Ветвление

Условный оператор

Условный оператор

Программы должны уметь выполнять разные действия в зависимости от введенных данных

Для принятия решения программа проверяет истинно или ложно определенное условие

Проверка условий и принятие решений по результатам этой проверки называется **ветвлением**. Программа таким способом выбирает, по какой из возможных ветвей ей двигаться дальше.

Условный оператор

Рассмотрим следующую программу:

```
print('Какой язык программирования мы изучаем?')
answer = input()
if answer == 'Python':
    print('Верно! Мы изучаем Python =)')
    print('Python - отличный язык!')
```

Программа считывает текст и проверяет, если введенный текст равен строке 'Python', то выводит пользователю текст:

```
Верно! Мы изучаем Python =)
Python - отличный язык!
```

Условный оператор


```
print('Какой язык программирования мы изучаем?')
answer = input()
if answer == 'Python':
    print('Верно! Мы изучаем Python =)')
    print('Python - отличный язык!')
```

Двоеточие (:) в конце строки сообщает, что дальше находится **блок команд**

В **блок** входят все строки, расположенные с **отступом**

Если условие истинно, выполняется весь **блок команд**

Условный оператор


```
print('Какой язык программирования мы изучаем?')
answer = input()
if answer == 'Python':
    print('Верно! Мы изучаем Python =)')
    print('Python - отличный язык!')
```

отступ

блок команд

Условный оператор


```
print('Какой язык программирования мы изучаем?')
answer = input()
if answer == 'Python':
    print('Верно! Мы изучаем Python =)')
    print('Python - отличный язык!')
else:
    print('Не совсем так!')
```

В этой программе мы обрабатываем сразу два случая:

- если условие истинно (ввели 'Python'),
- и если условие ложно (ввели что угодно, кроме 'Python')

Отступы

В Python **отступ** – это неотъемлемая часть кода

Отступ означает небольшое смещение строки кода вправо

Отступ сообщает, где начинается и где заканчивается **блок кода**

Для отступа блоков кода используются табуляция (клавиша tab) или 4 пробела

Операторы сравнения

Для проверки условия мы используем специальный символ:
двойное равенство (==):

```
print('Какой язык программирования мы изучаем?')
answer = input()
if answer == 'Python':
    print('Верно! Мы изучаем Python =)')
    print('Python - отличный язык!')
```

== это условный оператор сравнения (проверка на равенство)

= VS ==

Не путать:
оператор **присваивания** (=) VS **условный оператор** (==)

```
num = 1992
s = 'I love Python'
```

оператор присваивания (=) придает переменным значения

```
if answer == 'Python':
if name == 'Gvido':
if temperature == 40:
```

условный оператор (==) проверяет на равенство два значения

Путаница с операторами **==** и **=** является одной из самых распространенных ошибок в программировании

Операторы сравнения

В Python существует **6** основных операторов сравнения:

Оператор	Описание	Пример использования
>	больше	if x > 7:
<	меньше	if x < 7:
>=	больше либо равно	if x >= 7:
<=	меньше либо равно	if x <= 7:
==	равно	if x == 7:
!=	не равно	if x != 7:

Операторы сравнения

Программа может содержать несколько условных операторов:

```
num1 = int(input())
num2 = int(input())

if num1 < num2:
    print(num1, 'меньше чем', num2)

if num1 > num2:
    print(num1, 'больше чем', num2)

if num1 == num2:
    print(num1, 'равно', num2)

if num1 != num2:
    print(num1, 'не равно', num2)
```

Что будет выведено, если:
num1 = 3, num2 = 7?

Цепочки сравнений

Операторы сравнения в Python можно **объединять в цепочки:**

```
age = int(input())
if 3 <= age <= 6:
    print('Вы ребёнок')
```

код проверяет: находится ли значение переменной **age**, в диапазоне от 3 до 6

```
if a == b == c:
    print('числа равны')
else:
    print('числа не равны')
```

код проверяет: равны ли все три переменные **a, b, c** друг другу

Цепочки сравнения – это **особенность** Python

Логические операции

Логические операции

Сложное условие состоит из нескольких условий

В Python есть три логических операции, которые позволяют создавать сложные условия:

1. **and:** логическое умножение, «и»
2. **or:** логическое сложение, «или»
3. **not:** логическое отрицание, «не»

Операция and

Предположим мы хотим написать программу для учеников от двенадцати лет, которые учатся по крайней мере в 7 классе:

```
age = int(input())
grade = int(input())
if age >= 12 and grade >= 7:
    print('Доступ разрешен.')
else:
    print('Доступ запрещен.')
```

Блок выполняется только при выполнении обоих условий **одновременно**!

Мы объединили два условия при помощи операции **and**

Операция and

Операция **and** может объединять произвольное количество условий:

```
age = int(input())
grade = int(input())
city = input()
if age >= 12 and grade >= 7 and city == 'Тюмень':
    print('Доступ разрешен.')
else:
    print('Доступ запрещен.')
```

Мы объединили три условия при помощи операции **and**

Таблица истинности для операции and

a	b	a and b
False	False	False
False	True	False
True	False	False
True	True	**True**

Операция **and** возвращает **истину** когда оба выражения истинны

Операция or

Операция **or** также применяется для объединения условий:

```
city = input()
if city == 'Москва' or city == 'Екатеринбург':
    print('Доступ разрешен.')
else:
    print('Доступ запрещен.')
```

Блок выполняется в случае если **хотя бы одно** из условий выполняется!

Мы объединили два условия при помощи операции **or**

Операция or

Операция **or** может объединять произвольное количество условий:

```
city = input()
if city == 'Москва' or city == 'Уфа' or city == 'Тюмень':
    print('Доступ разрешен.')
else:
    print('Доступ запрещен.')
```

Мы объединили три условия при помощи операции **or**

Таблица истинности для операции or

a	b	a or b
False	False	**False**
False	True	True
True	False	True
True	True	True

Операция `or` возвращает **ложь** когда оба выражения ложны

Операция and и or вместе

Мы можем использовать обе логические операции одновременно:

```
age = int(input())
grade = int(input())
city = input()
if age >= 12 and grade >= 7 and (city == 'Москва' or
                                 city == 'Уфа' or city == 'Тюмень'):
    print('Доступ разрешен.')
else:
    print('Доступ запрещен.')
```

Операция not

Операция **not** позволяет инвертировать результат логического выражения:

```
age = int(input())
if not (age < 12):
    print('Доступ разрешен.')
else:
    print('Доступ запрещен.')
```

```
age = int(input())
if age >= 12:
    print('Доступ разрешен.')
else:
    print('Доступ запрещен.')
```

Мы поместили скобки вокруг выражения **age < 12** для того, чтобы было чётко видно, что операция **not** применяется к значению выражения **age < 12**, а не только к переменной **age**

Таблица истинности для операции not

a	not a
False	True
True	False

Приоритет логических операций

Приоритет логических операций:

Операция	Приоритет
not	0
and	1
or	2

- сначала отрицание **not**
- далее логическое умножение **and**
- далее логическое сложение **or**

Для явного указания порядка выполнения логических операций мы используем **скобки**

Примечания

Частой ошибкой является путаница логических **and** и **or**:

```
if x > 1 and x < 100:
if x > 1 or x < 100:
```

Другой частой ошибкой является код:

```
if age >= 7 and <= 9:
```

```
if age >= 7 and age <= 9:
```

В Python есть способ для проверки принадлежности диапазону:

```
if 7 <= age <= 9:
```

Вложенный условный оператор

Условный оператор

- Если условие истинно, то выполняется блок кода 1
- Если условие ложно, то выполняется блок кода 2

```
if условие:
    блок кода1
else:
    блок кода2
```

```
if условие:
    блок кода1
```

else является необязательным

Для отступа блоков кода используются табуляция (клавиша tab) или 4 пробела

Вложенный условный оператор

Внутри условного оператора можно использовать любые инструкции языка Python, в том числе и условный оператор

Получаем вложенное ветвление – после одной развилки появляется другая развилка:

```
if условие1:
    блок кода
else:
    if условие2:
        блок кода
    else:
        if условие3:
            блок кода
        ...
```

Вложенные блоки имеют больший размер отступа (8, 12, … пробелов)

Вложенный условный оператор

Рассмотрим программу которая переводит стобальную оценку в пятибальную:

```
grade = int(input())
if grade >= 90:
    print(5)
else:
    if grade >= 80:
        print(4)
    else:
        if grade >= 70:
            print(3):
        else:
            if grade >= 60:
                print(2):
            else:
                print(1)
```

Уровень вложенности настолько глубок, что код становится трудно понять ☹

Каскадный условный оператор

Если требуется проверить несколько условий, в языке Python есть **каскадный условный оператор:**

```
if условие1:
    блок кода1
elif условие2:
    блок кода2
elif условие3:
    блок кода3
...
else:
    блок кода
```

1. Сначала проверяется условие1:
   - Если оно истинно, то выполняется блок кода1
   - Если оно ложно, то переходим к условию2
1. Проверяется условие2:
   - Если оно истинно, то выполняется блок кода2
   - Если оно ложно, то переходим к условию3

   ……..
1. Процесс продолжается до тех пор, пока не будет найдено истинное условие, либо пока не закончатся выражения elif
2. Если ни одно условие не является истинным, то выполняется блок кода после else

Каскадный VS вложенный оператор


```
grade = int(input())

if grade >= 90:
    print(5)
elif grade >= 80:
    print(4)
elif grade >= 70:
    print(3):
elif grade >= 60:
    print(2):
else:
    print(1)
```

if, elif, else выровнены и все исполняемые по условию блоки выделены отступом (4 пробела)

```
grade = int(input())
if grade >= 90:
    print(5)
else:
    if grade >= 80:
        print(4)
    else:
        if grade >= 70:
            print(3):
        else:
            if grade >= 60:
                print(2):
            else:
                print(1)
```

Примечания

Каскадный оператор `if-elif-else` может быть запрограммирован вложенными операторами `if-else`

Каскадный оператор `if-elif-else` обычно легче, чем длинная серия вложенных операторов `if-else`

Заключительный блок `else` в каскадном операторе `if-elif-else` является необязательным!