Last Name: _____ First Name: _____ Cornell Netid: _____

Questions 2 and 3 from the CS 1110 Final December 7th, 2012

Adapted to the terminology and conventions of Spring 2013

Classes and Subclasses

In Assignment 7 you got experience with `GRectangle` and `GEllipse` which extended `GObject`. This question deals with three very similar classes: `Shape`, `Rectangle`, and `Circle`.

These classes are similar to those in A7 except for two very important details. First of all, there is no drawing code. More importantly **they do not use the advanced keyword arguments used by Kivy**. The expression `**keyword` should not appear anywhere in your solution.

(a) [8 points] The skeleton for class `Shape` is provided on the next page. Complete the initializer and method `__str__` according to their specification, being sure to maintain all class invariants.

(b) [10 points] You are to create the classes `Rectangle` and `Circle`, each of which is a subclass of `Shape`. We have not provided you with any skeleton code for these classes; you are to implement everything yourself.

The classes **have no new attributes** beyond those inherited from `Shape`. For each class implement its initializer and method calculateArea() according to the following constraints.

- The initializer for `Rectangle` should have an header that looks exactly like the initializer for `Shape`, including default values. The body of this constructor should be a single line call to the initializer of its superclass. You can skip the specifications in the interest of time.
- The initializer for `Circle` should have three parameters: `x`, `y`, and `radius`. Using the initializer of its superclass, it should set the `width` and `height` attributes to the diameter ($= 2r$).
- In both classes, method `calculateArea()` should return the area of the shape. For class `Circle`, you may use the constant `PI` in module `math`. Remember that the area of a circle is πr^2.

You can assume that `math` is imported; you do not need to write an import statement in your code. Implement your classes on the blank page after the class `Shape`.

```
class Shape(object):
    """Instance is a 2-dimensional geometric shape.
    Instance variables:
    x [float]:  x-coordinate of bottom-left corner
    y [float]:  y-coordinate of bottom-left corner
    width [float]:  shape's width; >= 0
    height [float]:  shape's height; >=0 """

    def __init__(  self, x=0.0, y=0.0, width=0.0, height=0.0  ):      # Fill in
        """Initializer: shape with given values x, y, width, and height (in order).
        Precondition: x, y, width, and height are floats with width, height >= 0.0.
        All parameters have a default of 0.0."""
        self.x = x
        self.y = y
        self.width = width
        self.height = height


    def __str__(self):
        """Returns: Description of shape geometry in format '[x,y,width,height]'.
        return ( '['+str(self.x)+sq,+str(self.y)+','+
                 str(self.width)+','+str(self.height)+']' )
```

(Answer Question (b) here)

```
class Rectangle(Shape):
    def __init__(self, x=0.0, y=0.0, width=0.0, height=0.0):
        Shape.__init__(self,x,y,width,height)

    def calculateArea(self):
        return self.width*self.height


class Circle(Shape):
    """Instance is a circular shape"""

    def __init__(self, x, y, radius):
        Shape.__init__(self,x,y,radius*2,radius*2)

    def calculateArea(self):
        return math.pi*(self.width/2.0)*(self.width/2.0)
```

[?? points total] **Call Frames and Diagrams**

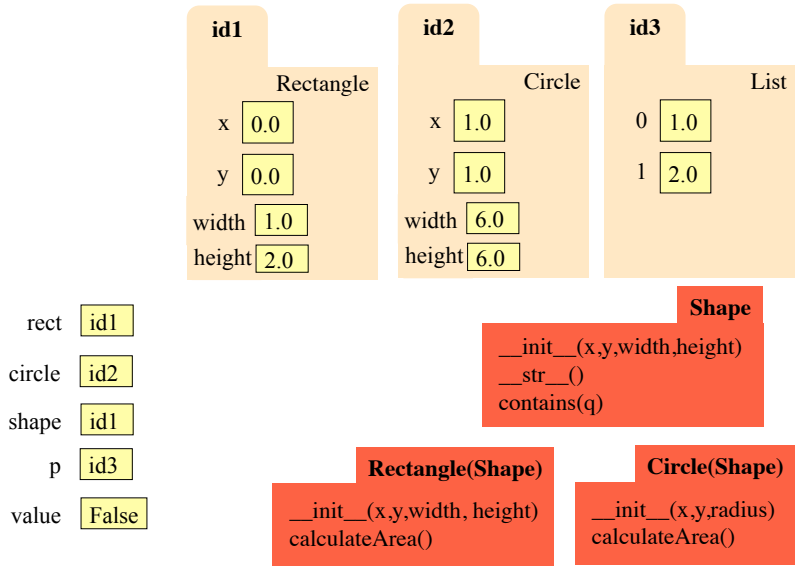
Suppose you were to modify class `Shape` to include the following method.

```
def contains(self,q):
    """Returns: True if point q is in this Shape's bounding rectangle; False
    otherwise.
    Precondition: q is a list [x,y]."""
    1   in_x = self.x < q[0] and q[0] < self.x+self.width
    2   in_y = self.y < q[1] and q[1] < self.y+self.height
    3   return in_x and in_y
```

Consider then the following code, placed in the same file as the definition of class Shape and your subclasses.

```
rect = Rectangle(0.0,0.0,1.0,2.0)
circle = Circle(1.0,1.0,3.0)
shape = rect
p = [1.0, 2.0]
value = shape.contains(p)
```

(a) [10 points] Execute the file, including the code above: draw all variables and folders that are created. Do not worry about call frames (yet). You do not have to draw the `object` class folder.



(b) [10 points] Draw your execution of the call `shape.contains(p)`. At the end, the relevant frame(s) should be crossed out. You do not need to redraw the folders; simply use the folder names for your answer in part (a).