

Оглавление

Об авторах	14
Предисловие	15
Python как язык полного спектра	15
Глава 1. Введение	21
1.1. Почему именно эта книга?	22
1.2. О Real Python	23
1.3. Как пользоваться книгой	24
1.4. Дополнительный материал и учебные ресурсы	25
Глава 2. Установка и настройка Python	27
2.1. О версиях Python	27
2.2. Windows	28
2.3. macOS	31
2.4. Linux	34
Глава 3. Первая программа Python	38
3.1. Написание программы Python	38
3.2. Ошибки	42
3.3. Создание переменной	44
3.4. Просмотр значений в интерактивном окне	48
3.5. Заметки на память	50
3.6. Итоги и дополнительные ресурсы	52

Глава 4. Строки и строковые методы 54

4.1. Что такое строка? 54

4.2. Конкатенация, индексирование и срезы 60

4.3. Манипуляции со строками с использованием методов 68

4.4. Взаимодействие с пользовательским вводом 73

4.5. Задача: разбор пользовательского ввода 75

4.6. Работа со строками и числами 76

4.7. Упрощение команд вывода 80

4.8. Поиск подстроки в строке 82

4.9. Задача: преобразование текста 84

4.10. Итоги и дополнительные ресурсы 85

Глава 5. Числа и математические вычисления 87

5.1. Целые числа и числа с плавающей точкой 87

5.2. Арифметические операторы и выражения 91

5.3. Задача: выполнение вычислений с пользовательским вводом 98

5.4. Когда Python говорит неправду 98

5.5. Математические функции и числовые методы 100

5.6. Оформление чисел при выводе 104

5.7. Комплексные числа 107

5.8. Итоги и дополнительные ресурсы 109

Глава 6. Функции и циклы 111

6.1. Что же такое функция? 111

6.2. Написание ваших собственных функций 115

6.3. Задача: конвертер температур 122

6.4. Циклическое выполнение 123

6.5. Задача: отслеживание прибыли по вкладу 130

6.6. Область видимости в Python	131
6.7. Итоги и дополнительные ресурсы	136
Глава 7. Поиск и исправление ошибок в коде	137
7.1. Использование окна Debug Control	137
7.2. Исправление ошибок	143
7.3. Итоги и дополнительные ресурсы	149
Глава 8. Условная логика и управление программой.....	151
8.1. Сравнение значений.....	151
8.2. Добавим немного логики.....	154
8.3. Управление последовательностью выполнения программы	161
8.4. Задача: поиск множителей числа	170
8.5. Управление циклами.....	171
8.6. Восстановление после ошибок	174
8.7. Моделирование событий и вычисление вероятностей.....	179
8.8. Задача: моделирование эксперимента с броском монеты.....	183
8.9. Задача: моделирование выборов.....	184
8.10. Итоги и дополнительные ресурсы	184
Глава 9. Кортежи, списки и словари	186
9.1. Кортежи как неизменяемые последовательности	186
9.2. Списки: изменяемые последовательности	195
9.3. Вложение, копирование и сортировка кортежей и списков	206
9.4. Задача: список списков	210
9.5. Задача: доступ вдохновения	212
9.6. Храните отношения в словарях	213
9.7. Задача: цикл по столицам.....	222
9.8. Как выбрать структуру данных.....	223

9.9. Задача: коты в шляпах	223
9.10. Итоги и дополнительные ресурсы	224
Глава 10. Объектно-ориентированное программирование (ООП)	226
10.1. Определение класса	226
10.2. Создание экземпляров (инстанцирование).....	230
10.3. Наследование от других классов.....	235
10.4. Задача: модель фермы	242
10.5. Итоги и дополнительные ресурсы	243
Глава 11. Модули и пакеты	244
11.1. Работа с модулями.....	244
11.2. Работа с пакетами	253
11.3. Итоги и дополнительные ресурсы	260
Глава 12. Операции ввода и вывода с файлами.....	261
12.1. Файлы и файловая система.....	261
12.2. Работа с путями к файлам в Python.....	265
12.3. Основные операции файловой системы.....	272
12.4. Задача: перемещение всех графических файлов в новый каталог	285
12.5. Чтение и запись файлов	286
12.6. Чтение и запись данных CSV	298
12.7. Задача: создание списка рекордов	307
12.8. Итоги и дополнительные ресурсы	307
Глава 13. Установка пакетов с помощью pip	309
13.1. Установка сторонних пакетов с помощью pip.....	309
13.2. Подводные камни сторонних пакетов	318
13.3. Итоги и дополнительные ресурсы	320

Глава 14. Создание и изменение файлов PDF	321
14.1. Извлечение текста из файла PDF	321
14.2. Извлечение страниц из файлов PDF	327
14.3. Задача: класс PdfFileSplitter	332
14.4. Конкатенация и слияние файлов PDF	333
14.5. Поворот и обрезка страниц PDF	339
14.6. Шифрование и дешифрование файлов PDF	348
14.7. Задача: восстановление порядка страниц	351
14.8. Создание файла PDF с нуля	352
14.9. Итоги и дополнительные ресурсы	357
Глава 15. Базы данных	359
15.1. Знакомство с SQLite	359
15.2. Библиотеки для работы с другими базами данных SQL	369
15.3. Итоги и дополнительные ресурсы	370
Глава 16. Веб-программирование	372
16.1. Скрапинг и парсинг текста с веб-сайтов	372
16.2. Использование парсера HTML для извлечения веб-данных	380
16.3. Работа с HTML-формами	385
16.4. Взаимодействие с веб-сайтами в реальном времени	390
16.5. Итоги и дополнительные ресурсы	393
Глава 17. Научные вычисления и построение графиков	395
17.1. Использование NumPy для матричных вычислений	395
17.2. Построение графиков с помощью Matplotlib	404
17.3. Итоги и дополнительные ресурсы	422
Глава 18. Графические интерфейсы	423
18.1. Добавление элементов GUI с помощью EasyGUI	423
18.2. Пример: программа для поворота страниц PDF	434

18.3. Задача: приложение для извлечения страницы PDF.....	439
18.4. Знакомство с Tkinter	440
18.5. Работа с виджетами.....	443
18.6. Управление макетом при помощи менеджеров геометрии	463
18.7. Интерактивность в приложениях	478
18.8. Пример приложения: конвертер температур.....	485
18.9. Пример приложения: текстовый редактор	489
18.10. Задача: возвращение поэта	496
18.11. Итоги и дополнительные ресурсы.....	498
Глава 19. Мысли напоследок и следующие шаги	500
19.1. Еженедельные бесплатные советы для питонистов	501
19.2. Книга «Чистый Python».....	501
19.3. Библиотека видеокурсов Real Python	502
19.4. Благодарности.....	503

ГЛАВА 6

Функции и циклы

Функции являются строительными блоками практически любой программы Python. Обычно именно в них происходят основные события!

Вы уже видели, как использовать некоторые функции, например `print()`, `len()` и `round()`. Все эти функции называются **встроенными**, потому что они реализованы непосредственно в языке Python. Вы также можете создавать **пользовательские** функции для решения конкретных задач.

Функции разбивают код на меньшие блоки. Их используют для определения действий, которые должны неоднократно выполняться в программном коде. Вам не придется писать один и тот же код каждый раз, когда программе потребуется выполнить эту задачу, достаточно вызвать функцию!

Но иногда некоторую часть кода требуется повторить несколько раз подряд. Для этого используются **циклы**.

В этой главе вы:

- создадите пользовательские функции;
- научитесь работать с циклами `for` и `while`;
- узнаете, что такое область видимости и почему она важна.

Итак, за дело!

6.1. ЧТО ЖЕ ТАКОЕ ФУНКЦИЯ?

В предыдущих главах мы использовали функции `print()` и `len()` для вывода текста и определения длины строки. Давайте разберемся с функциями более подробно.

В этом разделе на примере `len()` я покажу, что такое функция и как она выполняется.

Функции как значения

Одна из самых важных особенностей функций в языке Python состоит в том, что функции — это значения, которые могут присваиваться переменным.

Проверим имя `len` в интерактивном окне IDLE. Для этого введите его после приглашения:

```
>>> len
<built-in function len>
```

Python сообщает, что `len` является встроенной функцией. По аналогии с тем, как целочисленные значения имеют тип `int`, а строки — тип `str`, значения-функции также имеют тип:

```
>>> type(len)
<class 'builtin_function_or_method'>
```

Однако при желании с именем `len` можно связать другое значение:

```
>>> len = "I'm not the len you're looking for."
>>> len
"I'm not the len you're looking for."
```

Теперь имя `len` связано со строковым значением. Вы можете убедиться в том, что оно относится к типу `str`, при помощи функции `type()`:

```
>>> type(len)
<class 'str'>
```

И хотя значение, связанное с именем `len`, можно изменить, делать так обычно не рекомендуется. Изменение значения `len` только усложнит чтение вашего кода, потому что новое имя `len` легко перепутать со встроенной функцией. Сказанное относится к любой встроенной функции.

ВАЖНО!

После выполнения этих примеров кода встроенная функция `len` станет недоступной в IDLE. Чтобы вернуть ее, введите команду:

```
>>> del len
```

Ключевое слово `del` используется для отмены присваивания переменной. Это сокращение от `delete` (удалить), но значение не удаляется. Вместо этого связь имени со значением разрывается и удаляется имя.

Обычно после выполнения `del` при попытке использования имени удаленной переменной происходит ошибка `NameError`. Однако в данном случае имя `len` не удаляется:

```
>>> len
<built-in function len>
```

Так как `len` является именем встроенной функции, оно снова связывается с исходным значением-функцией.

Какой же вывод из этого можно сделать? У функций есть имена, но эти имена не имеют жесткой связи с функцией, и им можно присваивать другие значения.

Когда вы пишете собственные функции, будьте внимательны и не присваивайте им значения, используемые встроенными функциями.

Как Python работает с функциями

Давайте более детально разберемся в том, как Python выполняет функции.

Прежде всего следует заметить, что функцию невозможно выполнить, просто указав ее имя. Необходимо **вызвать** функцию, чтобы сообщить Python, как она должна выполняться.

Посмотрим, как работает механизм вызова на примере `len()`:

```
>>> # Печать имени функции не приводит к ее исполнению.
>>> # IDLE проверяет переменную как обычно.
>>> len
<built-in function len>

>>> # Используем скобки для вызова функции.
>>> len()
Traceback (most recent call last):
  File "<pyshell#3>", line 1, in <module>
    len()
TypeError: len() takes exactly one argument (0 given)
```

В этом примере Python выдает ошибку `TypeError` при вызове `len()`, потому что функция `len()` ожидает получить аргумент.

Аргумент представляет собой значение, которое передается функции. Некоторые функции вызываются без аргументов, другие могут получать сколько угодно аргументов. Функция `len()` должна получать только один аргумент. Завершив выполнение, функция **возвращает** значение. Возвращаемое значение обычно — хотя и не всегда — зависит от значений других аргументов, передаваемых функции.

Функция выполняется в три этапа.

1. Функция **вызывается**, и все аргументы передаются ей в качестве входных значений.
2. Функция **выполняется**, и с ее аргументами выполняются некоторые действия.
3. Функция **возвращает** управление, а исходный вызов функции заменяется возвращаемым значением.

Давайте рассмотрим конкретный пример и разберемся, как Python выполняет следующую строку кода:

```
>>> num_letters = len("four")
```

Сначала `len()` вызывается с аргументом `"four"`. Вычисляется длина строки `"four"`, которая равна 4. Затем `len()` возвращает число 4 и заменяет вызов функции полученным значением.

После выполнения функции строка кода будет выглядеть так:

```
>>> num_letters = 4
```

Затем Python присваивает значение 4 переменной `num_letters` и продолжает выполнение остальных строк кода в программе.

Функции могут обладать побочными эффектами

Вы узнали, как вызывать функции и что функции возвращают значение при завершении выполнения. Тем не менее иногда функции не ограничиваются простым возвращением значения.

Если функция изменяет что-то в программе за пределами своего собственного кода, говорят, что она имеет **побочный эффект**. Вы уже видели одну функцию с побочным эффектом: `print()`.

Когда вы вызываете `print()` со строковым аргументом, строка выводится в оболочке Python в текстовом виде. Однако `print()` не возвращает текстового значения.

Чтобы понять, что возвращает `print()`, присвойте возвращаемое значение `print()` переменной:

```
>>> return_value = print("What do I return?")
What do I return?
>>> return_value
>>>
```

Когда вы присваиваете `print("What do I return?")` переменной `return_value`, выводится строка "What do I return?". Но при проверке значения `return_value` ничего не выводится.

Функция `print()` возвращает специальное значение `None`, которое указывает на отсутствие данных; `None` относится к типу, который называется `NoneType`:

```
>>> type(return_value)
<class 'NoneType'>
>>> print(return_value)
None
```

При вызове `print()` выводимый текст не является возвращаемым значением, это побочный эффект `print()`.

6.2. НАПИСАНИЕ ВАШИХ СОБСТВЕННЫХ ФУНКЦИЙ

При написании более сложных программ может оказаться, что вам нужно многократно выполнять несколько строк кода — например, вычислять одну и ту же формулу для разных входных значений.

Возможно, у вас появится искушение скопировать код в другие части программы и изменить его по мере надобности, но так поступать не стоит! Если вы обнаружите ошибку в скопированном коде, исправление придется вносить во все копии. А это огромная работа!

В этом разделе вы научитесь создавать собственные функции, чтобы избежать дублирования кода при его многократном использовании.

Анатомия функции

Каждая функция состоит из двух частей.

1. Сигнатура функции определяет имя функции и все входные данные, которые она ожидает получить.
2. Тело функции содержит код, который выполняется при каждом использовании функции.

Напишем функцию, которая получает два числа на входе и возвращает их произведение. Вот как может выглядеть функция (сигнатура и тело обозначены в комментариях):

```
def multiply(x, y): # Сигнатура функции
    # Тело функции
    product = x * y
    return product
```

Создание функции для чего-то настолько простого, как оператор `*`, выглядит странно. Пожалуй, `multiply()` — не та функция, которую вы будете использовать в реальной программе. Но это хороший первый пример, чтобы понять, как создаются функции!

ВАЖНО!

Когда вы определяете функцию в интерактивном окне IDLE, необходимо нажать `Enter` дважды после строки, содержащей команду `return`, чтобы функция была зарегистрирована Python:

```
>>> def multiply(x, y):
...     product = x * y
...     return product
...     # <--- Здесь нужно нажать Enter во второй раз.
>>>
```

Разобьем функцию на части и посмотрим, что в какой момент происходит.

Сигнатура функции

Первая строка кода функции называется **сигнатурой функции**. Она всегда начинается с ключевого слова `def` (сокращение от `define` — определить).

Рассмотрим повнимательнее сигнатуру `multiply()`:

```
def multiply(x, y):
```

Сигнатура функции состоит из четырех частей.

1. Ключевое слово `def`.
2. Имя функции `multiply`.
3. Список параметров `(x, y)`
4. Двоеточие `(:)` в конце строки.

Когда Python читает строку, начинающуюся с ключевого слова `def`, он создает новую функцию. Эта функция присваивается переменной, имя которой совпадает с именем функции.