Lab-03: Python programming (loops, functions, classes)

3.1 Loops in Python:

3.1.1 The 'while' loop

```
a = 0

while a < 10:

    a = a + 1

    print (a )
```

3.1.2 Range function:

Range(5) #[0,1,2,3,4]

Range(1,5) #[1,2,3,4]

Range(1,10,3) #[1,4,7]

3.1.3 The 'for' loop

```
for i in range(1, 5):

      print (i )


for i in range(1, 5):

      print (i)

else:

print ('The for loop is over')
```

3.2 Functions:

3.2.1 How to call a function?

function_name(parameters)

Code Example - Using a function

```
def greet():   #function definition

   print("Hello")

   print("Good Morning")
```

```
greet()    #function calling


def add_sub(x,y)
   a=x+y
   b=x-y
   return a,b
result1, result2 = add_sub(5,10)
print(result1, result2)



def multiplybytwo(x):
   return x*2
a = multiplybytwo(70)
```

The computer would actually see this:

a=140

3.2.2 Define a Function?

def function_name(parameter_1,parameter_2):

{this is the code in the function}

return {value (e.g. text or number) to return to the main program}

3.2.3 range() Function:

If you need to iterate over a sequence of numbers, the built-in function range() comes in handy. It generates iterator containing arithmetic progressions:

```
>>> range(10) [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
```

It is possible to let the range start at another number, or to specify a different increment (even negative; sometimes this is called the 'step'):

```
>>> list(range(5, 10))
[5, 6, 7, 8, 9]
>>> list(range(0, 10, 3) )
[0, 3, 6, 9]
>>> list(range(-10, -100, -30) )
```

```
[-10, -40, -70]
```

The range() function is especially useful in loops.

3.3 Classes & Inheritance:

The word 'class' can be used when describing the code where the class is defined.
A variable inside a class is known as an *Attribute*
A function inside a class is known as a *method*

- A class is like a
  - Prototype
  - Blue-print
  - An object creator
- A class defines potential objects
  - What their structure will be
  - What they will be able to do
- Objects are instances of a class
  - An object is a container of data: attributes
  - An object has associated functions: methods

3.3.1 Syntax:

```
# Defining a class
class class_name:
[statement 1]
[statement 2]
[statement 3] [etc]
```

3.3.2 Inheritance Syntax:

```
class child_class(parent_class):
        def __init__(self,x):
                # it will modify the _init_ function from parent class
        # additional methods can be defined here
```

‘self’ keyword:

The first argument of every class method, including __init__, is always a reference to the current instance of the class. By convention, this argument is always named self. In the __init__ method, self refers to the newly created object; in other class methods, it refers to the instance whose method was called.

Anytime you create an object of the class, the first thing it does before going on to any other line is it looks for init function and it calls whatever is written in here. You don’t have to call it explicitly like any other function

Example1:

```
class MyClass:
   i = 12345
   def f(self):
           return 'hello world'
x = MyClass()
print (x.i)
print (x.f())
```

Example2:

```
class Complex:
    def __init__(self, realpart, imagpart):
        self.r = realpart
        self.i = imagpart
x = Complex(3.0, -4.5)
print (x.r,"      ",x.i )
```

Example3:

```
class Shape:
    def __init__(self,x,y): #The __init__ function always runs
first
        self.x = x
        self.y = y
    description = "This shape has not been described yet"
    author = "Nobody has claimed to make this shape yet"

    def area(self):
        return self.x * self.y
    def perimeter(self):
        return 2 * self.x + 2 * self.y
    def describe(self,text):
        self.description = text
    def authorName(self,text):
        self.author = text
    def scaleSize(self,scale):
        self.x = self.x * scale
        self.y = self.y * scale

a=Shape(3,4)
print (a.area())
```

Inheritance Example:

```
class Square(Shape):
   def __init__(self,x):
         self.x = x
         self.y = x

class DoubleSquare(Square):
   def __init__(self,y):
         self.x = 2 * y
         self.y = y
   def perimeter(self):
         return 2 * self.x + 2 * self.y
```

3.4 Module:

A module is a python file that (generally) has only definitions of variables, functions, and classes.

Example: Module name mymodule.py

```
# Define some variables:
ageofqueen = 78

# define some functions
def printhello():
    print ("hello")
# define a class
class Piano:
    def __init__(self):
        self.type = input("What type of piano?: ")
        self.height = input("What height (in feet)?: ")
        self.price = input("How much did it cost?: ")
        self.age = input("How old is it (in years)?: ")

    def printdetails(self):
        print ("This piano is a/an " + self.height + " foot")
        print (self.type, "piano, " + self.age, "years old and costing " +
self.price + " dollars.")
```

3.4.1 Importing module in main program:

```
### mainprogam.py ##
# IMPORTS ANOTHER MODULE

import mymodule
print (mymodule.ageofqueen )
cfcpiano = mymodule.Piano()
cfcpiano.printdetails()
```

Another way of importing the module is:

```
from mymodule import Piano, ageofqueen
print (ageofqueen)
cfcpiano = Piano()
cfcpiano.printdetails()
```

3.5 TASKS:

3.5.1 LAB TASK 1:

Write a program to find the largest of ten numbers provided by user, using functions.

3.5.2 LAB TASK 2:

Create a class name basic_calc with following attributes and methods;

Two integers (values are passed with instance creation)

Different methods such as addition, subtraction, division, multiplication