Python Tutorials and Notes

Python break, continue, pass statements with Examples

SETScholars & WACAMLDS

The concept of loops is available in almost all programming languages. Python loops help to iterate over a list, tuple, string, dictionary, and a set. There are two types of loop supported in Python "for" and "while". The block of code is executed multiple times inside the loop until the condition fails.

The loop control statements break the flow of execution and terminate/skip the iteration as per our need. Python break and continue are used inside the loop to change the flow of the loop from its standard procedure.

A for-loop or while-loop is meant to iterate until the condition given fails. When you use a break or continue statement, the flow of the loop is changed from its normal way.

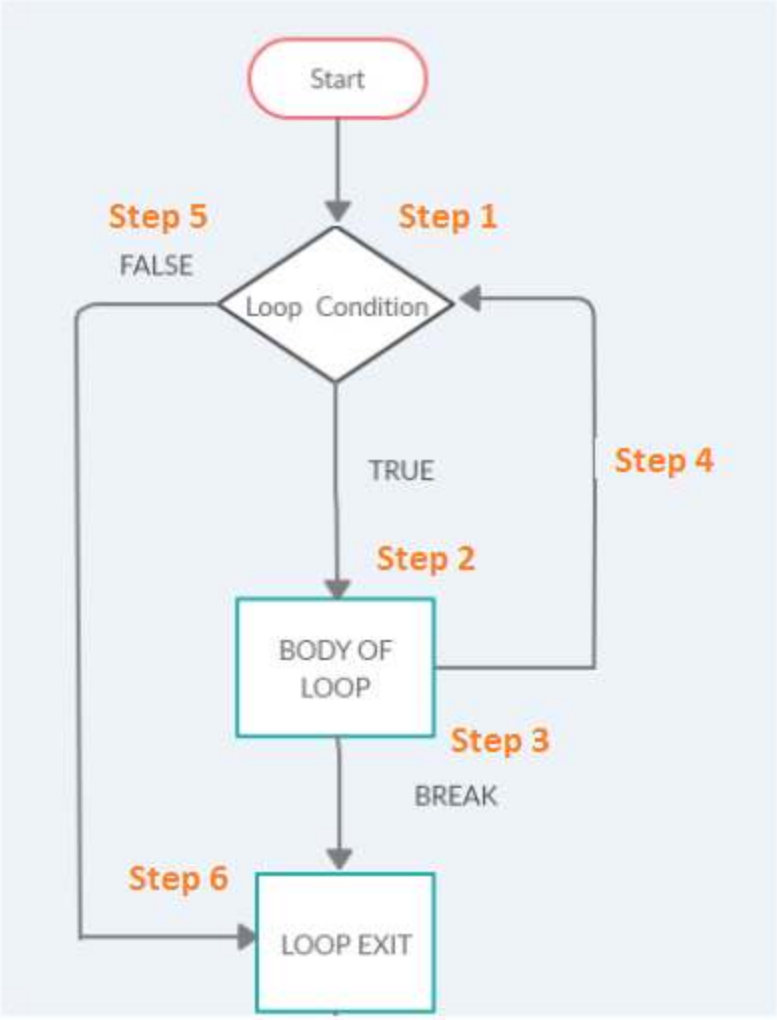
In this Python tutorial, you will learn:

- Python break statement
- Break statement execution flow
- Python continue statement
- Continue statement execution flow
- Python pass statement
- What is pass statement in Python?
- When to use a break and continue statement?

Python break statement

The break statement takes care of terminating the loop in which it is used. If the break statement is used inside nested loops, the current loop is terminated, and the flow will continue with the code followed that comes after the loop.

The flow chart for the break statement is as follows:



The following are the steps involved in the flowchart.

Step 1)

The loop execution starts.

Step 2)

If the loop condition is true, it will execute step 2, wherein the body of the loop will get executed.

Step 3)

If the loop's body has a break statement, the loop will exit and go to Step 6.

Step 4)

After the loop condition is executed and done, it will proceed to the next iteration in Step 4.

Step 5)

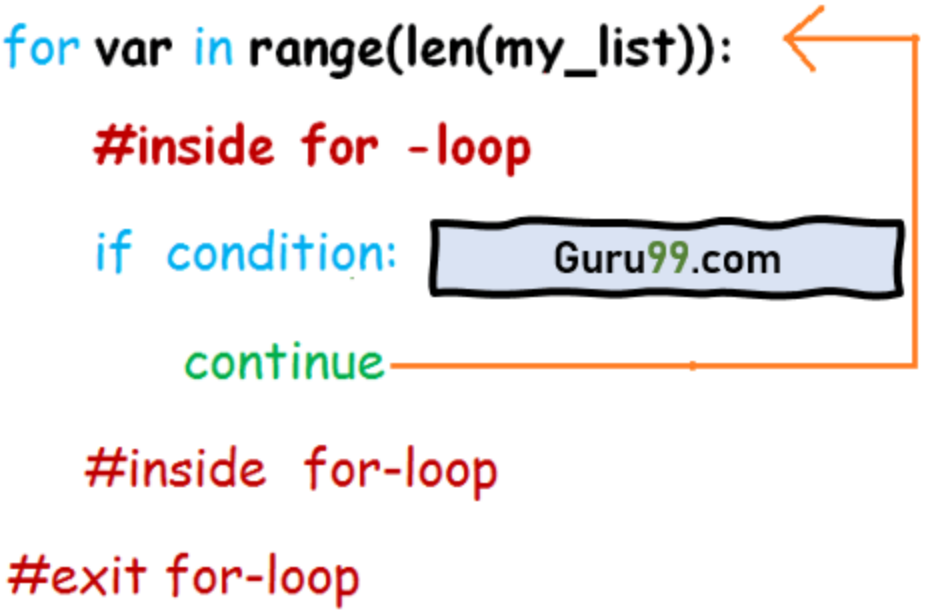
If the loop condition is false, it will exit the loop and go to Step 6.

Step 6)

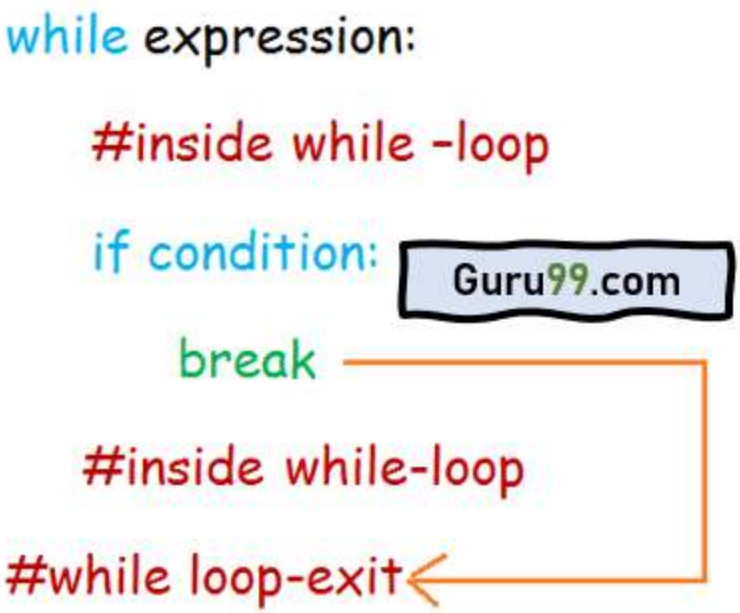
End of the loop.

Break statement execution flow

When the for-loop starts executing, it will check the if-condition. If **true**, the break statement is executed, and the for–loop will get terminated. If the condition is false, the code inside for-loop will be executed.



When the while loop executes, it will check the if-condition; if it is **true,** the break statement is executed, and the while –loop will exit. If if the condition is false, the code inside while-loop will get executed.



Example: Break statement inside for-loop

The list my_list = ['Siya', 'Tiya', 'Guru', 'Daksh', 'Riya', 'Guru'] is looped using for-loop.We are interested in searching for the name 'Guru ' from the list my_list.

Inside the for-loop, the if-condition compares each item from the list with the name 'Guru'. If the condition becomes true, it will execute the break statement, and the loop will get terminated.

The working example using break statement is as shown below:

```
my_list = ['Siya', 'Tiya', 'Guru', 'Daksh', 'Riya', 'Guru']

for i in range(len(my_list)):
    print(my_list[i])
    if my_list[i] == 'Guru':
        print('Found the name Guru')
        break
        print('After break statement')

print('Loop is Terminated')
```

Output:

```
Siya
Tiya
Guru
Found the name Guru
Loop is Terminated
```

Example: Break statement inside while-loop

```
my_list = ['Siya', 'Tiya', 'Guru', 'Daksh', 'Riya', 'Guru']
i = 0

while True:
    print(my_list[i])
    if (my_list[i] == 'Guru'):
        print('Found the name Guru')
        break
        print('After break statement')
    i += 1

print('After while-loop exit')
```

Output:

```
Siya
Tiya
Guru
Found name Guru
After while-loop exit
```

Example: Break Statement inside nested loops

In the example, we have 2 for-loops. Both for-loops are iterating from a range of 0 to 3. In the second for-loop, we have added a condition where-in if the value of the second for-loop index is 2, it should break.

So because of the break statement, the second for-loop will never iterate for 2 and 3.

```
for i in range(4):
    for j in range(4):
        if j==2:
            break
        print("The number is ",i,j);
```

Output:

```
The number is  0 0
The number is  0 1
The number is  1 0
The number is  1 1
The number is  2 0
The number is  2 1
The number is  3 0
The number is  3 1
```

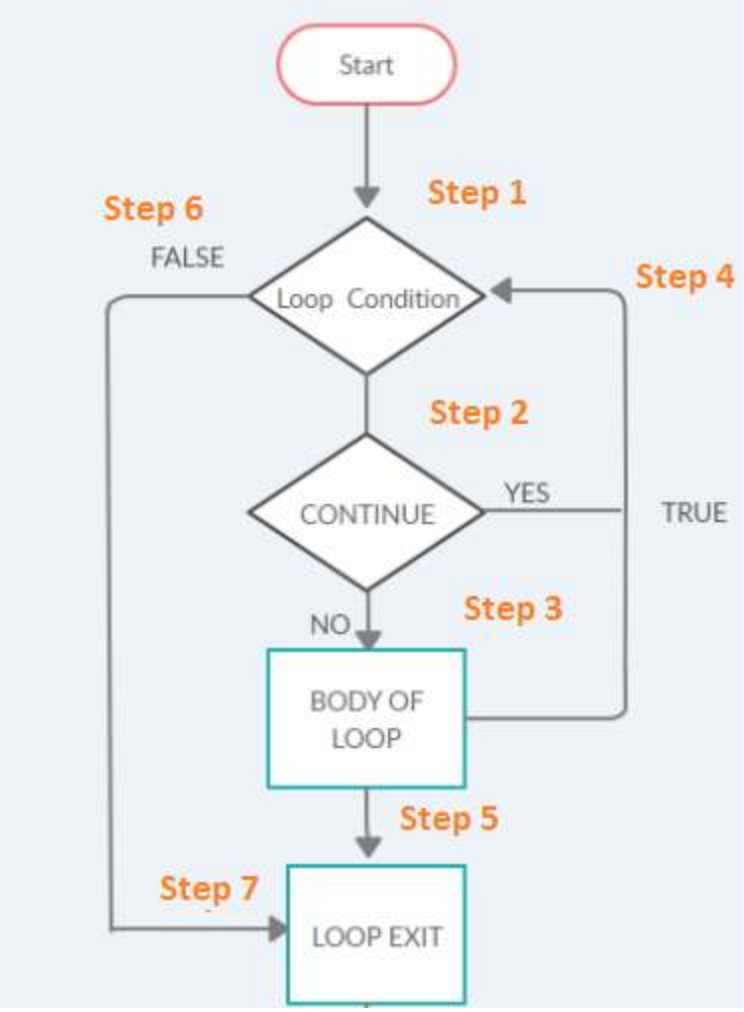
Python continue statement

The **continue** statement skips the code that comes after it, and the control is passed back to the start for the next iteration.

Syntax:

```
continue
```

Continue flow Chart



The following are the steps involved in the flowchart.

Step 1)

The loop execution starts.

Step 2)

The execution of code inside the loop will be done. If there is a continued statement inside the loop, the control will go back to Step 4, i.e., the start of the loop for the next iteration.

Step 3)

The execution of code inside the loop will be done.

Step 4)

If there is a continue statement or the loop execution inside the body is done, it will call the next iteration.

Step 5)

Once the loop execution is complete, the loop will exit and go to step 7.

Step 6)

If the loop condition in step 1 fails, it will exit the loop and go to step 7.

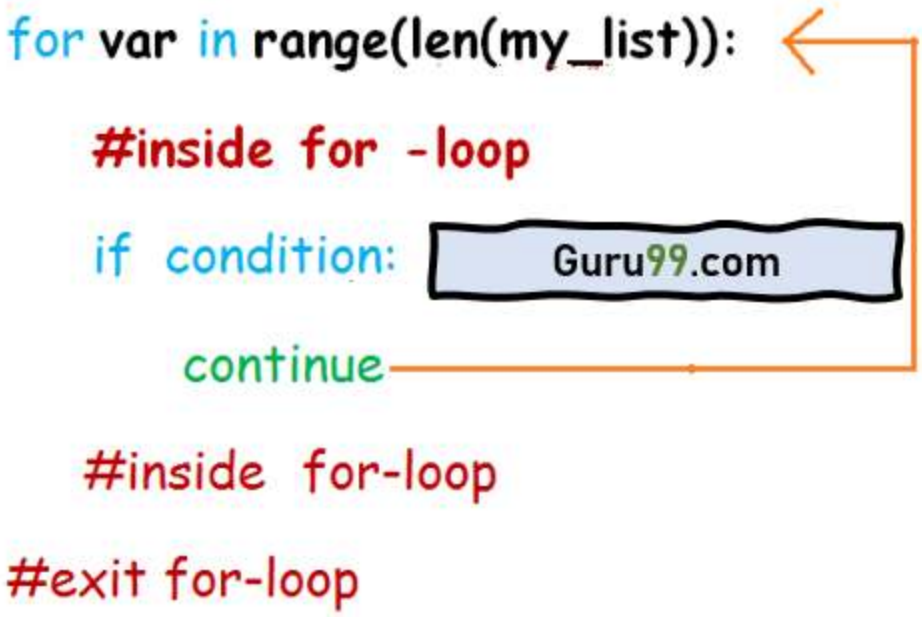
Step 7)

End of the loop.

Continue statement execution flow

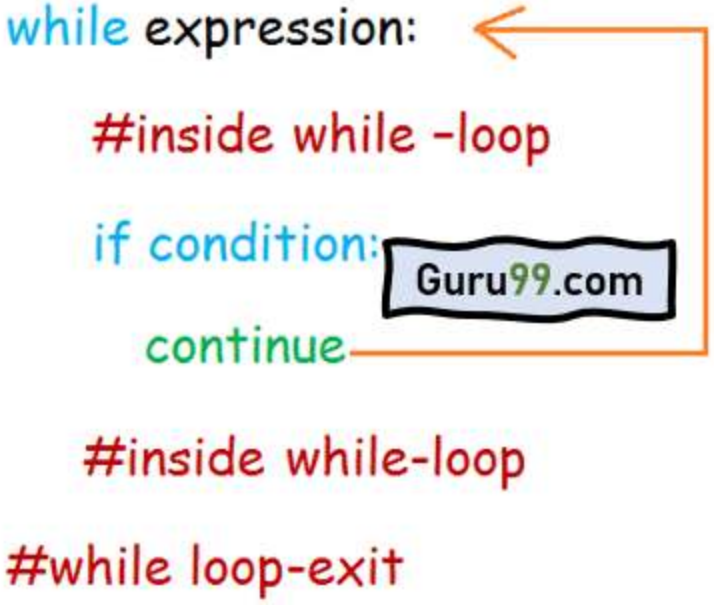
The for –loop, loops through my_list array given. Inside the for-loop, the if-condition gets executed. If the condition is **true**, the continue statement is executed, and the control will pass to the start of the loop for the next iteration.

The flow of the code is as shown below:



When the while loop executes, it will check the if-condition, if it is **true,** the continue statement is executed. The control will go back to the start of while – loop for the next iteration. If if the condition is false, the code inside while-loop will get executed.

The flow of the code is as shown below:



Example : Continue inside for-loop

```
for i in range(10):
    if i == 7:
        continue
    print("The Number is :" , i)
```

Output:

```
The Number is : 0
The Number is : 1
The Number is : 2
The Number is : 3
The Number is : 4
The Number is : 5
The Number is : 6
The Number is : 8
The Number is : 9
```

Example : Continue inside while-loop

```
i = 0
while i <= 10:
    if i == 7:
        i += 1
        continue
    print("The Number is  :" , i)
    i += 1
```

Output:

```
The Number is  : 0
The Number is  : 1
The Number is  : 2
The Number is  : 3
The Number is  : 4
The Number is  : 5
The Number is  : 6
The Number is  : 8
The Number is  : 9
The Number is  : 10
```

Example: Continue inside nested-loop

The below example shows using 2 for-loops. Both for-loops are iterating from a range of 0 to 3. In the second for-loop, there is a condition, wherein if the value of the second for-loop index is 2, it should **continue**. So because of the **continue** statement, the second for-loop will skip iteration for 2 and proceed for 3.

```
for i in range(4):
    for j in range(4):
        if j==2:
            continue
        print("The number is ",i,j);
```

Output:

```
The number is  0 0
The number is  0 1
The number is  0 3
The number is  1 0
The number is  1 1
The number is  1 3
The number is  2 0
The number is  2 1
The number is  2 3
The number is  3 0
The number is  3 1
The number is  3 3
```

Python pass statement

Python pass statement is used as a placeholder inside loops, functions, class, if-statement that is meant to be implemented later.

Syntax

```
pass
```

What is pass statement in Python?

Python pass is a null statement. When the Python interpreter comes across the across pass statement, it does nothing and is ignored.

When to use the pass statement?

Consider you have a function or a class with the body left empty. You plan to write the code in the future. The Python interpreter will throw an error if it comes across an empty body.

A comment can also be added inside the body of the function or class, but the interpreter ignores the comment and will throw an error.

The pass statement can be used inside the body of a function or class body. During execution, the interpreter, when it comes across the pass statement, ignores and continues without giving any error.

Example: pass statement inside a function

In the example, the pass is added inside the function. It will get executed when the function is called as shown below:

```
def my_func():
    print('pass inside function')
    pass
my_func()
```

Output:

```
pass inside function
```

Example: pass statement inside the class

In the example below, we have created just the empty class that has a print statement followed by a pass statement. The pass statement is an indication that the code inside the class "My_Class" will be implemented in the future.

```
classMy_Class:
print("Inside My_Class")
pass
```

Output:

```
Inside My_Class
```

Example: pass statement inside the loop

In the example below, the string 'Guru' is used inside for-loop. The if-condition checks for character 'r' and calls the print statement followed by pass.

```
# Pass statement in for-loop
test = "Guru"
for i in test:
   if i == 'r':
      print('Pass executed')
      pass
   print(i)
```

Output:

```
G
u
Pass executed
r
u
```

Example : pass statement inside if-loop

In the example the if loop checks for the value of a and if the condition is true it goes and prints the statement "pass executed" followed by pass.

```
a=1
if a==1:
   print('pass executed')
   pass
```

Output:

```
pass executed
```

When to use a break and continue statement?

- A **break** statement, when used inside the loop, will terminate the loop and exit. If used inside nested loops, it will break out from the current loop.
- A **continue** statement will stop the current execution when used inside a loop, and the control will go back to the start of the loop.

The main difference between break and continue statement is that when break keyword is encountered, it will exit the loop.

In case of continue keyword, the current iteration that is running will be stopped, and it will proceed with the next iteration.

Summary:

- Python break and continue are used inside the loop to change the flow of the loop from its normal procedure.
- A for-loop or while-loop is meant to iterate until the condition given fails. When you use a break or continue statement, the flow of the loop is changed from its normal way.
- A **break** statement, when used inside the loop, will terminate the loop and exit. If used inside nested loops, it will break out from the current loop.
- A **continue** statement, when used inside a loop, will stop the current execution, and the control will go back to the start of the loop.
- The main difference between **break** and **continue** statement is that when **break** keyword is encountered, it will exit the loop.
- Python Pass Statement is used as a placeholder inside loops, functions, class, if-statement that is meant to be implemented later.
- Python pass is a null statement. When the execution starts and the interpreter comes across the pass statement, it does nothing and is ignored.