Python Functions

A function is a block of code which only runs when it is called.

You can pass data, known as parameters, into a function.

A function can return data as a result.

A function is a block of organized, reusable code that is used to perform a single, related action. Functions provide better modularity for your application and a high degree of code reusing.

As you already know, Python gives you many built-in functions like print, range, length etc. but you can also create your own functions. These functions are called user-defined functions.

**Defining a Function**

You can define functions to provide the required functionality. Here are simple rules to define a function in Python.

* Function blocks begin with the keyword def followed by the function name and parentheses ( ).
* Any input parameters or arguments should be placed within these parentheses. You can also define parameters inside these parentheses.
* The code block within every function starts with a colon : and is indented.

Creating a Function

In Python a function is defined using the def keyword:

Example

def my\_function():  
  print("Hello from a function")

Calling a Function

To call a function, use the function name followed by parenthesis:

Example

def my\_function():

print("Hello from a function")

my\_function()

Hello from a function

## Arguments

Information can be passed into functions as arguments.

Arguments are specified after the function name, inside the parentheses. You can add as many arguments as you want, just separate them with a comma.

The following example has a function with one argument (fname). When the function is called, we pass along a first name, which is used inside the function to print the full name:

### Example

def my\_function(fname):

print(fname + " Refer")

my\_function("john")

my\_function("sony")

my\_function("indhu")

john Refer  
sony Refer  
indhu Refer

Parameters or Arguments?

The terms *parameter* and *argument* can be used for the same thing: information that are passed into a function.

From a function's perspective:

A parameter is the variable listed inside the parentheses in the function definition.

An argument is the value that is sent to the function when it is called.

Number of Arguments

By default, a function must be called with the correct number of arguments. Meaning that if your function expects 2 arguments, you have to call the function with 2 arguments, not more, and not less.

Example

This function expects 2 arguments, and gets 2 arguments:

def my\_function(fname, lname):

print(fname + " " + lname)

my\_function("indhu", "refer")

indhu Refer

If you try to call the function with 1 or 3 arguments, you will get an error:

Example

This function expects 2 arguments, but gets only 1:

def my\_function(fname, lname):  
  print(fname + " " + lname)  
  
my\_function("Emil")

## Arbitrary Arguments, \*args

If you do not know how many arguments that will be passed into your function, add a \* before the parameter name in the function definition.

This way the function will receive a tuple of arguments, and can access the items accordingly:

### Example

If the number of arguments is unknown, add a \* before the parameter name:

def my\_function(\*kids):

print("The youngest child is " + kids[2])

my\_function("Emil", "Tobias", "Linus")

The youngest child is Linus

## Keyword Arguments

You can also send arguments with the key = value syntax.

This way the order of the arguments does not matter.

### Example

def my\_function(child3, child2, child1):

print("The youngest child is " + child3)

my\_function(child1 = "Emil", child2 = "Tobias", child3 = "Linus")

The youngest child is Linus

## Default Parameter Value

The following example shows how to use a default parameter value.

If we call the function without argument, it uses the default value:

### Example

def my\_function(country = "Norway"):

print("I am from " + country)

my\_function("Sweden")

my\_function("India")

my\_function()

my\_function("Brazil")

I am from Sweden  
I am from India  
I am from Norway  
I am from Brazil

## Passing a List as an Argument

You can send any data types of argument to a function (string, number, list, dictionary etc.), and it will be treated as the same data type inside the function.

E.g. if you send a List as an argument, it will still be a List when it reaches the function:

def my\_function(fruits):

for x in fruits:

print(x)

fruits = ["apple", "banana", "cherry"]

my\_function(fruits)

apple  
banana  
cherry

# Function definition is here

def printme( str ):

#"This prints a passed string into this function"

      print (str)

      return;

# Now you can call printme function

printme("I'm first call to user defined function!")

printme("Again second call to the same function")

I'm first call to user defined function!

Again second call to the same function

## Return Values

To let a function return a value, use the return statement:

### Example

def my\_function(x):

return 5 \* x

print(my\_function(3))

print(my\_function(5))

print(my\_function(9))

15  
25  
45

The return Statement The statement return [expression] exits a function, optionally passing back an expression to the caller. A return statement with no arguments is the same as return None. All the above examples are not returning any value. You can return a value from a function as follows –

# Function definition is here

def sum( arg1, arg2 ):

# Add both the parameters and return them."

     total = arg1 + arg2

     print ("Inside the function : ", total)

     return total;

# Now you can call sum function

total = sum( 10, 20 );

print ("Outside the function : ", total)

Inside the function : 30

Outside the function : 30