LAB # 2:

OBJECT ORIENTED DESIGN PRACTICES IN PYTHON

Objectives:
- To study object oriented design practices in python
- Applying object oriented constructs to create hierarchical data structures

Hardware/Software required:
Hardware: Desktop/ Notebook Computer
Software Tool: Python 2.7/ 3.6.2

Introduction:
Python supports object oriented programming constructs that can be utilized to create sophisticated codebase employing decent data structures. In this lab, different object oriented programming constructs will be revised and they will be used to create different algorithms employing different data structures.

Lab Tasks:

1. Classes

Classes are the fundamental constructs in object oriented paradigms and they are used to hold methods and variables related to common entity. All the attributes within the class can be accessed through its instance or object. Following examples shows the usage of classes:

Example 1:

```python
class MyClass:

    i = 12345

    def f(self):

        return 'hello world'

x = MyClass()

print x.i

print x.f()
```

Example 2:

```python
class Shape:

    def __init__(self,x,y): # Constructor
```

```
    self.x = x

    self.y = y

  description = "This shape has not been described yet"

  author = "Nobody has claimed 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()
```

2. Inheritance

All classes have a property that they can inherit from other classes. This is one of the fundamental concept of object oriented design practices. Python also supports inheritance. Following code snippet describes how you can inherit from another classes:

```
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): # Method Overriding

        return 2 * self.x + 3 * self.y
```

3. Modules

A module allows you to logically organize your Python code. Grouping related code into a module makes the code easier to understand and use. A module is a Python object with arbitrarily named attributes that you can bind and reference.

Simply, a module is a file consisting of Python code. A module can define functions, classes and variables. A module can also include runnable code. For example:

Define a module .py file and write a following code script.

```
# Define a variable:

Age = 78

# Define a method

def Print():

    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.")
```

Now make a main .py file and import the module:

```
import module


print (module.Age)

module.Print()

o=module.Piano()

o.PrintDetails()
```

3. Trees

Python does not the in-built support for trees. So, we would be utilizing classes in order to create trees. For example:

```
class Tree():

    def __init__(self):

        self.left = None

        self.right = None

        self.data = None


root = Tree()

root.data = "root"

root.left = Tree()

root.left.data = "left"

root.right = Tree()

root.right.data = "right"
```

```
root.left.left = Tree()

root.left.left.data = "left 2"

root.left.right = Tree()

root.left.right.data = "left-right"


print(root.left.left.data)
```

Lab Evaluation:

Q1: Create a class named 'Complex' that must have the following attributes:
>   Variables named 'Real' and 'Imaginary'
>   Methods named Magnitude() and Orientation()

Take a complex number from user in main and print its magnitude and orientation. You have a liberty to create methods signature as you like.

Q2: Use the previously created 'Complex' class and compute an 8-point Discrete Fourier Transform (DFT) of the following real-valued discrete sequence:

$$x = [2, 2, 2, 2, 2, 2, 2, 2, 2]$$

After computing DFT, print the magnitude and phase spectra. Tell the time performance of your algorithm in Big-O Notation. DFT can be computed as:

$$X[k] = \sum_{n=0}^{M-1} x[n]e^{-j\frac{2\pi}{N}kn}$$

where M is the length of input sequence, k ranges from 0 to $N-1$ and $N = 8$ here.

Q3: Create the following Binary Search Tree and search for the node '13'. You can hard code the tree as well but it's better if you create it dynamically at run time (You must have learned in Data Structures & Algorithms). Also, tell the time performance of searching the node '13' in Big-O notation.
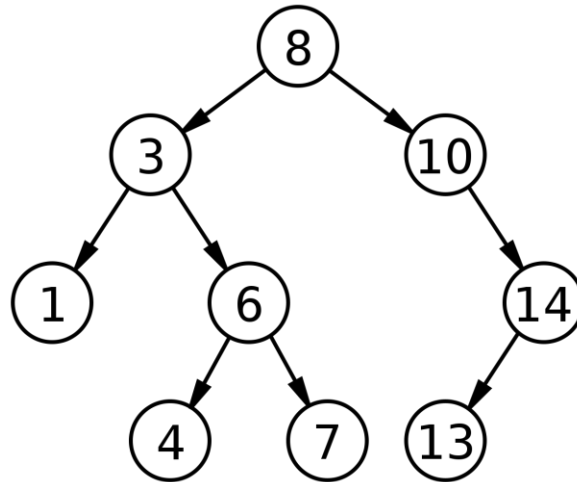
Conclusion:
Write the conclusion about this lab

NOTE: A lab journal is expected to be submitted for this lab.