Занятие 2. Базовые типы: численные типы

1. Целые числа (int)

Команда input() считывает строку текста. Однако во многих случаях нам нужно работать именно с числами. Чтобы в Python создать переменную целого типа данных, нужно опустить кавычки при объявлении переменной. Рассмотрим следующий код:

In [1]:
```
num = 13
print(num)

num = 0
print(num)

num = -10
print(num)
```

```
13
0
-10
```

Начиная с Python 3.6 появилась возможность разделять порядки символом нижнего подчеркивания в длинных числах, тем самым мы можем улучшить читаемость этих чисел в исходном коде.

In [2]:
```
num = 100_000_000
print(num)
```

```
100000000
```

Стоит отметить, что Python поддерживает длинную арифметику при работе с целыми числами, то есть поддерживаются числа неограниченной длины.

Встроенная функция type

Функция type доступна в глобальном пространстве имен и она позволяет в рантайме, в процессе работы программы посмотреть на тип объекта

In [3]:
```
num = 13
print(type(num))
```

```
<class 'int'>
```

2. Вещественные числа (float)

Python умеет работать с вещественными числами, то есть с числами с плавающей точкой, это тип float. Точка используется для того чтобы разделить целую и дробную часть вещественного числа, то же самое для 0, и то же самое для отрицательного вещественного числа.

In [4]:
```
num = 13.4
print(num)

num = 0.0
print(num)

num = -10.2
print(num)
```

```
13.4
0.0
-10.2
```

По аналогии с типом int мы можем использовать символ нижнего подчёркивания, для того, чтобы разделять порядки в длинном вещественном числе

In [5]:
```
num = 100_000.000_001
print(num)
```

```
100000.000001
```

Также Python поддерживает экспоненциальную нотацию записи вещественного числа

In [6]:
```
# 1.5 умножить на 10 в степени 2
num = 1.5e2
print(num)
```

```
150.0
```

Конвертация типов:

Переменная num, связана с вещественным объектом со значением 150,2, имеет тип float. В любой момент мы можем воспользоваться встроенным классом int, чтобы переменную, которая связана с типом float, перевести в объект типа integer и затем преобразовать integer обратно к вещественному типу, используя класс float, тем самым в процессе работы программы мы можем из одного типа конвертировать переменные в другой тип.

In [7]:
```
num = 150.2
print(type(num))
```

```
<class 'float'>
```

In [8]:
```
num = int(num)
print(num, type(num))

num = float(num)
print(num, type(num))
```

```
150 <class 'int'>
150.0 <class 'float'>
```

Обратите внимание, что в процессе конвертаций, мы потеряли дробную часть вещественного числа.

3. Комплексные числа (complex)

Python умеет работать с комплексными числами, это тип complex, для того, чтобы определить комплексное число нужно для мнимой части использовать символ **j**,

In [9]:
```
num = 14 + 1j

print(type(num))
print(num.real)
print(num.imag)
```

```
<class 'complex'>
14.0
1.0
```

Чтобы достучаться до реальной и мнимой части комплексного числа, мы можем воспользоваться атрибутами real и imac.

В языке Python есть еще два модуля, которые позволяют работать с числами, эти модули

очень полезны, мы не будем рассматривать их подробно, но знать о них нужно. В первую очередь, это модуль **deccimal**, который позволяет работать с вещественными числами с фиксированной точностью, а второй модуль - это модуль **fractions**, который позволяет работать с рациональными числами, проще говоря с дробями.

4. Основные операции с числами

Сложение:

Сумма двух целых чисел дает нам целое число, если мы складываем целое число и вещественное число, то результат у нас получается вещественный

```
In [10]: 1 + 1
```

```
Out[10]: 2
```

```
In [11]: 1 + 2.0
```

```
Out[11]: 3.0
```

Вычитание:

То же самое с вычитанием, если мы вычитаем из целого целое, получается целое число, если мы вычитаем из вещественного целое, то результат вещественный

```
In [12]: 10 - 1
```

```
Out[12]: 9
```

```
In [13]: 4.2 - 1
```

```
Out[13]: 3.2
```

Деление:

Для того, чтобы разделить числа, мы можем воспользоваться символом прямого слэша, и обратите внимание, что результат деления всегда вещественный.

```
In [14]: 10 / 2
```

```
Out[14]: 5.0
```

Делить на 0 нельзя:

```
In [15]: 2 / 0
```

```
---------------------------------------------------------------------------
ZeroDivisionError                         Traceback (most recent call last)
<ipython-input-15-8b4ac6d3a3e1> in <module>
----> 1 2 / 0

ZeroDivisionError: division by zero
```

Если мы попытаемся разделить на 0, то мы получим исключение zerodivisionerror. Позже мы научимся их обрабатывать.

Умножение:

Для того чтобы умножить числа, мы используем звездочку, для возведения в степень - две

звездочки.

In [16]:
```
4 * 5.25
```

Out[16]:
```
21.0
```

Возведение в степень

In [17]:
```
2 ** 4
```

Out[17]:
```
16
```

Целочисленное деление:

In [18]:
```
10 // 3
```

Out[18]:
```
3
```

Остаток от деления:

In [19]:
```
10 % 3
```

Out[19]:
```
1
```

Порядок операций в выражениях с числами:

In [20]:
```
print(10 * 3 + 3)
print(10 * (3 + 3))
```

```
33
60
```

Побитовые операции:

In [21]:
```
x = 4
y = 3

print("Побитовое или:", x | y)
print("Побитовое исключающее или:", x ^ y)
print("Побитовое и:", x & y)
print("Битовый сдвиг влево:", x << 3)
print("Битовый сдвиг вправо:", x >> 1)
print("Инверсия битов:", ~x)
```

```
Побитовое или: 7
Побитовое исключающее или: 7
Побитовое и: 0
Битовый сдвиг влево: 32
Битовый сдвиг вправо: 2
Инверсия битов: -5
```

Задача: найти расстояние между двумя точками в декартовых координатах. Для того, чтобы ее решить, нам нужно найти гипотенузу прямоугольного треугольника.

In [22]:
```
x1, y1 = 0, 0
x2 = 3
y2 = 4

distance = ((x2 - x1) ** 2 + (y2 - y1) ** 2) ** 0.5
print(distance)
```

```
5.0
```

Меняем местами значения 2-х переменных:

Python позволяет поменять значения у 2 переменных без использования временной переменной

In [23]:
```
a = 100
b = 200
print(a, b)

a, b = b, a
print(a, b)
```

```
100 200
200 100
```

Вместо x, y = 0, 0

Примитивные основные типы, не изменяемы, поэтому запись x = y = 0 оправдана

In [24]:
```
x = y = 0
x += 1

print(x)
print(y)
```

```
1
0
```

Но нужно помнить об отличии изменяемых (mutable) и неизменяемых (immutable) типов:

In [27]:
```
x = y = []
x.append(1)
x.append(2)

print(x)
print(y)
```

```
[1, 2]
[1, 2]
```

Но если мы напишем x = y = пустой список , пустой список - это объект-контейнер, который позволяет хранить последовательность объектов в себе, этот объект уже изменяемый, поэтому, когда мы присваиваем x = y = пустой список, а затем в список x добавляем два элемента, то видим, что изменились значения обеих переменных.

In [ ]:
```
```