ГЛАВА 1
Основы Python

В этой главе рассмотрены основные особенности языка Python — переменные, типы данных, выражения, функции, классы, ввод/вывод, а также управление логикой выполнения программы. Глава завершается обсуждением модулей, написанием сценариев и пакетов, советами по организации больших программ. Здесь вы не найдете исчерпывающих описаний каждой возможности или инструментов, необходимых для более крупных проектов на Python. Опытные программисты смогут почерпнуть нужную информацию для написания полнофункциональных программ. Новичкам желательно опробовать примеры в простой среде — например, в окне терминала или текстовом редакторе.

1.1. ЗАПУСК PYTHON

Программы на языке Python выполняются интерпретатором. Есть много разных сред, в которых он может работать, — в IDE, браузерах или окнах терминала. Но интерпретатор в первую очередь — текстовое приложение, которое можно запустить командой `python` из командной строки, такой как `bash`. Python 2 может быть установлен на одном компьютере с Python 3. Поэтому вам придется ввести команду `python2` или `python3` для выбора версии. В книге предполагается, что вы используете Python 3.8 или более новую версию.

При запуске интерпретатора появляется приглашение. В нем можно вводить программы в режиме REPL (Read-Evaluation-Print Loop, то есть «цикл чтение/вычисление/печать»). Например, в следующем примере интерпретатор выводит сообщение об авторских правах и отображает приглашение `>>>`, где пользователь вводит известное приветствие Hello World:

```
Python 3.8.0 (default, Feb 3 2019, 05:53:21)
[GCC 4.2.1 Compatible Apple LLVM 8.0.0 (clang-800.0.38)] on darwin
Type "help", "copyright", "credits" or "license" for more information.
```

```
>>> print('Hello World')
Hello World
>>>
```

В некоторых средах приглашение может выглядеть иначе. Следующий вывод получен в `ipython` (альтернативная оболочка для Python):

```
Python 3.8.0 (default, Feb 4, 2019, 07:39:16)
Type 'copyright', 'credits' or 'license' for more information
IPython 6.5.0 -- An enhanced Interactive Python. Type '?' for help.

In [1]: print('Hello World')
Hello World

In [2]:
```

Независимо от конкретной формы вывода общий принцип остается один. Вы вводите команду, она выполняется, и вы немедленно получаете результат.

Интерактивный режим Python — одна из самых полезных возможностей языка. Вы можете ввести любую команду и немедленно увидеть результат. Такой режим полезен для отладки и экспериментов. Многие используют интерактивный режим Python как настольный калькулятор:

```
>>> 6000 + 4523.50 + 134.25
10657.75
>>> _ + 8192.75
18850.5
>>>
```

Когда Python используют в интерактивном режиме, переменная `_` содержит результат последней операции. Это удобно, если вы хотите задействовать ее в дальнейшем. Такая переменная определяется только в интерактивном режиме, не пытайтесь использовать ее в сохраняемых программах.

Для выхода из интерактивного интерпретатора введите команду `quit()` или символ EOF (End Of File). В системе UNIX это сочетание клавиш Ctrl+D, а в Windows — Ctrl+Z.

1.2. ПРОГРАММЫ PYTHON

Если вы хотите создать программу, которую можно запускать многократно, поместите операторы в текстовый файл. Например:

```
# hello.py
print('Hello World')
```

Исходные файлы Python — это текстовые файлы в кодировке UTF-8, обычно имеющие суффикс `.py`. Символ `#` — это комментарий, продолжающийся до конца строки. Международные символы («Юникод») могут свободно применяться в исходном коде при условии использования кодировки UTF-8 (она выбирается по умолчанию в большинстве редакторов, но, если вы не уверены, проверьте конфигурацию редактора).

Чтобы выполнить файл `hello.py`, укажите его имя в командной строке интерпретатора:

```
shell % python3 hello.py
Hello World
shell %
```

Интерпретатор часто указывается в первой строке программы символами `#!`:

```
#!/usr/bin/env python3
print('Hello World')
```

В UNIX вы сможете запустить программу командой `hello.py` в командной оболочке. Например, `chmod +x hello.py` (если файлу были предоставлены разрешения выполнения).

В Windows для запуска можно дважды щелкнуть на файле `.py` или ввести имя программы в поле команды Выполнить меню Пуск. Строка `#!`, если она есть, используется для выбора версии интерпретатора (Python 2 или 3). Программа выполняется в консольном окне, которое исчезает сразу после завершения программы — часто до того, как вы успеете прочитать ее вывод. Для отладки лучше запускать программу в среде разработки Python.

Интерпретатор выполняет команды по порядку, пока не достигнет конца входного файла. В этот момент программа прекращает работу, а интерпретатор Python завершается.

1.3. ПРИМИТИВЫ, ПЕРЕМЕННЫЕ И ВЫРАЖЕНИЯ

Python — это набор примитивных типов — целых чисел, чисел с плавающей точкой, строк и т. д.:

```
42          # int
4.2         # float
'forty-two' # str
True        # bool
```

Переменная — имя, указывающее на значение. Значение представляет объект некоторого типа:

```
x = 42
```

Иногда тип явно указывается для имени:

```
x: int = 42
```

Тип — лишь подсказка, упрощающая чтение кода. Он может использоваться сторонними инструментами проверки кода. В остальных случаях он полностью игнорируется. Указание типа никак не помешает вам присвоить переменной значение другого типа.

Выражение — это комбинация примитивов, имен и операторов, в результате вычисления которой будет получено некоторое значение:

```
2 + 3 * 4 # -> 14
```

Следующая программа использует переменные и выражения для вычисления сложных процентов:

```
# interest.py

principal = 1000        # Исходная сумма
rate = 0.05             # Процентная ставка
numyears = 5            # Количество лет
year = 1
while year <= numyears:
    principal = principal * (1 + rate)
    print(year, principal)
    year += 1
```

При выполнении программа выдает следующий результат:

```
1 1050.0
2 1102.5
3 1157.625
4 1215.5062500000001
5 1276.2815625000003
```

Команда `while` проверяет условное выражение, следующее сразу за ключевым словом. В случае истинности проверяемого условия выполняется тело команды `while`. Затем это условие проверяется повторно и тело выполняется снова, пока условие не станет ложным. Тело цикла обозначается отступами. Так, три оператора, следующие за `while` в `interest.py`, выполняются при каждой итерации. В спецификации Python не указана величина отступов.

Важно лишь, чтобы отступ был единым в границах блока. Чаще всего используются отступы из четырех пробелов на один уровень.

Один из недостатков программы `interest.py` — не очень красивый вывод. Для его улучшения можно выровнять столбцы по правому краю и ограничить точность вывода чисел двумя знаками в дробной части. Измените функцию `print()`, чтобы в ней использовалась так называемая *f-строка*:

```
print(f'{year:>3d} {principal:0.2f}')
```

В f-строках могут вычисляться выражения и имена переменных. Для этого они заключаются в фигурные скобки. К каждому заменяемому элементу может быть присоединен спецификатор формата. Так, `'>3d'` обозначает трехзначное десятичное число, выравниваемое по правому краю, `'0.2f'` обозначает число с плавающей точкой, выводимое с двумя знаками точности. Больше информации о кодах форматирования вы найдете в главе 9.

Теперь вывод программы выглядит так:

```
1 1050.00
2 1102.50
3 1157.62
4 1215.51
5 1276.28
```

1.4. АРИФМЕТИЧЕСКИЕ ОПЕРАТОРЫ

Python поддерживает стандартный набор математических операторов (табл. 1.1). Их значение такое же, как и в большинстве языков программирования.

Таблица 1.1. Арифметические операторы

ОПЕРАТОР	ОПИСАНИЕ
`x + y`	Сложение
`x - y`	Вычитание
`x * y`	Умножение
`x / y`	Деление
`x // y`	Целочисленное деление
`x ** y`	Возведение в степень (`x` в степень `y`)
`x % y`	Остаток (от деления `x` на `y`)
`-x`	Унарный минус
`+x`	Унарный плюс

Применение оператора деления (`/`) к целым числам дает результат с плавающей точкой. Так, результат выражения `7/4` равен `1.75`. Оператор целочисленного деления `//` (его еще называют делением с остатком) усекает результат до целого числа и работает как с целыми числами, так и с числами с плавающей точкой. Оператор по модулю возвращает остаток от деления `x // y`. Например, результат выражения `7 % 4` равен `3`. Для чисел с плавающей точкой оператор по модулю возвращает остаток от деления `x // y` в виде числа с плавающей точкой, то есть `x - (x // y) * y`.

Встроенные функции из табл. 1.2 реализуют некоторые часто используемые числовые операции.

Таблица 1.2. Распространенные математические функции

ФУНКЦИЯ	ОПИСАНИЕ
`abs(x)`	Модуль (абсолютная величина)
`divmod(x,y)`	Возвращает `(x // y, x % y)`
`pow(x,y [,modulo])`	`(x ** y) % modulo`
`round(x,[n])`	Округляет до ближайшего кратного 10 в степени `n`

Функция `round()` реализует так называемое банковское округление. Если округляемое значение находится на равных расстояниях от двух кратных, оно округляется до ближайшего четного кратного (например, 0,5 округляется до 0, а 1,5 — до 2,0).

С целыми числами можно использовать ряд дополнительных операторов для битовых операций (табл. 1.3).

Таблица 1.3. Битовые операторы

ОПЕРАЦИЯ	ОПИСАНИЕ
`x << y`	Битовый сдвиг влево
`x >> y`	Битовый сдвиг вправо
`x & y`	Битовая операция И
`x | y`	Битовая операция ИЛИ
`x ^ y`	Битовый сдвиг ИСКЛЮЧАЮЩЕЕ ИЛИ
`~x`	Битовое отрицание

Эти операции обычно используют с двоичными целыми числами:

```
a = 0b11001001
mask = 0b11110000
x = (a & mask) >> 4 # x = 0b1100 (12)
```

В этом примере `0b11001001` — запись целого числа в двоичном виде. Его можно записать в десятичной форме `201` или шестнадцатеричной форме `0xc9`. Но, если вы работаете на уровне отдельных битов, двоичная запись помогает наглядно представить происходящее.

Суть битовых операций в том, что целые числа используют представление в дополнительном коде, а знаковый бит бесконечно распространяется влево. Если вы работаете с низкоуровневыми битовыми последовательностями, которые должны представлять целые числа для оборудования, будьте внимательны. Python не усекает биты и не поддерживает переполнение — вместо этого результат неограниченно растет. Вы сами должны следить, чтобы результат был нужного размера или усекался при необходимости.

Для сравнения чисел используются операторы сравнения (табл. 1.4).

Таблица 1.4. Операторы сравнения

ОПЕРАЦИЯ	ОПИСАНИЕ
`x == y`	Равно
`x != y`	Не равно
`x < y`	Меньше
`x > y`	Больше
`x >= y`	Больше или равно
`x <= y`	Меньше или равно

Результат сравнения — логическое (булевское) значение `True` или `False`.

Операторы `and`, `or` и `not` (не путайте с битовыми операциями из табл. 1.3) могут использоваться для построения более сложных логических выражений. Поведение этих операторов описано в табл. 1.5.

Таблица 1.5. Логические операторы

ОПЕРАЦИЯ	ОПИСАНИЕ
`x or y`	Если значение `x` ложно, возвращается `y`, в противном случае возвращается `x`
`x and y`	Если значение `x` ложно, возвращается `x`, в противном случае возвращается `y`
`not x`	Если значение `x` ложно, возвращается `True`, в противном случае возвращается `False`

Значение интерпретируется как ложное, если это `False`, `None`, числовой нуль или пустое значение. Во всех остальных случаях оно интерпретируется как истинное.

Очень часто встречаются выражения, обновляющие значение:

```
x = x + 1
y = y * n
```

В таких случаях можно использовать следующие сокращенные операции:

```
x += 1
y *= n
```

Сокращенная форма обновления может использоваться с любым из операторов `+`, `-`, `*`, `**`, `/`, `//`, `%`, `&`, `|`, `^`, `<<`, `>>`. В Python нет операторов инкремента (`++`) или декремента (`--`), встречающихся в других языках программирования.

1.5. УСЛОВНЫЕ КОМАНДЫ И УПРАВЛЕНИЕ ПРОГРАММНОЙ ЛОГИКОЙ

Команды `while`, `if` и `else` используются для повторения и условного выполнения кода:

```
if a < b:
    print('Computer says Yes')
else:
    print('Computer says No')
```

Тела операторов `if` и `else` обозначаются отступами. Наличие оператора `else` необязательно. Для создания пустой секции используйте команду `pass`:

```
if a < b:
    pass # Ничего не делать
else:
    print('Computer says No')
```

Для реализации выбора со многими вариантами предназначена команда `elif`:

```
if suffix == '.htm':
    content = 'text/html'
elif suffix == '.jpg':
    content = 'image/jpeg'
```

```
elif suffix == '.png':
    content = 'image/png'
else:
    raise RuntimeError(f'Unknown content type {suffix!r}')
```

Если значение присваивается по результатам проверки условия, используйте условное выражение:

```
maxval = a if a > b else b
```

Это то же, но короче:

```
if a > b:
    maxval = a
else:
    maxval = b
```

Иногда вы можете увидеть назначение переменной и условного оператора, объединенных с помощью оператора `:=`. Это называется выражением присваивания (или в просторечии моржом, потому что `:=` напоминает голову моржа, повернутую на 90°). Например:

```
x = 0
while (x := x + 1) < 10: # Выводит 1, 2, 3, ..., 9
    print(x)
```

Круглые скобки, в которые заключено выражение присваивания, обязательны.

Команда `break` может использоваться для преждевременного прерывания цикла. Она работает только в цикле с наибольшим уровнем вложенности. Пример:

```
x = 0
while x < 10:
    if x == 5:
        break   # Прерывает цикл, переходит к выводу Done
    print(x)
    x += 1

print('Done')
```

Команда `continue` пропускает остаток тела цикла и возвращает управление к началу цикла. Пример:

```
x = 0
while x < 10:
    x += 1
```

```
    if x == 5:
        continue   # Пропустить print(x), вернуться к началу цикла
    print(x)

print('Done')
```

1.6. СТРОКИ

Для определения строкового литерала заключите его в одинарные, двойные или тройные кавычки:

```
a = 'Hello World'
b = "Python is groovy"
c = '''Computer says no.'''
d = """Computer still says no."""
```

Для завершения и открытия строки должны использоваться одинаковые кавычки. Строки в тройных кавычках захватывают весь текст до завершающей тройной кавычки, в отличие от строк в одинарных и двойных кавычках, которые должны быть указаны в одной логической строке. Строки в тройных кавычках полезны, когда содержимое строкового литерала занимает несколько строк текста:

```
print('''Content-type: text/html

<h1> Hello World </h1>
Click <a href="http://www.python.org">here</a>.
''')
```

Строковые литералы, следующие друг за другом, объединяются в одну строку. Так, предыдущий пример можно было записать в следующем виде:

```
print(
'Content-type: text/html\n'
'\n'
'<h1> Hello World </h1>\n'
'Click <a href="http://www.python.org">here</a>\n'
)
```

Если перед открывающей кавычкой строки находится префикс `f`, то в строке выполняется вычисление и подстановка экранированных выражений. Например, в прошлом примере для вывода результатов вычислений использовалась следующая команда:

```
print(f'{year:>3d} {principal:0.2f}')
```

И хотя здесь в строку включаются простые имена переменных, в ней могут находиться любые допустимые выражения:

```
base_year = 2020
...
print(f'{base_year + year:>4d} {principal:0.2f}')
```

Как альтернатива f-строкам для форматирования строк иногда используются метод `format()` и оператор `%`:

```
print('{0:>3d} {1:0.2f}'.format(year, principal))
print('%3d %0.2f' % (year, principal))
```

Подробнее форматирование строк рассматривается в главе 9.

Строки хранятся в виде последовательностей символов «Юникода», которые индексируются целыми числами, начиная с 0. Отрицательные индексы отсчитываются от конца строки. Длина строки `s` вычисляется функцией `len(s)`. Чтобы извлечь из строки один символ, используйте оператор индексирования `s[i]`, где `i` — индекс.

```
a = 'Hello World'
print(len(a))            # 11
b = a[4]                 # b = 'o'
c = a[-1]                # c = 'd'
```

Для извлечения подстроки используется оператор сегментации `s[i:j]`. Он извлекает из `s` все символы, индекс `k` которых лежит в диапазоне `i <= k < j`. Если один из индексов опущен, по умолчанию используется начало или конец строки соответственно:

```
c = a[:5]                # c = 'Hello'
d = a[6:]                # d = 'World'
e = a[3:8]               # e = 'lo Wo'
f = a[-5:]               # f = 'World'
```

Строки поддерживают разные методы для выполнения операций с их содержимым. Например, `replace()` выполняет простую замену текста:

```
g = a.replace('Hello', 'Hello Cruel') # f = 'Hello Cruel World'
```

В табл. 1.6 перечислены некоторые распространенные методы строк. Здесь и далее аргументы в квадратных скобках необязательны.

Конкатенация строк выполняется оператором `+`:

```
g = a + 'ly'     # g = 'Hello Worldly'
```

Python никогда не интерпретирует содержимое строки как числовые данные. Поэтому + всегда выполняет конкатенацию строк:

```
x = '37'
y = '42'
z = x + y       # z = '3742' (конкатенация строк)
```

Таблица 1.6. Распространенные методы строк

МЕТОД	ОПИСАНИЕ
`s.endswith(prefix [,start [,end]])`	Проверяет, завершается ли строка подстрокой `prefix`
`s.find(sub [, start [,end]])`	Находит первое вхождение заданной подстроки `sub` или `-1`, если строка не найдена
`s.lower()`	Преобразует строку к нижнему регистру
`s.replace(old, new [,maxreplace])`	Заменяет подстроку
`s.split([sep [,maxsplit]])`	Разбивает строку по разделителю `sep`. `maxsplit` – максимальное количество выполняемых разбиений
`s.startswith(prefix [,start [,end]])`	Проверяет, начинается ли строка с префикса `prefix`
`s.strip([chrs])`	Удаляет начальные и конечные пропуски/символы, переданные в `chrs`
`s.upper()`	Преобразует строку в верхний регистр

Для математических вычислений строку `first` нужно сначала преобразовать в числовое значение функцией `int()` или `float()`:

```
z = int(x) + int(y)     # z = 79 (целочисленное сложение)
```

Для преобразования нестроковых значений в строковое представление можно воспользоваться функциями `str()`, `repr()` или `format()`:

```
s = 'The value of x is ' + str(x)
s = 'The value of x is ' + repr(x)
s = 'The value of x is ' + format(x, '4d')
```

И хотя обе функции, `str()` и `repr()`, создают строки, их вывод часто различается. `str()` выдает результат, получаемый при использовании функции `print()`, а `repr()` создает строку, которая вводится в программе для точного представления значения объекта. Например:

```
>>> s = 'hello\nworld'
>>> print(str(s))
hello
world
>>> print(repr(s))
'hello\nworld'
>>>
```

В процессе отладки для вывода обычно используется функция `repr(s)`. Она выводит больше информации о значении и его типе.

Функция `format()` преобразует одно значение в строку с применением определенного форматирования:

```
>>> x = 12.34567
>>> format(x, '0.2f')
'12.35'
>>>
```

Функции `format()` передаются те же коды форматирования, что используются с f-строками для получения отформатированного вывода. Например, предыдущий код можно заменить таким:

```
>>> f'{x:0.2f}'
'12.35'
>>>
```

1.7. ФАЙЛОВЫЙ ВВОД И ВЫВОД

Следующая программа открывает файл и построчно читает его содержимое:

```
with open('data.txt') as file:
    for line in file:
        print(line, end='')  # end='' опускает лишний символ новой строки
```

Функция `open()` возвращает новый объект файла. Команда `with`, предшествующая открытию файла, объявляет блок команд (или контекст), где будет использоваться файл (`file`). При выходе управления за его пределы файл автоматически закрывается. Без команды `with` код должен выглядеть примерно так:

```
file = open('data.txt')
for line in file:
    print(line, end='')  # end='' опускает лишний символ новой строки
file.close()
```

О вызове `close()` легко забыть. Лучше использовать команду `with`, которая закроет файл за вас.