CMPE 30052
LAB ACTIVITY #3
PYTHON'S CLASSES AND OBJECTS

1. (Geometry: n-sided regular polygon) An n-sided regular polygon's sides all have the same length and all of its angles have the same degree (i.e., the polygon is both equilateral and equiangular). Design a class named RegularPolygon that contains:

■ A private int data field named n that defines the number of sides in the polygon.
■ A private float data field named side that stores the length of the side.
■ A private float data field named x that defines the x-coordinate of the center of the polygon with default value 0.
■ A private float data field named y that defines the y-coordinate of the center of the polygon with default value 0.
■ A constructor that creates a regular polygon with the specified n (default 3), side (default 1), x (default 0), and y (default 0).
■ The accessor and mutator methods for all data fields.
■ The method getPerimeter() that returns the perimeter of the polygon.
■ The method getArea() that returns the area of the polygon. The formula for

$$\text{computing the area of a regular polygon is } Area = \frac{n \times s^2}{4 \times \tan\left(\frac{\pi}{n}\right)}.$$

Write a test program that creates three RegularPolygon objects, created using RegularPolygon(), using RegularPolygon(6, 4) and RegularPolygon(10, 4, 5.6, 7.8). For each object, display its perimeter and area.

2. Write code that defines a class named Animal:
   * Add an attribute for the animal name.
   * Add an eat() method for Animal that prints ``Munch munch.''
   * A make_noise() method for Animal that prints ``Grrr says [animal name].''
   * Add a constructor for the Animal class that prints ``An animal has been born.''

A class named Cat:
   * Make Animal the parent.
   * A make_noise() method for Cat that prints ``Meow says [animal name].''
   * A constructor for Cat that prints ``A cat has been born.''
   * Modify the constructor so it calls the parent constructor as well.

A class named Dog:
   * Make Animal the parent.
   * A make_noise() method for Dog that prints ``Bark says [animal name].''
   * A constructor for Dog that prints ``A dog has been born.''
   * Modify the constructor so it calls the parent constructor as well.

Create a test program that will:

* Code that creates a cat, two dogs, and an animal.
* Sets the name for each animal.
* Code that calls eat() and make_noise() for each animal. (Don't forget this!)

3. (The Stock class) Design a class named Stock to represent a company's stock that contains:

■ A private string data field named symbol for the stock's symbol.
■ A private string data field named name for the stock's name.
■ A private float data field named previousClosingPrice that stores the stock price for the previous day.
■ A private float data field named currentPrice that stores the stock price for the current time.
■ A constructor that creates a stock with the specified symbol, name, previous price, and current price.
■ A get method for returning the stock name.
■ A get method for returning the stock symbol.
■ Get and set methods for getting/setting the stock's previous price.
■ Get and set methods for getting/setting the stock's current price.
■ A method named getChangePercent() that returns the percentage changed from previousClosingPrice to currentPrice.

Write a test program that creates a Stock object with the stock symbol INTC, the name Intel Corporation, the previous closing price of 20.5, and the new current price of 20.35, and display the price-change percentage.

4. Implement a class ComboLock that works like the combination lock in a gym locker. The lock is constructed with a combination—three numbers between 0 and 39. The reset method resets the dial so that it points to 0. The turnLeft and turnRight methods turn the dial by a given number of ticks to the left or right. The open method attempts to open the lock. The lock opens if the user first turned it right to the first number in the combination, then left to the second, and then right to the third.

class ComboLock :

```
    def ComboLock(self, secret1, secret2, secret3) :

    def reset(self) :
     . . .
    def turnLeft(self, ticks) :
     . . .
    def turnRight(self, ticks) :
     . . .
     def open(self) :
```