BCA – 502: Python Programming

Rahul Kumar Singh

In today's Class we have discussed on following:-

1) For loop using ranges, string, list and dictionaries

2) Loop manipulation using pass, continue, break and else

Python For Loops using ranges, string, list and dictionaries:-

- A **for** loop is used for iterating over a sequence (that is either a list, a tuple, a dictionary, a set, or a string).
- This is less like the **for** keyword in other programming languages, and works more like an iterator method as found in other object-orientated programming languages.
- The **for** loop does not require an indexing variable to set beforehand.
- With the **for** loop we can execute a set of statements, once for each item in a list, tuple, set etc.

Syntax:-

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

Example:-

```
fruits = ["apple", "banana", "cherry"]

for x in fruits:

  print(x)
```

Output:-

apple

banana

cherry

Looping Through a String:-

Even strings are iterable objects, they contain a sequence of characters:

Example:- Loop through the letters in the word "banana":

```
for x in "banana":

  print(x)
```

Output:-

b

a

n

a

n

a

Looping through range() Function:-

To loop through a set of code a specified number of times, we can use the **range()** function,

The **range()** function returns a sequence of numbers, starting from 0 by default, and increments by 1 (by default), and ends at a specified number.

Example:- Using the range() function:

```
for x in range(6):
  print(x)
```

Output:-

0

1

2

3

4

5

Note that range(6) is not the values of 0 to 6, but the values 0 to 5.

The **range()** function defaults to 0 as a starting value, however it is possible to specify the starting value by adding a parameter: **range(2, 6)**, which means values from 2 to 6 (but not including 6):

Example:- Using the start parameter:

```
for x in range(2, 6):
  print(x)
```

Output:-

2

3

4

5

The **range()** function defaults to increment the sequence by 1, however it is possible to specify the increment value by adding a third parameter: range(2, 30, 3):

Example:- Increment the sequence with 3 (default is 1):

```
for x in range(2, 15, 3):
  print(x)
```

Output:-

2

5

8

11

14

Else in For Loop:-

The **else** keyword in a **for** loop specifies a block of code to be executed when the loop is finished:

Example:-

Print all numbers from 0 to 5, and print a message when the loop has ended:

```
for x in range(6):
  print(x)
else:
  print("Finally finished!")
```

Output:-

0

1

2

3

4

5

Finally finished!

Note: The else block will NOT be executed if the loop is stopped by a break statement.

Example

Break the loop when x is 3, and see what happens with the else block:

```
for x in range(6):
  if x == 3: break
  print(x)
else:
  print("Finally finished!")

#If the loop breaks, the else block is not executed.
```

Output:-

0

1

2

Nested Loops:-

- A nested loop is a loop inside a loop.
- The "inner loop" will be executed one time for each iteration of the "outer loop":

Example:-

Print each adjective for every fruit:

```
adj = ["red", "big", "tasty"]
fruits = ["apple", "banana", "cherry"]
for x in adj:
  for y in fruits:
    print(x, y)
```

Output:-

```
red apple
red banana
red cherry
big apple
big banana
big cherry
tasty apple
tasty banana
tasty cherry
```

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 3 types of control statements. Which are as below:

1) Break Statement

2) Continue Statement

3) Pass Statement

1) Break Statement:-

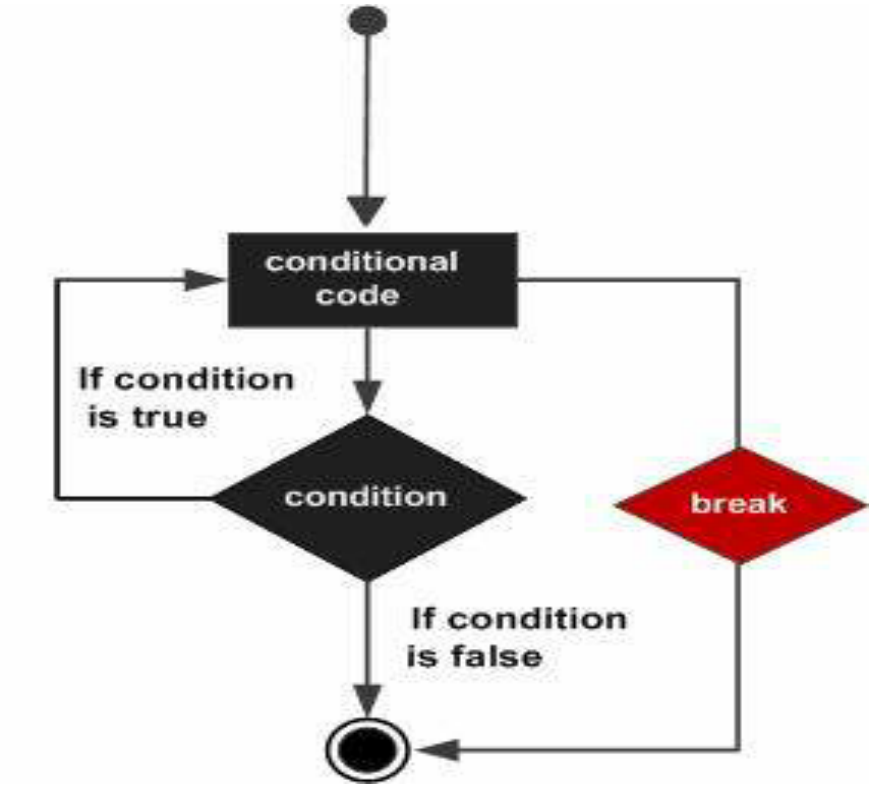
- Terminates the loop statement and transfers execution to the statement immediately following the loop.
- It terminates the current loop and resumes execution at the next statement, just like the traditional break statement in C.
- The most common use for break is when some external condition is triggered requiring a hasty exit from a loop. The break statement can be used in both while and for loops.
- If you are using nested loops, the break statement stops the execution of the innermost loop and start

executing the next line of code after the block.

Syntax:-

break

Flow Diagram



Example:-

```
#!/usr/bin/python
for letter in 'Python':     # First Example
   if letter == 'h':
      break
```

```
   print 'Current Letter :', letter
var = 10                    # Second Example
while var > 0:
   print 'Current variable value :', var
   var = var -1
   if var == 5:
      break
print "Good bye!"
```

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

```
Current Letter : P
Current Letter : y
Current Letter : t
Current variable value : 10
Current variable value : 9
Current variable value : 8
Current variable value : 7
Current variable value : 6
Good bye!
```

2) Continue Statement:-

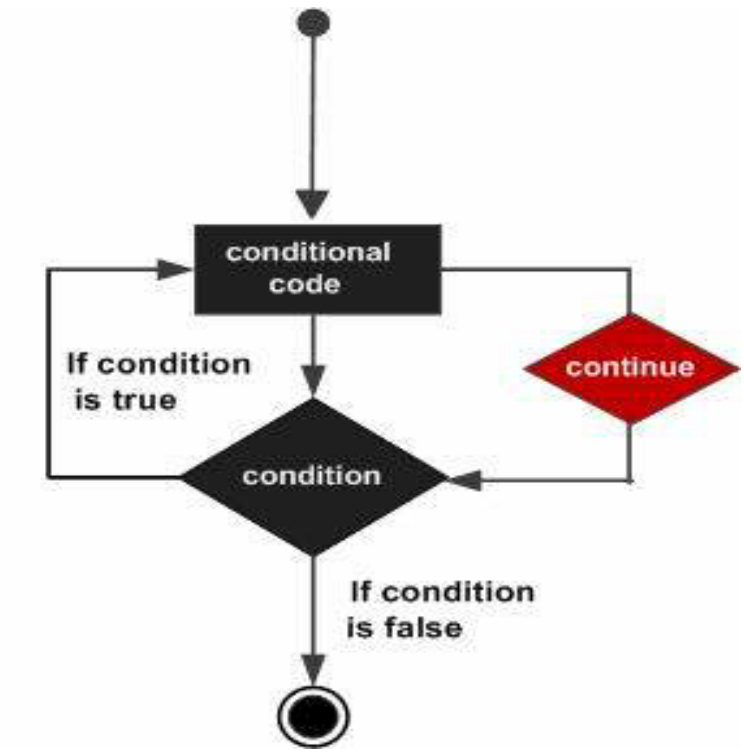
- Causes the loop to skip the remainder of its body and immediately retest its condition prior to reiterating.
- It returns the control to the beginning of the while loop.. The continue statement rejects all the remaining statements in the current iteration of the loop and moves the control back to the top of the loop.

The continue statement can be used in both while and for loops.

Syntax:-

continue

Flow Diagram:-



Example:-

```
#!/usr/bin/python

for letter in 'Python':     # First Example
   if letter == 'h':
      continue
   print 'Current Letter :', letter
var = 9                   # Second Example
while var > 0:
   var = var -1
   if var == 5:
      continue
   print 'Current variable value :', var
print "Good bye!"
```

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

```
Current Letter : P
Current Letter : y
Current Letter : t
Current Letter : o
Current Letter : n
Current variable value : 8
```

Current variable value : 7

Current variable value : 6

Current variable value : 4

Current variable value : 3

Current variable value : 2

Current variable value : 1

Current variable value : 0

Good bye!

3) Pass Statement:-

- The pass statement in Python is used when a statement is required syntactically but you do not want any command or code to execute.
- It is used when a statement is required syntactically but you do not want any command or code to execute.
- The pass statement is a null operation; nothing happens when it executes. The pass is also useful in places where your code will eventually go, but has not been written yet (e.g., in stubs for example) –

Syntax:-

pass

Example:-

```
#!/usr/bin/python
for letter in 'Python':
   if letter == 'h':
      pass
      print 'This is pass block'
   print 'Current Letter :', letter
print "Good bye!"
```

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

```
Current Letter : P
Current Letter : y
Current Letter : t
This is pass block
Current Letter : h
Current Letter : o
Current Letter : n
Good bye!
```