# Python Loop with Syntax and Examples

data-flair.training/blogs/python-loops

DataFlair Team December 22, 2017

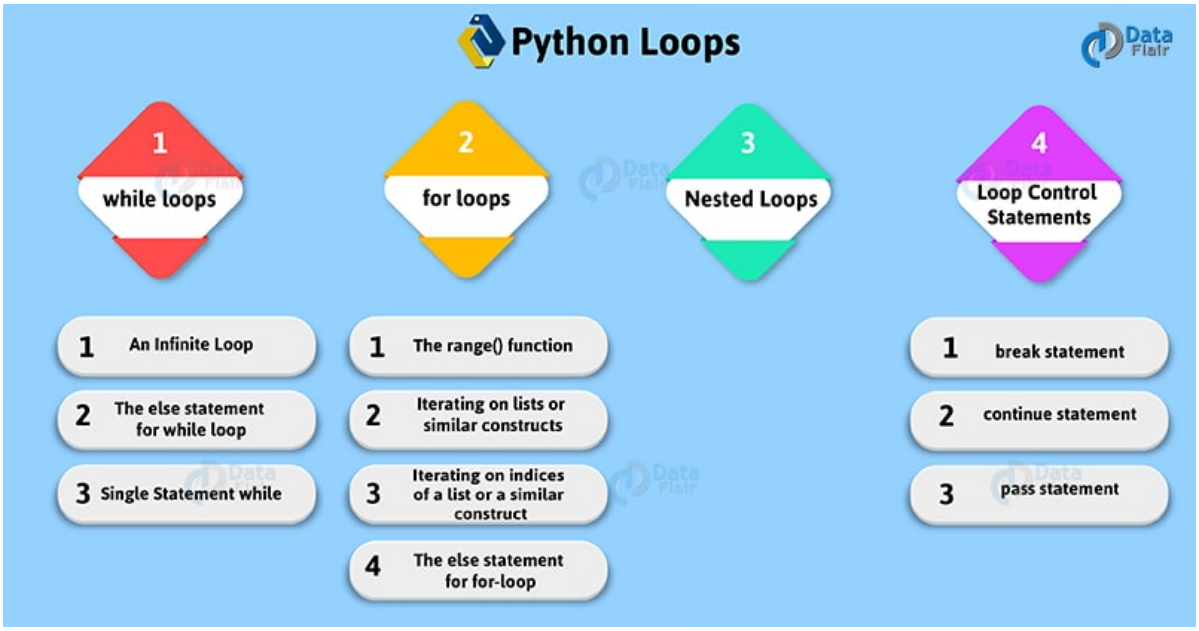
Contents



## 1. Different Python Loops

In this tutorial, we will learn about different types of Python Loops, and various conditions that we can use to control Python loops. Python provides support for two kinds of loop- ‘for’ and ‘while’.

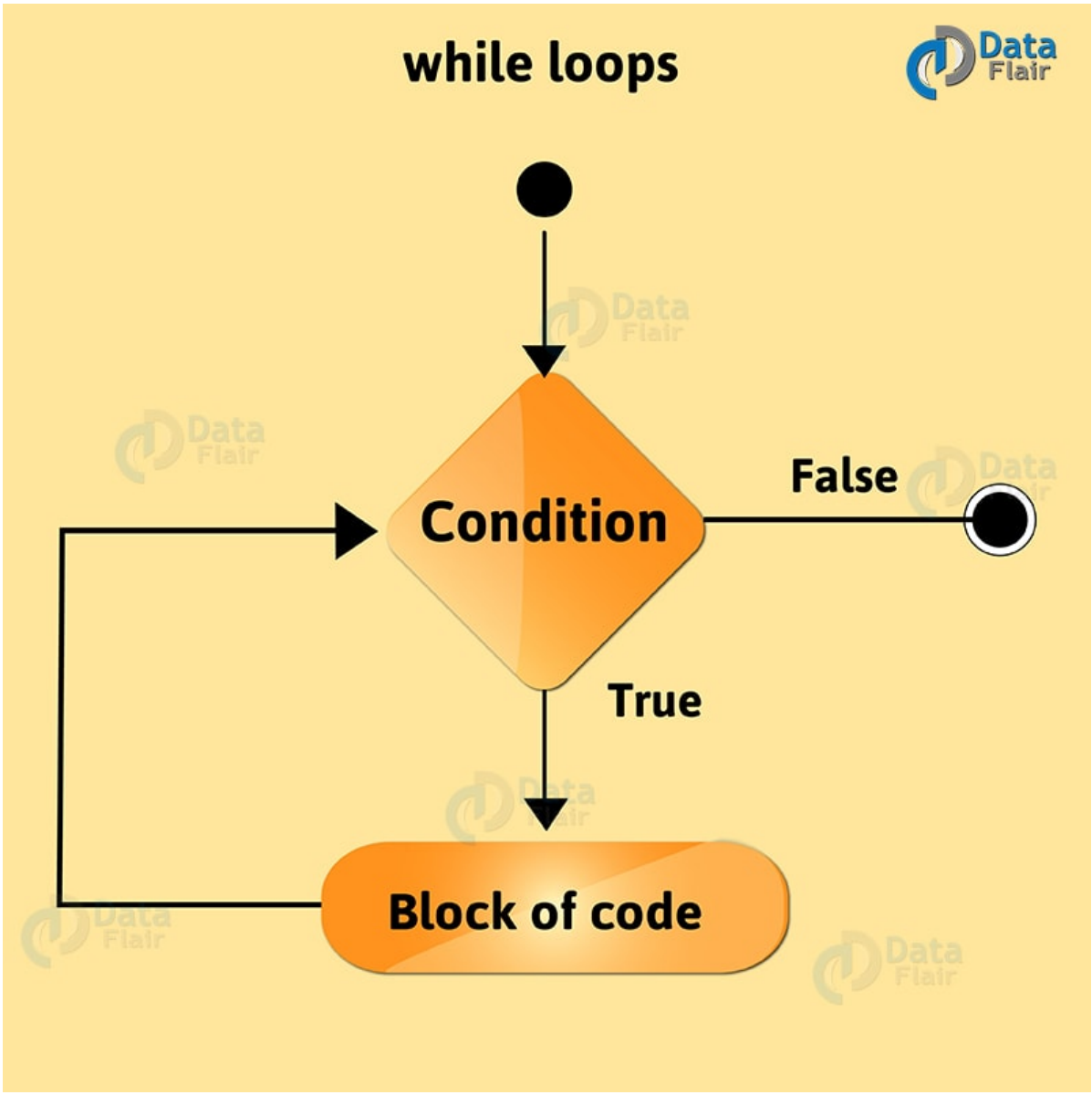
When you want some statements to execute a hundred times, you don’t repeat them 100 times. Think of when you want to print numbers 1 to 99. Or that you want to say Hello to 99 friends. In such a case, you can use loops in python.




Python Loops

## 2. Python While Loop

A while loop in python iterates till its condition becomes False. In other words, it executes the statements under itself while the condition it takes is True.



Python While loop

When the program control reaches the while loop, the condition is checked. If the condition is true, the block of code under it is executed. Remember to indent all statements under the loop equally. After that, the condition is checked again. This continues until the condition becomes false. Then, the first statement, if any, after the loop is executed.

1. >>> a=3
2. >>>while(a>0):
3. print(a)
4. a-=1

3

2

1

This loop prints numbers from 3 to 1. In Python, a—wouldn't work. We use a-=1 for the same.

## a. An Infinite Loop

Be careful while using a while loop. Because if you forget to increment the countervariable in python, or write flawed logic, the condition may never become false. In such a case, the loop will run infinitely, and the conditions after the loop will starve. To stop execution, press Ctrl+C. However, an infinite loop may actually be useful. This in cases when a semaphore is needed, or for client/server programming. A semaphore is a variable used solely for synchronization in accessing shared resources.

**Learn: Python Function with Syntax and Examples**

## b. The else statement for while loop

A while loop may have an else statement after it. When the condition becomes false, the block under the else statement is executed. However, it doesn't execute if you break out of the loop or if an exception is raised.

```
1. >>> a=3
2. >>>while(a>0):
3. print(a)
4. a-=1
5. else:
6. print("Reached 0")
```

3

2

1

Reached 0

In the following code, we put a break statement in the body of the while loop for a==1. So, when that happens, the statement in the else block is not executed.

```
1. >>> a=3
2. >>>while(a>0):
3. print(a)
4. a-=1
5. if(a==1):
6. break;
7. else:
8. print("Reached 0")
```

3

2

## c. Single Statement while

Like an if statement, if we have only one statement in while's body, we can write it all in one line.

```
1. >>> a=3
2. >>>while a>0: print(a); a-=1;
```

3

2

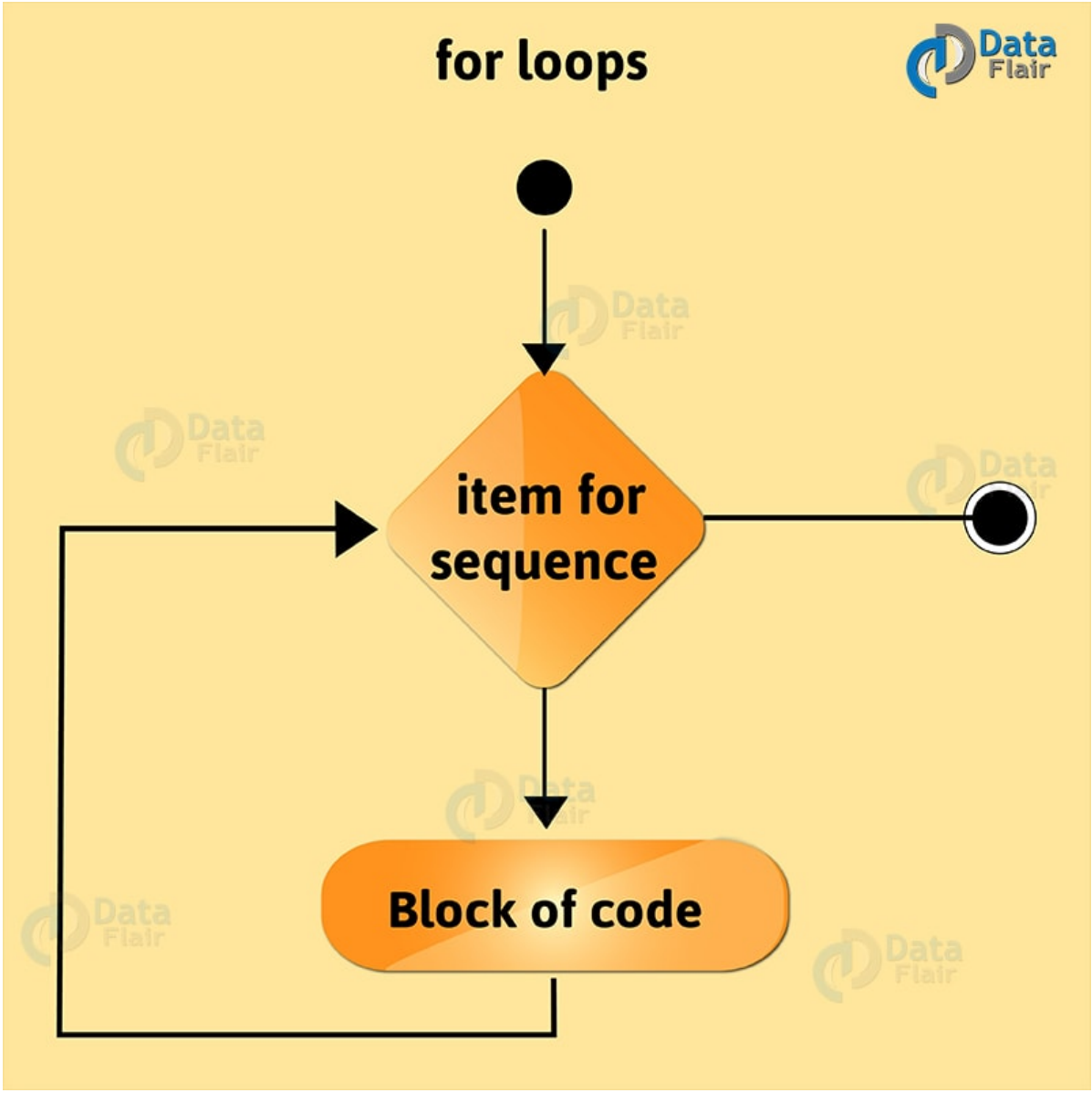
1

You can see that there were two statements in while's body, but we used semicolons to separate them. Without the second statement, it would form an infinite loop.

**Learn: Methods vs Functions in Python**

## 3. Python For Loop

Python for loop can iterate over a sequence of items. The structure of a for loop in Python is different than that in C++ or Java. That is, for(int i=0;i<n;i++) won't work here. In Python, we use the 'in' keyword. Lets see a Python for loop Example



Python For Loop

```
1. >>>for a in range(3):
2. print(a)
```

0

1

2

If we wanted to print 1 to 3, we could write the following code.

```
1. >>>for a in range(3):
2. print(a+1)
```

1

2

3

## a. The range() function

This function yields a sequence of numbers. When called with one argument, say n, it creates a sequence of numbers from 0 to n-1.

>>>list(range(10))

[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]

We use the list function to convert the range object into a list object.

Calling it with two arguments creates a sequence of numbers from the first to the second.

>>>list(range(2,7))

[2, 3, 4, 5, 6]

You can also pass three arguments. The third argument is the interval.

>>>list(range(2,12,2))

[2, 4, 6, 8, 10]

Remember, the interval can also be negative.

>>>list(range(12,2,-2))

[12, 10, 8, 6, 4]

However, the following codes will return an empty list.

>>>list(range(12,2))

[]

>>>list(range(2,12,-2))

[]

>>>list(range(12,2,2))

[]

## b. Iterating on lists or similar constructs

You aren’t bound to use the range() function, though. You can use the loop to iterate on a list or a similar construct.

1. >>>for a in [1,2,3]:
2. print(a)

1

2

3

```
1. >>>for i in {2,3,3,4}:
2. print(i)
```

2

3

4

You can also iterate on a string.

```
1. >>>for i in 'wisdom':
2. print(i)
```

w

i

s

d

o

m

## c. Iterating on indices of a list or a similar construct

The len() function returns the length of the list. When you apply the range() function on that, it returns the indices of the list on a range object. You can iterate on that.

```
1. >>>list=['Romanian','Spanish','Gujarati']
2. >>>for i in range(len(list)):
3. print(list[i])
```

Romanian

Spanish

Gujarati

**Learn: Data Structures in Python – Lists, Tuples, Sets, Dictionaries**

## d. The else statement for for-loop

Like a while loop, a for-loop may also have an else statement after it. When the loop is exhausted, the block under the else statement executes.

```
1. >>>for i in range(10):
2. print(i)
3. else:
4. print("Reached else")
```

0

1

2

3

4

5

6

7

8

9

Reached else

Like in the while loop, it doesn't execute if you break out of the loop or if an exception is raised.

```
1. >>>for i in range(10):
2. print(i)
3. if(i==7):
4. break
5. else:
6. print("Reached else")
```

0

1

2

3

4

5

6

7

## 4. Nested for Loops Python

You can also nest a loop inside another. You can put a for loop inside a while, or a while inside a for, or a for inside a for, or a while inside a while. Or you can put a loop inside a loop inside a loop. You can go as far as you want.

```
1. >>>for i in range(1,6):
2. for j in range(i):
3. print("*",end=' ')
4. print("\n",end='')
```

```
*

* *

* * *

* * * *

* * * * *
```

Let’s look at some nested while loops to print the same pattern.

```
1. >>> i=6
2. >>>while(i>0):
3. j=6
4. while(j>i):
5. print("*",end=' ')
6. j-=1
7. i-=1
8. print("\n",end='')
```

```
*

* *

* * *

* * * *

* * * * *
```

## 5. Loop Control Statements in Python

Sometimes, you may want to break out of normal execution in a loop. For this, we have three keywords in Python- break, continue, and pass.

### a. break statement

When you put a break statement in the body of a loop, the loop stops executing, and control shifts to the first statement outside it. You can put it in a for or while loop.

```
1. >>>for i in 'break':
2. print(i)
3. if i=='a': break;
```

```
b
```

r

e

a

## Learn: Python Dictionaries with Methods, Functions and Dictionary Operations

### b. continue statement

When the program control reaches the continue statement, it skips the statements after ‘continue’. It then shifts to the next item in the sequence and executes the block of code for it. You can use it with both for and while loops.

```
1. >>> i=0
2. >>>while(i<8):
3. i+=1
4. if(i==6): continue
5. print(i)
```

1

2

3

4

5

7

8

If here, the iteration i+=1 succeeds the if condition, it prints to 5 and gets stuck in an infinite loop. You can break out of an infinite loop by pressing Ctrl+C.

```
1. >>> i=0
2. >>>while(i<8):
3. if(i==6): continue
4. print(i)
5. i+=1
```

0

1

2

3

4

5

Traceback (most recent call last):

1. File "<pyshell#14>", line 1, in <module>
2. while(i<8):
3. KeyboardInterrupt

### c. pass statement

In Python, we use the pass statement to implement stubs. When we need a particular loop, class, or function in our program, but don't know what goes in it, we place the pass statement in it. It is a null statement. The interpreter does not ignore it, but it performs a no-operation (NOP).

1. for i in 'selfhelp':
2. pass
3. print(i)

p

To run this code, save it in a .py file, and press F5. It causes a syntax error in the shell.

**Learn: Python Operators with Syntax and Examples**

## 6. Conclusion

In this tutorial on Python Loops, we learnt about while and for loops in Python. We also learnt how to nest loops, and use the range() function to iterate through a sequence of items. Lastly, we learnt about break, continue, and pass statements to control loops. So, try out your own combinations in the shell, and don't forget to leave your feedback in the comments.

To learn more about it, refer Best Python books.

### Related Posts