Операторы сравнения

Всего в python доступны 6 операторов:

- **<** - меньше
- **>** - больше
- **<=** - меньше или равно
- **>=** - больше или равно
- **==** - равно
- **!=** - не равно

Определим несколько переменных

In [ ]:
```
a = 5
b = -3
c = 8
```

Выведем логические значения, которые получаются в результате использования операторов.

- **True** - если условие выполняется
- **False** - если условие не выполняется

In [ ]:
```
print(a > 0)
print(b == 7)
print(a*b*c != 3)
```

```
True
False
True
```

Условные операторы

Работа с оператором **if** предполагает схему:

Если *условие*:

> Блок кода, который надо выполнить, если условие выполнятся.

Частные случаи:

- **if True:** - внутренний блок кода выполняется всегда
- **if False:** - внутренний блок кода не выполняется никогда

In [ ]:
```
if True:
    print('Истина')
```

```
if False:
    print('Ложь')
```

```
Истина
```

Ещё немного примеров

In [ ]:
```
if a > 0:
    print('a - положительное')

if a >= 0 and c >= 0:
    print('a и c - неотрицательные')

if b != 42:
    print('b не равно 42')
```

```
a - положительное
a и c - неотрицательные
b не равно 42
```

Конструкции условных операторов могут быть сложнее.

- **if** - если
- **elif** - иначе если (**else + if**)
- **else** - иначе

Введем с клавиатуры значение t, после чего проверим его значение. Если $t \in [-1, 1]$, будем считать его нормальным.

Помним, что изначально с помощью **input()** вводится строка, поэтому сначала преобразуем её к типу **float**.

In [ ]:
```
t = float(input())

if t < -1:
    print('Слишком мало')
elif t > 1:
    print('Слишом много')
else:
    print('Нормальное значение')
```

```
Нормальное значение
```

Конструкция match/case

In [ ]:
```
p = int(input())

match p:
    case 1:
        print('Один')
    case 2:
        print('Два')
    case _:
        print('Другое значение')
```

```
Один
```

Циклы

Оператор **while**

Покажем его работу на примере.

Тело цикла выполняется **до тех пор, пока** выполняется условие

In [ ]:
```
j = 0

while j < 5:
    print(j)
    j += 1
```

```
0
1
2
3
4
```

Оператор **while** удобно использовать, если число итераций заранее не известно, но стоит помнить что если условие будет выполняться всегда, цикл станет бесконечным

In [ ]:
```
# j=0

# while j<5:
#     print(j)
```

Иногда работать с бесконечными циклами удобно. Покажем это на предыдущем примере.

Если значение "нормальное", цикл прерывается командой **break**

In [ ]:
```
while True:
    t = int(input())
    if t < -1:
        print('Слишком мало')
    elif t > 1:
        print('Слишом много')
    else:
        print('Нормальное значение')
        break
```

```
Нормальное значение
```

Оператор **for**

Этот оператор работает по схеме:

Для *i* **в** <*Диапазон\список*>:

> Блок кода, который надо выполнить при текущем значении *i*.

range(N) задаёт диапазон целых чисел от 0, до $N-1$

In [ ]:
```
for i in range(5):
    print(i)
```

```
0
1
2
3
4
```

Можно указать и дополнительные параметры **range(start, stop, step)**:

- **start** - с какого начать
- **stop** - перед каким остановиться
- **step** - с каким шагом двигаться

In [ ]:
```
for i in range(2, 16, 2):
    print(i)
```

```
2
4
6
8
10
12
14
```

В цикле **for** можно проходить не только по целым числам, но и, например, по элементам списка

In [ ]:
```
values = [1, 2, 'asd', 5, '6sda', 1.3]

for item in values:
    print(item)
```

```
1
2
asd
5
6sda
1.3
```

Также полезно помнить про функцию **enumerate()**, которая позволяет пробегать и по индексам, и по элементам сразу же

In [ ]:
```
for i, item in enumerate(values):
    print(f'Элемент №{i+1} - это {item}')
```

```
Элемент №1 - это 1
Элемент №2 - это 2
Элемент №3 - это asd
Элемент №4 - это 5
Элемент №5 - это 6sda
Элемент №6 - это 1.3
```

Два слова* про функции

* подробнее будет в следующих сериях

- Задаётся с помощью ключеевого слова **def**, затем следует название функции, после чего в круглых скобках входные параметры.
- Тело функции идёт с отступом.
- Функция возвращает параметры с помощью ключевого слова **return**.
- Функция - своего рода набор инструкций, которые выполняются в момент вызова функции в основном коде

In [ ]:
```
def sum(a, b):
    return a+b


sum(2, 3)
```

Out[ ]:
```
5
```

Задание

Дана булева фунция от двух переменных, вывести таблицу истинности.

x_1	x_2	$f(x_1, x_2)$
0	0	
0	1	
1	0	
1	1	

Рассмотрим импликацию $x \to y = \neg x \vee y$ в качестве булевой функции

Решение

In [ ]:
```
def boolean_function(x, y):
    result = (not x) or y
    return int(result)


x = [0, 0, 1, 1]
y = [0, 1, 0, 1]

header = '| x | y | x-->y |'
hline = '-'*len(header)
```

```
print(header)
print(hline)
for i in range(len(x)):
    print(f'| {x[i]} | {y[i]} |   {boolean_function(x[i],y[i])}   |')
```

```
| x | y | x-->y |
-----------------
| 0 | 0 |   1   |
| 0 | 1 |   1   |
| 1 | 0 |   0   |
| 1 | 1 |   1   |
```