Python учебник

Выпуск 3.8.8

Гвидо ван Россум
и команда разработчиков Python

августа 12, 2021

https://Digitology.tech
Email: tweakit@bk.ru

Оглавление

Python — простой в изучении и при этом мощный язык программирования. Он содержит эффективные высокоуровневые структуры данных и простой, но эффективный подход к объектно-ориентированному программированию. Элегантный синтаксис и динамическая типизация Python вместе с его интерпретируемой природой делают его идеальным языком для написания сценариев и быстрой разработки приложений во многих областях на большинстве платформ.

Интерпретатор Python и его обширная стандартная библиотека свободно доступны в исходных кодах или двоичной форме для всех основных платформ с веб-сайта Python https://www.python.org/, и могут свободно распространяться. Тот же сайт также содержит дистрибутивы и ссылки на многие свободные сторонние модули Python, программы и инструменты, а также дополнительная документация.

Интерпретатор Python может быть легко расширен с помощью новых функций и типов данных, написанных на C/C++ (или других языков, к которым можно получить доступ из C). Также Python можно применять как язык расширений для настраиваемых приложений.

Этот учебник неформально представляет читателю основные концепции и возможности языка и системы Python. Полезно держать интерпретатор Python под рукой для получения практического опыта, но при этом все примеры самодостаточны, так что учебник вполне возможно читать вне сети.

Описание стандартных объектов и модулей см. в разделе library-index. reference-index даёт более формальное определение языка. Для написания расширений на C или C++, ознакомьтесь с extending-index и c-api-index. Также существует несколько книг, освещающих Python более подробно.

Этот учебник не пытается быть всеобъемлющим и охватить каждую функцию в отдельности или даже все часто используемые функции. Вместо этого он знакомит со многими наиболее примечательными функциями Python и дает вам хорошее представление о вкусе и стиле языка. После прочтения учебника вы сможете писать и читать программы и модули, написанные на Python, и будете готовы узнать больше о различных модулях библиотеки Python, описанных в library-index.

Стоит также познакомиться и с *Глоссарий*.

ГЛАВА 1

Разжигая ваш аппетит

Если вы много работаете за компьютером, то появится потребность в автоматизации какой-либо задачи. Например, может потребоваться выполнить поиск и замену по большому количеству текстовых файлов или переименование и перегруппирования кучи фотографий сложным образом. Возможно, вы хотели бы написать маленькую пользовательскую базу данных, специализированное приложение с графическим интерфейсом или простую игру.

Если вы профессиональный разработчик программных продуктов — вероятно, вы привыкли работать с несколькими библиотеками на C/C++/Java, но находите обычный цикл написания/компиляции/тестирования/перекомпиляции кода чересчур медленным. Возможно, вы пишете набор тестов для такой библиотеки и процесс написания тестирующего кода воспринимается утомительным. Или же вы написали программу, которая должна использовать специальный язык для расширений и не хотите проектировать и разрабатывать полностью новый язык для вашего приложения.

Python — язык для тебя.

Вы можете написать шелл-сценарий для Unix или использовать пакетные файлы Windows для некоторых из этих задач, но шелл-сценарии хороши лишь для перемещения файлов и замены текстовых данных — они вряд ли подойдут для написания приложений, снабженных графическим интерфейсом, или игр. Вы можете написать программу на C/C++/Java, но разработка может занять довольно много времени — даже на то, чтобы получить первый рабочий набросок, его требуется немало. Python — проще в использовании, доступен на операционных системах Windows, Mac OS X и Unix, и поможет сделать работу намного быстрее.

Даже учитывая лёгкость использования, Python — полноценный язык программирования, предлагающий много больше возможностей для структурирования и поддержки крупных программ, чем могут позволить себе шелл-сценарии или пакетные файлы. С другой стороны, Python также предоставляет намного больше информации для отладки ошибок чем C и, будучи *сверхвысокоуровневым языком*, содержит встроенные высокоуровневые типы данных — такие, как гибкие массивы и словари. Благодаря наличию обобщённых типов данных Python применим для более широкого круга приложений чем Awk или даже Perl, и при этом очень многие вещи остаются в языке Python как минимум настолько же простыми, насколько просты в этих языках.

Python позволяет вам разделить вашу программу на модули, которые могут повторно использоваться в других программах на Python. Он поставляется вместе с внушительной коллекцией стандартных модулей, которые вы можете использовать в качестве фундамента ваших программ; или в качестве примеров

для того, чтобы начать изучение Python. Многие из этих модулей предоставляют различную полезную функциональность: например, ввод-вывод для файлов, системные вызовы, сокеты, и даже инструменты для создания графического пользовательского интерфейса — такие, как Tk.

Python — интерпретируемый язык, который позволяет сэкономить значительное время на разработку программ, поскольку нет необходимости в компиляции и линковки. Интерпретатор можно использовать в интерактивном режиме, что позволяет легко экспериментировать с возможностями языка и написания одноразовых программ или для тестирования функций при разработке программы снизу вверх. Он также удобен как настольный калькулятор.

Python позволяет создавать компактные и удобочитаемые программы. Текст программы на Python обычно намного короче, чем эквивалентные программы на C, C++ или Java, по нескольким причинам:

- Типы данных высокого уровня позволяют выражать сложные операции одним оператором;
- Блочный оператор реализуется с помощью отступов вместо открывающихся и закрывающихся скобок;
- Объявления типов переменных или аргументов не требуются.

Python *расширяем*: если вы знаете, как программировать на C, то вам будет легко добавить к интерпретатору новую встроенную функцию или модуль, выполнить критические операции на максимальной скорости или связать программы на Python с библиотеками, которые могут быть доступны только в бинарной форме (например, зависящие от поставщика графические библиотеки). Если вы действительно увлечены — вы можете привязать интерпретатор Python к приложению, написанному на C — чтобы использовать его как язык расширений или командный язык этого приложения.

Кстати, язык назван в честь шоу на BBC «Летающий цирк Монти Пайтон» и не имеет никакого отношения к рептилиям. Ссылки на скетчи Монти Пайтон в документации не только разрешены, но и поощряются!

Теперь, когда вы все в восторге от Python, вы захотите изучить его более подробно. Поскольку лучший способ выучить язык — это писать на нём. Учебник предлагает вам поиграть с интерпретатором Python во время чтения.

В следующей главе поясняется механика использования интерпретатора. Она является довольно поверхностной информацией, но имеет важное значение для проверки приведенных примеров позже.

В остальной части учебника рассматриваются различные особенности языка Python и примеры, начиная с простых выражений, операторов и типов данных, функций и модулей, и, наконец, коснемся таких понятий, как исключения и определяемые пользователем классы.

ГЛАВА 2

Использование интерпретатора Python

2.1 Запуск интерпретатора

Интерпретатор Python после установки располагается, обычно, по пути `/usr/local/bin/python3.8` — на тех компьютерах, где этот путь доступен. Добавление каталога `/usr/local/bin` к пути поиска Unix-шелла позволит запустить интерпретатор набором команды

```
python3.8
```

прямо из шелла.[1] Поскольку выбор каталога, в котором будет обитать интерпретатор, осуществляется при его установке, то возможны и другие варианты — посоветуйтесь с вашим Python-гуру или системным администратором. (Например, путь `/usr/local/python` тоже популярен в качестве альтернативного расположения.)

На компьютерах с Windows, на которых установлен Python из Microsoft Store будет доступна команда: `python3.8`. Если у вас установлен py.exe запускальщик, вы можете использовать `py` команду. Другие способы запуска setting-envvars см. в разделе Python.

При наборе символа конца файла (`Control-D` в Unix, `Control-Z` Windows) в ответ на основное приглашение интерпретатора, последний будет вынужден закончить работу с нулевым статусом выхода. Если это не сработает — вы можете выйти из интерпретатора путём ввода следующей команды: `quit()`.

Функции редактирования строк интерпретатора включают интерактивное редактирование, замену истории и завершение кода в системах, поддерживающих библиотеку GNU Readline.Самый быстрый, наверное, способ проверить, поддерживается ли расширенное редактирование командной строки, заключается в нажатии `Control-P` в ответ на первое полученное приглашение Python. Если вы услышите звуковой сигнал, значит вам доступно редактирование командной строки; Введение в ключи см. в Приложении *Интерактивное редактирование ввода и подстановка из истории*. Если на ваш взгляд ничего не произошло или отобразился символ `^P` — редактирование командной строки недоступно — удалять символы из текущей строки возможно будет лишь использованием клавиши Backspace.

Интерпретатор ведёт себя сходно шеллу Unix: если он вызван, когда стандартный ввод привязан к устройству tty — он считывает и выполняет команды в режиме диалога; будучи вызванным с именем

[1] В опереционных системах семейства Unix, интерпретатор Python 3.x по-умолчанию не запускается по имени `python`, чтобы он не конфликтовал с уже установленным Python 2.x.

файла в качестве параметра или с файлом, назначенным на стандартный ввод — он читает и выполняет *сценарий* из этого файла.

Другой способ запустить интерпретатор — `python -c command [arg] ...`, — при её использовании поочередно выполняются операторы(-ор) из *command* (как при использовании опции `-c` Unix-шелла). В связи с тем, что операторы Python часто содержат пробелы или другие специальные для шелла символы, рекомендуется полностью заключать *command* в одинарные кавычки.

Некоторые модули Python оказываются полезными при использовании их в качестве сценариев. Они могут быть запущены в виде командой `python -m module [arg] ...`, — таким образом исполняется исходный файл модуля *module* (как произошло бы, если бы вы ввели его полное имя в командной строке).

При использовании файла сценария иногда полезно иметь возможность запустить сценарий и затем войти в интерактивный режим. Это может быть сделано через указание параметра `-i` перед именем сценария.

Все опции командной строки описаны в using-on-general.

2.1.1 Передача параметров

В случае, если интерпретатору известны имя сценария и дополнительные параметры, с которыми он вызван, все они передаются сценарию в переменной `argv` модуля `sys`, представляющей собой список строк. Вы можете получить доступ к этому списку, выполнив `import sys`. Длина списка — минимум, единица; если не переданы ни имя сценария, ни аргументы — то `sys.argv[0]` содержит пустую строку. Когда в качестве имени сценария передан `'-'` (означает стандартный ввод), `sys.argv[0]` устанавливается в `'-'`. Если используется директива `-c` *command*, то `sys.argv[0]` устанавливается как `'-c'`. Когда используется `-m` *module* , то `sys.argv[0]` устанавливается равным полному имени модуля по расположению. Опции, обнаруженные после сочетаний `-c` *command* или `-m` *module* не обрабатываются интерпретатором Python, но остаются в переменной `sys.argv`, чтобы обеспечить возможность отслеживания в самой команде или в модуле.

2.1.2 Интерактивный режим

Когда команды считываются из tty, интерпретатор находится в *интерактивном режиме*. В этом режиме запрашивается следующая команда с *основного приглашения*, обычно три знака больше (`>>>`); в то же время, для продолжающих строк выводится *вспомогательное приглашение*, по умолчанию три точки (`...`). Перед выводом первого приглашения интерпретатор отображает приветственное сообщение, содержащее номер его версии и пометку о правах копирования:

```
$ python3.8
Python 3.8 (default, Sep 16 2015, 09:25:04)
[GCC 4.8.2] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>>
```

Продолжающие строки используются в случаях, когда необходимо ввести многострочную конструкцию. Взгляните, например, на следующий оператор `if`:

```
>>> the_world_is_flat = True
>>> if the_world_is_flat:
...     print("Be careful not to fall off!")
...
Be careful not to fall off!
```

Подробнее об интерактивном режиме смотрите *Интерактивный режим*.

2.2 Интерпретатор и его окружение

2.2.1 Кодировка исходных файлов

По умолчанию, исходники Python считаются созданными в кодировке UTF-8. В этой кодировке в строковых литералах, идентификаторах и комментариях могут быть использованы символы большинства языков мира — хотя стандартная библиотека Python использует только символы ASCII для именования идентификаторов — и этому соглашению должен следовать любой переносимый код. Для корректного отображения всех этих символов, ваш редактор должен опознавать файл как закодированный в UTF-8 и должен использовать шрифт, который содержит все символы, используемые в файле.

Для объявления кодировки, отличной от кодировки по умолчанию, используется специальная строка комментария которую следует добавить в качестве *первой* строки файла. Синтаксис выглядит следующим образом:

```
# -- coding: encoding --
```

Где *encoding* является одной из допустимых в `codecs`, поддерживаемых Python.

Например, если ваш текстовый редактор не поддерживает кодировку файлов UTF-8 и настаивает на какой-либо другой кодировке, скажем, Windows-1252, можно написать:

```
# -- coding: cp1252 --
```

Исключение из правила если *первая строка* в исходном коде начинается с *UNIX «shebang» строки*. В этом случае объявление кодировки должно быть добавлено в качестве второй строки файла. Например:

```
#!/usr/bin/env python3
# -*- coding: cp1252 -*-
```

ГЛАВА 3

Неофициальное знакомство в Python

В приведенных далее примерах, ввод и вывод различаются присутствием и отсутствием приглашений соответственно (приглашениями являются (>>> и ...): чтобы воспроизвести пример — вам нужно ввести всё, что следует за приглашением, после его появления; строки, не начинающиеся с приглашений являются выводом интерпретатора. Обратите внимание, что строка, в которой содержится лишь вспомогательное приглашение («...») означает, что вам нужно ввести пустую строку — этот способ используется для завершения многострочных команд.

Большинство примеров в этом руководстве, даже те, которые вводятся в интерактивном режиме — содержат комментарии. Комментарии в Python начинаются с символа решетки # и продолжаются до физического конца строки. Комментарии могут находиться как в начале строки, так и следовать за пробельными символами или кодом, но не содержаться внутри строки. Символ решётки в строке остаётся лишь символом решётки. Поскольку комментарии предназначены для того, чтобы сделать код более понятным, и не интерпретируются Python — при вводе примеров они могут быть пропущены.

Примеры:

```
# первый комментарий
spam = 1  # и здесь второй комментарий
          # ... здесь третий!
text = "# Это не комментарий, потому что внутри кавычек."
```

3.1 Использование Python в качестве калькулятора

Давайте опробуем несколько простых команд Python. Запустите интерпретатор и дождитесь появления основного приглашения — >>>. (Это не должно занять много времени).

3.1.1 Числа

Поведение интерпретатора сходно поведению калькулятора: вы вводите выражение, а в ответ он выводит значение. Синтаксис выражений привычен: операции +, -, `` * `` и / работают также как и в большинстве других языков (например, Pascal или C); для группировки можно использовать скобки (( )). Например:

```
>>> 2 + 2
4
>>> 50 - 5*6
20
>>> (50 - 5*6) / 4
5.0
>>> 8 / 5  # деление всегда возвращает число с плавающей точкой
1.6
```

Целые числа (например, `2`, `4`, `20`) имеют тип `int`, числа с дробной частью (например, `5.0`, `1.6`) имеют тип `float`. Дополнительные сведения о числовых типах см. ниже в учебном пособии.

Функция деления (`/`) всегда возвращает float значение. Выполнение *целочисленного деления* возвращает целочисленный результат (отбрасывая любой дробный результат), для этого вы можете использовать оператор `//`; чтобы посчитать остаток от деления, вы можете использовать ``%`:

```
>>> 17 / 3  # классическое деление возвращающее число с плавающей точкой
5.666666666666667
>>>
>>> 17 // 3  # целочисленное деление отбрасывающее дробную часть
5
>>> 17 % 3  # оператор % возвращает остаток от деления
2
>>> 5 * 3 + 2  # result * divisor + remainder
17
```

С помощью Python можно использовать оператор `**` для возведения в степень[1]:

```
>>> 5 ** 2  # квадрат 5
25
>>> 2 ** 7  # 2 в степени 7
128
```

Знак равенства (`=`) используется для присвоения значения переменной. После этого действия в интерактивном режиме ничего не выводится

```
>>> width = 20
>>> height = 5 * 9
>>> width * height
900
```

Если переменная не «определена» (ей не присвоено значение), то попытка использовать ее выдаст ошибку:

```
>>> n  # попытка доступа к неопределенной переменной
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
NameError: name 'n' is not defined
```

Присутствует полная поддержка операций с плавающей точкой; операции над операндами смешанного типа конвертируют целочисленный операнд в число с плавающей запятой:

[1] Поскольку `**` имеет более высокий приоритет, чем `-`, `-3**2` интерпретируется как `-(3**2)` и, таким образом, приводит к `-9`. Чтобы избежать этого и получить `9`, можно использовать `(-3)**2`.

```
>>> 4 * 3.75 - 1
14.0
```

В интерактивном режиме последнее напечатанное выражение присваивается переменной _. Это означает, что при использовании Python в качестве настольного калькулятора немного проще продолжать расчеты, например:

```
>>> tax = 12.5 / 100
>>> price = 100.50
>>> price * tax
12.5625
>>> price + _
113.0625
>>> round(_, 2)
113.06
```

Эту переменную следует использовать только для чтения. Не присваивайте явно ей значение — вы создадите независимую локальную переменную с тем же именем, скрыв встроенную переменную с ее магическим поведением.

В дополнение к `int` и `float`, Python поддерживает другие типы чисел, такие как `Decimal` и `Fraction`. Python также имеет встроенную поддержку комплексных чисел, и использует суффикс `j` или `J` для обозначения мнимой части (например, `3+5j`).

3.1.2 Строки

Кроме чисел, Python также может работать со строками, которые могут быть описаны несколькими способами. Они могут быть заключены в одиночные кавычки (`'...'`) или двойные кавычки (`"..."`) с тем же[2] результатом. \ может использоваться для экранирования кавычек:

```
>>> 'spam eggs'  # одиночные кавычки
'spam eggs'
>>> 'doesn\'t'  #  используется \' для экранирования одиночной кавычки...
"doesn't"
>>> "doesn't"  # ... или использовать двойные кавычки
"doesn't"
>>> '"Yes," they said.'
'"Yes," they said.'
>>> "\"Yes,\" they said."
'"Yes," they said.'
>>> '"Isn\'t," they said.'
'"Isn\'t," they said.'
```

При интерактивном выполнении вывод строк заключается в кавычки и спец. символы экранируются обратными слэшами. Несмотря на то, что иногда это может выглядеть отлично от ввода (окаймляющие кавычки могут измениться), обе строки одинаковы. Строка заключается в двойные кавычки, если строка содержит одинарные кавычки, а не двойные кавычки, иначе она заключается в одинарные кавычки. Функция `print()` производит более удобочитаемый вывод, опуская окаймляющие кавычки и печатая экранируемые и специальные символы:

[2] В отличие от других языков, специальные символы, такие как `\n`, имеют одинаковое значение с одинарными (`'...'`) и двойными (`"..."`) кавычками. Единственное различие между ними состоит в том, что в пределах одиночных кавычек нет необходимости экранирования `"` (но необходимо экранировать `\'`) и наоборот.

```
>>> '"Isn\'t," they said.'
'"Isn\'t," they said.'
>>> print('"Isn\'t," they said.')
"Isn't," they said.
>>> s = 'First line.\nSecond line.'  # \n означает перевод строки
>>> s  # без print(), \n  вывелся в вывод
'First line.\nSecond line.'
>>> print(s)  # с print(), \n создает новую строку
First line.
Second line.
```

Если вы не хотите, чтобы символы, предшествующие \, интерпретировались как специальные символы, можно использовать *сырые строки*, добавив `r` перед первой кавычкой:

```
>>> print('C:\some\name')  # здесь \n создает новую строку!
C:\some
ame
>>> print(r'C:\some\name')  # замечание, r перед кавычкой
C:\some\name
```

Строковые литералы могут охватывать несколько строк. Одним из способов является использование тройных кавычек: `"""..."""` или `'''...'''`. Конец строки автоматически включается в строку, но это можно предотвратить, добавив \ в конце строки. Пример:

```
print("""\
Usage: thingy [OPTIONS]
     -h                        Display this usage message
     -H hostname               Hostname to connect to
""")
```

Вернёт следующий вывод (заметьте, что начальный перевод строки не включен):

```
Usage: thingy [OPTIONS]
     -h                        Display this usage message
     -H hostname               Hostname to connect to
```

Строки могут быть конкатенированы (склеены) оператором `+` и повторены с `*`:

```
>>> # 3 повторения 'un', следующих за 'ium'
>>> 3 * 'un' + 'ium'
'ununium'
```

Два или более *строковых литерала* (т.е. заключенные между кавычками) между собой автоматически объединяются.:

```
>>> 'Py' 'thon'
'Python'
```

Эта функция особенно полезна при необходимости разбиения длинных строк:

```
>>> text = ('Put several strings within parentheses '
...         'to have them joined together.')
>>> text
'Put several strings within parentheses to have them joined together.'
```

Это работает только с двумя литералами, но не с переменными или выражениями:

```
>>> prefix = 'Py'
>>> prefix 'thon'  # не могут быть конкатенированы переменные и строковые␣
↪литералы
  File "<stdin>", line 1
    prefix 'thon'
                ^
SyntaxError: invalid syntax
>>> ('un' * 3) 'ium'
  File "<stdin>", line 1
    ('un' * 3) 'ium'
                   ^
SyntaxError: invalid syntax
```

Если требуется объединить переменные или переменные и литералы, используйте +:

```
>>> prefix + 'thon'
'Python'
```

Строки могут быть *индексируемыми* (индексироваться), причем первый символ имеет индекс 0. Отдельный тип символа отсутствует; символ является простой строкой единичного размера:

```
>>> word = 'Python'
>>> word[0]  # символ в позиции 0
'P'
>>> word[5]  # символ в позиции 5
'n'
```

Индексы также могут быть отрицательными числами, начало отсчёта справа:

```
>>> word[-1]  # последний символ
'n'
>>> word[-2]  # предпоследний символ
'o'
>>> word[-6]
'P'
```

Обратите внимание, что поскольку значение -0 равно 0, отрицательные индексы начинаются с -1.

Кроме индексирования, также поддерживаются *слайсы (нарезки)*. При индексировании используются для получения отдельных символов, *слайсы* позволяют получить подстроку:

```
>>> word[0:2]  # символы с позиции 0 (включительно) до 2 (исключен)
'Py'
>>> word[2:5]  # символы с позиции 2 (включительно) до 5 (исключен)
'tho'
```

Обратите внимание, что начало всегда включено, а конец всегда исключен. Это подтверждается том, что `s[:i] + s[i:]` всегда эквивалентно `s`:

```
>>> word[:2] + word[2:]
'Python'
>>> word[:4] + word[4:]
'Python'
```

Индексы слайсов имеют полезные значения по умолчанию; опущенный первый индекс по умолчанию равен нулю, опущенный второй индекс по умолчанию соответствует размеру нарезаемой строки.:

```
>>> word[:2]   # символы с начала строки до 2 позиции (исключен)
'Py'
>>> word[4:]   # символы с начала 4 (включительно) до конца
'on'
>>> word[-2:]  # символы с предпоследнего (включительно) до конца
'on'
```

Один из способов запомнить, как работают слайсы, это думать об индексах как о указателях *между* символом левого края первого символа, с номером 0. Тогда как правый край последнего символа строки из *n* символов имеет *n* индекс, например:

```
 +---+---+---+---+---+---+
 | P | y | t | h | o | n |
 +---+---+---+---+---+---+
 0   1   2   3   4   5   6
-6  -5  -4  -3  -2  -1
```

Первый ряд чисел дает позицию индексов 0...6 в строке; второй ряд дает соответствующие отрицательные индексы. Слайс от *i* до *j* состоит из всех символов между краями, отмеченными *i* и *j*, соответственно.

Для неотрицательных индексов длина среза — это разность между индексами, если оба находятся в границах. Например, длина `word[1:3]` равна 2.

Попытка использовать слишком большой индекс приведет к ошибке:

```
>>> word[42]  # в слове 6 символов
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
IndexError: string index out of range
```

Однако индексы слайсов вне диапазона обрабатываются корректно при нарезке:

```
>>> word[4:42]
'on'
>>> word[42:]
''
```

В Python строки неизменяемые — они *неизменны*. Поэтому присвоение значения индексированной позиции в строке приводит к ошибке:

```
>>> word[0] = 'J'
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: 'str' object does not support item assignment
>>> word[2:] = 'py'
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: 'str' object does not support item assignment
```

Если требуется другая строка, следует создать новую: