Python - For Loops

The **for loop** in Python provides the ability to loop over the items of any sequence, such as a list, tuple or a string. It performs the same action on each item of the sequence. This loop starts with the **for** keyword, followed by a variable that represents the current item in the sequence. The **in** keyword links the variable to the sequence you want to iterate over. A **colon (:)** is used at the end of the loop header, and the indented block of code beneath it is executed once for each item in the sequence.

Syntax of Python for Loop

```
for iterating_var in sequence:
   statement(s)
```

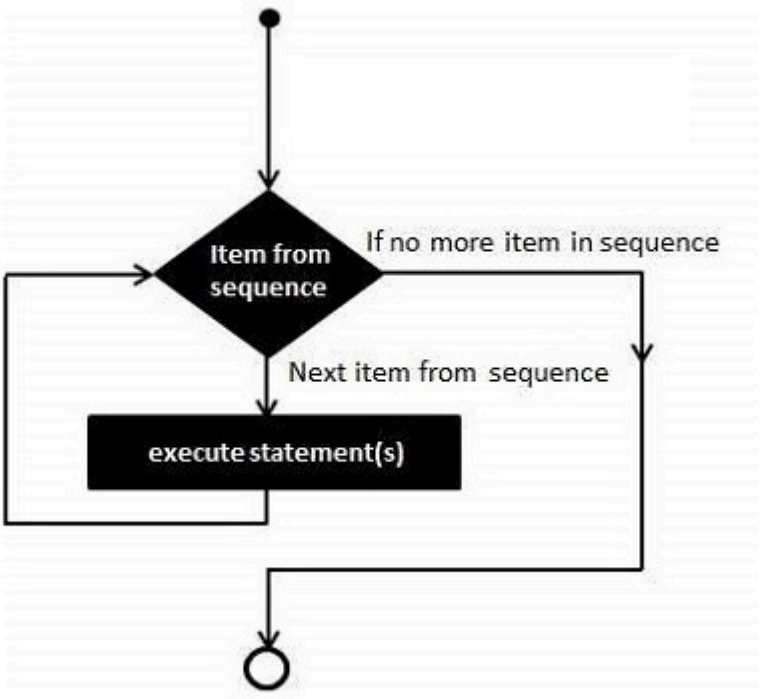
Here, the **iterating_var** is a variable to which the value of each sequence item will be assigned during each iteration. **Statements** represents the block of code that you want to execute repeatedly.

Before the loop starts, the sequence is evaluated. If it's a list, the expression list (if any) is evaluated first. Then, the first item (at index 0) in the sequence is assigned to **iterating_var** variable.

During each iteration, the block of statements is executed with the current value of **iterating_var**. After that, the next item in the sequence is assigned to **iterating_var**, and the loop continues until the entire sequence is exhausted.

Flowchart of Python for Loop

The following flow diagram illustrates the working of **for** loop –



Learn **Python** in-depth with real-world projects through our **Python certification course**. Enroll and become a certified expert to boost your career.

Python for Loop with Strings

A string is a sequence of Unicode letters, each having a positional index. Since, it is a sequence, you can iterate over its characters using the for loop.

Example

The following example compares each character and displays if it is not a vowel ('a', 'e', 'i', 'o', 'u').

</> Open Compiler

```
zen = '''
Beautiful is better than ugly.
Explicit is better than implicit.
Simple is better than complex.
Complex is better than complicated.
'''
for char in zen:
   if char not in 'aeiou':
      print (char, end='')
```

On executing, this code will produce the following output –

```
Btfl s bttr thn gly.
Explct s bttr thn mplct.
Smpl s bttr thn cmplx.
Cmplx s bttr thn cmplctd.
```

Python for Loop with Tuples

Python's tuple object is also an indexed sequence, and hence you can traverse its items with a **for loop**.

Example

In the following example, the for loop traverses a tuple containing integers and returns the total of all numbers.

</> Open Compiler

```
numbers = (34,54,67,21,78,97,45,44,80,19)
total = 0
```

```
for num in numbers:
   total += num
print ("Total =", total)
```

On running this code, it will produce the following output −

```
Total = 539
```

Python for Loop with Lists

Python's list object is also an indexed sequence, and hence you can iterate over its items using a **for loop**.

Example

In the following example, the for loop traverses a list containing integers and prints only those which are divisible by 2.

</> Open Compiler

```
numbers = [34,54,67,21,78,97,45,44,80,19]
total = 0
for num in numbers:
   if num%2 == 0:
      print (num)
```

When you execute this code, it will show the following result −

```
34
54
78
44
80
```

Python for Loop with Range Objects

Python's built-in **range()** function returns an iterator object that streams a sequence of numbers. This object contains integers from start to stop, separated by step parameter. You can run a for loop with range as well.

Syntax

The range() function has the following syntax −

```
range(start, stop, step)
```

Where,

- **Start** – Starting value of the range. Optional. Default is 0
- **Stop** – The range goes upto stop-1
- **Step** – Integers in the range increment by the step value. Option, default is 1.

Example

In this example, we will see the use of range with for loop.

</> Open Compiler

```
for num in range(5):
   print (num, end=' ')
print()
for num in range(10, 20):
   print (num, end=' ')
print()
for num in range(1, 10, 2):
   print (num, end=' ')
```

When you run the above code, it will produce the following output –

```
0 1 2 3 4
10 11 12 13 14 15 16 17 18 19
1 3 5 7 9
```

Python for Loop with Dictionaries

Unlike a list, tuple or a string, dictionary data type in Python is not a sequence, as the items do not have a positional index. However, traversing a dictionary is still possible with the for loop.

Example

Running a simple for loop over the dictionary object traverses the keys used in it.

</> Open Compiler

```
numbers = {10:"Ten", 20:"Twenty", 30:"Thirty",40:"Forty"}
for x in numbers:
   print (x)
```

On executing, this code will produce the following **output** –

```
10
20
30
40
```

Once we are able to get the key, its associated value can be easily accessed either by using square brackets operator or with the **get()** method.

Example

The following example illustrates the above mentioned approach.

</> Open Compiler

```
numbers = {10:"Ten", 20:"Twenty", 30:"Thirty",40:"Forty"}
for x in numbers:
   print (x,":",numbers[x])
```

It will produce the following **output** –

```
10 : Ten
20 : Twenty
30 : Thirty
40 : Forty
```

The items(), keys() and values() methods of dict class return the view objects dict_items, dict_keys and dict_values respectively. These objects are iterators, and hence we can run a for loop over them.

Example

The dict_items object is a list of key-value tuples over which a for loop can be run as follows –

</> Open Compiler

```
numbers = {10:"Ten", 20:"Twenty", 30:"Thirty",40:"Forty"}
for x in numbers.items():
   print (x)
```

It will produce the following **output** –

```
(10, 'Ten')
(20, 'Twenty')
(30, 'Thirty')
(40, 'Forty')
```

Using else Statement with For Loop

Python supports to have an else statement associated with a loop statement. However, the **else** statement is executed when the loop has exhausted iterating the list.

Example

The following example illustrates the combination of an else statement with a for statement that searches for prime numbers from 10 to 20.

</> Open Compiler

```
#For loop to iterate between 10 to 20
for num in range(10, 20):
   #For loop to iterate on the factors
   for i in range(2,num):
      #If statement to determine the first factor
      if num%i == 0:
         #To calculate the second factor
         j=num/i
         print ("%d equals %d * %d" % (num,i,j))
         #To move to the next number
         break
      else:
         print (num, "is a prime number")
         break
```

When the above code is executed, it produces the following result –

```
10 equals 2 * 5
11 is a prime number
12 equals 2 * 6
```

```
13 is a prime number
14 equals 2 * 7
15 equals 3 * 5
16 equals 2 * 8
17 is a prime number
18 equals 2 * 9
19 is a prime number
```