Table of Contents

Chapter 10

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.

The following are the different types of Python Functions:
Python Built-in Functions.
Python Recursion Functions.
Python Lambda Functions.
Python User-defined Functions

Creating a Function

In Python a function is defined using the def keyword:

```
def perfect():
   print("Madi is perfect")
```

Calling a Function

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

```
def perfect():
   print("Madi is perfect")

perfect()  # function calling
```

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.

```
# Function with arguments
def message(myName):
   print("Brooooo ",myName,", Madi is a feeling!!!")

message("Omar")
```

Parameters or Arguments?

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.

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.

```
def myArgsFunc(*pracHb):
    print("I am from myArgsFunc")
    print("The hobby to be played now: " + pracHb[1])

myArgsFunc("Chess", "Football", "Cricket")
```

Keyword Arguments

You can also send arguments with the key = value syntax.
This way the order of the arguments does not matter.

```
def myKeyArgs(hobby1, hobby2, hobby3):
    print("I am from myKeyArgs")
    print("The hobby to be played now: " + hobby2)

myKeyArgs(hobby1="Chess", hobby2="Football", hobby3="Cricket")
```

Arbitrary Keyword Arguments, **kwargs

If you do not know how many keyword arguments that will be passed into your function, add two asterisk: ** before the parameter name in the function definition.
This way the function will receive a dictionary of arguments, and can access the items accordingly:

```
def arbKeyArg(**ch):
    print("Arbitary Keyword Arguments")
    print("Madi is " + ch["heart"])

arbKeyArg(heart = "Feeling", face = "Emotion")
```

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:

```
def defaArg(country = "India"):
  print("I am from " + country)

defaArg()
```

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 listAsArg(food):
    for x in food:
        print(x)

fruits = ["apple", "banana", "cherry"]
listAsArg(fruits)
```

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

```
def my_function(x):
  return 5 * x

print(my_function(3))
print(my_function(5))
print(my_function(9))
```

The pass Statement

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

```
def willUse():
    pass
```

Recursion

Python also accepts function recursion, which means a defined function can call itself.
Recursion is a common mathematical and programming concept. It means that a function calls itself. This has the benefit of meaning that you can loop through data to reach a result.

```
def factRecurssive(n):
    if n == 1 or n == 0:
        return 1
    else:
        return n*factRecurssive(n-1)

fr = factRecurssive(3)
print("Recursive Factorial is ",fr)
```

Lambda Function:
A lambda function is a small anonymous function.
A lambda function can take any number of arguments, but can only have one expression.
Syntax:
lambda arguments : expression

```
# def myAdd(x,y):
#        return x+y

# print(myAdd(5,7))

x = lambda a, b: a + b
print(x(7, 5))

# Ex. 2
def myfunc(n):
  return lambda a : a * n

mydoubler = myfunc(2)

print(mydoubler(11))
```

```
# Ex. 3
def myfunc(n):
  return lambda a : a * n

mydoubler = myfunc(2)
mytripler = myfunc(3)
print(mydoubler(11))
print(mytripler(11))
```