Python Break, Continue, and Pass

We use break, continue statements to alter the loop's execution in a certain manner.

Statement	Description
`break`	Terminate the current loop. Use the break statement to come out of the loop instantly.
`continue`	Skip the current iteration of a loop and move to the next iteration
`pass`	Do nothing. Ignore the condition in which it occurred and proceed to run the program as usual

Loop control statements in Python

The `break` and `continue` statements are part of a control flow statements that helps you to understand the basics of Python.

Break Statement in Python

The `break` statement is used inside the loop to exit out of the loop. In Python, when a `break` statement is encountered inside a loop, the loop is immediately terminated, and the program control transfer to the next statement following the loop.

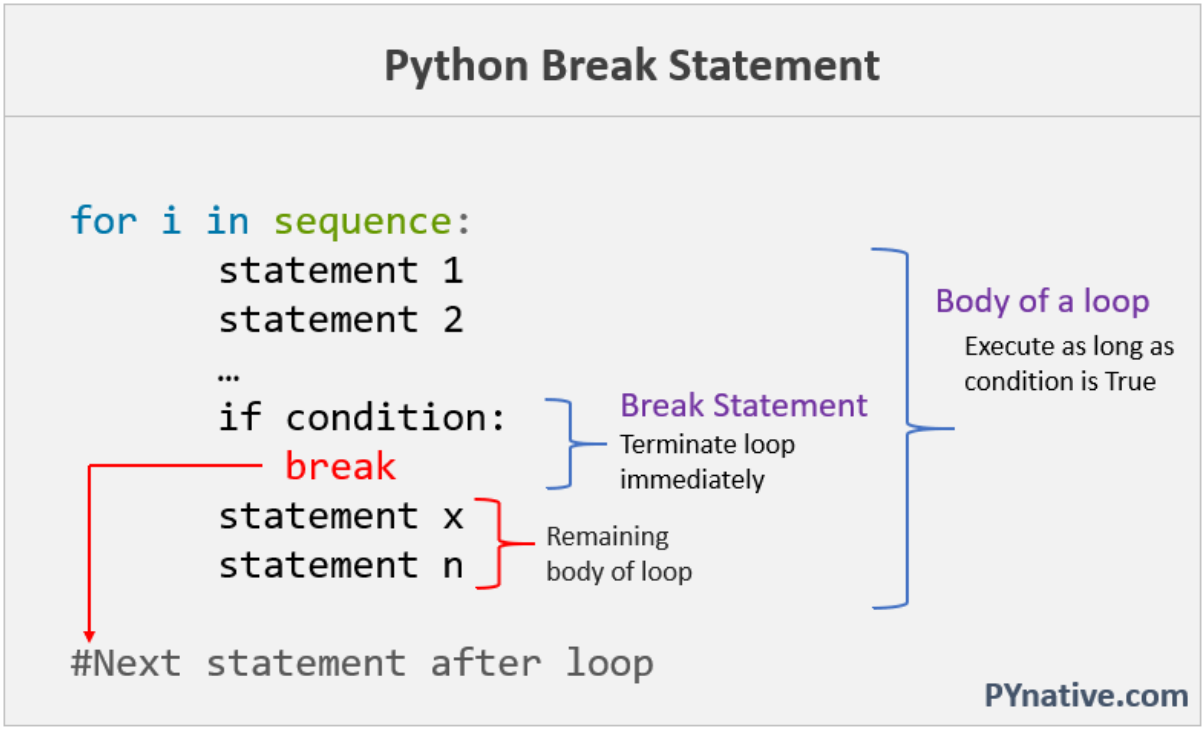
In simple words, A `break` keyword terminates the loop containing it. If the `break` statement is used inside a nested loop (loop inside another loop), it will terminate the innermost loop.

For example, you are searching a specific email inside a file. You started reading a file line by line using a loop. When you found an email, you can stop the loop using the break statement.

We can use Python `break` statement in both for loop and while loop. It is helpful to terminate the loop as soon as the condition is fulfilled instead of doing the remaining iterations. It reduces execution time.

Syntax of `break`:

```
break
```



Break loop in Python

Let us see the usage of the `break` statement with an example.

Example: Break for loop in Python

In this example, we will iterate numbers from a list using a for loop, and if we found a number greater than 100, we will break the loop.

Use the if condition to terminate the loop. If the condition evaluates to true, then the loop will terminate. Else loop will continue to work until the main loop condition is true.

```
numbers = [10, 40, 120, 230]
for i in numbers:
    if i > 100:
        break
    print('current number', i)
```

Output:

```
curret number 10

curret number 40
```

Note: As you can see in the output, we got numbers less than 100 because we used the `break` statement inside the if condition (the number is greater than 100) to terminate the loop

How break statement works

We used a `break` statement along with if statement. Whenever a specific condition occurs and a `break` statement is encountered inside a loop, the loop is immediately terminated, and the program control transfer to the next statement following the loop.

Let's understand the above example iteration by iteration.

- In the first iteration of the loop, 10 gets printed, and the condition `i > 100` is checked. Since the value of variable `i` is 10, the condition becomes false.
- In the second iteration of the loop, 20 gets printed again, and the condition `i > 100` is checked. Since the value of `i` is 40, the condition becomes false.
- In the third iteration of the loop, the condition `i > 100` becomes true, and the `break` statement terminates the loop

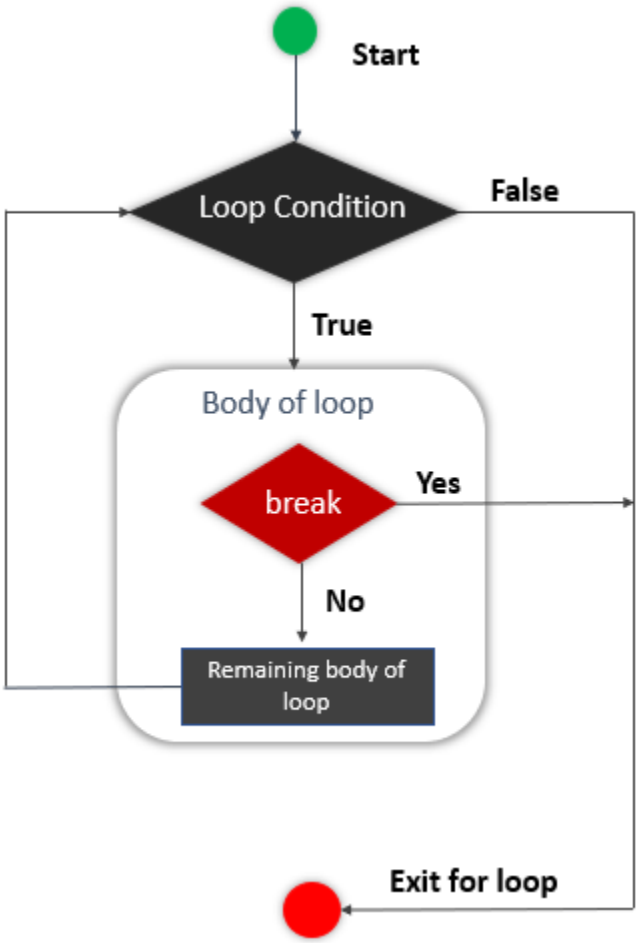
Example: Break while loop

We can use the `break` statement inside a while loop using the same approach.

Write a while loop to display each character from a string and if a character is a space then terminate the loop.

Use the if condition to stop the while loop. If the current character is space then the condition evaluates to true, then the `break` statement will execute and the loop will terminate. Else loop will continue to work until the main loop condition is true.

```
name = 'Jesaa29 Roy'

size = len(name)
i = 0
# iterate loop till the last character
while i < size:
    # break loop if current character is space
    if name[i].isspace():
        break
    # print current character
    print(name[i], end=' ')
    i = i + 1
```



Flow chart of a break statement

Break Nested Loop in Python

To terminate the nested loop, use a `break` statement inside the inner loop. Let's see the example.

In the following example, we have two loops, the outer loop, and the inner loop. The outer for loop iterates the first 10 numbers using the range () function, and the internal loop prints the multiplication table of each number.

But if the current number of both the outer loop and the inner loop is greater than 5 then terminate the inner loop using the break statement.

Example: Break nested loop

```
for i in range(1, 11):
    print('Multiplication table of', i)
    for j in range(1, 11):
        # condition to break inner loop
        if i > 5 and j > 5:
            break
        print(i * j, end=' ')
    print('')
```

Break Outer loop in Python

To terminate the outside loop, use a `break` statement inside the outer loop. Let's see the example.

In the following example, we have two loops, the outer loop, and the inner loop. The outer loop iterates the first 10 numbers, and the internal loop prints the multiplication table of each number.

But if the current number of the outer loop is greater than 5 then terminate the outer loop using the `break` statement.

Example: Break outer loop

```
for i in range(1, 11):
    # condition to break outer loop
    if i > 5:
        break
    print('Multiplication table of', i)
    for j in range(1, 11):
        print(i * j, end=' ')
    print('')
```

Continue Statement in Python

The `continue` statement skip the current iteration and move to the next iteration. In Python, when the `continue` statement is encountered inside the loop, it skips all the statements below it and immediately jumps to the next iteration.

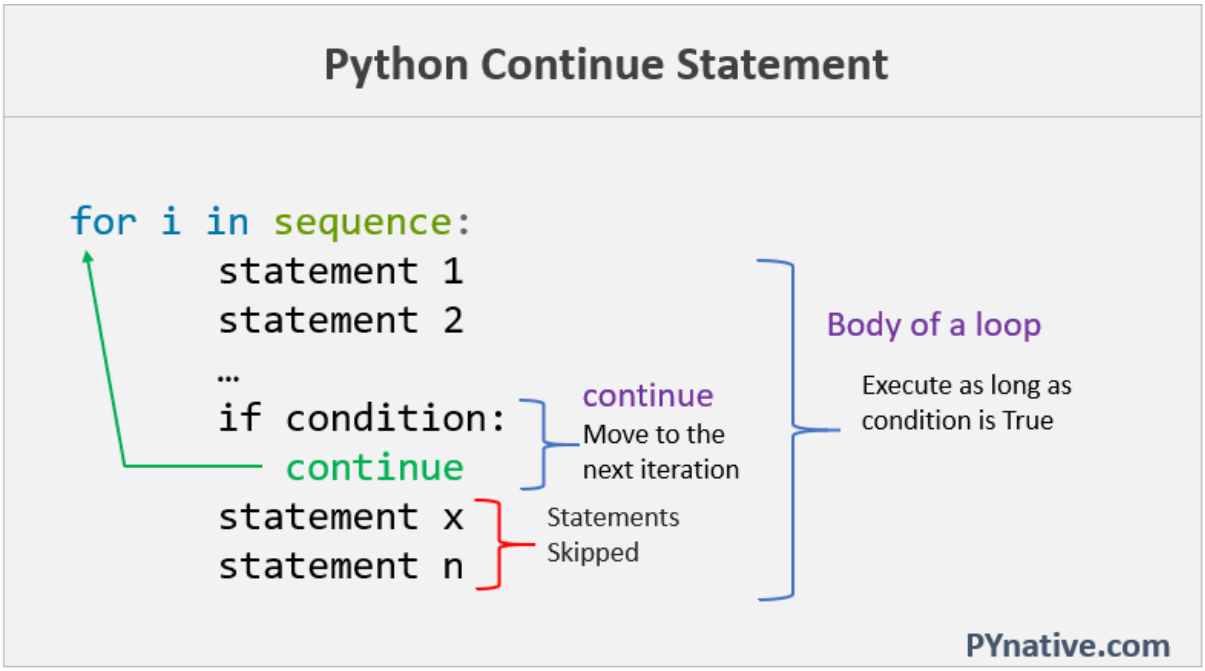
In simple words, the `continue` statement is used inside loops. Whenever the `continue` statement is encountered inside a loop, control directly jumps to the start of the loop for the next iteration, skipping the rest of the code present inside the loop's body for the current iteration.

In some situations, it is helpful to skip executing some statement inside a loop's body if a particular condition occurs and directly move to the next iteration.

Syntax of `continue`:

```
continue
```


Let us see the use of the `continue` statement with an example.



Python continue statement in loop

Example: continue statement in for loop

In this example, we will iterate numbers from a list using a for loop and calculate its square. If we found a number greater than 10, we will not calculate its square and directly jump to the next number.

Use the if condition with the `continue` statement. If the condition evaluates to true, then the loop will move to the next iteration.

```
numbers = [2, 3, 11, 7]
for i in numbers:
    print('Current Number is', i)
    # skip below statement if number is greater than 10
    if i > 10:
        continue
    square = i * i
    print('Square of a current number is', square)
```

Output:

```
Current Number is 2

Square of a current number is 4
```

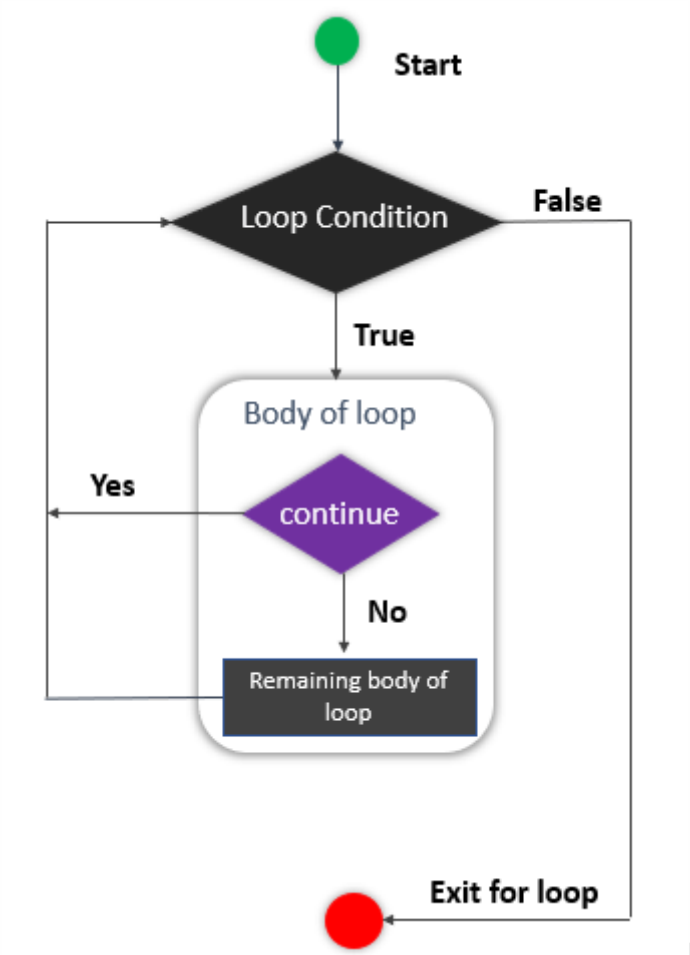
```
Current Number is 3

Square of a current number is 9

Current Number is 11

Current Number is 7

Square of a current number is 49
```

Note: As you can see in the output, we got square of 2, 3, and 7, but the loop ignored number 11 because we used the if condition to check if the number is greater than ten, and the condition evaluated to true, then loop skipped calculating the square of 11 and moved to the next number.

How continue statement works

We used the `continue` statement along with the if statement. Whenever a specific condition occurs and the `continue` statement is encountered inside a

loop, the loop immediately skips the remeaning body and move to the next iteration.



Flow chart of a continue statement

Let's understand the above example iteration by iteration.

- In the first iteration of the loop, 4 gets printed, and the condition `i > 10` is checked. Since the value of `i` is 2, the condition becomes false.
- In the second iteration of the loop, 9 gets printed, and the condition `i > 10` is checked. Since the value of `i` is 9, the condition becomes false.
- In the third iteration of the loop, the condition `i > 10` becomes true, and the `continue` statement skips the remaining statements and moves to the next iteration of the loop

- In the second iteration of the loop, 49 gets printed, and the condition `i > 10` is checked. Since the value of `i` is 7, the condition becomes false.

Example: continue statement in while loop

We can also use the `continue` statement inside a while loop. Let's understand this with an example.

Write a while loop to display each character from a string and if a character is a space, then don't display it and move to the next character.

Use the if condition with the `continue` statement to jump to the next iteration. If the current character is space, then the condition evaluates to true, then the `continue` statement will execute, and the loop will move to the next iteration by skipping the remeaning body.

```
name = 'Je sa a'

size = len(name)
i = -1
# iterate loop till the last character
while i < size - 1:
    i = i + 1
    # skip loop body if current character is space
    if name[i].isspace():
        continue
    # print current character
    print(name[i], end=' ')
```


Output:

```
J e s a a
```

Continue Statement in Nested Loop

To skip the current iteration of the nested loop, use the `continue` statement inside the body of the inner loop. Let's see the example.

In the following example, we have the outer loop and the inner loop. The outer loop iterates the first 10 numbers, and the internal loop prints the multiplication table of each number.

But if the current number of the inner loop is equal to 5, then skip the current iteration and move to the next iteration of the inner loop using the `continue` statement.

Example: continue statement in nested loop

```
for i in range(1, 11):
    print('Multiplication table of', i)
    for j in range(1, 11):
        # condition to skip current iteration
        if j == 5:
            continue
        print(i * j, end=' ')
    print('')
```

Continue Statement in Outer loop

To skip the current iteration of an outside loop, use the `continue` statement inside the outer loop. Let's see the example

In the following example, The outer loop iterates the first 10 numbers, and the internal loop prints the multiplication table of each number.

But if the current number of the outer loop is even, then skip the current iteration of the outer loop and move to the next iteration.

Note: If we skip the current iteration of an outer loop, the inner loop will not be executed for that iteration because the inner loop is part of the body of an outer loop.

Example: `continue` statement in outer loop

```
for i in range(1, 11):
    # condition to skip iteration
    # Don't print multiplication table of even numbers
    if i % 2 == 0:
        continue
    print('Multiplication table of', i)
    for j in range(1, 11):
        print(i * j, end=' ')
    print('')
```

Pass Statement in Python

The `pass` is the keyword In Python, which won't do anything. Sometimes there is a situation in programming where we need to define a syntactically empty block. We can define that block with the pass keyword.

A `pass` statement is a Python null statement. When the interpreter finds a pass statement in the program, it returns no operation. Nothing happens when the `pass` statement is executed.

It is useful in a situation where we are implementing new methods or also in exception handling. It plays a role like a placeholder.

Syntax of `pass` statement:

```
for element in sequence:
    if condition:
        pass
```

Example

```
months = ['January', 'June', 'March', 'April']
for mon in months:
    pass
```

```
print(months)
```


Output

```
['January', 'June', 'March', 'April']
```