# Assignment 3

## 1. Why are functions advantageous to have in your programs?

**Answer**:

i. With the help of functions, we can avoid rewriting the same logic or code again and again in a program. Functions reduce the need for duplicate code. This makes programs shorter, easier to read, and easier to update.
ii. We can track a large Python program easily when it is divided into multiple functions.
iii. The main achievement of Python functions is their Reusability.

## 2. When does the code in a function run: when it's specified or when it's called?

**Answer:**

The code inside a function is not executed when the function is defined. The code inside a function is executed when the function is invoked. It is common to use the term "call a function" instead of "invoke a function". Ex.:

```
def myfunction( ):
    print("Hello World")
myfunction( )
```

## 3. What statement creates a function?

**Answer:** Define a function with the 'def' keyword, then write the function identifier (name) followed by parentheses and a colon. Ex. `def myfunction( ):`

## 4. What is the difference between a function and a function call?

**Answer:** ***Function:*** A Function is block of code than accepts some values processes the desire task on it and return the result value. ***Function call:*** Using a function to do a particular task any point in program is called as function call. So, the difference between the function and function call is, A function is a procedure to achieve a particular result while a function call is using this function to achieve that task.

**5. How many global scopes are there in a Python program? How many local scopes?**

**Answer:** There is one global scope, and a local scope is created whenever a function is called.

Self-Note: A variable is only available from inside the region it is created. This is called **scope**. In Python, we can declare variables in three different scopes: local scope, global, and nonlocal scope**.**

A variable scope specifies the region where we can access a variable. For example,

```
def add_numbers():
    sum = 5 + 4
```

Here, the **sum** variable is created inside the function, so it can only be accessed within it (local scope). This type of variable is called a local variable.

Based on the scope, we can classify Python variables into three types:

I. Local Variables
II. Global Variables
III. Nonlocal Variables

**Local Variables**

When we declare variables inside a function, these variables will have a local scope (within the function). We cannot access them outside the function.

These types of variables are called local variables. For example,

```
def greet():
    # local variable
    message = 'Hello'
    print('Local', message)
greet()
# try to access message variable
# outside greet() function
print(message)
```

**Output:**

```
Local Hello
NameError: name 'message' is not defined
```

Here, the message variable is local to the greet() function, so it can only be accessed within the function. That's why we get an error when we try to access it outside the greet() function. To fix this issue, we can make the variable named message global.

**Global Variables**

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

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

```
# declare global variable
message = 'Hello'


def greet():
    # declare local variable
    print('Local', message)


greet()
print('Global', message)
```

**Output:**

```
Local Hello
Global Hello
```

This time we can access the message variable from outside of the greet() function. This is because we have created the message variable as the global variable.

```
# declare global variable
message = 'Hello'
```

Now, message will be accessible from any scope (region) of the program.

**Nonlocal Variables**

In Python, nonlocal variables are used in nested functions whose local scope is not defined. This means that the variable can be neither in the local nor the global scope.

We use the nonlocal keyword to create nonlocal variables. For example,

```
# outside function
def outer():
    message = 'local'

    # nested function
    def inner():

        # declare nonlocal variable
        nonlocal message

        message = 'nonlocal'
        print("inner:", message)

    inner()
    print("outer:", message)

outer()
```

**Output:**

```
inner: nonlocal
outer: nonlocal
```

In the above example, there is a nested inner() function. We have used the nonlocal keywords to create a nonlocal variable.

The inner() function is defined in the scope of another function outer().

Note: If we change the value of a nonlocal variable, the changes appear in the local variable.

**6. What happens to variables in a local scope when the function call returns?**

**Answer:** When a function returns, the local scope is destroyed, and all the variables in it are forgotten.

**7. What is the concept of a return value? Is it possible to have a return value in an expression?**

**Answer:** A return value is the value that a function call evaluates to. Like any value, a return value can be used as part of an expression.

Self-Note: **return() in Python**

The return() statement, like in other programming languages ends the function call and returns the result to the caller. It is a key component in any function or method in a code which includes the return keyword and the value that is to be returned after that.

The Python return statement is used to return a value from a function. The user can only use the return statement in a function. It cannot be used outside of the Python function. A return statement includes the return keyword and the value that will be returned after that.
***Some points to remember while using return():***

The statements after the return() statement are not executed.

return() statement can not be used outside the function.

If the return() statement is without any expression, then the NONE value is returned.

Syntax of return() in Python:

```
def func_name():

statements....


return [expression]
```

**8. If a function does not have a return statement, what is the return value of a call to that function?**

**Answer:** If there is no return statement for a function, its return value is None.

**9. How do you make a function variable refer to the global variable?**

**Answer:** A global statement will force a variable in a function to refer to the global variable. Normally, when you create a variable inside a function, that variable is local, and can only be used inside that function. To create a global variable inside a function, you can use the global keyword. Ex.

```
x = "awesome"

def myfunc():

    global x

   x = "fantastic"

myfunc()

print("Python is " + x)
```

**10. What is the data type of None?**

**Answer:** The data type of None is NoneType.

**11. What does the sentence import areallyourpetsnamederic do?**

**Answer:** That import statement imports a module named areallyourpetsnamederic.

**12. If you had a bacon() feature in a spam module, what would you call it after importing spam?**

**Answer:** spam.bacon()

**13. What can you do to save a programme from crashing if it encounters an error?**

**Answer:** Place the line of code that might cause an error in a try clause.

**14. What is the purpose of the try clause? What is the purpose of the except clause?**

**Answer:** The code that could potentially cause an error goes in the try clause. The code that executes if an error happens goes in the except clause. The try block lets you test a block of code for errors. The except block lets you handle the error.