Тернарный условный оператор. Вложенное тернарное условие

На этом занятии по курсу языка Python мы поговорим о тернарном условном операторе, который появился в версии 2.5. Как всегда, вначале поясню, что это за оператор. В простейшем варианте его синтаксис можно представить так:

<значение 1> if <условие> else <значение 2>

Он возвращает значение 1, если условие истинно, а иначе – значение 2. Давайте поясню его работу на конкретном примере. Предположим, у нас имеются две переменные:

```
a = 12
b = 7
```

и мы хотим для них определить максимальное значение. Используя имеющиеся знания, это можно было бы сделать так:

```
if a > b:
    res = a
else:
    res = b

print(res)
```

А с использованием тернарного оператора это можно реализовать так:

```
res = a if a > b else b
```

Видите, какая простая, понятная и компактная запись в итоге получилась. Это одно из удобств данного оператора. Но, все же, между предыдущей программой с условным оператором if и тернарным оператором есть одно принципиальное отличие. Тернарный оператор автоматически возвращает результат. В данном примере – это или переменная a или переменная b. Тогда как обычный условный оператор предназначен для выполнения блока кода по определенному условию и сам по себе не возвращает никаких значений. То есть, если бы мы попытались записать что-то вроде:

```
d = if a > b:
```

то попросту возникла бы синтаксическая ошибка – так делать нельзя. А вот результат работы тернарного оператора мы, обычно, сохраняем в некой переменной.

Второе важное отличие – в тернарном операторе нет внутренних блоков, где бы мы могли записывать несколько операторов. Вместо a и b можно прописывать только одну какую-либо конструкцию. Часто это некое значение, или результат работы операторов, например, арифметических:

```
res = a + 2 if a > b else b - 5
```

Так делать вполне допустимо. Или же, можно использовать какие-либо функции:

```
a = -12
b = -7
res = abs(a) if a > b else abs(b)
```

То есть, здесь может быть любая конструкция языка Python, но только одна. По идее, можно даже записать функцию print():

```
res = print(a) if a > b else prin
```

Но тогда переменная res будет принимать значение None, так как функция print() ничего возвращает, то есть, None.

```
s = 'python'
type = 'upper'
```

Давайте приведу еще один пример. Пусть имеется некая строка и способ преобразования этой строки:

Пусть 'upper' означает преобразование строки s в верхний регистр. Сделаем это через тернарный оператор, следующим образом:

```
res = s.upper() if type == 'upper
```

Видите, как элементарно и просто реализуется эта идея? В этом большой плюс тернарного оператора. Вообще, его можно рассматривать, как некий активный объект, который возвращает определенное значение в зависимости от условия. Благодаря этому, его можно вызывать прямо внутри различных конструкций, например:

```
[1, 2, a if a < b else b, 4, 5]
"a - " + ("четное" if a % 2 == 0 else 'нечетное')
```

Обратите внимание здесь на круглые скобки. Они необходимы, чтобы следующий оператор + применялся к результирующей строке, а не к тернарному оператору. Если круглые скобки убрать, то последняя добавка пропадет.

Или его можно указать, как один из аргументов функции: