Learn Python Programming

Course outline for module 3:

- The while Loop, break and continue,The for Loop

Python for Loop

In this article, you'll learn to iterate over a sequence of elements using the different variations of for loop.

The for loop in Python is used to iterate over a sequence (list, tuple, string) or other iterable objects. Iterating over a sequence is called traversal.

Syntax of for Loop

```
for val in sequence:
       Body of for
```

Here, `val` is the variable that takes the value of the item inside the sequence on each iteration.

Loop continues until we reach the last item in the sequence. The body of for loop is separated from the rest of the code using indentation.

Flowchart of for Loop

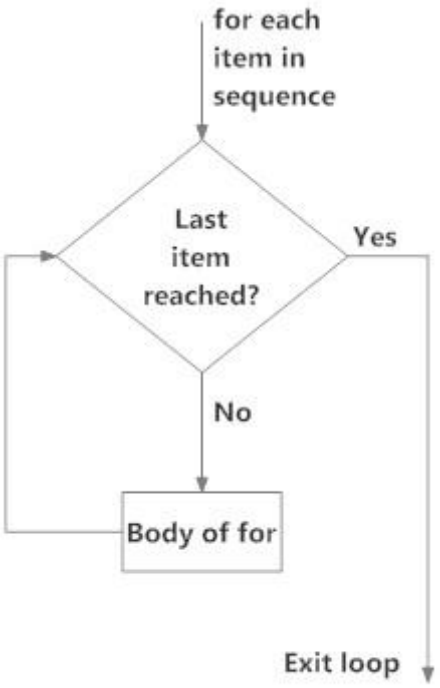


Fig: operation of for loop

Example: Python for Loop

```
# Program to find the sum of all numbers stored in a list
# List of numbers
numbers = [6, 5, 3, 8, 4, 2, 5, 4, 11]
# variable to store the sum
sum = 0
# iterate over the list
for val in numbers:
    sum = sum+val
# Output: The sum is 48
print("The sum is", sum)
```

when you run the program, the output will be:

```
The sum is 48
```

The range() function

We can generate a sequence of numbers using `range()` function. `range(10)` will generate numbers from 0 to 9 (10 numbers).

We can also define the start, stop and step size as `range(start,stop,step size)`. step size defaults to 1 if not provided.

This function does not store all the values in memory, it would be inefficient. So it remembers the start, stop, step size and generates the next number on the go.

To force this function to output all the items, we can use the function `list()`.

The following example will clarify this.

```
# Output: range(0, 10)
print(range(10))

# Output: [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
print(list(range(10)))

# Output: [2, 3, 4, 5, 6, 7]
print(list(range(2, 8)))

# Output: [2, 5, 8, 11, 14, 17]
print(list(range(2, 20, 3)))
```

We can use the range() function in for loops to iterate through a sequence of numbers. It can be combined with the `len()` function to iterate though a sequence using indexing. Here is an example.

```
# Program to iterate through a list using indexing
genre = ['pop', 'rock', 'jazz']
# iterate over the list using index
for i in range(len(genre)):
    print("I like", genre[i])
```

When you run the program, the output will be:

```
I like pop
I like rock
I like jazz
```

for loop with else

A for loop can have an optional else block as well. The `else` part is executed if the items in the sequence used in for loop exhausts.

break statement can be used to stop a for loop. In such case, the else part is ignored.

Hence, a for loop's else part runs if no break occurs.

```
digits = [0, 1, 5]
for i in digits:
    print(i)
else:
    print("No items left.")
```

When you run the program, the output will be:

```
0
1
5
No items left.
```

Here, the for loop prints items of the list until the loop exhausts. When the for loop exhausts, it executes the block of code in the `else` and prints

```
No items left.
```

Python while Loop

Loops are used in programming to repeat a specific block of code. In this article, you will learn to create a while loop in Python.

The while loop in Python is used to iterate over a block of code as long as the test expression (condition) is true.

We generally use this loop when we don't know beforehand, the number of times to iterate.

Syntax of while Loop in Python

```
while test_expression:

    Body of while
```

In while loop, test expression is checked first. The body of the loop is entered only if the `test_expression` evaluates to `True`. After one iteration, the test expression is checked again. This process continues until the `test_expression` evaluates to `False`.

In Python, the body of the while loop is determined through indentation.

Body starts with indentation and the first un indented line marks the end.

Python interprets any non-zero value as `True`. `None` and `0` are interpreted as `False`.

Flowchart of while Loop

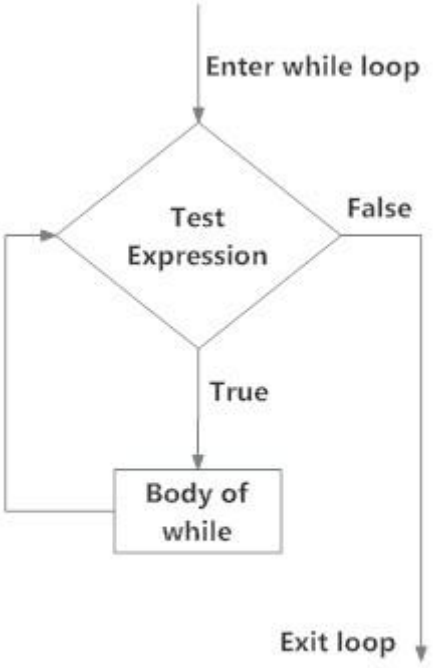


Fig: operation of while loop

Example: Python while Loop

```
# Program to add natural
# numbers upto
# sum = 1+2+3+...+n
# To take input from the user,
n = int(input("Enter n: "))
n = 10
# initialize sum and counter
sum = 0
i = 1
while i <= n:
    sum = sum + i
    i = i+1    # update counter
# print the sum
print("The sum is", sum)
```

When you run the program, the output will be:

```
Enter n: 10

The sum is 55
```

In the above program, the test expression will be `True` as long as our counter variable `i` is less than or equal to `n` (10 in our program).

We need to increase the value of counter variable in the body of the loop. This is very important (and mostly forgotten). Failing to do so will result in an infinite loop (never ending loop).

Finally the result is displayed.

while loop with else

Same as that of for loop, we can have an optional `else` block with while loop as well.

The `else` part is executed if the condition in the while loop evaluates to `False`. The while loop can be terminated with a break statement.

In such case, the `else` part is ignored. Hence, a while loop's `else` part runs if no break occurs and the condition is false.

Here is an example to illustrate this.

```
# Example to illustrate
# the use of else statement
# with the while loop
counter = 0
while counter < 3:
    print("Inside loop")
    counter = counter + 1
else:
    print("Inside else")
```

Output

```
Inside loop

Inside loop
```

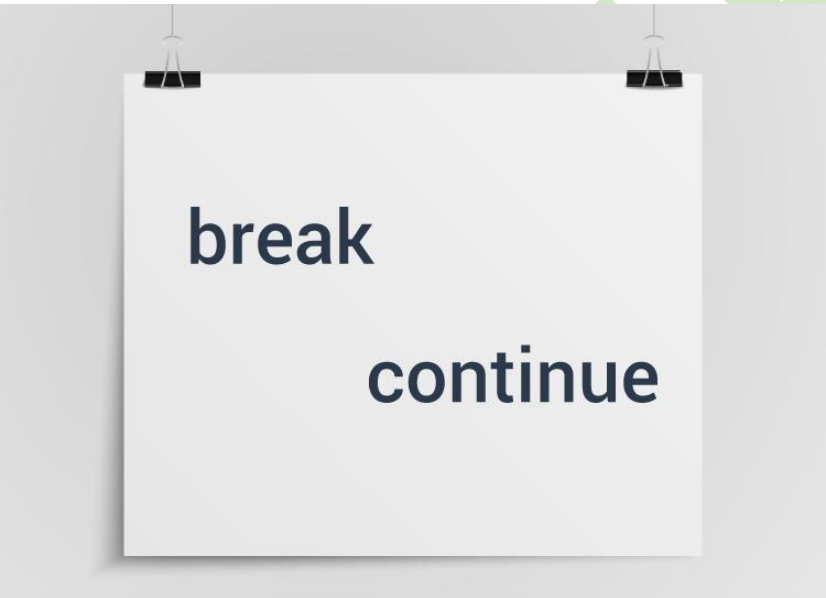
```
Inside loop

Inside else
```

Here, we use a counter variable to print the string `Inside loop` three times.

On the forth iteration, the condition in while becomes `False`. Hence, the `else` part is executed.

Python break and continue

In this article, you will learn to use break and continue statements to alter the flow of a loop.



In Python, break and continue statements can alter the flow of a normal loop.

Loops iterate over a block of code until test expression is false, but sometimes we wish to terminate the current iteration or even the whole loop without checking test expression.

The break and continue statements are used in these cases.

Python break statement

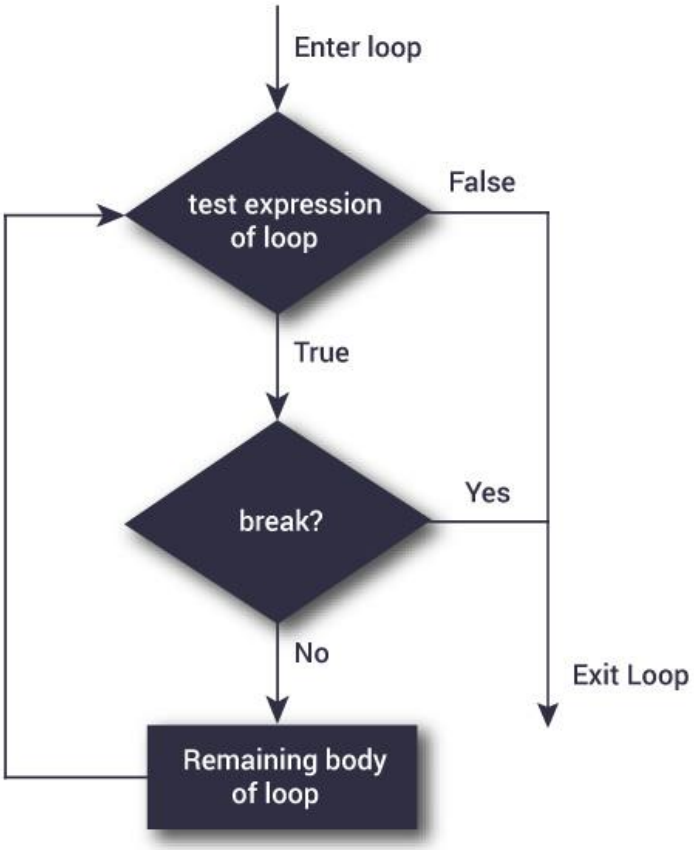
The break statement terminates the loop containing it. Control of the program flows to the statement immediately after the body of the loop.

If break statement is inside a nested loop (loop inside another loop), break will terminate the innermost loop.

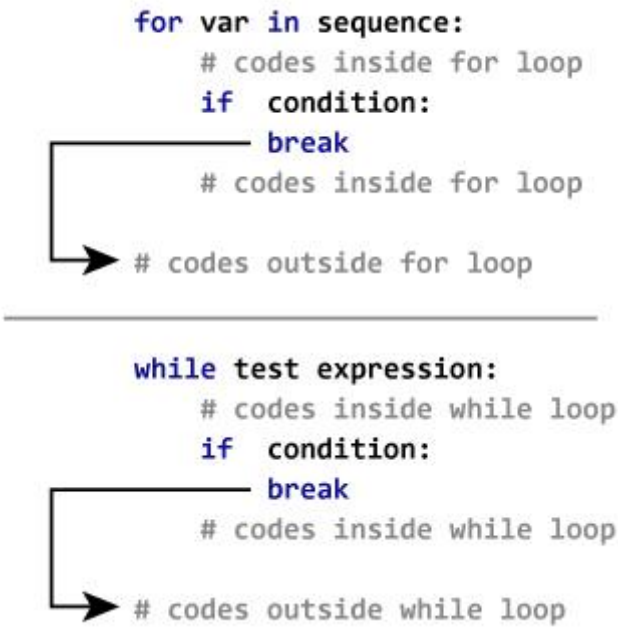
Syntax of break

```
break
```

Flowchart of break



The working of break statement in for loop and while loop is shown below.



Example: Python break

```
# Use of break statement inside loop
for val in "string":
    if val == "i":
        break
    print(val)
print("The end")
```

Output

```
s

t

r

The end
```

In this program, we iterate through the `"string"` sequence. We check if the letter is `"i"`, upon which we break from the loop. Hence, we see in our output that all the letters up till `"i"` gets printed. After that, the loop terminates.

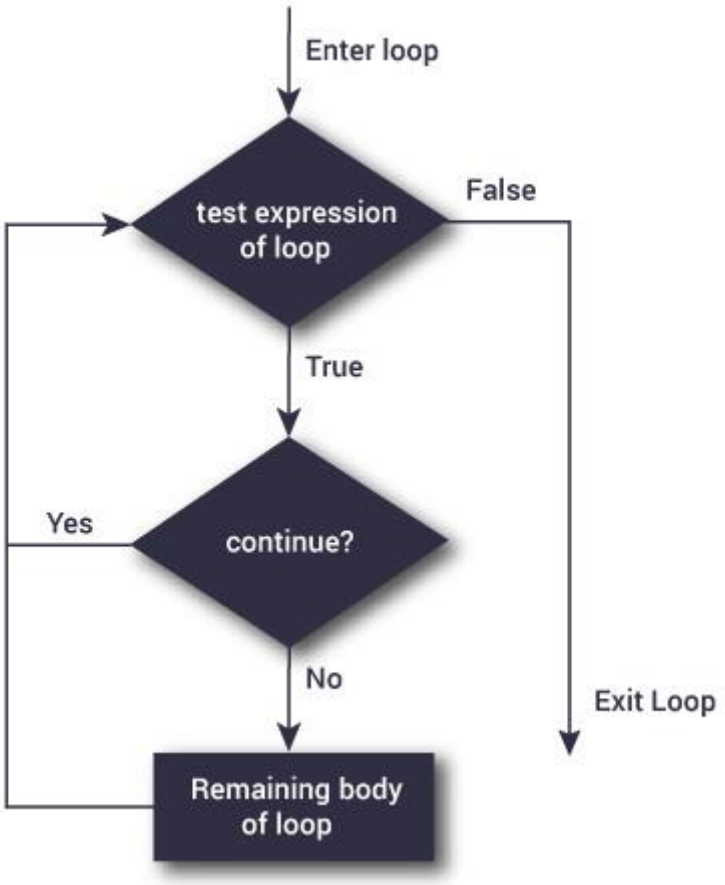
Python continue statement

The continue statement is used to skip the rest of the code inside a loop for the current iteration only. Loop does not terminate but continues on with the next iteration.

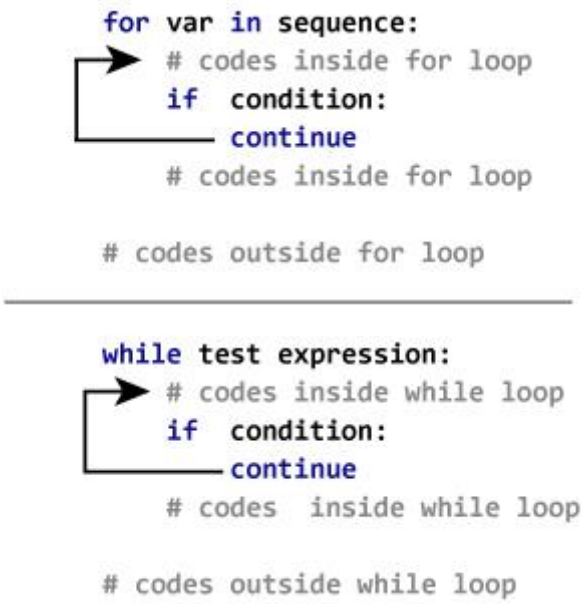
Syntax of Continue

```
continue
```

Flowchart of continue



The working of continue statement in for and while loop is shown below.



Example: Python continue

```
# Program to show the use of continue statement inside loops
for val in "string":
    if val == "i":
        continue
    print(val)
print("The end")
```

Output

```
s

t

r

n

g

The end
```

This program is same as the above example except the break statement has been replaced with continue.

We continue with the loop, if the string is `"i"`, not executing the rest of the block. Hence, we see in our output that all the letters except `"i"` gets printed.

Python Looping Techniques

In this article, you'll learn to control the execution of a loop by using loop control statements like break and continue.

Python programming offers two kinds of loop, the for loop and the while loop. Using these loops along with loop control statements like break and continue, we can create various forms of loop.

The infinite loop

We can create an infinite loop using while statement. If the condition of while loop is always `True`, we get an infinite loop.

Example #1: Infinite loop using while

```
# An example of infinite loop
# press Ctrl + c to exit from the loop

while True:
    num = int(input("Enter an integer: "))
    print("The double of",num,"is",2 * num)
```

Output

```
Enter an integer: 3

The double of 3 is 6

Enter an integer: 5

The double of 5 is 10

Enter an integer: 6

The double of 6 is 12

Enter an integer:

Traceback (most recent call last):
```

Loop with condition at the top

This is a normal while loop without break statements. The condition of the while loop is at the top and the loop terminates when this condition is `False`.

Flowchart of Loop With Condition at Top

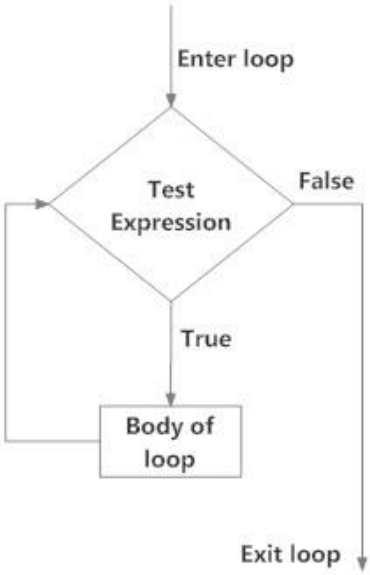


Fig: loop with condition at top

Example #2: Loop with condition at the top

```
# Program to illustrate a loop with condition at the top
# Try different numbers
n = 10
# Uncomment to get user input
#n = int(input("Enter n: "))
# initialize sum and counter
sum = 0
i = 1
while i <= n:
  sum = sum + i
  i = i+1    # update counter
# print the sum
print("The sum is",sum)
```

When you run the program, the output will be:

```
The sum is 55
```

Loop with condition in the middle

This kind of loop can be implemented using an infinite loop along with a conditional break in between the body of the loop.

Flowchart of Loop with Condition in Middle

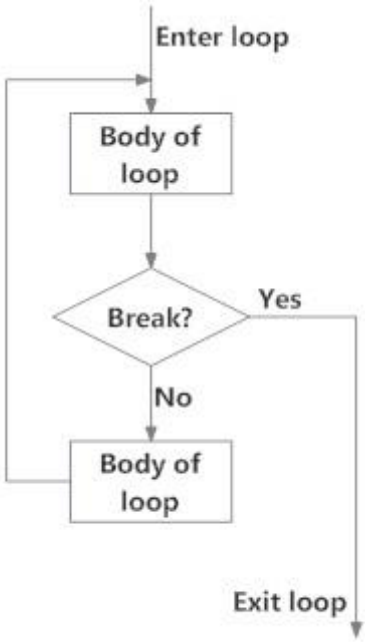


Fig: loop with condition in middle

Example #3: Loop with condition in the middle

```
# Program to illustrate a loop with condition in the middle.
# Take input from the user untill a vowel is entered

vowels = "aeiouAEIOU"

# infinite loop
while True:
   v = input("Enter a vowel: ")
   # condition in the middle
   if v in vowels:
       break
   print("That is not a vowel. Try again!")

print("Thank you!")
```

Output

```
Enter a vowel: r

That is not a vowel. Try again!

Enter a vowel: 6

That is not a vowel. Try again!

Enter a vowel: ,

That is not a vowel. Try again!

Enter a vowel: u

Thank you!
```

Loop with condition at the bottom

This kind of loop ensures that the body of the loop is executed at least once. It can be implemented using an infinite loop along with a conditional break at the end. This is similar to the do...while loop in C.

Flowchart of Loop with Condition at Bottom

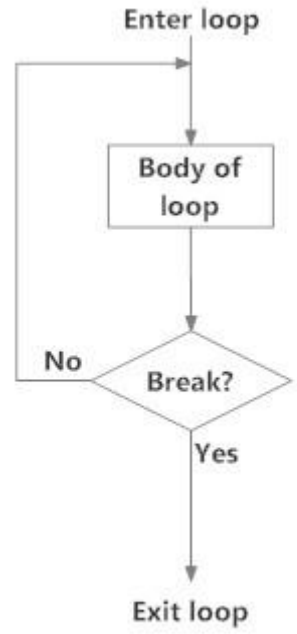


Fig: loop with condition at bottom

Example #4: Loop with condition at the bottom

```
# Python program to illustrate a loop with condition at the bottom
# Roll a dice untill user chooses to exit

# import random module
import random

while True:
  input("Press enter to roll the dice")

  # get a number between 1 to 6
  num = random.randint(1,6)
  print("You got",num)
  option = input("Roll again?(y/n) ")

  # condition
  if option == 'n':
   break
```

Output

```
Press enter to roll the dice

You got 1

Roll again?(y/n) y

Press enter to roll the dice

You got 5

Roll again?(y/n) n
```