Урок Условия

Условный оператор

Аннотация

Во втором 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.

Чаты 1

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.')
```

Чаты 1

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 == "Тук-тук":
```

~~**Неправильно:**~~

```
if bird=="Тук-тук":
```

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

3. Сложное условие. Логические операции

Иногда в условном операторе нужно задать сложное условие. Для этого можно использовать логические операции **and** («и»), **or** («или») и **not** («не»).

Чтобы задать одновременное выполнение двух условий, используем `and` («и»), если достаточно выполнения одного из двух вариантов (или обоих сразу) — используем `or` («или»), а если нужно убрать какой-то вариант — `not` («не»).

Приоритет выполнения операций:

1. not
2. and
3. or

Если нужно изменить приоритет операций или вы забыли правила, используйте скобки.

1 Чаты

Например, вот так можно проверить, что оба условия выполнены:

```
print('Как называются первая и последняя буквы греческого алфавита?')
greek_letter_1 = input()
greek_letter_2 = input()
if greek_letter_1 == 'альфа' and greek_letter_2 == 'омега':
    print('Верно.')
```

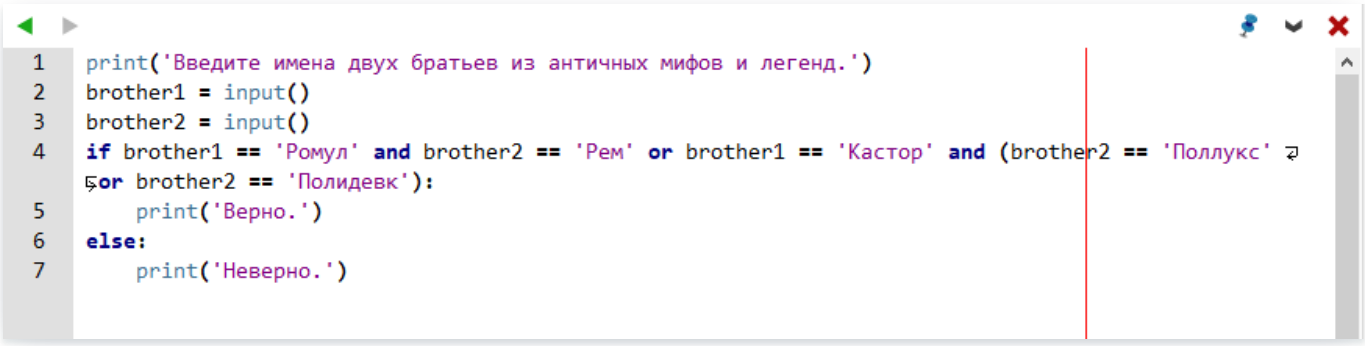
```
else:
    print('Неверно.')
```

Ниже еще несколько примеров.

```
print('Как греки или римляне называли главу своего пантеона - бога грома?')
ancient_god = input()
if ancient_god == 'Зевс' or ancient_god == 'Юпитер':
    print('Верно.')
else:
    print('Неверно.')


print('Введите имена двух братьев из античных мифов и легенд.')
brother1 = input()
brother2 = input()
if brother1 == 'Ромул' and brother2 == 'Рем' or brother1 == 'Кастор' and (brother2 == 'По
    print('Верно.')
else:
    print('Неверно.')
```

Обратите внимание: если программу из предыдущего примера вставить в IDE Wing, часть кода условного оператора будет выходить за ограничительную красную черту среды.



PEP 8

По стандарту PEP 8 длина строки должна быть ограничена максимум 79 символами.

Есть несколько способов переноса длинных строк.

- Использование подразумеваемых продолжений Python внутри круглых, квадратных и фигурных скобок: длинные строки могут быть разбиты на несколько строк, заключенных в скобки.
- Использование символа '\' (обратный слеш, или бэкслеш) для обозначения места разрыв

Чаты 1

Мы будем использовать первый способ.

Если после перенесенной строки идет один или несколько вложенных операторов (например, мы переносим строку с условием в операторе `if`), отступ у перенесенной части должен быть на четыре пробела больше, чем у вложенного оператора.

Сделайте правильные отступы для перенесенной строки.

Тогда представленный выше программный код может быть записан так:

```
print('Введите имена двух братьев из античных мифов и легенд.')
brother1 = input()
brother2 = input()
if (brother1 == 'Ромул' and brother2 == 'Рем' or brother1 == 'Кастор'
        and (brother2 == 'Поллукс' or brother2 == 'Полидевк')):
    print('Верно.')
else:
    print('Неверно.')
```

Рассмотрим еще несколько примеров.

```
print('Введите любые два слова, но это не должны быть "белый" и "медведь" разом.')
word1 = input()
word2 = input()
if not (word1 == 'белый' and word2 == 'медведь'):
    print('Верно.')
else:
    print('Неверно.')
```

А теперь попробуйте решить задачи: «Елочка, гори», «Елочка-2», «Елочка-3».

4. Вложенные условия

Блок кода

В команде `if` при выполнении условия можно выполнять более одной команды. Для этого все их необходимо выделить отступом. Такая запись называется **блоком кода**. По отступам интерпретатор определяет, при выполнении каких условий какие команды исполнять. Аналогично можно делать и для команды `else`.

```
print('Представься, о незнакомец!')
name = input()
if name == 'Цезарь' or name == 'Caesar':
    print('Аве, Цезарь!')
    print('Слава императору!')
else:
    print('Приветик.')
    print('Погода сегодня хорошая.')
print('Засим - заканчиваем.')
```

Чаты 1

Перед последней строчкой нет отступа, это означает, что она будет выполнена в конце работы программы в любом случае. А вот две предыдущие строчки будут выполнены, только если условие `if` окажется ложным.

Блоки кода в Python очень гибко устроены: внутри них можно писать любой другой код, в том числе условные операторы. Среди команд, которые выполняются, если условие `if` истинно («внутри if») или ложно («внутри else»), могут быть и другие условные операторы. Тогда команды, которые выполняются внутри этого внутреннего `if` или `else`, записываются с дополнительным отступом.

Изучите пример ниже. `elif` — это короткая запись для «`else: if`». Если не пользоваться короткой записью, if пришлось бы писать на отдельной строчке и с отступом (а все, что внутри этого `if`, — с дополнительным отступом). Это не очень удобно, и `elif` избавляет от такой необходимости.

```
print('Представься, о незнакомец!')
name = input()
if name == 'Цезарь' or name == 'Caesar':
    print('Аве, Цезарь!')
    print('В честь какого бога устроим сегодня празднество?')
    god = input()
    if god == 'Юпитер':
        print('Ура Громовержцу!')
    # если оказалось, что имя бога не 'Юпитер', то проверяем,
    # не равно ли оно строке 'Минерва'
    elif god == 'Минерва':
        print('Ура мудрой воительнице!')
    # следующая строка будет выполнена,
    # только если имя бога не 'Юпитер' и не 'Минерва'
    else:
        print('Бога по имени', god, 'мы не знаем, но слово Цезаря - закон.')
    # эта команда будет выполнена независимо от того,
    # какое имя бога ввёл пользователь, если только изначально
    # он представился Цезарем
    print('Слава императору!')
else:
    print('Приветик.')
    print('Погода сегодня хорошая.')
print('Засим - заканчиваем.')
```

А более простой вариант этой программы теперь попробуйте написать самостоятельно.

5. 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.

1
Чаты

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 «При» + «вет» в результате даст строку «Привет».

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')
```

1 Чаты

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?

6. Команда in

Теперь рассмотрим новую команду для работы со строками — команду `in`.

Команда in

Команда `in` позволяет проверить, что одна строка находится внутри другой.

Например: строка «на» находится внутри строки «сложная задача».

В таком случае обычно говорят, что одна строка является **подстрокой** для другой.

```
text = input()
if 'хорош' in text and 'плох' not in text:
    print('Текст имеет положительную эмоциональную окраску.')
elif 'плох' in text and 'хорош' not in text:
    print('Текст имеет отрицательную эмоциональную окраску.')
else:
    print('Текст имеет нейтральную или смешанную эмоциональную окраску.')
```

Первое условие окажется истинным, например, для строк «все хорошо» и «какой хороший день», но не для «ВсЕ ХоРоШо» и не для «что-то хорошо, а что-то и плохо». Аналогично второе условие окажется истинным для строк «все плохо», «плохое настроение» и т. д.

Справка
Исключительное право на учебную программу и все сопутствующие ей учебные материалы, доступные в рамках проекта «Лицей Академии Яндекса», принадлежат АНО ДПО «ШАД». Воспроизведение, копирование, распространение и иное использование программы и материалов допустимо только с предварительного письменного согласия АНО ДПО «ШАД».
Пользовательское соглашение.

© 2018 – 2022 ООО «Яндекс»

Чаты 1