Basic Assignment-03

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

Functions and procedures are the basic building blocks of programs. It is used to avoid repetition of codes within the **program**. This makes **programs** shorter, easier to read, and easier to update.

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

A code in **a function** run when **the function** is called, not when **the function** is specified

3. **What statement creates a function?**

The **"def"** keyword is a statement for defining a function in Python. It starts a function with the def keyword, define a name followed by a colon (:) sign. The "def" call creates the function object and assigns it to the name given, whenever that specific name is called within the program, code will execute.

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

Using "def" keyword we define/ specify any argument or statement as a function. A function is procedure to obtain the output while 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?**

Local Scope: It's created at function call, *not* at function definition, so we can have as many different local scopes as function calls. This is True even if we call the same function multiple times, or recursively. Each call will result in a new local scope being created. It is also called as Function scope

Global scope : It is the top-most scope in a Python program. . There is only one global scope as per program. It contains all of the names that is defined at the top level of a program or a module. This is also called as Module scope.

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

Local variable retains its value until the next time function is called. A local variable becomes undefined after the function call complete. The local variable can be used outside the function any time after the function call completes.

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

The return keyword is used to exit a function and return a value.
In programming, return is a statement that instructs a program to leave the subroutine and go back to the return address. The return statement is either return or return value, where value is a variable or other information coming back from the subroutine. Yes, It is possible to have a return value in an expression and if return statement is without any expression, then the special value None is returned. One point to be noted that Return statement can't be used outside the function.

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

function without an explicit return statement returns None.

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

   when we 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, we can use the “**global”** keyword.
   as

```
def func():
        global x
        x = "fantastic"
func()
print("Python is " + x)
```

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

    None keyword is an object, and it is a data type of the class NoneType. We can assign None to any variable, but we can not create other NoneType objects.

11. **What does the sentence import are all your pets name derick do?**

    It will import a dictionary of the same definition, previously defined.

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

    This function can be called with spam. bacon()

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

    We can process all code inside the try and except statement. When it encounters an error, the control is passed to the except block, skipping the code in between.

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

The **try clause** allows us to test a block of code for errors. The **except clause** permit us to handle the error. The finally it lets us execute code, regardless of the result of the try- and except blocks.

we use the try and except statements to handle exceptions. Whenever the code breaks down, an exception is thrown without crashing the program. Let's modify the add number program to include the try and except statements.

```
def addNumbers(a, b):
    try:
        return a + b
    except Exception as e:
        return 'Error occurred : ' + str(e)

print addNumbers('', 10)
```

As seen in the above code, we have moved our code inside a try and except statement. Try running the program and it should throw an error message instead of crashing the program. The reason for the exception is also returned as an exception message.