**FOR & NESTED FOR LOOPS**

**For in Loop:** For loops are used for sequential traversal. For example: traversing a list or string or array etc. There is a “for in” loop which is similar to for each loop in other languages. Let us learn how to use for in loop for sequential traversals.

**Syntax:**

for iterator\_var in sequence:

statements(s)

It can be used to iterate over a range and iterators.

| # Python program to illustrate  # Iterating over range 0 to n-1    n = 4  for i in range(0, n):  print(i) |
| --- |

**Output :**

0

1

2

3

| # Python program to illustrate  # Iterating over a list  print("List Iteration")  l = ["geeks", "for", "geeks"]  for i in l:  print(i) |
| --- |

**Output:**

List Iteration

geeks

for

geeks

**Iterating by index of sequences**: We can also use the index of elements in the sequence to iterate. The key idea is to first calculate the length of the list and then iterate over the sequence within the range of this length.

See the below example:

| # Python program to illustrate  # Iterating by index    list = ["geeks", "for", "geeks"]  for index in range(len(list)):  print list[index] |
| --- |

**Output:**

geeks

for

geeks

**Using else statement with for loops:** We can also combine else statement with for loop like in while loop. But as there is no condition in for loop based on which the execution will terminate so the else block will be executed immediately after the for block finishes execution.

Below example explains how to do this:

| # Python program to illustrate  # combining else with for    list = ["geeks", "for", "geeks"]  for index in range(len(list)):  print list[index]  else:  print "Inside Else Block" |
| --- |

**Output:**

geeks

for

geeks

Inside Else Block

**Nested Loops:** Python programming language allows us to use one loop inside another loop. Following section shows a few examples to illustrate the concept.

Syntax:

| for iterator\_var in sequence:  for iterator\_var in sequence:  statements(s)  statements(s) |
| --- |

-------------------X--------------X------------X

Nested for loops, although very useful, should be avoided as much as possible because they iterate through the sequence more than once and doing so can lead to increased code runtime.

-------X---------X-------X

The range function should also be explained in this topic only

The range function - range(i,j) where i,j are both integers returns a range object that gives all the integers from [ i , j -1](both inclusive).

We can also get a list of consecutive integers using the range function by printing

***list(range(i,j))***.

**List Comprehension**

Python is renowned for encouraging developers and programmers to write efficient, easy-to-understand, and almost as simple-to-read code. One of the most distinctive aspects of the language is the python list and the list compression feature, which one can use within a single line of code to construct powerful functionality.A list comprehension consists of brackets containing the expression, which is executed for each element along with the for loop to iterate over each element.

## **Advantages of List Comprehension**

* More time-efficient and space-efficient than loops.
* Require fewer lines of code.
* Transforms iterative statements into a formula.

## **List Comprehensions vs For Loop**

There are various ways to iterate through a list. However, the most common approach is to use the *for* loop. Let us look at the below example:

* Python3

| # Empty list  List = []    # Traditional approach of iterating  for character in 'Geeks 4 Geeks!':  List.append(character)    # Display list  print(List) |
| --- |

**Output:**

*[‘G’, ‘e’, ‘e’, ‘k’, ‘s’, ‘ ‘, ‘4’, ‘ ‘, ‘G’, ‘e’, ‘e’, ‘k’, ‘s’, ‘!’]*

Above is the implementation of the traditional approach to iterate through a list, string, tuple, etc. Now list comprehension does the same task and also makes the program more simple.

List Comprehensions translate the traditional iteration approach using for loop into a simple formula hence making them easy to use. Below is the approach to iterate through a list, string, tuple, etc. using list comprehension.

* Python3

| # Using list comprehension to iterate through loop  List = [character for character in 'Geeks 4 Geeks!']    # Displaying list  print(List) |
| --- |

**Output:**

*[‘G’, ‘e’, ‘e’, ‘k’, ‘s’, ‘ ‘, ‘4’, ‘ ‘, ‘G’, ‘e’, ‘e’, ‘k’, ‘s’, ‘!’]*

\**However, in bigger datasets, list comprehensions just offer more readability and fewer lines of code. The time complexity remains almost unchanged.*