Тема 3. Организация ветвления (Управляющие конструкции)

3.1. Тип данных bool

До этого мы писали программы, которые выполнялись строго по порядку записи. Но таким образом невозможно написать сложную программу. Поэтому неизбежно приходится изучать более сложные конструкции языка программирования.

Суть алгоритма ветвления заключается в том, что в программе присутствует одно или несколько условий, которые будут проверяться программой, и в зависимости от того, выполнено условие (**True** – истина) или нет (**False** – ложь), будет выполняться тот или иной код программы.

Для изучения данной темы рассмотрим подробно тип **bool**.

Переменная типа **bool** принимает 2 значения: **True** или **False**.

Значения **True** возвращают следующие объекты:

- Любое число, не равное нулю

  bool(1)
  bool(2.5)
  bool(-1)

- Любой не пустой объект

  bool('1')

Значения **False** возвращают следующие объекты:

- Любое число равное нулю

  bool(0)
  bool(0.0)

- Любой пустой объект

  bool('')

- Значение N**one**

  bool(None)

3.2. Операторы сравнения

Условия в программе строятся при помощи операторов сравнения. Для удобства, приведем все операторы в виде таблицы. Предположим, что у нас есть 2 переменные с заданными значениями: **a = 2**, **b = 5**.

Обратите внимание, некоторые операторы получаются комбинацией двух знаков. Между знаками пробел ставить нельзя!

Оператор	Описание	Пример
==	Равно. Проверяет, совпадают ли значения операндов, если совпадают, то возвращается значение True иначе False.	(a==b), в результате получим False.
!=	Не равно. Проверяет, совпадают ли значения операндов, если не совпадают, то возвращается значение True иначе False.	(a!=b), в результате получим True.
>	Больше. Проверяет, больше левый операнд правого или нет, если больше, то возвращается значение True, иначе False.	(a>b), в результате получим False.
<	Меньше. Проверяет, меньше левый операнд правого или нет, если меньше, то возвращается значение True, иначе False.	(a<b), в результате получим значение True.
>=	Больше или равно. Проверяет, значение левого операнда, если он больше или равен правому, то возвращается значение True, иначе False.	(a>=b), в результате получим False.
<=	Меньше или равно. Проверяет, значение левого операнда, если он меньше или равен правому, то возвращается значение True, иначе False.	(a<=b), в результате получим значение True.
is	Проверяет, ссылаются переменные на один и тот же объект или нет.	a=b=2 print(a is b), в результате получим True. А если сделать так: a=b=2 a=3 print(a is b), в результате получим False.

3.3. Логические операторы

Логические операторы используются для создания более сложных условий. Python в отличие от других языков программирования позволяет явно создавать сложные условия, типа **a** < **b** < **c** и т. д.

Пример:

```
a = 2
b = 5
c = 4
print(a < c < b)
```

В результате получим **True**. Также можно создавать сложные условия при помощи дополнительных логических операторов.

Для удобства выведем их в виде таблицы. Предположим, у нас есть две переменные: **a = 5** и **b = -4**.

Оператор	Описание	Пример
and	Логическое **И**. Требует одновременного выполнения всех условий для получения результата равного True.	(a>0) and (b>0), в результате получим False, так как условие, b>0, не выполнено.
or	Логическое **ИЛИ**. Не требует выполнения одновременно всех условий. Для получения результата равного True, достаточно что бы хотя бы одно условие приняло значение True.	(a>0) or (b>0), в результате получим True, так как условие, a>0, приняло значение True.
not	Логическое **НЕ**. Меняет результат условия на противоположный, если было True, то станет False, и наоборот.	not(b>0), в результате получим True, так как условие, b>0, приняло значение False, а мы взяли от него отрицание.

Таблицы истинности логических операторов

AND

True **and** True = True

True **and** False = False

False **and** True = False

False **and** False = False

При использовании оператора **and** (логическое **и**) истину мы получаем только тогда, когда все входящие условия истинны. Во всех остальных случаях получаем ложь.

Пример:

(5<6) **and** (7>6) имеет значение True

(5>6) **and** (7>6) имеет значение False

OR

True **or** True = True

True **or** False = True

False **or** True = True

False **or** False = False

При использовании оператора **or** (логическое **или**) ложь мы получаем только тогда, когда все входящие условия ложны. Во всех остальных случаях получаем истину.

Пример:

(5<6) **or** (7>6) имеет значение True

(5>6) **or** (7>6) имеет значение True

NOT

not True = False

not False = True

3.4. Проверка выполнимости условия

При работе с линейным алгоритмом выполнимость условия можно проверить двумя способами.

Первый способ, это указание условия в функции **print()** в качестве аргумента. Условие проверится, и результат выведется на экран.

```
a = 5
b = 6
print(a < b)
```

В результате получим **True**.

Второй способ подразумевает присваивание логического выражения переменной. В результате создастся переменная логического типа, и ей уже будет присвоен результат логического выражения.

```
a = 5
b = 6
c = a < b
print(c)
```

В результате получим **True**.

Но для построения полноценных программ этого недостаточно. В программе важна не только проверка результата, но и описание дальнейшего поведения программы в зависимости от выполнимости условия. Для этих целей служит алгоритм ветвления.

3.5. Оператор if

Оператор **if** проверяет логическое условие, в котором происходит сравнение данных, и в зависимости от полученного результата выбираются дальнейшие действия.

Перед тем как будем разбирать оператор ветвления, вспомним следующее. В Python, в отличие от других языков программирования, нет операторных скобок, которые в других языках обозначаются по-разному.

В Pascal, к примеру, они выглядят так: **begin** действия **end**.

В C-подобных языках: **{** действия **}**.

В Python объединение действий в блоки осуществляется при помощи отступов. Все действия, написанные с одинаковым отступом, будут относиться к одному блоку.

Рекомендованный отступ равен четырем пробелам. Также отступ можно ставить табуляцией, в редакторах для Python она настроена на 4 пробела.

Синтаксис оператора if

if *<условие>:*
    <действия>

Обратите внимание на то, что все действия, которые должны выполняться при выполнении условия, должны быть напечатаны с одинаковым отступом.

Пример 1:

```
a = 10
if a > 0:
    print('Да')
    print(a)
print('Пока')
```

Результат:

```
Да
10
Пока
```

Пример 2:

```
a = -10
if a > 0:
    print('Да')
    print(a)
print('Пока')
```

```
Пока
```

Оператор else

Оператор **else** используется вместе с оператором **if**. Оператор **else** содержит блок кода, который должен выполниться, если условие вернуло значение **False**.

Синтаксис оператора else

if *<условие>:*
    <действия1>
else*:*
    <действия2>

Пример:

```
a = 2
b = 4
if a > b:
    print('Да')
else:
    print('Нет')
print('Пока')
```

Результат:

```
Нет
Пока
```

Оператор elif

Оператор **elif** используется в случае, когда необходимо проверить несколько взаимоисключающих условий. Данный оператор используется совместно с операторами **if** и **else**.

Синтаксис оператора elif

if *<условие1>:*
    <блок действий 1>
elif *<условие2>:*
    <блок действий 2>
...
elif *<условие n>:*
    <блок действий n>
else*:*
    <блок действий>

Пример:

С клавиатуры вводятся два целых числа, нужно найти большее из них, а если они равны, то выдать соответствующее сообщение.

```
a = int(input())
b = int(input())
if a > b:
    print('число a больше')
elif b > a:
    print('число b больше')
else:
    print('числа равны')
```

В **Python** нет таких операторов (операторов множественного выбора) как **switch** (C++, C#) или **case** (Pascal), но можно использовать операторы **if…elif...** вместо них.

3.6. Вложенные конструкции

В некоторых задачах может возникнуть ситуация, когда после проверки условия, если оно вернуло значение **True**, нужно проверить еще условие. В данном случае нужно использовать вложенные конструкции **if**.

Синтаксис вложенной конструкции if

Здесь нужно особое внимание уделить отступам, чтобы верно распределить блоки кода по условиям.

if *<условие1>:*
    <блок действий 1>
    if *<условие 2>:*
        <блок действий 2>
    elif *<условие 3>:*
        <блок действий 3>
    else*:*
        <блок действий 4>
elif *<условие 5>:*
    <блок действий 5>
else*:*
    <блок действий 6>

В данном случае мы рассмотрели полную конструкцию вложенных операторов, но она не обязательно будет содержать все операторы. Может использоваться и более простая конструкция.

if *<условие1>:*
    <блок действий 1>
    if *<условие 2>:*
        <блок действий 2>

Разумеется, все случаи расписать не представляется возможным. Главное понять принцип.

Пример 1:

С клавиатуры вводится целое число. Если оно положительное, то проверить его на четность, иначе вывести сообщение, что число не положительное.

```
a = int(input())
if a > 0:
    if a % 2 == 0:
        print('Число четное')
    else:
        print('Число нечетное')
else:
    print('Число не положительное')
```

Пример 2:

С клавиатуры вводится число. Проверить, попадает ли оно в отрезок [-100; 100].

```
a = float(input('Введите любое число: '))
if (a >= -100) and (a <= 100):
    print('Число попадает в отрезок')
else:
    print('Число не попадает в отрезок')
```

3.7. Трехместное выражение if/else

При проверке условий не всегда приходится выполнять сложные наборы действий, зачастую, в зависимости от выполнимости условия, просто присваивается то или иное значение результирующей переменной.

В общем виде эта запись выглядит так:

допустим, нам нужно в зависимости от выполнения условия, для переменной **a** присвоить значение **b** или **c**.

if *<условие>:*
  a = b
else*:*
  a = c

Именно такую запись можно заменить трехместным выражением **if/else** следующим образом:

a = b ***if*** *<условие>* ***else*** *c*

В данной инструкции, если условие будет истинно, то переменной **a** присвоится значение **b**, иначе переменной **a** присвоится значение **c**. При этом **b** и **c** могут быть как просто значениями, так и выражениями.

Пример:

Вычислить значение функции следующего вида:

$$y = \begin{cases} \sqrt{x}, & \text{если } x > 0 \\ x^2, & \text{в противном случае} \end{cases}$$

x – задаем после запуска программы.

Решение:

```
import math

x = int(input('x = '))
y = math.sqrt(x) if x > 0 else pow(x, 2)
print('y =', y)
```