Ветвления

Ветвление - это такая форма организации действий, при которой программа выполняется по одной из возможных ветвей в результате проверки условия.

Условные операторы в Python используют для разработки программ, которые учитывают разные условия и на их основе выполняют определённые действия. Чаще всего их используют для следующих задач:

Принятие решений. Позволяют программе выбирать между различными путями выполнения. Например, отправить пользователю письмо на почту или сообщение в мессенджере.

Контроль потока выполнения. Условные операторы управляют тем, какие части кода будут выполнены, а какие проигнорированы.

Оптимизация кода. Позволяют избегать выполнения ненужных операций, что улучшает производительность.

Действия в программе выполняются последовательно, но иногда некоторые операции надо пропустить. - Обработка ошибок. Условные операторы помогают обнаруживать и реагировать на ошибки или нестандартные ситуации. С помощью условных операторов можно описать, как программе реагировать на возникающие в коде ошибки.

Условным оператором в языках программирования является if

If...else

Полная форма ветвления

```
if условие:
   блок_операторов1
else:
    блок_операторов2
```

Блок_операторов1 выполняется, если **условие** принимает значение **True** (Истина). В противном случае (условие равно **False**) выполнется блок_операторов2

В качестве условия используется логическое выражение. Оно может быть простым и сложным (составным).

В простом условии сравниваются два операнда. Операции сравнения:

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

Сложные условия состоят из простых, связанных между собой логическими операторами **and**, **or**, **not**

При записи необходимо соблюдать синтаксические правила:

после условия ставится двоеточие;

вложенность операторов выполняется путем вставки отступов. Отступ - это **4 пробела** или **Tab**.

Нежелательно в одной программе сочетать эти варианты простановки отступов.

Примеры:

```
a = 33b = 200if b >= a:
    print(f'b больше a')else:
    print(f'b не больше a')
```

```
b больше a
```

Неполная форма ветвления

```
if условие:
    блок_операторов1
```

```
a = 33b = 20if b >= a:
    print(f'b больше a')print('Ветвление закончено')
```

```
Ветвление закончено
```

В данном случае условие приняло значение **False**, но ветка **else** отсутствует, поэтому управление сразу передано инструкции, следующей за ветвлением.

if...elif...else

Данная конструкция используется для проверки нескольких условий и выполнения только подходящего блока операторов:

if — проверяет первое условие. elif (сокращение от else if) — проверяет следующие условия, если предыдущее условие ложно. else — выполняет блок кода, если все предыдущие условия ложны.

Синтаксис:

```
if условие1:

        блок_кода1elif условие2:

        блок_кода2elif условие3:

        блок_кода3else:

        блок_кодаN
```

```
a = 9

if a == 10:

        print('a равно 10')elif a < 10:

        print('a меньше 10')else:

        print('a больше 10')
```

```
a меньше 10
```

```
# Запрос фамилииlast_name = input('Введите фамилию')

# Взять окончание - два символа справаend_fam = last_name[-2:]

if end_fam == 'ов':

        print('Вы мужского пола')elif end_fam == 'ий':

        print('Вы мужского пола')elif end_fam == 'ва':

        print('Вы женского пола')elif end_fam == 'ая':

        print('Вы женского пола')else:

        print('Программа находится в режиме отладки. Еще не все фамилии проанализированы')
```

```
Введите фамилию: СидоровВы мужского пола

Введите фамилию: ВасильеваВы женского пола

Введите фамилию: ВасильевПрограмма находится в режиме отладки. Еще не все фамилии проанализированы
```

конструкция if построена на условиях: после if и elif всегда пишется условие. Блоки if/elif выполняются только когда условие

возвращает True, поэтому первое с чем надо разобраться - это что является истинным, а что ложным в Python.

True и False

В Python, кроме очевидных значений True и False, всем остальным объектам также соответствует ложное или истинное значение:

истинное значение:

- любое ненулевое число
- любая непустая строка
- любой непустой объект

ложное значение:

- 0
- None
- пустая строка
- пустой объект

Примеры:

```
my_list = [1,2]if my_list:
        print(True)else:
        print(False)
```

```
True
```

```
my_list = [[]]if my_list:
        print(True)else:
        print(False)
```

```
True
```

Здесь **my_list = [[]]** - создан список, содержащий пустой список. То есть my_list сам пустым не является.

```
my_list = []if my_list:
        print(True)else:
        print(False)
```

```
False
```

Логические операторы

Используются для работы с булевыми значениями **True** и **False**

and

Результат операции принимает значение одного из операндов, если **оба** операнда являются **True**, то будет возвращен последний операнд.

Если есть один False, то будет возвращен первый False

or

Результат операции принимает значение одного из операндов, если **оба** операнда являются **True**, то будет возвращен первый операнд.

False возвращается, если все операнды **False**

Указанные особенности полезны при составлении логики проверки. Для повышения эффективности программ.

not

Принимает значение True, если применяется к операнду, имеющему значение False и наоборот.

Оператор in

Оператор in выполняет проверку на наличие элемента в последовательности (например, элемента в списке или подстроки в строке):

```
my_val = 10list1 = [10, 20, 30, 77]if my_val in list1:
        print(True)else:print(False)
True
my_str = 'full'string1 = 'List is empty'print(my_str in string1)
False
```