**Python for Loop**

In Python, we use a for loop to iterate over sequences such as [lists](https://www.programiz.com/python-programming/list), [strings](https://www.programiz.com/python-programming/string), [dictionaries](https://www.programiz.com/python-programming/dictionary), etc. For example,

languages = ['Swift', 'Python', 'Go']

# access elements of the list one by one

for lang in languages:

print(lang)

[Run Code](https://www.programiz.com/python-programming/online-compiler)

**Output**

Swift

Python

Go

In the above example, we have created a list named languages. Since the list has three elements, the loop iterates **3** times.

The value of lang is

* Swift in the first iteration.
* Python in the second iteration.
* Go in the third iteration.

**for loop Syntax**

for val in sequence:

# run this code

The for loop iterates over the elements of **sequence** in order, and in each iteration, the body of the loop is executed.

The loop ends after the body of the loop is executed for the last item.

**Indentation in Loop**

In Python, we use indentation (spaces at the beginning of a line) to define a block of code, such as the body of a loop. For example,

languages = ['Swift', 'Python', 'Go']

# start of the loop

for lang in languages:

print(lang)

print('-----')

# end of the for loop

print('Last statement')

[Run Code](https://www.programiz.com/python-programming/online-compiler)

**Output**

Swift

-----

Python

-----

Go

-----

Last statement

Here, print('Last statement') is outside the body of the loop. Therefore, this statement is executed only once at the end.

**Example: Loop Through a String**

If we iterate through a string, we get individual characters of the string one by one.

language = 'Python'

# iterate over each character in language

for x in language:

print(x)

[Run Code](https://www.programiz.com/python-programming/online-compiler)

**Output**

P

y

t

h

o

n

Here, we have printed each character of the string language using a for loop.

**for Loop with Python range()**

In Python, the [range()](https://www.programiz.com/python-programming/methods/built-in/range) function returns a sequence of numbers. For example,

# generate numbers from 0 to 3

values = range(0, 4)

Here, range(0, 4) returns a sequence of **0**, **1**, **2** ,and **3**.

Since the range() function returns a sequence of numbers, we can iterate over it using a for loop. For example,

# iterate from i = 0 to i = 3

for i in range(0, 4):

print(i)

[Run Code](https://www.programiz.com/python-programming/online-compiler)

**Output**

0

1

2

3

Here, we used the for loop to iterate over a range from **0** to **3**.

This is how the above program works.

|  |  |  |  |
| --- | --- | --- | --- |
| Iteration | Value of i | print(i) | Last item in sequence? |
| 1st | 0 | Prints 0 | No The body of the loop executes. |
| 2nd | 1 | Prints 1 | No The body of the loop executes. |
| 3rd | 2 | Prints 2 | No The body of the loop executes. |
| 4th | 3 | Prints 3 | Yes The body of the loop executes and the loop terminates. |

**break and continue Statement**

The break and continue statements are used to alter the flow of loops.

**The break Statement**

The break statement terminates the for loop immediately before it loops through all the items. For example,

languages = ['Swift', 'Python', 'Go', 'C++']

for lang in languages:

if lang == 'Go':

break

print(lang)

[Run Code](https://www.programiz.com/python-programming/online-compiler)

**Output**

Swift

Python

Here, when lang is equal to 'Go', the break statement inside the if condition executes which terminates the loop immediately. This is why Go and C++ are not printed.

**The continue Statement**

The continue statement skips the current iteration of the loop and continues with the next iteration. For example,

languages = ['Swift', 'Python', 'Go', 'C++']

for lang in languages:

if lang == 'Go':

continue

print(lang)

[Run Code](https://www.programiz.com/python-programming/online-compiler)

**Output**

Swift

Python

C++

Here, when lang is equal to 'Go', the continue statement executes, which skips the remaining code inside the loop for that iteration.

However, the loop continues to the next iteration. This is why C++ is displayed in the output.

Visit [Python break and continue](https://www.programiz.com/python-programming/break-continue) article to learn more.

**Nested for loops**

A loop can also contain another loop inside it. These loops are called nested loops.

In a nested loop, the inner loop is executed once for each iteration of the outer loop.

# outer loop

attributes = ['Electric', 'Fast']

cars = ['Tesla', 'Porsche', 'Mercedes']

for attribute in attributes:

for car in cars:

print(attribute, car)

# this statement is outside the inner loop

print("-----")

[Run Code](https://www.programiz.com/python-programming/online-compiler)

**Output**

Electric Tesla

Electric Porsche

Electric Mercedes

-----

Fast Tesla

Fast Porsche

Fast Mercedes

-----

**Using for loop without accessing sequence items**

If we don't intend to use items of sequence inside the body of a loop, it is clearer to use the \_ (underscore) as the loop variable. For example,

# iterate from i = 0 to 3

for \_ in range(0, 4:

print('Hi')

[Run Code](https://www.programiz.com/python-programming/online-compiler)

**Output**

0

1

2

3

Here, the loop runs four times. In each iteration, we have displayed Hi. Since we are not using the items of the sequence(**0**, **1**, **2** and **4**) in the loop body, it is better to use \_ as the loop variable.

**Also read:** [Python while loop](https://www.programiz.com/python-programming/while-loop)

Before we wrap up, let’s put your knowledge of Python for loop to the test! Can you solve the following challenge?

Challenge:

Write a function to calculate the factorial of a number.

* The factorial of a non-negative integer n is the product of all positive integers less than or equal to n.
* For example, if n is **5**, the return value should be **120** because 1\*2\*3\*4\*5 is **120**.