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 1: First Program [CLO-1]

The first Program:

Displaying a message on the output screen:

To display a message on the output screen we use `print()` statement in Python. Here is the program to display `Hello World` on the screen.

```
print("Hello World!")
```

If you run the code, you will see the message displayed.

Syntax of Python:

Every computer language has its own set of rules to type in the statements/functions known as the syntax of that language very much like the grammar we have in human language e.g. rules of punctuations in English.

Syntax of print() function:

Let's observe the above statement in detail:

The function to print a message on the screen is `print()` which is a built-in function. Here the function name is print followed by brackets () inside which we have to write the message within double quotes " that we want to print. To type in Python programs, you have to consider these:

Case: Case matters. To Python, print, Print, and PRINT are all different things. For now, stick with lowercase as most Python statements are in lowercase. Try to run the following code:

```
Print("Hello World!")
```

And you will see that it will generate an error something like:

```
NameError: name 'Print' is not defined
```

White Space or Tabs: Spaces matter at the start of the line for proper indentation; something we will see in detail in coming lab sessions. But other than that, you can add spaces. Tab and Space work same for indentation in Python. We will be using Tabs instead of Space for the indentation but the choice is totally up to you. Following lines gives the same output (see carefully the different spaces/tabs):

```
print("Hello World!")
print    ("Hello World!")
print( "Hello World!")
print ("Hello World!" )
```

But of course, if we write space/tab within the double quotes " it become the part of the message and is printed on the screen. Try this:

```
print    ("   Hello World!")
```

Now re-try the same print function but with one space/tab at the start of the line as:

```
     print("Hello World!")
```

You will get an error of "Unexpected Indent".

Note→ *The general text in Programming Languages is known as* ***String****; something we will study in detail later. So, from now onward instead of calling it message, we will call it as* ***string*** *to be printed.*

More about print function

A few more useful features of the print function are given here:

- **Single Quote→**Instead of double quotation marks, we can write the string inside single quotes e.g. `print('Hello World!')`
- **New Line→**The print function will automatically advance to the next line. For instance, the following will print on two lines:

  ```
  print('Computer')
  print('Programming')
  ```

  If we write print command without any input string, it will create a new empty line. Try this:

  ```
  print('Computer')
  print()
  print('Programming')
  ```

- **Numbers→**If we want to print a number on the screen it can be printed without single or double quotes. For example, following two lines have same output:

  ```
  print(50)
  print("50")
  ```

  The output displayed on the screen is same but in first line `50` is taken as a number by Python and in second line it is taken as string. These different data types will be discussed in detail.

- **Mathematical Expressions→**We can write some mathematical expression without quotation marks and it will print the result of that expression. For example, the first line will print `5+5` as it is and the second will print the result i.e. `10`

  ```
  print("5+5")
  print(5+5)
  ```

  The details of different mathematical operators will be covered later.

- **Repeated String**→If you want to print repeated characters; for example, ten dollar signs `$`; of course the first direct approach is to write like:

```
print("$$$$$$$$$$")
```

But Pythons provides a better method to be used inside print function shown here:

```
print("$" * 10)
```

- **Concatenation**→Multiple strings can be concatenated/joined using a `+` or `,` sign. Both of these operators have certain advantages over the other. See the output of this line:

```
print('Computer','Programming')
```

You will see single string **`Computer Programming`** on the screen. Note that a space is inserted automatically in between two strings being concatenated. The white-space is not inserted using a `+` sign as shown here:

```
print('Computer'+'Programming')
```

Following line also has the same output as of **`print('Computer','Programming')`**

```
print('Computer' , 'Programming')
```

Because the spaces around `,` sign are ignored by the Python Interpreter as described earlier.

Now try this:

```
print('5+5=' , 5+5)
```

You will see this output:

```
5+5= 10
```

Again, there is space in between two parts being concatenated. If you don't want to display space before 10 in above output, the obvious choice is to use `+` symbol instead of `,` but if you try this you will get an error:

```
print('5+5=' + 5+5)
```

```
TypeError: can only concatenate str (not "int") to str
```

May be its difficult to understand the error statement at the moment but one simple reason is that the interpreter cannot differentiate whether to use **+** symbol for mathematical addition or the concatenation. We will explore the solution in next section.

Tasks:

[1] Display the following box (20 asterisks in one row) on the screen using the feature of repeated string display.

```
********************
********************
********************
********************
```

[2] Display the following triangle again using the feature of repeated string display:

```
*
**
***
****
```

[3] Display the following rectangle, again using the feature of repeated string display. For the spaces too you have to use the repeated feature. In second and third line of the box you have to use string concatenation using **+** symbol.

```
********************
*                  *
*                  *
********************
```

Using sep and end in string concatenation:

As we have seen that concatenating a string using **,** operator inserts a space in between strings being concatenated. We can control the separator using **sep** parameter of the **print()** function as shown here:

```
print('A','B','C','D',sep='--')
```

Here the **sep** property specifies that the separator between strings will be **--** instead of a white space. The output is shown here:

```
A--B--C--D
```

We can also use empty single or double quotes in **sep** parameter to specify that there will not be any separator between the strings as shown:

```
 print('A','B','C','D',sep="")
```

The output is:

```
ABCD
```

There is another parameter of **print()** function named as **end**. This is used to specify the ending character after concatenation as shown here:

```
print('AB','BC','CD','DE',end="$")
```

The out will be:

```
AB BC CD DE$
```

We can use both of above parameters together as well as shown here:

```
print('AB','BC','CD','DE',sep="--",end="$")
```

And the output will be:

```
AB--BC--CD--DE$
```

We can also use empty single or double quotes in **end** parameter just like in **sep**. The effect is that the print line will not end there and the next print will start from the same line. See this code:

```
print('A',end="")
print('B',end="")
print('C')
```

The output is:

```
ABC
```

Note→*The exact scenarios where we need such kind of displays will arise as we move along. At this stage, you are supposed to understand the use of these features.*

One more thing you should observe in case of using the **end** parameter is that the next **print** statement will print the value on the same line instead of new line. See this code:

```
print('ABC',end='$')
print('CDE')
```

This will be the output:

```
ABC$CDE
```

Now guess the output of this code and verify that:

```
print('ABC',end='$')
print('CDE')
print('FGH')
```