Website: https://www.programiz.com/python-programming/function

A function is a block of code that performs a specific task.

Suppose, you need to create a program to create a circle and color it. You can create two functions to solve this problem:

* create a circle function
* create a color function

Dividing a complex problem into smaller chunks makes our program easy to understand and reuse.

**Types of function**

There are two types of function in Python programming:

* **Standard library functions** - These are built-in functions in Python that are available to use.
* **User-defined functions** - We can create our own functions based on our requirements.

**Python Function Declaration**

The syntax to declare a function is:

def function\_name(arguments):

# function body

return

Here,

* def - keyword used to declare a function
* function\_name - any name given to the function
* arguments - any value passed to function
* return (optional) - returns value from a function

Let's see an example,

def greet():

print('Hello World!')

Here, we have created a function named greet(). It simply prints the text Hello World!.

This function doesn't have any arguments and doesn't return any values. We will learn about arguments and return statements later in this tutorial.

**Calling a Function in Python**

In the above example, we have declared a function named greet().

def greet():

print('Hello World!')

Now, to use this function, we need to call it.

Here's how we can call the greet() function in Python.

# call the function

greet()

**Example: Python Function**

def greet():

print('Hello World!')

# call the function

greet()

print('Outside function')

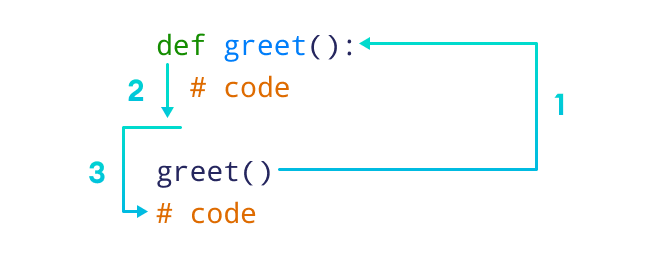
[Run Code](https://www.programiz.com/python-programming/online-compiler)

**Output**

Hello World!

Outside function

In the above example, we have created a function named greet(). Here's how the program works:

Working of Python Function

Here,

* When the function is called, the control of the program goes to the function definition.
* All codes inside the function are executed.
* The control of the program jumps to the next statement after the function call.

**Python Function Arguments**

As mentioned earlier, a function can also have arguments. An argument is a value that is accepted by a function. For example,

# function with two arguments

def add\_numbers(num1, num2):

sum = num1 + num2

print('Sum: ',sum)

# function with no argument

def add\_numbers():

# code

If we create a function with arguments, we need to pass the corresponding values while calling them. For example,

# function call with two values

add\_numbers(5, 4)

# function call with no value

add\_numbers()

Here, add\_numbers(5, 4) specifies that arguments num1 and num2 will get values **5** and **4** respectively.

**Example 1: Python Function Arguments**

# function with two arguments

def add\_numbers(num1, num2):

sum = num1 + num2

print("Sum: ",sum)

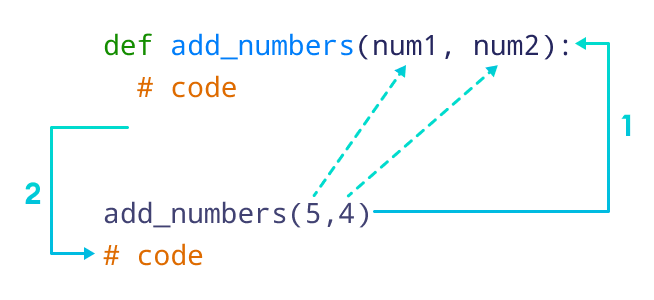
# function call with two values

add\_numbers(5, 4)

# Output: Sum: 9

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

In the above example, we have created a function named add\_numbers() with arguments: num1 and num2.

Python Function with Arguments

We can also call the function by mentioning the argument name as:

add\_numbers(num1 = 5, num2 = 4)

In Python, we call it Keyword Argument (or named argument). The code above is equivalent to

add\_numbers(5, 4)

**The return Statement in Python**

A Python function may or may not return a value. If we want our function to return some value to a function call, we use the return statement. For example,

def add\_numbers():

...

return sum

Here, we are returning the variable sum to the function call.

**Note:** The return statement also denotes that the function has ended. Any code after return is not executed.

**Example 2: Function return Type**

# function definition

def find\_square(num):

result = num \* num

return result

# function call

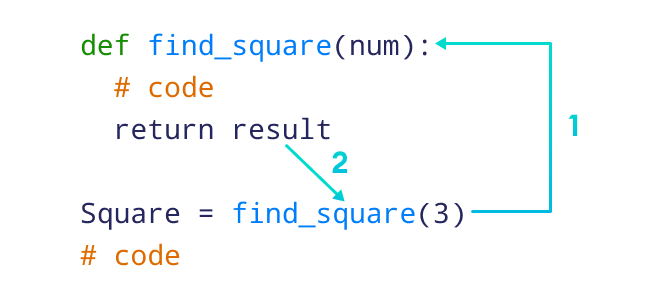
square = find\_square(3)

print('Square:',square)

# Output: Square: 9

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

In the above example, we have created a function named find\_square(). The function accepts a number and returns the square of the number.

Working of functions in Python

**Example 3: Add Two Numbers**

# function that adds two numbers

def add\_numbers(num1, num2):

sum = num1 + num2

return sum

# calling function with two values

result = add\_numbers(5, 4)

print('Sum: ', result)

# Output: Sum: 9

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

**Python Library Functions**

In Python, standard library functions are the built-in functions that can be used directly in our program. For example,

* print() - prints the string inside the quotation marks
* sqrt() - returns the square root of a number
* pow() - returns the power of a number

These library functions are defined inside the module. And, to use them we must include the module inside our program.

For example, sqrt() is defined inside the math module.

**Example 4: Python Library Function**

import math

# sqrt computes the square root

square\_root = math.sqrt(4)

print("Square Root of 4 is",square\_root)

# pow() comptes the power

power = pow(2, 3)

print("2 to the power 3 is",power)

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

**Output**

Square Root of 4 is 2.0

2 to the power 3 is 8

In the above example, we have used

* math.sqrt(4) - to compute the square root of **4**
* pow(2, 3) - computes the power of a number i.e. **23**

Here, notice the statement,

import math

Since sqrt() is defined inside the math module, we need to include it in our program.

**Benefits of Using Functions**

**1. Code Reusable** - We can use the same function multiple times in our program which makes our code reusable. For example,

# function definition

def get\_square(num):

return num \* num

for i in [1,2,3]:

# function call

result = get\_square(i)

print('Square of',i, '=',result)

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

**Output**

Square of 1 = 1

Square of 2 = 4

Square of 3 = 9

In the above example, we have created the function named get\_square() to calculate the square of a number. Here, the function is used to calculate the square of numbers from **1** to **3**.

Hence, the same method is used again and again.

**2. Code Readability** - Functions help us break our code into chunks to make our program readable and easy to understand.

# Python Function Arguments

In this tutorial, we will learn about function arguments in Python with the help of examples.

In computer programming, a function is a value that is accepted by a function.

Before we learn about function arguments, make sure to know about [Python Functions](https://www.programiz.com/python-programming/function).

**Example 1: Python Function Arguments**

def add\_numbers(a, b):

sum = a + b

print('Sum:', sum)

add\_numbers(2, 3)

# Output: Sum: 5

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

In the above example, the function add\_numbers() takes two parameters: a and b. Notice the line,

add\_numbers(2, 3)

Here, add\_numbers(2, 3) specifies that parameters a and b will get values **2** and **3** respectively.

**Function Argument with Default Values**

In Python, we can provide default values to function arguments.

We use the = operator to provide default values. For example,

def add\_numbers( a = 7, b = 8):

sum = a + b

print('Sum:', sum)

# function call with two arguments

add\_numbers(2, 3)

# function call with one argument

add\_numbers(a = 2)

# function call with no arguments

add\_numbers()

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

**Output**

Sum: 5

Sum: 10

Sum: 15

In the above example, notice the function definition

def add\_numbers(a = 7, b = 8):

...

Here, we have provided default values **7** and **8** for parameters a and b respectively. Here's how this program works

**1. add\_number(2, 3)**

Both values are passed during the function call. Hence, these values are used instead of the default values.

**2. add\_number(2)**

Only one value is passed during the function call. So, according to the positional argument **2** is assigned to argument a, and the default value is used for parameter b.

**3. add\_number()**

No value is passed during the function call. Hence, default value is used for both parameters a and b.

**Python Keyword Argument**

In keyword arguments, arguments are assigned based on the name of arguments. For example,

def display\_info(first\_name, last\_name):

print('First Name:', first\_name)

print('Last Name:', last\_name)

display\_info(last\_name = 'Cartman', first\_name = 'Eric')

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

**Output**

First Name: Eric

Last Name: Cartman

Here, notice the function call,

display\_info(last\_name = 'Cartman', first\_name = 'Eric')

Here, we have assigned names to arguments during the function call.

Hence, first\_name in the function call is assigned to first\_name in the function definition. Similarly, last\_name in the function call is assigned to last\_name in the function definition.

In such scenarios, the position of arguments doesn't matter.

**Python Function With Arbitrary Arguments**

Sometimes, we do not know in advance the number of arguments that will be passed into a function. To handle this kind of situation, we can use arbitrary arguments in Python.

Arbitrary arguments allow us to pass a varying number of values during a function call.

We use an asterisk (\*) before the parameter name to denote this kind of argument. For example,

# program to find sum of multiple numbers

def find\_sum(\*numbers):

result = 0

for num in numbers:

result = result + num

print("Sum = ", result)

# function call with 3 arguments

find\_sum(1, 2, 3)

# function call with 2 arguments

find\_sum(4, 9)

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

**Output**

Sum = 6

Sum = 13

In the above example, we have created the function find\_sum() that accepts arbitrary arguments. Notice the lines,

find\_sum(1, 2, 3)

find\_sum(4, 9)

Here, we are able to call the same function with different arguments.

**Note**: After getting multiple values, numbers behave as an array so we are able to use the for loop to access each value.

# Python Recursion

In this tutorial, you will learn to create a recursive function (a function that calls itself).

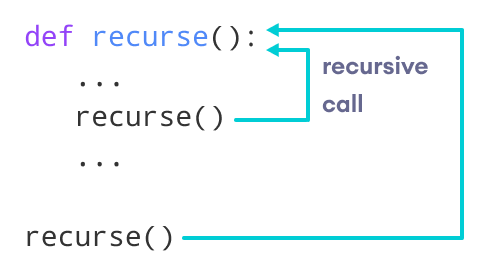
Recursion is the process of defining something in terms of itself.

A physical world example would be to place two parallel mirrors facing each other. Any object in between them would be reflected recursively.

## Python Recursive Function

In Python, we know that a [function](https://www.programiz.com/python-programming/function) can call other functions. It is even possible for the function to call itself. These types of construct are termed as recursive functions.

The following image shows the working of a recursive function called recurse.



Following is an example of a recursive function to find the factorial of an integer.

Factorial of a number is the product of all the integers from 1 to that number. For example, the factorial of 6 (denoted as 6!) is 1\*2\*3\*4\*5\*6 = 720.

### Example of a recursive function

def factorial(x):

"""This is a recursive function

to find the factorial of an integer"""

if x == 1:

return 1

else:

return (x \* factorial(x-1))

num = 3

print("The factorial of", num, "is", factorial(num))

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

**Output**

The factorial of 3 is 6

In the above example, factorial() is a recursive function as it calls itself.

When we call this function with a positive integer, it will recursively call itself by decreasing the number.

Each function multiplies the number with the factorial of the number below it until it is equal to one. This recursive call can be explained in the following steps.

factorial(3) # 1st call with 3

3 \* factorial(2) # 2nd call with 2

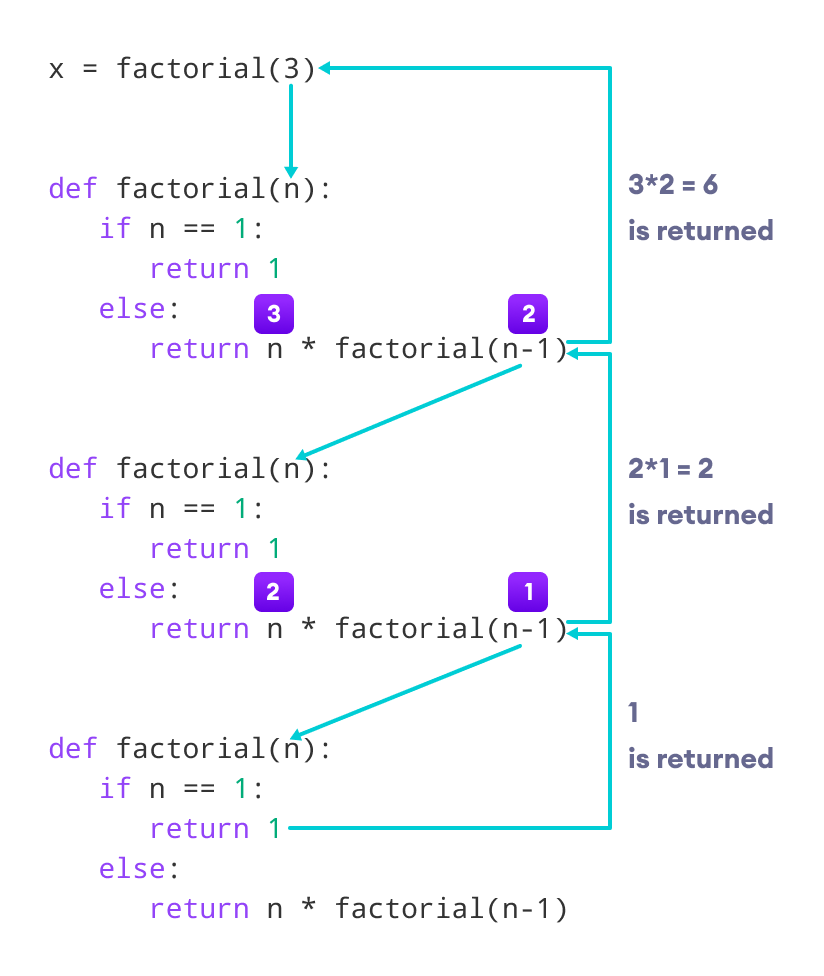
3 \* 2 \* factorial(1) # 3rd call with 1

3 \* 2 \* 1 # return from 3rd call as number=1

3 \* 2 # return from 2nd call

6 # return from 1st call

Let's look at an image that shows a step-by-step process of what is going on:



Our recursion ends when the number reduces to 1. This is called the base condition.

Every recursive function must have a base condition that stops the recursion or else the function calls itself infinitely.

The Python interpreter limits the depths of recursion to help avoid infinite recursions, resulting in stack overflows.

By default, the maximum depth of recursion is 1000. If the limit is crossed, it results in RecursionError. Let's look at one such condition.

def recursor():

recursor()

recursor()

**Output**

Traceback (most recent call last):

File "<string>", line 3, in <module>

File "<string>", line 2, in a

File "<string>", line 2, in a

File "<string>", line 2, in a

[Previous line repeated 996 more times]

RecursionError: maximum recursion depth exceeded

## Advantages of Recursion

1. Recursive functions make the code look clean and elegant.
2. A complex task can be broken down into simpler sub-problems using recursion.
3. Sequence generation is easier with recursion than using some nested iteration.

## Disadvantages of Recursion

1. Sometimes the logic behind recursion is hard to follow through.
2. Recursive calls are expensive (inefficient) as they take up a lot of memory and time.
3. Recursive functions are hard to debug.

# Python Lambda/Anonymous Function

In this tutorial, we'll learn about Python lambda functions with the help of examples.

In Python, a lambda function is a special type of function without the function name. For example,

lambda : print('Hello World')

Here, we have created a lambda function that prints 'Hello World'.

Before you learn about lambdas, make sure to know about [Python Functions](https://www.programiz.com/python-programming/function).

**Python lambda Function Declaration**

We use the lambda keyword instead of def to create a lambda function. Here's the syntax to declare the lambda function:

lambda argument(s) : expression

Here,

* argument(s) - any value passed to the lambda function
* expression - expression is executed and returned

Let's see an example,

greet = lambda : print('Hello World')

Here, we have defined a lambda function and assigned it to the variable named greet.

To execute this lambda function, we need to call it. Here's how we can call the lambda function

# call the lambda

greet()

The lambda function above simply prints the text 'Hello World'.

**Note**: This lambda function doesn't have any arguments.

**Example: Python lambda Function**

# declare a lambda function

greet = lambda : print('Hello World')

# call lambda function

greet()

# Output: Hello World

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

In the above example, we have defined a lambda function and assigned it to the greet variable.

When we call the lambda function, the print() statement inside the lambda function is executed.

**Python lambda Function with an Argument**

Similar to normal functions, the lambda function can also accept arguments. For example,

# lambda that accepts one argument

greet\_user = lambda name : print('Hey there,', name)

# lambda call

greet\_user('Delilah')

# Output: Hey there, Delilah

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

In the above example, we have assigned a lambda function to the greet\_user variable.

Here, name after the lambda keyword specifies that the lambda function accepts the argument named name.

Notice the call of lambda function,

greet\_user('Delilah')

Here, we have passed a string value 'Delilah' to our lambda function.

And finally, the statement inside the lambda function is executed.

**Frequently Asked Questions**

How to use the lambda function with filter()?

How to use the lambda function with map()?

The [map()](https://www.programiz.com/python-programming/methods/built-in/map) function in Python takes in a function and an iterable (lists, tuples, and strings) as arguments.

The function is called with all the items in the list and a new list is returned which contains items returned by that function for each item.

Let's see an example,

# Program to double each item in a list using map()

my\_list = [1, 5, 4, 6, 8, 11, 3, 12]

new\_list = list(map(lambda x: x \* 2 , my\_list))

print(new\_list)

# Output: [2, 10, 8, 12, 16, 22, 6, 24]

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

Here, the map() function doubles all the items in a list.

# Python Variable Scope

In this tutorial, we'll learn about Python Global variables, Local variables, and Nonlocal variables with the help of examples.

In Python, we can declare variables in three different scopes: local scope, global, and nonlocal scope.

A variable scope specifies the region where we can access a variable. For example,

def add\_numbers():

sum = 5 + 4

Here, the sum variable is created inside the function, so it can only be accessed within it (local scope). This type of variable is called a local variable.

Based on the scope, we can classify Python variables into three types:

1. Local Variables
2. Global Variables
3. Nonlocal Variables

**Python Local Variables**

When we declare variables inside a function, these variables will have a local scope (within the function). We cannot access them outside the function.

These types of variables are called local variables. For example,

def greet():

# local variable

message = 'Hello'

print('Local', message)

greet()

# try to access message variable

# outside greet() function

print(message)

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

**Output**

Local Hello

NameError: name 'message' is not defined

Here, the message variable is local to the greet() function, so it can only be accessed within the function.

That's why we get an error when we try to access it outside the greet() function.

To fix this issue, we can make the variable named message global.

**Python Global Variables**

In Python, a variable declared outside of the function or in global scope is known as a global variable. This means that a global variable can be accessed inside or outside of the function.

Let's see an example of how a global variable is created in Python.

# declare global variable

message = 'Hello'

def greet():

# declare local variable

print('Local', message)

greet()

print('Global', message)

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

**Output**

Local Hello

Global Hello

This time we can access the message variable from outside of the greet() function. This is because we have created the message variable as the global variable.

# declare global variable

message = 'Hello'

Now, message will be accessible from any scope (region) of the program.

**Python Nonlocal Variables**

In Python, nonlocal variables are used in nested functions whose local scope is not defined. This means that the variable can be neither in the local nor the global scope.

We use the nonlocal keyword to create nonlocal variables.For example,

# outside function

def outer():

message = 'local'

# nested function

def inner():

# declare nonlocal variable

nonlocal message

message = 'nonlocal'

print("inner:", message)

inner()

print("outer:", message)

outer()

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

**Output**

inner: nonlocal

outer: nonlocal

In the above example, there is a nested inner() function. We have used the nonlocal keywords to create a nonlocal variable.

The inner() function is defined in the scope of another function outer().

**Note** : If we change the value of a nonlocal variable, the changes appear in the local variable.