ТОМСКИЙ ПОЛИТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ

tpu.ru

Углубленный курс информатики

Лекция 2

Операторы управления потоком команд в Python

14 февраля 2022 г.

Долганов Игорь Михайлович,
к.т.н., доцент ОХИ ИШПР

ТОМСКИЙ ПОЛИТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ

Содержание

ТОМСКИЙ
ПОЛИТЕХНИЧЕСКИЙ
УНИВЕРСИТЕТ

Логический тип данных

Логический тип данных

- Для логического типа данных (`bool`) можно объявлять логические переменные, инициализируя их логическими значениями или присваивая им результат вычисления логических выражений.
- Логических констант в Python две: `True` (истина) и `False` (ложь).

```
1 x = True
2 y = False
3 z = 2 > -1
4 print(x, y, z)   #True False True
5
```

ТОМСКИЙ
ПОЛИТЕХНИЧЕСКИЙ
УНИВЕРСИТЕТ

Операции сравнения

Операции сравнения

Логические выражения являются аналогом математической формулы, результатом его вычисления будет одна из двух логических констант – `True` или `False`.

Операции сравнения

Операция	**Описание**
`x < y`	Меньше
`x <= y`	Меньше или равно
`x > y`	Больше
`x >= y`	Больше или равно
`x == y`	Равно
`x != y`	Не равно

- Приоритет операций сравнения ниже, чем у арифметических операций:

```
1 print(2 + 3 * 5 > 7 / 2 + 3.5)   #True
2
```

Операции сравнения

Операции сравнения в Python сравнивают относительные величины своих операндов и возвращают результат логического типа, значение которого обычно используется для выбора следующего действия в более крупном операторе или программе:

```
>>> 1 < 2          #Меньше
True
>>> 2.0 >= 1       #Больше или равно: число целого типа 1
                   #преобразуется к 1.0
True
>>> 3.0 == 3.0     #Значения равны
True
>>> 3.0 != 3.0     #Значения не равны
False
```

ТОМСКИЙ
ПОЛИТЕХНИЧЕСКИЙ
УНИВЕРСИТЕТ

Логические операции

Логическая операция `not`

- Унарная логическая операция `not` называется логическим отрицанием («НЕ», инверсия) и указывается перед логическим выражением для получения его противоположного значения.
- Приоритет операции `not` ниже, чем у операций сравнения, поэтому следующее за ней логическое выражение можно не заключать в скобки:

```
1 print(3 + 5 >= 8)   #True
2 print(not 3 + 5 >= 8)   #False
3
```

Логическая операция and

- Бинарная логическая операция `and` называется логическим умножением (логическое «И»).
- Результатом операции `and` будет `True` только тогда, когда оба логических выражения будут иметь значения `True`, а в прочих случаях результатом будет `False`.

```
1 print(9 + 3 < 10 and 2 + 2 == 4)   #False
2 print(4 + 2 != 0 and 10 * 2 == 20)   #True
3
```

Логическая операция `or`

- Бинарная логическая операция `or` называется логическим сложением (логическое «ИЛИ»).
- Результатом операции `or` будет `False` только тогда, когда оба логических выражения будут иметь значения `False`, а в прочих случаях результатом будет `True`.

```
1 print(9 + 3 < 10 or 2 + 2 == 4)   #True
2 print(4 + 2 < 0 or 10 * 2 >= 100)   #False
3
```

Приведение к логическому типу

- Конструктор типа `bool(x)` может использоваться для приведения любого значения к логическому типу (если это значение можно интерпретировать как логический тип). Если аргумент *x* ложь или опущен, то будет возвращено значение `False`.

```
1 print(bool(1), bool(-1.0))   #True True
2 print(bool(0), bool(0.0))   #False False
3 print(bool())   #False
4
```

ТОМСКИЙ ПОЛИТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ

Сцепленные операции сравнения

Сцепленные операции сравнения

В Python также есть возможность выстраивать цепочки из нескольких операторов сравнения для выполнения проверок вхождения в диапазон. Сцепленные сравнения являются краткой записью для более массивных булевых выражений.

```
>>> x < y > z
False
>>> x < y and y > z
False
>>> 3 < 6 < 9.0 < 12
True
>>> 3 > 6 > 9 > 12
False
```

ТОМСКИЙ
ПОЛИТЕХНИЧЕСКИЙ
УНИВЕРСИТЕТ

Сравнение чисел с плавающей точкой

Сравнение чисел с плавающей точкой

Числа с плавающей точкой в логических операциях сравнения могут вести себя неожиданным образом, требуя определенных преобразований для содержательного сравнения:

```
>>> 0.1 + 1.1 == 1.2  #Кажется, что должно быть True
False
>>> 0.1 + 1.1  #Близко к 1.2, но не точно: ограниченная точность
1.2000000000000002
>>> round(0.1 + 1.1, 1) == round(1.2, 1)  #Работает нормально, если
    #выполнить преобразование: функция round выполняет округление,
    #в данном случае до первого знака
True
```

ТОМСКИЙ
ПОЛИТЕХНИЧЕСКИЙ
УНИВЕРСИТЕТ

Оператор `if`

Оператор `if`

- Оператор `if` используется для проверки условий: **если** условие верно, то выполняется блок выражений (называемый «if-блок»), **иначе** выполняется другой блок выражений (называемый «else-блок»).
- Блок «else» является необязательным.

```
>>> if 1:
...     print("true")
...
true
```

Напоминаем, что 1 – это булевское значение «истина» (его эквивалентом является слово `True`), таким образом, проверка в операторе всегда проходит. Для обработки ложного результата понадобится добавить часть `else`:

```
>>> if not 1:
...     print("true")
... else:
...     print("false")
...
false
```

Множественное ветвление

Рассмотрим пример оператора `if`, содержащего все необязательные части:

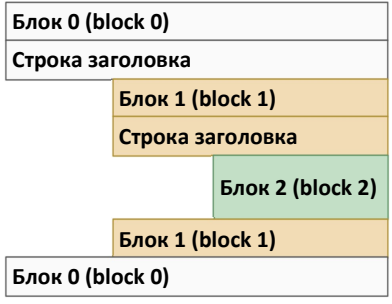
```
number = 23
guess = int(input("Введите целое число: "))

if guess == number:
    print("Поздравляю, Вы угадали,") #Здесь начинается новый блок
    print("хотя и не выиграли никакого приза!") #Конец нового блока
elif guess < number:
    print("Нет, загаданное число немного больше этого.") #Ещё один блок
    #Внутри блока Вы можете выполнять любые операторы
else:
    print("Нет, загаданное число немного меньше этого.")
```

- Интерпретатор выполняет операторы, вложенные внутрь первой проверки, которая вернет `True`, или часть `else`, если все проверки вернули `False`.
- Части `elif` и `else` могут быть опущены, а в каждой части допускается указывать более одного вложенного оператора.
- Части `if`, `elif` и `else` связываются друг с другом выравниванием по вертикали с одинаковыми отступами.

Ограничения блоков: правила отступов

- Python определяет границы блоков автоматически по *отступам* строк.
- Все операторы с отступами на одинаковое расстояние вправо принадлежат тому же самому блоку кода.
- Блок заканчивается тогда, когда встречается конец файла или строка с меньшим отступом, и более глубоко вложенный блок просто смещается дальше вправо, чем операторы во включающем блоке.

```
1  x = 10
2  if x:
3      y = 20
4      if y:
5          print("block 2")
6      print("block 1")
7  print("block 0")
8
```



ТОМСКИЙ
ПОЛИТЕХНИЧЕСКИЙ
УНИВЕРСИТЕТ

Тернарное выражение `if/else`

Тернарное выражение `if`/`else`

- Зачастую элементы, использованные в операторе `if`, достаточно просты, так что распространение такого оператора на четыре строки выглядит излишеством.
- В других случаях конструкцию подобного рода может понадобиться вложить в более крупный оператор, а не присваивать ее результат какой-то переменной.
- По указанным причинам в Python был добавлен новый формат условного выражения, который позволяет определить то же самое в одном действии (тернарный оператор).

```
1 if x:
2     a = y
3 else:
4     x = z
5
```

```
1 a = y if x else z
2
```

Тернарное выражение if/else

```
1  x = "f" if 10 else "t"   #Числа, не равные нулю, истина
2  print(x)  #f
3
4  x = "f" if 0 else "t"   #Ноль - это ложь
5  print(x)  #t
6
```

Использовать тернанрый оператор нужно крайне умеренно и только в тех случаях, когда все его составные части относительно просты, иначе предпочтительнее использовать форму полного оператора if для облегчения его будущего модифицирования.

ТОМСКИЙ
ПОЛИТЕХНИЧЕСКИЙ
УНИВЕРСИТЕТ

Операторы цикла в Python

Операторы цикла

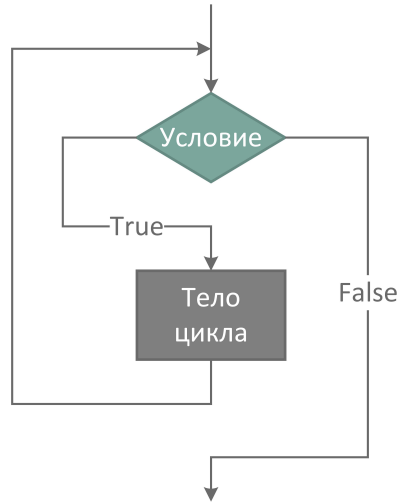
- Алгоритмы решения многих задач требуют некоторого количества повторений своих отдельных частей.
- Такие повторяющиеся участки называют циклическими, а операторы языка Python, реализующие соответствующие повторения – **операторами цикла**.
- Цикл состоит из **заголовка цикла** и **тела цикла**.
- Заголовок определяет условие прекращения (или выполнения) цикла, а тело цикла содержит операторы, которые нужно повторять.

Операторы цикла в Python:

1. Цикл `while`
2. Цикл `for`

Оператор цикла `while`

- Оператор `while` многократно повторяет блок операторов до тех пор, пока проверка в заголовочной части оценивается как истина.
- Управление продолжает возвращаться к началу оператора, пока проверка не даст ложное значение. Когда результат проверки становится ложным, управление переходит на оператор, следующий после блока `while`.
- Если проверка оценивается в ложное значение с самого начала, тогда тело цикла никогда не выполнится и оператор `while` пропускается.



Оператор цикла `while`

Общий формат цикла `while`:

```
1 while expression:   #Проверка цикла
2     operators       #Тело цикла
3
```

Цикл `while` можно использовать:

- в математических итерационных алгоритмах для проведения вычислений с заданной точностью;
- при вводе данных, когда их количество заранее неизвестно, а условие завершения ввода определено некоторым введенным значением;
- при поиске нужного элемента в какой-либо структуре данных.

Примеры использования цикла `while`

Вывод чисел в диапазоне $[0, 10)$ с шагом 1:

```
1 a, b = 0, 10
2 while a < b:
3     print(a, end="")
4     a += 1
5
```

```
0123456789
```

Примеры использования цикла while


```
number = 24
running = True

while running:
    guess = int(input("Введите целое число: "))
    if guess == number:
        print("Поздравляем, Вы угадали!")
        running = False  #это останавливает цикл while
    elif guess < number:
        print("Нет, загаданное число немного больше этого")
    else:
        print("Нет, загаданное число немного меньше этого")

#другие операторы программы
print("Завершение.")
```

```
Введите целое число: 12
Нет, загаданное число немного больше этого
Введите целое число: 24
Поздравляем, Вы угадали!
Завершение.
```

Оператор цикла `for`

- Оператор `for...in` также является оператором цикла, который осуществляет итерацию по *последовательности* объектов, т.е. проходит через каждый элемент в последовательности.
- *Последовательность* – это упорядоченный или неупорядоченный набор элементов.
- Во многих случаях в заголовке цикла `for` используется функция `range`, которая является генератором арифметических прогрессий:

```
1 for i in range(10):   #10 не включительно
2     print(i, end=" ")
3
```

```
0 1 2 3 4 5 6 7 8 9
```

Оператор цикла for

Функция range – генератор арифметических прогрессий:

```
for i in range(1, 11): #можно задать начальное значение
    print(i, end=" ")

```

```
1 2 3 4 5 6 7 8 9 10
```

```
for i in range(0, 11, 2):  #также можно менять шаг
    print(i, end=" ")

```

```
0 2 4 6 8 10
```

Вложенные циклы for

Операторы цикла for могут быть вложены друг в друга на произвольную глубину:

```
for i in range(3):
    for j in range(3):
        if i != j:
            print(i, j, round(1 / (i + j), 2))

```

```
0 1 1.0
0 2 0.5
1 0 1.0
1 2 0.33
2 0 0.5
2 1 0.33
```

ТОМСКИЙ ПОЛИТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ

Операторы `break`, `continue`, `pass` и конструкция `else` цикла

Оператор `break`

Оператор `break` выполняет немедленный выход из цикла, т.е. остановку выполнения команд, даже если условие выполнения цикла еще не приняло значение `False` или последовательность элементов не закончилась.

```
while True:
    name = input("Enter name: ")
    if name == "stop":
        break
    age = input("Enter age: ")
    print("Hello", name, "=>", int(age) * 2)

```

```
Enter name: John
Enter age: 35
Hello John => 70
Enter name: Julya
Enter age: 24
Hello Julya => 48
Enter name: stop
```

Оператор `continue`

Оператор `continue` используется для немедленного перехода в начало цикла.

```
x = 10

while x:
    x -= 1
    #Нечетное? Тогда пропустить
    if x % 2 != 0:
        continue
        print(x, end=" ")

```

```
8 6 4 2 0
```

```
x = 10

while x:
    x -= 1
    #Четное? Тогда выводим
    if x % 2 == 0:
        print(x, end=" ")

```

```
8 6 4 2 0
```

Оператор `pass`

- Оператор `pass` – это заполнитель, обозначающий отсутствие действий, используемый в ситуациях, когда синтаксис требует оператора, но нет возможности выполнить что-либо полезное.
- Данный оператор часто применяется для кодирования пустого тела для составного оператора.

К примеру, с помощью `pass` можно написать бесконечный цикл, который на каждом проходе ничего не делает:

```
1 while True:
2     pass  #Для прекращения работы нажмите <Ctrl+C>!
3
```

Конструкция else цикла

Полная форма записи циклов while и for выглядит следующим образом:

```
while True:
    operators

    #Выход с пропуском else
    if exitTest():
        break

    #Переход к заголовку цикла
    if skipTest():
        continue

#Выполняется, если не было break
else:
    operators
```

```
for x in collection:
    operators

    #Выход с пропуском else
    if exitTest():
        break

    #Переход к заголовку цикла
    if skipTest():
        continue

#Выполняется, если не было break
else:
    operators
```

Если циклы while или for прервать оператором break, соответствующие им блоки else выполняться не будут.

Конструкция `else` цикла

В приведенном примере выполняется проверка, является ли положительное целое число *y* простым, за счет поиска сомножителей больше 1:

```
x = y // 2  #Для y > 1
while x > 1:
    if y % x == 0:  #Остаток от деления
        print(y, "has factor", x)   #Имеет сомножитель
        break  #Пропуск else
    x -= 1

else:  #Нормальный выход
    print(y, "is prime")  #Является простым
```

ТОМСКИЙ
ПОЛИТЕХНИЧЕСКИЙ
УНИВЕРСИТЕТ

Примеры

Пример 1 – Fizz Buzz

Дано: Положительное целое число (`int`)

«Fizz buzz» – это игра со словами. Необходимо написать программу, которая принимает положительное целое число и возвращает следующие значения:

1. «Fizz Buzz», если число делится на 3 и 5;
2. «Fizz», если число делится на 3;
3. «Buzz», если число делится на 5;
4. Исходное число в остальных случаях.

Пример:

$x = 15$	результат:	«Fizz Buzz»
$x = 6$	результат:	«Fizz»
$x = 5$	результат:	«Buzz»
$x = 7$	результат:	7

Пример 1 – Fizz Buzz


```
1  x = int(input("Введите положительное целое число: "))
2
3  if not x % 3 and not x % 5:
4      print("Fizz Buzz")
5
6  elif not x % 3:
7      print("Fizz")
8
9  elif not x % 5:
10     print("Buzz")
11
12 else:
13     print(x)
14
```

Пример 2 – Вычисление факториала числа n

Необходимо написать программу для вычисления $n!$:

$$n! = 1 \cdot 2 \cdot 3 \cdot \ldots \cdot (n-1) \cdot n$$

где n – положительное целое число.

Пример 2 – Вычисление факториала числа n

С использованием цикла `while`:

```
1  n = int(input("Введите положительное целое число: "))
2
3  f = 1
4
5  while n:
6      f *= n
7      n -= 1
8
9  print("n! =", f)
10
```

Пример 2 – Вычисление факториала числа n

С использованием цикла `for`:

```
1  n = int(input("Введите положительное целое число: "))
2
3  f = 1
4
5  for i in range(1, n + 1):
6      f *= i
7
8  print("n! =", f)
9
```

ТОМСКИЙ ПОЛИТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ

Контакты

Долганов Игорь Михайлович,
к.т.н., доцент ОХИ ИШПР

Учебный корпус №2, ауд. 136
dolganovim@tpu.ru
+7-960-978-43-07

Благодарю за внимание!