Условный оператор if. Конструкция if-else

Задачи

На этом занятии мы свами познакомимся с работой условного оператора if. Что он делает и для чего нужен? Помните, на одном из прошлых занятий, мы с вами рассматривали операции сравнения, например, проверяли:

```
4 > 2
```

или с переменной:

```
a = 5
a < 0
```

На выходе получали булевы значения True и False. Но ничего не говорили, как их дальше использовать в программе. Пришло время восполнить этот пробел. Оператор if позволяет выполнить группу операторов, при истинности указанного условия. Давайте я сразу поясню эту конструкцию на конкретном примере. Предположим, мы пишем программу для вычисления модуля числа:

```
x = -4
if x < 0:
    x = -x
print(x)
```

Смотрите, после оператора if записываем условие, то есть, определяем, что x отрицательное число. И если это так, то оператор меньше вернет значение True, условие сработает и будет выполнена строчка x = -x. А вот следующая строчка (функция print()) стоит уже вне этого условия и будет выполняться всегда. Так как Python
«понимает», что находится в блоке условного оператора, а что нет?Все дело в форматировании текста программы. Когда мы ставим двоеточие, то открывается новый блок, где может располагаться множество операторов. И все конструкции языка Python, что имеют отступ (обычно четыре пробела) относительно оператора if, будут располагаться внутри этого блока. Именно поэтому строчка x = -x выполняется только при истинности условия x < 0, а функция print() стоит уже после оператора if и выполняется всегда.

Запустим эту программу и убедимся, что все работает. Видим значение 4, то есть, условие x < 0 вернуло значение True, оператор if сработал и выполнилась строчка x = -x.

А давайте теперь определим:

```
x = 4
```

В этом случае также увидим 4, но строка x = -x уже не выполнялась, так как условие для значения False не срабатывает. И, смотрите, если функцию print() сместить на уровень предыдущего оператора:

```
x = 4
if x < 0:
    x = -x
print(x)
```

то в консоли мы уже ничего не увидим. Это произошло, как раз, из-за того, что эта функция переместилась внутрь блока оператора if и будет теперь выполняться только при истинности условия. Вернем, снова: `x = -4`

и, теперь, видим значение 4 в консоли. Этот пример показывает, что в Python форматирование текста программы имеет ключевое значение. И благодаря этому, кстати, программы становятся более наглядными и читабельными, а также заставляют начинающих программистов правильно их оформлять. На мой взгляд – это большой плюс данного языка программирования.

Чтобы все было понятно, приведу еще пару примеров. Предположим, мы хотим поменять значения двух переменных:

```
a = float(input("a: "))
b = float(input("b: "))
```

Если a меньше b:

```
if a < b:
    a, b = b, a
print(a, b)
```

Также можно прописывать более сложные, составные условия, например, для проверки попадания числа в заданный диапазон:

```
x = int(input())
if x >= -4 and x <= 10:
    print("x в диапазоне [-4; 10]")
```

О составных условиях мы с вами уже говорили на отдельном занятии, поэтому здесь все должно быть понятно.

Или, то же самое условие в Python допускается записывать и так:

```
if -4 <= x <= 10:
```

То есть, в качестве условия можно записывать любые конструкции, которые можно интерпретировать как истину (True) и ложь (False). Даже так, просто указав числовое значение:

```
x = int(input())
if x:
    print("x = True")
```

Мы здесь получим истину, если x не равен нулю.

Наконец, мы можем использовать операторы проверки для списков. Например, студент имеет следующие оценки по результатам сдачи сессии:

```
marks = [3, 3, 4, 2, 4]
```

и мы хотим узнать, будет ли он отчислен. Выполним для этого, следующую проверку:

```
if 2 in marks:
    print("студент будет отчислен
```

Вот в этом последнем примере нам бы хотелось в противном случае выдать сообщение, что студент успешно сдал сессию. Это лучше всего реализовать с помощью оператора else:

```
marks = [3, 3, 4, 2, 4]
if 2 in marks:
    print("студент будет отчислен
else:
    print("студент успешно сдал с
```

То есть, смотрите, если срабатывает первое условие, то блок else не выполняется. И, наоборот, если условие не срабатывает, то выполняется блок else. Это слово можно перевести как «иначе» и получается либо выполнение первого блока, либо второго, но никогда не обоих вместе. То есть, здесь определены два взаимоисключающих случая. Действительно, студен не может одновременно быть отчисленным и успешно сдавшим сессию (будем полагать, что по собственному желанию он не уходит).

Или другой подобный пример:

```
x = int(input())
if x < 0:
    print("x - отрицательное числ
else:
    print("x - неотрицательное чи
```

Очевидно, что x не может быть одновременно отрицательным и неотрицательным, поэтому для проверки такого условия можно использовать оператор else.

В последнем примере этого занятия мы проверим введенное число на четность. Сделать это можно так:

```
x = int(input())
if x % 2 == 0:
    print("x - четное число")
else:
    print("x - нечетное число")
```

Надеюсь, из всех этих примеров, вам понятно, что из себя представляет условный оператор if, как в нем можно прописывать различные условия и как работает оператор else. Вас еще ждет набор практических заданий, а я буду вас ждать на следующем уроке.