Условный оператор. Отступы. Операции над строками

Аннотация

На этом уроке мы впервые познакомимся с одной из фундаментально важных тем в программировании — условным оператором. Он позволяет организовать ветвление в вашей программе (выполнение одной ветки кода в зависимости от условия).

1.Повторение

На прошлом уроке мы познакомились с переменными. Переменная имеет имя и значение. Имя переменной может начинаться только с буквы и включать в себя буквы, цифры и символ подчеркивания. Имя переменной должно отражать ее назначение.

Чтобы задать переменной значение, необходимо после знака равно (оператора присваивания) указать значение переменной.

Еще значение переменной можно получить из ввода. Для этого используем команду `input()`. В этом случае значение переменной задает пользователь.

2.Условный оператор

Условный оператор используется, когда некая часть программы должна быть выполнена, только **если верно** какое-либо условие. Для записи условного оператора используются ключевые слова `if` и `else` («если», «иначе»), двоеточие и **отступ в четыре пробела**.

```
if условие:
    Действия, если условие верно
else:
    Действия, если условие неверно
```

PEP 8

Отступ в четыре пробела принят в сообществе Python (PEP 8). При этом программа может работать и при других вариантах, но читать ее будет неудобно. Пробелы — самый предпочтительный метод отступов.

Табуляция должна использоваться только для поддержки кода, написанного с отступами с помощью табуляции.

Python 3 запрещает смешивание табуляции и пробелов в отступах.

Рассмотрим пример:

```
print('Введите пароль:')
password = input()
if password == 'qwerty':
    print('Доступ открыт.')
else:
    print('Ошибка, доступ закрыт!')
```

Обратите внимание: в начале условного оператора `if` выполняется сравнение, а не присваивание. Разница вот в чем.

Сравнение

Сравнение — это проверка, которая не меняет значение переменной (в сравнении может вообще не быть переменных), а присваивание — команда, которая меняет значение переменной.

Для сравнения нужно использовать двойной знак равенства: `==`.

Также заметьте, что после `else` никогда не пишется никакого условия.

Другой пример:

```
print('Представься, о незнакомец!')
name = input()
if name == 'Цезарь':
    print('Аве, Цезарь!')
else:
    print('Приветик.')
```

В качестве условия можно использовать и другие операции отношения:

```
< меньше
> больше
<= меньше или равно
>= больше или равно
== равно
!= не равно
```

PEP 8

Все операции отношения оформляются с помощью симметричных пробелов.

Правильно:

```
if bird == "Т у к - т у к":
```

~~**Неправильно:**~~

```
if bird=="Т у к - т у к":
```

Объекты любой однородной группы можно сравнивать между собой. Подумайте над тем, как можно сравнивать, например, строки.

3.Операции над строками

Во всех примерах, которые мы рассматривали, переменные хранили строки. Мы вводили, выводили и хранили строки. Кроме уже описанных действий, строки еще можно складывать.

Давайте попробуем:

```
x = '10'
y = '20'
z = x + y
print(z)
```

PEP 8

И опять немного рекомендаций по оформлению (PEP 8): ставьте пробелы вокруг знаков операций (*, +, - и т. д.)

Правильно:

```
z = x + y
```

~~**Неправильно:**~~

```
z = x+y
```

В данной программе мы задали переменным x и y значение, переменной z присвоили значение результата сложения x и y.

Результатом выполнения программы будет строка **1020**.

Конкатенация

Операция сложения для строк выполняет конкатенацию двух строк, то есть склеивает их содержимое вместе.

Например: операция «П р и» + «в е т» в результате даст строку «П р и в е т».

Обратите внимание: запись: x + y = z недопустима. Оператор присваивания ожидает слева переменную, которой надо присвоить значение, а в правой части находится значение или выражение, которое надо сначала вычислить, а затем присвоить.

Мы могли сократить нашу программу и написать в таком виде:

```
x = '10'
y = '20'
print(x + y)
```

Результат будет такой же, проверьте. Оператор print() сначала вычислил значение выражения x + y, а потом вывел на экран полученное значение.

А еще такой результат можно получить вот таким образом:

```
print('10' + '20')
```

Дублирование

Для строк также можно выполнять умножение. Умножать можно строку на число или число на строку. Операция называется дублирование. В результате нее начальная строка будет повторена заданное количество раз.

Например: 3 * "20" то же, что и "20" * 3 и , результат будет 202020 и в том, и в другом случае.

Примеры использования:

```
x = '10'
y = '20'
print(x * 2 + y * 3)
```

Что будет на экране после запуска такой программы?

4.Сравнение строк

При сравнении строк принимаются во внимание символы и их регистр. Так, цифровой символ условно меньше, чем любой алфавитный символ. Алфавитный символ в верхнем регистре условно меньше, чем алфавитные символы в нижнем регистре. Например:

```
str1 = "1a"
str2 = "aa"
str3 = "Aa"
print(str1 > str2)  # False, так как первый символ в str1 - цифра
print(str2 > str3)  # True, так как первый символ в str2 в нижнем регистре
```

Поэтому строка «1a» условно меньше, чем строка «aa». Вначале сравнение идет по первому символу. Если начальные символы обеих строк представляют цифры, то меньшей считается меньшая цифра, например, «1a» меньше, чем «2a».

Символ в верхнем регистре меньше, чем такой же в нижнем:

```
str1 = "Tom"
str2 = "tom"
print(str1 == str2)  # False - строки не равны
```

Если начальные символы представляют алфавитные символы в одном и том же регистре, то сравнивают положение в алфавите. Так, «aa» меньше, чем «ba», а «ba» меньше, чем «ca».

Если первые символы одинаковые, в расчет берутся вторые символы (при их наличии).

```
str1 = "Лесной"
str2 = "Лесная"
print(str1 > str2)  # True
# после одинаковых символов в строке 1 стоит буква "о", а в строке 2 "а"
```