# Reading: Functions, Classes, Objects

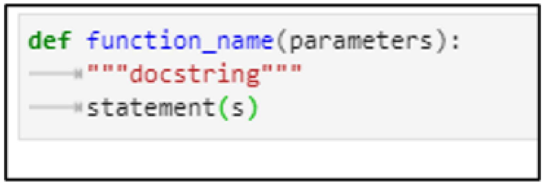
Bookmark this page

## FUNCTIONS

In Python, a function is a group of related statements that performs a specific task and can be reused. Functions help break our program into smaller and modular chunks. As our program grows larger and larger, functions make it more organized and manageable.

If we define a function to do the task, we just need to call the function.

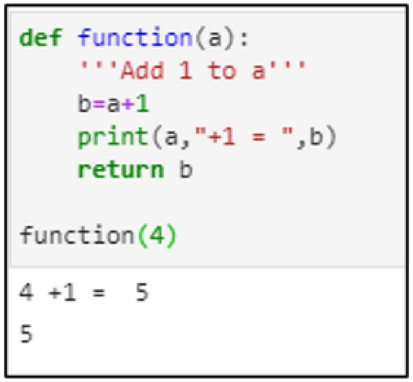
The syntax is:



```
def function_name(parameters):
    """docstring"""
    statement(s)
```

Keyword def marks the start of the function header. A function name uniquely identifies the function. Parameters are the arguments used to pass values to a function. They are optional. A colon (:) marks the end of the function header. Optional documentation string (docstring) describes what the function does. One or more valid python statements makes up the function body. An optional return statement is used at the end of the function definition which returns the value of the function.

Consider the following example:



```
def function(a):
    '''Add 1 to a'''
    b=a+1
    print(a,"+1 = ",b)
    return b

function(4)
```

```
4 +1 =  5
5
```

Once we have defined a function, we call the defined function by writing the function name with the appropriate parameters.

Here in this example, we call the defined function ***function(a)*** by writing ***function(4)*** ‘4’ is the parameter passed.

## Docstrings

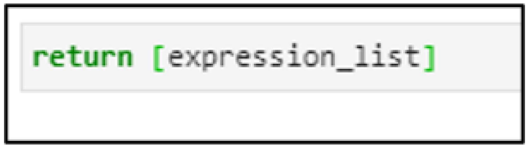
The first string after the function header is called the docstring and is short for documentation string. It is used to briefly explain what the function does.

In the above example: ***‘’’Add 1 to a’’’*** is the docstring.

## The *return* Statement

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

The syntax is:



```
return [expression_list]
```

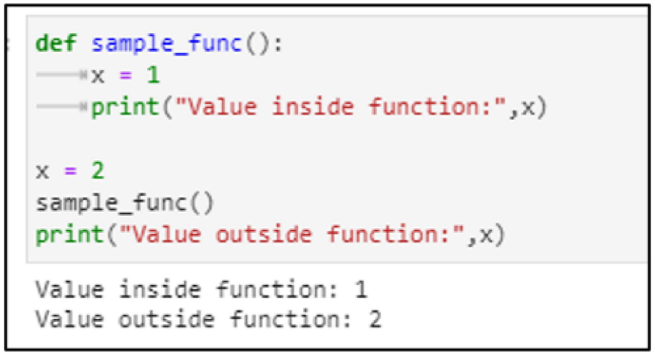
This statement can contain an expression that gets evaluated and the value is returned. If there is no expression in the statement or the return statement itself is not present inside a function, then the function will return the ***None*** object.

## Scope and Lifetime of Variables

The scope of a variable is the part of the program where that variable is accessible. Parameters and variables that are defined outside of any function are said to be within the **global scope**, meaning they can be accessed anywhere after they are defined. Parameters and variables defined inside a function are not visible from outside the function. Hence, they have a **local scope**

The 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.

Consider the following example:



```
def sample_func():
    x = 1
    print("Value inside function:",x)

x = 2
sample_func()
print("Value outside function:",x)
```

```
Value inside function: 1
Value outside function: 2
```

Here, we can see that the value of **x is 2 initially**. Even though the function ***sample_func()*** changed the value of x to 1, it **did not affect** the value outside the function. This is because the variable x inside the function is **local to the function**. Although they have the same names, they are two different variables with different scopes.

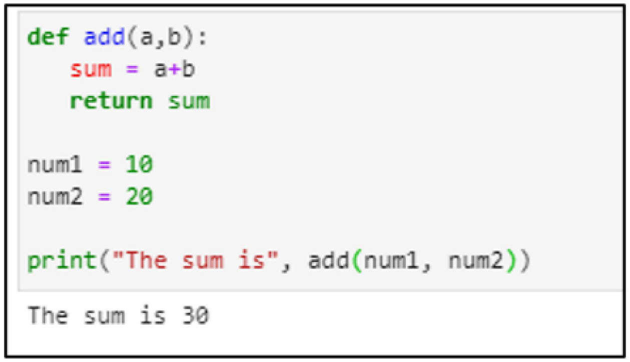
On the other hand, variables outside of the function are visible from inside. They have a **global scope**. We can read these values from inside the function but **cannot** change them. In order to modify the value of variables outside the function, they must be declared as global variables using the keyword ***global***.

### Types of Functions: Built-in and User Defined

There are 2 types of functions in Python:

a. Built-in Functions: Functions that are built into Python. These are readily available for use. There are a number of built-in functions available in Python. You may find them here.

b. User-defined Functions: Functions that are defined by users for their use.
Consider the following example:



```
def add(a,b):
    sum = a+b
    return sum

num1 = 10
num2 = 20

print("The sum is", add(num1, num2))
```

```
The sum is 30
```

## OBJECTS AND CLASSES

## Python Object-Oriented Programming

Python supports The **Object-Oriented Programming (OOP)** approach. Almost everything in python is an object, with its properties and methods. An object has two characteristics:

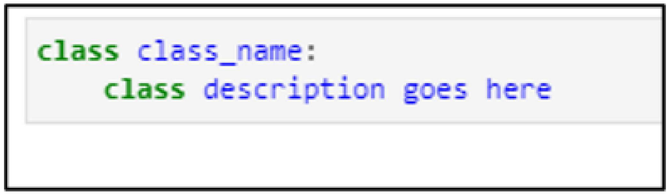
a. attributes

b. an internal data representation (a blueprint)

## Python Class

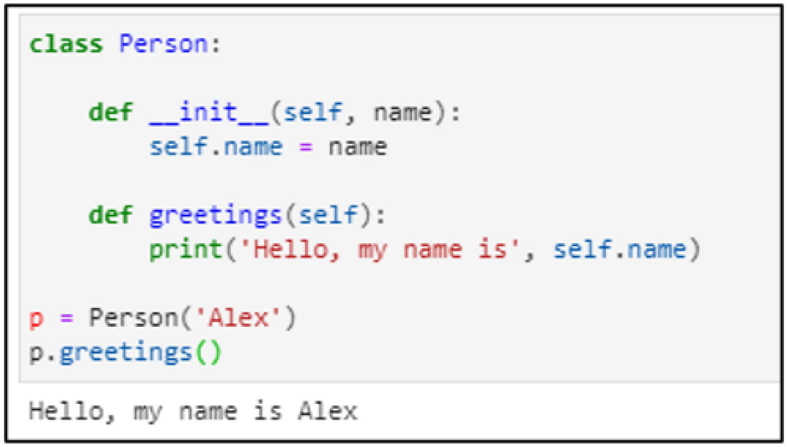
A class is a blueprint for the object. To create a class, we use the keyword ***class.***

The syntax is:



```
class class_name:
    class description goes here
```

Consider the following example:



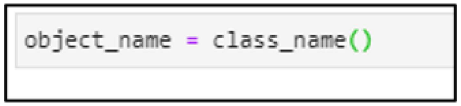
```
class Person:

    def __init__(self, name):
        self.name = name

    def greetings(self):
        print('Hello, my name is', self.name)

p = Person('Alex')
p.greetings()
```

```
Hello, my name is Alex
```

## Objects in Python

An object (instance) is an instantiation of a class. When class is defined, only the description for the object is defined. Therefore, no memory or storage is allocated.

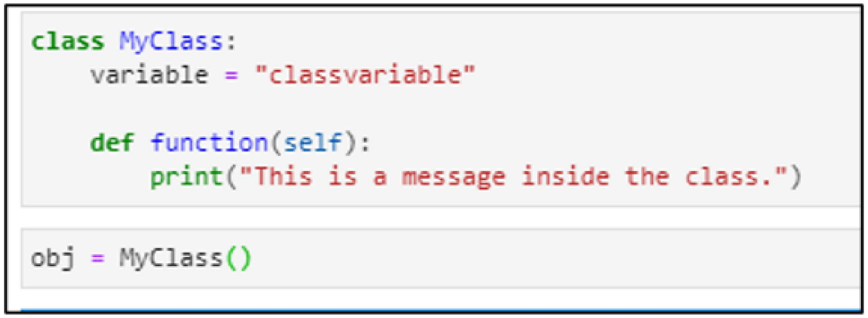
The syntax is:



```
object_name = class_name()
```

Here, *object_name* is the object of the class *class_name.*

Consider the following example:
A very basic class would look something like this:



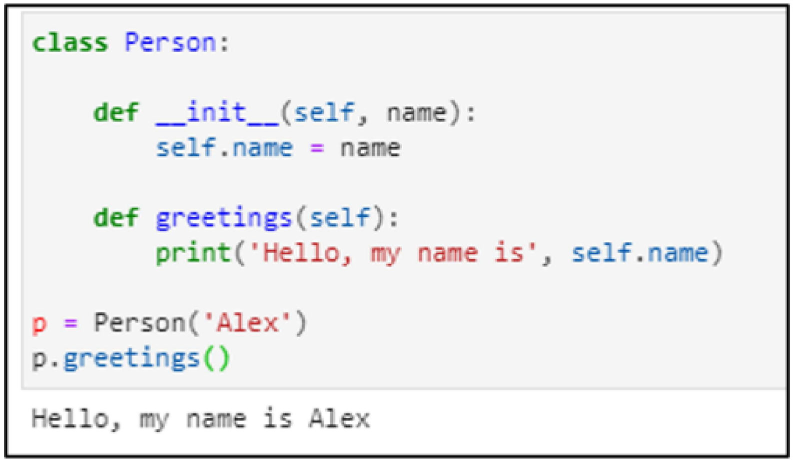
```
class MyClass:
    variable = "classvariable"

    def function(self):
        print("This is a message inside the class.")
```

```
obj = MyClass()
```

Here the class name is ***MyClass*** and ***obj*** is the object of *MyClass.*

**Methods:** Methods are functions defined inside the body of a class. They are used to define the behaviours of an object.

**_init_ method:** The __init__ method is similar to constructors in C++ and Java. It is run as soon as an object of a class is instantiated. The method is useful to do any initialization you want to do with your object.



```
class Person:

    def __init__(self, name):
        self.name = name

    def greetings(self):
        print('Hello, my name is', self.name)

p = Person('Alex')
p.greetings()
```

```
Hello, my name is Alex
```

Here, ***def greetings(self):*** is a method which prints the name of the user. ***'p'*** is the object of class ***Person.***

---

In today's modern age of disruption, SkillUp Online is your ideal learning platform that enables you to upskill to the most in-demand technology skills like Data Science, Big Data, Artificial Intelligence, Cloud, Front-End Development, DevOps & many more. In your journey of evolution as a technologist, SkillUp Online helps you work smarter, get to your career goals faster and create an exciting technology led future.

## Corporate

- Home

- About Us
- Enterprise
- Blog
- Press

## Support

- Contact us
- Terms of Service
- Privacy Policy

Copyright ©2020 Skillup. All Rights Reserved