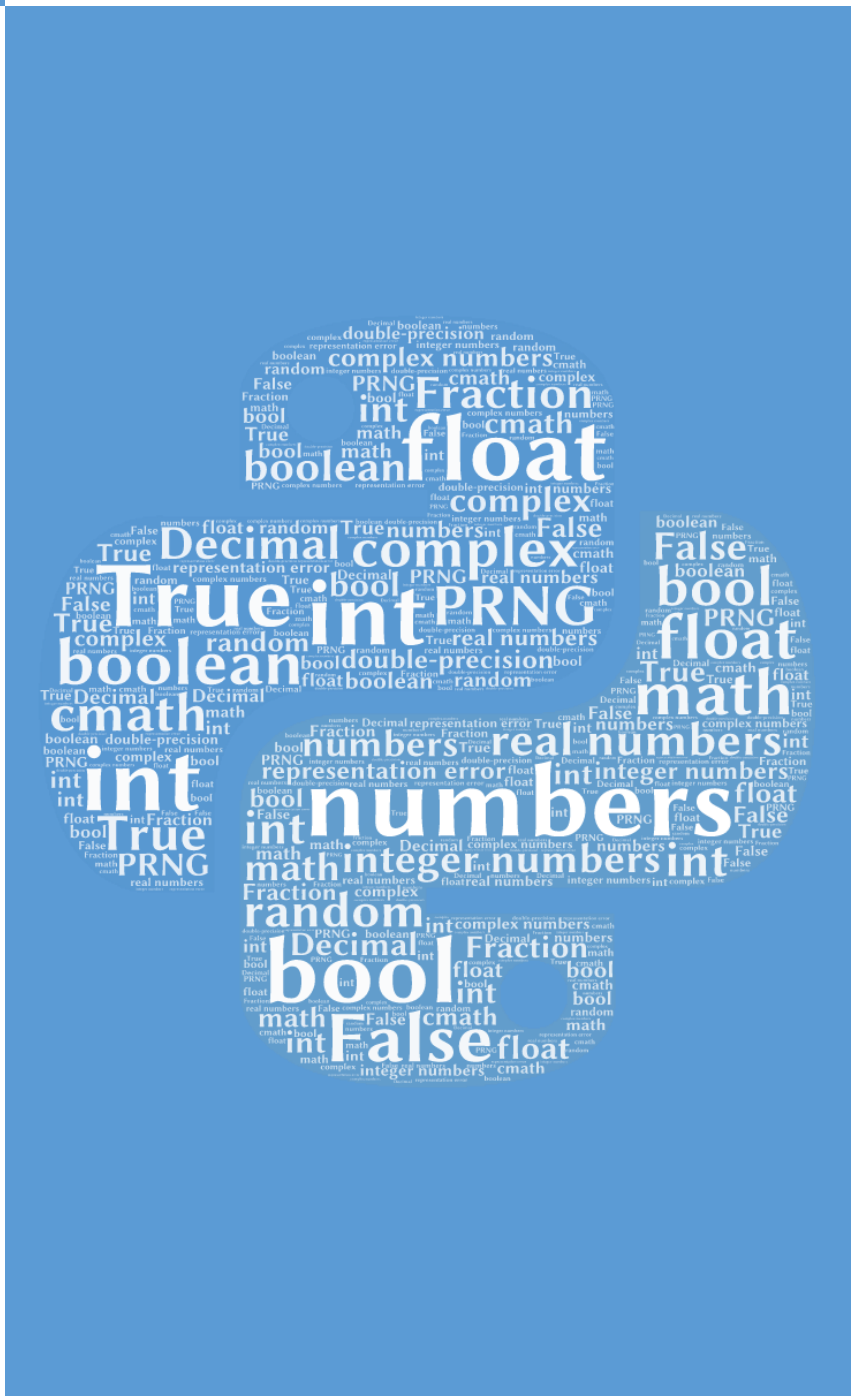
Университет «КРОК»
Кафедра компьютерных наук
Алгоритмизация и программирование. Лекции.
Автор: к.т.н., доцент Тимчук О.С.

6 ЧИСЛА В PYTHON



6

Числа в Python

6.1 Встроенные числовые типы данных

В Python выделено 4 встроенных числовых типа данных для представления целых чисел, вещественных чисел, комплексных чисел и логических значений. Рассмотрим основные особенности встроенных числовых типов данных.

1. Числовые объекты создаются с помощью числовых литералов, как результат выполнения некоторой функции / операции или с помощью конструкторов int( ), float( ), complex( ), bool( ).
2. Логический тип данных является подклассом целого типа. Это можно проверить с помощью вызова функции isinstance(x, int), например:

```
>>> isinstance(True, int)
True
>>> isinstance(False, int)
True
```

3. Любые числовые объекты являются немутируемыми.
4. Python поддерживает смешанную арифметику, которая позволяет выполнять бинарные операции над числовыми объектами, типы данных которых не совпадают. При смешанных операциях выполняется неявное приведение типов объектов путем их расширения или сужения по схеме bool → int → float → complex при расширении и в обратном порядке при сужении.

```
>>> # Расширение объекта типа int до объекта типа float.
>>> x = 10 + 10.5
>>> # Результат операции - объект типа float.
>>> type(x)
<class 'float'>
>>> # Расширение объекта типа int до объекта типа complex.
>>> x = 10 + complex(10.5, 3)
>>> # Результат операции - объект типа complex.
>>> type(x)
<class 'complex'>
>>> # Сужение объекта типа float до объекта типа int.
>>> x = int(10.5)
>>> x, type(x)
(10, <class 'int'>)
```

5. Встроенные числовые типы данных поддерживают стандартный набор операций, который приведен в таблице 6.1.

Таблица 6.1 – Стандартные операции над числами в Python

Операция	Название	Результат	Пример
x + y	Сложение	Сумма x и y	`>>> 8 + 0.33` `8.33`
x – y	Вычитание	Разность x и y	`>>> 8 - 0.33` `7.67`
x * y	Умножение	Произведение x и y	`>>> 2 * 2.5` `5.0`
x / y	Деление	Частное x и y	`>>> 1 / 3` `0.3333333333333333`

x // y
см. прим. 1, 4
	Целочисленное деление	Частное от деления x на y	`>>> 10 // 3`
			`3`
			`>>> -1 // 2`
			`-1`
			`>>> 1 // -2`
			`-1`
			`>>> -1 // -2`
			`0`
			`>>> 8.5 // 3`
			`2.0`

x % y
см. прим. 2, 4
	Деление по модулю	Остаток от деления x на y	`>>> 5 % 3`
			`2`
			`>>> 5 % -3`
			`-1`
			`>>> 6.4 % 3.2`
			`0.0`

x ** y	Возведение в степень	Число x, возведённое в степень y	`>>> 5 ** -1`
			`0.2`
			`>>> 16 ** 0.5`
			`4.0`
			`>>> 0 ** 0`
			`1`

-x
см. прим. 3
	Арифметическое отрицание	Смена знака x	`>>> -5`
			`-5`
			`>>> --5`
			`5`

+x см. прим. 3	Унарный плюс	x без изменений	`>>> +5` `5`
abs(x)	Модуль	Абсолютное значение x	`>>> abs(-5)` `5`
divmod(x, y) см. прим. 4	Определение частного и остатка от деления	Кортеж (x // y, x % y)	`>>> divmod(13, 5)` `(2, 3)`

pow(x, y)	Возведения в степень с опциональным делением по модулю	Степень x и y	`>>> pow(2, 3)`
			`8`
			`>>> pow(2, 3, 3)`
			`2`
			`>>> pow(0, 0)`
			`1`

Примечание:

1. Если x и y – целые числа, то операция возвращает объект целого типа, если x и/или y – вещественные числа, то операция возвращает объект вещественного типа с целым значением; если x или y имеют отрицательное значение, то результат операции округляется в сторону минус бесконечность, то есть при - 3 // 2 результат будет округляться в сторону -2, а не – 1.
2. Знак остатка от деления всегда равен знаку делителя.
3. Оператор «унарный плюс» и «унарный минус» с операндом логического типа выполняет неявное преобразование объекта в целый тип, например

```
>>> +True
1
>>> +False
0
```

4. Оператор не поддерживается комплексными числами.

6.2 Целые числа

В Python для определения целых чисел (англ. integer number) используется встроенный тип данных int. Тип данных int позволяет определять как положительные, так и отрицательные целые числа неограниченной длины, то есть для хранения значения объекта целого типа используется вся доступная память, а размер максимального и минимального значения ограничены только объемом этой памяти. Рассмотрим основные способы создания объекта типа int.

1. Создание объекта типа int с помощью литерала. Правила определения целочисленных литералов описываются с помощью следующей конструкции EBNF

```
integer     ::= decinteger | bininteger | octinteger | hexinteger
decinteger  ::= nonzerodigit (["_"] digit)* | "0"+ (["_"] "0")*
bininteger  ::= "0" ("b" | "B") (["_"] bindigit)+
octinteger  ::= "0" ("o" | "O") (["_"] octdigit)+
hexinteger  ::= "0" ("x" | "X") (["_"] hexdigit)+
nonzerodigit ::= "1"..."9"
digit       ::= "0"..."9"
bindigit    ::= "0" | "1"
octdigit    ::= "0"..."7"
hexdigit    ::= digit | "a"..."f" | "A"..."F"
```

Как видно из EBNF описания, целочисленные литералы могут быть определены в десятичной, двоичной, восьмеричной и шестнадцатеричной системах счисления. Для определения целого числа в десятичной системе счисления используются цифры 0…9, в двоичной системе счисления – префикс «b»(«B») и цифры 0…1, в восьмеричной системе счисления – префикс «o»(«O») и цифры 0…7, в

шестнадцатеричной системе счисления - префикс «x»(«X»), цифры 0…9, и символы «a»(«A»)… «f»(«F»). Примеры определения целочисленных литералов:

```
>>> 255
255
>>> 0b1111_1111
255
>>> 0o03_77
255
>>> 0xFF
255
```

2. Создание объекта типа int с помощью вычисления значения выражения. Результатом выполнения некоторой операции (ий) или функции (ий) может быть объект целого типа, например

```
>>> # Функция ord(x) возвращает код символа x.
>>> x = ord('a')
>>> x
97
>>> type(x)
<class 'int'>
```

3. Создание объекта типа int с помощью конструктора int.

```
int([x])
int(x, base=10)
```

Конструктор int без аргумента x возвращает объект целого типа со значением 0, например:

```
>>> int()
0
```

Конструктор int с одним аргументом x выполняет преобразование x в целое десятичное число. При этом x может быть любым числом (кроме комплексного) или строкой – символьным представлением числового литерала в десятичной системе счисления, например:

```
>>> # Логические значения True и False \
>>> # преобразуются в 1 и 0 соответственно.
>>> int(True)
1
>>> int(False)
0
>>> # Целое число возвращается без преобразований.
>>> int(10)
10
>>> # Вещественные числа округляются в сторону нуля,\
>>> # то есть вещественная часть числа отбрасывается.
>>> int(10.4), int(10.5), int(10.6)
(10, 10, 10)
>>> # Строка преобразуется в целое число.
>>> int('10')
10
```

Конструктор int с двумя аргументами выполняет преобразование строки x, которая представляет символьный числовой литерал в заданной системе счисления, в целое десятичное число. При этом значение аргумента base должно находиться в диапазоне от 2 до 36; если base установлено в 0, то x воспринимается как числовой литерал, например:

```
>>> int('FF', base=16)
255
>>> int('377', base=8)
255
>>> int('1111_1111', base=2)
255
>>> int('0b1111_1111', base=0)
255
```

Если конструктор int не может выполнить преобразование x в целое число, то генерируется исключительная ситуация, например:

```
>>> int('AAA')
ValueError: invalid literal for int() with base 10: 'AAA'
>>> int(3j)
TypeError: can't convert complex to int
```

Тип данных int кроме стандартных операций, которые приведены в таблице 6.1, поддерживает также набор побитовых операций, которые соответствуют таким булевым функциям, как: дизъюнкция (|), исключающее «или» (^), конъюнкция (&) и отрицание (~), а также левый и правый сдвиги. Для вычисления значения булевых функций используются таблицы истинности:

x	y	|
0	0	0
0	1	1
1	0	1
1	1	1

x	y	^
0	0	0
0	1	1
1	0	1
1	1	0

x	y	&
0	0	0
0	1	0
1	0	0
1	1	1

x	~
0	1
1	0

Описание побитовых операций приведено в таблице 6.2.

Таблица 6.2 – Побитовые операции над типом данных int

Операция	Описание	Пример

x | y	Побитовое ИЛИ (OR).
	Выполняет операцию OR над каждой парой бит из x и y
		`>>> x = 0b0110_0111`
		`>>> y = 0b1010_1111`
		`>>> bin(x | y)`
		`'0b11101111'`

x ^ y	Побитовое исключающее ИЛИ (XOR).
	Выполняет операцию XOR над каждой парой бит из x и y
		`>>> x = 0b0110_0111`
		`>>> y = 0b1010_1111`
		`>>> bin(x ^ y)`
		`'0b11001000'`

x & y	Побитовое И (AND).
	Выполняет операцию AND над каждой парой бит из x и y
		`>>> x = 0b0110_0111`
		`>>> y = 0b1010_1111`
		`>>> bin(x & y)`
		`'0b100111'`

Операция	Описание	Пример

x << y	Левый сдвиг.
	Сдвигает двоичное представление x на y битов влево, добавляя справа нули
		`>>> x = 0b0110_0111`
		`>>> y = 4`
		`>>> bin(x << y)`
		`'0b11001110000'`

x >> y	Правый сдвиг. Сдвигает двоичное представление x на y битов вправо, отбрасывая сдвигаемые биты	`>>> x = 0b0110_0111`
		`>>> y = 4`
		`>>> bin(x >> y)`
		`'0b110'`

~x	Побитовое отрицание (NOT).
	Выполняет побитовую инверсию x.
		`>>> x = 0b0110_0111`
		`>>> bin(~x)`
		`'-0b1101000'`

6.3 Логические значения

В Python для определения логических значений (англ. boolean) используется встроенный тип данных bool. Рассмотрим основные способы создания объекта типа bool.

1. Создание объекта типа bool с помощью констант True и False. Объект логического типа может принимать только одно из двух значений – True или False. Значение True соответствует «правда», «истина», «да», а False - «не правда», «ложь», «нет». Пример создание объекта типа bool:

```
>>> x, y = True, False
>>> type(x), type(y)
(<class 'bool'>, <class 'bool'>)
```

2. Создание объекта типа bool с помощью вычисления значения выражения. Результатом выполнения некоторой операции (ий) или функции (ий) может быть объект логического типа, например

```
>>> x = 2 ** 10
>>> y = 10 ** 2
>>> z = x > y
>>> z, type(z)
(True, <class 'bool'>)
```

   В результате сравнения двух объектов был создан объект логического типа со значением True.

3. Создание объекта типа bool с помощью конструктора bool.

```
bool([x])
```

   Конструктор bool без аргумента x создает объект логического типа со значением False, например:

```
>>> bool()
False
```

   Конструктор bool с аргументом x выполняет проверку истинности значения x и создает объект логического типа со значением True или False. Правила проверки истинности значения x:

   - объект любого типа может быть проверен на истинность значения;
   - x – ложь, если None;
   - x – ложь, если False;

- x – ложь, если является нулем любого числового типа данных;
- x – ложь, если является пустой последовательностью;
- x – истина, во всех остальных случаях.

Принято со значением False ассоциировать целочисленное значение 0, а со значением True – 1. Пример создания объекта логического типа:

```
>>> bool(None), bool(False)
(False, False)
>>> bool(0), bool(0.0), bool(0j)
(False, False, False)
>>> bool([]), bool(()), bool({})
(False, False, False)

>>> bool(True), bool(100), bool([1, 2, 3]), bool('Hello')
(True, True, True, True)
```

Тип данных bool кроме стандартных операций, которые приведены в таблице 6.1, поддерживает также набор логических операций, описание которых приведено в таблице 6.3. Значения логических операций, аналогично побитовым операциям, определяются с помощью таблиц истинности:

x	y	OR
False	False	False
False	True	True
True	False	True
True	True	True

x	y	AND
False	False	False
False	True	False
True	False	False
True	True	True

x	NOT
False	True
True	False

Таблица 6.3 – Логические операции над типом данных bool

Операция	Описание	Пример

x or y	Логическое ИЛИ (OR).
	Возвращает y, если x «ложь», иначе – x
		`>>> x, y = False, True`
		`>>> x or y`
		`True`
		`>>> x, y = True, False`
		`>>> x or y`
		`True`

x and y	Логическое И (AND).
	Возвращает x, если x «ложь», иначе – y
		`>>> x, y = False, True`
		`>>> x and y`
		`False`
		`>>> x, y = True, False`
		`>>> x and y`
		`False`

not x	Логическое отрицание.
	Возвращает True, если x «ложь», иначе False.
		`>>> x = True`
		`>>> not x`
		`False`
		`>>> x = False`
		`>>> not x`
		`True`

6.4 Вещественные числа

В Python для определения вещественных чисел (англ. real number) используется встроенный тип данных float. Тип данных float позволяет определять вещественные числа в форме с плавающей точкой двойной точности. Согласно международному стандарту IEEE 754, вещественные числа с плавающей точкой в нормализованном виде представляются следующим образом:

$$N = (-1)^s M B^p$$

где s – знак числа N (0 – положительное число, 1 - отрицательное);

M – мантисса числа N, $M \in [0, B)$ (для двоичной записи числа - 1 ≤ M < 2, для десятичной - 1 ≤ M < 10);

B – основание системы счисления;

p – порядок числа N.

Рассмотрим пример представления вещественного числа в нормализованном виде:

$$123.456 = (-1)^0 \cdot 123.456 \cdot 10^0 = (-1)^0 \cdot 12.3456 \cdot 10^1 = (-1)^0 \cdot 1.23456 \cdot 10^2$$

Полученное нормализованное число $(-1)^0 \cdot 1.23456 \cdot 10^2$ состоит из следующих частей: знак числа - 0, мантисса числа - 1.23456, основание системы счисления – 10 и порядок числа - 2.

Двоичное представление вещественного числа с плавающей точкой представляет собой набор битов, который характеризуется тремя составляющими: знаком числа (s), порядком числа (p) и мантиссой (M). Так как данные в памяти хранятся в двоичном коде, то основание системы счисления не указывается. Стандарт IEEE–754 определяет четыре формата представления вещественных чисел с плавающей точкой:

- – с одинарной точностью (англ. single-precision);
- – с двойной точностью (англ. double-precision);
- – с одинарной расширенной точностью (англ. single-extended precision);
- – с двойной расширенной точностью (англ. double-extended precision).

В формате двойной точности для хранения значения числа отводится 64 бита (8 байт), из которых один бит отводится под знак числа, 11 бит – под порядок числа и 52 бита – под мантиссу (рис. 6.1).

Рис. 6.1. Представление числа с плавающей точкой двойной точности в памяти компьютера

Основная информация «низкого» уровня о типе данных float в Python представлена в структуре float_info, которая находится в модуле sys. Подробное описание атрибутов структуры float_info приведено в таблице 6.4.

Таблица 6.4 – Характеристика типа данных float

Атрибут	Значение	Описание
epsilon	$2.220446049250313 \cdot 10^{-16}$	Разность между 1.0 и ближайшим числом с плавающей точкой больше 1.0
dig	15	Количество десятичных знаков, которые могут быть представлены без изменений после точки
mant_dig	53 (52 + 1 бит по умолчанию, так как в нормализованном виде старший бит мантиссы всегда равен 1)	Количество цифр мантиссы в системе счисления по основанию, указанному в атрибуте float_info.radix
max	$1.7976931348623157 \cdot 10^{+308}$	Максимально возможное число с плавающей точкой
max_10_exp	308	Максимальная величина порядка в 10-ой системе счисления
max_exp	1024	Максимальная величина порядка в системе счисления по основанию, указанному в атрибуте float_info.radix.
min	$2.2250738585072014 \cdot 10^{-308}$	Минимально возможное положительное число с плавающей точкой
min_10_exp	-307	Минимальная величина порядка в 10-ой системе счисления
min_exp	-1021	Минимальная величина порядка в системе счисления по основанию, указанному в атрибуте float_info.radix.
radix	2	Основание системы счисления
rounds	1	Алгоритм округления (-1 – не определено, 0 – в сторону нуля, 1 – до ближайшего значения, 2 – в сторону положительной бесконечности, 3 – в сторону отрицательной бесконечности).

Рассмотрим основные способы создания объекта типа float.

1. Создание объекта типа float с помощью литерала. Правила определения вещественных литералов описываются с помощью следующей конструкции EBNF

```
floatnumber     ::= pointfloat | exponentfloat
pointfloat      ::= [intpart] fraction | intpart "."
exponentfloat   ::= (intpart | pointfloat) exponent
```

```
intpart      ::= digit+
fraction     ::= "." digit+
exponent     ::= ("e" | "E") ["+" | "-"] digit+
```

Как видно из EBNF описания, вещественные числа могут быть представлены в привычном виде, то есть содержать символ десятичной точки и в экспоненциальной форме - содержать символ экспоненты («e» или «E») со знаком. Символ «e» или «E» в записи числа означает «…умножить на десять в степени…», например, 123e2 означает $123 \cdot 10^2$, а 123e2 - $123 \cdot 10^{-2}$. Примеры определения вещественных литералов:

```
>>> 3.14
3.14
>>> 0.314e1
3.14
>>> 31.4e-1
3.14
```

2. Создание объекта типа float с помощью вычисления значения выражения. Результатом выполнения некоторой операции (ий) или функции (ий) может быть объект типа float, например

```
>>> # Оператор '/' с возвращает объект вещественного типа.
>>> a = 1/3 + 1/3 + 1/3
>>> a
1.0
>>> type(a)
<class 'float'>
```

3. Создание объекта типа float с помощью конструктора float.

```
float(x=0)
```

Конструктор float выполняет преобразование x в число с плавающей точкой. При этом x может быть любым числом (кроме комплексного) или строкой – символьным представлением числового литерала в десятичной системе счисления, например:

```
>>> # Конструктор float без аргумента x создает объект \
>>> # вещественного типа со значением 0.0.
>>> float()
0.0
>>> # К целому числу добавляется десятичная точка.
>>> float(10)
10.0
>>> # Вещественное число остается без изменений.
>>> float(10.0)
10.0
>>> # Строка преобразуется в вещественное число.
>>> float('10')
10.0
>>> float('1e1')
10.0
```

Аргумент x может также быть строкой, которая представляет NaN (not-a-number), положительную или отрицательную неопределенность, например:

```
>>> float('nan')
nan
>>> float('+inf')
inf
>>> float('-inf')
-inf
```

Если конструктор float не может выполнить преобразование x в число с плавающей точкой, то генерируется исключительная ситуация, например:

```
>>> float('0xFF')
ValueError: could not convert string to float: '0xFF'
>>> float(3j)
TypeError: can't convert complex to float
```

Тип данных float поддерживает весь набор стандартных операций, которые приведены в таблице 6.1. Однако при работе с числами с плавающей точкой необходимо помнить о ошибке представления (англ. representation error), которая возникает из-за того, что размер мантиссы числа в двоичном представлении, как правило, превышает допустимые 53 бита. Поэтому при явном сравнении двух чисел с плавающей точкой может получиться совсем не ожидаемый результат, например:

```
>>> 0.1 + 0.1 + 0.1 == 0.3
False
>>> 0.2 + 0.2 + 0.2 == 0.6
False
>>> 0.3 + 0.3 + 0.3 == 0.9
False
```

Согласно данным из таблицы 6.4, при представлении числа с плавающей точкой двойной точности, гарантируется точное представление только 15 десятичных цифр числа после точки, например:

```
>>> format(0.3, '.15')
'0.3'
>>> format(0.3, '.17')
'0.29999999999999999'
```

С целью уменьшения накопления погрешности при работе с числами с плавающей точкой необходимо применять специальные алгоритмы или использовать дополнительные типы Decimal и Fraction, которые позволяют работать с привычными «точными» действительными и рациональными числами.

6.5 Комплексные числа

В Python для определения комплексных чисел используется встроенный тип данных complex. Рассмотрим основные способы создания объекта типа данных complex.

1. Создание объекта типа complex с помощью литерала мнимой части комплексного числа. Правила определения литералов мнимой части комплексных чисел описываются с помощью следующей конструкции EBNF

```
imagnumber ::= (floatnumber | digitpart) ("j" | "J")
```

Как видно из EBNF описания, для формирования мнимой части комплексного числа необходимо добавить символ j или J к одному из двух числовых литералов – целочисленному или с плавающей точкой, например:

```
>>> x = 3j
>>> x, type(x)
(3j, <class 'complex'>)
>>> x = 3.14j
>>> x, type(x)
(3.14j, <class 'complex'>)
```

Для получения комплексного числа с мнимой и вещественной частью, необходимо литерал с мнимой частью сложить с числовым литералом, например:

```
>>> 2 + 3j
(2+3j)
```

2. Создание объекта типа complex с помощью вычисления значения выражения. Результатом выполнения некоторой операции (ий) или функции (ий) может быть объект типа complex, например

```
>>> (-1) ** 0.5
(6.123233995736766e-17+1j)
```

В результате вычисления квадратного корня из отрицательного числа был создан объект комплексного типа со значением (6.123233995736766e-17+1j).

3. Создание объекта типа complex с помощью конструктора complex.

```
complex([real[, imag]])
```

Конструктор complex создает комплексное число со значением real + imag * 1j, например:

```
>>> complex(real=2, imag=3)
(2+3j)
```

По умолчанию аргументы real и imag равны 0, например:

```
>>> complex()
0j
```

Аргумент real может быть любым числом (кроме комплексного) или строкой, а imag – только числом. Если real является строковым представлением комплексного числа, то второй аргумент imag не используется. Примеры:

```
>>> complex(2, 3.)
(2+3j)
>>> complex(2.5, 3.)
(2.5+3j)
>>> complex('2.5+3j')
(2.5+3j)
```

Отдельные части комплексного числа доступны в виде атрибутов real и imag, например:

```
>>> x.real, x.imag
(2.0, 3.0)
```

Если конструктор complex не может создать объект комплексного типа, то генерируется исключительная ситуация, например:

```
>>> # В строковом представлении между операцией и \
>>> # операндами не может быть пробелов.
>>> complex('2.5 + 3j')
ValueError: complex() arg is a malformed string
```

Тип данных complex частично поддерживает стандартные операции, которые приведены в таблице 6.1 (не поддерживаемые операции указаны в примечании к таблице).

6.6 Модули math и random

Кроме встроенных функций и операций, которые поддерживают числовые типы данных, Python также предоставляет большой набор стандартных математических функций и констант, которые находятся в модулях math и cmath. Функции модуля math оперируют целыми числами и числами с плавающей точкой, а функции модуля cmath – комплексными числами. Описание основных функций и констант модуля math представлены в таблице 6.5. Подключение модулей в Python осуществляется с помощью оператора import, например:

```
>>> import math
```

Таблица 6.5 – Основные константы и функции модуля math

Функция	Описание	Пример
Константы		
math.pi	Возвращает число *π = 3.141592…*	`>>> math.pi` `3.141592653589793`
math.e	Возвращает число e = 2.718281…	`>>> math.e` `2.718281828459045`
math.tau	Возвращает число *τ = 6.283185…*	`>>> math.tau` `6.283185307179586`
Функции		
math.ceil(x)	Возвращает округленное число x до ближайшего большего целого числа	`>>> math.ceil(3.2)` `4`
math.fabs(x)¶	Возвращает модуль числа x	`>>> math.fabs(-3.2)` `3.2`
math.factorial(x)¶	Возвращает факториал числа x	`>>> math.factorial(5)` `120`
math.floor(x)¶	Возвращает округленное число x до ближайшего меньшего целого числа	`>>> math.floor(3.2)` `3`

Функция	Описание	Пример
math.gcd(a, b)	Возвращает наибольший общий делитель целых чисел a и b.	`>>> math.gcd(20, 225)` `5`

math.isclose(a, b, *, rel_tol=1e-09, abs_tol=0.0)¶	Возвращает True, если значения a и b близки друг к другу, и False - в противном случае	`>>> math.isclose(0.3, 0.31)`
		`False`
		
		`>>> math.isclose(0.3, 0.3000000001)`
		`True`

math.modf(x)	Возвращает дробную и целую часть числа	`>>> math.modf(3.2)` `(0.20000000000000018, 3.0)`
math.trunc(x)	Возвращает x с усеченной дробной частью	`>>> math.trunc(3.2)` `3`
math.exp(x)	Возвращает e^x	`>>> math.exp(3)` `20.085536923187668`
math.log(x[, base])	Возвращает логарифм числа x по основанию base	`>>> math.log(10, 2)` `3.3219280948873626`
math.pow(x, y)	Возвращает число x в степени y	`>>> math.pow(3, 2)` `9.0`
math.sqrt(x)¶	Возвращает квадратный корень числа x	`>>> math.sqrt(16)` `4.0`
math.acos(x)	Возвращает в радианах арккосинус угла x	`>>> math.acos(0.5)` `1.0471975511965979`
math.asin(x)	Возвращает в радианах арксинус угла x	`>>> math.asin(0.5)` `0.5235987755982989`
math.atan(x)	Возвращает в радианах арктангенс угла x	`>>> math.atan(1)` `0.7853981633974483`
math.cos(x)	Возвращает косинус угла x, заданного в радианах	`>>> math.cos(math.pi/3)` `0.5000000000000001`
math.sin(x)	Возвращает синус угла x, заданного в радианах	`>>> math.sin(math.pi/6)` `0.49999999999999994`
math.tan(x)	Возвращает тангенс угла x, заданного в радианах	`>>> math.tan(math.pi/4)` `0.9999999999999999`
math.degrees(x)	Конвертирует радианную меру угла x в градусную	`>>> math.degrees(math.pi / 6)` `29.999999999999996`
math.radians(x)	Конвертирует градусную меру угла x в радианную	`>>> math.radians(30)` `0.5235987755982988`

Примечание: все функции модуля math возвращают объект вещественного числа с плавающей точкой.

Для эмуляции случайности в Python используется модуль random, который предоставляет различные генераторы псевдослучайных чисел (англ. pseudo-random number generator, PRNG) — алгоритмы, порождающие последовательности чисел, элементы которых почти независимы друг от друга и подчиняются заданному распределению. Почти все функции модуля random опираются на базовую функцию

random( ), которая с помощью алгоритма Mersenne Twister генерирует псевдослучайные числа с плавающей точкой, равномерно распределенные на интервале [0.0, 1.0). Описание основных функций модуля random представлены в таблице 6.6.

Таблица 6.6 – Основные функции модуля random

Функция	Описание	Пример
random.randint(a, b)	Возвращает псевдослучайное целое число в диапазоне [a, b]	`>>> random.randint(10, 20)` `16`
random.random()	Возвращает псевдослучайное число с плавающей точкой в диапазоне [0.0, 1.0)	`>>> random.random()` `0.20452261760976953`
random.uniform(a, b)	Возвращает псевдослучайное число с плавающей точкой в диапазоне [a, b]	`>>> random.uniform(10, 20)` `11.221968542787712`

6.7 Терминология

приведение типа

смешанная арифметика

целое число

логическое значение

вещественное число

вещественное число с плавающей точкой

нормализованное вещественное число с плавающей точкой

мантисса числа

порядок числа

ошибка представления

комплексное число

генераторы псевдослучайных чисел

6.8 Контрольные вопросы и упражнения

Контрольные вопросы:

1. Опишите особенности типа данных int.
2. Опишите особенности типа данных bool.
3. Опишите особенности типа данных float.
4. Опишите особенности типа данных complex.
5. В программе определены две переменные a = True и b = False. Объясните, почему первая строчка кода является синтаксически корректной конструкцией, а вторая - нет?

```
>>> not a == b
True
>>> a == not b
SyntaxError: invalid syntax
```

6. Какие выделяют форматы представления вещественных чисел с плавающей точкой?

7. Опишите особенности нормализованного представления вещественного числа с плавающей точкой.
8. Принято считать, что под мантиссу числа с плавающей точкой двойной точности отводится 52 бита. Почему значение атрибута float_info.mant_dig равно 53?
9. Приведите особенности операторов /, // и % при работе с целыми и вещественными числами.
10. Приведите правила проверки истинности значения объекта.

Вопросы для самостоятельной работы:

1. Стандарт IEEE 754-2008.
2. Числа с фиксированной точностью (модуль decimal).
3. Рациональные числа (модуль fraction).
4. Управление состоянием генератора псевдослучайных чисел.
5. Функции модуля statistics.