

Числовые типы, None, арифметические операции

Знание базы работы с числовыми типами в Python — это фундамент для разработки практически всех программных приложений.

Числовые типы

В Python выделяют разные типы чисел, они объединены в группу Numeric:

- целые числа — **int** (от англ. integer, «целое число»);
- числа с «плавающей запятой» (десятичные дроби) — **float** (от англ. float number, «число с плавающей точкой»);
- комплексные числа — **complex** (от англ. complex number, «комплексное число»).

Целые числа

Целые числа относятся к классу **int** (от англ. integer — целое число). Это все положительные, отрицательные числа без дробной части и ноль.

Целые числа можно записывать, разделяя разряды знаком «_»: 432246123 — 432_246_123.

```
x = 231
z = -100
r = 432_246_123
```

Числа с плавающей точкой

Числа с «плавающей точкой» относятся к типу **float** (от англ. float number — число с плавающей точкой). Это все десятичные дроби; целая часть от дробной отделяется точкой.

```
b = 2.0
c = 0.07
# Сокращённый синтаксис.
x = 2.
y = .07
```

Как не потерять точность при вычислениях

```
input_data_1 = 3.3
input_data_2 = 4.18
# Чтобы дробная часть всех слагаемых стала равна нулю —
# каждое слагаемое умножаем на один и тот же множитель:
input_data = input_data_1*100 + input_data_2*100

# После получения результата возвращаем всё как было:
input_data /= 100
print(input_data)

# В терминал выведется:
# 7.48
```

None

Тип `None` используется, когда:

- нужно указать, что переменная или значение функции не определено или его вовсе нет;
- нужно задать значение по умолчанию для аргументов функции, которые не обязательно должны быть переданы при вызове функции.

```
a = None
print(type(a))
# Вывод в терминал: <class 'NoneType'>

# Переопределим переменную:
a = 1
print(type(a))
# Вывод в терминал: <class 'int'>
```



`None` не используется при объявлении переменной, которой не нужно присваивать значение. В Python переменные не нужно объявлять без присваивания им значений. Вместо этого, переменные создаются при присваивании им значений.

Арифметические операции

Сложение и вычитание

```
# Согласно правилам оформления кода PEP8 операторы отделяются пробелами.
# Отсутствие пробелов не приведёт к ошибке, но правила лучше соблюдать.

x = 40 - 11
print(x)

# Вывод в терминал: 29
```

Инкрементирование — операция пошагового увеличения значения переменной на определённое число. Декрементирование — пошаговое уменьшение значения.

В Python инкрементирование можно записать через комбинированный оператор присваивания:

```
# Выражение...
steps = steps + 1
# ...можно записать через комбинированный оператор присваивания:
steps += 1
```

Комбинированные операторы присваивания применимы для всех арифметических операций: `-=`, `*=`, `/=` и других.

Умножение

```
x = 5 * 2
print(x)
# Вывод в терминал: 10
```

Возведение в степень

```
x = 5 ** 2
print(x)
# Вывод в терминал: 25
```

Целочисленное деление

```
x = 22 // 2
print(x)
# Вывод в терминал: 11, ничего неожиданного (делится ровно, остатка нет).

y = 22 // 3
print(y)
# Вывод в терминал: 7
# Это результат деления с остатком.
# Возвращается только неполное частное.

z = 2 // 3
print(z)
# Вывод в терминал: 0
# Неполное частное – 0, остаток – 2, но он не возвращается.
```

Получение остатка от деления: оператор «модуло»

```
x = 11 % 3
print(x)
# Вывод в терминал: 2
```

Деление

```
x = 20 / 2
print(x)
# Вывод в терминал: 10.0

x = 11 / 3
print(x)
# Вывод в терминал: 3.6666666666666665
```

Модуль Decimal

Модуль **Decimal** даёт почти стопроцентную точность при работе с дробями.

Для объявления таких переменных нужно:

- импортировать в код класс `Decimal`;
- при объявлении переменной указать этот класс;
- передать в конструктор класса значение числа в кавычках — в виде строки:

```
from decimal import Decimal

x = Decimal('3.3')
print(type(x))

# Вывод в терминал: <class 'decimal.Decimal'>
```

Приоритеты арифметических операций

Приоритеты при выполнении арифметических операторов в Python точно такие же, как в математике:

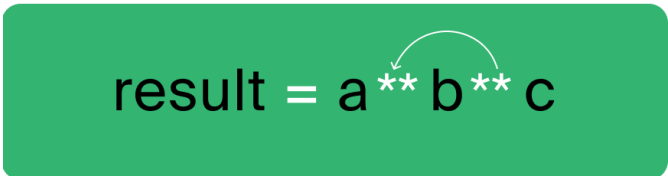
1. Самый приоритетный арифметический оператор — возведение в степень.
2. Следующими идут операторы умножения, деления, получения остатка и целочисленное деление.
3. Самые низкоприоритетные — операторы сложения и вычитания.

При равном уровне приоритетов операторы выполняются по очереди, слева направо:



`sum = a + b + c + d`

Если рядом находится две операции возведения в степень, то последовательность их выполнения будет обратной — справа налево:



`result = a ** b ** c`

Чтобы изменить приоритетность выполнения операций — применяют скобки. Операции в скобках выполняются в первую очередь.

```
print(1 + 2 * 3)
# Вывод в терминал: 7
# Но
print((1 + 2) * 3)
# Вывод в терминал: 9
```

Полезные ресурсы

[Статья в Википедии о комплексных числах](#)

[Комплексные числа в документации Python](#)