Object-Oriented Programming
50:198:113 (Spring 2022)

Homework:	**6**	**Professor:**	**Suneeta Ramaswami**
Due Date:	**5/2/22**	**E-mail:**	`suneeta.ramaswami@rutgers.edu`
Office:	**321 BSB**	**URL:**	`http://crab.rutgers.edu/~rsuneeta`
		Phone:	**(856)-225-6439**

Homework Assignment 6

The assignment is due by 11:59PM of the due date. The point value is indicated in square braces next to each problem. Each solution must be the student's own work. Assistance should only be sought or accepted from the course instructor. Any violation of this rule will be dealt with harshly.

This assignment, which contains one problem, requires you to use inheritance to implement a hierarchy of classes. In this problem, you are asked to implement six classes for points and polygons called `Point`, `SimplePoly`, `ConvPoly`, `EquiTriangle`, `Rectangle`, and `Square`. Further details are provided below. As usual, you are graded not only on the correctness of the code, but also on clarity and readability. I will deduct points for not following the guidelines for your class design, poor indentation, poor choice of object names, and lack of documentation. For documentation, use a common sense approach. While I do not expect every line of code to be explained, all code blocks that carry out a significant task should be documented *briefly* in clear English.

Implement your classes in a module called `polys.py`. In addition, you are provided a test file, called `testpolys.py`, to test your class implementations.

Please read the submission guidelines at the end of this document before you start your work.

Problem 1 [80 points] Polygons. A *polygon* is a closed loop made up of straight line segments. When the loop does not intersect itself, the polygon is said to be a *simple* polygon. We can think of a simple polygon as a sequence of vertices specified in counter-clockwise order. Each vertex is a point with an (x, y) coordinate. For example, the simple polygon $Q = \{v_0, v_1, v_2, v_3, v_4, v_5, v_6\}$ in Figure 1(a) has 7 vertices and 7 edges.

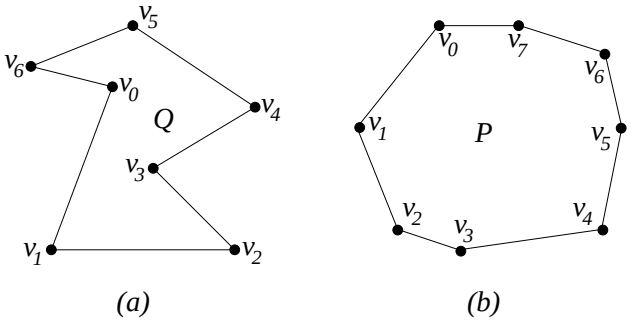

Figure 1: (a) Q is a simple polygon. (b) P is a convex polygon.

A *convex polygon* is a simple polygon whose shape satisfies the property that as we walk around the boundary of the polygon in counter-clockwise order, we always turn left at each

vertex. (Another way to define a convex polygon is that for any two points on or within the polygon, the straight line segment connecting those points lies entirely within the polygon.) See Figure 1(b) for an example of a convex polygon $P = \{v_0, v_1, v_2, v_3, v_4, v_5, v_6, v_7\}$ with 8 vertices.

In this problem, you are asked to implement a `Point` class, a `SimplePoly` class, a `ConvPoly` class, an `EquiTriangle` class, a `Rectangle` class, and a `Square` class. Instances of the `SimplePoly` class contain a collection of `Point` instances. The `ConvPoly` class is a sub-class of `SimplePoly`, the `EquiTriangle` and `Rectangle` classes are sub-classes of `ConvPoly`, and the `Square` class is a sub-class of `Rectangle`. Create a module called `polys.py` to contain all these classes. Further details are provided below.

`Point` class (12 points) . Instances of this class are points in two-dimensional space. Hence, each instance has an x coordinate and a y coordinate. We first state a few definitions. See Figure 2 for illustrations.

- **Translation:** If a point (x, y) is *translated* (moved) by an amount s in the x direction and an amount t in the y direction, its new coordinates are $(x + s, y + t)$.
- **Rotation:** If a point (x, y) is *rotated* by angle θ about the origin, its new coordinates are $(x\cos\theta - ysin\theta, x\sin\theta + y\cos\theta)$.
- **Distance:** The *distance* between two points (x_1, y_1) and (x_2, y_2) is $\sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2}$.
- **Sidedness:** Let $p = (p_x, p_y)$, $q = (q_x, q_y)$, and $r = (r_x, r_y)$ be three points. We say that p *lies to the left of* $\overrightarrow{qr}$ (that is, the points $< q, r, p >$ make a left turn) if and only if $(r_x p_y - p_x r_y) + (q_x r_y - q_x p_y) + (q_y p_x - q_y r_x) > 0$. We say that p *lies to the right of* $\overrightarrow{qr}$ (that is, the points $< q, r, p >$ make a right turn) if and only if $(r_x p_y - p_x r_y) + (q_x r_y - q_x p_y) + (q_y p_x - q_y r_x) < 0$. Finally, p, q, and r are collinear (all lie on a straight line) if and only if $(r_x p_y - p_x r_y) + (q_x r_y - q_x p_y) + (q_y p_x - q_y r_x) = 0$.

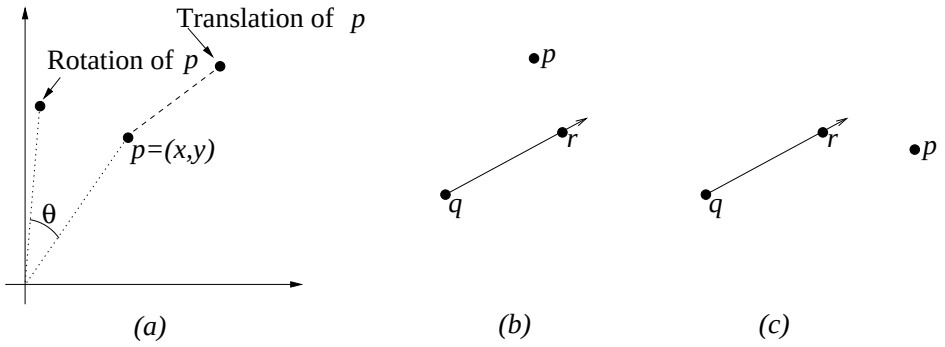

Figure 2: (a) Translation and rotation of a point p. (b) p lies to the left of $\overrightarrow{qr}$. (c) p lies to the right of $\overrightarrow{qr}$.

Methods for the `Point` class are described below:

1. `__init__`: The constructor sets the values of the x and y coordinates of the point. The default values for the point is $(0, 0)$.
2. `translate`: Translates the point by (s, t). The instance is modified after this method is called.

3. `rotate`: Rotates the point by an angle θ. The instance is modified after this method is called.
4. `distance`: Returns the distance between the point and another point `p`. Hence this method has one parameter, namely a point p.
5. `left_of`: This method has two parameters `q` and `r`. It returns `True` if the point lies to the left of $\overrightarrow{\texttt{qr}}$, and `False` otherwise.
6. `right_of`: This method has two parameters `q` and `r`. It returns `True` if the point lies to the right of $\overrightarrow{\texttt{qr}}$, and `False` otherwise.
7. `__str__`: Returns a string representation of the point. Use the usual notation to represent a point as its x and y coordinates surrounded by parentheses.
8. `__repr__`: Returns a string representation of the point.

`SimplePoly` **class (26 points)** . Instances of this class are simple polygons. Methods for this class are described below:

1. `__init__`: The constructor takes an *arbitrary number* of points as parameters, where the points are listed in counter-clockwise order about the boundary. The parameter to the constructor should be of the form `*vertices`. Store the points in a list. *You are not required to check the polygon for simplicity, i.e., that no two segments of the polygon intersect.*
2. `translate`: Translates the simple polygon by (s, t). This is equivalent to translating every vertex of the polygon by (s, t).
3. `rotate`: Rotates the simple polygon by angle θ. This is equivalent to rotating every vertex of the polygon by θ.
4. `__iter__`: Return an iterator. This should be an instance of a new iterator class. *Make sure you include a* `__next__` *method in that class*, which should raise the `StopIteration` exception suitably.
5. `__len__`: This method allows us to use the built-in function `len` with simple polygon instances (just as `__str__` allows us to use the built-in function `str`). It returns the number of vertices in the polygon.
6. `__getitem__`: Overload the index operator. If the parameters of this method are `self` and `i`, it returns the `i`-th vertex of the convex polygon. If the index is out of range (less than zero or greater than or equal to the number of vertices of the polygon), raise the `IndexError` exception.
7. `__str__`: Produces a string representation of the polygon. One obvious way to do this is to return a string containing the vertices of the polygon in counter-clockwise order.
8. `perimeter`: Return the perimeter of the polygon. *Hint:* Use the `distance` method for points.

`ConvPoly` **class (16 points)** . Instances of this class are convex polygons. Since a convex polygon is a simple polygon, this class is a subclass of the `SimplePoly` class. This class customizes one method, namely the constructor. All other methods are inherited.

1. `__init__`: As for simple polygons, the constructor takes an arbitrary number of points as parameters, where the points are listed in counter-clockwise order about the boundary. The parameter to the constructor should be of the form `*vertices`. Store the points in a list. **You are required to check that the vertices indeed form a convex polygon. If they do not, raise an exception.** (*Hint:* The

`left_of` or `right_of` methods will come in handy here.) If the polygon is convex, you should utilize the `SimplePoly` constructor to customize.

`EquiTriangle` **class (10 points)** . Instances of this class are equilateral triangles. Since an equilateral triangle is a convex polygon, this class is a subclass of the `ConvPoly` class. This class customizes one method (the constructor) and extends one method (`area`).

1. `__init__`: The constructor takes a single parameter, which is the length of the edges of the equilateral triangle. An equilateral triangle of that edge length, with one vertex at the origin is created. *Note:* Utilize the `ConvPoly` constructor to customize.
2. `area`: Returns the area of the equilateral triangle.

`Rectangle` **class (10 points)** Instances of this class are rectangles. Since a rectangle is a convex polygon, this class is a subclass of the `ConvPoly` class.

1. `__init__`: The constructor takes two parameters, which are the length and width of the rectangle. A rectangle of those dimensions, with one vertex at the origin is created. *Note:* Utilize the `ConvPoly` constructor to customize.
2. `area`: Returns the area of the rectangle.

`Square` **class (6 points)** Instances of this class are squares. Since a square is a rectangle, this class is a subclass of the `Rectangle` class.

1. `__init__`: The constructor takes a single parameter, which is the length of the square. *Note:* Utilize the `Rectangle` constructor to customize.

Submission Guidelines

Create a module called `polys.py` for the problem in this assignment. *Your name and RUID should appear as a comment at the very top of each file.*

Test each of your programs thoroughly before submitting your homework. When you are ready to submit, upload your files on Canvas as follows:

1. Go to the "Assignments" tab of the Canvas site for this course.
2. Click on "Programming Assignment #6" under Homework Assignments.
3. Upload your homework file (`polys.py`) when you are ready to submit.

You must submit your assignment at or before 11:55PM on May 2, 2022.