ГЛАВА 5

Числа

Язык Python 3 поддерживает следующие числовые типы:

- `int` — целые числа. Размер числа ограничен лишь объемом оперативной памяти;
- `float` — вещественные числа;
- `complex` — комплексные числа.

Операции над числами разных типов возвращают число, имеющее более сложный тип из типов, участвующих в операции. Целые числа имеют самый простой тип, далее идут вещественные числа и самый сложный тип — комплексные числа. Таким образом, если в операции участвуют целое число и вещественное, то целое число будет автоматически преобразовано в вещественное число, а затем произведена операция над вещественными числами. Результатом этой операции будет вещественное число.

Создать объект целочисленного типа можно обычным способом:

```
>>> x = 0; y = 10; z = -80
>>> x, y, z
(0, 10, -80)
```

Кроме того, можно указать число в двоичной, восьмеричной или шестнадцатеричной форме. Такие числа будут автоматически преобразованы в десятичные целые числа.

- Двоичные числа начинаются с комбинации символов `0b` (или `0B`) и содержат цифры от `0` или `1`:

```
>>> 0b11111111, 0b101101
(255, 45)
```

- Восьмеричные числа начинаются с нуля и следующей за ним латинской буквы `o` (регистр не имеет значения) и содержат цифры от `0` до `7`:

```
>>> 0o7, 0o12, 0o777, 0O7, 0O12, 0O777
(7, 10, 511, 7, 10, 511)
```

- Шестнадцатеричные числа начинаются с комбинации символов `0x` (или `0X`) и могут содержать цифры от `0` до `9` и буквы от `A` до `F` (регистр букв не имеет значения):

```
>>> 0X9,0xA, 0x10, 0xFFF, 0xfff
(9, 10, 16, 4095, 4095)
```

- Вещественное число может содержать точку и (или) быть представлено в экспоненциальной форме с буквой `E` (регистр не имеет значения):

При выполнении операций над вещественными числами следует учитывать ограничения точности вычислений. Например, результат следующей операции может показаться странным:

```
>>> 0.3 - 0.1 - 0.1 - 0.1
-2.7755575615628914e-17
```

Ожидаемым был бы результат `0.0`, но, как видно из примера, мы получили совсем другое значение. Если необходимо производить операции с фиксированной точностью, то следует использовать модуль `decimal`:

```
>>> from decimal import Decimal
>>> Decimal("0.3") - Decimal("0.1") - Decimal("0.1") - Decimal("0.1")
Decimal('0.0')
```

Кроме того, можно использовать дроби, поддержка которых содержится в модуле `fractions`. При создании дроби можно как указать два числа: числитель и знаменатель, так и одно число или строку, содержащую число, которое будет преобразовано в дробь.

Для примера создадим несколько дробей. Вот так формируется дробь $^4/_5$:

```
>>> from fractions import Fraction
>>> Fraction(4, 5)
Fraction(4, 5)
```

А вот так — дробь $^1/_2$, причем можно сделать это тремя способами:

```
>>> Fraction(1, 2)
Fraction(1, 2)
>>> Fraction("0.5")
Fraction(1, 2)
>>> Fraction(0.5)
Fraction(1, 2)
```

Над дробями можно производить арифметические операции, как и над обычными числами:

```
>>> Fraction(9, 5) - Fraction(2, 3)
Fraction(17, 15)
>>> Fraction("0.3") - Fraction("0.1") - Fraction("0.1") - Fraction("0.1")
Fraction(0, 1)
>>> float(Fraction(0, 1))
0.0
```

Комплексные числа записываются в формате:

```
<Вещественная часть>+<Мнимая часть>J
```

Здесь буква `J` может стоять в любом регистре. Примеры комплексных чисел:

```
>>> 2+5J, 8j
((2+5j), 8j)
```

Подробное рассмотрение модулей `decimal` и `fractions`, а также комплексных чисел выходит за рамки нашей книги. За подробной информацией обращайтесь к соответствующей документации.

5.1. Встроенные функции и методы для работы с числами

Для работы с числами предназначены следующие встроенные функции:

- `int([<Объект>[, <Система счисления>]])` — преобразует объект в целое число. Во втором параметре можно указать систему счисления преобразуемого числа (значение по умолчанию `10`). Пример:

```
>>> int(7.5), int("71", 10), int("0o71", 8), int("0xA", 16)
(7, 71, 57, 10)
>>> int(), int("0b11111111", 2)
(0, 255)
```

- `float([<Число или строка>])` — преобразует целое число или строку в вещественное число:

```
>>> float(7), float("7.1"), float("12.")
(7.0, 7.1, 12.0)
>>> float("inf"), float("-Infinity"), float("nan")
(inf, -inf, nan)
>>> float()
0.0
```

- `bin(<Число>)` — преобразует десятичное число в двоичное. Возвращает строковое представление числа. Пример:

```
>>> bin(255), bin(1), bin(-45)
('0b11111111', '0b1', '-0b101101')
```

- `oct(<Число>)` — преобразует десятичное число в восьмеричное. Возвращает строковое представление числа. Пример:

```
>>> oct(7), oct(8), oct(64)
('0o7', '0o10', '0o100')
```

- `hex(<Число>)` — преобразует десятичное число в шестнадцатеричное. Возвращает строковое представление числа. Пример:

```
>>> hex(10), hex(16), hex(255)
('0xa', '0x10', '0xff')
```

- `round(<Число>[, <Количество знаков после точки>])` — для чисел с дробной частью меньше `0.5` возвращает число, округленное до ближайшего меньшего целого, а для чисел с дробной частью больше `0.5` возвращает число, округленное до ближайшего большего целого. Если дробная часть равна `0.5`, то округление производится до ближайшего четного числа. Пример:

```
>>> round(0.49), round(0.50), round(0.51)
(0, 0, 1)
>>> round(1.49), round(1.50), round(1.51)
(1, 2, 2)
>>> round(2.49), round(2.50), round(2.51)
(2, 2, 3)
>>> round(3.49), round(3.50), round(3.51)
(3, 4, 4)
```

Во втором параметре можно указать желаемое количество знаков после запятой. Если оно не указано, используется значение `0` (т. е. число будет округлено до целого):

```
>>> round(1.524, 2), round(1.525, 2), round(1.5555, 3)
(1.52, 1.52, 1.556)
```

- `abs(<Число>)` — возвращает абсолютное значение:

```
>>> abs(-10), abs(10), abs(-12.5)
(10, 10, 12.5)
```

- `pow(<Число>, <Степень>[, <Делитель>])` — возводит `<Число>` в `<Степень>`:

```
>>> pow(10, 2), 10 ** 2, pow(3, 3), 3 ** 3
(100, 100, 27, 27)
```

Если указан третий параметр, то возвращается остаток от деления полученного результата на значение этого параметра:

```
>>> pow(10, 2, 2), (10 ** 2) % 2, pow(3, 3, 2), (3 ** 3) % 2
(0, 0, 1, 1)
```

- `max(<Список чисел через запятую>)` — максимальное значение из списка:

```
>>> max(1, 2, 3), max(3, 2, 3, 1), max(1, 1.0), max(1.0, 1)
(3, 3, 1, 1.0)
```

- `min(<Список чисел через запятую>)` — минимальное значение из списка:

```
>>> min(1, 2, 3), min(3, 2, 3, 1), min(1, 1.0), min(1.0, 1)
(1, 1, 1, 1.0)
```

- `sum(<Последовательность>[, <Начальное значение>])` — возвращает сумму значений элементов последовательности (например: списка, кортежа) плюс `<Начальное значение>`. Если второй параметр не указан, начальное значение принимается равным `0`. Если последовательность пустая, то возвращается значение второго параметра. Примеры:

```
>>> sum((10, 20, 30, 40)), sum([10, 20, 30, 40])
(100, 100)
>>> sum([10, 20, 30, 40], 2), sum([], 2)
(102, 2)
```

- `divmod(x, y)` — возвращает кортеж из двух значений `(x // y, x % y)`:

```
>>> divmod(13, 2)              # 13 == 6 * 2 + 1
(6, 1)
>>> 13 // 2, 13 % 2
(6, 1)
>>> divmod(13.5, 2.0)          # 13.5 == 6.0 * 2.0 + 1.5
(6.0, 1.5)
>>> 13.5 // 2.0, 13.5 % 2.0
(6.0, 1.5)
```

Следует понимать, что все типы данных, поддерживаемые Python, представляют собой *классы*. Класс `float`, представляющий вещественные числа, поддерживает следующие полезные *методы*:

- `is_integer()` — возвращает `True`, если заданное вещественное число не содержит дробной части, т. е. фактически представляет собой целое число:

```
>>> (2.0).is_integer()
True
>>> (2.3).is_integer()
False
```

- as_integer_ratio() — возвращает кортеж из двух целых чисел, представляющих собой числитель и знаменатель дроби, которая соответствует заданному числу:

```
>>> (0.5).as_integer_ratio()
(1, 2)
>>> (2.3).as_integer_ratio()
(2589569785738035, 1125899906842624)
```

5.2. Модуль *math*. Математические функции

Модуль math предоставляет дополнительные функции для работы с числами, а также стандартные константы. Прежде чем использовать модуль, необходимо подключить его с помощью инструкции:

```
import math
```

> ***Примечание***
>
> Для работы с комплексными числами необходимо использовать модуль cmath.

Модуль math предоставляет следующие стандартные константы:

- pi — возвращает число π:

```
>>> import math
>>> math.pi
3.141592653589793
```

- e — возвращает значение константы *e*:

```
>>> math.e
2.718281828459045
```

Перечислим основные функции для работы с числами:

- sin(), cos(), tan() — стандартные тригонометрические функции (синус, косинус, тангенс). Значение указывается в радианах;
- asin(), acos(), atan() — обратные тригонометрические функции (арксинус, арккосинус, арктангенс). Значение возвращается в радианах;
- degrees() — преобразует радианы в градусы:

```
>>> math.degrees(math.pi)
180.0
```

- radians() — преобразует градусы в радианы:

```
>>> math.radians(180.0)
3.141592653589793
```

- exp() — экспонента;
- log(<Число>[, <База>]) — логарифм по заданной базе. Если база не указана, вычисляется натуральный логарифм (по базе *e*);

- `log10()` — **десятичный логарифм;**
- `log2()` — **логарифм по базе 2;**
- `sqrt()` — **квадратный корень:**

```
>>> math.sqrt(100), math.sqrt(25)
(10.0, 5.0)
```

- `ceil()` — **значение, округленное до ближайшего большего целого:**

```
>>> math.ceil(5.49), math.ceil(5.50), math.ceil(5.51)
(6, 6, 6)
```

- `floor()` — **значение, округленное до ближайшего меньшего целого:**

```
>>> math.floor(5.49), math.floor(5.50), math.floor(5.51)
(5, 5, 5)
```

- `pow(<Число>, <Степень>)` — **возводит** `<Число>` **в** `<Степень>`:

```
>>> math.pow(10, 2), 10 ** 2, math.pow(3, 3), 3 ** 3
(100.0, 100, 27.0, 27)
```

- `fabs()` — **абсолютное значение:**

```
>>> math.fabs(10), math.fabs(-10), math.fabs(-12.5)
(10.0, 10.0, 12.5)
```

- `fmod()` — **остаток от деления:**

```
>>> math.fmod(10, 5), 10 % 5, math.fmod(10, 3), 10 % 3
(0.0, 0, 1.0, 1)
```

- `factorial()` — **факториал числа:**

```
>>> math.factorial(5), math.factorial(6)
(120, 720)
```

- `fsum(<Список чисел>)` — **возвращает точную сумму чисел из заданного списка:**

```
>>> sum([.1, .1, .1, .1, .1, .1, .1, .1, .1, .1])
0.9999999999999999
>>> fsum([.1, .1, .1, .1, .1, .1, .1, .1, .1, .1])
1.0
```

> ***Примечание***
>
> **В этом разделе мы рассмотрели только основные функции. Чтобы получить полный список функций, обращайтесь к документации по модулю** `math`.

5.3. Модуль *random*. Генерация случайных чисел

Модуль `random` **позволяет генерировать случайные числа. Прежде чем использовать модуль, необходимо подключить его с помощью инструкции:**

```
import random
```

Перечислим основные его функции:

- `random()` — **возвращает псевдослучайное число от** `0.0` **до** `1.0`:

```
>>> import random
>>> random.random()
0.9753144027290991
>>> random.random()
0.5468390487484339
>>> random.random()
0.13015058054767736
```

- `seed([<Параметр>][, version=2])` — **настраивает генератор случайных чисел на новую последовательность. Если первый параметр не указан, в качестве базы для случайных чисел будет использовано системное время. Если значение первого параметра будет одинаковым, то генерируется одинаковое число:**

```
>>> random.seed(10)
>>> random.random()
0.5714025946899135
>>> random.seed(10)
>>> random.random()
0.5714025946899135
```

- `uniform(<Начало>, <Конец>)` — **возвращает псевдослучайное вещественное число в диапазоне от** `<Начало>` **до** `<Конец>`:

```
>>> random.uniform(0, 10)
9.965569925394552
>>> random.uniform(0, 10)
0.4455638245043303
```

- `randint(<Начало>, <Конец>)` — **возвращает псевдослучайное целое число в диапазоне от** `<Начало>` **до** `<Конец>`:

```
>>> random.randint(0, 10)
4
>>> random.randint(0, 10)
10
```

- `randrange([<Начало>, ]<Конец>[, <Шаг>])` — **возвращает случайный элемент из числовой последовательности. Параметры аналогичны параметрам функции** `range()`. **Можно сказать, что функция** `randrange` **«за кадром» создает диапазон, из которого и будут выбираться возвращаемые случайные числа:**

```
>>> random.randrange(10)
5
>>> random.randrange(0, 10)
2
>>> random.randrange(0, 10, 2)
6
```

- `choice(<Последовательность>)` — **возвращает случайный элемент из заданной последовательности (строки, списка, кортежа):**

```
>>> random.choice("string")          # Строка
'i'
```

```
>>> random.choice(["s", "t", "r"])    # Список
'r'
>>> random.choice(("s", "t", "r"))    # Кортеж
't'
```

- `shuffle(<Список>[, <Число от 0.0 до 1.0>])` — **перемешивает элементы списка случайным образом. Если второй параметр не указан, то используется значение, возвращаемое функцией** `random()`. **Никакого результата при этом не возвращается. Пример:**

```
>>> arr = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
>>> random.shuffle(arr)
>>> arr
[8, 6, 9, 5, 3, 7, 2, 4, 10, 1]
```

- `sample(<Последовательность>, <Количество элементов>)` — **возвращает список из указанного количества элементов, которые будут выбраны случайным образом из заданной последовательности. В качестве таковой можно указать любые объекты, поддерживающие итерации. Примеры:**

```
>>> random.sample("string", 2)
['i', 'r']
>>> arr = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
>>> random.sample(arr, 2)
[7, 10]
>>> arr # Сам список не изменяется
[1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
>>> random.sample((1, 2, 3, 4, 5, 6, 7), 3)
[6, 3, 5]
>>> random.sample(range(300), 5)
[126, 194, 272, 46, 71]
```

Для примера создадим генератор паролей произвольной длины (листинг 5.1). Для этого добавляем в список `arr` **все разрешенные символы, а далее в цикле получаем случайный элемент с помощью функции** `choice()`. **По умолчанию будет выдаваться пароль из 8 символов.**

Листинг 5.1. Генератор паролей

```
# -*- coding: utf-8 -*-
import random # Подключаем модуль random
def passw_generator(count_char=8):
    arr = ['a','b','c','d','e','f','g','h','i','j','k','l','m',
    'n','o','p','q','r','s','t','u','v','w','x','y','z',
    'A','B','C','D','E','F','G','H','I','J','K','L',
    'M','N','O','P','Q','R','S','T','U','V', 'W',
    'X','Y','Z','1','2','3','4','5','6','7','8','9','0']
    passw = []
    for i in range(count_char):
        passw.append(random.choice(arr))
    return "".join(passw)

# Вызываем функцию
print( passw_generator(10) )  # Выведет что-то вроде ngODHE8J8x
print( passw_generator() )    # Выведет что-то вроде ZxcpkF5O
input()
```