Lab 4- Object Oriented Programming

For all of the exercises below, make sure you provide tests of your solutions.

1. Write a "counter" class that can be incremented up to a specified maximum value, will print an error if an attempt is made to increment beyond that value, and allows reseting the counter.

```
class Counter:
    def __init__(self, max_value):
        """Initialize the counter with a maximum value and set the initial count to zero."""
        self.max_value = max_value
        self.count = 0

    def increment(self):
        """Increment the counter by one if the maximum value has not been reached."""
        if self.count < self.max_value:
            self.count += 1
        else:
            print("Error: Cannot increment. Maximum value reached.")

    def reset(self):
        """Reset the counter to zero."""
        self.count = 0

    def get_count(self):
        """Return the current value of the counter."""
        return self.count

    def __str__(self):
        """Return a string representation of the counter."""
        return f"Counter: {self.count}/{self.max_value}"

# Test the Counter class
def test_counter():
    print("Testing Counter Class")

    # Create a counter with a maximum value of 5
    counter = Counter(5)

    # Increment the counter
    for _ in range(6):  # Attempt to increment 6 times
        counter.increment()
        print(counter)

    # Reset the counter
    counter.reset()
    print("After reset:", counter)

    # Increment again after reset
    for _ in range(3):
        counter.increment()
        print(counter)

    # Final reset and print
    counter.reset()
    print("Final state:", counter)

# Run the test
if __name__ == "__main__":
    test_counter()
```

```
Testing Counter Class
 Counter: 1/5
 Counter: 2/5
 Counter: 3/5
 Counter: 4/5
 Counter: 5/5
 Error: Cannot increment. Maximum value reached.
 Counter: 5/5
 After reset: Counter: 0/5
 Counter: 1/5
 Counter: 2/5
```

```
Counter: 3/5
Final state: Counter: 0/5
```

2. Copy and paste your solution to question 1 and modify it so that all the data held by the counter is private. Implement functions to check the value of the counter, check the maximum value, and check if the counter is at the maximum.

```
class Counter:
    def __init__(self, max_value):
        """Initialize the counter with a maximum value and set the initial count to zero."""
        self.__max_value = max_value  # Private variable for maximum value
        self.__count = 0  # Private variable for current count

    def increment(self):
        """Increment the counter by one if the maximum value has not been reached."""
        if self.__count < self.__max_value:
            self.__count += 1
        else:
            print("Error: Cannot increment. Maximum value reached.")

    def reset(self):
        """Reset the counter to zero."""
        self.__count = 0

    def get_count(self):
        """Return the current value of the counter."""
        return self.__count

    def get_max_value(self):
        """Return the maximum value of the counter."""
        return self.__max_value

    def is_at_max(self):
        """Check if the counter is at the maximum value."""
        return self.__count == self.__max_value

    def __str__(self):
        """Return a string representation of the counter."""
        return f"Counter: {self.__count}/{self.__max_value}"

# Test the Counter class
def test_counter():
    print("Testing Counter Class")

    # Create a counter with a maximum value of 5
    counter = Counter(5)

    # Increment the counter
    for _ in range(6):  # Attempt to increment 6 times
        counter.increment()
        print(counter)

    # Check maximum value
    print("Maximum value:", counter.get_max_value())

    # Check if at max
    print("Is at max:", counter.is_at_max())

    # Reset the counter
    counter.reset()
    print("After reset:", counter)

    # Increment again after reset
    for _ in range(3):
        counter.increment()
        print(counter)

    # Check if at max again
    print("Is at max after increments:", counter.is_at_max())

    # Final reset and print
    counter.reset()
    print("Final state:", counter)

# Run the test
if __name__ == "__main__":
```

```
    test_counter()
```

```
Testing Counter Class
Counter: 1/5
Counter: 2/5
Counter: 3/5
Counter: 4/5
Counter: 5/5
Error: Cannot increment. Maximum value reached.
Counter: 5/5
Maximum value: 5
Is at max: True
After reset: Counter: 0/5
Counter: 1/5
Counter: 2/5
Counter: 3/5
Is at max after increments: False
Final state: Counter: 0/5
```

3. Implement a class to represent a rectangle, holding the length, width, and x and y coordinates of a corner of the object. Implement functions that compute the area and perimeter of the rectangle. Make all data members private and privide accessors to retrieve values of data members.

```
class Rectangle:
    def __init__(self, length, width, x, y):
        """Initialize the rectangle with length, width, and coordinates of a corner."""
        self.__length = length  # Private variable for length
        self.__width = width    # Private variable for width
        self.__x = x            # Private variable for x-coordinate
        self.__y = y            # Private variable for y-coordinate

    def area(self):
        """Calculate and return the area of the rectangle."""
        return self.__length * self.__width

    def perimeter(self):
        """Calculate and return the perimeter of the rectangle."""
        return 2 * (self.__length + self.__width)

    def get_length(self):
        """Return the length of the rectangle."""
        return self.__length

    def get_width(self):
        """Return the width of the rectangle."""
        return self.__width

    def get_coordinates(self):
        """Return the coordinates of the rectangle's corner."""
        return (self.__x, self.__y)

    def __str__(self):
        """Return a string representation of the rectangle."""
        return (f"Rectangle(length={self.__length}, width={self.__width}, "
                f"coordinates=({self.__x}, {self.__y}))")

# Test the Rectangle class
def test_rectangle():
    print("Testing Rectangle Class")

    # Create a rectangle with length 10, width 5, and corner at (0, 0)
    rectangle = Rectangle(10, 5, 0, 0)

    # Display rectangle information
    print(rectangle)

    # Calculate and print area and perimeter
    print("Area:", rectangle.area())
    print("Perimeter:", rectangle.perimeter())

    # Access and print length, width, and coordinates
    print("Length:", rectangle.get_length())
    print("Width:", rectangle.get_width())
    print("Coordinates:", rectangle.get_coordinates())

# Run the test
```

```
if __name__ == "__main__":
    test_rectangle()
```

```
Testing Rectangle Class
Rectangle(length=10, width=5, coordinates=(0, 0))
Area: 50
Perimeter: 30
Length: 10
Width: 5
Coordinates: (0, 0)
```

4. Implement a class to represent a circle, holding the radius and x and y coordinates of center of the object. Implement functions that compute the area and perimeter of the rectangle. Make all data members private and privide accessors to retrieve values of data members.

```
import math

class Circle:
    def __init__(self, radius, x, y):
        """Initialize the circle with radius and coordinates of the center."""
        self.__radius = radius  # Private variable for radius
        self.__x = x            # Private variable for x-coordinate
        self.__y = y            # Private variable for y-coordinate

    def area(self):
        """Calculate and return the area of the circle."""
        return math.pi * (self.__radius ** 2)

    def perimeter(self):
        """Calculate and return the circumference of the circle."""
        return 2 * math.pi * self.__radius

    def get_radius(self):
        """Return the radius of the circle."""
        return self.__radius

    def get_coordinates(self):
        """Return the coordinates of the circle's center."""
        return (self.__x, self.__y)

    def __str__(self):
        """Return a string representation of the circle."""
        return (f"Circle(radius={self.__radius}, center=({self.__x}, {self.__y}))")

# Test the Circle class
def test_circle():
    print("Testing Circle Class")

    # Create a circle with radius 5 and center at (0, 0)
    circle = Circle(5, 0, 0)

    # Display circle information
    print(circle)

    # Calculate and print area and perimeter (circumference)
    print("Area:", circle.area())
    print("Perimeter (Circumference):", circle.perimeter())

    # Access and print radius and coordinates
    print("Radius:", circle.get_radius())
    print("Coordinates:", circle.get_coordinates())

# Run the test
if __name__ == "__main__":
    test_circle()
```

```
Testing Circle Class
Circle(radius=5, center=(0, 0))
Area: 78.53981633974483
Perimeter (Circumference): 31.41592653589793
Radius: 5
Coordinates: (0, 0)
```

5. Implement a common base class for the classes implemented in 3 and 4 above which implements all common methods as not implemented functions (virtual). Re-implement your regtangle and circule classes to inherit from the base class and overload the functions accordingly.

```
from abc import ABC, abstractmethod
import math

class Shape(ABC):
    @abstractmethod
    def area(self):
        """Calculate and return the area of the shape."""
        pass

    @abstractmethod
    def perimeter(self):
        """Calculate and return the perimeter (or circumference) of the shape."""
        pass

    @abstractmethod
    def get_coordinates(self):
        """Return the coordinates associated with the shape."""
        pass

    @abstractmethod
    def __str__(self):
        """Return a string representation of the shape."""
        pass

class Rectangle(Shape):
    def __init__(self, length, width, x, y):
        """Initialize the rectangle with length, width, and coordinates of a corner."""
        self.__length = length  # Private variable for length
        self.__width = width    # Private variable for width
        self.__x = x            # Private variable for x-coordinate
        self.__y = y            # Private variable for y-coordinate

    def area(self):
        """Calculate and return the area of the rectangle."""
        return self.__length * self.__width

    def perimeter(self):
        """Calculate and return the perimeter of the rectangle."""
        return 2 * (self.__length + self.__width)

    def get_coordinates(self):
        """Return the coordinates of the rectangle's corner."""
        return (self.__x, self.__y)

    def __str__(self):
        """Return a string representation of the rectangle."""
        return (f"Rectangle(length={self.__length}, width={self.__width}, "
                f"coordinates=({self.__x}, {self.__y}))")

class Circle(Shape):
    def __init__(self, radius, x, y):
        """Initialize the circle with radius and coordinates of the center."""
        self.__radius = radius  # Private variable for radius
        self.__x = x            # Private variable for x-coordinate
        self.__y = y            # Private variable for y-coordinate

    def area(self):
        """Calculate and return the area of the circle."""
        return math.pi * (self.__radius ** 2)

    def perimeter(self):
        """Calculate and return the circumference of the circle."""
        return 2 * math.pi * self.__radius

    def get_coordinates(self):
        """Return the coordinates of the circle's center."""
        return (self.__x, self.__y)

    def __str__(self):
        """Return a string representation of the circle."""
```

```
        return (f"Circle(radius={self.__radius}, center=({self.__x}, {self.__y}))")

# Test the classes
def test_shapes():
    print("Testing Shape Classes")

    # Create a rectangle and circle
    rectangle = Rectangle(10, 5, 0, 0)
    circle = Circle(5, 0, 0)

    # Display rectangle information
    print(rectangle)
    print("Area:", rectangle.area())
    print("Perimeter:", rectangle.perimeter())
    print("Coordinates:", rectangle.get_coordinates())

    # Display circle information
    print(circle)
    print("Area:", circle.area())
    print("Perimeter (Circumference):", circle.perimeter())
    print("Coordinates:", circle.get_coordinates())

# Run the tests
if __name__ == "__main__":
    test_shapes()
```

```
Testing Shape Classes
Rectangle(length=10, width=5, coordinates=(0, 0))
Area: 50
Perimeter: 30
Coordinates: (0, 0)
Circle(radius=5, center=(0, 0))
Area: 78.53981633974483
Perimeter (Circumference): 31.41592653589793
Coordinates: (0, 0)
```

6. Implement a triangle class analogous to the rectangle and circle in question 5.

```
from abc import ABC, abstractmethod
import math

class Shape(ABC):
    @abstractmethod
    def area(self):
        """Calculate and return the area of the shape."""
        pass

    @abstractmethod
    def perimeter(self):
        """Calculate and return the perimeter of the shape."""
        pass

    @abstractmethod
    def get_coordinates(self):
        """Return the coordinates associated with the shape."""
        pass

    @abstractmethod
    def __str__(self):
        """Return a string representation of the shape."""
        pass

class Triangle(Shape):
    def __init__(self, side_a, side_b, side_c, x, y):
        """Initialize the triangle with the lengths of its sides and the coordinates of a vertex."""
        self.__side_a = side_a  # Private variable for side A
        self.__side_b = side_b  # Private variable for side B
        self.__side_c = side_c  # Private variable for side C
        self.__x = x            # Private variable for x-coordinate of a vertex
        self.__y = y            # Private variable for y-coordinate of a vertex

    def area(self):
        """Calculate and return the area of the triangle using Heron's formula."""
        s = self.perimeter() / 2  # Semi-perimeter
        return math.sqrt(s * (s - self.__side_a) * (s - self.__side_b) * (s - self.__side_c))
```

```
        def perimeter(self):
            """Calculate and return the perimeter of the triangle."""
            return self.__side_a + self.__side_b + self.__side_c

        def get_coordinates(self):
            """Return the coordinates of the triangle's vertex."""
            return (self.__x, self.__y)

        def __str__(self):
            """Return a string representation of the triangle."""
            return (f"Triangle(sides=({self.__side_a}, {self.__side_b}, {self.__side_c}), "
                    f"coordinates=({self.__x}, {self.__y}))")

    # Test the Triangle class
    def test_triangle():
        print("Testing Triangle Class")

        # Create a triangle with sides 3, 4, 5 and vertex at (0, 0)
        triangle = Triangle(3, 4, 5, 0, 0)

        # Display triangle information
        print(triangle)

        # Calculate and print area and perimeter
        print("Area:", triangle.area())
        print("Perimeter:", triangle.perimeter())
        print("Coordinates:", triangle.get_coordinates())

    # Run the test
    if __name__ == "__main__":
        test_triangle()
```

```
Testing Triangle Class
Triangle(sides=(3, 4, 5), coordinates=(0, 0))
Area: 6.0
Perimeter: 12
Coordinates: (0, 0)
```

7. Add a function to the object classes, including the base, that returns a list of up to 16 pairs of x and y points on the parameter of the object.

```
from abc import ABC, abstractmethod
import math

class Shape(ABC):
    @abstractmethod
    def area(self):
        """Calculate and return the area of the shape."""
        pass

    @abstractmethod
    def perimeter(self):
        """Calculate and return the perimeter (or circumference) of the shape."""
        pass

    @abstractmethod
    def get_coordinates(self):
        """Return the coordinates associated with the shape."""
        pass

    @abstractmethod
    def get_points_on_perimeter(self, num_points=16):
        """Return a list of points on the perimeter of the shape."""
        pass

    @abstractmethod
    def __str__(self):
        """Return a string representation of the shape."""
        pass

class Rectangle(Shape):
    def __init__(self, length, width, x, y):
        """Initialize the rectangle with length, width, and coordinates of a corner."""
        self.__length = length
        self.__width = width
```

```
            self.__x = x
            self.__y = y

        def area(self):
            return self.__length * self.__width

        def perimeter(self):
            return 2 * (self.__length + self.__width)

        def get_coordinates(self):
            return (self.__x, self.__y)

        def get_points_on_perimeter(self, num_points=16):
            """Return points on the perimeter of the rectangle."""
            points = []
            for i in range(num_points):
                t = i / (num_points - 1)  # Normalized parameter
                if t < 0.25:
                    # Top side
                    x = self.__x + t * self.__length
                    y = self.__y
                elif t < 0.5:
                    # Right side
                    x = self.__x + self.__length
                    y = self.__y + (t - 0.25) * self.__width
                elif t < 0.75:
                    # Bottom side
                    x = self.__x + (0.75 - t) * self.__length
                    y = self.__y + self.__width
                else:
                    # Left side
                    x = self.__x
                    y = self.__y + (1 - t) * self.__width
                points.append((x, y))
            return points

        def __str__(self):
            return (f"Rectangle(length={self.__length}, width={self.__width}, "
                    f"coordinates=({self.__x}, {self.__y}))")

    class Circle(Shape):
        def __init__(self, radius, x, y):
            self.__radius = radius
            self.__x = x
            self.__y = y

        def area(self):
            return math.pi * (self.__radius ** 2)

        def perimeter(self):
            return 2 * math.pi * self.__radius

        def get_coordinates(self):
            return (self.__x, self.__y)

        def get_points_on_perimeter(self, num_points=16):
            """Return points on the perimeter of the circle."""
            points = []
            for i in range(num_points):
                angle = 2 * math.pi * i / num_points
                x = self.__x + self.__radius * math.cos(angle)
                y = self.__y + self.__radius * math.sin(angle)
                points.append((x, y))
            return points

        def __str__(self):
            return (f"Circle(radius={self.__radius}, center=({self.__x}, {self.__y}))")

    class Triangle(Shape):
        def __init__(self, side_a, side_b, side_c, x, y):
            self.__side_a = side_a
            self.__side_b = side_b
            self.__side_c = side_c
            self.__x = x
            self.__y = y

        def area(self):
            s = self.perimeter() / 2
```

```
            return math.sqrt(s * (s - self.__side_a) * (s - self.__side_b) * (s - self.__side_c))

        def perimeter(self):
            return self.__side_a + self.__side_b + self.__side_c

        def get_coordinates(self):
            return (self.__x, self.__y)

        def get_points_on_perimeter(self, num_points=16):
            """Return points on the perimeter of the triangle."""
            points = []
            # Assuming the triangle vertices are at (x, y), (x + side_a, y), and some height.
            height = (2 * self.area()) / self.__side_a
            vertex_b = (self.__x + self.__side_a, self.__y)
            vertex_c = (self.__x + (self.__side_b - height), self.__y + height)

            # Interpolating points between vertices
            for i in range(num_points):
                t = i / (num_points - 1)
                if t < 0.5:
                    # Interpolating between vertex A and B
                    x = self.__x + t * (vertex_b[0] - self.__x)
                    y = self.__y + t * (vertex_b[1] - self.__y)
                else:
                    # Interpolating between vertex B and C
                    x = vertex_b[0] + (t - 0.5) * (vertex_c[0] - vertex_b[0])
                    y = vertex_b[1] + (t - 0.5) * (vertex_c[1] - vertex_b[1])
                points.append((x, y))
            return points

        def __str__(self):
            return (f"Triangle(sides=({self.__side_a}, {self.__side_b}, {self.__side_c}), "
                    f"coordinates=({self.__x}, {self.__y}))")

    # Test the classes
    def test_shapes():
        print("Testing Shape Classes")

        # Create a rectangle, circle, and triangle
        rectangle = Rectangle(10, 5, 0, 0)
        circle = Circle(5, 0, 0)
        triangle = Triangle(3, 4, 5, 0, 0)

        # Display information and points on perimeter
        for shape in [rectangle, circle, triangle]:
            print(shape)
            print("Area:", shape.area())
            print("Perimeter:", shape.perimeter())
            print("Coordinates:", shape.get_coordinates())
            print("Points on Perimeter:", shape.get_points_on_perimeter())
            print()

    # Run the tests
    if __name__ == "__main__":
        test_shapes()
```

```
Testing Shape Classes
Rectangle(length=10, width=5, coordinates=(0, 0))
Area: 50
Perimeter: 30
Coordinates: (0, 0)
Points on Perimeter: [(0.0, 0), (0.6666666666666666, 0), (1.3333333333333333, 0), (2.0, 0), (10, 0.08333333333333331), (10, 0.4166666666

Circle(radius=5, center=(0, 0))
Area: 78.53981633974483
Perimeter: 31.41592653589793
Coordinates: (0, 0)
Points on Perimeter: [(5.0, 0.0), (4.619397662556434, 1.913417161825449), (3.5355339059327378, 3.5355339059327373), (1.9134171618254492,

Triangle(sides=(3, 4, 5), coordinates=(0, 0))
Area: 6.0
Perimeter: 12
Coordinates: (0, 0)
Points on Perimeter: [(0.0, 0.0), (0.2, 0.0), (0.4, 0.0), (0.6000000000000001, 0.0), (0.8, 0.0), (1.0, 0.0), (1.2000000000000002, 0.0),
```

8. Add a function to the object classes, including the base, that tests if a given set of x and y coordinates are inside of the object. You'll have to think through how to determine if a set of coordinates are inside an object for each object type.

```
import math

class Shape:
    def area(self):
        raise NotImplementedError

    def perimeter(self):
        raise NotImplementedError

    def get_coordinates(self):
        raise NotImplementedError

    def get_points_on_perimeter(self, num_points=16):
        raise NotImplementedError

    def contains(self, x, y):
        raise NotImplementedError

class Triangle(Shape):
    def __init__(self, side_a, side_b, side_c, x, y):
        """Initialize the triangle with side lengths and coordinates of one vertex."""
        if not self.is_valid_triangle(side_a, side_b, side_c):
            raise ValueError("Invalid triangle sides provided.")

        self.__side_a = side_a
        self.__side_b = side_b
        self.__side_c = side_c
        self.__x = x
        self.__y = y

    def is_valid_triangle(self, a, b, c):
        """Check if the sides can form a triangle."""
        return a + b > c and a + c > b and b + c > a

    def area(self):
        """Calculate and return the area of the triangle using Heron's formula."""
        s = self.perimeter() / 2
        return math.sqrt(s * (s - self.__side_a) * (s - self.__side_b) * (s - self.__side_c))

    def perimeter(self):
        """Calculate and return the perimeter of the triangle."""
        return self.__side_a + self.__side_b + self.__side_c

    def get_coordinates(self):
        """Return the coordinates of one vertex of the triangle."""
        return (self.__x, self.__y)

    def get_points_on_perimeter(self, num_points=16):
        """Return points on the perimeter of the triangle."""
        points = []
        for i in range(num_points + 1):
            t = i / num_points
            if t < 1/3:
                # Point on side a
                x = self.__x + t * self.__side_a
                y = self.__y
            elif t < 2/3:
                # Point on side b
                ratio = (t - 1/3) * 3
                x = self.__x + self.__side_a - ratio * (self.__side_b)
                y = self.__y + ratio * (math.sqrt(self.__side_c**2 - (self.__side_a - ratio * self.__side_b)**2))
            else:
                # Point on side c
                ratio = (t - 2/3) * 3
                x = self.__x
                y = self.__y + ratio * (self.__side_c)
            points.append((x, y))
        return points

    def contains(self, x, y):
        """Check if the point (x, y) is inside the triangle using barycentric coordinates."""
        A = (self.__x, self.__y)
```

```
            B = (self.__x + self.__side_a, self.__y)
            C = (self.__x, self.__y + math.sqrt(self.__side_c**2 - (self.__side_a / 2)**2))

            # Compute the area of the whole triangle
            area_ABC = self.area()
            # Compute areas of sub-triangles
            area_PAB = Triangle(
                math.sqrt((B[0] - x) ** 2 + (B[1] - y) ** 2),
                           math.sqrt((C[0] - x) ** 2 + (C[1] - y) ** 2),
                           self.__side_a, A[0], A[1]).area()
            area_PBC = Triangle(
                math.sqrt((A[0] - x) ** 2 + (A[1] - y) ** 2),
                           math.sqrt((C[0] - x) ** 2 + (C[1] - y) ** 2),
                           self.__side_b, B[0], B[1]).area()
            area_PCA = Triangle(
                math.sqrt((A[0] - x) ** 2 + (A[1] - y) ** 2),
                           math.sqrt((B[0] - x) ** 2 + (B[1] - y) ** 2),
                           self.__side_c, C[0], C[1]).area()

            # Check if the sum of the areas of the sub-triangles equals the original triangle area
            return math.isclose(area_ABC, area_PAB + area_PBC + area_PCA)

        def __str__(self):
            return (f"Triangle(sides=({self.__side_a}, {self.__side_b}, {self.__side_c}), "
                    f"coordinates=({self.__x}, {self.__y}))")

    # Testing function
    def test_shapes():
        print("Testing Shape Classes")

        # Create a valid triangle
        try:
            triangle = Triangle(3, 4, 5, 0, 0)
            print(triangle)
            print("Area:", triangle.area())
            print("Perimeter:", triangle.perimeter())
            print("Coordinates:", triangle.get_coordinates())
            print("Points on Perimeter:", triangle.get_points_on_perimeter())

            # Test points
            test_points = [(1, 1), (11, 1), (0, 0), (5, 5), (3, 3)]
            for point in test_points:
                print(f"Point {point} inside {triangle}: {triangle.contains(*point)}")

        except ValueError as e:
            print(e)

    # Run the tests
    if __name__ == "__main__":
        test_shapes()
```

```
Testing Shape Classes
Triangle(sides=(3, 4, 5), coordinates=(0, 0))
Area: 6.0
Perimeter: 12
Coordinates: (0, 0)
Points on Perimeter: [(0.0, 0), (0.1875, 0), (0.375, 0), (0.5625, 0), (0.75, 0), (0.9375, 0), (2.5, 0.5412658773652744), (1.749999999999
Invalid triangle sides provided.
```

9. Add a function in the base class of the object classes that returns true/false testing that the object overlaps with another object.

```
import math

class Shape:
    def area(self):
        raise NotImplementedError

    def perimeter(self):
        raise NotImplementedError

    def get_coordinates(self):
        raise NotImplementedError

    def get_points_on_perimeter(self, num_points=16):
```

```
            raise NotImplementedError

        def contains(self, x, y):
            raise NotImplementedError

        def overlaps(self, other):
            raise NotImplementedError


    class Rectangle(Shape):
        def __init__(self, length, width, x, y):
            self.__length = length
            self.__width = width
            self.__x = x
            self.__y = y

        def area(self):
            return self.__length * self.__width

        def perimeter(self):
            return 2 * (self.__length + self.__width)

        def get_coordinates(self):
            return (self.__x, self.__y)

        def get_points_on_perimeter(self, num_points=16):
            points = []
            for i in range(num_points + 1):
                t = i / num_points
                x = self.__x + t * self.__length
                y = self.__y + (1 - t) * self.__width
                points.append((x, y))
            return points

        def contains(self, x, y):
            return (self.__x <= x <= self.__x + self.__length) and (self.__y <= y <= self.__y + self.__width)

        def overlaps(self, other):
            if isinstance(other, Rectangle):
                return not (self.__x + self.__length < other.__x or
                            self.__x > other.__x + other.__length or
                            self.__y + self.__width < other.__y or
                            self.__y > other.__y + other.__width)
            return False  # For other shapes, implement specific checks


    class Circle(Shape):
        def __init__(self, radius, x, y):
            self.__radius = radius
            self.__x = x
            self.__y = y

        def area(self):
            return math.pi * (self.__radius ** 2)

        def perimeter(self):
            return 2 * math.pi * self.__radius

        def get_coordinates(self):
            return (self.__x, self.__y)

        def get_points_on_perimeter(self, num_points=16):
            points = []
            for i in range(num_points):
                angle = (2 * math.pi / num_points) * i
                points.append((self.__x + self.__radius * math.cos(angle), self.__y + self.__radius * math.sin(angle)))
            return points

        def contains(self, x, y):
            return math.sqrt((self.__x - x) ** 2 + (self.__y - y) ** 2) <= self.__radius

        def overlaps(self, other):
            if isinstance(other, Circle):
                distance = math.sqrt((self.__x - other.__x) ** 2 + (self.__y - other.__y) ** 2)
                return distance < (self.__radius + other.__radius)
            elif isinstance(other, Rectangle):
                # Check for overlap with rectangle using closest point to circle
```

```
            closest_x = max(other.get_coordinates()[0], min(self.__x, other.get_coordinates()[0] + other.__length))
            closest_y = max(other.get_coordinates()[1], min(self.__y, other.get_coordinates()[1] + other.__width))
            return self.contains(closest_x, closest_y)
        return False


class Triangle(Shape):
    def __init__(self, side_a, side_b, side_c, x, y):
        if not self.is_valid_triangle(side_a, side_b, side_c):
            raise ValueError("Invalid triangle sides provided.")

        self.__side_a = side_a
        self.__side_b = side_b
        self.__side_c = side_c
        self.__x = x
        self.__y = y

    def is_valid_triangle(self, a, b, c):
        return a + b > c and a + c > b and b + c > a

    def area(self):
        s = self.perimeter() / 2
        return math.sqrt(s * (s - self.__side_a) * (s - self.__side_b) * (s - self.__side_c))

    def perimeter(self):
        return self.__side_a + self.__side_b + self.__side_c

    def get_coordinates(self):
        return (self.__x, self.__y)

    def get_points_on_perimeter(self, num_points=16):
        points = []
        for i in range(num_points + 1):
            t = i / num_points
            if t < 1/3:
                x = self.__x + t * self.__side_a
                y = self.__y
            elif t < 2/3:
                ratio = (t - 1/3) * 3
                x = self.__x + self.__side_a - ratio * (self.__side_b)
                y = self.__y + ratio * (math.sqrt(self.__side_c**2 - (self.__side_a - ratio * self.__side_b)**2))
            else:
                ratio = (t - 2/3) * 3
                x = self.__x
                y = self.__y + ratio * (self.__side_c)
            points.append((x, y))
        return points

    def contains(self, x, y):
        A = (self.__x, self.__y)
        B = (self.__x + self.__side_a, self.__y)
        C = (self.__x, self.__y + math.sqrt(self.__side_c**2 - (self.__side_a / 2)**2))

        area_ABC = self.area()
        area_PAB = Triangle(
            math.sqrt((B[0] - x) ** 2 + (B[1] - y) ** 2),
                      math.sqrt((C[0] - x) ** 2 + (C[1] - y) ** 2),
                      self.__side_a, A[0], A[1]).area()
        area_PBC = Triangle(
            math.sqrt((A[0] - x) ** 2 + (A[1] - y) ** 2),
                      math.sqrt((C[0] - x) ** 2 + (C[1] - y) ** 2),
                      self.__side_b, B[0], B[1]).area()
        area_PCA = Triangle(
            math.sqrt((A[0] - x) ** 2 + (A[1] - y) ** 2),
                      math.sqrt((B[0] - x) ** 2 + (B[1] - y) ** 2),
                      self.__side_c, C[0], C[1]).area()

        return math.isclose(area_ABC, area_PAB + area_PBC + area_PCA)

    def overlaps(self, other):
        if isinstance(other, Triangle):
            # Simple bounding box check for overlap
            min_x1 = self.__x
            max_x1 = self.__x + self.__side_a
            min_y1 = self.__y
            max_y1 = self.__y + math.sqrt(self.__side_c**2 - (self.__side_a / 2)**2)
```

```
            min_x2, min_y2 = other.get_coordinates()
            max_x2 = min_x2 + other.__side_a
            max_y2 = min_y2 + math.sqrt(other.__side_c**2 - (other.__side_a / 2)**2)

            return not (max_x1 < min_x2 or min_x1 > max_x2 or max_y1 < min_y2 or min_y1 > max_y2)
        return False


# Testing function
def test_shapes():
    print("Testing Shape Classes")

    # Create instances
    rect1 = Rectangle(10, 5, 0, 0)
    rect2 = Rectangle(6, 6, 5, 2)
    circle1 = Circle(5, 3, 3)
    triangle1 = Triangle(3, 4, 5, 0, 0)

    # Test overlap with rectangles
    print("Rectangle 1 overlaps Rectangle 2:", rect1.overlaps(rect2))

    # Test overlap with circle
    print("Rectangle 1 overlaps Circle 1:", rect1.overlaps(circle1))

    # Test overlap with triangle
    print("Rectangle 1 overlaps Triangle 1:", rect1.overlaps(triangle1))

    # Test circle overlap
    circle2 = Circle(3, 6, 6)
    print("Circle 1 overlaps Circle 2:", circle1.overlaps(circle2))

    # Test triangle overlap
    triangle2 = Triangle(5, 5, 5, 1, 1)
    print("Triangle 1 overlaps Triangle 2:", triangle1.overlaps(triangle2))


# Run the tests
if __name__ == "__main__":
    test_shapes()
```

```
Testing Shape Classes
Rectangle 1 overlaps Rectangle 2: True
Rectangle 1 overlaps Circle 1: False
Rectangle 1 overlaps Triangle 1: False
Circle 1 overlaps Circle 2: True
Triangle 1 overlaps Triangle 2: True
```

10. Copy the `Canvas` class from lecture to in a python file creating a `paint` module. Copy your classes from above into the module and implement paint functions. Implement a `CompoundShape` class. Create a simple drawing demonstrating that all of your classes are working.

```
import math

class Shape:
    def area(self):
        raise NotImplementedError

    def perimeter(self):
        raise NotImplementedError

    def get_coordinates(self):
        raise NotImplementedError

    def get_points_on_perimeter(self, num_points=16):
        raise NotImplementedError

    def contains(self, x, y):
        raise NotImplementedError

    def overlaps(self, other):
        raise NotImplementedError


class Rectangle(Shape):
    def __init__(self, length, width, x, y):
```

```
            self.__length = length
            self.__width = width
            self.__x = x
            self.__y = y

        def area(self):
            return self.__length * self.__width

        def perimeter(self):
            return 2 * (self.__length + self.__width)

        def get_coordinates(self):
            return (self.__x, self.__y)

        def get_points_on_perimeter(self, num_points=16):
            points = []
            for i in range(num_points + 1):
                if i < num_points / 2:
                    x = self.__x + (i / (num_points / 2)) * self.__length
                    y = self.__y
                else:
                    x = self.__x + self.__length
                    y = self.__y + ((i - (num_points / 2)) / (num_points / 2)) * self.__width
                points.append((x, y))
            return points

        def contains(self, x, y):
            return (self.__x <= x <= self.__x + self.__length) and (self.__y <= y <= self.__y + self.__width)

        def overlaps(self, other):
            if isinstance(other, Rectangle):
                return not (self.__x + self.__length < other.__x or
                            self.__x > other.__x + other.__length or
                            self.__y + self.__width < other.__y or
                            self.__y > other.__y + other.__width)
            return False  # For other shapes, implement specific checks


    class Circle(Shape):
        def __init__(self, radius, x, y):
            self.__radius = radius
            self.__x = x
            self.__y = y

        def area(self):
            return math.pi * (self.__radius ** 2)

        def perimeter(self):
            return 2 * math.pi * self.__radius

        def get_coordinates(self):
            return (self.__x, self.__y)

        def get_points_on_perimeter(self, num_points=16):
            points = []
            for i in range(num_points):
                angle = (2 * math.pi / num_points) * i
                points.append((self.__x + self.__radius * math.cos(angle), self.__y + self.__radius * math.sin(angle)))
            return points

        def contains(self, x, y):
            return math.sqrt((self.__x - x) ** 2 + (self.__y - y) ** 2) <= self.__radius

        def overlaps(self, other):
            if isinstance(other, Circle):
                distance = math.sqrt((self.__x - other.__x) ** 2 + (self.__y - other.__y) ** 2)
                return distance < (self.__radius + other.__radius)
            elif isinstance(other, Rectangle):
                closest_x = max(other.get_coordinates()[0], min(self.__x, other.get_coordinates()[0] + other.__length))
                closest_y = max(other.get_coordinates()[1], min(self.__y, other.get_coordinates()[1] + other.__width))
                return self.contains(closest_x, closest_y)
            return False


    class Triangle(Shape):
        def __init__(self, side_a, side_b, side_c, x, y):
            if not self.is_valid_triangle(side_a, side_b, side_c):
```

```
            raise ValueError("Invalid triangle sides provided.")

        self.__side_a = side_a
        self.__side_b = side_b
        self.__side_c = side_c
        self.__x = x
        self.__y = y

    def is_valid_triangle(self, a, b, c):
        return a + b > c and a + c > b and b + c > a

    def area(self):
        s = self.perimeter() / 2
        return math.sqrt(s * (s - self.__side_a) * (s - self.__side_b) * (s - self.__side_c))

    def perimeter(self):
        return self.__side_a + self.__side_b + self.__side_c

    def get_coordinates(self):
        return (self.__x, self.__y)

    def get_points_on_perimeter(self, num_points=16):
        points = []
        for i in range(num_points + 1):
            t = i / num_points
            if t < 1/3:
                x = self.__x + t * self.__side_a
                y = self.__y
            elif t < 2/3:
                ratio = (t - 1/3) * 3
                x = self.__x + self.__side_a - ratio * (self.__side_b)
                y = self.__y + ratio * (math.sqrt(self.__side_c**2 - (self.__side_a - ratio * self.__side_b)**2))
            else:
                ratio = (t - 2/3) * 3
                x = self.__x
                y = self.__y + ratio * (self.__side_c)
            points.append((x, y))
        return points

    def contains(self, x, y):
        A = (self.__x, self.__y)
        B = (self.__x + self.__side_a, self.__y)
        C = (self.__x, self.__y + math.sqrt(self.__side_c**2 - (self.__side_a / 2)**2))

        area_ABC = self.area()
        area_PAB = Triangle(
            math.sqrt((B[0] - x) ** 2 + (B[1] - y) ** 2),
            math.sqrt((C[0] - x) ** 2 + (C[1] - y) ** 2),
            self.__side_a, A[0], A[1]).area()
        area_PBC = Triangle(
            math.sqrt((A[0] - x) ** 2 + (A[1] - y) ** 2),
            math.sqrt((C[0] - x) ** 2 + (C[1] - y) ** 2),
            self.__side_b, B[0], B[1]).area()
        area_PCA = Triangle(
            math.sqrt((A[0] - x) ** 2 + (A[1] - y) ** 2),
            math.sqrt((B[0] - x) ** 2 + (B[1] - y) ** 2),
            self.__side_c, C[0], C[1]).area()

        return math.isclose(area_ABC, area_PAB + area_PBC + area_PCA)

    def overlaps(self, other):
        if isinstance(other, Triangle):
            min_x1 = self.__x
            max_x1 = self.__x + self.__side_a
            min_y1 = self.__y
            max_y1 = self.__y + math.sqrt(self.__side_c**2 - (self.__side_a / 2)**2)

            min_x2, min_y2 = other.get_coordinates()
            max_x2 = min_x2 + other.__side_a
            max_y2 = min_y2 + math.sqrt(other.__side_c**2 - (other.__side_a / 2)**2)

            return not (max_x1 < min_x2 or min_x1 > max_x2 or max_y1 < min_y2 or min_y1 > max_y2)
        return False


class CompoundShape(Shape):
    def __init__(self):
```

```
        self.shapes = []

    def add_shape(self, shape):
        self.shapes.append(shape)

    def area(self):
        return sum(shape.area() for shape in self.shapes)

    def perimeter(self):
        return sum(shape.perimeter() for shape in self.shapes)

    def get_coordinates(self):
        return [(shape.get_coordinates()) for shape in self.shapes]

    def get_points_on_perimeter(self, num_points=16):
        points = []
        for shape in self.shapes:
            points.extend(shape.get_points_on_perimeter(num_points))
        return points

    def contains(self, x, y):
        return any(shape.contains(x, y) for shape in self.shapes)

    def overlaps(self, other):
        return any(shape.overlaps(other) for shape in self.shapes)


class Canvas:
    def __init__(self):
        self.shapes = []

    def add_shape(self, shape):
        self.shapes.append(shape)

    def draw(self):
        for shape in self.shapes:
            print(f"Drawing {shape.__class__.__name__} at {shape.get_coordinates()} with area {shape.area()} and perimeter {shape.perimeter(
            print(f"Points on perimeter: {shape.get_points_on_perimeter(5)}")
```

11. Create a `RasterDrawing` class. Demonstrate that you can create a drawing made of several shapes, paint the drawing, modify the drawing, and paint it again.

```
import math

class Shape:
    def area(self):
        raise NotImplementedError

    def perimeter(self):
        raise NotImplementedError

    def get_coordinates(self):
        raise NotImplementedError

    def get_points_on_perimeter(self, num_points=16):
        raise NotImplementedError

    def contains(self, x, y):
        raise NotImplementedError

    def overlaps(self, other):
        raise NotImplementedError


class Rectangle(Shape):
    def __init__(self, length, width, x, y):
        self.__length = length
        self.__width = width
        self.__x = x
        self.__y = y
```

```
        def area(self):
            return self.__length * self.__width

        def perimeter(self):
            return 2 * (self.__length + self.__width)

        def get_coordinates(self):
            return (self.__x, self.__y)

        def get_points_on_perimeter(self, num_points=16):
            points = []
            for i in range(num_points + 1):
                if i < num_points / 2:
                    x = self.__x + (i / (num_points / 2)) * self.__length
                    y = self.__y
                else:
                    x = self.__x + self.__length
                    y = self.__y + ((i - (num_points / 2)) / (num_points / 2)) * self.__width
                points.append((x, y))
            return points

        def contains(self, x, y):
            return (self.__x <= x <= self.__x + self.__length) and (self.__y <= y <= self.__y + self.__width)

        def overlaps(self, other):
            if isinstance(other, Rectangle):
                return not (self.__x + self.__length < other.__x or
                            self.__x > other.__x + other.__length or
                            self.__y + self.__width < other.__y or
                            self.__y > other.__y + other.__width)
            return False


    class Circle(Shape):
        def __init__(self, radius, x, y):
            self.__radius = radius
            self.__x = x
            self.__y = y

        def area(self):
            return math.pi * (self.__radius ** 2)

        def perimeter(self):
            return 2 * math.pi * self.__radius

        def get_coordinates(self):
            return (self.__x, self.__y)

        def get_points_on_perimeter(self, num_points=16):
            points = []
            for i in range(num_points):
                angle = (2 * math.pi / num_points) * i
                points.append((self.__x + self.__radius * math.cos(angle), self.__y + self.__radius * math.sin(angle)))
            return points

        def contains(self, x, y):
            return math.sqrt((self.__x - x) ** 2 + (self.__y - y) ** 2) <= self.__radius

        def overlaps(self, other):
            if isinstance(other, Circle):
                distance = math.sqrt((self.__x - other.__x) ** 2 + (self.__y - other.__y) ** 2)
                return distance < (self.__radius + other.__radius)
            elif isinstance(other, Rectangle):
                closest_x = max(other.get_coordinates()[0], min(self.__x, other.get_coordinates()[0] + other.__length))
                closest_y = max(other.get_coordinates()[1], min(self.__y, other.get_coordinates()[1] + other.__width))
                return self.contains(closest_x, closest_y)
            return False


    class Triangle(Shape):
        def __init__(self, side_a, side_b, side_c, x, y):
            if not self.is_valid_triangle(side_a, side_b, side_c):
                raise ValueError("Invalid triangle sides provided.")

            self.__side_a = side_a
            self.__side_b = side_b
            self.__side_c = side_c
```

```
            self.__x = x
            self.__y = y

        def is_valid_triangle(self, a, b, c):
            return a + b > c and a + c > b and b + c > a

        def area(self):
            s = self.perimeter() / 2
            return math.sqrt(s * (s - self.__side_a) * (s - self.__side_b) * (s - self.__side_c))

        def perimeter(self):
            return self.__side_a + self.__side_b + self.__side_c

        def get_coordinates(self):
            return (self.__x, self.__y)

        def get_points_on_perimeter(self, num_points=16):
            points = []
            for i in range(num_points + 1):
                t = i / num_points
                if t < 1/3:
                    x = self.__x + t * self.__side_a
                    y = self.__y
                elif t < 2/3:
                    ratio = (t - 1/3) * 3
                    x = self.__x + self.__side_a - ratio * (self.__side_b)
                    y = self.__y + ratio * (math.sqrt(self.__side_c**2 - (self.__side_a - ratio * self.__side_b)**2))
                else:
                    ratio = (t - 2/3) * 3
                    x = self.__x
                    y = self.__y + ratio * (self.__side_c)
                points.append((x, y))
            return points

        def contains(self, x, y):
            A = (self.__x, self.__y)
            B = (self.__x + self.__side_a, self.__y)
            C = (self.__x, self.__y + math.sqrt(self.__side_c**2 - (self.__side_a / 2)**2))

            area_ABC = self.area()
            area_PAB = Triangle(
                math.sqrt((B[0] - x) ** 2 + (B[1] - y) ** 2),
                math.sqrt((C[0] - x) ** 2 + (C[1] - y) ** 2),
                self.__side_a, A[0], A[1]).area()
            area_PBC = Triangle(
                math.sqrt((A[0] - x) ** 2 + (A[1] - y) ** 2),
                math.sqrt((C[0] - x) ** 2 + (C[1] - y) ** 2),
                self.__side_b, B[0], B[1]).area()
            area_PCA = Triangle(
                math.sqrt((A[0] - x) ** 2 + (A[1] - y) ** 2),
                math.sqrt((B[0] - x) ** 2 + (B[1] - y) ** 2),
                self.__side_c, C[0], C[1]).area()

            return math.isclose(area_ABC, area_PAB + area_PBC + area_PCA)

        def overlaps(self, other):
            if isinstance(other, Triangle):
                min_x1 = self.__x
                max_x1 = self.__x + self.__side_a
                min_y1 = self.__y
                max_y1 = self.__y + math.sqrt(self.__side_c**2 - (self.__side_a / 2)**2)

                min_x2, min_y2 = other.get_coordinates()
                max_x2 = min_x2 + other.__side_a
                max_y2 = min_y2 + math.sqrt(other.__side_c**2 - (other.__side_a / 2)**2)

                return not (max_x1 < min_x2 or min_x1 > max_x2 or max_y1 < min_y2 or min_y1 > max_y2)
            return False


    class CompoundShape(Shape):
        def __init__(self):
            self.shapes = []

        def add_shape(self, shape):
            self.shapes.append(shape)
```

```
        def area(self):
            return sum(shape.area() for shape in self.shapes)

        def perimeter(self):
            return sum(shape.perimeter() for shape in self.shapes)

        def get_coordinates(self):
            return [(shape.get_coordinates()) for shape in self.shapes]

        def get_points_on_perimeter(self, num_points=16):
            points = []
            for shape in self.shapes:
                points.extend(shape.get_points_on_perimeter(num_points))
            return points

        def contains(self, x, y):
            return any(shape.contains(x, y) for shape in self.shapes)

        def overlaps(self, other):
            return any(shape.overlaps(other) for shape in self.shapes)


    class Canvas:
        def __init__(self):
            self.shapes = []

        def add_shape(self, shape):
            self.shapes.append(shape)

        def draw(self):
            for shape in self.shapes:
                print(f"Drawing {shape.__class__.__name__} at {shape.get_coordinates()} with area {shape.area()} and perimeter {shape.perimeter(
                print(f"Points on perimeter: {shape.get_points_on_perimeter(5)}")


    class RasterDrawing:
        def __init__(self):
            self.canvas = Canvas()
            self.shapes = []

        def add_shape(self, shape):
            self.shapes.append
```

12. Implement the ability to load/save raster drawings and demonstate that your method works. One way to implement this ability:
    - Overload `__repr__` functions of all objects to return strings of the python code that would construct the object.
    - In the save method of raster drawing class, store the representations into the file.
    - Write a loader function that reads the file and uses `eval` to instantiate the object.

For example:

```
class foo:
    def __init__(self,a,b=None):
        self.a=a
        self.b=b

    def __repr__(self):
        return "foo("+repr(self.a)+","+repr(self.b)+")"

    def save(self,filename):
        f=open(filename,"w")
        f.write(self.__repr__())
        f.close()


def foo_loader(filename):
    f=open(filename,"r")
    tmp=eval(f.read())
    f.close()
    return tmp
```

```
# Test
print(repr(foo(1,"hello")))
```

```
foo(1,'hello')
```

```
# Create an object and save it
ff=foo(1,"hello")
ff.save("Test.foo")
```

```
# Check contents of the saved file
!cat Test.foo
```

```
foo(1,'hello')
```

```
# Load the object
ff_reloaded=foo_loader("Test.foo")
ff_reloaded
```

```
foo(1,'hello')
```

```
import math

class Shape:
    def area(self):
        raise NotImplementedError

    def perimeter(self):
        raise NotImplementedError

    def get_coordinates(self):
        raise NotImplementedError

    def get_points_on_perimeter(self, num_points=16):
        raise NotImplementedError

    def contains(self, x, y):
        raise NotImplementedError

    def overlaps(self, other):
        raise NotImplementedError

    def __repr__(self):
        return f"{self.__class__.__name__}()"


class Rectangle(Shape):
    def __init__(self, length, width, x, y):
        self.__length = length
        self.__width = width
        self.__x = x
        self.__y = y

    def area(self):
        return self.__length * self.__width

    def perimeter(self):
        return 2 * (self.__length + self.__width)

    def get_coordinates(self):
        return (self.__x, self.__y)

    def get_points_on_perimeter(self, num_points=16):
        points = []
        for i in range(num_points + 1):
            if i < num_points / 2:
                x = self.__x + (i / (num_points / 2)) * self.__length
                y = self.__y
            else:
                x = self.__x + self.__length
                y = self.__y + ((i - (num_points / 2)) / (num_points / 2)) * self.__width
            points.append((x, y))
        return points

    def contains(self, x, y):
        return (self.__x <= x <= self.__x + self.__length) and (self.__y <= y <= self.__y + self.__width)
```

```
        def overlaps(self, other):
            if isinstance(other, Rectangle):
                return not (self.__x + self.__length < other.__x or
                            self.__x > other.__x + other.__length or
                            self.__y + self.__width < other.__y or
                            self.__y > other.__y + other.__width)
            return False

        def __repr__(self):
            return f"Rectangle({self.__length}, {self.__width}, {self.__x}, {self.__y})"


    class Circle(Shape):
        def __init__(self, radius, x, y):
            self.__radius = radius
            self.__x = x
            self.__y = y

        def area(self):
            return math.pi * (self.__radius ** 2)

        def perimeter(self):
            return 2 * math.pi * self.__radius

        def get_coordinates(self):
            return (self.__x, self.__y)

        def get_points_on_perimeter(self, num_points=16):
            points = []
            for i in range(num_points):
                angle = (2 * math.pi / num_points) * i
                points.append((self.__x + self.__radius * math.cos(angle), self.__y + self.__radius * math.sin(angle)))
            return points

        def contains(self, x, y):
            return math.sqrt((self.__x - x) ** 2 + (self.__y - y) ** 2) <= self.__radius

        def overlaps(self, other):
            if isinstance(other, Circle):
                distance = math.sqrt((self.__x - other.__x) ** 2 + (self.__y - other.__y) ** 2)
                return distance < (self.__radius + other.__radius)
            elif isinstance(other, Rectangle):
                closest_x = max(other.get_coordinates()[0], min(self.__x, other.get_coordinates()[0] + other.__length))
                closest_y = max(other.get_coordinates()[1], min(self.__y, other.get_coordinates()[1] + other.__width))
                return self.contains(closest_x, closest_y)
            return False

        def __repr__(self):
            return f"Circle({self.__radius}, {self.__x}, {self.__y})"


    class Triangle(Shape):
        def __init__(self, side_a, side_b, side_c, x, y):
            if not self.is_valid_triangle(side_a, side_b, side_c):
                raise ValueError("Invalid triangle sides provided.")

            self.__side_a = side_a
            self.__side_b = side_b
            self.__side_c = side_c
            self.__x = x
            self.__y = y

        def is_valid_triangle(self, a, b, c):
            return a + b > c and a + c > b and b + c > a

        def area(self):
            s = self.perimeter() / 2
            return math.sqrt(s * (s - self.__side_a) * (s - self.__side_b) * (s - self.__side_c))

        def perimeter(self):
            return self.__side_a + self.__side_b + self.__side_c

        def get_coordinates(self):
            return (self.__x, self.__y)

        def get_points_on_perimeter(self, num_points=16):
            points = []
```

```
            points = []
            for i in range(num_points + 1):
                t = i / num_points
                if t < 1/3:
                    x = self.__x + t * self.__side_a
                    y = self.__y
                elif t < 2/3:
                    ratio = (t - 1/3) * 3
                    x = self.__x + self.__side_a - ratio * (self.__side_b)
                    y = self.__y + ratio * (math.sqrt(self.__side_c**2 - (self.__side_a - ratio * self.__side_b)**2))
                else:
                    ratio = (t - 2/3) * 3
                    x = self.__x
                    y = self.__y + ratio * (self.__side_c)
                points.append((x, y))
            return points

        def contains(self, x, y):
            A = (self.__x, self.__y)
            B = (self.__x + self.__side_a, self.__y)
            C = (self.__x, self.__y + math.sqrt(self.__side_c**2 - (self.__side_a / 2)**2))

            area_ABC = self.area()
            area_PAB = Triangle(
                math.sqrt((B[0] - x) ** 2 + (B[1] - y) ** 2),
                math.sqrt((C[0] - x) ** 2 + (C[1] - y) ** 2),
                self.__side_a, A[0], A[1]).area()
            area_PBC = Triangle(
                math.sqrt((A[0] - x) ** 2 + (A[1] - y) ** 2),
                math.sqrt((C[0] - x) ** 2 + (C[1] - y) ** 2),
                self.__side_b, B[0], B[1]).area()
            area_PCA = Triangle(
                math.sqrt((A[0] - x) ** 2 + (A[1] - y) ** 2),
                math.sqrt((B[0] - x) ** 2 + (B[1] - y) ** 2),
                self.__side_c, C[0], C[1]).area()

            return math.isclose(area_ABC, area_PAB + area_PBC + area_PCA)

        def overlaps(self, other):
            if isinstance(other, Triangle):
                min_x1 = self.__x
                max_x1 = self.__x + self.__side_a
                min_y1 = self.__y
                max_y1 = self.__y + math.sqrt(self.__side_c**2 - (self.__side_a / 2)**2)

                min_x2, min_y2 = other.get_coordinates()
                max_x2 = min_x2 + other.__side_a
                max_y2 = min_y2 + math.sqrt(other.__side_c**2 - (other.__side_a / 2)**2)

                return not (max_x1 < min_x2 or min_x1 > max_x2 or max_y1 < min_y2 or min_y1 > max_y2)
            return False

        def __repr__(self):
            return f"Triangle({self.__side_a}, {self.__side_b}, {self.__side_c}, {self.__x}, {self.__y})"


    class CompoundShape(Shape):
        def __init__(self):
            self.shapes = []

        def add_shape(self, shape):
            self.shapes.append(shape)

        def area(self):
            return sum(shape.area() for shape in self.shapes)

        def perimeter(self):
            return sum(shape.perimeter() for shape in self.shapes)

        def get_coordinates(self):
            return [(shape.get_coordinates()) for shape in self.shapes]

        def get_points_on_perimeter(self, num_points=16):
            points = []
            for shape in self.shapes:
                points.extend(shape.get_points_on_perimeter(num_points))
            return points
```

```
        def contains(self, x, y):
            return any(shape.contains(x, y) for shape in self.shapes)

        def overlaps(self, other):
            return any(shape.overlaps(other) for shape in self.shapes)
```

Start coding or generate with AI.