Python - For Loop

Python's for keyword provides a more comprehensive mechanism to constitute a loop. The for loop is used with sequence types such as list, tuple and set. The body of the for loop is executed for each member element in the sequence. Hence, it doesn't require explicit verification of Boolean expression controlling the loop (as in the while loop).

Syntax:

```
for x in sequence:
   statement1
   statement2
   ...
   statementN
```

To start with, variable x in the for statement refers to the item at the 0 index in the sequence. The block of statements with increased uniform indent after the : symbol will be executed. Variable x now refers to the next item and repeats the body of the loop till the sequence is exhausted.

The following example demonstrates the for loop with a list object.

```
>>>> mylist=[10, 20, 30, 40, 50]
>>>> for i in mylist:
...     print(i)
```

Output

```
10
20
30
40
50
```

The following demonstrates the for loop with a tuple object.

```
>>>> mytuple=(10, "twenty", 30, "fourty", 50)
>>>> for i in mytuple:
...     print(i)
```

Output

```
10
twenty
30
fourty
50
```

The following code computes the average of all the numbers in a list.

```
sum=0
numbers =[10,20,30,40]
for num in numbers:
        print(num, end=',')
        sum=sum+num
avg=sum/len(numbers)
print ('\nAverage:', avg)
```

Output

```
10, 20, 30, 40
Average: 25.0
```

For Loop over Dictionary

Key-value pair items in a dictionary do not have an index. The for statement offers a convenient way to traverse a dictionary object. We know that the items() method of the dictionary object returns a list of tuples, each tuple having a key and value corresponding to each item. The following code uses the for statement to print all key-value pairs in the dictionary.

```
dict={ 1:100, 2:200, 3:300 }
for pair in dict.items():
    print (pair)
```

Result:
(1, 100)
(2, 200)
(3, 300)

You can unpack each pair tuple to two variables in the for statement to get the key and value separately.

```
dict={ 1:100, 2:200, 3:300 }
for k,v in dict.items():
   print("key=" + k + ", value=" + v)
```

The output will be the same, but without the tuple notation.

Result:
key=1, value=100
key=2, value=200
key=3, value=300

There is another way to traverse the dictionary. The keys() and values() methods of the dictionary object return list of keys and values from the dictionary, respectively. We also know that the get() method fetches the value corresponding to a key. Here we iterate over the keys and print the corresponding value using the for loop.

```
dict={1:100, 2:200, 3:300}
for k in dict.keys():
   print(k, dict.get(k))
```

Output is same though.

For Loop over String

The object of any Python sequence data type can be iterated using the for statement.

Example: for over string

```
for char in "Hello":
    print (char)
```

Output

```
H
e
l
l
o
```

Nested for Loop

If a loop (for loop or while loop) contains another loop in its body block, we say that the two loops are nested. If the outer loop is designed to perform m iterations and the inner loop is designed to perform n repetitions, the body block of the inner loop will get executed m X n times.

Example: Nested for loop

```
for x in range(1,4):
    for y in range(1,3):
        print ("Hello World")
```

Here, x (the variable controlling the outer loop) will have three values (1, 2, 3) and y (the variable controlling the inner loop) will take two values (1, 2). The 'Hello World' message will be displayed six times.

Result:

```
Hello World
Hello World
Hello World
Hello World
Hello World
Hello World
```