**Table of Contents**

* What is a function in Python?
  + Syntax of Function
  + Example of a function
  + How to call a function in python?
* Docstring
* The return statement
  + Syntax of return
  + Example of return
* How Function works in Python?
* Scope and Lifetime of variables
* Types of Functions

## What is a function in Python?

In Python, function is a group of related statements that perform a specific task.

Functions help break our program into smaller and modular chunks. As our program grows larger and larger, functions make it more organized and manageable.

Furthermore, it avoids repetition and makes code reusable.

### Syntax of Function

def function\_name(parameters):

"""docstring"""

statement(s)

## The return statement

The return statement is used to exit a function and go back to the place from where it was called.

### Syntax of return

return [expression\_list]

This statement can contain expression which gets evaluated and the value is returned. If there is no expression in the statement or the return statement itself is not present inside a function, then the function will return the None object.

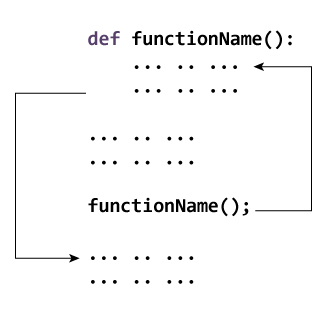
For example:

>>> print(greet("May"))

Hello, May. Good morning!

None

## How Function works in Python?



## Scope and Lifetime of variables

Scope of a variable is the portion of a program where the variable is recognized. Parameters and variables defined inside a function is not visible from outside. Hence, they have a local scope.

Lifetime of a variable is the period throughout which the variable exits in the memory. The lifetime of variables inside a function is as long as the function executes.

They are destroyed once we return from the function. Hence, a function does not remember the value of a variable from its previous calls.

Here is an example to illustrate the scope of a variable inside a function.

def my\_func():

x = 10

print("Value inside function:",x)

x = 20

my\_func()

print("Value outside function:",x)

## Arguments

In [user-defined function](https://www.programiz.com/python-programming/user-defined-function) topic, we learned about defining a function and calling it. Otherwise, the function call will result into an error. Here is an example.

def greet(name,msg):

"""This function greets to

the person with the provided message"""

print("Hello",name + ', ' + msg)

### Python Arbitrary Arguments

Sometimes, we do not know in advance the number of arguments that will be passed into a function.Python allows us to handle this kind of situation through function calls with arbitrary number of arguments.

In the function definition we use an asterisk (\*) before the parameter name to denote this kind of argument. Here is an example.

def greet(\*names):

"""This function greets all

the person in the names tuple."""

# names is a tuple with arguments

for name in names:

print("Hello",name)

### Python Keyword Arguments

When we call a function with some values, these values get assigned to the arguments according to their position.

For example, in the above function greet(), when we called it as greet("Bruce","How do you do?"), the value "Bruce" gets assigned to the argument name and similarly "How do you do?" to msg.

Python allows functions to be called using keyword arguments. When we call functions in this way, the order (position) of the arguments can be changed. Following calls to the above function are all valid and produce the same result.

>>> # 2 keyword arguments

>>> greet(name = "Bruce",msg = "How do you do?")

>>> # 2 keyword arguments (out of order)

>>> greet(msg = "How do you do?",name = "Bruce")

>>> # 1 positional, 1 keyword argument

>>> greet("Bruce",msg = "How do you do?")

## What is recursion in Python?

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.

## Recursive Function

# An example of a recursive function to

# find the factorial of a number

def calc\_factorial(x):

"""This is a recursive function

to find the factorial of an integer"""

if x == 1:

return 1

else:

return (x \* calc\_factorial(x-1))

num = 4

print("The factorial of", num, "is", calc\_factorial(num))

## 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 Anonymous/Lambda Function

## What are lambda functions in Python?

In Python, anonymous function is a [function](https://www.programiz.com/python-programming/function) that is defined without a name.

While normal functions are defined using the def keyword, in Python anonymous functions are defined using the lambda keyword.

Hence, anonymous functions are also called lambda functions.

## How to use lambda Functions in Python?

A lambda function in python has the following syntax.

### Syntax of Lambda Function in python

lambda arguments: expression

Lambda functions can have any number of arguments but only one expression. The expression is evaluated and returned. Lambda functions can be used wherever function objects are required.

### Example of Lambda Function in python

Here is an example of lambda function that doubles the input value.

# Program to show the use of lambda functions

double = lambda x: x \* 2

# Output: 10

print(double(5))

In the above program, lambda x: x \* 2 is the lambda function. Here x is the argument and x \* 2 is the expression that gets evaluated and returned.

This function has no name. It returns a function object which is assigned to the identifier double. We can now call it as a normal function. The statement

double = lambda x: x \* 2

is nearly the same as

def double(x):

return x \* 2

### Use of Lambda Function in python

We use lambda functions when we require a nameless function for a short period of time.

In Python, we generally use it as an argument to a higher-order function (a function that takes in other functions as [arguments](https://www.programiz.com/python-programming/function-argument" \o "Python function arguments)). Lambda functions are used along with built-in functions like filter(), map() etc.

#### Example use with filter()

The filter() function in Python takes in a function and a list as arguments.

The function is called with all the items in the list and a new list is returned which contains items for which the function evaluats to True.

Here is an example use of filter() function to filter out only even numbers from a list.

# Program to filter out only the even items from a list

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

new\_list = list(filter(lambda x: (x%2 == 0) , my\_list))

# Output: [4, 6, 8, 12]

print(new\_list)

#### Example use with map()

The map() function in Python takes in a function and a list.

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.

Here is an example use of map() function to double all the items in a list.

# 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))

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

print(new\_list)

# Python Global, Local and Nonlocal variables

## Global Variables

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

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

### Example 1: Create a Global Variable

x = "global"

def foo():

print("x inside :", x)

foo()

print("x outside:", x)

## Global and local variable

x = "global"

def foo():

global x

y = "local"

x = x \* 2

print(x)

print(y)

foo()