loops in python

Python programming language provides 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.

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

# while loop

count = 0

while (count < 3):

    count = count + 1

    print("Hello Dev")
```

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

2. for in Loop: 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)
```

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

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

# Iterating over a String
print("\nString Iteration")
s = "python"
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]))
```

3.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 program to illustrate
# combining else with for
```

```
list = ["python", "java", "ruby"]
for index in range(len(list)):
   print list[index]
else:
   print "Inside Else Block"
```

4.Loop Control Statements: Loop control statements change execution from its 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.

```
# Prints all letters except 'e' and 's'
for letter in 'python coding':
   if letter == 'e' or letter == 's':
      continue
   print 'Current Letter :', letter
   var = 10
```

Asymptotic Analysis

Asymptotic analysis refers to the computing of the running time of any piece of code or the operation in a mathematical unit of a computation. Its operation is computed in terms of a function like f(n). In mathematical analysis, asymptotic analysis, also known as asymptotics, is a method of describing limiting behavior.

The time required by the algorithm falls under the three types: Worst case - Maximum time required by an algorithm and it is mostly used or done while analyzing the algorithm. Best case - Minimum time required for the algorithm or piece of code and it is not normally calculated while analyzing the algorithm. Average case - Average time required for an algorithm or portion of code and it is sometimes done while analyzing the algorithm.

Asymptotic notation

The commonly used notation for calculating the running time complexity of the algorithm is as follows:

- Big O notation
- Big θ notation
- Big Ω notation

Big Oh Notation, O

Big O is used to measure the performance or complexity of an algorithm. In more mathematical term, it is the upper bound of the growth rate of a function, or that if a function g(x) grows no faster than a function f(x), then g is said to be a member of O(f).In general, it is used to express the upper bound of an algorithm and which gives the measure for the worst time complexity or the longest time an algorithm possibly take to complete.

Big Omega Notation, Ω

The notation Ω(n) is the formal way to express the lower bound of an algorithm's running time. It measures the best case time complexity or the best amount of time an algorithm can possibly take to complete.

Big Theta Notation, θ

The notation θ(n) is the formal way to express both the lower bound and the upper bound of an algorithm's running time.

Notation is used to determine the complexity of various algorithm's

Big O notation is mostly used, and it is used mainly all the times to find the upper bound of an algorithm whereas the Big θ notation is sometimes used which is used to detect the average case and the Ω notation is the least used notation among the three.

You will be seeing the examples of the notation used in the algorithm to determine the complexity for that particular algorithm.

What is the time, space complexity of following code :

```
int a = 0, b = 0;
   for (i = 0; i < N; i++) {
      for (j = 0; j < N; j++) {
         a = a + j;
      }
   }
   for (k = 0; k < N; k++) {
      b = b + k;
   }
```