BRAC UNIVERSITY

Inspiring Excellence

Course Title: Programming Language II

Course Code: CSE 111

Semester: Summer 2020

Topic: Introduction to Functions

Table of Contents

Functions

A function is a set of statements taking inputs, performing some specific computation and producing output. It only runs when it is called.

You may pass data into a function, known as **parameters**.

As a result, function can return **data**.

There can be 4 types of functions.

1. **Built-In Function**
2. **User-Defined Functions**
3. **Lambda Function**
4. **Recursion Function**

Why do we need functions?

The idea is to bring together some frequently or repeatedly performed task and create a function, so that we can call the function rather than always rewriting the same code over and over for various inputs. Functions give a program greater modularity and a higher level of code reuse.

```
def myfunc(x,y):
    sum=a+b
    return sum

result = myfunc (10,20)
result = myfunc (result,30)
result = myfunc (result,40)
print(result)
```

Output:

```
100
```

Built-in functions

The Python interpreter contains a variety of functions and types, which are always accessible. We can always call them when it is needed.

For example,

print()	For printing an object to the terminal
len()-	For getting the length (the number of items) of an object
max()-	For getting the maximum value
min()-	For getting the minimum value

abs()-	For getting the absolute value of a number
open()-	For opening file and return a corresponding file object
sorted()-	For getting a sorted list

User Defined Functions

These type of functions are created by the users to help themselves out. There are two types of user defined functions.

1. **Void function**- functions that do not return anything.
2. **Fruitful function**- functions that returns a value rather than **None**.

In Python a function is declared using the **def** keyword:

Example:

```
def my_function():
    print("My first function")
```

Use the function name with the parenthesis for calling a function and showing the output.

```
def my_function():
    print("My first function")
my_function()
```

```
Output:

My first function
```

Parameters and arguments

A parameter is a variable defined by a function (within the parenthesis) that receives a value when the function is called.

An argument is a value that is passed to a function when it is invoked.

Example 1:

```
def my_function(stuname):
    print(“Hi”,stuname)

my_function(“Harry")
my_function(“Ron”)
my_function(“Hermione")
```

```
Output:
Hi Harry
Hi Ron
Hi Hermione
```

In the above example, **stuname** is the parameter and **Harry, Ron, Hermione** are arguments.

Example 2:

```
def my_function(x,y):
  z=x+y
  print(x,"+",y,"is",z)

my_function(5,5)
my_function(20,20)
my_function(50,50)
```

```
Output:
5 + 5 is 10
20 + 20 is 40
50 + 50 is 100
```

In the above example, **x, y** are the parameter and **5, 5; 10, 10** etc are the arguments.

Number of Arguments

A function has to be called with the correct number of arguments. This means that if your function has 2 parameters, the function must be called with exactly 2 arguments.

```
def my_function(name, place):
    print(name+" goes to "+place)

my_function("Harry", "School")
```

```
Output:

Harry goes to School
```

When you try to call it with more or less than 2 arguments, it will generate an error.

```
def my_function(name, place):
    print(name+" goes to "+place)

my_function("Harry")
```

This will give an error, no output will be shown.

Unknown Number of Arguments (args*)

If the number of arguments is unknown such as you do not know how many arguments will be passed in the function, then add an asterisk (‘*’) before the parameter name:

Example:

```
def my_function(*kids):
    print("The first child is " + kids[0])

my_function("Bil", "Ron", "Tom")
```

```
Output:
The first child is Bil
```

Keyword Arguments

The arguments can also be sent by using use **key = value** syntax. This way the order of the arguments does not make a difference.

```
def my_function(stu3, stu2, stu1):
    print("The best student is " + stu3)

my_function(stu1 = "Ron", stu2 = "Bob", stu3 = "Tom")
```

```
Output:
The best student is Tom
```

Default Argument Values

Default arguments are those that take a default value if no argument value is passed during the function call. A default value can be written in the format "argument = value". So we have the option to assign a value for those arguments or not.

```
def my_function ( name, age = 20 ):
    print ("Name: ", name)
    print ("Age ", age)

my_function (name = "Harry", age = 35)
my_function (name = "Harry")
```

```
Output:
Name: Harry
Age  35
Name: Harry
Age  20
```

‘Return’ Statement

Functions those return a value are called Fruitful functions. Use return statement if you want the function to return a value.

Example 1:

```
def addition(x):
    return 5 + x

print(addition (10))
print(addition (20))
print(addition (30))
```

```
Output:
15
25
35
```

Example 2:

```
def nsquare(x, y):
    return (x*x + 2*x*y + y*y)

print("The square of the sum of 2 and 3 is : ", nsquare(2, 3))
```

```
Output:

The square of the sum of 2 and 3 is :  25
```

Empty and Lambda Function

Function definitions cannot be empty, but if you have a function definition with no content for some reason, put in the **pass** statement to avoid an error.

```
def myfunction():
    pass
myfunction()
```

```
There will be no output
```

A **lambda** function (or a lambda expression more accurately) is simply a function that you can define on-the-spot, right where you might need it.

The general syntax for lambda function:

lambda argument_list: expression

Here, the argument list consists of a list separated by comma, of arguments and the expression is an arithmetic expression that uses these arguments.

```
def addition(x, y):
    return (x + y)


print(addition(20,30))
```

```
print((lambda x, y: (x + y))(20,30))
```

Both of the above codes produce the same output.

```
Output:
50
```

Scope

A variable is only accessible from within the place it is generated. It is called scope.

Local Variable

A variable that is declared within a function is called Local variable and the scope is local scope.

Example,

```
def myfunc():
  name = "Harry"
  print(name)

myfunc()
```

```
Output:

Harry
```

Now if we try to print the name outside the function, it will give an error.

```
def myfunc():
  name = "Harry"

myfunc()
print(name)
```

```
NameError: name
'name' is not defined
```

Global Variable

A variable that is declared outside the function is called global variable and the scope is global scope.

```
name = "Harry"
def myfunc():
  print(name,"is local now")

myfunc()
print(name,"is global now")
```

```
Output:

Harry is local now

Harry is global now
```

Now watch this example,

```
name = "Harry"
def myfunc():
  name="Ron"
  print(name)

myfunc()
print(name)
```

```
Output:
Ron
Harry
```

So, if we work with same variable name within the local scope and also in the global scope, python treats them as separate variables.

Now, if you are in a local scope, but you want to create a global variable, you can do so using **global** keyword.

```
def myfunc():
  global name
  name="Harry"

myfunc()
print(name)
```

```
Output:

Harry
```