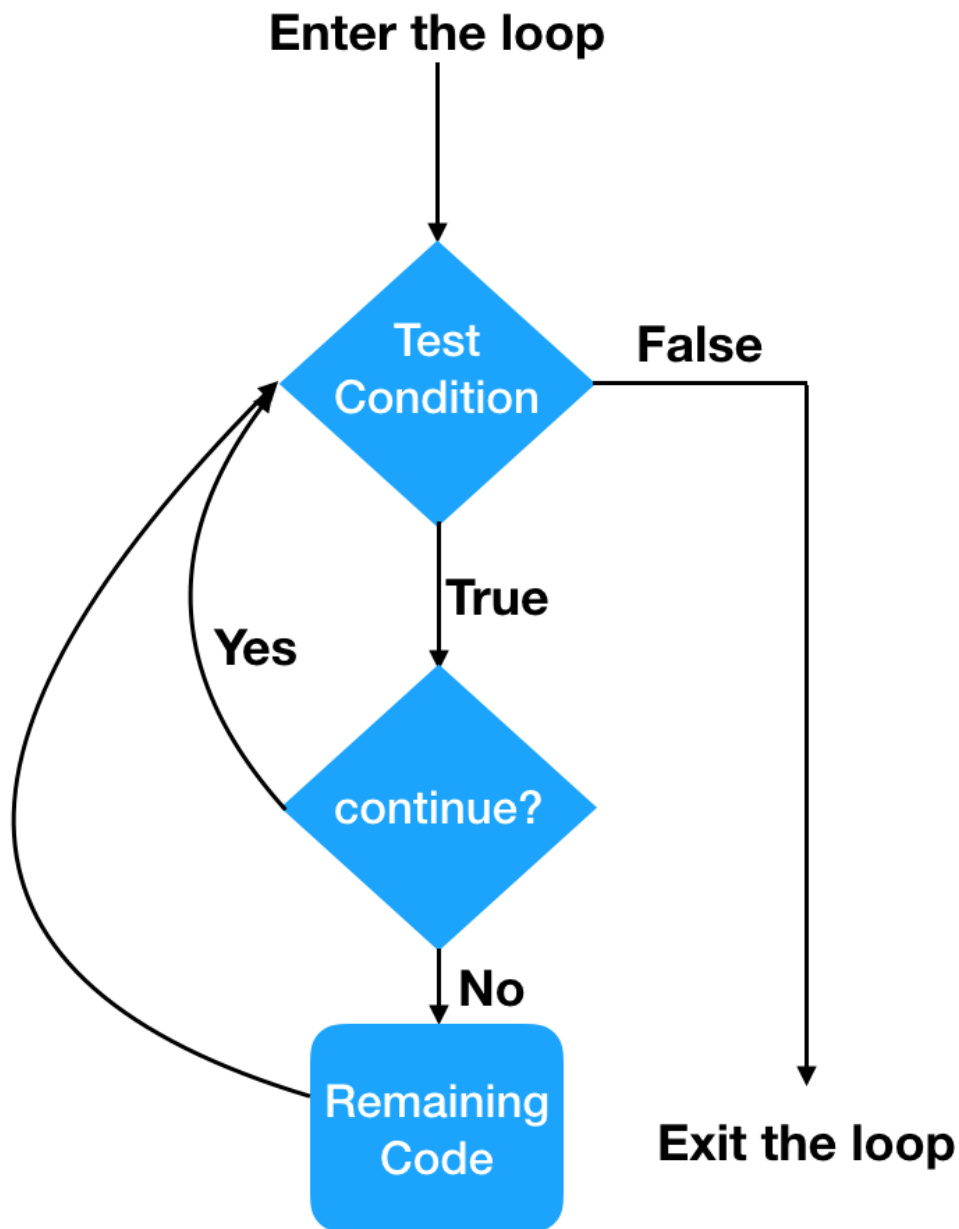


# Python continue Statement

- Python continue statement is used to skip the execution of the current iteration of the loop.
  - We can't use continue statement outside the loop, it will throw an error as "**SyntaxError: 'continue' outside loop**".
  - We can use continue statement with [for loop](#) and [while loops](#).
  - If the continue statement is present in a **nested loop**, it skips the execution of the inner loop only.
  - The "continue" is a [reserved keyword in Python](#).
  - Generally, the continue statement is used with the [if statement](#) to determine the condition to skip the current execution of the loop.
-



Python continue Statement Flow Diagram

---

## Python continue Statement Syntax

The continue statement syntax is:

```
1 | continue
```

We can't use any option, label or condition with the continue statement.

---

## Python continue Statement Examples

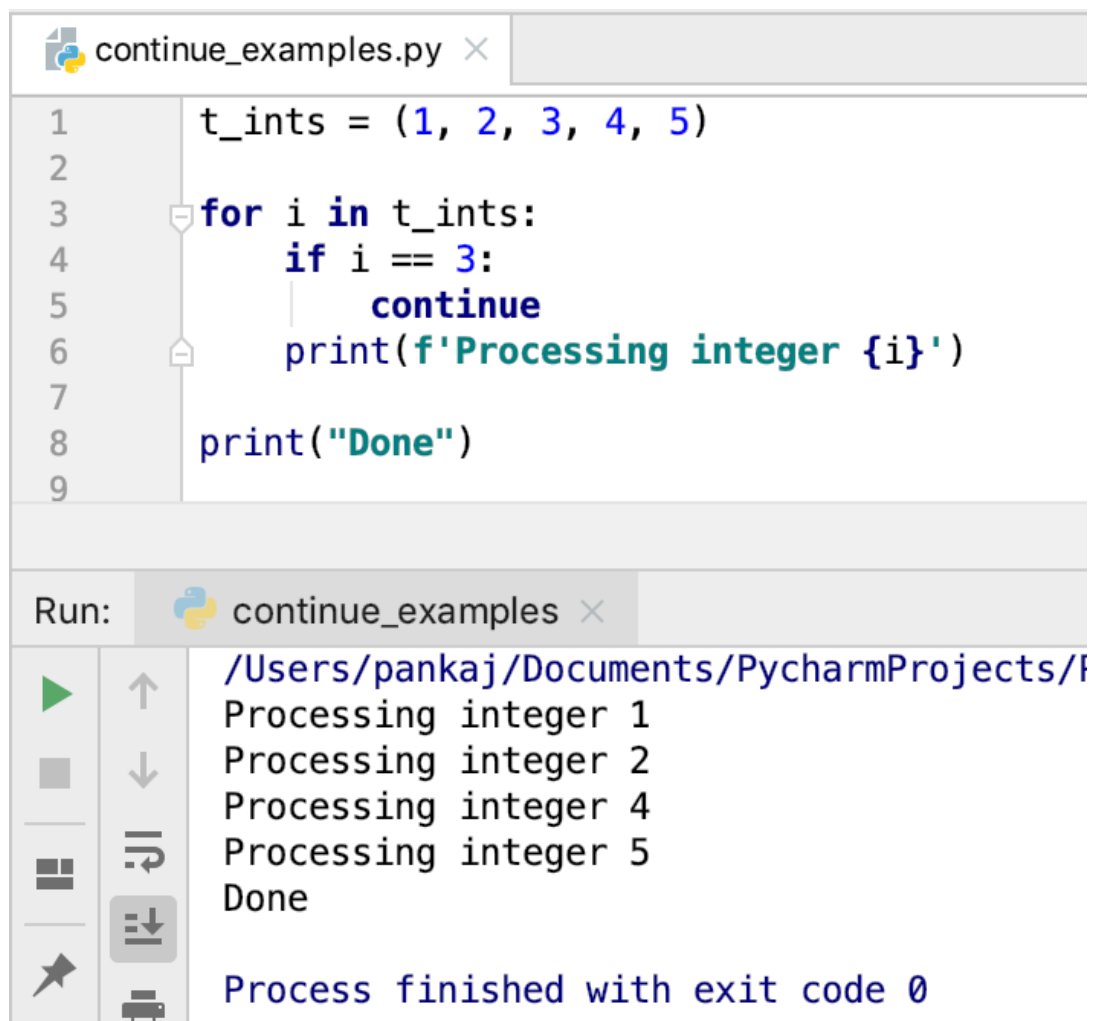
Let's look at some examples of using the continue statement in Python.

## 1. continue with for loop

Let's say we have a sequence of integers. We have to skip processing if the value is 3. We can implement this scenario using for loop and continue statement.

```
1 t_ints = (1, 2, 3, 4, 5)
2
3 for i in t_ints:
4     if i == 3:
5         continue
6     print(f'Processing integer {i}')
7
8 print("Done")
```

**Output:**

A screenshot of a Python IDE window titled 'continue\_examples.py'. The code in the editor is: 

```
1 t_ints = (1, 2, 3, 4, 5)
2
3 for i in t_ints:
4     if i == 3:
5         continue
6     print(f'Processing integer {i}')
7
8 print("Done")
```

 Below the editor is a 'Run' panel. It shows the execution path with a green arrow for the first iteration (i=1), a grey square for the second iteration (i=2), and a red arrow for the third iteration (i=3) which is skipped. The output text is: 

```
/Users/pankaj/Documents/PycharmProjects/I
Processing integer 1
Processing integer 2
Processing integer 4
Processing integer 5
Done
Process finished with exit code 0
```

Python continue Statement with for Loop

---

## 2. continue statement with the while loop

Here is a simple example of using continue statement with the while loop.

```
1 count = 10
2
3 while count > 0:
```

```

4     if count % 3 == 0:
5         count -= 1
6         continue
7     print(f'Processing Number {count}')
8     count -= 1

```

Output:

```

continue_examples.py x
9
10    count = 10
11
12    while count > 0:
13        if count % 3 == 0:
14            count -= 1
15            continue
16        print(f'Processing Number {count}')
17        count -= 1
18
Run: continue_examples x
/Users/pankaj/Documents/PycharmProjects/P
Processing Number 10
Processing Number 8
Processing Number 7
Processing Number 5
Processing Number 4
Processing Number 2
Processing Number 1
Process finished with exit code 0

```

Python continue Statement with while Loop

### 3. continue statement with a nested loop

Let's say we have a list of tuples to process. The tuple contains integers. The processing should be skipped for below conditions.

- skip the processing of tuple if its size is greater than 2.
- skip the execution if the integer is 3.

We can implement this logic with nested for loops. We will have to use two continue statements for implementing above conditions.

```

1 list_of_tuples = [(1, 2), (3, 4), (5, 6, 7)]
2
3 for t in list_of_tuples:
4     # don't process tuple with more than 2 elements
5     if len(t) > 2:
6         continue
7     for i in t:
8         # don't process if the tuple element value is 3
9         if i == 3:
10            continue
11            print(f'Processing {i}')

```

**Output:**

The screenshot shows a PyCharm IDE window titled 'continue\_examples.py'. The code in the editor is as follows:

```

18
19
20 list_of_tuples = [(1, 2), (3, 4), (5, 6, 7)]
21
22 for t in list_of_tuples:
23     # don't process tuple with more than 2 elements
24     if len(t) > 2:
25         continue
26     for i in t:
27         # don't process if the tuple element value is 3
28         if i == 3:
29             continue
30         print(f'Processing {i}')
31

```

Below the editor, the 'Run' tab shows the execution output:

```

Run: continue_examples x
/Users/pankaj/Documents/PycharmProjects/PythonTutorialP
Processing 1
Processing 2
Processing 4

Process finished with exit code 0

```

Python continue Statement with Nested Loop

## Why Python doesn't support labeled continue statement?

Many popular programming languages support a labeled continue statement. It's mostly used to skip the iteration of the outer loop in case of nested loops. However, Python doesn't support labeled continue statement.

[PEP 3136](#) was raised to add label support to continue statement. But, it was rejected because it's a very rare scenario and it will add unnecessary complexity to the language. We can always write the condition in the outer loop to skip the current execution.

# Python continue vs break vs pass

continue	break	pass
The continue statement skips only the current iteration of the loop.	The break statement terminates the loop.	The pass statement is used to write empty code blocks.
We can use continue statement only inside a loop.	We can use break statement only inside a loop.	We can use pass statement anywhere in the Python code.