Лекція 6

Представлення чисел у мові Python

Числові типи в мові Python 3

`int` – цілі числа. Розмір числа обмежений лише обсягом оперативної пам'яті:

Приклад: `1, 2 10, 34563`

`float` – дійсні числа;

Формат десяткового дробу

`1.1, 2.234 10.12, 34563.1`

Математичний формат: `mep-m-мантиса, p-порядок`

Цілочисельна мантиса: `12e2=1200, 1e-1=0.1`

Дробова мантиса: `1.2e3=1200, 0.01e1=0.1`

Нормалізована мантиса: `0.12e4=1200, 0.1e2=10`

`complex` – комплексні числа

```
>>> 1+2j+2+3j
(3+5j)
```

Операції над числами різних типів повертають число, що має більш складний тип з типів, що брав участь в операції.

Співвідношення типів чисел при арифметичних операціях

1. Цілі числа мають найпростіший тип,
2. Далі йдуть дійсні числа.
3. Найскладніший тип – комплексні числа.

Правило перетворення

Якщо в операції беруть участь ціле число й дійсне, то ціле число буде автоматично перетворене в дійсне число, а потім виконана операція над дійсними числами.

Результатом цієї операції буде дійсне число.

`>>> 1+2.0` `3.0`	`>>> 2+0.2e1` `4.0`	`>>> 3.1+3+4j` `(6.1+4j)`

Створення об'єктів цілочисельного типу в десятковій системі числення

Створення об’єктів перерахуванням елементів

Приклад 1.

```
>>> x, y, z = 1,10,-80
>>> x,y,z
(1, 10, -80)

>>> p = 1,10,-80
>>> list(p)
[1, 10, -80]

>>> a="12345678"
>>> list(a)
['1', '2', '3', '4', '5', '6', '7','8']
```

Створення об’єктів за допомогою функції range()

Приклад 2.

```
>>> list(range(9))
[0, 1, 2, 3, 4, 5, 6, 7, 8]

>>> tuple(range(9))
(0, 1, 2, 3, 4, 5, 6, 7, 8)

>>> tuple(range(1,20,2))
(1, 3, 5, 7, 9, 11, 13, 15, 17, 19)

>>> tuple(range(50,1,-4))
(50,46,42,38,34,30,26,22,18,14,10,6,2)
```

Створення об'єкта цілочисельного типу у двійковій системі числення

Двійкові числа починаються з комбінації символів `0b` (або `0B`) і містять цифри 0 або 1.

Приклад 3.

```
>>> 0b111, 0b101
(7, 5)

>>> range(0b1010)
range(0, 10)
>>> list(range(0b1010))
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
```

0000 – 0
0001 – 1
0010 – 2
0011 – 3
0100 – 4
0101 – 5
0110 – 6
0111 – 7
1000 – 8
1001 – 9
1010 – 10

Створення об'єкта цілочисельного типу у вісімковій системі числення

Вісімкове число починається з нуля й наступної за ним латинської букви o (регістр не має значення) і містять цифри від 0 до 7:

Приклад 4.

```
>>> >>> 0o0, 0o1, 0o7, 0o10, 0o11
(0, 1, 7, 8, 9)

>>> range(0o12)
range(0, 10)
>>> tuple(range(0o12))
(0, 1, 2, 3, 4, 5, 6, 7, 8, 9)
```

00 – 0
01 – 1
02 – 2
03 – 3
04 – 4
05 – 5
06 – 6
07 – 7
10 – 8
11 – 9
12 – 10

Створення об'єкта цілочисельного типу в шістнадцятковій системі числення

Шістнадцяткові числа починаються з комбінації символів 0x (або 0X) і можуть містити цифри від 0 до 9 і букви від A до F (регістр букв не має значення):

Приклад 5.

```
>>> 0x1,0x2, 0X9, 0xa, 0xf
(1, 2, 9, 10, 15)

>>> range(0xb)
range(0, 11)
>>> list(range(0xb))
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
```

0 – 0
1 – 1
2 – 2
3 – 3
4 – 4
5 – 5
6 – 6
7 – 7
8 – 8
9 – 9
a – 10
b – 11
c – 12
d – 13

Створення об'єкта дійсного типу

Дійсне число може містити крапку й (або) бути задане в науковому форматі (експонентній формі) з буквою E (регістр не має значення):

Приклад 6.

```
>>> x, y, z = 1.1,10.12,-80.455
>>> x,y,z
(1.1, 10.12, -80.455)
```

```
>>> p = [1.1, 2,345, 0.23, 45.67, 11.678, 4.67, 11.56]
>>> p[1:3]
[2, 345]
>>> p[2:7]
[345, 0.23, 45.67, 11.678, 4.67]
```

Проблема точного представлення числа для чисел з плаваючою крапкою

При виконанні операцій над дійсними числами слід враховувати обмеження точності обчислень.

Результат наступної операції може здатися дивним.

Приклад 7.

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

>>> 0.6-0.2-0.4
-5.551115123125783e-17
```

Очікуваним був би результат `0.0`, але, як видно з прикладу, ми одержали зовсім інше значення.

Використання функції `Decimal`

Якщо необхідно виконувати операції з фіксованою точністю, то слід використовувати модуль `decimal`.

Перед виконання таких операцій викликаємо:

```
from decimal import Decimal
```

Приклад 8.

```
>>> from decimal import Decimal

>>>Decimal("0.2")-Decimal("0.1")-Decimal("0.1")
Decimal('0.0')

>>>p=float(Decimal("0.2")-Decimal("0.1")-Decimal("0.1"))
>>> p
0.0
```

Використання функції `Fraction`

Підтримка дробів виконується за допомогою модуля `fractions`.

При створенні дробу можна:

1. Указати два числа: чисельник і знаменник.
2. Указати одне число.
3. Указати рядок, що містить число, яке буде перетворено в дріб.

Приклад 9. Формування дробу 4/5:

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

Приклад 10. Задати дріб ½ трьома способами

```
Задавання за допомогою дробу
>>> Fraction(1, 2)
Fraction(1, 2)

Задавання за допомогою рядка
>>> Fraction("0.5")
Fraction(1, 2)

Задавання за допомогою числа
>>> Fraction(0.5)
Fraction(1, 2)
```

Арифметичні операції над дробами

Над дробами можна виконувати арифметичні операції, як і над звичайними числами.

Приклад 11. Виконати операцію віднімання:

```
from fractions import Fraction
from math import *
p=Fraction(1, 2)+Fraction(1, 4)
c=float(p)
l=trunc(p)
m=modf(p)
print("c = ",c)
print("l = ",l)
print("m = ",m)
print(p)
```

```
c =  0.75
l =  0
m =  (0.75, 0.0)
3/4
```

Представлення комплексних чисел

Комплексні числа записуються у форматі:

<Дійсна частина>+<Уявна частина>J

Для задавання можна використовувати функцію `complex`.

Приклад 12.

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

Клас комплексних чисел має кілька вбудованих методів.

Приклад 13.

```
>>> z = 2+3j
>>> z.real
2.0             # ціла частина
>>> z.imag
3.0             # уявна частина
>>> z.conjugate()
(2-3j)          # спряжене (поєднане?) число
```

Операції з комплексними числами

Приклад 14.

```
>>> complex(2,3)==2+3J
True
>>> 2+3J+4+7J
(6+10j)
>>> complex(2,3)+complex(4,7)
(6+10j)

Правило множення: (a+bJ)*(c+dJ)
                  M=(ac-bd)+(ad+bc)J
                M=(2*4-3*7)+(2*7+3*4)J

>>> complex(2,3)*complex(4,7)
(-13+26j)
```

Вбудовані функції для роботи із числами

Для роботи із числами будемо використовувати наступні вбудовані функції:

1. `Int([<Об'єкт>[,<Система числення>]])`

Функція перетворює об'єкт у ціле число. У другому параметрі можна вказати систему числення перетвореного числа (значення за замовчуванням 10).

Приклад 15.

```
>>> int(7.5)
7
>>> int("71",10)
71
>>> int("0B1010",2)
10
>>> int("0o71",8)
57
>>> int("0xa",16)
10
```

2. bin(<Число>)

Перетворює десяткове число у двійкове. Повертає строкове представлення числа.

Приклад 16.

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

3. oct(<Число>)

Перетворює десяткове число у вісімкове. Повертає строкове представлення числа.

Приклад 17.

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

4. hex(<Число>)

Перетворює десяткове число в шістнадцяткове. Повертає строкове представлення числа.

Приклад 18.

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

5. float([<число або рядок>])

Перетворює ціле число або рядок у дійсне число

Приклад19.

```
>>> float(7)
7.0
>>> float("7.1")
7.1
>>> float("inf")
inf
```

```
>>> float("-Infinity")
-inf
>>> float("nan")
nan
>>> float()
0.0
```

6.round(<Число>[,<Кількість знаків після точки>])

Для чисел із дробовою частиною, меншою за 0.5, повертає число, округлене до найближчого меншого цілого, а для чисел із дробовою частиною, більшою за 0.5, повертає число, округлене до найближчого більшого цілого.

Якщо дробова частина дорівнює 0.5, то округлення проводиться до найближчого парного числа.

Приклад 20.

```
>>> 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)
```

У другому параметрі можна вказати бажану кількість знаків після коми. Якщо вона не зазначена, то використовується значення 0 (тобто число буде округлено до цілого):

Приклад 21.

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

7. abs(<Число>) Повертає абсолютне значення:

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

8. pow(<Число>,<Степінь>[,<Дільник>])

Підносить <Число> до <Степеня>

Приклад 21.

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

Якщо зазначений третій параметр, то повертається залишок від ділення отриманого результату на значення цього параметра.

Приклад 22.

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

9. max(<Список чисел через кому>)
   max(<Послідовність>)

Повертає максимальне значення зі списку або послідовності.

Приклад 23.

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

`>>> p="string"`
`>>> max(p)`
`'t'`
	`>>> p=(1,0.5,0.8,True)`
	`>>> max(p)`
	`1`

`>>>p=[34,11.34,124]`
`>>> max(p)`
`124`
	`>>> p={1, 3, 1.4, True}`
	`>>> max(p)`
	`3`

10. min (<список чисел через кому>)
    min(<Послідовність>)

Мінімальне значення зі списку:

Приклад 24.

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

`>>> p="string"`
`>>> min(p)`
`'g'`
	`>>> p=(4,5,6,7,8,3)`
	`>>> min(p)`
	`3`

`>>>p=[34,11.34,124]`
`>>> min(p)`
`11.34`
	`>>> p={1, 3, 1.4, True}`
	`>>> min(1)`
	`1`

Застосування функцій max і min в словниках

Приклад 25.

```
>>>max({"a":2016, "b":"London", "c": "West"})
'c'
>>>  p = {"a":2016, "b":"London", "c": "West"}
>>> l =max(p)
>>> print(p[l])
West

>>> l = min(p)
>>> print(p[l])
2016
```

11. sum (<Послідовність>[, <Початкове значення>])
повертає суму значень елементів послідовності (наприклад: списку, кортежу) плюс <Початкове значення>. Якщо другий параметр не зазначений, початкове значення вважають рівним 0. Якщо послідовність порожня, то повертається значення другого параметра.

Приклад 26.

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

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

Кортеж:

```
>>> p=(1,2,3,4,5,6,7,8,9)
>>> sum(p)
45
>>> sum(p,3)
48
```

Список:	Множина:

>>> p = [11,12,15,17]
>>> sum(p)
55
>>> sum(p,5)
60
	>>> p = {67,11,78,12}
	>>> sum(p)
	168
	>>> sum(p,2)
	170

12. `divmod(x, y)` **Приклад 27.**

повертає кортеж з двох значень `(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)
```

Вбудовані функції для комплексних чисел:

$\sqrt{3^2+4^2}=5$

```
>>> abs(3 + 4j)
    5.0
```

(3+4J)*(3+4J)

```
>>> pow(3 + 4j, 2)
(-7+24j)
```

Модуль `math`. Математичні функції

Модуль **`math`** надає додаткові функції для роботи із числами, а також стандартні константи. Перш ніж використовувати модуль, необхідно підключити його за допомогою інструкції:

```
import math
from math import *
```

ПРИМІТКА

Для роботи з комплексними числами необхідно використовувати модуль для комплексних чисел

```
import cmath
```

Стандартні константи модуля **math**

- `pi` — повертає число π :

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

- `e` — повертає значення константи e:

```
>>> import math
>>> math.e
2.7182818284590
```

Основні функції для роботи з числами з модуля **math**

- `sin(), cos(), tan ( )` — стандартні тригонометричні функції (синус, косинус, тангенс). Значення вказується в радіанах;

- `asin (), acos (), atan()` — обернені тригонометричні функції (арксинус, арккосинус, арктангенс). Значення повертається в радіанах;

- `degrees ()` — перетворює радіани в градуси:

Приклад 28.

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

- `radians ()` — перетворює градуси в радіани:

Приклад 29.

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

`exp()` — експонента; Приклад 30.

```
>>> import math
>>> math.exp(1)
2.718281828459045
```

- `log(<число>[,<База>])` — логарифм по заданій базі. Якщо база не зазначена, обчислюється натуральний логарифм (по базі *e)*
- `log10()` – десятковий логарифм;
- `log2()` – логарифм по базі 2;
- `sqrt()` – квадратний корінь:

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

- `ceil()` — значення, округлене до найближчого більшого цілого:

```
>>> math.ceil(6.49), math.ceil(6.50),
math.ceil(6.51)
(7,7,7)
```

- `floor()` — значення, округлене до найближчого меншого цілого:

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

- pow(<Число>,<Степінь>) — підносить <Число> до <Степеня>:

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

- fabs ( ) — абсолютне значення:

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

- fmod() — залишок від ділення:

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

- `factorial ()` — факторіал числа:

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

- `fsum(<Список чисел>)` — повертає точну суму чисел із заданого списку:

```
>>> 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
```

Примітка

У цьому розділі ми розглянули тільки основні функції. Щоб одержати повний список функцій, звертайтеся до документації по модулю `math`.

Модуль `random`. Генерація випадкових чисел

Модуль `random` дозволяє генерувати випадкові числа. Перш ніж використовувати модуль, необхідно підключити його за допомогою інструкції:

```
import random
from random import *
```

Основні функції модуля **`random`**:

- `random()` — повертає псевдовипадкове число від 0.0 до 1.0:

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

- `seed([<параметр>][,version=2])`

налаштовує генератор випадкових чисел на нову послідовність. Якщо перший параметр не зазначений, за базу для випадкових чисел буде використаний системний час. Якщо значення першого параметра буде однаковим, то генерується однакове число:

```
>>> randorn.seed(10)

>>> random.random()
0.5714025946899135

>>> randorn.seed(10)

>>> random.random()
0.5714025946899135
```

- uniform(<початок>,<кінець>) — повертає псевдовипадкове дійсне число в діапазоні від <Початок> до <Кінець>:

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

  - randint (<початок>, <кінець>) — повертає псевдовипадкове ціле число в діапазоні від <Початок> до <Кінець>:

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

- `randrange([<початок>,]<кінець>[,<Крок>])` повертає випадковий елемент із числової послідовності. Параметри аналогічні параметрам функції `range ()`. Можна сказати, що функція `randrange` «за кадром» створює діапазон, з якого й будуть вибиратися випадкові числа, що повертаються:

```
>>> random.randrange(10)
5

>>> random.randrange(0,10)
2

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

- `choice (<послідовність>)` — повертає випадковий елемент із заданої послідовності (рядка, списку, кортежу):

```
>>> random.choice("string")
't'

>>>random.choice(["s","t","r"])#список
'r'

>>>random.choice(("s","t","r"))#кортеж
't'
```

- `shuffle(<список>[,<Число від 0.0 до 1.0>])` — перемішує елементи списку випадковим чином. Якщо другий параметр не зазначений, то використовується значення, що повертається функцією `random()`. Жодного результату при цьому не повертається.

Приклад 13.

```
>>> 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(<Послідовність>,<Кількість елементів>)` — повертає список із зазначеної кількості елементів, які будуть обрані випадковим чином із заданої послідовності.

За послідовність можна взяти будь-які об'єкти, що підтримують ітерації.

Приклад 14

```
>>> 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 змінюється
[1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
>>> random.sample((l, 2, 3, 4, 5, 6, 7), 3)
[6, 3, 5]
>>> random.sample(range(300), 5)
[126, 194, 272, 46, 71]
```

Приклад створення генератора паролів довільної довжини

Для цього додаємо в список `arr` усі дозволені символи, а далі в циклі одержуємо випадковий елемент за допомогою функції `choice ()`. За замовчуванням буде видаватися пароль з 8 символів.

```
import random # Підключаємо модуль randorn
def passw_generator(count_char=8):
arr = ['a', 'b', 'c', 'd', 'e', 'f', 'g',
'h', 'i', 'j ', 'k', '1', '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)

# Викликаємо функцію
print(passw_generator(10)
# Виведе щось на кшталт ngodhebjbx
print(passw_generator())
# Виведе щось на кшталт Zxcpkfso
```

Встановлення пакетів для Python

Для встановлення пакетів на Python використовуємо систему управління пакетами pip

pip - використовується для установки і управління програмними пакетами, написаними на Python.

Для версії Python 3.5, pip поставляється разом з інтерпретатором python.

Якщо з якоїсь причини pip відсутній, то він може бути завантажена з https://bootstrap.pypa.io/

Назва файлу: `get-pip.py`

Інструкція по pip лежить тут:
https://pip.pypa.io/en/latest/installing/

Приклад. Як завантажити пакет PyGame під Windows

1. Вибираємо «Пуск-Выполнить»
2. В рядку «Открыть» набираємо «cmd». ОК.
3. У вікні «Администратор» набираємо: pip install PyGame

При цьому має бути підключення до мережі Internet