Оператор цикла for. Функция range()

На предыдущих занятиях мы с вами познакомились с оператором цикла while, а также вспомогательными операторами break, continue и else. На этом занятии вы узнаете о втором операторе цикла for, который довольно часто используется в Python.

Он имеет следующий синтаксис:

for <переменная> in <итерируемый объект>:

    оператор 1
    оператор 2
    ...
    оператор N

С его помощью очень легко реализовывать перебор, так называемых, итерированных объектов. Что это такое, мы будем говорить на одном из следующих занятий, а сейчас, вам достаточно знать, что это объекты, состоящие из множества элементов, которые можно перебирать. Например, списки или строки.

Как всегда, постичь магию работу этого оператора лучше всего на конкретных примерах. Пусть у нас имеется список:

```
d = [1, 2, 3, 4, 5]
```

И мы хотим перебрать все его элементы. Через оператор цикла for сделать это можно, следующим образом:

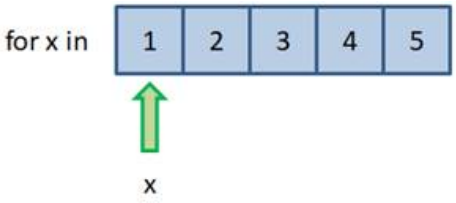
```
for x in d:
    print(x)
```

Или, вместо списка, можно взять строку:

```
for x in "python":
    print(x)
```

Тогда переменная x на каждой итерации будет ссылаться на очередной символ этой строки.

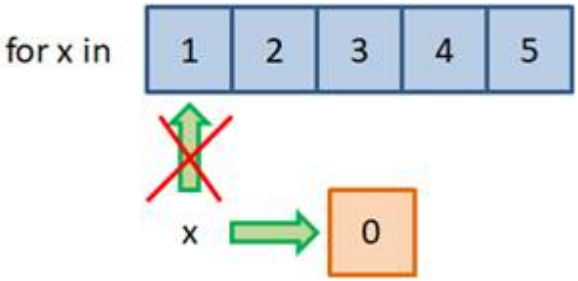
Но давайте посмотрим, как все это работает в деталях. Вернемся к примеру со списком. На первой итерации переменная x будет ссылаться на первый элемент со значением 1. Соответственно, функция print() выводит это значение в консоль. На следующей итерации переменная x ссылается уже на второй элемент и print() выводит значение 2. И так до тех пор, пока не будет достигнут конец списка.



print(x) -> 1
print(x) -> 2
print(x) -> 3
print(x) -> 4
print(x) -> 5

В этой демонстрации ключевое, что переменная x ссылается на элемент списка. То есть, если мы захотим изменить значение в списке, используя переменную x, например, вот так:

```
for x in [1, 2, 3, 4, 5]:
    x = 0
```

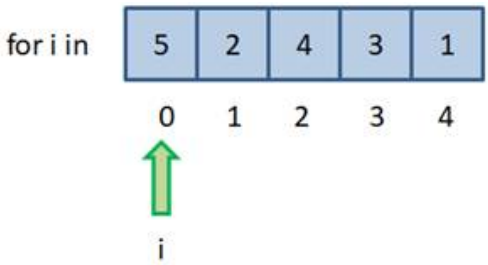


то ничего не получится. Здесь x просто будет ссылаться на другой объект со значением 0, но элементы списка это никак не затронет. То есть, в такой реализации оператора цикла for мы можем лишь перебирать значения элементов и что-то с ними делать, например, вычислять их произведение:

```
d = [5, 2, 4, 3, 1]

p = 1
for x in d:
    p *= x

print(p)
```

А как тогда менять значения элементов в списке с помощью for? Для этого к элементам списка нужно обращаться по индексу. То есть, цикл должен перебирать не элементы списка, а его индексы:

```
d = [5, 2, 4, 3, 1]

for i in [0, 1, 2, 3, 4]:
    d[i] = 0

print(d)
```



d[0] = 0
d[1] = 0
d[2] = 0
d[3] = 0
d[4] = 0

В этом случае, мы на каждой итерации цикла, обращаемся сначала к первому элементу списка, присваиваем ему ноль, затем, ко второму

элементу, присваиваем ему ноль и так для всех остальных элементов. В итоге меняется сам список.

Однако описывать индексы через еще один список, далеко не лучшая практика. Для подобных целей в Python существует специальная функция range(), которая генерирует арифметическую последовательность чисел с параметрами:

range(start, stop, step)

Например, для генерации последовательности от 0 до 4, функцию range() можно записать в виде:

```
range(5)
range(0, 5)
range(0,5,1)
```

То есть, последнее значение 5 не включается в диапазон. Чтобы увидеть сгенерированные числа, преобразуем их последовательность в список с помощью известной нам функции list():

```
list(range(5))
list(range(0, 5))
list(range(0, 5, 1))
```

Если же мы запишем: `list(range(0))`

то получим пустой список, так как значения здесь начинаются с нуля и заканчиваются нулем, при этом, ноль в интервал не входит.

То же самое произойдет при указании любого отрицательного значения, например:

```
list(range(-5))
list(range(0, -5))
```

Но как нам тогда формировать последовательность отрицательных значений? Очень просто, в качестве стартового значения нужно указать число меньше конечного значения:

```
list(range(-10, -5))
```

Или с шагом:

```
list(range(-10, -5, 2))
```

Но если указать отрицательный шаг: `list(range(-10, -5, -2))`

то снова увидим пустой список. Я, думаю, вы догадались почему?Теперь, мы начинаем двигаться от -10 в меньшую сторону и значение -5 становится недостижимым. В этом случае функция range() не выдает никаких значений. Чтобы поправить ситуацию, в качестве конечного значения нужно записать число меньше -10, например, -20: `list(range(-10, -20, -2))`

Вот так генерируются последовательности в обратном порядке. То же самое можно проделать и в положительной области:

```
list(range(5, 0, -1))
```

Обратите внимание, мы начинаем с 5 и заканчиваем 1. Здесь конечное значение 0 не включается в диапазон. Если нам нужно дойти до нуля, то в данном случае следует указать -1 в качестве конечного значения:

```
list(range(5, -1, -1))
```

Вот это всегда следует помнить при работе с функцией range() –конечное значение не включается в диапазон.

Итак, теперь, когда мы с вами узнали, как работает функция range(), перепишем нашу программу с перебором элементов списка по их индексам, следующим образом:

```
d = [5, 2, 4, 3, 1]
for i in range(5):
    d[i] = 0
print(d)
```

Здесь нам не нужно функцию range() превращать в список, оператор цикла for умеет перебирать любые итерируемые объекты, а range(), как раз возвращает такой объект, поэтому переменная i будет принимать значения от 0 до 4 включительно и мы видим, что все значения списка стали равны нулю.

И последний штрих в этой программе. Число 5 лучше заменить вызовом функции определения длины списка: `len(d)`

Тогда получим универсальную программу, работающую со списком любой длины:

```
for i in range(len(d)):
    d[i] = 0
```

В заключение этого занятия приведу еще один пример использования цикла for для вычисления суммы ряда:

S = 1/2 + 1/3 + 1/4 + … + 1/1000

Программу можно реализовать, следующим образом:

```
S = 0
for i in range(2, 1001):
    S += 1 / i
print(S)
```