LABORATORY 3 : Python OOP and Turtle Graphics

OBJECTIVES

- to see how OOP concepts are implemented in Python
- to introduce a turtle module
- to draw with turtle

BACKGROUND

❖ Object-oriented programming (OOP) is a programming paradigm that revolves around objects.

Principles of OOP

1. Encapsulation
2. Inheritance
3. Polymorphism

Encapsulation

Class : a blueprint for how to build a certain type of object. Class describes attributes and behavior of objects instantiated from the class. Python uses the keyword `class` to define a class.

Object : an instance of a class

Constructor : a special that creates an instance of the class. Constructor initializes values of attributes of an object. In Python, constructor of a class is defined as `__init__`.

Example:

`Point` class and how `Point` objects are created and used (on Python console).

```
class Point:
    """ Create a new Point, at coordinates x, y """

    def __init__(self, x=0, y=0):
        """ Create a new point at x, y """
        self.x = x
        self.y = y

    def distance_from_origin(self):
        """ Compute my distance from the origin """
        return ((self.x ** 2) + (self.y ** 2)) ** 0.5
```

```
>>> p = Point(3, 4)
>>> p.x
3
>>> p.y
4
>>> p.distance_from_origin()
5.0
>>> q = Point(5, 12)
>>> q.x
```

```
5
>>> q.y
12
>>> q.distance_from_origin()
13.0
>>> r = Point()
>>> r.x
0
>>> r.y
0
>>> r.distance_from_origin()
0.0
```

Inheritance

Inheritance is the ability to define a new class that is an extension of an existing class. The new class (derived class) inherits all of the methods of the existing class (base class).

Example :

`Person` class (base class), `Employee` class (derived class) and `Boss` class (derived class). The example shows how derived class object is initialized—how constructors are called.

```
# ----- Parent class: person --------
class Person:
    def __init__(self,first,last):
        self.firstName = first
        self.lastName = last

    def getName(self):
        return self.firstName + " " + self.lastName


# ----- child class: Employee inherited from person --------
class Employee(Person):
    def __init__(self, first, last, id):
        super().__init__(self, first, last)
        self.id = id

    def getEmployee(self):
        return self.getName() + ", ID : " + self.id


# ----- child class: Boss inherited from person --------
class Boss(Person):
    def __init__(self, first, last, title):
        super().__init__(self, first, last)
        self.title = title

    def getBoss(self):
        return self.getName() + ", title : " + self.title

# ----- instantiate class --------
```

```
person_1 = Person("Bart", "Simpson")
employee_1 = Employee("Fred", "Flintstone", "12345")
employee_2 = Employee("Barney", "Rubble", "12346")
boss_1 = Boss("Dino", "Flintstone", "CEO")

# ------ print out information ------
print(person_1.getName())
print(employee_1.getEmployee())
print(employee_2.getEmployee())
print(boss_1.getBoss())
```

Output :

```
Bart Simpson
Fred Flintstone, ID : 12345
Barney Rubble, ID : 12346
Dino Flintstone, title : CEO
```

Polymorphism

Methods usually work for a specific type of objects. When a new class is defined, methods that operate on objects of that class are included in the class.

There is another type of operations that work for many types of objects. Each object type responds to the operation differently.

Example :
`Bear` and `Dog` classes. Both inherit from the built-in `Object` class. A bear and a dog are capable of making sound. However, their sounds are different. The example shows how polymorphism is implemented in Python.

```
class Bear(object):
    def sound(self):
        print("Groarrr")


class Dog(object):
    def sound(self):
        print("Woof woof!")


def makeSound(animalType):
    animalType.sound()


bearObj = Bear()
dogObj = Dog()

makeSound(bearObj)
makeSound(dogObj)
```

Output :

```
Groarrr
Woof woof!
```

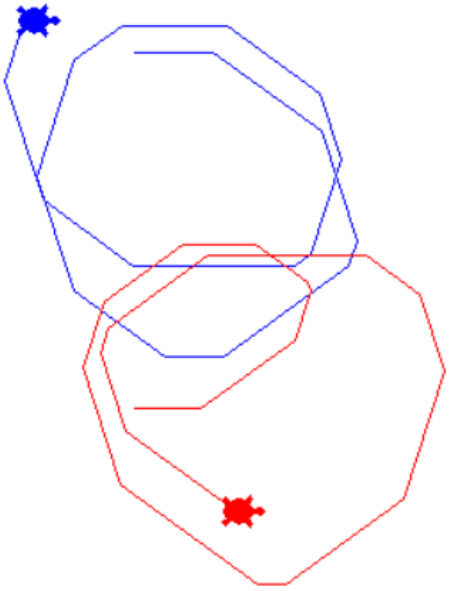
❖ Turtle Graphics

There are many Python packages that can be used to create graphics and GUI's. Two graphics modules, called turtle and tkinter, come as a part of Python's standard library. tkinter is primarily designed for creating GUI's. turtle is primarily used as a simple graphics package. It also can be used to create simple GUI's.

The turtle module is an implementation of turtle graphics and uses tkinter for the creation of the underlying graphics.

Among other things, the methods in the turtle module allow us to draw images. The idea behind the turtle part of "turtle graphics" is based on a metaphor. Imagine you have a turtle on a canvas that is holding a pen. The pen can be either up (not touching the canvas) or down (touching the canvas). Now think of the turtle as a robot that you can control by issuing commands. When the pen it holds is down, the turtle leaves a trail when you tell it to move to a new location. When the pen is up, the turtle moves to a new position but no trail is left. In addition to position, the turtle also has a heading, i.e., a direction, of forward movement. The turtle module provides commands that can set the turtle's position and heading, control its forward and backward movement, specify the type of pen it is holding, etc. By controlling the movement and orientation of the turtle as well as the pen it is holding, you can create drawings from the trails the turtle leaves.
(*ref: Algorithmic Problem Solving with Python, John B. Schneider, Shira Lynn Broschat, and Jess Dahmen, www.eecs.wsu.edu/~schneidj/swan, 2013.*)

Below is a drawing created from traced paths of two turtles doing series of a random distance forward move + a 36 degree turn.



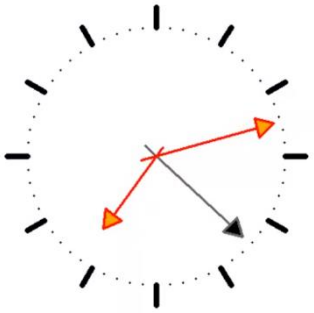
LABORATORY 3: Pre-lab

1. Study Goodrich's Chapter 2 thoroughly.
2. Read about `turtle` from https://docs.python.org/3/library/turtle.html.
3. Read about `tkinter` from https://docs.python.org/3/library/tk.html.

LABORATORY 3: In-lab

1. Work on all the items in the tutorial
   https://www.tutorialspoint.com/turtle-programming-in-python
   https://www.tutorialsandyou.com/python/turtle-programming-tutorial-in-python-5.html
   Make sure that you understand how to control `turtles`.
2. Create an analog clock (design your own clock face) with Python turtle. Prompt for a time (hh:mm:ss). Then draw clock hands according to the time given by the user.

   This clock shows time at 19:12:22.

   
3. Write a test plan to test your clock. Verify correctness and completeness of your clock against the test plan.

LABORATORY 3: Post-lab

1. Create a class, turtle 4.0 or `TurtleFPO`, that leaps forward (or backward) by a certain distance each time. Each `TurtleFPO` turtle has limited energy. Energy is reduced each time when turtle moves. The longer the move, the more energy consumed. `TurtleFPO` turtle stops moving when its energy runs out.
2. Create a few turtles (maybe a mix of normal `Turtle`'s and `TurtleFPO`'s). Give them different colors, speeds and leap distances. You may consider using random number generator to make the race less predictable.
3. Have the turtles race from one side of the Screen to the other side. Enjoy turtle race !!
4. Write a test plan for your turtle race. Test your race according to the plan.

Submission:

Due dates:

pre-lab and in-lab	:	by the end of lab period, Wednesday Aug 17
post-lab	:	by the end of lab period, Tuesday Aug 23

You are to review your work with the TAs during lab period.