Лекція 4

Оператори мови Python

Навіщо потрібні оператори

Оператори алгоритмічної мови використовують для виконання з даними певних дій.

1. Оператори присвоювання використовують для збереження даних у змінній.
2. Математичні оператори дозволяють виконати арифметичні обчислення.
3. Оператор конкатенації рядків використовують для з'єднання двох рядків в один і т. д.

Розглянемо оператори, доступні в Python 3, докладніше.

Математичні оператори

Виконувати операції над числами дозволяють наступні оператори:

+ — додавання: $12+3=15, 10.4+5.4=15.8, 10+11.2=21.2$

```
# Цілі числа
>>> 12+3                    print(12+3)
15                          Результат: 15
#Дійсні числа
>>> 10.4 + 5.4              print(10.4+5.4)
15.8                        Результат: 15.8
#Цілі й дійсні числа
>>> 10 + 11.2               print(10+11.2)
21.2                        Результат: 21.2
```

Математичні оператори (2)

– — віднімання: $11-3=8,\ 14.4-5.2=9.2, 15-6.2=8.8$

```
# Цілі числа
>>> 12-3                    print(11-3)
8                           Результат: 8
#Дійсні числа
>>> 14.4 - 5.2              print(14.4-5.2)
9.2                         Результат: 9.2
#Цілі й дійсні числа
>>> 15 - 6.2                print(15-6.2)
8.8                         Результат: 8.8
```

Математичні оператори (3)

* — множення: $60 \times 5 = 300,\ 12.4 \times 5.2 = 64.48, 10 \times 5.2 = 52.0$

```
# Цілі числа
>>> 60 * 5                    print(60 * 5)
300                           Результат: 300
#Дійсні числа
>>> 12.4 * 5.2                print(12.4 * 5.2)
64.48                         Результат: 64.48
#Цілі й дійсні числа
>>> 10 * 5.2                  print(10*5.2)
52.0                          Результат: 52.0
```

Математичні оператори (4)

/ — ділення. $10/5=2, 10/3=3.333...$

Особливість. Результатом ділення завжди є дійсне число, навіть якщо проводиться ділення цілих чисел.

```
# Ділення цілих чисел без остачі
>>> 10/5                        print(10/5)
2.0                             Результат: 2.0
# Ділення цілих чисел з остачею
>>> 10/3                        print(10/3)
3.3333333333333335              Результат:
                                3.3333333333333335
# Ділення дійсних чисел
>>> 10.2/5.7                    print(10.2/5.7)
1.789473684210526               Результат:
                                1.789473684210526
```

Математичні оператори (5)

/ / — Ділення з округленням униз. Незалежно від типу чисел остача відкидається: $\lfloor 10/5 \rfloor = 2, \lfloor 10/3 \rfloor = 3$

Приклад 1.

```
>>> 10 // 5 # Ділення цілих чисел без остачі
2
>>> 10 // 3  # Ділення цілих чисел з остачею
3
>>> -10 // 3  # Ділення цілих чисел з остачею
-4
>>> 10.0 // 5.0 # Ділення дійсних чисел
2.0
>>> 10.0 // 3.0  # Ділення дійсних чисел
3.0
>>> -10 // 5.0 # Ділення цілого числа на дійсне
-2.0
>>> 10 // 3.0 # Ділення цілого числа на дійсне
3.0
```

Математичні оператори (6)

% — залишок від ділення: $10 \bmod 5 = 0, 10 \bmod 3 = 1$

Приклад 2.

```
>>> 10 % 5   # Ділення цілих чисел без остачі
0
>>> 10 % 3   # Ділення цілих чисел з остачею
1
>>> 10.0 % 5.0 # Операція над дійсними числами
0.0
>>> 10.0 % 3.0 # Операція над дійсними числами
1.0
>>> 10%5.0 # Операція над цілими й дійсними числами
0.0
>>> 10%3.0 # Операція над цілими й дійсними числами
1.0
```

Математичні оператори (7)

** — піднесення до степеня: $10^2 = 100,\ 10.0^2 = 100.0$

>>> 10**2,10.0**2	print(10**2)
(100, 100.0)	100

унарный мінус (-) і унарный плюс (+) :

```
>>> +10, +10.0, -10, -10.0, -(-10), -(-10.0)
    (10,  10.0, -10, -10.0,    10,      10.0)
```

Як видно із прикладів, операції над числами різних типів повертають число, що має більш складний тип з типів, що брав участь в операції.

Складність типів розглядають в такому порядку:

1. Цілі числа мають найпростіший тип

2. Наступними по складності йдуть дійсні числа

3. Найскладніший тип — комплексні числа.

Таким чином, якщо в операції беруть участь ціле число й дійсне, то ціле число буде автоматично перетворене в дійсне число, а потім зроблена операція над дійсними числами. Результатом цієї операції стане дійсне число.

Точність представлення дійсного числа обмежена

При виконанні операцій над дійсними числами слід враховувати обмеження точності обчислень. Наприклад, результат наступної операції може здатися дивним:

Приклад 3.

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

Очікуваним був би результат 0.0, але, як видно з прикладу, ми одержали зовсім інший результат. Якщо необхідно виконувати операції з фіксованою точністю, то слід використовувати модуль `decimal`:

Приклад 4.

```
>>> from decimal import Decimal
>>> Decimal("0.3")-Decimal("0.1")-Decimal("0.1")-Decimal("0.1")
Decimal('0.0')
```

Двійкові оператори

Побітові оператори призначені для маніпуляції окремими бітами. Мова Python підтримує наступні побітові оператори:

~ — двійкова інверсія

Побітова інверсія числа x визначається як $-(x+1)$

Приклад 5.

```
>>> x = 0b10
>>> x = ~x
>>> bin(x)
'-0b11'

>>> x = -0b11
>>> x = ~x
>>> bin(x)
'0b10'
```

& — двійкове AND:

Таблиця істинності для операції &

0 & 0 = 0

0 & 1 = 0

1 & 0 = 0

1 & 1 = 1

Приклад 6.

```
>>> x = 0b101
>>> y = 0b110
>>> z = x & y
>>> bin(z)
'0b100'

>>> "{0:b} & {1:b} = {2:b}". format(x, y, z)
'101 & 110 = 100'
```

| — двійкове OR:

Таблиця істинності для операції |

$$0 \mid 0 = 0$$
$$0 \mid 1 = 1$$
$$1 \mid 0 = 1$$
$$1 \mid 1 = 1$$

Приклад 7.

```
>>> x = 0b101
>>> y = 0b110
>>> z = x | y
>>> bin(z)
'0b111'

>>> "{0:b}|{1:b}={2:b}".format(x, y, z)
'101 | 110 = 111'
```

^ — операція XOR:

Таблиця істинності для операції

$$0 \mid 0 = 0$$
$$0 \mid 1 = 1$$
$$1 \mid 0 = 1$$
$$1 \mid 1 = 0$$

Приклад 8.

```
>>> x = 0b001
>>> y = 0b111
>>> z = x ^ y
>>> bin(z)
'0b110'
>>> "{0:b}^{1:b}={2:b}".format(x, y, z)

'1 ^ 111 = 110'
```

x<<y — зсув вліво – зсуває двійкове представлення числа x вліво на y розрядів, а розряди праворуч заповнюються нулями. Це те саме, що виконати операцію x*y**2

```
   1111
  11110 ←
 111100 ←
1111000 ←
11110000
```

Приклад 9.

```
>>> x = 0b1111 # Введення x
>>> y = x << 1 # Зсув на 1 розряд
>>> bin(y)
'0b11110'
>>> z = y << 1 # Зсув на 1 розряд
>>> bin(z)
'0b111100'
>>> k = z << 2; bin(k) # Зсув на 2 розряди
'0b11110000'
```

x>>y - зсув вправо додатного числа –
зсуває двійкове представлення числа x вправо на y розрядів. Це те саме, що виконати операцію x//y**2

Приклад 10.

Додатне число	**Від'ємне число**

>>> x = 0b11
>>>y=x>>1; bin(y)
'0b1'
Оскільки 3//2=1
>>>x=0b1001
>>>y=x>>>1; bin(y)
'0b100'
Оскільки 9//2=4
	>>> x=-0b11
	>>> y=x>>1; bin(y)
	'-0b10'
	Оскільки -3//2=-2
	>>> x=-0b1001
	>>> y=x>>1; bin(y)
	'-0b101'
	Оскільки -9//2=-5

Оператори для послідовностей

Для роботи з послідовностями призначені наступні оператори:

\+ - конкатенація:

```
>>> print("Рядок1"+"Рядок2") # Конкатенація рядків
Рядок1Рядок2
>>> [ 1, 2, 3] + [ 4, 5, 6]    # Рядки
[1, 2, 3, 4, 5, 6]
>>> ( 1, 2, 3) + ( 4, 5, 6)    # Кортежі
(1, 2, 3, 4, 5, 6)
Для конкатенації змінних типу dict
>>> x={'a':1,'b':2}
>>> y={'c':3,'d':4}
>>> x.update(y)
>>> x
{'d': 4, 'c': 3, 'b': 2, 'a': 1}
```

* - повторення:

```
Повторити 20 раз символ s
>>> "s"*20   # Рядок
'ssssssssssssssssssss'

Повторити 3 раза список (list):
>>> [1, 2] * 3   # Список
[ 1, 2, 1, 2, 1, 2]

Повторити 3 раза кортеж (tuple):
>>> ( 1, 2) * 3     # Кортеж
(1, 2, 1, 2, 1, 2)
```

in - перевірка на входження.

Якщо елемент входить у послідовність, то повертається логічне значення `True`:

```
>>> "Рядок" in "Рядок для пошуку" # Рядки
True

>>> "Рядок2" in "Рядок для пошуку" # Рядки
False

>>> 2 in [1,2,3], 4 in [1,2,3]  # Рядки
(True, False)

>>> 2 in (1,2,3), 6 in (1, 2, 3) # Кортежі
(True, False)
```

not in - перевірка на невходження. Якщо елемент не входить у послідовність, повертається True:

```
>>> "Рядок" not in "Рядок для пошуку" # Рядки
False
>>> "Рядок2" not in "Рядок для пошуку" # Рядки
True
```

Приклад 11.

```
>>> 2 not in [1,2,3], 4 not in [1,2,3] # список
(False, True)

>>> 2 not in [1,2,3]; 4 not in [1,2,3] # список
False
True
```

```
>>> 2 not in (1,2,3), 6 not in (1,2,3) #кортеж
(False, True)
```

Оператори присвоювання

Оператори присвоювання призначені для збереження значення в змінній.

= - присвоює змінній значення:

```
>>> x = 5; x
5
```

+= - збільшує значення змінної на зазначену величину:

```
>>> x = 5; x += 10 # Еквівалентно x = x + 10
>>> x
15
```

Для послідовностей оператор += виконує конкатенацію:

```
>>> s = "Ряд"; s += "ок"
>>> print(s)
Рядок
```

-= - зменшує значення змінної на зазначену величину:

```
>>> x = 10; x -= 5 # Еквівалентно x = x - 5
>>> x
5
```

*= - множить значення змінної на зазначену величину:

```
>>> x = 10; x *= 5 # Еквівалентно x = x * 5
>>> x
50
```

Для послідовностей оператор *= виконує повторення:

```
>>> s = "*"; s *= 20
>>> s
'********************'
```

/= - ділить значення змінної на зазначену величину:

```
>>> x = 10; x /= 3   # Еквівалентно x = x / 3
>>> x
3.3333333333333335
>>> y = 10.0; y /= 3.0 # Еквівалентно y = y/3.0
>>> y
3.3333333333333335
```

//= - ділення **з округленням вниз** і **присвоюванням:**

Приклад 12.

```
>>> x = 10; x //= 3 # Еквівалентно x = x // 3
>>> x
3
>>> x = -10; x //= 3 # Еквівалентно x = x // 3
>>> x
-4
>>> y = 10.0; y //= 3.0 # Еквівалентно y=y//3.0
>>> y
3.0
```

% = - ділення по модулю й присвоювання:

Приклад 13.

```
>>> x = 10; x %= 2  # Еквівалентно x = x % 2
>>> x
0

>>> y = 10; y %=3   # Еквівалентно y = y % 3
>>> y
1
```

** = - піднесення до степеня і присвоювання:

```
>>> x = 10; x **= 2 #Еквівалентно x = x ** 2
>>> x
100
```

Пріоритет виконання операторів

У якій послідовності буде обчислюватися вираз?

```
x = 5 + 10 * 3/2
```

Це залежить від пріоритету виконання операторів. У цьому випадку послідовність обчислення виразу буде такою:

1. Число 10 буде помножено на 3, тому що пріоритет оператора множення вищий за пріоритет оператора додавання.
2. Отримане значення буде поділено на 2, тому що пріоритет оператора ділення рівний пріоритету оператора множення (а оператори з рівними пріоритетами виконуються зліва направо), але вищий, ніж в оператора додавання.
3. До отриманого значення буде додане число 5, тому що оператор присвоювання = має найменший пріоритет.
4. Значення буде присвоєно змінній x.

```
>>> x = 5 + 10 * 3/2
>>> x
20.0
```

За допомогою дужок можна змінити послідовність обчислення виразу:

```
x = (5 + 10) * 3/2
```

Тепер порядок обчислень стане іншим:

1. До числа 5 буде додано 10.
2. Отримане значення буде помножено на 3.
3. Отримане значення буде поділено на 2.
4. Значення буде присвоєно змінній x.

```
>>> x = (5 + 10) * 3/2
>>> x
22.5
```

Оператори в порядку зменшення пріоритету:

1. `-x, +x, ~x, **` - унарный мінус, унарный плюс, двійкова інверсія, піднесення до степеня.

Якщо унарні оператори розташовані зліва від оператора **, то піднесення до степеня має більший пріоритет, а якщо праворуч - то менший. Наприклад, вираз $-10^{-2} \rightarrow$ `-10 ** -2`

еквівалентний наступному розміщенню дужок: `-(10 ** (-2))`

2. `*, %, /, //` - множення (повторення), залишок від ділення, ділення, ділення з округленням униз.
3. `+, -` - додавання (конкатенація), віднімання.
4. `<<, >>` - двійкові зсуви.
5. `&` -двійкове AND.
6. `^` двійковий XOR.
7. `|` - двійкове OR.
8. `=, +=, -=, *=, /=, //=, % =, **=` - присвоювання.

Оператори порівняння

Оператори порівняння є логічними виразами, які повертають тільки два значення: `True` (істина) або `False` (неправда), які поводяться як цілі числа 1 і 0 відповідно:

```
>>> True + 2 # Еквівалентно 1 + 2
3
>>> False + 2 # Еквівалентно 0 + 2
2
```

Логічне значення можна зберегти в змінній:

```
>>> x = True; y = False
>>> x, y
(True, False)
```

Будь-який об'єкт у логічному контексті може бути інтерпретований як істина (`True`) або як неправда (`False`).

Використання функції `bool()` для визначення логічного значення об'єкта

Значення `True` повертають наступні об'єкти:

будь-яке число, що не дорівнює нулю:

Приклад 14.

```
>>> bool(1), boo1(20), bool(-20)
(True, True, True)
>>> bool(1.0), bool(0.1), bool(-20.0)
(True, True, True)
```

не порожній об'єкт:

```
>>> bool("0"), bool([0,None])
(True, True)
>>> bool((None,)), bool({"x":5})
(True, True)
```

Наступні об'єкти інтерпретують як `False`:

Приклад 15

число, що дорівнює нулю:

```
>>> bool(0), bool(0.0)
(False, False)
```

порожній об'єкт:

```
>>> bool(""), bool([]), bool(())
(False, False, False)
```

значення None:

```
>>> bool(None)
False
```

Логічні вирази в операторах порівняння

== - дорівнює:

```
>>> 1 == 1, 1 == 5
(True, False)
```

!= - не дорівнює:

```
>>> 1 != 5, 1 != 1
(True, False)
```

< - менше:

```
>>> 1 < 5, 1 < 0
(True, False)
```

> - більше:

```
>>> 1 > 0, 1 > 5
(True, False)
```

Логічні вирази (2)

<= - менше або дорівнює:

```
>>> 1 <= 5, 1 <= 0, 1 <= 1
(True, False, True)
```

>= - більше або дорівнює:

```
>>> 1 >= 0, 1 >= 5, 1 >= 1
(True, False, True)
```

in - перевірка на входження в послідовність:

```
>>> "Рядок" in "Рядок для пошуку" # Рядки
True
>>> 2 in [1,2,3], 4 in [1,2,3]       # Списки
(True, False)
>>> 2 in (1,2,3), 4 in (1,2,3)       # Кортежі
(True, False)
```

Оператор **in** можна також використовувати для перевірки існування ключа словника:

```
>>>"x" in {"x":1, "y":2}, "z" in {"x":1, " y":2}
(True, False)
```

Логічні вирази (3)

`not in` - перевірка на невходження в послідовність:

```
>>> "Рядок" not in "Рядок для пошуку" # Рядки
False
>>> 2 not in (1, 2, 3], 4 not in [1, 2, 3] # Списки
(False, True)
>>> 2 not in (1, 2, 3), 4 not in (1, 2, 3) # Кортежі
(False, True)
```

`is` – перевіряє, чи посилаються дві змінні на той самий об'єкт. Якщо змінні посилаються на той самий об'єкт, то оператор `is` повертає значення `True`:

```
>>> x = y = [1, 2]
>>> x is y
True
>>> x = [1, 2] ; y = [1, 2]
>>> x is y
False
```

Кешування малих об'єктів

Слід помітити, що з метою підвищення ефективності інтерпретатор виконує кешування малих цілих чисел і невеликих рядків.

Це означає, що якщо ста змінним присвоєно число 2, то в цих змінних буде збережено посилання на той самий об'єкт.

Приклад 16.

```
>>> x = 2; y = 2; z = 2
>>> x is y, y is z
(True, True)
>>> x=3
>>> x is y
False
```

`is not` - перевіряє, чи посилаються дві змінні на різні об'єкти

Якщо це так, повертається значення `True`:

```
>>> x = y = [ 1, 2)
>>> x is not y
False
>>> x = [ 1, 2] ; y = [1, 2]
>>> x is not y
True
```

Значення логічного виразу можна інвертувати за допомогою оператора `not`:

```
>>> x = 1; y = 1
>>> x == y
True
>>> not (x == y), not x == y
(False, False)
```

Якщо змінні `x` та `y` дорівнюють одна одній, то повертається значення `True`.

Через те, що перед виразом стоїть оператор `not`, вираз поверне `False`.

Круглі дужки можна не вказувати, оскільки оператор `not` має більш низький пріоритет виконання, ніж оператори порівняння.

У логічному виразі можна вказувати одночасно кілька умов:

Приклад 17

```
>>> x = 10
>>> 1 < x < 20, 11 < x < 20
(True, False)
```

Кілька логічних виразів можна об'єднати в один великий за допомогою наступних операторів:

`and` – логічне `AND`. Якщо `x` у виразі `x and y` інтерпретується як `False`, то повертається `x`, а якщо ні, то – `y`:

Приклад 18

```
>>> 1 < 5 and 2 < 5 True and True == True
True

>>> 1 > 5 and 2 < 5 #false and True == False
False


>>> 10 and 20, 0 and 20, 10 and 0
(20, 0, 0)
```

`or` – логічне `OR`. Якщо `x` у виразі `x or y` інтерпретується як `False`, то повертається `y`, а якщо ні, то – `x`:

Приклад 19

```
>>> 1 < 5 or 2 < 5 # True or True == True
True
>>> 1 < 5 or 2 > 5 # True or False == True
True
>>> 1 > 5 or 2 < 5 # False or True == True
True
>>> 1 > 5 or 2 > 5 # False or False == False
False
>>> 10 or 20, 0 or 20, 10 or 0
(10, 20, 10)
>>> 0 or "" or None or [] or "s"
's'
```

Наступний вираз поверне `True` тільки у випадку, якщо обидва вирази повернуть `True`:

```
x1 == x2 and x2 != x3
```

А цей вираз поверне `True`, якщо хоча б один з виразів поверне `True`:

```
x1 == x2 or x3 == x4
```

Оператори порівняння в порядку зменшення пріоритету:

1. `<, >, <=, >=, =, ! =, <>, is, is not, in, not in.`
2. `not` - логічне NOT.
3. `and` - логічне AND.
4. `or` - логічне OR.