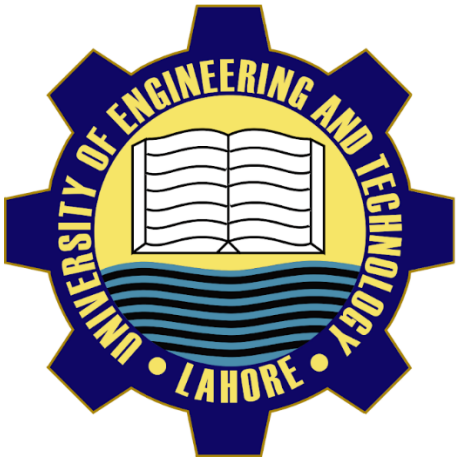
MCT-242 : COMPUTER PROGRAMMING-I

using Python 3.9



Prepared By:

Mr. Muhammad Ahsan Naeem

Lab 3: Mathematical Operations: CLO 1

Tasks:

We will start this lab session with the practice of the things we covered in the last lab session:

[1] Write a program that will take two integers as input and will display first integer as a percentage of other.

Sample output is:

```
Enter 1st integer: 20
Enter 2nd integer: 30
20 is 66.66666666666666% of 30.
```

[2] Write a program that will take a floating input from the user and will display the integer and the fractional part separately. *(Use data type casting to get the integer part and then subtract it from the original number to get the fractional part)*

Sample output is:

```
Enter a floating number: 2.34
Integer part is: 2
Fractional part is: 0.33999999999999986
```

Yes, there is little problem with the output that instead of 0.34, it is 0.33999999999999986 which is because of hardware limitation of computer. We will explore the solution for this in future.

[3] Write a program that will take a three-digit number as input and will display three digits separately in a column and show their sum. *(At present we can assume that user is sensible enough to actually enter a three-digits number. Later, in coming lab sessions we will see how to check if user has actually entered valid input and ask to re-enter if it is invalid)*

Sample output is:

```
Enter a 3-digit number: 564

 5
 6
 4

 5 + 6 + 4 = 15
```

How to do this:

Whenever you are given any program to write, firstly break that problem into simple parts and then try implementing each part. For example, in the above case first of all just think about how to take input from user. It's very simple as here:

```
x=eval(input('Enter a 3-digit number: '))
```

Now think about getting the three digits out of the 3-digit number stored in **x**. It's always better to consider some test value e.g. 597 is entered by user and is stored in **x**. We have to get 5, 9 and 7 separately in three other variables. The modulus or remainder division is helpful here. If we write:

```
a=x%10
```

you will get the unit digit in variable a i.e. 7 in case **x=597**. After the above line we have **x=597** and **a=7**.

Now to get the middle digit i.e. 9 first we have to get rid of 7 from **x** and that can be done as:

```
x=x//10
```

x gets updated to 59 after the above line. Again, using the remainder division and simple division we can separate 5 and 9 from **x=59**. Once you get all three digits, use the print function to display the output in the format shown in sample output. It is suggested to use single line print function with escape sequences to display the output.

For example, if **a**, **b** and **c** variable are for unit, tenth and hundredth digit, this can be a single print command:

```
print('\n',c,'\n',b,'\n',a,'\n',c,'+',b,'+',a,'=',c+b+a)
```

Comments in Program:

We can add comments to the lines of the program code. Comments are used to describe the logic you are applying there. Comments are important as in case of large program involving complex logic, it might be difficult for anyone or even yourself to understand the code later.

In Python comments are added using **#** symbol. Anything written after **#** is a comment and Python interpreter will simply ignore it. For example, you can see different comments are added in the last lab task code:

```
#A program for Lab Task
print('This program calculates area and volume of the sphere.')
r=eval(input('Enter the Radius: ')) #Taking input
A=4*22/7*r**2  #Formula for Area
V=4/3*22/7*r**3     #Formula for Volume
print('Area of the sphere is A: ',A)
print('Volume of the sphere is V: ',V)
```

It is not required to add comments on most of the lines. These are added on selective lines to explain the logic to the reader.

There is also a possibility to add **multiline comments** using three quotation marks (single or double) at beginning and ending as shown here:

```
"""
```

```
These lines
are ignored
by the interpreter
"""
```

The multiline comments come in very handy in at least these two situations:

- To add multiline description of your program, usually at the start of the program as shown here:

```
"""
Program: Lab 3 Task 2
Author: Ahmad Hassan
Date: 14/08/2019
"""
```

- Often you will want to modify your program but don't want to delete your old code in case your changes don't work. You can comment out the old code so that it is still there if you need it, and it will be ignored when your new program is running.

Numbers in Python:

We have used numbers data type in previous task. Here we will explore few more features in Python related to numbers.

Different Types of Numbers:

Numbers can be of different types. Python supports three different numerical types:

- int
- float
- complex

int is for integers both positive and negative e.g. 5, 3 or -20; **float** is for the floating real value both positive and negative e.g. 5.2, -2.3 and the **complex** is for the complex number e.g. 2+3j etc.

In most of the programming languages, it is required to specify the data type of the variable for example as `int x=10` rather than simply `x=10`. Moreover, within `int` they have different types for different ranges e.g. `short`, `int`, `long int`, `long long int` etc. Luckily, in Python we do not need to worry about all this. All we need is to assign the value to a variable and Python interpreter sets its data type all by itself depending upon the value entered. So, if we write:

```
x=10
```

The Python interpreter will declare `x` as `int` internally. And for the case:

```
y=10.1
```

the Python interpreter will declare `y` as `float` internally.

We will explore complex numbers soon in coming lab sessions.

Data Type Casting/Conversion:

Number Castings:

Some time it is required to convert one type of number into other e.g. converting `float` to `int`. This can be done using **Type Conversion** functions as shown here:

```
x=10.23
y=int(10.23)#Or we could write y=int(x)
print (x)
print(y)
```

The output of the above code is as:

```
10.23
10
```

The `y` is changed to an integer because of `int()` function. Likewise, we have `float()` function to convert a number into float as shown here:

```
x=10
y=float(10)
print(x)
print(y)
```

The output of the above code is as:

```
10
10.0
```

In Task 2 of this lab session now you can use `int()` function to get a proper integer value.

Multiple Assignment of Variables

Python allows range of options for multiple assignments to variable. Such options are generally not available in most of the languages. For example, if you want to assign a value say 5 to three variables, you can do it like:

```
a=b=c=5
print(a,b,c)
```

The output shows that 5 is assigned to all three variables as shown here:

```
5 5 5
```

Now consider this case:

```
x=5
a=10
c=8
a=b=c=x
```

```
print(a,b,c)
```

Here the value of **x** will be assigned to the other three variable and their previous values will be overwritten by the new. The output is shown here:

```
5 5 5
```

There is also the possibility of assigning different values to different variables in one line as shown here:

```
x,y=100,200
print(x,y)
```

The output is:

```
100 200
```

Even we can assign values of different data types using above method as shown here:

```
x,y,z=100,200,"Pakistan"
print(x,y,z)
```

So here **x** and **y** are of integer data type and **z** is string. The output is:

```
100 200 Pakistan
```

Swapping the values of two variables:

Suppose two variables are defined as:

```
a=50
b=-100
```

If you want to swap the values of the two variables, a common mistake is using the following logic:

```
a=b
b=a
```

This will not swap the values. In first line value of **b** is assigned to **a** making it **-100** and the value of **b** itself remain unchanged to **-100**. So, both variables have value **-100** and in second line value of **a** i.e. **-100** is assigned to **b** so effectively both variables have value **-100** after above code.

The solution is to use another variable to store the copy of **a** before the first line of the code as shown here:

```
x=a  # x gets the value 50
a=b  # a gets the value -100
b=x  # b gets the value 50
```

This is what we usually do in any programming language. But Python offers an amazing solution for this assignment as shown here:

```
a,b=b,a # values swapped
```

Run the following code to see the effect:

```
a=50
b=-100
print('Before Swapping')
print('a=',a,'b=',b)
a,b=b,a # values swapped
print('After Swapping')
print('a=',a,'b=',b)
```

Not just swapping between two variables, in Python we can swap multiple variable as shown here:

```
x,y,z=10,20,30
x,y,z=z,x,y
print(x,y,z)
```

Multiple Inputs from user:

We can take multiple inputs from user using multiple input statements but Python also allows to take multiple inputs in single statement. Taking multiple strings inputs will be considered later and at present we will consider multiple numeric input only. Here is the code to take in two variables `x` and `y` from user in single statement:

```
x,y=eval(input("Enter x and y: "))
print("x =",x,"y =",y)
```

The sample output is:

```
Enter x and y: 456,2.3
x = 456 y = 2.3
```

Carefully note the way two inputs are entered. A better approach will be to guide the user about the format to enter multiple values as:

```
x,y=eval(input("Enter x and y separated by commas: "))
print("x=",x,"y=",y)
```

Formatted Strings (fstring):

We have to display the results in each program and we use print function and pass in the message. But most of time, the results are stored in different variables which we have to incorporate inside the display message using `+` or `,` operator. Many times, that becomes quite cumbersome to write so many string concatenations.

We can use the formatted string concept for this. Formatted string is also known as **f-String**. Let's say we have two variables having some integer stored as:

```
a=5
b=20
```

If we want to print their values in this format:

Value of a is 5 and value of b is 20.

Using the string concatenation format, we will do this like:

```
print("Value of a is",a,"and value of b is",b,".")
```

In case of f-string, we do not need concatenations. All we need is to put the variable name inside curly bracket `{}` and interpreter will take the value from program variable. But to declare a f-string we need to write `f` before starting quotation marks of the string.

So the above print statement with f-string will be like:

```
print(f"Value of a is {a} and value of b is {b}.")
```

Math Module:

We have used the functions `print()`, `input()`, `eval()` etc. There are many other built-in functions of Python and their use will be covered as we proceed in the course. The list can be checked here:

https://docs.python.org/3/library/functions.html

Other than built-in functions we may require more for different functionality that is not available as built-in. For example, we may require to calculate square-root, logarithm, or trigonometric function of some number and we don't have any built-in function for them. For that, we will use **Math Module**.

What is Module:

Module can be considered as a code library in the form of a file containing a set of functions you want to include in your application. In coming lab sessions, we will be creating our own modules as well but at this moment we will see how to include and use available modules. For example, we have **Math Module** that contains numerous math functions. To see the list of functions available in Math Module, you can write the code on Python Shell as:

```
>>>import math
>>>dir(math)
```

As Python program you can write:

```
import math
print(dir(math))
```

This gives a list of all the functions and variables in the math module. You can ignore all of the ones that start with double-underscores. To get help on a specific function, say the sqrt function, you can type:

```
>>>help(math.sqrt)
```

Typing help(math) will give you help for everything in the math module. The detail is given in Python Documentation:
https://docs.python.org/3/library/math.html

Let's see how to use square root function of Math Module:

```
from math import sqrt
x=eval(input("Enter a number: "))
s=sqrt(x)
print(f"The Square root of {x} is {s}")
```

What if you want to use two or more functions from one module?

Let's see if we want to display square root and the exponent e^x for the input number x. The function for exponent in math module is **`exp()`**.

```
from math import sqrt,exp
x=eval(input("Enter a number: "))
print(f"The Square root of {x} is {sqrt(x)}.")
print(f"The Exponent of of {x} is {exp(x)}.")
```

There is also the possibility that we don't mention the function of the module and simple import only the module and then you can any of the functions in that module as required. The syntax is a bit different as shown here:

```
import math
x=eval(input("Enter a number: "))
print(f"The Square root of {x} is {math.sqrt(x)}")
print(f"The Exponent of of {x} is {math.exp(x)}")
print(f"The SINE of of {x} is {math.sin(x)}")
```

See carefully how **`sqrt`** is used now. Instead to **`sqrt(x)`** now we have to write **`math.sqrt(x)`**.

Finally there is also possibility of importing all functions of a module and then we will not to write that like **`math.sqrt(x)`** and will be able to use any function simply as **`sqrt(x)`**. For that we use **`import *`** as shown here:

```
from math import *
x=eval(input("Enter a number: "))
print(f"The Square root of {x} is {sqrt(x)}")
print(f"The Exponent of of {x} is {exp(x)}")
print(f"The SINE of of {x} is {sin(x)}")
```

Tasks:

[4] A Cultural Musical Concert is being held in **Al-hamra Complex**. There are three seating categories in the hall. **Class A** gallery seats cost **Rs. 1000**, **Class B** gallery seats cost **Rs. 700** and **Class C** gallery seats cost **Rs. 500.** Write a program that asks how many tickets for each class of seats were sold, then displays the amount of income generated from ticket sales.

Sample output is:

```
How many tickets sold for class A: 16
```

```
How many tickets sold for class B: 21
How many tickets sold for class C: 27

Total income generated from tickets is: 44200
```

[5] Write a program that will take three sides of the triangle as input and will display the area of the triangle at the output using **hero's formula**. If `a`, `b` and `c` are sides of triangle, area of the triangle is:

$$Area = \sqrt{s(s-a)(s-b)(s-c)}$$

$$where: s = \frac{a+b+c}{2}$$

We will assume that user will enter the sides of valid triangle. A triangle is valid if sum of any of its two sides is greater than the third side. Later, we will see how to handle the case if entered triangle is invalid.

Sample output is:

```
Enter the three sides of triangle (separated by commas): 4,3,2
The area of entered triangle is:  2.9047375096555625
```