: '))
 b = int(input('Введите возраст младшего брата: '))
 print('Разница в возрасте между братьями', a-b, 'лет.')
```

- ```
7.1 a = input('Введите число: ')
    print(a)

7.2 a = int(input('Введите целое число: '))
    print(a+1)

7.3 a = float(input('Введите дробное число: '))
    print(a+1, a+2, a+3)

7.4 a = input('Введите целое число: ')
    print(str(a)*10)

7.5 a = input('Введите любой символ: ')
    print((a+' ')*5)

7.6 a = int(input('Введите первое целое число: '))
    b = int(input('Введите второе целое число: '))
    print(a+b, a-b, a*b)

7.7 a = eval(input('Введите первое число: '))
    b = eval(input('Введите второе число: '))
    print(a+b, a-b, a*b)

8 a = int(input('Введите первое число: '))
  b = int(input('Введите второе число: '))
  print(str(a*b)*3)

9 a = input('Введите строку: ')
  b = int(input('Введите количество повторений: '))
  print(a*b)

10 a = input('Введите сообщение: ')
    print('Для вас одно новое сообщение: "', a, '"')
```
- ```
11 a = int(input('Количество покупаемых единиц хлеба: '))
 b = int(input('Количество покупаемых единиц молока: '))
 c = int(input('Количество покупаемых единиц масла: '))
 print('\nОбщая сумма к оплате -',
 a*35.5+b*74+c*120, 'руб.')
```

### Глава 3

- 1.1 Если звенит звонок, то это конец урока или перемены.
- 1.2 Если посмотреть, с какой стороны растёт мох на дереве, то можно определить северную сторону света без компаса.
- 1.3 Если в мобильном телефоне отсутствует sim-карта, то можно совершить звонок только по сети Wi-fi.
- 2.1 Истинное: От перестановки слагаемых сумма не меняется.  
Ложное: От перестановки множителей произведение равно нулю.
- 2.2 Истинное: Предложение начинается с заглавной буквы.  
Ложное: От перестановки знаков препинания смысл не меняется.
- 3 Если собака лает, значит, по забору идет кошка.  
Если собака лает, значит, по забору идет кошка или пришел сосед.

4

| Условие          | Соответствующий вопрос  | Ответ на вопрос | Истинность (1) / ложность (0) условия |
|------------------|-------------------------|-----------------|---------------------------------------|
| $12 \geq 12 - 0$ | 12 больше или равно 12? | Да              | 1                                     |
| $5 == 2 + 3.0$   | 5 равно 5.0?            | Да              | 1                                     |
| $7 < 2 * 3$      | 7 меньше 6?             | Нет             | 0                                     |
| $0 > -23$        | 0 больше -23?           | Да              | 1                                     |
| $5 <= 0.2 + 3$   | 5 меньше или равно 3.2? | Нет             | 0                                     |

5

| Условие                             | Истинность (1) / ложность (0) условия |
|-------------------------------------|---------------------------------------|
| $(23 - 23) == (70 - 70)$            | 1                                     |
| $7 != 2 * 3$                        | 1                                     |
| $(1 + 2 * 3 - 4) > (1 - 2 * 3 + 4)$ | 1                                     |
| $2 ** 10 >= 32 * 32$                | 1                                     |
| $5 \% 2 < 5 // 2$                   | 1                                     |
| $10 / 2 <= (2 + 100 \% 3)$          | 0                                     |



6.2 7

6.3 9

6.4 error

6.5 11  
12

6.6 8  
16  
18

7.1 D

7.2 A

7.3 H

7.4 F

7.5 C

7.6 E

7.7 G

7.8 B

```
8.1 i = 100
 if i == 100:
 i = i - int('96')
 print(i)
```

```
8.2 i = 1
 if i > 1:
 i = 1
 elif i == 1:
 print(i)
 print(i)
```

```
9 a = float(input('Введите сторону ромба: '))
 h = float(input('Введите высоту ромба: '))
 if a > 0 and h > 0:
 s = 0.5*a*h
 print('Площадь ромба со
 стороной', a, 'и высотой', h, 'равна', s)
 else:
 print('Error')
```

- 10 `a = input('Введите пароль: ')  
 if a == 'qwerty':  
 print('ENG')  
 elif a == 'йцукен':  
 print('РУС')`
- 11 `a = int(input('Введите целое число: '))  
 if a < 0:  
 print(a - 1)  
 elif a > 0:  
 print(a + 1)  
 else:  
 print('0')`
- 12 `a = int(input('Введите целое число: '))  
 if a%2 == 0:  
 print('Число', a, 'четное')  
 elif a%2 != 0:  
 print('Число', a, 'нечетное')`
- 13 `a = float(input('Введите a: '))  
 b = float(input('Введите b: '))  
 print('a + b =', a, '+', b, '=', a+b)  
 print('a - b =', a, '-', b, '=', a-b)  
 print('a * b =', a, '*', b, '=', a*b)  
  
 if b != 0:  
 print('a / b =', a, '/', b, '=', a/b)  
 else:  
 print('a / b не существует (делить на 0 нельзя!')`
- 14 `a = float(input('Введите число: '))  
 if a >= -50 and a <= 50:  
 print('Число', a, 'принадлежит диапазону [-50;50]')  
 else:  
 print('Число', a, 'не принадлежит диапазону  
 [-50;50]')`
- 15 `a = float(input('Введите число: '))  
 if a > -50 and a <= 50:  
 print('Число', a, 'принадлежит диапазону (-50;50]')  
 else:  
 print('Число', a, 'не принадлежит диапазону  
 (-50;50]')`

- ```
16 a = eval(input('Введите левую границу промежутка: '))
    b = eval(input('Введите правую границу промежутка: '))
    c = input('Вывести интервал, полуинтервалы или отрезок
    с заданными границами? - ')
    if c == 'интервал' or c == 'Интервал' or
    c == 'ИНТЕРВАЛ':
        print('Интервал (' , a , ';' , b , ')')
    elif c == 'полуинтервалы' or c == 'Полуинтервалы' or
    c == 'ПОЛУИНТЕРВАЛЫ':
        print('Полуинтервалы (' , a , ';' , b , ')
    и [' , a , ';' , b , ')')
    elif c == 'отрезок' or c == 'Отрезок' or
    c == 'ОТРЕЗОК':
        print('Отрезок [' , a , ';' , b , ']')

17 a = int(input('Введите двузначное положительное
число: '))
    if a >= 10 and a <= 99:
        a1 = a // 10
        a2 = a % 10
        if a1 > a2:
            print(str(a2) + str(a1))
        elif a1 < a2:
            print(str(a1) + str(a2))
        else:
            print(a)
    else:
        print('Error')

18 a = int(input('Введите четырехзначное положительное
число: '))
    b = a % 10
    if a >= 1000 and a <= 9999 and b != 0:
        a4 = a % 10
        a3 = a % 100 // 10
        a2 = a % 1000 // 100
        a1 = a % 10000 // 1000
        print(str(a4) + str(a3) + str(a2) + str(a1))
    else:
        print('Error')

19 a = int(input('Введите номер билета: '))
    if a >= 1000 and a <= 9999:
        a4 = a % 10
        a3 = a % 100 // 10
        a2 = a % 1000 // 100
        a1 = a % 10000 // 1000
```

```

    if (a4 + a3) == (a1 + a2):
        print('Билет №',a,'- счастливый по-питерски!')
    else:
        print('Билет №',a,'- несчастливый по-питерски!')
else:
    print('Error')

```

20 a = int(input('Введите трехзначное положительное число: '))

```

if a >= 100 and a <= 999:
    a3 = a % 10
    a2 = (a % 100) // 10
    a1 = a // 100
    if a1 <= a2 and a1 <= a3:
        if a2 <= a3:
            print(str(a1) + str(a2) + str(a3))
        else:
            print(str(a1) + str(a3) + str(a2))
    elif a2 <= a1 and a2 <= a3:
        if a1 <= a3:
            print(str(a2) + str(a1) + str(a3))
        else:
            print(str(a2) + str(a3) + str(a1))
    elif a3 <= a1 and a3 <= a2:
        if a1 <= a2:
            print(str(a3) + str(a1) + str(a2))
        else:
            print(str(a3) + str(a2) + str(a1))
else:
    print('Error')

```

21 print('''

Используемые операции и их обозначения:

Сложение - '+'

Вычитание или унарный минус - '-'

Умножение - '*'

''')

a = input('Введите арифметическое выражение, используя данные операции: ')

print('\nОтвет:',eval(a))

Глава 4

| | | | |
|---|---------------------------------------|---------------------------------------|--------------------------------|
| 1 | range(-6, -3) | от -6, до -3 не включая, с шагом 1 | [-6, -5, -4] |
| | range(9) | от 0, до 9 не включая, с шагом 1 | [0, 1, 2, 3, 4, 5, 6, 7, 8] |
| | range(7, -3, -5) | от 7, до -3 не включая, с шагом -5 | [7, 2] |
| | range(1, 4) | от 1, до 4 не включая, с шагом 1 | [1, 2, 3] |
| | range(0, 4, 4) | от 0, до 4 не включая, с шагом 4 | [0] |
| | range(-6, 2, 3) | от -6, до 2 не включая, с шагом 3 | [-6, -3, 0] |
| | range(5, 10) | от 5, до 10 не включая, с шагом 1 | [5, 6, 7, 8, 9] |
| | range(7) | от 0, до 7 не включая, с шагом 1 | [0, 1, 2, 3, 4, 5, 6] |
| | <i>Вариант 1:</i> range(-10, 1, 5) | от -10, до 1 не включая, с шагом 5 | [-10, -5, 0] |
| | <i>Вариант 2:</i> range(-10, 2, 5) | от -10, до 2 не включая, с шагом 5 | |
| | <i>Вариант 3:</i> range(-10, 3, 5) | от -10, до 3 не включая, с шагом 5 | |
| | <i>Вариант 4:</i> range(-10, 4, 5) | от -10, до 4 не включая, с шагом 5 | |
| | <i>Вариант 5:</i> range(-10, 5, 5) | от -10, до 5 не включая, с шагом 5 | |

```
2.1 s = 0
for k in range(3,10):
    s = s + 6
print(s)
```

Тип цикла: с заданным числом повторений

Количество повторений/предусловие: 7, заданное с помощью функции range(3, 10)

Характеристика итерации: увеличение значения переменной s на 6

Количество итераций: 7

```
2.2 k = 0
    for m in range(1,10,2):
        k = k + 6
        print(k)
```

Тип цикла: с заданным числом повторений

Количество повторений/предусловие: 5, заданное с помощью функции `range(1, 10, 2)`

Характеристика итерации: увеличение значения переменной `k` на 6 и вывод значения переменной `k` на экран

Количество итераций: 5

```
2.3 s = 10
    for k in range(5):
        s -= k
    print(s)
```

Тип цикла: с заданным числом повторений

Количество повторений/предусловие: 5, заданное с помощью функции `range(1, 10, 2)`

Характеристика итерации: уменьшение значения переменной `s` на `k`, значения которой сформированы функцией `range(5)`

Количество итераций: 5

```
3.1 s = -30
    while s < 0:
        s += 6
    print(s)
```

Тип цикла: с предусловием

Количество повторений/предусловие: `s < 0`

Характеристика итерации: увеличение значения переменной `s` на 6

Количество итераций: 5

```
3.2 s = 1
    n = 15
    while n > 0:
        n = n - s
        s = s * 2
    print(n)
```

Тип цикла: с предусловием

Количество повторений/предусловие: `n > 0`

Характеристика итерации: уменьшение значения переменной `n` на `s`, увеличение значения переменной `s` в 2 раза

Количество итераций: 4

```

3.3 k = 1
while k <= 64:
    k = k*2
    print(k)

```

Тип цикла: с предусловием

Количество повторений/предусловие: $k \leq 64$

Характеристика итерации: увеличение значения переменной k в 2 раза и вывод этого значения на экран

Количество итераций: 7

- 4 *Примечание:* за нулевую итерацию принимаются входные значения.

Таблица итераций для задания 2.1

| № итерации | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
|--------------|---|---|----|----|----|----|----|----|
| k | | 3 | 4 | 5 | 6 | 7 | 8 | 9 |
| s | 0 | 6 | 12 | 18 | 24 | 30 | 36 | 42 |
| <i>Вывод</i> | | | | | | | | 42 |

Таблица итераций для задания 2.2

| № итерации | 0 | 1 | 2 | 3 | 4 | 5 |
|--------------|---|---|----|----|----|----|
| t | | 1 | 3 | 5 | 7 | 9 |
| k | 0 | 6 | 12 | 18 | 24 | 30 |
| <i>Вывод</i> | | 6 | 12 | 18 | 24 | 30 |

Таблица итераций для задания 2.3

| № итерации | 0 | 1 | 2 | 3 | 4 | 5 |
|--------------|----|----|---|---|---|---|
| k | | 0 | 1 | 2 | 3 | 4 |
| s | 10 | 10 | 9 | 7 | 4 | 0 |
| <i>Вывод</i> | | | | | | 0 |

Таблица итераций для задания 3.1

| № итерации | 0 | 1 | 2 | 3 | 4 | 5 | |
|---------------------------|-----|----------------|----------------|----------------|----------------|---------------|--------------|
| <i>Условие</i> $s < 0$ | | $-30 < 0$ + | $-24 < 0$ + | $-18 < 0$ + | $-12 < 0$ + | $-6 < 0$ + | $0 < 0$ - |
| s | -30 | -24 | -18 | -12 | -6 | 0 | |
| <i>Вывод</i> | | | | | | | 0 |

Таблица итераций для задания 3.2

| № итерации | 0 | 1 | 2 | 3 | 4 | |
|--------------------|----|---------------|---------------|---------------|--------------|--------------|
| Условие $n > 0$ | | $15 > 0$ + | $14 > 0$ + | $12 > 0$ + | $8 > 0$ + | $0 > 0$ - |
| n | 15 | 14 | 12 | 8 | 0 | |
| s | 1 | 2 | 4 | 8 | 16 | |
| Вывод | | | | | | 0 |

Таблица итераций для задания 3.3

| № итерации | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | |
|------------------------|---|------------------|------------------|------------------|------------------|-------------------|-------------------|-------------------|--------------------|
| Условие $k \leq 64$ | | $1 \leq 64$ + | $2 \leq 64$ + | $4 \leq 64$ + | $8 \leq 64$ + | $16 \leq 64$ + | $32 \leq 64$ + | $64 \leq 64$ + | $128 \leq 64$ - |
| k | 1 | 2 | 4 | 8 | 16 | 32 | 64 | 128 | |
| Вывод | | 2 | 4 | 8 | 16 | 32 | 64 | 128 | |

```

5 s = 0
  for k in range(3,10):
    s = s + 6
    if s == 30:
      break
  print(s)
6 k = 1
  while k <= 64:
    k = k*2
    if k == 8 or k == 64:
      continue
  print(k)
7 s = 10
  k = 0
  while k < 5:
    s -= k
    k += 1
  print(s)
8 s = -30
  for i in range(5):
    s += 6
  print(s)

```


9.1 25
27
29

9.2 -2

9.3 Ура!
Ура!
Ура!

9.4 35
37
39
41

9.5 1
2
0

```
10 count = 0
    i = 1
    while i <= 3:
        j = 1
        while j <= 4:
            count = count + 1
            j = j + 1
        i = i + 1
    print(count)
```

11.1 B

11.2 A

11.3 C

11.4 F

11.5 D

11.6 G

11.7 E

11.8 H

```
12 a = int(input('Введите натуральное число: '))
    if a>=1 and a<1000000:
        s=0
        while a != 0:
            s += a%10
            a = (a - a%10)//10
```

```

    if s/3 == int(s/3):
        print('Введенное число делится на 3 без
              остатка')
    else:
        print('Введенное число не делится на 3 без
              остатка')
else:
    print('Неверный ввод')

```

```

13.1 count = 0
    for j in range(1,3):
        for i in range(1,6):
            count = count + 1
    print(count)

```

```

13.2 count = 0
    for i in range(1,6):
        for j in range(1,3):
            count = count + 1
    print(count)

```

```

14.1 for i in range(25,51):
        print(i)

```

```

14.2 for i in range(25,51,2):
        print(i+1)

```

```

14.3 for i in range(25,51,2):
        if i > 45:
            continue
        print(i)

```

```

15.1 a = 50
    while a >= 25:
        print(a)
        a -= 1

```

```

15.2 a = 25
    while a <= 50:
        if a % 2 == 1:
            print(a)
        a += 2

```

```

15.3 a = 25
    while a <= 50:
        if a % 2 == 0:
            print(a)
        a += 1
        if a == 46:
            a += 1

```

- ```
16 a = input('Введите натуральное число любой
разрядности: ')
b = int(a)

for i in a:
 print(b % 10)
 b = b // 10
```
- ```
17 a = int(input('Введите неотрицательное целое
число: '))
if a >= 0:
    p = 1
    if a > 0:
        while a > 0:
            p *= a
            a -= 1
    print('Факториал числа равен',p)
else:
    print('Неверный ввод')
```
- ```
18 a = input('Введите натуральное число любой
разрядности: ')
a1 = int(a)
a2 = ''

for i in a:
 a2 += str(a1 % 10)
 a1 = a1 // 10

if a == a2:
 print('Число',a,'является палиндромом')
else:
 print('Число',a,'не является палиндромом')
```
- ```
19 a = int(input())
b = int(input())

while a != 0 and b != 0:
    if a > b:
        a = a % b
    else:
        b = b % a

print(a+b)
```

Глава 5

1.1 +

1.2 +

1.3 -

1.4 -

1.5 +

1.6 +

2

| Команда | Результат |
|---|---|
| <code>random.randrange(5)</code> | случайное число от 0, до 5 не включая, с шагом 1 |
| <code>random.randrange(1, 4)</code> | случайное число от 1, до 4 не включая, с шагом 1 |
| <code>random.randrange(-1, 5, 1)</code> | случайное число от -1, до 5 не включая, с шагом 1 |
| <code>random.randrange(-2, 8, 2)</code> | случайное число от -2, до 8 не включая, с шагом 2 |
| <code>random.randrange(8, 0, -3)</code> | случайное число от 8, до 0 не включая, с шагом -3 |

3

| Операция | Запись | Результат |
|---|---------------------------------|-----------|
| Модуль числа -5 | <code>math.fabs(-5)</code> | 5 |
| Округление 2,404 до ближайшего целого числа сверху | <code>math.ceil(2,404)</code> | 3 |
| Округление -2,404 до ближайшего целого числа сверху | <code>math.ceil(-2,404)</code> | -2 |
| Округление 2,404 до ближайшего целого числа снизу | <code>math.floor(2,404)</code> | 2 |
| Округление -2,404 до ближайшего целого числа снизу | <code>math.floor(-2,404)</code> | -3 |
| Остаток от деления 54 на 3 | <code>math.fmod(54, 3)</code> | 0.0 |
| Квадратный корень из 4096 | <code>math.sqrt(4096)</code> | 64.0 |

4.1 64.0

4.2 2

4.3 11

4.4 $< 0.0 >$
 $\quad \quad \quad \wedge$
 $\quad \quad \quad \text{---} \quad \text{---}$

5.1 D

5.2 A

5.3 C

5.4 E

5.5 F

5.6 B

5.7 G

5.8 H

6 `import random`

```
n = int(input('За какое максимальное число попыток вы
хотите угадать число? '))
a = random.randint(0,10)
p = 0
```

```
while p < n:
    b = int(input('Введите число: '))
    if a == b:
        print('Поздравляем! Вы выиграли!')
        break
    elif a > b:
        print('Загаданное число больше введенного')
    else:
        print('Загаданное число меньше введенного')
    p = p + 1
if p == n:
    print('Вы проиграли. Загаданное число:', a)
```

7 `import math`

```
a = float(input('Введите длину ребра: '))
if a <= 0:
    print('Error')
else:
    V = math.pow(a,3)
    print('Объем куба со стороной', a, 'равен', V)
```

8 import math as m

```
a = float(input('Введите коэффициент a: '))
b = float(input('Введите коэффициент b: '))
c = float(input('Введите коэффициент c: '))
```

```
D = m.pow(b,2) - 4*a*c
print('D =', D)
```

```
if D > 0:
    x1 = (-b - m.sqrt(D))/(2*a)
    x2 = (-b + m.sqrt(D))/(2*a)
    print('x1 =', x1, '\nx2 =', x2)
elif D == 0:
    x = (-b - m.sqrt(D))/(2*a)
    print('x =', x)
else:
    print('Корней нет')
```

9 import math

```
a = math.sqrt(11**3-8**3)/math.factorial(6)*math.pi
print(a)
```

10.1 import math

```
r = float(input('Введите радиус окружности: '))

if r > 0:
    C = 2*math.pi*r
    S = math.pi*math.pow(r,2)
    print('Длина окружности радиусом', r, 'равна', C)
    print('Площадь круга радиусом', r, 'равна', S)
else:
    print('Error')
```

10.2 import math

```
d = float(input('Введите диаметр окружности: '))

if d > 0:
    C = math.pi*d
    S = math.pi*math.pow(d/2,2)
    print('Длина окружности диаметром', d, 'равна', C)
    print('Площадь круга диаметром', d, 'равна', S)
else:
    print('Error')
```

11 import random

```
a = int(input('Введите левую границу диапазона: '))
b = int(input('Введите правую границу диапазона: '))

r = random.randint(a,b)

print('\nЗадача:\nВычислите длину окружности
радиусом',r, '. Значение числа пи округлите до сотых.')
```

12 import random

```
a = int(input('Введите левую границу диапазона: '))
b = int(input('Введите правую границу диапазона: '))

r = random.randint(a,b)

print('\nЗадача:\nВычислите длину окружности
радиусом ',r, '. Значение числа пи округлите до сотых.')
```

```
d = float(input('\nПроверка ответа: '))

if d == 2*3.14*r:
    print('Верно')

else:
    print('Неверно')
```

13 import random

```
a = int(input('Введите левую границу диапазона: '))
b = int(input('Введите правую границу диапазона: '))

r = random.randint(a,b)

print('\nЗадача:\nВычислите длину окружности
радиусом',r, '. Значение числа пи округлите до сотых.')
```

```
d = float(input('\nПроверка ответа: '))

if d == 2*3.14*r:
    print('Верно')

while d != 2*3.14*r:
    print('\nНеверно. Попробуй еще раз')
    d = float(input('\nНовый ответ: '))
```

```

if d == 2*3.14*r:
    print('Верно. Ты нашел правильный ответ!')
    break

```

14.1 import random

```

a = int(input('Введите левую границу диапазона: '))
b = int(input('Введите правую границу диапазона: '))

```

```

c = random.randint(a,b)

```

```

print('\nЗадача:\nВычислите периметр квадрата со
стороной',c)

```

```

d = int(input('\nПроверка ответа: '))

```

```

if d == 4*c:
    print('Верно')

```

```

while d != 4*c:
    print('\nНеверно. Попробуй еще раз')
    d = int(input('\nНовый ответ: '))
    if d == 4*c:
        print('Верно. Ты нашел правильный ответ!')
        break

```

14.2 import random

```

a = int(input('Введите левую границу диапазона: '))
b = int(input('Введите правую границу диапазона: '))

```

```

c = random.randint(a,b)
d = random.randint(a,b)

```

```

print('\nЗадача:\nВычислите площадь прямоугольника со
сторонами',c,'и',d, '.')

```

```

m = int(input('\nПроверка ответа: '))

```

```

if m == c*d:
    print('Верно')

```

```

while m != c*d:
    print('\nНеверно. Попробуй еще раз')
    m = int(input('\nНовый ответ: '))
    if m == c*d:
        print('Верно. Ты нашел правильный ответ!')
        break

```


Глава 6

| | | |
|---|------|---|
| 1 | fd() | 1 |
| | bk() | 2 |
| | lt() | 3 |
| | rt() | 4 |

| | |
|---|------------|
| 3 | left() |
| 1 | forward() |
| 4 | right() |
| 2 | backward() |

2 #Рисование окружности в квадрате

```
import turtle as t
t.color('red')
t.fd(100)
t.lt(90)
t.fd(100)
t.lt(90)
t.fd(100)
t.lt(90)
t.fd(100)
t.lt(90)
t.bk(50)
t.circle(50)
t.up()
t.fd(50)
```

3.1 Окружность**3.2 Треугольник****3.3 Прямоугольник****3.4 Квадрат****3.5 Круг****4.1 С****4.2 Е****4.3 А****4.4 G****4.5 D****4.6 В****4.7 F**

```
5 import turtle as t
```

```

print('Выберите цвет пера:
      \n\nblue(синий) - 1\ngreen(зеленый) - 2
      \norange(оранжевый) - 3\npink(розовый) - 4')

c = input('\nВведите номер цвета: ')

if c == '1':
    t.color('blue')
elif c == '2':
    t.color('green')

elif c == '3':
    t.color('orange')
elif c == '4':
    t.color('pink')

for i in range(4):
    t.forward(50)
    t.left(90)

```

```

t.left(30)
t.backward(20)
t.forward(20)
t.left(60)
t.forward(50)
t.right(60)
t.backward(20)
t.left(60)
t.backward(50)
t.right(90)
t.forward(50)
t.left(30)
t.forward(20)

```

```

6.1 import turtle as t
t.forward(150)
t.right(120)
t.forward(150)
t.right(120)
t.forward(150)
t.up()
t.forward(50)

```

```
6.2 import turtle as t
    t.backward(150)
    t.left(120)
    t.backward(150)
    t.left(120)
    t.backward(150)
    t.up()
    t.backward(50)
```

```
6.3 import turtle as t
    t.backward(150)
    t.right(120)
    t.backward(150)
    t.right(120)
    t.backward(150)
```

```
7.1 import turtle as t
    t.color('red')
    t.forward(100)
    t.left(90)
    t.forward(100)
    t.left(90)
    t.forward(100)
    t.left(90)
    t.forward(100)
    t.backward(50)
    t.circle(50)
    t.up()
    t.forward(100)
```

```
7.2 import turtle as t

    t.color('red')
    t.forward(100)
    t.left(90)
    t.forward(100)
    t.left(90)
    t.forward(100)
    t.left(90)
    t.forward(100)

    t.begin_fill()
    t.color('green','green')
    t.circle(50)
    t.end_fill()
```

```
t.up()
t.forward(50)
```

```
7.3 import turtle as t
t.color('red')
t.forward(100)
t.left(90)
t.forward(100)
t.left(90)
t.forward(100)
t.left(90)
t.forward(100)
t.left(90)
t.forward(100)
t.circle(-50)
t.up()
t.forward(50)
```

Движение Черепашки против часовой стрелки при положительном аргументе функции `circle()` и по часовой при отрицательном аргументе.

```
8 import turtle as t
#И
t.left(90)
t.backward(50)
t.right(30)
t.forward(58)
t.left(30)
t.backward(50)
```

#Поворот Черепашки и отступ

```
t.right(90)
t.up()
t.forward(20)
t.down()
t.left(90)
```

#М

```
t.forward(50)
t.right(150)
t.forward(30)
t.left(120)
t.forward(30)
t.right(150)
t.forward(50)
```

#Поворот Черепашки и отступ

```
t.left(90)
t.up()
t.forward(20)
t.down()
t.left(50)
```

#Я

```
t.forward(35)
t.left(130)
t.forward(23)
t.right(90)
t.forward(23)
t.right(90)
t.forward(23)
t.right(90)
t.forward(50)
t.up()
t.forward(70)
t.left(90)
```

```
9 import turtle as t
  t.color('green')
```

#И

```
t.left(90)
t.backward(50)
t.right(30)
t.forward(58)
t.left(30)
t.backward(50)
```

#Поворот Черепашки и отступ

```
t.right(90)
t.up()
t.forward(20)
t.down()
t.left(90)
```

#М

```
t.forward(50)
t.right(150)
t.forward(30)
t.left(120)
t.forward(30)
```

```
t.right(150)
t.forward(50)
```

#Поворот Черепашки и отступ

```
t.left(90)
t.up()
t.forward(20)
t.down()
t.left(50)
```

#я

```
t.forward(35)
t.left(130)
t.forward(23)
t.right(90)
t.forward(23)
t.right(90)
t.forward(23)
t.right(90)
t.forward(50)
```

#Подчеркивание

```
t.up()
t.goto(0, -70)
t.color('black')
t.down()
t.left(90)
t.forward(121)
```

#Рамка

```
t.width(3)
t.up()
t.goto(-20, -90)
t.down()
t.forward(161)
t.left(90)
t.forward(110)
t.left(90)
t.forward(161)
t.left(90)
t.forward(110)
```

```
t.up()  
t.forward(30)
```

```
10 import turtle as t  
t.forward(90)  
t.up()  
t.left(180)  
t.forward(30)  
t.left(90)  
t.forward(30)  
t.left(180)  
t.down()  
t.forward(90)  
t.up()  
t.left(180)  
t.forward(30)  
t.left(90)  
t.forward(30)  
t.left(180)  
t.down()  
t.forward(90)  
t.up()  
t.left(180)  
t.forward(30)  
t.left(90)  
t.forward(30)  
t.left(180)  
t.down()  
t.forward(90)  
t.backward(2)
```

#Крестик

```
t.up()  
t.right(90)  
t.forward(28)  
t.right(135)  
t.down()  
t.forward(35)  
t.up()  
t.backward(35)  
t.right(45)  
t.forward(25)  
t.left(135)
```

```
t.down()
t.forward(35)
```

#0

```
t.up()
t.right(45)
t.forward(36)
t.right(90)
t.forward(42)
t.down()
t.circle(12)
t.up()
t.forward(70)
t.left(90)
```

```
11.1 import turtle as t
for j in range (5):
    t.begin_fill()
    t.color('red','red')
    for i in range(3):
        t.forward(50)
        t.right(120)
        if i == 2:
            t.forward(55)
    t.end_fill()
```

```
11.2 import turtle as t
for j in range (5):
    if j%2 == 0:
        t.color('red')
    else:
        t.color('green')
    for i in range(3):
        t.forward(50)
        t.right(120)
        if i == 2:
            t.forward(55)
```

```
11.3 import turtle as t

for j in range (5):
    t.begin_fill()
    if j%2 == 0:
        t.color('red','red')
```



```
else:
    t.color('green','green')
for i in range(3):
    t.forward(50)
    t.right(120)
    if i == 2:
        t.forward(50)
t.end_fill()
```

```
12.1 import turtle as t
for j in range (5):
    t.begin_fill()
    t.color('purple','purple')
    for i in range(4):
        t.forward(50)
        t.right(90)
        if i == 3:
            t.forward(60)
    t.end_fill()
```

```
12.2 import turtle as t
for j in range (5):
    if j%2 == 0:
        t.color('purple')
    else:
        t.color('orange')
    for i in range(4):
        t.forward(50)
        t.right(90)
        if i == 3:
            t.color('black')
            t.forward(60)
```

```
13 import turtle as t
t.width(3)
for j in range(3):
    t.color('brown')
    t.circle(25)
    t.right(120)
t.up()
t.forward(150)
t.down()
```

```
for j in range(3):
    t.begin_fill()
    t.color('black','khaki')
    for i in range(3):
        t.forward(50)
        t.left(120)
    t.end_fill()
    t.right(120)
t.up()
t.forward(150)
t.down()
for j in range(3):
    t.begin_fill()
    t.color('grey','pink')
    for i in range(4):
        t.forward(40)
        t.left(90)
    t.end_fill()
    t.right(120)
t.up()
t.home()
t.goto(75,-150)
t.down()
for j in range(3):
    t.begin_fill()
    t.color('purple','yellow')
    for i in range(5):
        t.forward(33)
        t.left(72)
    t.end_fill()
    t.right(120)
t.up()
t.forward(150)
t.down()
for j in range(3):
    t.begin_fill()
    t.color('red','orange')
    for i in range(6):
        t.forward(28)
        t.left(60)
    t.end_fill()
    t.right(120)
t.up()
t.forward(900)
```

- 14 Функция `speed()` отвечает за скорость передвижения Черепашки по холсту: чем больше аргумент, тем быстрее она движется.

Глава 7

1

| | int | float | str | list | tuple | dict |
|------------------------|-----|-------|-----|------|-------|------|
| -5 | + | - | - | - | - | - |
| [7, -9, 'good'] | - | - | - | + | - | - |
| 46.0 | - | + | - | - | - | - |
| (17, 0, [1,2,3]) | - | - | - | - | + | - |
| -1000000000000000 | + | - | - | - | - | - |
| '2+5' | - | - | + | - | - | - |
| {17: 0, 0: [1,2,3]} | - | - | - | - | - | + |

2

| Пример создания словаря | +/- |
|---|-----|
| <code>D = dict(1 = 'total', 2 = 'money')</code> | - |
| <code>D = dict(1, 'total', 2, 'money')</code> | - |
| <code>D = dict(a1 = 'total', a2 = 'money')</code> | + |
| <code>D = dict(a1 = 'total'; a2 = 'money')</code> | - |
| <code>D = dict([(1, 'total'), (2, 'money')])</code> | + |
| <code>D = dict([(a1, 'total'), (a2, 'money')])</code> | - |
| <code>D = dict([(a1; 'total'), (a2; 'money')])</code> | - |
| <code>D = dict([('a1', 'total'), ('a2', 'money')])</code> | + |

3

```
>>> L = [0,1,1,1,1,1]
>>> L[3] = L[1] + L[2]
>>> L[4] = L[2] + L[3]
>>> L[5] = L[3] + L[4]
>>> L
[0, 1, 1, 2, 3, 5]
```

```
4 >>> A = {'int': '_', 'float': '_', 'str': '_', 'list':
  '_', 'tuple': '_', 'dict': '_'}
>>> A['int'] = 5
>>> A['float'] = 0.0
>>> A['str'] = 'hello'
>>> A['list'] = [1,2]
>>> A['tuple'] = (1,2.9,'0',[3,0,'+'])
>>> A['dict'] = {'a': 'Автобус', 'б': 'Бензин',
  'в': 'Водитель'}
>>> A
{'int': 5, 'float': 0.0, 'str': 'hello', 'list': [1,
2], 'tuple': (1, 2.9, '0', [3, 0, '+']), 'dict':
{'a': 'Автобус', 'б': 'Бензин', 'в': 'Водитель'}}
```

5 [[26,8,-1],0,15]
 Да, элементы переменной *B* можно изменить, так как ее тип данных — список.

6 [0, 4, 3]
 Да, элементы переменной *B* можно изменить, так как ее тип данных — словарь.

7 ([1, 2, 3], [4, 5, 6], {1: [1, 2, 3], 2: [4, 5, 6], 3: [7, 8, 9]})
 Нет, элементы переменной *D* изменить нельзя, так как ее тип данных — кортеж.

8 ['T', 'w', 'i', 's', 't']

9 [1, 2, ['y', 'e', 'a', 'r', 's']]

10 [1,['abc', 0, '2'], 36,('hello', ',', 'world','!'),[100, 500]]

11.1 С

11.2 Е

11.3 А

11.4 F

11.5 G

11.6 D

11.7 В

11.8 Н

```
12.1 password = tuple()
a = 5
q = input('Сменить пароль? Да - 1, Нет - 0: ')
if q == '1':
    print('Количество символов пароля -',a)
    password = tuple(input('Введите пароль: '))
    if len(password) >= a:
        n = input('Показать текущий пароль? Да - 1,
        Нет - 0: ')
        if n == '1':
            print(password)
        elif n == '0':
            print('Спасибо за внимание!')
        else:
            print('Команда введена неверно')
    else:
        print('Неверное количество символов')
elif q == '0':
    n = input('Показать текущий пароль? Да - 1,
    Нет - 0: ')
    if n == '1':
        print(password)
    elif n == '0':
        print('Спасибо за внимание!')
    else:
        print('Команда введена неверно')
else:
    print('Команда введена неверно')
```

```
12.2 password = tuple()
a = 5
q = input('Сменить пароль? Да - 1, Нет - 0: ')
if q == '1':
    print('Количество символов пароля -',a)
    print('Пароль должен содержать хотя бы один
    символ из следующих: ! @ $')
    password = tuple(input('Введите пароль: '))
    if len(password) >= a and (password.count('!')
    > 0 or password.count('@') > 0 or password.
    count('$') > 0):
        n = input('Показать текущий пароль? Да - 1,
        Нет - 0: ')
        if n == '1':
            print(password)
```

```

        elif n == '0':
            print('Спасибо за внимание!')
        else:
            print('Команда введена неверно')
    else:
        print('Неверное количество СИМВОЛОВ
или отсутствует хотя бы один СИМВОЛ ИЗ
следующих: ! @ $')
elif q == '0':
    n = input('Показать текущий пароль? Да - 1,
Нет - 0: ')
    if n == '1':
        print(password)
    elif n == '0':
        print('Спасибо за внимание!')
    else:
        print('Команда введена неверно')
else:
    print('Команда введена неверно')

```

```

13 import random
list_code = []
i = 1
while i < 21:
    code = random.randint(1,20)
    for j in range(21):
        if code in list_code:
            code = 0
    if code != 0:
        list_code.append(code)
        print('code ', i ,': ',code)
        i += 1
all_codes = tuple(list_code)
print('Призовые коды: ', all_codes[::5])

```

```

14 bd_shop = {}
print('Ввести новый товар в базу данных магазина?')
n = int(input('Да - 1/Нет - 0: '))
while n == 1:
    a = input('Введите артикул: ')
    b = input('Введите название: ')
    if a in bd_shop:
        print('Произвести замену позиции ',a,' - ',
bd_shop[a],'?')

```

```
c = int(input('Да - 1/Нет - 0: '))
if c == 1:
    bd_shop[a] = b
    print('Замена произведена')
    print('Ввести новый товар в базу данных магазина?')
    n = int(input('Да - 1/Нет - 0: '))
elif c == 0:
    c1 = input('Введите артикул заново: ')
    c2 = input('Введите название заново: ')
    bd_shop[c1] = c2
else:
    bd_shop[a] = b
    print('Позиция добавлена')
    print('Ввести новый товар в базу данных магазина?')
    n = int(input('Да - 1/Нет - 0: '))

print(bd_shop)
```

```
15 names = input('Введите фамилии участников через запятую:\n')

list_names = names.split(',')
list_names.sort()

n = len(list_names)

print('\nОбщий список участников:')
for i in range(len(list_names)):
    print(i+1,list_names[i])

k = int(input('\nВведите количество участников в подгруппе: '))

while (k < 1):
    print('Некорректный ввод')
    k = int(input('\nВведите количество участников в подгруппе: '))
```

```
list_groups = []
if n//k == n/k:
    for i in range(n//k):
        list_groups.append([])
        for j in range(k):
            list_groups[i].append(list_names[0])
            del list_names[0]

    for i in range(len(list_groups)):
        print('\nПодгруппа ',i+1)
        for j in range(k):
            print(list_groups[i][j])

else:
    for i in range(n//k):
        list_groups.append([])
        for j in range(k):
            list_groups[i].append(list_names[0])
            del list_names[0]
    list_groups.append([])
    list_groups[-1] = list_names

    for i in range(len(list_groups)-1):
        print('\nПодгруппа ',i+1)
        for j in range(k):
            print(list_groups[i][j])

    print('\nРезерв:')
    for i in range(len(list_groups[-1])):
        print(list_groups[-1][i])
```

```
16.1 numbers = input('Введите числа через пробел: ')
list_numbers = numbers.split()
count = 0
i = 0

while list_numbers != []:
    count += 1
    del list_numbers[i]

print('Количество чисел:',count)
```


- ```
16.2 numbers = input('Введите числа через запятую: ')
list_numbers = numbers.split(',')
count = 0
i = 0

while list_numbers != []:
 if float(list_numbers[i]) <= 0:
 count += 1
 del list_numbers[i]

print('Количество неположительных чисел:', count)
```
- ```
16.3 numbers = input('Введите числа через запятую: ')
list_numbers = numbers.split(',')
count = 0
i = 0

while list_numbers != []:
    if (float(list_numbers[i]) > 0 and
        float(list_numbers[i])%3 == 0):
        count += 1
        del list_numbers[i]

print('Количество положительных чисел, кратных 3:',
      count)
```
- ```
17 D = dict(a1 = '4', a2 = '3', a3 = '2', a4 = '1')
list_keys = list(D.keys())
list_keys.sort(reverse = True)

for i in list_keys:
 print(i, ':', D[i])
```
- ```
18.1 a = input('Введите целые числа через пробел: ')
list_all = a.split()
list1 = []
list2 = []

for i in range(len(list_all)):
    if int(list_all[i]) % 2 == 0:
        list1.append(list_all[i])
    else:
        list2.append(list_all[i])
print('Список четных чисел:\n',list1)
print('Список нечетных чисел:\n',list2)
```

```

18.2 a = input('Введите целые числа через пробел: ')
list_all = a.split()
list1 = []
list2 = []
list3 = []
for i in range(len(list_all)):
    if int(list_all[i]) > 0:
        list1.append(list_all[i])
    elif int(list_all[i]) < 0:
        list2.append(list_all[i])
    else:
        list3.append(list_all[i])
print('Список положительных чисел:\n',list1)
print('Список отрицательных чисел:\n',list2)
print('Дополнительный список:\n',list3)

```

```

18.3 a = input('Введите целые числа через пробел: ')
list_all = a.split()
list1 = []
list2 = []
for i in range(len(list_all)):
    if int(list_all[i])//5 != 0 and
        int(list_all[i]) % 5 == 0:
        list1.append(list_all[i])
    else:
        list2.append(list_all[i])
print('Числа, кратные 5:\n',list1)
print('Все остальные числа:\n',list2)

```

```

18.4 a = input('Введите целые числа через пробел: ')
list_all = a.split()
list1 = []
list2 = []
list3 = []

for i in range(len(list_all)):
    if int(list_all[i]) == 0:
        list3.append(list_all[i])
    elif int(list_all[i]) % 3 == 0 and
        int(list_all[i]) % 9 != 0:
        list1.append(list_all[i])
    elif int(list_all[i]) % 3 == 0 and
        int(list_all[i]) % 9 == 0:
        list2.append(list_all[i])

```

```
else:
    list3.append(list_all[i])

print('Список чисел, которые делятся на 3, но не
делятся на 9:\n',list1)
print('Список чисел, которые делятся и на 3,
и на 9:\n',list2)
print('Дополнительный список:\n',list3)

19 a = input('Добавить нового ученика?(+/-): ')
list_students = []
while a == '+':
    student = input('Введите фамилию И.О.: ')
    print()
    list_students.append(student)
    a = input('Добавить нового ученика?(+/-): ')
if a == '-':
    print('\nСписок учащихся: ')
list_students.sort()
for i in range(len(list_students)):
    print(i+1, '-', list_students[i])

20 new_class = input('Класс: ')
while new_class != '-':
    print("Чтобы закончить ввод, в поле
'Фамилия И.О.:' укажите '-'.")

    student = input('\nФамилия И.О.: ')
    list_students = []
    i = 0
    while student != '-':
        alg = int(input('Алгебра: '))
        ph = int(input('Физика: '))
        rus = int(input('Русский: '))
        list_students.append([student, alg, ph, rus])
        i += 1
        student = input('\nФамилия И.О.: ')

    if student == '-':
        print()

List1 = []
List2 = []
```

```

for i in range(len(list_students)):
    if list_students[i].count(5) == 3:
        List1.append(list_students[i][0])
    elif list_students[i].count(2) == 0 and
list_students[i].count(3) == 0:
        List2.append(list_students[i][0])

List1.sort()
List2.sort()

print('\nОтличники:')
if len(List1) != 0:
    for i in range(len(List1)):
        print(List1[i])
else:
    print('-')
print('\nХорошисты:')
if len(List2) != 0:
    for i in range(len(List2)):
        print(List2[i])
else:
    print('-')
new_class = input('\nВвести данные другого
класса?(+/-): ')

```

21 print("Для завершения игры укажите '-' вместо слова.\n")
 alph = ('к', 'е', 'н', 'т', 'а', 'в', 'р')
 print("Алфавит игры:", alph)
 print("(Буквы можно использовать неоднократно).\n")

```

name1 = input('Имя первого участника: ')
name2 = input('Имя второго участника: ')

```

```

basket1 = []
basket2 = []

```

```

i = 1
print()
time1 = name1 + ' (ход ' + str(i) + '): '
time2 = name2 + ' (ход ' + str(i) + '): '
word1 = input(time1)
word2 = input(time2)

```

```
while word1 != '-' and word2 != '-':
    print()
    i += 1
    if word1 != '-':
        check = 0
        for j in range(len(word1)):
            if word1[j] in alph:
                if (word1 in basket1) or (word1 in
                    basket2):
                    print('Слово "' + word1 + '" уже
                        было.\n')
                    break
                else:
                    check += 1
            else:
                print('Слово "' + word1 + '" не
                    удовлетворяет условиям игры.\n')

        if check == len(word1):
            basket1.append(word1)

    if word2 != '-':
        check = 0
        for j in range(len(word2)):
            if (word2[j] in alph) :
                if (word2 in basket1) or (word2 in
                    basket2):
                    print('Слово "' + word2 + '" уже
                        было.\n')
                    break
                else:
                    check += 1
            else:
                print('Слово "' + word2 + '" не
                    удовлетворяет условиям игры.')

        if check == len(word2):
            basket2.append(word2)

time1 = name1 + ' (ход ' + str(i) + '): '
time2 = name2 + ' (ход ' + str(i) + '): '
word1 = input(time1)
word2 = input(time2)
```

```

if word1 == '-' or word2 == '-':
    print('\nКорзина слов (' + name1 + '):', basket1)
    print('Корзина слов (' + name2 + '):', basket2)
    if len(basket1) > len(basket2):
        print(name1, ', поздравляем! Вы выиграли!')
    elif len(basket1) < len(basket2):
        print(name2, ', поздравляем! Вы выиграли!')
    else:
        print('Поздравляем! Ничья!')

```

```

basket1.clear()
basket2.clear()

```

22 print("Чтобы закончить ввод, в поле 'Введите слово:' укажите '-'.\n")

```

Dict = {}
list_amt = []

```

```

word = input('Введите слово: ')
amt = len(word)
list_amt.append(amt)
key = str(amt) + '-' + str(list_amt.count(amt))

```

```

while word != '-':
    Dict[key] = word
    print('Слову "' + word + '" присвоен ключ ' + key)
    word = input('\nВведите слово: ')
    amt = len(word)
    list_amt.append(amt)
    key = str(amt) + '-' + str(list_amt.count(amt))

```

```

if word == '-':
    list_keys = list(Dict.keys())
    list_keys.sort()
    print('\nСловарь:')
    for i in list_keys:
        print(i, ':', Dict[i])

```

```

print('\nОбщее количество слов: ', len(Dict))

```

Глава 8

| | | |
|---|---------------------------------|-----------|
| 1 | str = 'Сегодня хорошая погода!' | |
| | str[:7] | 'Сегодня' |
| | str[8:11] | 'хор' |
| | str[8:14] | 'хороша' |
| | str[8:-10] | 'хорош' |
| | str[-4:-1] | 'ода' |
| | str[:7:3] | 'Соя' |

| | | |
|---|--------------------------------|------------------|
| 2 | str = 'мама мыла раму' | |
| | str[:5] + str[-4:] + str[4:-5] | 'мама раму мыла' |
| | str[5:-4] + str[:5] + str[-4:] | 'мыла мама раму' |
| | str[-4:] + str[4:-4] + str[:4] | 'раму мыла мама' |

Возможные перестановки:

'мыла раму мама' и 'раму мама мыла'

3.1 2

3.2 256256

3.3 *w*r*y

3.4 T H E B E S T !!!

4.1 C

4.2 B

4.3 E

4.4 A

4.5 D

```
5 n = input('Введите шестизначное число: ')
while int(n[0]) == 0 or len(n) != 6:
    if int(n[0]) == 0 and len(n) != 6:
        print('Десятичное число не начинается
        с нуля')
        n = input('И пожалуйста, введите число
        с шестью разрядами: ')
    elif int(n[0]) == 0:
        print('Десятичное число не начинается
        с нуля')
```

```

n = input('Введите шестизначное число
снова: ')

elif len(n) != 6:
    n = input('Пожалуйста, введите число с шестью
разрядами: ')

if int(n[0]) != 0 and len(n) == 6:
    n1 = str(int(n[0]) + int(n[-1])) + str(int(n[1])
+ int(n[-2])) + str(int(n[2]) + int(n[-3]))

print(n1)
print('Количество разрядов нового числа: ', len(n1))

```

6

```

n = input('Введите пятизначное число: ')
while int(n[0]) == 0 or len(n) != 5:
    if int(n[0]) == 0 and len(n) != 5:
        print('Десятичное число не начинается
с нуля')
        n = input('И пожалуйста, введите число
с пятью разрядами: ')
    elif int(n[0]) == 0:
        print('Число не начинается с нуля')
        n = input('Введите пятизначное число снова: ')
    elif len(n) != 6:
        n = input('Пожалуйста, введите число с пятью
разрядами: ')

if int(n[0]) != 0 and len(n) == 5:
    print(n[-2] + n[-1] + n[2] + n[3] + n[4])

```

7

```

n = input('Введите натуральное число: ')
while int(n) < 1:
    print('Натуральное число - это целое число,
которое больше или равно 1.')
    n = input('Пожалуйста, введите натуральное число
снова: ')

```



```
if int(n) >= 1 and len(n) > 4:
    sum_n = 0
    for i in range(len(n)):
        sum_n += int(n[i])
    print(sum_n)

elif int(n) >= 1 and len(n) == 4:
    new_n = ''
    i = len(n)
    while i != 0:
        new_n += n[i-1]
        i -= 1
    print(new_n)

elif int(n) >= 1 and len(n) < 4:
    list_n = list(n)
    print(tuple(list_n))
```

- 8 a = 'Год 2038-й. Спустя 18 месяцев жизни и работы на поверхности Марса команда из шести исследователей садится обратно в космический аппарат и возвращается на Землю.'

```
print(a.upper())
print(a.lower())
```

9.1 a = 'ВЕЗДЕХОД'
print(a.replace('ВЕЗДЕ', 'МАРСО'))

9.2 a = 'МОЛОКО'
b = a.replace('К', 'В')
print(b[1:].lower())

9.3 a = 'революционер'
b = a.replace('ре', 'э')
b = b.replace('онер', 'я')
print(b)
c = a.replace('револю', 'коллек')
print(c.upper())

Глава 9

| | | |
|---|--|--|
| 1 | <pre>def tr_error(a,b,c): sum = a + b + c if sum != 180: print ('Error')</pre> | tr_error(a,b,c) — процедура |
| | <pre>def abc(a,b,c): return a+b+c def abc_new(a,b,c): print (2*abc(a,b,c))</pre> | abc(a,b,c) — функция abc_new(a,b,c) — процедура |
| | <pre>def sum(a,b): return a+b for i in range(5): i = i + sum(i,i) print(i)</pre> | sum(a,b) — функция |

| Команда | Результат |
|--------------------------|-----------------|
| tr_error(36,72,72) | - |
| tr_error(90,45,55) | Error |
| abc_new(3,7,14) | 48 |
| abc_new('tik','-','tak') | tik-tak tik-tak |
| sum(2,3) | 5 |
| sum('+','-') | +- |

3.1 5

3.2 230

3.3 None

```
4 def AB(n,m):
    h = []
    for i in range(n):
        h.append([])
        h[i] = list('.'*m)

    for i in range(n):
        if i%2 != 0:
            for j in range(m):
                if j%2 == 0:
                    h[i][j] = '*'
```

```
    else:
        for j in range(m):
            if j%2 != 0:
                h[i][j] = '*'

    for i in range(n):
        for j in range(m):
            print(h[i][j], end = ' ')
        print()

l = int(input('Введите количество строк: '))
s = int(input('Введите количество столбцов: '))
AB(l,s)
```

```
5 def max(l):
    max = 0
    r = 0
    for i in range(len(l)):
        for j in range(len(l[0])):
            if l[i][j] > max:
                max = l[i][j]

    for i in range(len(l)):
        for j in range(len(l[0])):
            if l[i][j] == max:
                r = '(' + str(i+1) + ';' + str(j+1) + ')'
                print('Максимальный элемент: ', max)
    return r
```

```
n = int(input('Количество строк матрицы: '))
m = int(input('Количество столбцов матрицы: '))
h = []
```

```
for i in range(n):
    h.append([])
    c = input()
    h[i] = c.split(' ')

for i in range(n):
    for j in range(m):
        h[i][j] = int(h[i][j])
```

```
print(max(h))
```

```

6 def Test(k,m):
    n = (k/m)*100
    if n < 50:
        return 2
    elif n >= 50 and n < 65:
        return 3
    elif n >= 65 and n < 85:
        return 4
    elif n >= 85 :
        return 5
mb = int(input('МАХ количество баллов за тест: '))
Dict_students = {}
z = '-'
while z != '-':
    f = input('\nФамилия И.О. учащегося: ')
    kb = int(input('Количество набранных баллов: '))
    Dict_students[f] = Test(kb,mb)
    print('Отметка:', Test(kb,mb))
    z = input('\nПродолжить ввод?(+/-): <')
    while z != '-' or z != '+':
        print('Неверный ввод')
        z = input('\nПродолжить ввод?(+/-): <')
List_students = list(Dict_students.keys())
List_students.sort()
print('\nИтог теста:')
for i in List_students:
    print(str(i) + ' - ' + str(Dict_students[i]))
7 def A(n):
    if n % 2 == 0:
        print('Число',n,'- четное')
    else:
        print('Число',n,'- нечетное')

a = int(input('Введите целое число: '))
A(a)
8 def A(n):
    if n > 0 and n < 3000:
        return 0
    elif n >= 3000 and n < 5000:
        return 5
    elif n >= 5000 and n < 10000:
        return 7
    elif n >= 10000:
        return 10

```

```
a = float(input('Сумма покупки (в руб.): '))
print('Ваша скидка составляет ' + str(A(a)) + '%')
print('Сумма к оплате', a - A(a)/100*a, 'руб.')
```

```
9 def Bonus(a,b):
    if a >= b:
        return 1
    else:
        return b-a
```

```
n = float(input('Количество бонусных рублей на
карте: '))
m = float(input('Стоимость заказа: '))
k = input('Списать бонусные рубли? (да/нет): ')
```

```
if k == 'да':
    print('Сумма к оплате: ', Bonus(n,m))
else:
    print('Сумма к оплате: ', m)
```

```
10 def min(l):
    min = l[0][0]
    r = 0
    for i in range(len(l)):
        for j in range(len(l[0])):
            if l[i][j] < min:
                min = l[i][j]

    for i in range(len(l)):
        for j in range(len(l[0])):
            if l[i][j] == min:
                r = '('+str(i)+';'+str(j)+')'
    print('Минимальный элемент: ', min)
    return r
```

```
def max(l):
    max = 0
    r = 0
    for i in range(len(l)):
        for j in range(len(l[0])):
            if l[i][j] > max:
                max = l[i][j]
```

```

for i in range(len(l)):
    for j in range(len(l[0])):
        if l[i][j] == max:
            r = '('+str(i)+';'+str(j)+')'
            print('Максимальный элемент: ', max)
            return r

def s(l):
    sum = 0
    r = 0
    for i in range(len(l)):
        for j in range(len(l[0])):
            sum += l[i][j]
            r += 1
    s = sum/r
    print('Среднее арифметическое: ', s)
    return s

n = int(input('Количество строк матрицы: '))
m = int(input('Количество столбцов матрицы: '))
h = []

for i in range(n):
    h.append([])
    c = input()
    h[i] = c.split(' ')

for i in range(n):
    for j in range(m):
        h[i][j] = int(h[i][j])

min(h)
max(h)
s(h)

```

Оглавление

| | |
|---|-----------|
| Введение | 3 |
| Программа и программирование | 3 |
| В чем ценность умения программировать? | 3 |
| | |
| Глава 1. Знакомство со средой программирования IDLE и первая программа | 5 |
| Установка среды IDLE | 5 |
| Интерфейс среды IDLE | 6 |
| Первая программа на языке Python | 7 |
| Оператор print() — вывод данных на экран | 7 |
| Эксперименты в интерактивной оболочке | 8 |
| Эксперименты с кавычками | 10 |
| Арифметические операторы | 13 |
| Разработка программ в IDLE | 16 |
| Первая многострочная программа | 19 |
| Ключи оптимизации | 20 |
| Что нового мы узнали в первой главе? | 24 |
| Практикум | 26 |
| Основные понятия главы | 26 |
| Чтение кода | 27 |
| Поиск ошибок | 28 |
| Оптимизация | 29 |
| Разработка | 30 |
| | |
| Глава 2. Переменные, типы данных. Функции input() и eval() | 33 |
| Понятие переменной | 33 |
| Оператор input() — ввод данных в программу | 37 |
| Ключи оптимизации | 41 |
| Функция преобразования eval() | 42 |
| Числовой тип данных | 44 |
| Результат арифметических операций с числами в Python | 43 |
| Логический тип данных | 45 |
| Строковый тип данных | 46 |

| | |
|---|-----------|
| Что нового мы узнали во второй главе? | 47 |
| Практикум | 48 |
| Основные понятия главы | 48 |
| Чтение кода | 49 |
| Поиск ошибок | 49 |
| Оптимизация | 50 |
| Разработка | 50 |
| | |
| Глава 3. Условия | 52 |
| Неполная форма ветвления: конструкция «если..., то...» | 52 |
| Полная форма ветвления: конструкция «если..., то..., иначе...» | 54 |
| Примеры неполной и полной форм ветвления | 55 |
| Условный оператор <code>elif</code> | 55 |
| Операторы <code>and</code> и <code>or</code> | 58 |
| Что нового мы узнали в третьей главе? | 59 |
| Практикум | 60 |
| Основные понятия главы | 60 |
| Чтение кода | 61 |
| Поиск ошибок | 63 |
| Оптимизация | 66 |
| Разработка | 66 |
| | |
| Глава 4. Циклы | 70 |
| Цикл с заданным числом повторений — <code>for</code> | 70 |
| Функция <code>range()</code> | 71 |
| Переменная-счетчик | 72 |
| Цикл с предусловием — <code>while</code> | 72 |
| Операторы <code>break</code> и <code>continue</code> | 74 |
| Вложенные циклы | 74 |
| Что нового мы узнали в четвертой главе? | 76 |
| Практикум | 77 |
| Основные понятия главы | 77 |
| Чтение кода | 78 |
| Поиск ошибок | 81 |
| Оптимизация | 82 |
| Разработка | 83 |

| | |
|--|------------|
| Глава 5. Псевдослучайные числа и математика | 85 |
| Подключение модулей и библиотек | 85 |
| Псевдослучайные числа | 85 |
| Модуль random | 86 |
| Игра «Угадай-ка!» | 88 |
| Модуль math | 90 |
| Использование псевдонимов, оператор as | 91 |
| Что нового мы узнали в пятой главе? | 92 |
| Практикум | 92 |
| Основные понятия главы | 92 |
| Чтение кода | 94 |
| Поиск ошибок | 94 |
| Оптимизация | 96 |
| Разработка | 97 |
| | |
| Глава 6. Исполнитель Черепашка | 100 |
| Команды перемещения и рисования Черепашки | 100 |
| Рисование элементарных фигур | 101 |
| Отрезок | 101 |
| Равносторонний треугольник | 102 |
| Квадрат | 103 |
| Окружность | 104 |
| Работа с цветом | 105 |
| Работа с полем | 107 |
| Реализация условных и циклических конструкций | 109 |
| Что нового мы узнали в шестой главе? | 113 |
| Практикум | 114 |
| Основные понятия главы | 114 |
| Чтение кода | 114 |
| Поиск ошибок | 115 |
| Оптимизация | 117 |
| Разработка | 118 |
| | |
| Глава 7. Массивы | 123 |
| Списки | 123 |
| Операции со списками | 125 |
| Конкатенация и дублирование | 125 |
| Добавление и удаление элементов | 126 |
| Очистка списка | 127 |

| | |
|---|------------|
| Еще несколько методов для списков | 127 |
| Сортировка числовых списков | 128 |
| Алгоритм сортировки «пузырьком» на Python | 128 |
| Метод split() и функция len() | 131 |
| Метод sort() | 133 |
| Подсчет количества элементов списка. Метод count() .. | 136 |
| Кортежи | 137 |
| Операции с кортежами | 138 |
| Срезы списков, строк и кортежей | 141 |
| Словари | 145 |
| Операции со словарями | 147 |
| Поиск по словарю | 150 |
| Сортировка словаря по ключам | 151 |
| Что нового мы узнали в седьмой главе? | 152 |
| Практикум | 155 |
| Основные понятия главы | 155 |
| Чтение кода | 156 |
| Поиск ошибок | 157 |
| Оптимизация | 159 |
| Разработка | 160 |
| | |
| Глава 8. Строки | 165 |
| Оператор преобразования в строку str() | 165 |
| Строка как массив | 166 |
| Обращение по индексу | 166 |
| Функция len() | 168 |
| Срезы строк | 168 |
| Методы find, replace и count | 169 |
| Что нового мы узнали в восьмой главе? | 170 |
| Практикум | 171 |
| Основные понятия главы | 171 |
| Чтение кода | 172 |
| Поиск ошибок | 172 |
| Оптимизация | 173 |
| Разработка | 174 |
| | |
| Глава 9. Функции | 176 |
| От программы к подпрограмме | 176 |
| Типы подпрограмм | 176 |

| | |
|--|------------|
| Определение функции в программе, оператор <code>def</code> | 177 |
| Передача параметров (аргументов) в функцию | 178 |
| Возврат значения из функции, оператор <code>return</code> | 179 |
| Стандартные функции Python | 181 |
| Ситуации, в которых целесообразно использование функций | 182 |
| Рекурсивные функции | 182 |
| Что нового мы узнали в девятой главе? | 185 |
| Практикум | 186 |
| Основные понятия главы | 186 |
| Чтение кода | 187 |
| Поиск ошибок | 187 |
| Оптимизация | 189 |
| Разработка | 190 |
| | |
| Заключение | 192 |
| Ответы | 193 |

Минимальные системные требования определяются соответствующими требованиями программ Adobe Reader версии не ниже 11-й либо Adobe Digital Editions версии не ниже 4.5 для платформ Windows, Mac OS, Android и iOS; экран 10"

Электронное издание для дополнительного образования

Серия: «Школа юного программиста»

Щерба Анастасия Владимировна

ПРОГРАММИРОВАНИЕ НА PYTHON®

Первые шаги

Для детей среднего и старшего школьного возраста

Ведущие редакторы *М. С. Стригунова, Т. В. Лаэрова*

Ведущий методист *А. А. Салахова*

Обложка: *И. К. Дилоян*

Технический редактор *Т. Ю. Федорова*

Корректор *И. Н. Панкова*

Компьютерная верстка: *О. Г. Лапко*

Подписано к использованию 11.01.22.

Формат 150×225 мм

Издательство «Лаборатория знаний»

125167, Москва, проезд Аэропорта, д. 3

Телефон: (499) 157-5272

e-mail: info@pilotLZ.ru, <http://www.pilotLZ.ru>



Программирование – это грамотность XXI века!

Книги новой серии «Школа юного программиста» издательства «Лаборатория знаний» построены на методике пошагового обучения программированию. Следуя этой методике, любой желающий, от школьника до студента вуза, сможет научиться писать программы, разрабатывать мобильные приложения и компьютерные игры и даже освоить технологии машинного обучения и нейросетей.

Издательство предлагает следующие учебные пособия:

- «Учимся вместе со Scratch: программирование, игры, робототехника» (5–6 классы)
- «Scratch 2.0: от новичка к продвинутому пользователю. Пособие для подготовки к Scratch-Олимпиаде» (1–11 классы)
- «Творческие задания в среде Scratch. Рабочая тетрадь для 5–6 классов»
- «Scratch 3.0: творческие проекты на вырост. Рабочая тетрадь для 7–8 классов»
- «Создаем игры с Kodu Game Lab» (4–5 классы)
- «Разработка мобильных приложений. Первые шаги» (8–11 классы)
- «Компьютерное зрение на PYTHON. Первые шаги» (4–9 классы)