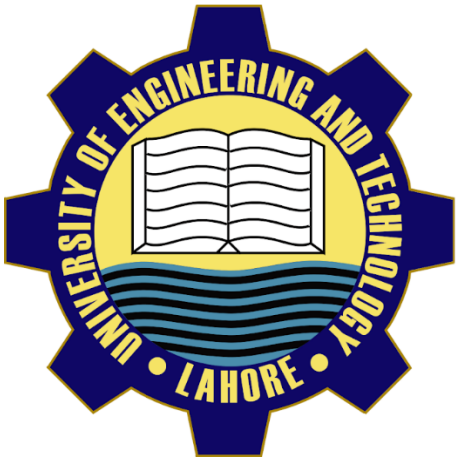
MCT-242 : COMPUTER PROGRAMMING-I

using Python 3.9



Prepared By:

Mr. Muhammad Ahsan Naeem

YouTube Playlist

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

Lab 2: Basic Input Output Streams: CLO 1

Escape Sequences:

Using the `print()` function if we try to print following string:

If you can't convince them confuse them!

The obvious solution is this line:

```
print('If you can't convince them confuse them!')
```

But you will get a **Syntax Error**. The reason is very obvious that the Python interpreter considers the apostrophe ' of can't as the ending ' of the input string. One solution could be to use double quote around string as:

```
print("If you can't convince them confuse them!")
```

This works but now the other problem is that using double " around string we cannot print " within the string itself, if any. The better solution is to use escape sequence. Python uses the backslash as an escape sequence in string literals. The escape sequence to print ' is `\'`. So, if we print the message like:

```
print('If you can\'t convince them confuse them!')
```

Now `\'` will tell the interpreter that this single ' is not the end mark of the string but the part of the string and you will see this output:

```
If you can't convince them confuse them!
```

You can find the list of Escape Sequences supported by python at: https://docs.python.org/3/reference/lexical_analysis.html#strings

And they are given here as well:

Escape Sequence	Name	Description	
		Command	Output
\\	Backslash (\)	`print('a\\b')`	`a\b`
\'	Single quote (')	`print('Python\'s')`	`Python's`
\"	Double quote (")	`print('\"Python\" ')`	`"Python"`
\n	ASCII Linefeed (LF)	`print('Hello\nWorld ')`	`Hello` `World`
\t	ASCII Horizontal Tab (TAB)	`print('Hello\tWorld ')`	`Hello World`
\ooo	Character with octal value ooo e.g. H has octal value of 110	`print('\110')`	`H`
\xhh	Character with hex value hh e.g. H has Hex value of 48	`print('\x48')`	`H`

Tasks

Most of these are repeated tasks of the last lab manual but now you have to do these using **escape sequences.**

[1] Display your complete name and registration number on two separate lines but using ONLY ONE `print()` statement.

[2] Display the following box (20 asterisks in one row) on the screen using ONLY ONE `print()` statement.

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

[3] Display the following triangle again using ONLY ONE `print()` statement.

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

[4] Display the following triangle again using ONLY ONE `print()` statement.

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

Data Types:

The data can be of many types. For example, your name consists of alphabets, age is a numeric value and your complete registration number is alphanumeric characters. Python has various standard data types but at this stage we will consider only two:

- String
- Numbers

String→ Strings in Python are identified as a contiguous set of characters represented in the quotation marks. Python allows either pair of single or double quotes. We have used strings in the last lab manual and the further detail will be covered later.

Number→ Python supports three different numerical types:

- int (signed integers e.g. 23, -40, 100 etc)
- float (floating point real values e.g. 12.4, -45.873 etc)
- complex (complex numbers e.g. 3+5j)

Variables:

Variables are used in computer programming to store values and use them when required very much like we do in algebra. In case of a computer program memory location is reserved against every variable. Based on the data type of a variable, the interpreter allocates memory. Therefore, by assigning different data types to the variables, you can store integers, decimals or characters in these variables.

We can define and store a value into a variable like:

```
x=10
```

Here `x` is the name of the variable and it is assigned a value `10`. Python interpreter automatically detects the type of the data and we need not specify that explicitly. Now if you want to print the value of `x` on screen, we can simply write:

```
print(x)
```

Note `x` is not inside quotation marks. If you will write `print('x')`, it will display `x` on the screen and not the value of `x`.

For assigning a value to a variable, `=` sign is used as shown above and this operator is known as **Assignment Operator**. Here are two important rules to use this operator:

- On the left side of operator `=` , we always have some variable. We cannot place constant there. Therefor `10=x` is invalid in almost every programming language although x=10 and 10=x are exactly same in algebra.
- On the right side of operator `=` , we can have direct value as shown above or some other variable or some mathematical expression that may involve one or more other variables. Therefore `z=w` means we are assigning the value of variable `w` to variable `z`. Likewise we can have `s=2+t` meaning that the value in variable `t` is added by `2` and assigned to `s`.

Rules of naming a variable:

In elementary algebra you have seen that variables are named using single alphabet. In programing languages variable name may contain more than one alphabet following the rules described below:

i. First character must be from alphabets a-z or A-Z or underscore (_).
ii. After first character you can use alphabets a-z or A-Z or underscore or digits 0-9.
iii. The names of variables are case sensitive.

Below are examples of legal variables:

```
Amount
Total_Amount
_Registration
Session2019
```

Below are examples of illegal variables:

```
2019session
#STUDENTS
```

Keywords:

It is always recommended to use the descriptive names for the variables, for example if you want to store the price of an item in a variable it is better to name it as **`price`** rather than **`x`**.

There are some reserved words known as keywords that cannot be used as name of variables even though they are according to the rules mention above. These keywords can be viewed by executing following commands on Python Shell:

```
>>> import keyword
>>> keyword.kwlist
```

Number Variable Examples:

Now check the output of following code:

```
x=20
print('The value in variable x is:',x)
```

We can always re-assign/update the value of variable defined as shown here:

```
x=20
x=50
print('The value in variable x is:',x)
```

There can be as many variables in a program as required and they can be used in any valid mathematical calculations. For example:

```
x=20.5
y=2*x
print('The value in variable x is:',x)
print('The value in variable y is:',y)
```

We can also do the mathematical calculations directly inside print function like:

```
x=120.7
print('The value in variable x is:',x)
print('Double of x is:',2*x)
```

We can assign a variable value directly from some other variable or from calculations based on some other variable. For example:

```
x=23.4
y=x
z=x+2
```

In above code **`x`** is assigned a value 23.4 in first line, then in second line **`y`** is assigned the value of **`x`** i.e. 23.4 and in third line **`z`** is assigned **`x+2`** which is 25.4

Now consider the two cases:

Case: 1

```
x=20
y=50
x=y
print(x,y)
```

Case: 2

```
x=20
y=50
y=x
print(x,y)
```

The only difference is that in first case we have `x=y` while the second one has `y=x`. Algebraically both `x=y` and `y=x` are same but in programming language they have different effect. In first case, `x=y` means value in `y` will be assigned to `x` without any change in the value of `y` and hence the output will be:

```
50 50
```

On the other hand, in second case `y=x` means the value of `x` will be assigned to `y` without any change in the value of `y` and hence the output will be:

```
20 20
```

Finally, another amazing feature in programming languages is that possibility of assigning a variable via a mathematical expression that involves the same variable. For example:

```
z=40
z=z+10
print(z)
```

If you run this code `z` will show the value 50.

List of basic mathematical operations that can be applied is given here:

Operator	Name	Example (For a=10 and b=3)	
		Command	Output
+	Addition	`print(a+b)`	13
-	Subtraction	`print(a-b)`	7
*	Multiplication	`print(a*b)`	30
/	Division	`print(a/b)`	3.3333333333333335
%	Modulus (Remainder Division)	`print(a%b)`	1
**	Exponent	`print(a**b)`	1000
//	Floor Division	`print(a//b)`	3
In some other languages, the caret, ^ is used for exponentiation but in Python it is a bitwise operator called XOR that we will explore later.			

Finally, there is an option of **Augmented Assignment** where instead of writing:

```
x=x+5
```

Python allows to write:

```
x+=5
```

Both of the above statements are equivalent and this feature is valid for all operators listed in above table.

String Variable Example:

The method to define and use string variable is similar to the number variables. As we have seen earlier the string variables are defined using single or double quotation mark. For example:

```
x="Computer Programing Lab"
```

The whole string is now stored in variable x. It can be used in print command as:

```
print(x)
```

And it will print the string on screen.

Now check this code:

```
name="Ali"
reg="2019-MC-125"
print("Your name is",name,"and your Registration Number is",reg)
```

Number as String:

Sounds a bit strange but a number can be stored as a string. For example:

y="100"

Although 100 is a number but it is treated as a string by the interpreter because of the double quotes. If we have to simply display it, that is fine but we cannot use it for mathematical calculations. For example, see these two codes:

`y=100` `print("Half of y is:",y/2)`	`y="100"` `print("Half of y is:",y/2)`

The code on the left works correctly but the one on right will generate error something like:

```
TypeError: unsupported operand type(s) for /: 'str' and 'int'
```

The error is very obvious that the operator `/` is not supported for string variables and `y` is defined as a string.

Taking Input from User:

So far, we have seen variables and used them for very simple calculations and displays. But even for the simplest application we have to take user's input. For example, in simple unit converter e.g. Degrees to Fahrenheit conversion for the temperature, the program must take the Degrees value from the user and then display the converted unit on the screen.

To take input from the user there is a simple function in Python named as `input()`.

String Input→If it is a string that we want to input from user, we can directly use this function as:

```
x=input("Enter a String Input: ")
```

if you run the above line you will see the message `Enter a String Input:` on the screen followed by a blinking cursor. Now user can enter any string and that will be stored in variable `x`. We can use that variable for any further task for example displaying on the screen.

See the code here:

```
name=input("Enter your Name: ")
reg=input("Enter your Registration No: ")
print("Hello",name,"your Registration Number is",reg,"\nWelcome to
Lab")
```

When you run this code, you will see the program will ask for the Name and the Registration No. and the last line will use the values entered by the user to display a message on the screen.

Number Input→ For the numbers input we have to use `eval()` function as shown here:

```
x = eval(input('Enter a number: '))
```

The number enter by the user will be stored in variable `x` that can be used for the further calculations. For example, see the following code to convert a Degrees temperature entered by the user into Fahrenheit.

```
temp=eval(input('Enter temperature in Degrees: '))
f=temp*9/5+32
print(temp,'Degrees = ',f,'Fahrenheit')
```

If user enters `37` as Degrees, this will be the output:

```
Enter tempearture in Degrees: 37
37 Degrees =  98.6 Fahrenheit
```

Note→ In above code we stored the value entered by the user in variable temp, calculated Fahrenheit temperature and stored in variable f. We could also do that directly like:

```
temp=eval(input('Enter temperature in Degrees: '))
print(temp,'Degrees = ',temp*9/5+32)
```

Using these functions now you can program any unit converter using the conversion formula. It is important to know the precedence of mathematical operators we have seen so far and it is given here from highest to lowest:

Operator (Highest to Lowest Precedence)	Name
()	Parenthesis
**	Exponentiation
/ // %	Division, Floor division, Modulus (Remainder)
*	Multiplication
+ -	Addition and subtraction

So, if variables `a`, `b`, `c` and d are defined in a program and we type:

```
z=a/b+c
```

It will first calculate `a/b` and then will add `c` into it. However, if we are interested to calculate $\frac{a}{b+c}$ we must use `()` and not any other bracket. We will have to write:

```
z=a/(b+c)
```

Tasks

[5] Write a program that asks the user to enter a numeric value and prints both the value and the double of that value.

Sample output is:

```
Enter any value you want: 2.6
You Entered: 2.6 Double of which is: 5.2
```

[6] Write a program that takes two values as input and shows the product of them at output.

Sample output is:

```
Enter 1st number: 2.3
Enter 2nd number: 1.5
The product of two numbers is: 3.45
```

[7] Write a program to find the area and volume of the sphere from its radius. The formulas of area and volume of sphere are: $A = 4\pi r^2$ and $V = \frac{4}{3}\pi r^3$.

Sample output is:

```
This program calculates area and volume of the sphere.
Enter the radius: 4
Area of the sphere is A: 201.14285714285714
Volume of the sphere is V: 268.1904761904762
```