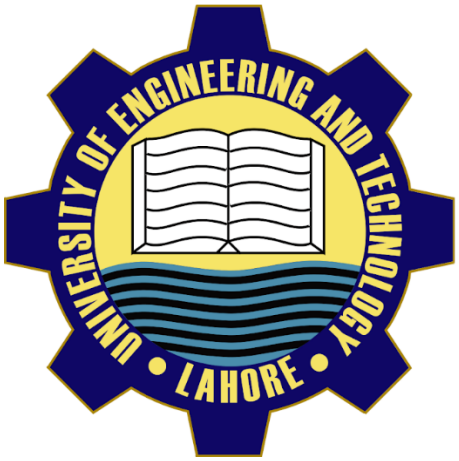
MCT-242 : COMPUTER PROGRAMMING-I

using Python 3.9



Prepared By:

Mr. Muhammad Ahsan Naeem

YouTube Playlist

https://www.youtube.com/playlist?list=PLWF9TXck7O_wMDB-VriREZ6EvwkWLNB7q

LAB 7: USER DEFINED FUNCTIONS: CLO 4

Suppose we want to display following table:

```
-----------------------------------
Most visited cities in the World
-----------------------------------
1    Bangkok    Thailand
-----------------------------------
2    London     UK
-----------------------------------
3    Paris      France
-----------------------------------
4    Dubai      UAE
-----------------------------------
```

It can be done as:

```
print('-'*35)
print('Most visited cities in the World')
print('-'*35)
print('1\tBangkok\tThailand')
print('-'*35)
print('2\tLondon\tUK')
print('-'*35)
print('3\tParis\tFrance')
print('-'*35)
print('4\tDubai\tUAE')
print('-'*35)
```

There are 6 dash-lines printed in above code. We can create our own function for this task and use it where required. The function is shown here:

```
def Line():
    print('-'*35)
```

Line is the name of the function chosen by us. It can by any name following the variable naming rules. From next line starts the body of the function with indentation. It is single line in above case but can be as many as required. Now where ever we want to print dashed line, we can use this function by its name as:

```
Line()
```

Using a function somewhere in program is termed as **Calling** the function. The complete code to display the above table using our own ceated function is given as under:

```
def Line():
    print('-'*35)
Line()
print('Most visited cities in the World')
Line()
print('1\tBangkok\tThailand')
Line()
print('2\tLondon\tUK')
Line()
print('3\tParis\tFrance')
Line()
print('4\tDubai\tUAE')
Line()
```

Is there any extra benefit of using function in above code?

Apparently, there is not much of the benefit of using above function for printing dashed line in terms of length of the code. But what if we want to change the length of dashed line to 33 instead of 35? In first case i.e. without using the function, we will have to change 35 to 33 at six places whereas in second case we need to do this at single place.

This is just a very simple and basic example of using functions. As we move along, will explore variety of benefits.

Before we move ahead, let's formulize two styling rules for using functions in our program:

i. In the first line of the block of any function, we will write the description of function within quotation marks. Python interpreter treats first line of any function block within quotes as comment.
ii. After ending the block of function (one or many), before starting the main logic of our program we will place a comment to indicate start of main program logic.

The above code will become something like:

```
def Line():
    'Displays a dashed-line'
    print('-'*33)
### Main Program Starts Below ###
Line()
print('Most visited cities in the World')
Line()
print('1\tBangkok\tThailand')
Line()
print('2\tLondon\tUK')
```

```
Line()
print('3\tParis\tFrance')
Line()
print('4\tDubai\tUAE')
Line()
```

Flow of execution of program having functions:

Program execution always begins at the first statement of the program. Statements are executed one at a time, in order from top to bottom. In case of function statements, it is different. The statements inside the function are not executed until the function is called.

A function call is like a detour in the flow of execution. Instead of going to the next statement, the flow jumps to the body of the function, executes all the statements there, and then comes back to pick up where it left off.

To understand this, let's explore the execution flow of above program shown here again with line numbers:

```
[1]     def Line():
[2]     'Displays a dashed-line'
[3]          print('-'*33)
[4]     ### Main Logic Starts Below ###
[5]     Line()
[6]     print('Most visited cities in the World')
[7]     Line()
[8]     print('1\tBangkok\tThailand')
[9]     Line()
[10]    print('2\tLondon\tUK')
[11]    Line()
[12]    print('3\tParis\tFrance')
[13]    Line()
[14]    print('4\tDubai\tUAE')
[15]    Line()
```

The first line which the interpreter will execute is line number **5** that is the first line of main program. At line **5** the function **`Line()`** is called so instead of moving to line number **6**, the interpreter will move to line number **3** which is the first line of the function block. There is no further statement after line **3** in function block, so interpreter will move to line number **5** and knowing that function has been executed it will move to line **6**. After executing line **6**, it will go to line number **7** where the function is called again and interpreter will move to line number **3** and then back to **7** followed by **8** and so on. The complete flow in terms of line numbers is as under:

5→3→5→6→7→3→7→8→9→3→9→10→11→3→11→12→13→3→13→14→15→3→15→ End of program

Functions with input arguments/parameters:

We can define a function with input argument(s). For example, in above case if we want to have function that can be used to print line of any character instead of fixing it to dashed line, we can control this by defining one input argument as shown here:

```
def Line(char):
    'Displays a character-line'
    print(char*33)
```

Here the function is defined with one input argument with name **`char`** which is simply a variable name to be used inside block of the function. To use this function, we will have to provide value of input parameter while calling the function. For example, if we want to print a line of asterisks, we will use the function as:

```
Line('*')
```

And it will display 33 asterisks. If we use it like:

```
Line('=')
```

And it will display 33 equality signs.

Now let's see the execution flow of the code below:

```
[1]     def Line(char):
[2]          'Displays a character-line'
[3]          print(char*33)
[4]     ### Main Program Starts Below ###
[5]     print('First Line')
[6]     Line('*')
[7]     print('Second Line')
[8]     Line('=')
[9]     print('Third Line')
[10]    Line('$')
```

The code executes as:

- Line number **5** is the first statement the interpreter will execute and the string **`First Line`** is displayed on the screen.
- Interpreter moves to line number **6** where the function is being called as **`Line('*')`**
- The interpreter moves to execute the function and assigns **`*`** to the variable **`char`** of the function definition. It executes line number **3** with **`char`** variable having the value **`*`**. So, 33 asterisks are displayed.
- Interpreter then moves to line number **6** followed by line **7** and displays the string **`Second Line`**.
- Interpreter moves to line number **8** where the function is being called as **`Line('=')`**

- The interpreter moves to execute the function and assigns equality sign `=` to the variable `char` of the function definition. It executes line number `3` with `char` variable having the value `=`. So, 33 equality signs are displayed.
- The interpreter then moves to line `8` and then to line `9` followed by line number `10`, where function is called again as `Line('$')`. So, it moves to execute the function and assigns dollar sign `$` to the variable `char` of the function definition. It executes line number `3` with `char` variable having the value `$`. So, 33 dollar signs are displayed.
- The interpreter then moves to line `10` and ends the code.

Different ways to assign input arguments:

Let's consider a different function now as shown here:

```
def printTwice(x):
    'prints the input text two times'
    print(x)
    print(x)
```

There is one input argument with name `x` and the function will simply print it two times. There can be different ways we may assign value to it. One way is directly assigning the value as:

```
printTwice('H')
```

It will assign `H` to variable `x` of the function and will display two `H`. The input string can be lengthy as shown here:

```
printTwice('Dil Dil Pakistan')
```

For this case variable `x` of the function will be assigned `Dil Dil Pakistan`

We can use some numeric value as well as:

```
printTwice(5)
```

For this case variable `x` of the function will be assigned `5` and `x` will be treated as integer numeric type instead of string.

We can also assign some constant value to input parameter as here:

```
printTwice(math.pi)
```

Finally, we can use a function and set the input argument value by a variable as shown here:

```
y='Computer Programming'
printTwice(y)
```

For this case variable `x` of the function will be assigned the value stored in variable `y` i.e. `Computer Programing`.

The important point here to note is that the function is defined with input variable `x` but can be used with any other variable. In fact, the function variables are **Local Variables** meaning that they exist only within that function. If we have a variable outside the function with the same name, for example, `x` for the above case, that will be different from the `x` of the function.

To understand this idea more clearly, let's check the following code:

```
[1]     def testFunction():
[2]          'A simple test function'
[3]          x=30
[4]          print(x)
[5]
[6]     #Main Program Starts Below
[7]     x=20
[8]     print(x)
[9]     testFunction()
[10]    print(x)
```

The code has following output:

```
20
30
20
```

The execution flow of the code is as:

- The first line of output, `20`, is output of line number `8` because variable `x` contains the value `20` assigned in line `7`.
- A function named as `testFunction` is being called on line number `9`, so the interpreter moves to the block of that function and executes line number `3` and assigns `30` to a variable `x` but this `x` is different from `x` outside this function.
- Line `4` is also part of the function block and it gets executed displaying the value `30`.
- The interpreter then moves to line number `9`, after finishing the function block and moves to line number `10` that prints the value of `x` which is printed as `20` because it is the `x` outside the function.
- To summarize, by calling a function at line number `9`, that has variable `x` in its block, has not changed the value of `x` defined at line number `7`, outside the function.

Tasks:

[1] Update the function `printTwice` so that it prints the input argument two times but on the same line separated by a white space. Then in main program, take a string input from user and use this function to print it twice on the screen.

Sample output is:

```
Enter any message: Mechatronics and Control
Mechatronics and Control Mechatronics and Control
```

[2] A function can have more than one input arguments. Create a function with two input arguments as `printTimes(s,n)` where `s` and `n` are those input arguments. `s` should be treated as string to be printed on screen and `n` is the number of times that string be printed. Take input the string and number of times user wants to print and display the output. Essentially, it is similar to first task but instead of printing twice now the string will be printed as many times as required.

Sample output is:

```
Enter any message: Mechatronics
Times to repeat: 3
Mechatronics Mechatronics Mechatronics
```

Functions with output Argument/Parameter

The above examples were simple displays of text. A function can be used to do calculations on input argument(s) and give the results as output argument. For this time, we will consider only one output argument of a function. For example, a simple function to calculate average of two numbers can be written as:

```
def avgOfTwo(a,b):
    'Calculates Average of two numbers'
    c=a+b
    mean=c/2
    return mean
```

Note following points:

- `avgOfTwo` is the name of the function.
- The function has two input arguments named as `a` and `b`.
- Inside function block is the calculation for the average of the `a` and `b`. Variable named as `c` is used to hold `a+b` and variable named as `mean` is used to store the mean value.
- The last line specifies which of the variable will be returned by the function as result.

Variable `c` is used to store `a+b` just to show that there can be as many variables inside function block as required and the variable specified in return statement will be the output of the function. We could have directly used `mean=(a+b)/2` or even we could have like:

```
def avgOfTwo(a,b):
    'Calculates Average of two numbers'
    return (a+b)/2
```

To use this function, we must consider that it is going to return (give) a value and that must be assigned to some variable. For example:

```
ans= avgOfTwo(5,6)
```

Note that when above statement is executed:

- Input argument **a** of function **avgOfTwo** gets the value **5** and **b** gets the value **6**.
- With above values in **a** and **b**, the expression **(a+b)/2** is evaluated and returned to the variable used to call the function i.e. **ans**

Again, the variables **a** and **b** of the function are local variable and have no existence outside that function. We can use direct values as shown above or any variables as input arguments while calling this function. For example:

```
def avgOfTwo(a,b):
    'Calculates Average of two numbers'
    c=a+b
    mean=c/2
    return mean
### Main Program Starts Below ###
x=eval(input('Enter first number:'))
y=eval(input('Enter second number:'))
print('Average of entered numbers is '+str(avgOfTwo(x,y)))
```

Tasks:

[3] Write a function that will find the distance between two points on XY Axis. Use this function in main program that will ask user to enter two XY points and will display the distance between them. The main program is given as under. You need to add the code for the function:

```
import math
def dist(x1,y1,x2,y2):
    'Calculates Distance between points (x1,y1) & (x2,y2)'
    #Complete Logic here
### Main Program Starts Below ###
x1,y1=eval(input('Enter 1st Point (Format: a,b) :'))
x2,y2=eval(input('Enter 2nd Point (Format: a,b) :'))
result=dist(x1,y1,x2,y2)
print('Distance between entered points is '+str(result))
```

[4] Write a function named as **triAreaSides** that will calculate the area of a triangle from three sides. Main program is given here, and you need to write the function:

```
### Main Program Starts Below ###
sideA=eval(input('Enter First side of triangle:'))
sideB=eval(input('Enter Second side of triangle:'))
sideC=eval(input('Enter Third side of triangle:'))
area=triAreaSides(sideA,sideB,sideC)
```

```
print('The Area of entered triangle is: '+str(area))
```

Extended Logic inside Function Block:

We have seen simple calculations within the function block. There can be decision based and other complex logics, that we will study later, inside the function block.

Check this function which decides largest of three numbers and its use in main program:

```
def biggestOfThree(a,b,c):
    'Returns the Largest of three numbers'
    if(a>=b):
        if(a>=c):
            big=a
        else:
            big=c
    else:
        if(b>=c):
            big=b
        else:
            big=c
    return big
### Main Program Starts Below ###
num1=eval(input('Enter First Number:'))
num2=eval(input('Enter Second Number:'))
num3=eval(input('Enter Third Number:'))
biggest=biggestOfThree(num1,num2,num3)
print('The Biggest of three numbers entered is: '+str(biggest))
```

We can also write the above function as:

```
def biggestOfThree(a,b,c):
    'Returns the Largest of three numbers'
    if(a>=b):
        if(a>=c):
            return a
        else:
            return c
    else:
        if(b>=c):
            return b
        else:
            return c
```

Tasks:

[5] We did a task of calculating the amount of electricity bill form the units consumed. It is given here again:

Electricity Bill Calculator: Electricity Bills are calculated on the basis of Units (KWhr) used. The LESCO tariffs for domestic users are given as:

Description	Rates
For first 100 Units	13.85 Rs/Unit
101-200 Units	15.86 Rs/Unit
201-300 Units	16.83 Rs/Unit
301-700 Units	18.54 Rs/Unit
Above 700 Units	20.94 Rs/Unit
Neelum Jhelum Surcharge NJ-Sur	0.1 Rs/Unit
Financial Cost Surcharge FC-Sur	0.43 Rs/Unit
TV Charges (Fix)	35 Rs
GST	12% of Calculated amount

Now you have to do the same task but using a function named as **`billElect(u)`** to calculate the amount of bill. In the main program you will be taking the number of units consumed, then use the function and simply display the result. All required calculations will be done in the function block. Also return the bill which will be charged after the due date.
The main code with function declaration is given as:

```
def billElect(u):
    'Calculates the electricity bill from u units'
    #Complete Logic here
### Main Program Starts Below ###
units=eval(input("Enter Electricity Units Consumed:"))
bill,bill2=billElect(units)
print(f'You bill is {bill} and After due date, bill is {bill2}')
```