Введение в Python

Лекция 7: Инструкции цикла в **Python**

Циклы

Практически каждый язык программирования содержит какую-нибудь конструкцию цикла. В большей части языков есть больше одной такой конструкции. В мире Python есть два типа циклов:

- Цикл for
- Цикл while

Циклы (loops) предназначены для повторения команд и блоков кода

Инструкция цикла *for*

Имеется список `num`. Вывести на экран каждый из его элементов отдельно:

```
>>> num=[0.8, 7.0, 6.8, -6]
>>> num
[0.8, 7.0, 6.8, -6]
>>> print(num[0],'- number')
0.8 - number
>>> print(num[1],'- number')
7.0 - number
```

Используем цикл for:

```
>>> num=[0.8, 7.0, 6.8, -6]
>>> for i in num:
print(i, '- number')
0.8 - number
7.0 - number
6.8 - number
-6 - number
>>>
```

В общем виде цикл `for` для перебора всех элементов указанного списка выглядит следующим образом:

for << переменная >> in << список >>:

<< тело цикла >>

Пример:

```
>>> for i in [1, 2, 'hi']:
print(i)
1
2
hi
>>>
```

Цикл `for` работает и для строк:

```
>>> for i in 'hello':
print(i)
h
e
l
l
o
>>>
```

В общем виде запись цикла `for` для заданной строки:

for << переменная >> in << строка >>:

<< тело цикла >>

Цикл `for` также позволяет производить определенные операции:

```
>>> num=[0.8, 7.0, 6.8, -6]
>>> for i in num:
if i == 7.0:
print (i, '- число 7.0')
7.0 - число 7.0
>>>
```

Похожим образом в цикле производится поиск необходимого символа в строке с помощью вызова строкового метода:

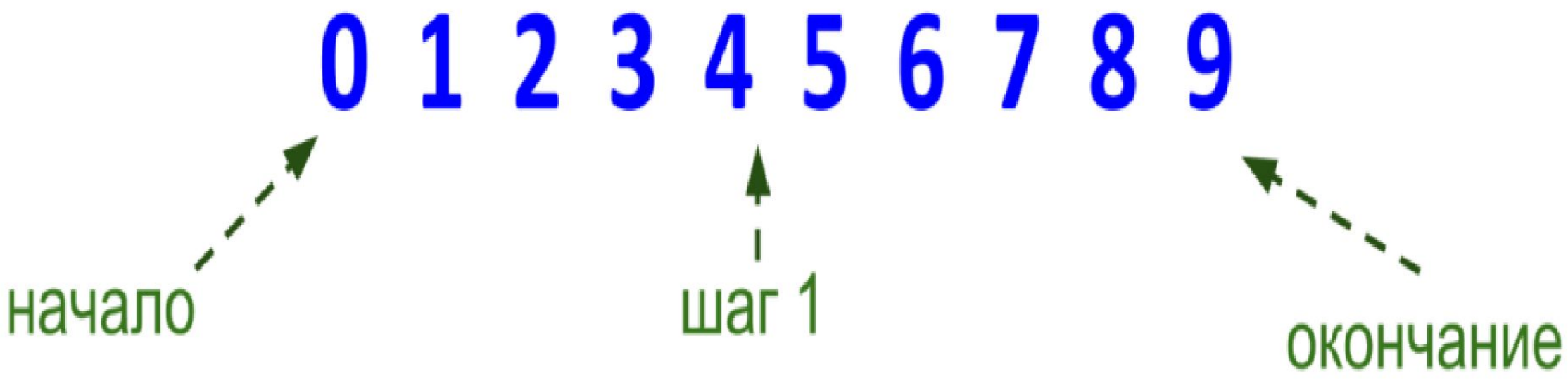
```
>>> country=“Kyrgyzstan"
>>> for ch in country:
  if ch.isupper():
        print(ch)

K
>>>
```

Функция range

- Достаточно часто при разработке программ необходимо получить последовательность (диапазон) целых чисел.
- Для решения этой задачи в Python предусмотрена функция `range()`, создающая последовательность (диапазон) чисел. В качестве аргументов функция принимает: начальное значение диапазона (по умолчанию 0), конечное значение (не включительно) и шаг (по умолчанию 1).

Функция range



Функция range() служит для создания списка чисел, который начинается с того числа, который мы указали первым внутри ее скобок, и заканчивается числом на единицу меньше второго числа. То есть функция выглядит так: range(начало_списка, конец_списка, шаг), где аргументы “начало_списка” и “шаг” являются необязательными. Мы можем обойтись только числом, обозначающим “конец_списка”. В этом случае в качестве начального числа подставится 0, а в качестве шага по умолчанию будет 1:

```
>>> range(0,10,1)
range(0, 10)
>>> range(10)
range(0, 10)
>>>
```

Примеры вызовов функции range

```
1range(5)            # 0, 1, 2, 3, 4
2range(1, 5)         # 1, 2, 3, 4
3range(2, 10, 2)     # 2, 4, 6, 8
4range(5, 0, -1)     # 5, 4, 3, 2, 1
```

Для создания диапазона чисел необходимо использовать цикл `for`:

```
>>> for i in range(0, 10, 1):
print(i, end=' ')
0 1 2 3 4 5 6 7 8 9
>>> for i in range(10):
print(i, end=' ')
0 1 2 3 4 5 6 7 8 9
>>> for i in range(2, 20, 2):
print(i, end=' ')
2 4 6 8 10 12 14 16 18
>>>
```

При желании можно получить диапазон в обратном порядке следования (обратите внимание на аргументы функции `range()`):

```
>>> for i in range(20, 2, -2):
print(i, end=' ')
20 18 16 14 12 10 8 6 4
>>>
```

Теперь с помощью диапазона найдем сумму чисел на интервале от 1 до 100:

```
>>> total=0
>>> for i in range(1, 101):
total=total+i
>>> total
5050
>>>
```

- Переменной `i` на каждом шаге цикла будет присваиваться значение из диапазона от 1 до 100 (крайнее значение не включаем). В цикле мы накапливаем счетчик.

В Python есть более красивое решение данной задачи:

```
>>> sum(list(range(1, 101)))
5050
>>>
```

Диапазоны можно использовать при создании списков:

```
>>> list(range(10))
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
>>> list(range(2, 10, 2))
[2, 4, 6, 8]
>>>
```

Вызов функции `sum()` для списка в качестве аргумента приводит к подсчету суммы всех элементов списка.

Упражнение:

Найдите все значения функции $y(x) = x_2 + 3$ на интервале от 10 до 30 с шагом 2.

Диапазон, создаваемый функцией `range()`, часто используется для задания индексов. Например, если необходимо изменить существующий список, умножив каждый его элемент на `2`:

```
lst = [4, 10, 5, -1.9]
print (lst)
for i in range(len(lst)):
lst [i]=lst [i] * 2
print (lst)
```

В результате выполнения программы:

```
>>>
========= RESTART: C:/Python35-32/myprog.py
======
[4, 10, 5, -1.9]
[8, 20, 10, -3.8]
>>>
```

- Необходимо пройти в цикле по всем элементам списка `lst`, для этого перебираются и изменяются последовательно элементы списка через указание их индекса. В качестве аргумента `range()` задается длина списка. В этом случае создаваемый диапазон будет от 0 до `len(lst)-1`. Python не включает крайний элемент диапазона, т.к. длина списка всегда на 1 больше, чем индекс последнего его элемента, т.к. индексация начинается с нуля.

Подходы к созданию списка

Рассмотрим различные способы создания списков. Самый очевидный способ:

```
>>> a = []
>>> for i in range(1,15):
        a.append(i)
>>> a
[1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14]
>>>
```

С созданием списка из диапазона мы уже встречались:

```
>>> a = list(range(1, 15))
>>> a
[1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14]
>>>
```

Можно также использовать «списковое включение» (или «генератор списка»):

```
>>> a = [i for i in range(1,15)]
>>> a
[1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14]
>>>
```

Правила работы для спискового включения:

```
>>> a = [ i for i in range(1,15) ]
```

что делаем с элементом

что берем

откуда берем

В следующем примере выбираем из диапазона все числа от 1 до 14, возводим их в квадрат и сразу формируем из них новый список:

```
>>> a = [i**2 for i in range(1,15)]
>>> a
[1, 4, 9, 16, 25, 36, 49, 64, 81, 100, 121, 144,
169, 196]
>>>
```

Списковое включение позволяет задавать условие для выбора значения из диапазона (в примере исключили значение 4):

```
>>> a = [i**2 for i in range(1,15) if i!=4]
>>> a
[1, 4, 9, 25, 36, 49, 64, 81, 100, 121, 144,
169, 196]
>>>
```

Вместо диапазонов списковое включение позволяет указывать существующий список:

```
>>> a = [2, -2, 4, -4, 7, 5]
>>> b = [i**2 for i in a]
>>> b
[4, 4, 16, 16, 49, 25]
>>>
```

В примере мы выбираем последовательно значения из списка `a`, возводим в квадрат каждый из его элементов и сразу добавляем полученные значения в новый список.

По аналогии можно перебирать символы из строки и формировать из них список:

```
>>> c = [c*3 for c in 'list' if c != 'i']
>>> c
['lll', 'sss', 'ttt']
>>>
```

В Python есть интересная функция `map()`, которая позволяет создавать новый список на основе существующего списка:

```
>>> def f(x):
        return x+5
>>> list(map(f,[1,3,4]))
[6, 8, 9]
>>>
```

- Функция `map()` принимает в качестве аргументов имя функции и список (или строку). Каждый элемент списка (или строки) подается на вход функции, и результат работы функции добавляется как элемент нового списка. Получить результат вызова функции `map()` можно через цикл `for` или функцию `list()`.
- Функции, которые принимают на вход другие функции, называются *функциями высшего порядка*.

Пример вызова `map()` для строки:

```
>>> def f(s):
return s*2
>>> list(map(f, "hello"))
['hh', 'ee', 'll', 'll', 'oo']
>>>
```

Рассмотрим, как получить список, состоящий из случайных целых чисел:

```
>>> from random import randint
>>> A = [randint(1, 9) for i in range(5)]
>>> A
[2, 1, 1, 7, 8]
>>>
```

- В данном примере функция `range()` выступает как счетчик числа повторений (цикл `for` сработает ровно 5 раз). Обратите внимание, что при формировании нового списка переменная `i` не используется. В результате пять раз будет произведен вызов функции `randint()`, которая сгенерирует целое случайное число из интервала, и уже это число добавится в новый список.

Перейдем к ручному вводу значений для списка. Зададим длину списка и введем с клавиатуры все его значения:

```
a = [] # объявляем пустой список
n = int (input()) # считываем количество элементов в списке
for i in range(n):
    new_element = int(input()) # считываем очередной элемент
    a.append(new_element) # добавляем его в список
# последние две строки можно было заменить одной:
    # a.append(int(input()))
print(a)
```

В результате запуска программы:

```
>>>
======== RESTART: C:\Python35-32\myprog.py ========
3
4
2
1
[4, 2, 1]
>>>
```

Теперь запишем решение этой задачи через списковое включение в одну строку:

```
>>> A = [int(input()) for i in range(int(input()))]
3
4
2
1
>>> A
[4, 2, 1]
>>>
```

Инструкция цикла *while*

- Цикл `for` используется, если заранее известно, сколько повторений необходимо выполнить (указывается через аргумент функции `range()` или пока не закончится список/строка).
- Если заранее количество повторений цикла неизвестно, то применяется другая конструкция, которая называется циклом `while`:

Цикл while:

тело цикла выполняет до тех пор, ПОКА выражение является истинным или пока не вышли по break

```
while << выражение >>:
    << тело цикла >>
```

Tab или 4 пробела

Определим количество кроликов:

```
rabbits = 3
while rabbits > 0:
   print(rabbits)
   rabbits = rabbits - 1
```

В результате выполнения программы:

```
>>>
========= RESTART: C:\Python35-32\myprog.py ==========
3
2
1
>>>
```

- В примере цикл `while` выполняется до тех пор, ПОКА число кроликов в условии положительное. На каждом шаге цикла мы переменную `rabbits` уменьшаем на 1, чтобы не уйти в бесконечный цикл, когда условие всегда будет являться истинным.
- Рассмотрим подробнее ход выполнения программы. В начале работы программы переменная `rabbits` равна 3, затем попадаем в цикл `while`, т.к. условие `rabbits` > 0 будет являться истинным (вернет значение `True`). В теле цикла вызывается функция `print()`, которая отобразит на экране текущее значение переменной `rabbits`. Далее переменная уменьшится на 1 и снова произойдет проверка условия `while`, т.е. 2 > 0 (вернет `True`). Попадаем в цикл и действия повторяются до тех пор, пока не дойдем до условия 0 > 0. В этом случае вернется логическое значение `False` и цикл `while` не сработает.

Рассмотрим следующий пример:

```
while True:
   text = input("Введите число или стоп для   выхода:
")
   if text == "стоп":
      print("Выход из программы! До встречи!")
      break # инструкция выхода из цикла
   elif text == '1':
      print("Число 1")
   else:
      print("Что это?!")
```

В результате работы программы получим:

```
>>>
============= RESTART: C:\Python35-32\myprog.py
============
Введите число или стоп для выхода: 4
Что это?!
Введите число или стоп для выхода: 1
Число 1
Введите число или стоп для выхода: стоп
Выход из программы! До встречи!
>>>
```

- Программа выполняется в бесконечном цикле, т.к. `True` всегда является истиной. Внутри цикла происходит ввод значения с клавиатуры и проверка введенного значения. Инструкция `break` осуществляет выход из цикла.
- В подобных программах необходимо внимательно следить за преобразованием типов данных.

В следующей программе реализован один из вариантов подсчета суммы чисел в строке:

```
s='aa3aBbb6ccc'
total=0
for i in range(len(s)):
  if s[i].isalpha(): # посимвольно проверяем наличие буквы
    continue # инструкция перехода к следующему шагу цикла
  total=total+int(s[i]) #накапливаем сумму, если встретилась цифра
print ("сумма чисел:", total)
```

Результат выполнения:

```
>>>
======== RESTART: C:\Python35-32\myprog.py
=======
сумма чисел: 9
>>>
```

В примере демонстрируется использование инструкции `continue`. Выполнение данной инструкции приводит к переходу к следующему шагу цикла, т.е. все команды, которые находятся после `continue`, будут проигнорированы.

Вложенные циклы

Циклы можно вкладывать друг в друга.

```
outer = [1, 2, 3, 4] # внешний цикл
inner = [5, 6, 7, 8] # вложенный (внутренний) цикл
for i in outer:
    for j in inner:
        print ('i=', i, 'j=', j)
```

Результат работы программы:

```
i= 1 j= 5
i= 1 j= 6
i= 1 j= 7
i= 1 j= 8
i= 2 j= 5
i= 2 j= 6
i= 2 j= 7
i= 2 j= 8
i= 3 j= 5
i= 3 j= 6
i= 3 j= 7
i= 3 j= 8
i= 4 j= 5
i= 4 j= 6
i= 4 j= 7
i= 4 j= 8
```

[1, 2, 3, 4]
[5, 6, 7, 8]

i=1
j=5, 6, 7, 8

i=2
j=5, 6, 7, 8

Пример с одним циклом for:

```
lst = [[1, 2, 3],
[4, 5, 6]]
for i in lst:
  print (i)
```

Результат выполнения программы:

```
>>>
====== RESTART: C:\Python35-32\myprog.py =======
[1, 2, 3]
[4, 5, 6]
>>>
```

Если мы хотим добраться до элементов вложенных списков, то придется использовать вложенный цикл `for`:

```
lst = [[1, 2, 3],
[4, 5, 6]]
for i in lst: # цикл по элементам внешнего
списка
  print()
    for j in i: # цикл по элементам элементов
внешнего списка
      print (j, end="")
```

Результат выполнения программы:

```
>>>
======= RESTART: C:\Python35-32\myprog.py
=======
123
456
>>>
```