User-created Functions

Chapter 03a

Lesson Objectives

- Review Functions already used
- What are user-created functions?
- User-created Function Syntax
- return Statement
- Global and Local Variable Scope

What is a user-created function?

- A reusable section of code with a name that simplify something.
- Give the function a descriptive self explanatory name.

Why create functions?

- Code reuse
  - You are doing the same thing over and over again.
- Simplify your code
  - Functions have names to define what they do.
  - Breaks down complex blocks of code.
- Easier to make changes
  - If its only been written once, you only have to update it once.

You have already used functions!

- print()
- input()
- len()
- str()
- int()
- float()

User-created Functions

- A function is a **mini program** within your main program.
- User can use the function by calling it and **using it over and over** and in different places in the entire program.
- Code inside the Function will not be executed, unless the **function is called**.

FUNCTION SYNTAX:

```
def function_name():                    #descriptive function name
                                        #end line with colon
        indented block of code
        code
        code
        return                          #exit the function
```

```
def hello():                    #define function
        print('Howdy!')
        print('Howdy!!!')
        print('Hello there.')

hello()                         #calls the function

hello()                         #calls the function

hello()                         #calls the function
```

How do you call a function? – simply use its name

```
def printMessage():                    #define function
        print('Biba GCC')
        return                         #exits the function


printMessage()                         #calls the function
```

return Statements Syntax

Syntax:

return variable value or expression

Examples:

return <nothing after>

return ‘it is certain’

return message

return bill

return Statements Syntax

- If a function does not have a return statement, Python adds **return None** to the end of the function.
- If a function has a return statement with no variable value or expression, the function also **returns the value None**
- None – absence of a value.

If Function **needs data, pass parameters** into the function

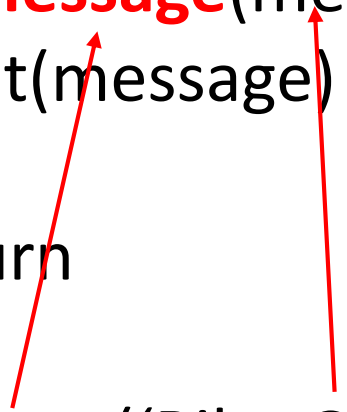
SYNTAX:

```
def function_name(parameters):      #use descriptive names
```

Example:

```
def printMessage(message)           #define function
    print(message)                  #message parameter passed into
                                    #function
    return                          #exits the function

printMessage('Biba GCC!')           #function is called
                                    #passes parameter to the function.
```



Function returns data using the keyword return

```
def tax(bill):
        bill = bill * 1.08                          (#or bill = bill * (bill *.08)
        print('Your bill is ', bill)
        return bill


meal_cost = 100

#calls the tax Function, passes the parameters to the function
meal_with_tax = tax(meal_cost)

#goes here after execution of return statement
```

Global and Local Scope: **Local**

- Variables assigned in a called function, exists in that function's **local scope** and are called a local variable. When the function returns, the local scope is destroyed, and the local variables are forgotten.
- Code in the **global scope** cannot use any local variable.
- Code in one function's local scope cannot use variables in any other local scope.
- You can use same name for different variables if they are in different scopes.

Global and Local Scope: **Global**

- Variables assigned outside all functions, exists in the **global scope** and are called a global variable.
- Local scope can access global variables.
- Code in the global scope cannot use any local variable.

```
#Parameters, a lower and upper bound number
#Returns the sum of the numbers from lower through upper

def summation(lower, upper):

        result = 0

        while lower <= upper:
                result = result + lower                    #variable for summing
                lower = lower + 1                          #counter


lower = int(input('Enter a lower bound number'))
upper = int(input('Enter a upper bound number'))

answer = summation(lower, upper)                           #calls the function
print(answer)                                              #What is the error?
```

```
#Parameters, a lower and upper bound number
#Returns the sum of the numbers from lower through upper

def summation(lower, upper):

        result = 0

        while lower <= upper:
                result = result + lower                    #variable for summing
                lower = lower + 1                          #counter
        return result

lower = int(input('Enter a lower bound number'))
upper = int(input('Enter a upper bound number'))

answer = summation(lower, upper)                           #calls the function
print(result)                                              #What is the error?
```

```
#Parameters, a lower and upper bound number
#Returns the sum of the numbers from lower through upper

def summation(lower, upper):

        result = 0

        while lower <= upper:
                result = result + lower                 #variable for summing
                lower = lower + 1                       #counter
        return result

lower = int(input('Enter a lower bound number'))
upper = int(input('Enter a upper bound number'))

answer = summation(lower, upper)                        #calls the function
print(answer)
```