Числа в Python

В Python существует 4 вида чисел:

Целые числа (int)
Вещественные числа (float)
Комплексные числа (complex)
Десятичные дроби (decimal)

Приоритет выполнения операций:

Приоритет операций	Пример
В скобках	4*(9-2) = 28
**(возведение в степень)	2*4**3 = 128
Равный приоритет * / // %	3*4/2 = 6
Равный приоритет + -	17-2+5 = 20

Целые числа (int)

Целые числа в Python ничем не отличаются от обычных чисел. Они поддерживают набор самых обычных математических операций:

x + y	Сложение
x – y	Вычитание
x * y	Умножение
x / y	Деление
x // y	Получение целой части от деления
x % y	Остаток от деления
-x	Смена знака числа
abs(x)	Модуль числа
divmod(x, y)	Пара (x // y, x % y)
x ** y	Возведение в степень
pow(x, y[, z])	xy по модулю (если модуль задан)

Битовые операции

Над целыми числами также можно производить битовые операции.

x | y	Побитовое *или*
x ^ y	Побитовое *исключающее или*
x & y	Побитовое *и*
x << n	Битовый сдвиг влево
x >> y	Битовый сдвиг вправо
~x	Инверсия битов

Методы

int.bit_length() – количество бит, необходимых для представления числа в двоичном виде, без учёта знака и лидирующих нулей. Пример:

```
n = -37
print(bin(n))   # Выведет: '-0b100101
print(n.bit_length())   # Выведет: 6
```

float.is_integer() – является ли значение целым числом.

```
a = float.is_integer(4.0)
print(a)
```

Выведет: True

```
a = float.is_integer(4.1)
print(a)
```

Выведет: False

float.hex() – переводит float в hex (шестнадцатеричную систему счисления).

```
a = float.hex(4.0)
print(a)
```

Выведет: 0x1.0000000000000p+2

float.fromhex(s) – float из шестнадцатеричной строки.

```
a = float.fromhex("0x1.0000000000000p+2")
print(a)
```

Выведет: 4.0

int.to_bytes(length, byteorder, *, signed=False) – возвращает строку байтов, представляющих это число. Пример:

```
print((1024).to_bytes(2, byteorder='big'))  # Выведет:
b'\x04\x00'

print((1024).to_bytes(10, byteorder='big'))  # Выведет:
b'\x00\x00\x00\x00\x00\x00\x00\x00\x04\x00'

print((-1024).to_bytes(10, byteorder='big', signed=True))
# Выведет: b'\xff\xff\xff\xff\xff\xff\xff\xff\xfc\x00'

x = 1000
print(x.to_bytes((x.bit_length() // 8) + 1,
byteorder='little'))  # Выведет: b'\xe8\x03'
```

classmethod int.from_bytes(bytes, byteorder, *, signed=False) – возвращает число из данной строки байтов. Пример:

```
print(int.from_bytes(b'\x00\x10', byteorder='big'))  # Выведет: 16

print(int.from_bytes(b'\x00\x10', byteorder='little'))  # Выведет:
4096

print(int.from_bytes(b'\xfc\x00', byteorder='big', signed=True))  #
Выведет: -1024

print(int.from_bytes(b'\xfc\x00', byteorder='big', signed=False))  #
Выведет: 64512

print(int.from_bytes([255, 0, 0], byteorder='big'))  # Выведет:
16711680
```

Вещественные числа (float)

Вещественные числа поддерживают те же операции, что и целые.

Но вещественные числа неточны, и это может привести к ошибкам:

```
print(0.1 + 0.1 + 0.1 + 0.1 + 0.1 + 0.1 + 0.1 + 0.1 + 0.1 +
0.1 + 0.1)
# Выведет: 0.9999999999999999)
```

Вещественные числа не поддерживают длинную арифметику:

```
a = 3 ** 1000
print(a)   # выведет
1322070819480806636890455259752...
a + 0.1
print(a)   # Выведет ошибку
```

Также для работы с числами в Python есть несколько полезных модулей.

Модуль math предоставляет более сложные математические функции. Например:

```
import math
print(math.pi)
# Выведет: 3.141592653589793

import math
print(math.sqrt(85))
# Выведет: 9.219544457292887
```

Модуль random реализует генератор случайных чисел и функции случайного выбора. Например:

```
import random
print(random.random())
```

Выведет: 0.15651968855132303

```
import random
a = random.randint(1,100)
print(a)
```

Выведет: случайные числа от 1 до 100

Комплексные числа (complex)
Примеры работы комплексных чисел в Python:

```
x = complex(1, 2)
print(x)  # Выведет: (1+2j)

y = complex(3, 4)
print(y)  # Выведет: (3+4j)

z = x + y
print(x)    # Выведет: (1+2j)

print(z) # Выведет: (4+6j)

 z = x * y
print(z)  # Выведет: (-5+10j)

z = x / y
print(z) # Выведет: (0.44+0.08j)

print(x.conjugate()) # Сопряжённое число. Выведет: (1-2j)
```

```
x = complex(1, 2)

print(x.imag) # Мнимая часть
# Выведет: 2.0

print(x.real) # Действительная часть
# Выведет: 1.0

print(x > y)  # Комплексные числа нельзя сравнить. Выведет ошибку

print(abs(3 + 4j)) # Модуль комплексного числа
# Выведет: 5.0

 print(pow(3 + 4j, 2)) # Возведение в степень
# Выведет: (-7+24j)
```

Также в Python существует интересный модуль под названием cmath. Данный модуль предоставляет функции для работы с комплексными числами:

```
import cmath
x = complex(1, 2)
print(cmath.phase(x))
```

- cmath.polar(x) – преобразование к полярным координатам. Возвращает пару (r, phi).
- cmath.rect(r, phi) – преобразование из полярных координат.
- cmath.exp(x) – e^x.
- cmath.log(x[, base]) – логарифм x по основанию base. Если base не указан, возвращается натуральный логарифм.
- cmath.log10(x) – десятичный логарифм.
- cmath.sqrt(x) – квадратный корень из x.

- cmath.acos(x) – арккосинус x.
- cmath.asin(x) – арксинус x.
- cmath.atan(x) – арктангенс x.
- cmath.cos(x) – косинус x.
- cmath.sin(x) – синус x.
- cmath.tan(x) – тангенс x.
- cmath.acosh(x) – гиперболический арккосинус x.
- cmath.asinh(x) – гиперболический арксинус x.
- cmath.atanh(x) – гиперболический арктангенс x.
- cmath.cosh(x) – гиперболический косинус x.
- cmath.sinh(x) – гиперболический синус x.

cmath.tanh(x) – гиперболический тангенс x.
cmath.isfinite(x) – True, если действительная и мнимая части конечны.
cmath.isinf(x) – True, если либо действительная, либо мнимая часть бесконечна.
cmath.isnan(x) – True, если либо действительная, либо мнимая часть NaN.
cmath.pi – π.
cmath.e – e.

Десятичные дроби

Числа данного типа позволяют производить вычисления над десятичными дробями с заданной точностью. Примеры:

```
 from decimal import *
getcontext().prec = 10 # число знаков после
запятой
a = Decimal(13) / Decimal(17)
print(a)
```

выведет: 0.7647058824

```
# квадратный корень из 3
from decimal import *
getcontext().prec = 10
a = Decimal(3).sqrt()
print(a)
input()
```

1.732050808

```
# корень 7-й степени
from decimal import *
getcontext().prec = 10
a = Decimal(3)**Decimal(1/7)
print(a)
input()
```

1.169930813

```
# натуральный логарифм
from decimal import *
getcontext().prec = 10
a = Decimal(3).ln()
print(a)
input()
```

1.098612289