Логический тип bool. Операторы сравнения и операторы and, or, not

На этом занятии вы увидите, как в программах можно делать логические выводы на уровне:

True – истина;
False – ложь.

Предположим, мы бы хотели узнать: `4 > 2`

Для этого достаточно записать такое сравнение и Python возвратит результат: либо True, либо False. В данном случае получаем True (истина), так как 4 действительно больше 2. Если же записать:

`4 > 7`

то получим False, так как 4 меньше 7. То же самое можно делать и с переменными:

```
a = 5
b = 7.8
a <= b
```

Мало того, полученный результат, при необходимости, можно сохранить в переменной, например, так: `res = a > b`

получим значение False в переменной res.

Давайте теперь посмотрим на тип этой переменной: `type(res)`

да, это новый тип bool (булевый тип), с которым мы только что познакомились и он может принимать только два значения: True или False.

Мы можем присвоить эти значения напрямую переменной, например, так: `res = False`

И здесь, естественно, возникает вопрос, какие операторы сравнения вообще существуют в Python? Они, следующие:

<	сравнение на меньше
>	сравнение на больше
<=	сравнение меньше или равно
>=	сравнение больше или равно
==	сравнение на равенство
!=	сравнение на неравенство

Приведу несколько примеров их использования: `5 == 7-2`

Здесь справа записана арифметическая операция. Так тоже можно делать. Вообще, в качестве операндов могут быть любые конструкции языка Python: переменные, вычисления, функции и т.п.

Следующий пример, сравнения: `8 >= 8` `8 > 8`

В первом случае получаем True, а во втором – False, так как 8 не больше 8. Также имейте в виду, что оператор >= записывать наоборот => нельзя. Помимо этого знаки: **>=, !=, <=, ==**

всегда записываются без пробелов. Если поставить пробел, будет ошибка.

Также частой, нет, очень частой ошибкой начинающих программистов – это запись вместо оператора сравнения == оператора присваивания =. Сравните две строчки:

```
x = 6
x == 6
```

В первом случае мы присваиваем переменной x число 6, а во втором случае сравниваем переменную x со значением 6. И эти операторы путать нельзя, хотя они и похожи друг на друга.

Еще несколько полезных примеров сравнения. Предположим, мы хотим проверить переменную x на четность. Я напомню, что четные числа – это, например, числа:

2, 4, 6, -8, -10

то есть, числа, кратные 2. Как выполнить такую проверку? Обычно, это делают, следующим образом: `x % 2 == 0`

Помните оператор вычисления остатка от деления? Он здесь будет возвращать 0, если двойка укладывается в x без остатка. Это и есть проверка на четность.

Обратную проверку на нечетность, можно записать так: `x % 2 != 0`

По аналогии мы можем проверить, кратна ли переменная какому-либо значению, например, трем: `a % 3 == 0`

и так можно делать проверки с любыми числами.

Операторы not, and, or и их приоритеты

Давайте теперь поставим более сложную задачу и определим, попадает ли число (значение переменной) в диапазон [-2; 5]? Какие логические умозаключения здесь нужно провести, чтобы ответить на этот вопрос? Для простоты представим, что у нас имеется переменная: `y = 1.85`

Очевидно, чтобы она попадала в диапазон [-2; 5], нужно соблюдение двух условий: `y >= -2` /\ `y <= 5`

В Python записать это можно, следующим образом: `y >= -2 and y <= 5`

Здесь общее условие будет истинно, если истинно каждое из подусловий.

Противоположный пример, вывод о непопадании в этот же диапазон. Его можно сделать проверками, что: `y < -2` или `y > 5`

В Python это записывается так: `y < -2 or y > 5`

Оператор or выдает истину, если истинно, хотя бы одно из подусловий. Понимаете разницу между and и or? В and общее условие истинно, если истинны оба подусловия, а в or – хотя бы одно.

Также, если мы используем оператор and, то условие: `y >= -2 and y <= 5`

можно записать в краткой форме: `-2 <= y <= 5`

Это будет одно и то же. Какой вариант записи использовать, решать только вам, что удобнее, то и пишите в своих программах.

Приведу еще один пример. Допустим, мы хотим проверить x на кратность 2 или 3. Очевидно, это можно записать так: `x % 2 == 0 or x % 3 == 0`

Также можно инвертировать это условие, то есть, определить, что x не кратна 2 и не кратна 3. Это можно сделать двумя способами:

`x % 2 != 0 and x % 3 != 0`

Или, используя оператор not (не): `not (x % 2 == 0 or x % 3 == 0)`

Обратите внимание на круглые скобки. У оператора not самый высокий приоритет, то есть, он выполняется в первую очередь (как умножение и деление в математике). Поэтому, для инвертирования всего составного условия его необходимо поместить в круглые скобки. Если бы мы записали все в виде:

`not x % 2 == 0 or x % 3 == 0`

Это было бы эквивалентно условию: `x % 2 != 0 or x % 3 == 0`

то есть, мы инвертировали бы только первое выражение. Поэтому, при определении составных условий очень важно знать приоритеты операторов:

or	1
and	2
not	3