Николай Прохоренок
Владимир Дронов

PRO
ПРОФЕССИОНАЛЬНОЕ ПРОГРАММИРОВАНИЕ

# Python 3 и PyQt 6

## Разработка приложений

Типы данных Python
Объектно-ориентированное программирование
Работа с файлами и каталогами
Взаимодействие с Windows
Создание оконных программ
Работа с базами данных
Мультимедиа
Запись звука, видео и фото
Печать и экспорт в формат PDF
Работающий пример: приложение «Судоку»

+ Материалы на www.bhv.ru

bhv®

УДК 004.43
ББК 32.973.26-018.1
П84

**Прохоренок, Н. А.**

П84 Python 3 и PyQt 6. Разработка приложений / Н. А. Прохоренок, В. А. Дронов. — СПб.: БХВ-Петербург, 2023. — 832 с.: ил. — (Профессиональное программирование)

ISBN 978-5-9775-1706-5

Описан язык Python 3: типы данных, операторы, условия ветвления и выбора, циклы, регулярные выражения, функции, классы, работа с файлами и каталогами, взаимодействие с механизмами Windows, часто используемые модули стандартной библиотеки. Особое внимание уделено библиотеке PyQt, позволяющей создавать приложения с графическим интерфейсом. Описаны средства для создания и вывода окон, основных компонентов (кнопок, полей, списков, таблиц, меню, панелей инструментов и др.). Рассмотрена обработка событий и сигналов, разработка многопоточных программ, работа с базами данных, вывод графики, воспроизведение мультимедиа, запись аудио, видео и фото, печать документов, экспорт их в формат Adobe PDF и сохранения настроек программ. Дан пример полнофункционального приложения для создания и решения головоломок судоку. На сайте издательства размещен электронный архив со всеми примерами из книги.

*Для программистов*

УДК 004.43
ББК 32.973.26-018.1

**Группа подготовки издания:**

| | |
|---|---|
| Руководитель проекта | *Евгений Рыбаков* |
| Зав. редакцией | *Людмила Гауль* |
| Редактор | *Григорий Добин* |
| Компьютерная верстка | *Ольги Сергиенко* |
| Дизайн серии | *Марины Дамбиевой* |
| Оформление обложки | *Зои Канторович* |

"БХВ-Петербург", 191036, Санкт-Петербург, Гончарная ул., 20

ISBN 978-5-9775-1706-5

# Оглавление

# ГЛАВА 1

# Первые шаги

Прежде чем мы начнем знакомство с языком Python, хочется отметить, что книги по программированию нужно не только читать, — весьма желательно выполнять все имеющиеся в них примеры, а также экспериментировать, что-нибудь в этих примерах изменяя. Поэтому, если вы удобно устроились на диване и настроились просто читать, у вас практически нет шансов изучить язык. Чем больше вы будете делать самостоятельно, тем большему научитесь.

> ***Внимание!***
> Текущая версия Python поддерживает только Microsoft Windows версий 8.1, 10 и 11. Более старые версии этой операционной системы не поддерживаются.

## 1.1. Установка Python

Вначале необходимо установить на компьютер *интерпретатор* (или *исполняющую среду*) Python — программный пакет, который переводит программы, написанные на этом языке, в представление, «понятное» центральному процессору компьютера. Без интерпретатора ни одну Python-программу запустить не получится.

1. Для загрузки дистрибутива исполняющей среды заходим на страницу **https://www.python.org/downloads/** и в списке доступных версий щелкаем на гиперссылке **Python 3.10.1** (эта версия является наиболее актуальной на момент подготовки книги). На открывшейся странице находим раздел **Files** и щелкаем на гиперссылке **Windows installer (32-bit)** (32-разрядная редакция интерпретатора) или **Windows installer (64-bit)** (его 64-разрядная редакция) — в зависимости от редакции нашей операционной системы. В результате на наш компьютер будет загружен файл python-3.10.1.exe или python-3.10.1-amd64.exe соответственно. Запускаем загруженный файл двойным щелчком на нем.
2. В открывшемся окне (рис. 1.1) проверяем, установлен ли флажок **Install launcher for all users (recommended)** (Установить утилиту launcher для всех пользователей), устанавливаем флажок **Add Python 3.10 to PATH** (Добавить Python 3.10 в список путей из переменной `PATH`) и нажимаем кнопку **Customize installation** (Настроить установку).

   Утилита launcher, поставляемая в составе дистрибутива, предназначена для запуска программ под определенной версией Python, если на компьютере установлены несколько разных версий этого языка. К сожалению, если ее не установить, не получится ассоциировать файлы Python-программ с исполняющей средой Python, в результате чего запускать программы щелчком мыши станет невозможно.


**Рис. 1.1.** Установка Python: шаг 1

3. В следующем диалоговом окне (рис. 1.2) предлагается выбрать устанавливаемые компоненты. Оставляем установленными все флажки, представляющие эти компоненты, и нажимаем кнопку **Next**.

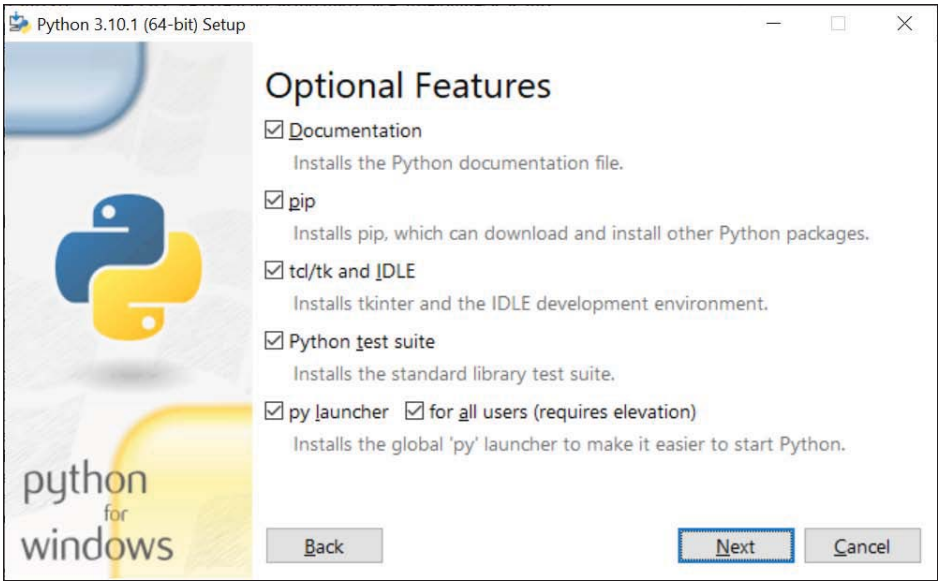


**Рис. 1.2.** Установка Python: шаг 2

4. На следующем шаге (рис. 1.3) задаем дополнительные настройки и выбираем путь установки. Проверяем, установлены ли флажки **Associate files with Python (requires the py launcher)** (Ассоциировать Python-файлы с исполняющей средой), **Create shortcuts for**

**installed applications** (Создать ярлыки для установленных приложений), **Add Python to environment variables** (Добавить Python в переменные окружения), устанавливаем флажки **Install for all users** (Установить для всех пользователей) и **Precompile standard library** (Предварительно откомпилировать стандартную библиотеку).

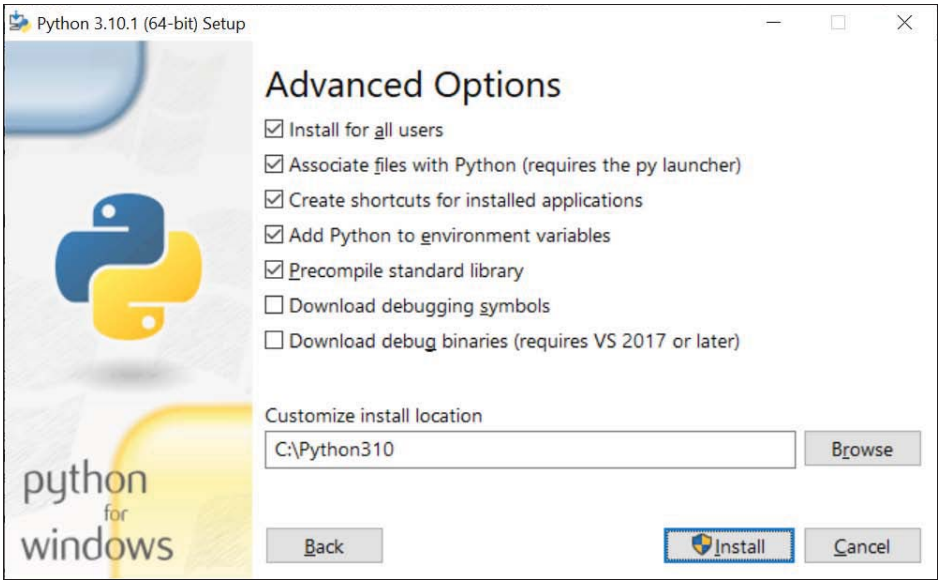


**Рис. 1.3.** Установка Python: шаг 3

Ассоциирование Python-файлов с исполняющей средой позволит запускать эти файлы щелчками на них. Добавление пути к интерпретатору Python в список путей из переменной `PATH` даст возможность запускать этот интерпретатор набором в *консоли* (это сокращенное название командной строки Windows, применяемое далее в книге) команды `python`. Предварительная компиляция стандартной библиотеки ускорит запуск Python-программ.

Обратите внимание на указание пути, по которому будет установлен Python. Изначально предлагается установить интерпретатор по пути c:\Program Files (x86)\Python310 или c:\Program Files\Python310. Однако в этом случае могут возникнуть проблемы при использовании дополнительных библиотек, расширяющих функциональность интерпретатора.

Поэтому авторы книги рекомендуют установить Python по пути c:\Python310 — т. е. непосредственно в корень диска (см. рис. 1.3). В этом случае никаких проблем при использовании дополнительных библиотек не возникнет.

Задав все необходимые параметры, нажимаем кнопку **Install** и положительно отвечаем на появившееся на экране предупреждение UAC.

5. После завершения установки откроется окно, показанное на рис. 1.4. Нажимаем в нем кнопку **Close** для выхода из программы установки.

В результате на компьютер будут установлены две редакции интерпретатора Python:

- python.exe — предназначена для исполнения консольных программ и задействуется при щелчке мышью на файле с расширением py.

С помощью этой редакции можно выполнять и оконные программы, однако в этом случае на экране появится окно консоли, что может обескуражить пользователя;

- pythonw.exe — служит для исполнения оконных программ и задействуется при щелчке мышью на файле с расширением pyw. Консоль при этом не выводится.

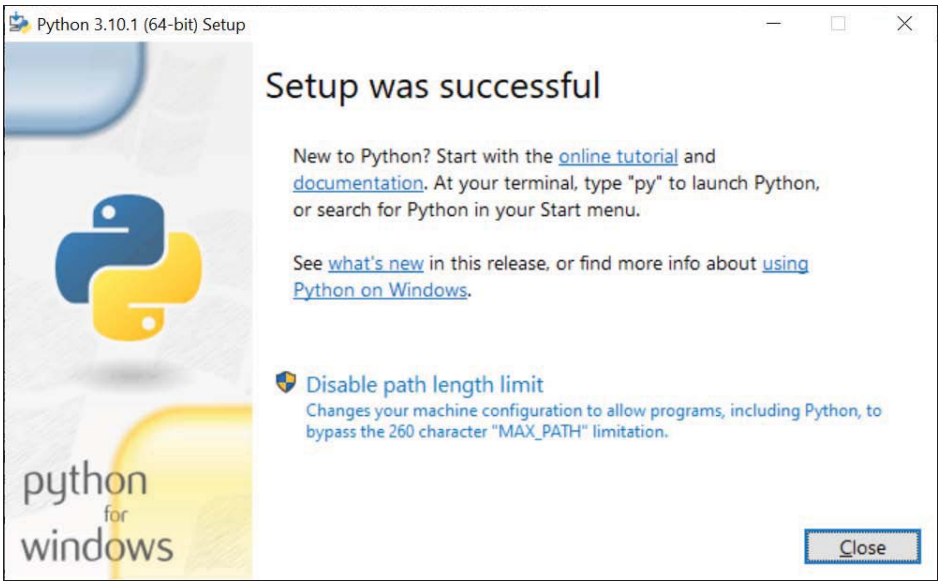


**Рис. 1.4.** Установка Python: шаг 4

## 1.2. Интерактивный режим Python. Утилита IDLE

Изучать Python удобнее всего, вводя инструкции этого языка вручную и тотчас получая результаты их выполнения. Для этого интерпретатор Python должен работать в *интерактивном режиме*.

Запустить интерпретатор Python в интерактивном режиме можно одним из следующих способов:

- выбрав в меню **Пуск | Python 3.10** пункт **Python 3.10 (32-bit)** или **Python 3.10 (64-bit)**;
- набрав в консоли `python` и нажав клавишу <Enter>;
- выполнив щелчок мышью (одинарный или двойной — в зависимости от настроек операционной системы) на файле python.exe в каталоге c:\Python310.

В результате на экране появится окно, напоминающее таковое у командной строки Windows (рис. 1.5). Символами >>> в этом окне помечается приглашение для ввода инструкций. Если после этих символов ввести, например, `2 + 2` и нажать клавишу <Enter>, то на следующей строке появится результат сложения — `4`, а затем — приглашение для ввода новой инструкции. Таким образом, интерактивный режим можно использовать и в качестве очень мощного калькулятора.

Однако значительно удобнее учить язык и выполнять вычисления, работая с утилитой IDLE, которая входит в комплект поставки Python. Эта утилита представляет собой своего рода обертку вокруг интерпретатора, работающего в интерактивном режиме, которая

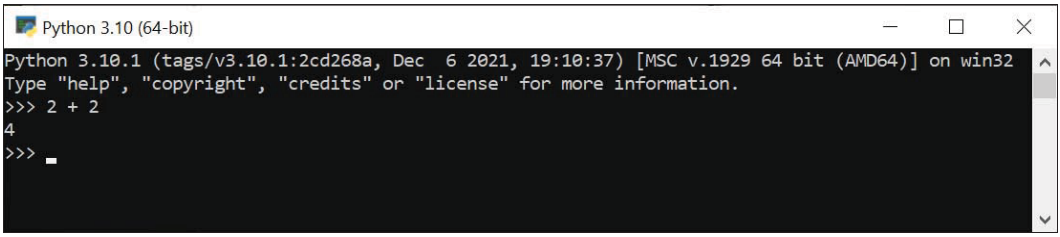


**Рис. 1.5.** Интерпретатор Python, запущенный в интерактивном режиме

добавляет интерпретатору расширенную функциональность (в частности, синтаксическую подсветку программного кода и вывод подсказок).

Для запуска IDLE достаточно выбрать в меню **Пуск | Python 3.10** пункт **IDLE (Python 3.10 32-bit)** или **IDLE (Python 3.10 64-bit)**. На экране появится окно **IDLE Shell** (рис. 1.6), в котором выводится интерактивный интерпретатор Python.

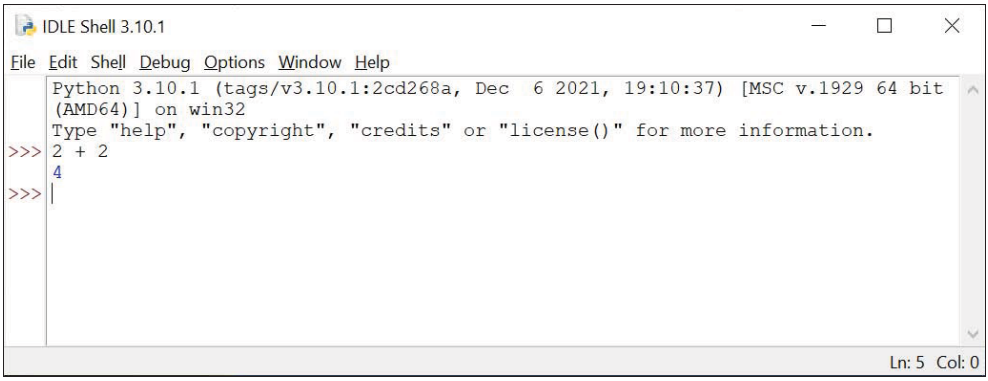


**Рис. 1.6.** Окно **IDLE Shell** утилиты IDLE

Отметим, что в окне **IDLE Shell** приглашение >>> отображается на свободной полосе, расположенной левее области редактирования, в которой вводятся инструкции Python и выводятся результаты. Поэтому при копировании кода в буфер обмена комбинацией клавиш <Ctrl>+<C> приглашение в него не попадает.

Кроме того, IDLE позволяет работать с Python-программами, хранящимися в файлах. Содержимое таких файлов выводится в отдельных окнах. Более подробно функциональные возможности этой утилиты будут рассмотрены позже.

> ***Внимание!***
>
> В дальнейшем для написания и выполнения Python-программ мы будем использовать утилиту IDLE.

## 1.3. Введение в Python-программирование

Изучение языков программирования принято начинать с программы, выводящей надпись «Привет, мир!» Не будем нарушать традиции и продемонстрируем, как это будет выглядеть на Python (листинг 1.1).

**Листинг 1.1. Первая программа на Python**

```
# Выводим надпись с помощью функции print()
print("Привет, мир!")
```

Эта программа состоит из двух *инструкций* (или *выражений*) — предписаний, указывающих Python выполнить какое-либо законченное действие и записываемых на отдельных строках. Первая инструкция представляет собой комментарий — пометку, написанную самим программистом с целью объяснить себе или коллегам, что делает тот или иной код. Вторая инструкция выводит в консоли строку «Привет, мир!»

Вывод значения во второй инструкции выполняется посредством функции `print()`. *Функцией* называется языковая конструкция, выполняющая над переданными ей значениями (*параметрами*) какое-либо относительно сложное действие (в нашем случае — вывод в консоль). Параметры записываются внутри круглых скобок и отделяются друг от друга запятыми. В листинге 1.1 функции `print()` передается всего один параметр — выводимое значение.

Запускаем интерпретатор Python в интерактивном режиме (как это сделать, было описано в *разд. 1.2*). Правее приглашения `>>>` вводим сначала первую строку из листинга 1.1, а затем вторую. После ввода каждой строки нажимаем клавишу <Enter>. В третьей строке сразу отобразится результат, а далее — приглашение для ввода новой команды. Мы увидим следующее (напоминаем, что символы `>>>` — это приглашение, их вводить не нужно):

```
>>> # Выводим надпись с помощью функции print()
>>> print("Привет, мир!")
    Привет, мир!
>>>
```

> ***Внимание!***
>
> В дальнейшем в подобных фрагментах кода инструкции, вводимые вручную, будут помечаться расположенными левее символами приглашения `>>>`, а результаты, выведенные интерпретатором, показываться без этих символов.

Для создания файла с Python-программой в меню **File** окна **IDLE Shell** выбираем пункт **New File** или нажимаем комбинацию клавиш <Ctrl>+<N>. В открывшемся окне набираем код из листинга 1.1, а затем сохраняем его в каком-либо каталоге в файле с именем hello_world.py, выбрав пункт меню **File | Save** (или нажав комбинацию клавиш <Ctrl>+<S>).

Кстати, файл с Python-кодом в терминологии этого языка называется *модулем*.

Запустить написанную программу на выполнение можно, выбрав в окне с кодом этой программы пункт меню **Run | Run Module** или нажав клавишу <F5>. Результат выполнения программы будет отображен в окне **IDLE Shell**.

Запустить Python-программу вне среды IDLE можно двумя способами:

- набрав имя хранящего ее файла вместе с расширением (например: `hello_world.py`) в консоли и нажав клавишу <Enter>.

  Результат выполнения будет выведен там же, в консоли;

- выполнив щелчок мышью (двойной или одинарный — в зависимости от настроек системы) на значке файла с этой программой.

  В этом случае после вывода результата окно консоли сразу закроется. Чтобы предотвратить его закрытие, необходимо добавить в конец кода программы вызов функции

`input()`, которая станет ожидать нажатия клавиши <Enter> и не позволит окну сразу закрыться. С учетом сказанного наша первая программа будет выглядеть так, как показано в листинге 1.2.

**Листинг 1.2. Программа для запуска с помощью щелчка мыши**

```
print("Привет, мир!")      # Выводим строку
input()                    # Ожидаем нажатия клавиши <Enter>
```

Отметим, что функция `input()` в листинге 1.2 вызывается без параметров. В таком случае после имени функции ставятся пустые круглые скобки.

> ***Примечание***
>
> Если до выполнения функции `input()` в коде возникнет ошибка, то сообщение о ней будет выведено в консоли, но последняя после этого сразу закроется, и вы не сможете прочитать сообщение об ошибке. Попав в подобную ситуацию, запустите программу из консоли или утилиты IDLE, и вы сможете прочитать это сообщение.

Чтобы открыть Python-файл для редактирования, запустим IDLE, выберем пункт меню **File | Open** (комбинация клавиш <Ctrl>+<O>) и выберем нужный файл в появившемся на экране диалоговом окне открытия файла. Файл будет открыт в новом окне утилиты IDLE.

# 1.4. Принципы написания Python-программ

- Программа, написанная на Python, представляет собой обычный текстовый файл с расширением py (будет исполнен обычным интерпретатором python.exe) или pyw (будет исполнен «оконным» интерпретатором pythonw.exe).

  Для написания Python-программ, как уже ранее отмечалось, можно использовать, помимо IDLE, любой подходящий текстовый редактор: стандартный Блокнот, Notepad++ (**https://notepad-plus-plus.org/**), Visual Studio Code (**https://code.visualstudio.com/**) и др. Также существует ряд специализированных текстовых редакторов, предназначенных именно для Python-программистов, в частности PyCharm[1] (**https://www.jetbrains.com/ru-ru/pycharm/**).
- Инструкции, записанные в коде программы, выполняются последовательно, в порядке сверху вниз.
- Каждая инструкция Python должна располагаться на отдельной строке. Признаком конца инструкции является перевод строки. Пример программы из двух выражений приведен в листинге 1.3.

**Листинг 1.3. Программа, выполняющая арифметические вычисления**

```
x = 15 + 20 + 30
print(x)
```

Первое выражение складывает числа 15, 20 и 30 и заносит сумму в переменную с именем `x` (*переменная* — ячейка памяти, имеющая уникальное имя, способная хранить ка-

[1] Доступны как базовая бесплатная, так и расширенная платная редакции этого редактора.

кое-либо значение — например, число, и создаваемая при присваивании ей значения). Второе выражение выводит содержимое этой переменной — число 65 — в консоли.

В первом выражении используются два оператора. *Оператором* называется языковая конструкция, выполняющая над переданными ему значениями какое-либо элементарное действие (например, сложение чисел или присваивание результата указанной переменной). Оператор сложения обозначается привычным символом +, а оператор присваивания — символом =.

В Windows перевод строки формируется комбинацией символов `\r` (перевод каретки) и `\n` (перевод строки), в UNIX — одним символом `\n`.

Если загрузить файл Python-программы по протоколу FTP в бинарном режиме, то символ `\r` вызовет фатальную ошибку. По этой причине файлы по протоколу FTP следует загружать только в текстовом (ASCII) режиме — тогда символ `\r` будет удален автоматически.

♦ Если строка с инструкцией получилась слишком длинной, инструкцию можно разделить на несколько строк одним из следующих способов:

- поставить в месте разрыва строк символ \, сразу после которого вставить перевод строки (как обычно, нажатием клавиши <Enter>):

```
x = 15 + 20 + \
    30
print(x)
```

Как правило, вторая и все последующие строки, содержащие длинное выражение, набираются с отступами слева — для улучшения читаемости кода. Следует только формировать эти отступы *исключительно пробелами* и делать *одинаковой длины* (содержащими одинаковое количество пробелов).

При вводе подобного рода многострочной инструкции в интерпретаторе, работающем в интерактивном режиме, строки с продолжением инструкции помечаются расположенным слева приглашением в виде трех точек (..., а не >>>, как обычно). Кроме того, интерпретатор сам выводит последующие строки с отступами. Чтобы завершить ввод многострочной инструкции, следует, введя ее последнюю строку, нажать клавишу <Enter> (при этом после многострочной инструкции будет создана пустая строка с приглашением из трех точек). Пример:

```
>>> x = 15 + 20 + \   # Вводим первую строку многострочной инструкции
...     30             # Вводим вторую, последнюю, строку
...           # Завершаем ввод многострочной инструкции, нажав <Enter>
>>> print(x)          # Вводим следующую инструкцию
```

- заключить выражение в круглые скобки, внутри которых вставить нужное количество переводов строки:

```
x = (15 +
     20 +
     30)
print(x)
```

- определение списка и словаря (описываются в *главах 8* и *9*), при оформлении которых используются квадратные и фигурные скобки соответственно, также можно разбить на несколько строк:

пример определения списка:

```
arr = [15, 20,
       30]
print(arr)
```

пример определения словаря:

```
arr = {"x": 15, "y": 20,
       "z": 30}
print(arr)
```

- Короткие инструкции можно записать в одной строке, разделив их символами точки с запятой (`;`):

```
x = 15 + 20 + 30; print(x)
```

После точки с запятой не возбраняется ставить пробел — для улучшения читаемости кода.

- *Блоки* (наборы из произвольного количества инструкций, входящие в состав более сложных инструкций) формируются *отступами слева*. Такие отступы должны формироваться *исключительно пробелами* и быть *одинаковой длины* (т. е. содержать одинаковое количество пробелов). Символы табуляции в отступах не допускаются и при выполнении программы вызывают возникновение ошибки.

Листинг 1.4 показывает код программы, выводящей последовательно числа от 1 до 10, которые разделяются тремя дефисами.

**Листинг 1.4. Пример блока**

```
for i in range(1, 11):
    print(i)
    print("---")
```

Первая строка содержит сложную инструкцию — цикл (описывается в *главе 4*), который выполняется 10 раз. При каждом исполнении он заносит в переменную `i` очередное число в диапазоне от 1 до 10 и выполняет блок, входящий в его состав. Этот блок состоит из двух инструкций, записанных во второй и третьих строках: первая инструкция выводит в консоли число из переменной `i`, а вторая — строку из трех дефисов. Обе инструкции, входящие в блок, имеют одинаковый отступ слева, содержащий 4 пробела.

Числа, перебираемые циклом (и, соответственно, количество выполнений, или *итераций*, цикла), указываются в функции `range()` (первая инструкция в листинге 1.4). Первый параметр задает начальное число перебираемого диапазона, второй — конечное число плюс 1.

***Примечание***

4 пробела — общепринятая величина отступа в Python.

При вводе инструкции, содержащей блок, в интерпретаторе, который работает в интерактивном режиме, интерпретатор предваряет выражения, входящие в блок, приглашением в виде трех точек (`...`) и сам вставляет отступы. Чтобы завершить ввод блока, следует, занеся последнее входящее в него выражение, нажать клавишу <Enter> (при этом в набранном коде появится пустая строка с приглашением в виде трех точек). Пример:

```
>>> for i in range(1, 11):
...     print(i)
...     print("---")
...
>>> # Следующая инструкция
```

- Если блок содержит одну короткую инструкцию, и сам блок, и сложную инструкцию, в состав которой он входит, можно записать в одну строку. При этом интерпретатор посчитает, что ввод блока продолжится после нажатия клавиши <Enter>, и выведет приглашение в виде трех точек. Чтобы завершить ввод инструкции, следует снова нажать <Enter>. Пример программы, выводящей в консоли числа от 1 до 10:

```
>>> for i in range(1, 11): print(i)
...
... Вывод пропущен ...
```

## 1.4.1. Комментарии и строки документирования

*Комментарий* — это произвольное пояснение, вставленное в код программы, предназначенное исключительно программисту и полностью игнорируемое интерпретатором. Внутри комментария может располагаться любой текст.

Комментарий в языке Python начинается с символа # и заканчивается концом строки:

```
# Выводим числа от 1 до 10
for i in range(1, 11): print(i)
```

Комментарий может располагаться после собственно инструкции:

```
print("Все, числа закончились") # Сообщаем об окончании чисел
```

Если символ # разместить перед инструкцией, то она не будет выполнена (*закомментированная инструкция*):

```
# print("Привет, мир!") Эта инструкция выполнена не будет
```

Если требуется разместить комментарий из нескольких строк, перед каждой из них придется ставить символ #:

```
# Это наша первая программа!
# Она выводит числа от 1 до 10
# Да, не впечатляет, но для начала неплохо...
for i in range(1, 11): print(i)
```

*Строки документирования* Python обычно применяются для написания инструкций к программам и модулям (подробности — в *главе 6*). Однако их можно использовать и для комментирования кода.

Строка документирования заключается в утроенные кавычки или апострофы:

```
"""
Это наша первая программа!
Она выводит числа от 1 до 10
Да, не впечатляет, но для начала неплохо...
"""
for i in range(1, 11): print(i)
```

## 1.4.2. Кодировки, поддерживаемые Python

Код Python-программы, написанный в IDLE, по умолчанию сохраняется в кодировке UTF-8 без BOM[1].

Если программу следует сохранить в какой-либо другой кодировке (что может пригодиться, например, при переписывании старого Python-кода), в первой строке ее кода следует указать кодировку с помощью инструкции формата:

```
# -*- coding: <Обозначение кодировки> -*-
```

Например, кодировка Windows-1251 указывается инструкцией:

```
# -*- coding: cp1251 -*-
```

Встретив такую инструкцию в коде программы, IDLE при сохранении файла самостоятельно переведет программу в кодировку с заданным обозначением. При использовании других редакторов следует перевести программу в указанную кодировку вручную.

Получить полный список поддерживаемых Python кодировок и их обозначения позволяет программа, приведенная в листинге 1.5.

**Листинг 1.5. Вывод списка поддерживаемых кодировок**

```
import encodings.aliases
arr = encodings.aliases.aliases
keys = list( arr.keys() )
keys.sort()
for key in keys: print("%s => %s" % (key, arr[key]))
```

## 1.4.3. Подготовка Python-программ для выполнения в UNIX

Если программа предназначена для исполнения в операционных системах семейства UNIX, в первой строке кода программы необходимо указать путь к интерпретатору Python:

```
#!/usr/bin/python
```

В некоторых UNIX-системах путь к интерпретатору выглядит по-другому:

```
#!/usr/local/bin/python
```

Иногда можно не указывать точный путь к интерпретатору, а передать название языка программе `env`:

```
#!/usr/bin/env python
```

В этом случае программа `env` произведет поиск интерпретатора Python в соответствии с настройками путей поиска.

Если программа, исполняемая в UNIX, сохранена в кодировке, отличной от UTF-8, обозначение кодировки указывается во второй строке ее кода:

```
#!/usr/bin/python
# -*- coding: cp1251 -*-
```

[1] *BOM* (Byte Order Mark) — метка порядка байтов. Указывает порядок, в котором записываются байты, кодирующие символы в UTF-8.

Также следует разрешить выполнять Python-программу, указав у ее файла права 755 (`-rwxr-xr-x`).

## 1.5. Дополнительные возможности IDLE

Поскольку далее для работы мы будем использовать утилиту IDLE, рассмотрим предлагаемые ею дополнительные возможности.

Обычно после ввода инструкции и нажатия клавиши <Enter> введенная инструкция выполняется, и на следующей строке выводится ее результат (при условии, что инструкция возвращает результат). После чего появляется приглашение для ввода новой инструкции.

При вводе сложной инструкции, содержащей блок, после нажатия клавиши <Enter> интерпретатор, работающий в интерактивном режиме, автоматически создаст отступ и будет ожидать ввода блока. Чтобы сообщить интерпретатору об окончании ввода инструкции, необходимо нажать <Enter> дважды:

```
>>> for n in range(1, 3):
...     print(n)
...
    1
    2
>>>
```

Если ввести какое-либо значение — например, строку или число, и нажать <Enter>, это значение появится в следующей строке:

```
>>> "Привет, мир!"
    'Привет, мир!'
>>>
```

Обратите внимание на то, что строки выводятся в апострофах. Этого не произойдет, если выводить строку с помощью функции `print()`:

```
>>> print("Привет, мир!")
    Привет, мир!
>>>
```

Как говорилось в *разд. 1.2*, окно **IDLE Shell** можно использовать для изучения языка, а также в качестве многофункционального калькулятора (здесь `*` — это оператор умножения):

```
>>> 12 * 32 + 54
    438
>>>
```

Результат вычисления последней инструкции сохраняется в переменной _ (одно подчеркивание). Это позволяет производить дальнейшие расчеты без ввода предыдущего результата — вместо него достаточно ввести символ подчеркивания. Пример (здесь `/` — оператор деления):

```
>>> 125 * 3      # Умножение
    375
>>> _ + 50       # Сложение. Эквивалентно 375 + 50
    425
```

```
>>> _ / 5        # Деление. Эквивалентно 425 / 5
    85.0
>>>
```

При вводе команды можно воспользоваться комбинацией клавиш <Ctrl>+<Пробел>. В результате будет отображен список, из которого можно выбрать нужную языковую конструкцию. Если при открытом списке начать вводить буквы, то показываться будут языковые конструкции, начинающиеся с этих букв. Выбирать конструкцию необходимо с помощью клавиш <↑> и <↓>. После выбора не следует нажимать клавишу <Enter>, иначе это приведет к выполнению инструкции, — просто вводите инструкцию дальше, а список закроется. Такой же список будет автоматически появляться (с некоторой задержкой) при обращении к атрибутам объекта или модуля после ввода точки. Для автоматического завершения языковой конструкции после ввода первых букв можно воспользоваться комбинацией клавиш <Alt>+</>. При каждом последующем нажатии этой комбинации будет вставляться следующая конструкция. Эти две комбинации клавиш очень удобны, если вы забыли, как пишется слово, или хотите, чтобы редактор закончил его за вас.

Иногда возникает необходимость выполнить ранее введенную инструкцию повторно. Для таких случаев IDLE предоставляет комбинации клавиш <Alt>+<N> (вставка первой введенной инструкции) и <Alt>+<P> (вставка последней инструкции). Каждое последующее нажатие этих клавиш будет вставлять следующую (или предыдущую) инструкцию. Для еще более быстрого повторного ввода инструкции следует предварительно ввести ее первые буквы — в этом случае перебираться будут только инструкции, начинающиеся с этих букв.

# 1.6. Вывод данных

Вывести заданные значения можно с помощью функции `print()`:

```
print([<Значения через запятую>][, sep=' '][, end='\n'][, file=sys.stdout][,
                                                        flush=False])
```

Указанные значения при необходимости преобразуются в строки и посылаются в стандартный поток вывода `stdout`. С помощью параметра `file` можно перенаправить вывод в другое место — например, в файл. При этом, если параметр `flush` имеет значение `False`, выводимые значения будут записаны в файл принудительно. Перенаправление вывода мы подробно рассмотрим при изучении файлов.

Если выполняется вывод одного значения, автоматически добавляется символ перевода строки:

```
print("Строка 1")
print("Строка 2")
```

Результат:

```
Строка 1
Строка 2
```

Можно вывести несколько значений в одну строку, указав их в вызове функции `print()` отдельными параметрами, разделенными запятыми:

```
print("Строка 1", "Строка 2")
```

Результат:

```
Строка 1 Строка 2
```

Как видно из примера, между выводимыми значениями автоматически вставляется пробел. С помощью параметра `sep` можно указать другой разделяющий символ. Например, выведем строки без пробела между ними, указав в качестве разделителя пустую строку:

```
print("Строка1", "Строка2", sep="")
```

Результат:

```
Строка 1Строка 2
```

Ряд функций, встроенных в Python, поддерживают так называемые *именованные параметры*. В число таких параметров и входит `sep`. Обратим внимание, как задается его значение в приведенном примере.

После вывода нескольких значений в одном вызове функции `print()` в конце добавляется символ перевода строки. Если необходимо произвести дальнейший вывод на той же строке, то в именованном параметре `end` следует указать другой символ:

```
print("Строка 1", "Строка 2", end=" ")
print("Строка 3")
# Выведет: Строка 1 Строка 2 Строка 3
```

Вызов функции `print()` без параметров выводит пустую строку:

```
for n in range(1, 5):
    print(n, end=" ")
print()
print("Это текст на новой строке")
```

Результат выполнения:

```
1 2 3 4
Это текст на новой строке
```

Здесь мы использовали цикл, выполняющийся четыре раза. На каждой итерации он присваивает переменной `n` число от 1 до 4 и выполняет блок, содержащий вызов функции `print()`, которая выводит число из переменной `n`. Не забываем вставить в выражение, входящее в блок, отступ слева.

Как только цикл выполнится четыре раза, будут исполнены два следующих за ним выражения. В них отступы слева указывать не следует, поскольку эти выражения не должны входить в состав блока. Первое выражение выведет пустую строку, второе — надпись «Это текст на новой строке».

Если необходимо вывести большой блок текста, его следует разместить между утроенными кавычками или утроенными апострофами. В этом случае текст сохраняет свое форматирование:

```
print("""Строка 1
Строка 2
Строка 3""")
```

В результате выполнения этого примера мы получим три строки:

```
Строка 1
Строка 2
Строка 3
```

Вместо функции `print()` можно использовать метод `write()` объекта `stdout` из модуля `sys` (методы очень похожи на функции и в подробностях, вместе с объектами, будут рассмотрены позже):

```
import sys                      # Подключаем модуль sys, чтобы использовать
                                # содержащийся в нем объект stdout
sys.stdout.write("Строка")
```

Как мы помним из *разд. 1.3*, модуль — это просто файл с Python-кодом. Однако модуль `sys` поставляется в составе Python и входит в *стандартную библиотеку* (набор модулей, содержащих полезные функции, объекты и др.) этого языка. Подключив этот модуль, мы можем использовать созданные в нем функции и объекты.

В первой строке с помощью оператора `import` подключаем модуль `sys`, в котором объявлен объект `stdout`. Далее с помощью метода `write()` этого объекта выводим строку. Следует заметить, что метод не вставляет символ перевода строки, поэтому при необходимости следует добавить его самим с помощью символа `\n`:

```
import sys
sys.stdout.write("Строка 1\n")
sys.stdout.write("Строка 2")
```

Метод `write()` возвращает *результат* — значение, полученное в процессе выполненных методом вычислений. Таковым результатом является количество символов в выведенной строке. Его можно присвоить какой-либо переменной и использовать в дальнейшем:

```
import sys
cnt = sys.stdout.write("Привет, Python\n")
print("Символов в выведенной строке: ", cnt)
```

Результат:

```
Привет, Python
Символов в выведенной строке:  15
```

# 1.7. Ввод данных

Для ввода данных в Python предназначена функция `input()`, которая получает данные со стандартного потока ввода `stdin`. Функция имеет следующий формат:

```
input([<Сообщение>])
```

Введенное значение она возвращает в качестве результата. Этот результат следует присвоить какой-либо переменной посредством оператора `=`.

Для примера переделаем нашу первую программу так, чтобы она здоровалась не со всем миром, а только с нами (листинг 1.6).

**Листинг 1.6. Пример использования функции `input()`**

```
name = input("Введите ваше имя: ")
print("Привет,", name)
input("Нажмите <Enter> для закрытия окна")
```

В первой инструкции значение, введенное пользователем и полученное функцией `input()`, присваивается переменной `name`.

Сохраняем программу в файле test.py и запускаем на выполнение с помощью двойного щелчка. Откроется черное окно, в котором мы увидим надпись: **Введите ваше имя:**. Вводим свое имя — например, `Николай`, и нажимаем клавишу <Enter>. В результате будет выведено приветствие: **Привет, Николай**.

При использовании функции `input()` следует учитывать, что при достижении конца файла или при нажатии комбинации клавиш <Ctrl>+<Z>, а затем клавиши <Enter> генерируется исключение `EOFError`. Если не предусмотреть обработку исключения, то программа аварийно завершится. Обработать исключение можно следующим образом:

```
try:
    s = input("Введите данные: ")
    print(s)
except EOFError:
    print("Обработали исключение EOFError")
```

Если внутри блока `try` возникнет исключение `EOFError`, то управление будет передано в блок `except`. После исполнения инструкций в блоке `except` программа нормально продолжит работу.

Передать данные можно в консоли, указав их после имени файла программы. Такие данные доступны через список `argv` из модуля `sys`. Первый элемент списка `argv` будет содержать имя файла запущенной программы, а последующие элементы — переданные данные. Для примера создадим файл test2.py в каталоге C:\book. Содержимое файла приведено в листинге 1.7.

**Листинг 1.7. Получение данных из консоли**

```
import sys
arr = sys.argv[:]
for n in arr:
    print(n)
```

Теперь запустим программу на выполнение из консоли, передав ей данные. Запустим консоль, для чего выберем в меню **Пуск** пункт **Выполнить**, в открывшемся окне наберем команду `cmd` и нажмем кнопку **OK**. Откроется черное окно с приглашением для ввода команд. Перейдем в каталог C:\book, набрав команду:

```
cd C:\book
```

В консоли должно появиться приглашение:

```
C:\book>
```

Для запуска нашей программы вводим команду:

```
test2.py uhud opk987
```

В этой команде мы передаем имя файла (`test2.py`) и некоторые данные (`-uNik` и `-p123`). Результат выполнения программы будет выглядеть так:

```
test2.py
uhud
opk987
```