Analysis of Common Loops

Python programming language provides the following types of loops to handle looping requirements. Python provides three ways for executing the loops. While all the ways provide similar basic functionality, they differ in their syntax and condition checking time.

While Loop in Python

In python, a while loop is used to execute a block of statements repeatedly until a given condition is satisfied. And when the condition becomes false, the line immediately after the loop in the program is executed.

Syntax :

```
while expression:
    statement(s)
```

All the statements indented by the same number of character spaces after a programming construct are considered to be part of a single block of code. Python uses indentation as its method of grouping statements.

Example:

Python

```
# Python program to illustrate
# while loop
count = 0
while (count < 3):
    count = count + 1
    print("Hello Geek")
```

Output:

```
Hello Geek
Hello Geek
Hello Geek
```

« Prev
Next »

Using else statement with while loops

Python

```
 # Python program to illustrate
# combining else with while
count = 0
while (count < 3):
    count = count + 1
    print("Hello Geek")
else:
    print("In Else Block")
```

As discussed above, while loop executes the block until a condition is satisfied. When the condition becomes false, the statement immediately after the loop is executed.

The else clause is only executed when your while condition becomes false. If you break out of the loop, or if an exception is raised, it won't be executed.

If else like this:

```
if condition:
    # execute these statements
else:
    # execute these statements
```

and while loop like this are similar

```
while condition:
     # execute these statements
else:
     # execute these statements
```

Examples:

Python

```
 # Python program to illustrate
# combining else with while
count = 0
while (count < 3):
    count = count + 1
    print("Hello Geek")
else:
    print("In Else Block")
```

Output:

```
Hello Geek
Hello Geek
Hello Geek
In Else Block
```

Single statement while block

Just like the if block, if the while block consists of a single statement then we can declare the entire loop in a single line as shown below:

Python

```
 # Python program to illustrate
# Single statement while block
count = 0
while (count == 0): print("Hello Geek")
```

Note: It is suggested **not to use** this type of loops as it is a never ending infinite loop where the condition is always true and you have to forcefully terminate the compiler.

For Loop in Python

For loops are used for sequential traversal. For example: traversing a list or string or array etc. In Python, there is no C style for loop, i.e., for (i=0; i<n; i++). There is “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

```
 # 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
```

Example with List, Tuple, string, and dictionary iteration using For Loops

Python

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

# Iterating over a tuple (immutable)
print("\nTuple Iteration")
t = ("geeks", "for", "geeks")
for i in t:
    print(i)

# Iterating over a String
print("\nString Iteration")
s = "Geeks"
for i in s:
    print(i)

# Iterating over dictionary
print("\nDictionary Iteration")
d = dict()
d['xyz'] = 123
d['abc'] = 345
for i in d:
    print("%s %d" % (i, d[i]))

# Iterating over a set
print("\nSet Iteration")
set1 = {1, 2, 3, 4, 5, 6}
```

```
for i in set1:
    print(i),
```

Dash

All

Articles

Videos

Quiz

Output:

```
List Iteration
geeks
for
geeks

Tuple Iteration
geeks
for
geeks

String Iteration
G
e
e
k
s

Dictionary Iteration
xyz  123
abc  345
```

Iterating by the 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 in iterate over the sequence within the range of this length. See the below example:

Python

```
 # 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 for block finishes execution.
Below example explains how to do this:

Python

```
 # 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 to use one loop inside another loop. Following section shows few examples to illustrate the concept.

Syntax:

```
for iterator_var in sequence:
   for iterator_var in sequence:
       statements(s)
       statements(s)
```

The syntax for a nested while loop statement in the Python programming language is as follows:

```
while expression:
   while expression:
       statement(s)
       statement(s)
```

A final note on loop nesting is that we can put any type of loop inside of any other type of loop. For example, a for loop can be inside a while loop or vice versa.

Python

```
 # Python program to illustrate
# nested for loops in Python
from __future__ import print_function
for i in range(1, 5):
    for j in range(i):
        print(i, end=' ')
    print()
```

Output:

```
1
2 2
3 3 3
4 4 4 4
```

Loop Control Statements

Loop control statements change execution from their normal sequence. When execution leaves a scope, all automatic objects that were created in that scope are destroyed. Python supports the following control statements.

Continue Statement:

It returns the control to the beginning of the loop.

Python

```
# Prints all letters except 'e' and 's'
for letter in 'geeksforgeeks':
    if letter == 'e' or letter == 's':
        continue
```

```
    print ('Current Letter :', letter)
    var = 10
```


Output:

```
Current Letter : g
Current Letter : k
Current Letter : f
Current Letter : o
Current Letter : r
Current Letter : g
Current Letter : k
```

Break Statement:

It brings control out of the loop

Python

```
for letter in 'geeksforgeeks':

    # break the loop as soon it sees 'e'
    # or 's'
    if letter == 'e' or letter == 's':
        break

print 'Current Letter :', letter
```

Output:

```
Current Letter : e
```

Pass Statement:

We use pass statement to write empty loops. Pass is also used for empty control statements, functions and classes.

Python

```
# An empty loop
for letter in 'geeksforgeeks':
    pass
print 'Last Letter :', letter
```

Output:

```
Last Letter : s
```

How for loop in Python works internally?

Before proceeding to this section, you should have a prior understanding of Python Iterators.

Firstly, lets see how a simple for loop looks like.

Python

```
# A simple for loop example

fruits = ["apple", "orange", "kiwi"]

for fruit in fruits:

print(fruit)
```

Output

```
apple
orange
kiwi
```

Here we can see the for loops iterates over iterable object fruit which is a list. Lists, sets, dictionaries are few iterable objects while an integer object is not an iterable object.

For loops can iterate over any iterable object (example: List, Set, Dictionary, Tuple or String).

Now with the help of the above example, let's dive deep and see what happens internally here.

1. Make the list (iterable) an iterable object with help of the iter() function.
2. Run an infinite while loop and break only if the StopIteration is raised.
3. In the try block, we fetch the next element of fruits with the next() function.

4. After fetching the element we did the operation to be performed with the element. (i.e print(fruit))

Python

```
fruits = ["apple", "orange", "kiwi"]

# Creating an iterator object
# from that iterable i.e fruits
iter_obj = iter(fruits)

# Infinite while loop
while True:
try:
    # getting the next item
    fruit = next(iter_obj)
    print(fruit)
except StopIteration:

    # if StopIteration is raised,
    # break from loop
    break
```

Output

```
apple
orange
kiwi
```