Butler_Nathan_Mini_Project

October 4, 2025

1 Mini Project

1.0.1 Due: 10/03/2025 23:59PM

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

2 Assignment Submission Guidelines

Please follow the guidelines below for submitting your assignment:

1. **Submission Deadline:**
   - All assignments must be submitted **no later than 10/03/2025 11:59 PM**.
   - Late submissions will not be accepted unless prior arrangements have been made by the instructor.
2. **Submission Platform:**
   - Submit your assignment through **Canvas**. Ensure that you upload the files to the correct assignment link.
3. **Required Files:**
   - **Jupyter Notebook file (.ipynb):** Submit the Jupyter Notebook file you used to complete the assignment. The file should contain your well-commented code.
   - **PDF Version (.pdf file):** Additionally, submit a PDF version of your Python code. This can be a printout or export of your script, showing all the code with any necessary explanations or output results included.
4. **File Naming Convention:**
   - Please name your files as follows: `Lastname_Firstname_AssignmentName`
   - Example: `Alex_John_mini_project.ipynb` and `Alex_John_mini_project.pdf`
5. **Technical Issues:**
   - If you encounter any technical issues with Canvas or your submission, please contact the TAs immediately **before the deadline** to avoid penalties.

[1]:
```
import math
```

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. (5 Points)

[2]:
```
class counter:
    # Can be incremented up to a specific value.
    def __init__(self, limit=0):
```

```
        self.count=0
        # specified maximum value.
        self.limit=limit

    # We need an incrememt value
    def increment(self):
        if self.count < self.limit:
            self.count += 1
            return print("Count: ", self.count)
        else:
            return ValueError("Count exceeded limit.")

    def reset(self):
        self.count = 0

# We make an object to hold a method.
count = counter(2)
count.increment()
count.increment()
#count.increment()
#count.reset()
#count.increment()
```

```
Count:  1
Count:  2
```

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. (5 Points)

[3]:
```
"""
 Implement functions to check the value of the counter,
 check the maximum value, and check if the counter is at the maximum.
 (5 Points)
"""
class counter:
    # Can be incremented up to a specific value.
    def __init__(self, limit=0):
        self.__count=0
        self.__limit=limit

    # The data held by increment is now inside a private function.
    def increment(self):
        if self.__count < self.__limit:
            self.__count += 1
            return self.__str__()
        raise ValueError("Count Exceeded Limit")
```

```
    def value(self):
        # counter value
        return self.__count

    def limit(self):
        # returns the limit value
        return self.__limit

    def if_max(self):
        # returns the max counter
        return self.__count == self.__limit

    # Print statement for UI
    def __str__(self):
        return f"counter({self.__count}/{self.__limit})"

    def reset(self):
        self.__count = 0

# We make an object to hold a method.
count = counter(5)
print("Limit:", count.limit())
print("Counter starting value:", count.value())
print(count)
print(count.increment())
print(count.increment())
print(count.increment())
print(count.increment())
print(count.increment())
```

```
Limit: 5
Counter starting value: 0
counter(0/5)
counter(1/5)
counter(2/5)
counter(3/5)
counter(4/5)
counter(5/5)
```

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. (10 Points)

[4]:
```
class Rectangle:
    def __init__(self, length, width, x, y):
        self.__length = length
        self.__width = width
```

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

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

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

    # Accessors (that are like quasi objects?)
    def get_length(self):
        return self.__length

    def get_width(self):
        return self.__width

    def get_x(self):
        return self.__x

    def get_y(self):
        return self.__y
```

[5]:
```
Rectangle = Rectangle(2, 3, 2, 3)
print("Length: ", Rectangle.get_length())
print("Width: ", Rectangle.get_width())
print("X: ", Rectangle.get_x())
print("Y: ", Rectangle.get_y())
print("Area: ", Rectangle.compute_area())
print("Perimeter: ", Rectangle.compute_perimeter())
```

```
Length:  2
Width:  3
X:  2
Y:  3
Area:  6
Perimeter:  10
```

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. (10 Points)

[6]:
```
import math
class Circle:
    def __init__(self, radius, x,y):
        self.__radius = radius
        self.__x = x
        self.__y = y
```

```
    def compute_area(self):
        # Area = pi* r^2
        return round((math.pi)*(self.__radius**2))

    def compute_circumference(self):
        # Circumference = 2* pi* r
        return round(2 * (math.pi) * self.__radius)

    # Accessors
    def get_radius(self):
        return self.__radius

    def get_x(self):
        return self.__x

    def get_y(self):
        return self.__y
```

[7]:
```
Circle = Circle(3, 2, 5)
Circle.get_radius
print("Radius: ", Circle.get_radius())
print("X: ", Circle.get_x())
print("Y: ", Circle.get_y())
print("Area: ", Circle.compute_area())
print("Circumference: ", Circle.compute_circumference())
```

```
Radius:  3
X:  2
Y:  5
Area:  28
Circumference:  19
```

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. (10 Points)

[8]:
```
class common_base:
    # if it is a sub class of common base, we need area implemented.
    # Basically, reference the Lecture using INHERITANCE.
    # We can try to use super here to call from the commmon base class, that␣
 ↪way we don't have to init every variable.
    def __init__(self, x=0, y=0):
        self.__x=x
        self.__y=y

    def compute_area(self): pass
```

```
    def compute_perimeter(self): pass
    def compute_circumference(self): pass
#=============================================================================================
class Rectangle(common_base):
    def __init__(self, length, width, x, y):
        super(Rectangle, self).__init__(x, y)
        self.__length=length
        self.__width=width

    # Computations=========================
    def compute_area(self):
        return (self.__length * self.__width)

    def compute_perimeter(self):
        return (2 * (self.__length + self.__width))
#=============================================================================================
class Circle(common_base):
    def __init__(self, radius, x, y):
        super(Circle, self).__init__(x, y)
        self.__radius=radius
    # Computations=========================
    def compute_area(self):
        # Area = pi* r^2
        return round((math.pi)*(self.__radius**2))

    def compute_circumference(self):
        # Circumference = 2* pi* r
        return round(2 * (math.pi) * self.__radius)

#␣
 ↪Testing=============================================================================================
circle = Circle(radius=3,x=2,y=3)
print("Circle Area:", circle.compute_area())
print("Circle Circumference:", circle.compute_circumference())
print("=========================================================")
rectangle = Rectangle(length=3,width=3,x=2,y=3)
print("Rectangle Area:", rectangle.compute_area())
print("Rectangle Perimeter:", rectangle.compute_perimeter())
```

```
Circle Area: 28
Circle Circumference: 19
=========================================================
Rectangle Area: 9
Rectangle Perimeter: 12
```

6. Implement a triangle class analogous to the rectangle and circle in question 5.(10 Points)

[9]:
```
class Triangle:
    def __init__(self, a,base,c,  height, x, y):
        self.__base=base
        self.__height=height
        self.__x=x
        self.__y=y
        self.__a=a
        self.__b=self.__base
        self.__c=c

    def compute_area(self):
        # Area = .5* base* height
        return (.5)*(self.__base)*(self.__height)

    def compute_perimeter(self):
        return (self.__a + self.__b + self.__c)

    # Accessors to seperate computations␣
 ↪==========================================================================
    def get_base(self):
        return self.__base
    def get_height(self):
        return self.__height
    def get_a(self):
        return self.__a
    def get_b(self):
        return self.__b
    def get_c(self):
        return self.__c
    def get_x_coor(self):
        return self.__x
    def get_y_coor(self):
        return self.__y
"""
Thinking of making a dictionary
to hold values and then use .items()
or something similar inside the function arguement.
"""
Triangle = Triangle(a=5, base=8, c=3, height=5.5, x=2, y=3)
print("Side A:",Triangle.get_a())
print("Side B:",Triangle.get_b())
print("Side C:",Triangle.get_c())
print("Base (also side b):",Triangle.get_base())
print("Height:",Triangle.get_height())
print("X:",Triangle.get_x_coor())
print("Y:",Triangle.get_y_coor())
print("Area:",Triangle.compute_area())
```

```
print("Perimeter:",Triangle.compute_perimeter())
```

```
Side A: 5
Side B: 8
Side C: 3
Base (also side b): 8
Height: 5.5
X: 2
Y: 3
Area: 22.0
Perimeter: 16
```

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 perimeter of the object. (10 Points)

[10]:
```
# Basically, we update the cell in question 6 with additional functionality to␣
 ↪return 16 (x,y) points along the perimeter of the shapes.
# Also, we added the triangle class in this cell to include all shapes.
# We have to import random for this one
import random as rand
class common_base:
    # if it is a sub class of common base, we need area implemented.
    # Basically, reference the Lecture using INHERITANCE.
    # We can try to use super here to call from the commmon base class, that␣
 ↪way we don't have to init every variable.
    def __init__(self, x=0, y=0):
        self._x=x
        self._y=y

    def compute_area(self):
        raise("Area for Shape is not defined.")
    def compute_perimeter(self):
        raise("Perimeter for Shape is not defined.")
    def compute_circumference(self):
        raise("Circumference for Shape not defined.")
    def gather_points(self, limit=16):
        raise("Domain or Range err. Maybe the limit was not set. Try␣
 ↪limit=number less than or eq to 16.")

#= RECTANGLE␣
 ↪================================================================================
class Rectangle(common_base):
    def __init__(self, length, width, x, y):
        super(Rectangle, self).__init__(x, y)
        self.__length=length
        self.__width=width
```

```
# Computations ========================
    def compute_area(self):
        self.__area = (self.__length * self.__width)
        return self.__area

    def compute_perimeter(self):
        self.__perimeter = (2 * (self.__length + self.__width))
        return self.__perimeter

# Points along the perimeter ==================================================
    def gather_points(self, limit=16):
        # Note that length and width are the Domain and Range limits!
        # eg) D in the set of [0,length]
        # eg) R in the set of [0,width]
        self.__limit=limit
        self.__point_list=list()

        y_max = self._y + self.__width
        x_max = self._x + self.__length
        x_min = self._x
        y_min = self._y
        # my thoughts:
        # perimiter has to follow a linear path.
        # so, we need to have points go along the lines
        perimeter=[]
        while len(perimeter)<self.__limit:
            edge = rand.choice(["left", "right", "top", "bottom"])
            if edge == "left":
                pt = (x_min, rand.randint(y_min, y_max))
            elif edge == "right":
                pt = (x_max, rand.randint(y_min, y_max))
            elif edge == "top":
                pt = (rand.randint(x_min, x_max), y_max)
            else:  # bottom
                pt = (rand.randint(x_min, x_max), y_min)

            if pt not in perimeter:
                perimeter.append(pt)

        self.__point_list = perimeter
        return self.__point_list


#= CIRCLE␣
 ↪============================================================================
class Circle(common_base):
    def __init__(self, radius, x, y):
```

```
        super(Circle, self).__init__(x, y)
        self.__radius=radius

# Computations ========================
    def compute_area(self):
        # Area = pi* r^2
        self.__area = round((math.pi)*(self.__radius**2))
        return self.__area

    def compute_circumference(self):
        # Circumference = 2* pi* r
        self.__circumference = round(2 * (math.pi) * self.__radius)
        return self.__circumference

# Points along the perimeter ==========================================
    def gather_points(self, limit=16):
        self.__limit=limit
        self.__point_list=list()

        # This should get a random point in the Domain and Range intervals.
        """
        Needs editing
        """
        circumference=[]
        for i in range(self.__limit):
            theta = (2*math.pi*i)/self.__limit
            x=round((self._x + self.__radius)*math.cos(theta))
            y=round((self._y + self.__radius)*math.sin(theta))
            self.__point_list.append((x,y))
        return self.__point_list

#= TRIANGLE␣
 ↪=============================================================================
class Triangle(common_base):
    def __init__(self, a,base,c, height, x, y):
        super(Triangle, self).__init__(x, y)
        self.__base=base
        self.__height=height
        self.__a=a
        self.__b=self.__base
        self.__c=c

# Computations ========================
    def compute_area(self):
        # Area = .5* base* height
        self.__area=(.5)*(self.__base)*(self.__height)
        return self.__area
```

```
    def compute_perimeter(self):
        self.__perimeter = (self.__a + self.__b + self.__c)
        return self.__perimeter

# Points along the perimeter ================================================
    def gather_points(self, limit=16):
        # Note that base and height are the Domain and Range limits!
        # eg) D in the set of [0,base]
        # eg) R in the set of [0,height]
        self.__limit=limit
        self.__point_list=list()

        A=(self._x, self._y)
        B=(self._x + self.__base, self._y)
        C=(self._x, self._y + self.__height)

        # This should get a point along the interval.
        edge_points=[(A,B),(B,C),(C,A)]
        points=set()

        while len(points) < limit:
            # 3) Choose a random edge
            p1, p2 = rand.choice(edge_points)

            # 4) Pick a random t in [0,1] to interpolate along that edge
            t = rand.random()
            x = round(p1[0] + t * (p2[0] - p1[0]))
            y = round(p1[1] + t * (p2[1] - p1[1]))

            points.add((x, y))
        self.__point_list = list(points)
        return self.__point_list

#============================================================================================
# Question goal to make a function that maps points along the perimeter of the␣
 ↪shapes:

#␣
 ↪============================================================================================
# Testing func␣
 ↪============================================================================================
def testing():
    circle = Circle(radius=5, x=0, y=0)
    print("Circle Area:", circle.compute_area())
    print("Circle Circumference:", circle.compute_circumference())
    print("Circumference Points:\n", circle.gather_points())
```

```
    print("=========================================================")
    rectangle = Rectangle(length=18, width=17, x=0, y=0)
    print("Rectangle Area:", rectangle.compute_area())
    print("Rectangle Perimeter:", rectangle.compute_perimeter())
    print("Periemeter Points:\n", rectangle.gather_points())
    print("=========================================================")
    triangle = Triangle(a=12, base=19, c=13, height=18, x=1, y=2)
    print("Triangle Area:", triangle.compute_area())
    print("Triangle Perimeter:", triangle.compute_perimeter())
    print("Perimeter Points:\n", triangle.gather_points())
testing()
# Testing func␣
 ↪=============================================================================================
```

```
Circle Area: 79
Circle Circumference: 31
Circumference Points:
 [(5, 0), (5, 2), (4, 4), (2, 5), (0, 5), (-2, 5), (-4, 4), (-5, 2), (-5, 0),
(-5, -2), (-4, -4), (-2, -5), (0, -5), (2, -5), (4, -4), (5, -2)]
=========================================================
Rectangle Area: 306
Rectangle Perimeter: 70
Periemeter Points:
 [(11, 17), (15, 17), (18, 1), (0, 16), (18, 9), (0, 15), (18, 5), (17, 0), (18,
10), (18, 7), (5, 0), (8, 0), (0, 0), (0, 7), (14, 17), (8, 17)]
=========================================================
Triangle Area: 171.0
Triangle Perimeter: 44
Perimeter Points:
 [(8, 14), (1, 2), (6, 15), (1, 15), (1, 11), (12, 10), (1, 14), (9, 2), (16,
6), (5, 16), (1, 16), (8, 2), (14, 2), (20, 2), (1, 9), (5, 2)]
```

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. (10 Points)

[11]:
```
# Basically, we update the cell in question 6 with additional functionality to␣
 ↪return 16 (x,y) points along the perimeter of the shapes.
# Also, we added the triangle class in this cell to include all shapes.
# We have to import random for this one
import random as rand
class common_base:
    # if it is a sub class of common base, we need area implemented.
    # Basically, reference the Lecture using INHERITANCE.
    # We can try to use super here to call from the commmon base class, that␣
 ↪way we don't have to init every variable.
    def __init__(self, x=0, y=0):
        self._x=x
```

```
        self._y=y
        x_min = self._x
        y_min = self._y

    def compute_area(self):
        raise("Area for Shape is not defined.")
    def compute_perimeter(self):
        raise("Perimeter for Shape is not defined.")
    def compute_circumference(self):
        raise("Circumference for Shape not defined.")
    def gather_points(self, limit=16):
        raise("Domain or Range err. Maybe the limit was not set. Try␣
 ↪limit=number less than or eq to 16.")
    def check_if_inside(self):
        raise("Error, not implemented in the sub-class")

#=============================================================================================
#= RECTANGLE␣
 ↪=============================================================================
class Rectangle(common_base):
    def __init__(self, length, width, x, y):
        super(Rectangle, self).__init__(x, y)
        self.__length=length
        self.__width=width

# Computations ========================
    def compute_area(self):
        self.__area = (self.__length * self.__width)
        return self.__area

    def compute_perimeter(self):
        self.__perimeter = (2 * (self.__length + self.__width))
        return self.__perimeter

# Points along the perimeter ==================================================
    def gather_points(self, limit=16):
        # Note that length and width are the Domain and Range limits!
        # eg) D in the set of [0,length]
        # eg) R in the set of [0,width]
        self.__limit=limit
        self.__point_list=list()
        x_min = self._x
        y_min = self._y
        y_max = self._y + self.__width
        x_max = self._x + self.__length

        # my thoughts:
```

```
        # perimiter has to follow a linear path.
        # so, we need to have points go along the lines
        perimeter=[]
        self.line_list=["left", "right", "top", "bottom"]
        while len(perimeter)<self.__limit:
            line = rand.choice(self.line_list)
            if line == "left":
                pt = (x_min, rand.randint(y_min, y_max))
                perimeter.append(pt)

            elif line == "right":
                pt = (x_max, rand.randint(y_min, y_max))
                perimeter.append(pt)

            elif line == "top":
                pt = (rand.randint(x_min, x_max), y_max)
                perimeter.append(pt)

            elif line == "bottom":
                pt = (rand.randint(x_min, x_max), y_min)
                perimeter.append(pt)


        self.__point_list = perimeter
        return self.__point_list
# Point checker ================================================================
    def check_if_inside(self, test_x, test_y):
        y_max = self._y + self.__width
        x_max = self._x + self.__length
        x_min = self._x
        y_min = self._y
        self.test_point=(test_x, test_y)
        self.Domain = range(x_min, x_max)
        self.Range = range(y_min, y_max)
        print("Test Point: ", self.test_point)

        # Note, we are working with discrete variables.
        # Continuous are not allowed because it would look like spaghetti.
        if self.test_point[0] == x_min or\
        self.test_point[0] == x_max or\
        self.test_point[1] == y_min or\
        self.test_point[1] == y_max:
            raise ValueError(self.test_point, " is a point in the Perimeter!␣
 ↪(Reached a bound) Try another point.")

        elif self.test_point[0] not in self.Domain:
```

```
            raise ValueError(self.test_point, " is outside the Perimeter!␣
 ↪(Domain error) Try another point.")

        elif self.test_point[1] not in self.Range:
            raise ValueError(self.test_point, " is outside the Perimeter!␣
 ↪(Range error) Try another point.")

        elif x_min < self.test_point[0] < x_max and\
        y_min < self.test_point[1] < y_max:
            print(self.test_point, " is in the sample space.")
        else:
            raise ValueError("Perimeter points are not set! Must call that␣
 ↪first.")


#=============================================================================␣
 ↪
#= CIRCLE␣
 ↪============================================================================
class Circle(common_base):
    def __init__(self, radius, x, y):
        super(Circle, self).__init__(x, y)
        self._radius=radius
        self.h=x
        self.k=y

# Computations ========================
    def compute_area(self):
        # Area = pi* r^2
        self.__area = round((math.pi)*(self._radius**2))
        return self.__area

    def compute_circumference(self):
        # Circumference = 2* pi* r
        self.__circumference = round(2 * (math.pi) * self._radius)
        return self.__circumference

# Points along the perimeter ==========================================
    def gather_points(self, limit=16):
        # Note that x,y are the center of the circle!
        # eg) D in the set of [x-r, x+r]
        # eg) R in the set of [y-r, y+r]
        self.__limit=limit
        self.__point_list=list()

        # This should get a point in the Domain and Range intervals.
        for i in range(self.__limit):
```

```
            theta = (2*math.pi*i)/self.__limit
            x=round((self.h + self._radius)*math.cos(theta))
            y=round((self.k + self._radius)*math.sin(theta))
            self.__point_list.append((x,y))
        return self.__point_list

# Point checker ================================================================
    def check_if_inside(self, test_x, test_y):
        x=test_x
        y=test_y
        test_point=(x,y)

        dx = (x - self.h)
        dy = (y - self.k)
        distance_squared = (dx**2 + dy**2)
        radius_squared = self._radius**2
        if self.__point_list:
            circumference_set = set(self.__point_list)

        if test_point in circumference_set:
            raise ValueError("Point on circumference")

        if distance_squared<radius_squared:
            print(test_point, " is inside the circle")
        else:
            print(test_point, " is outside the circle")



#=============================================================================================
#= TRIANGLE
  ==========================================================================
class Triangle(common_base):
    def __init__(self, a,base,c, height, x, y):
        super(Triangle, self).__init__(x, y)
        self.__base=base
        self.__height=height
        self.__a=a
        self.__b=self.__base
        self.__c=c

# Computations ========================
    def compute_area(self):
        # Area = .5* base* height
        self.__area=(.5)*(self.__base)*(self.__height)
        return self.__area
```

```
    def compute_perimeter(self):
        self.__perimeter = (self.__a + self.__b + self.__c)
        return self.__perimeter

# Points along the perimeter ==================================================
    def gather_points(self, limit=16):
        # Note that base and height are the Domain and Range limits!
        # eg) D in the set of [0,base]
        # eg) R in the set of [0,height]
        self.__limit=limit
        self.__point_list=list()

        A=(self._x, self._y)
        B=(self._x + self.__base, self._y)
        C=(self._x, self._y + self.__height)

        # This should get a point along the interval.
        edge_points=[(A,B),(B,C),(C,A)]
        points=set()

        while len(points) < limit:
            # 3) Choose a random edge
            p1, p2 = rand.choice(edge_points)

            # 4) Pick a random uniform, u
            u = rand.random()
            x = round(p1[0] + u * (p2[0] - p1[0]))
            y = round(p1[1] + u * (p2[1] - p1[1]))

            points.add((x, y))
        self.__point_list = list(points)
        return self.__point_list

# Point checker ===============================================================
    def check_if_inside(self, test_x, test_y):
        x=test_x
        y=test_y
        test_point=(x,y)
        A=(self._x, self._y)
        B=(self._x + self.__base, self._y)
        C=(self._x, self._y + self.__height)

        if self.__point_list:
            perimeter_set = set(self.__point_list)

        if test_point in perimeter_set:
```

```
            print(test_point, " in the Triangle.")

        # Recursion ============================
        def cross(P1, P2, P3):
            return (P1[0] - P3[0]) * (P2[1] - P3[1]) - (P2[0] - P3[0]) * (P1[1]␣
 ↪- P3[1])

        D1=cross(test_point, A, B)
        D2=cross(test_point, B, C)
        D3=cross(test_point, C, A)

        # Ccheck if signs are non-negative
        neg = (D1<0) or (D2<0) or (D3<0)
        pos = (D1>0) or (D2>0) or (D3>0)

        if not (neg and pos):
            print(test_point, " in the Triangle.")
        else:
            print(test_point, " not in the Triangle.")

#=============================================================================
# Question goal to make a function that maps points along the perimeter of the␣
 ↪shapes:

#␣
 ↪=============================================================================
# Testing func␣
 ↪=============================================================================
def testing():
    # Note that x,y are the starting points.
    rectangle = Rectangle(length=18, width=10, x=0, y=0)
    print("Rectangle Area:", rectangle.compute_area())
    print("Rectangle Perimeter:", rectangle.compute_perimeter())
    print("Periemeter Points:\n", rectangle.gather_points())
    print()
    print(rectangle.check_if_inside(test_x=2, test_y=9))
    print("=======================================================")
    circle = Circle(radius=5, x=0, y=0)
    print("Circle Area:", circle.compute_area())
    print("Circle Circumference:", circle.compute_circumference())
    print("Circumference Points:\n", circle.gather_points())
    print()
    print(circle.check_if_inside(test_x=2, test_y=3))
    print("=======================================================")
    triangle = Triangle(a=12, base=19, c=13, height=18, x=1, y=2)
    print("Triangle Area:", triangle.compute_area())
    print("Triangle Perimeter:", triangle.compute_perimeter())
```

```
    print("Perimeter Points:\n", triangle.gather_points())
    print()
    print(triangle.check_if_inside(test_x=2, test_y=3))
testing()
# Testing func␣
 ↪=============================================================================================
```

```
Rectangle Area: 180
Rectangle Perimeter: 56
Periemeter Points:
 [(18, 9), (16, 0), (5, 0), (17, 10), (18, 8), (15, 0), (0, 3), (18, 7), (18,
0), (18, 2), (18, 9), (5, 0), (18, 8), (0, 2), (2, 10), (18, 6)]

Test Point:  (2, 9)
(2, 9)  is in the sample space.
None
=========================================================
Circle Area: 79
Circle Circumference: 31
Circumference Points:
 [(5, 0), (5, 2), (4, 4), (2, 5), (0, 5), (-2, 5), (-4, 4), (-5, 2), (-5, 0),
(-5, -2), (-4, -4), (-2, -5), (0, -5), (2, -5), (4, -4), (5, -2)]

(2, 3)  is inside the circle
None
=========================================================
Triangle Area: 171.0
Triangle Perimeter: 44
Perimeter Points:
 [(18, 4), (1, 18), (6, 2), (1, 2), (20, 2), (15, 7), (18, 3), (1, 17), (16, 2),
(1, 13), (1, 10), (7, 2), (13, 2), (3, 2), (1, 3), (10, 2)]

(2, 3)  in the Triangle.
None
```

9. Add a function in the base class of the object classes that returns true/false testing that the object overlaps with another object. (10 Points)

[12]:
```
# Basically, we update the cell in question 6 with additional functionality to␣
 ↪return 16 (x,y) points along the perimeter of the shapes.
# Also, we added the triangle class in this cell to include all shapes.
# We have to import random for this one
import random as rand
class common_base:
    # if it is a sub class of common base, we need area implemented.
    # Basically, reference the Lecture using INHERITANCE.
    # We can try to use super here to call from the commmon base class, that␣
 ↪way we don't have to init every variable.
```

```
    def __init__(self, x=0, y=0):
        self._x=x
        self._y=y
        x_min = self._x
        y_min = self._y

    def compute_area(self):
        raise("Area for Shape is not defined.")
    def compute_perimeter(self):
        raise("Perimeter for Shape is not defined.")
    def compute_circumference(self):
        raise("Circumference for Shape not defined.")
    def gather_points(self, limit=16):
        raise("Domain or Range err. Maybe the limit was not set. Try␣
 ↪limit=number less than or eq to 16.")
    def check_if_inside(self):
        raise("Error, not implemented in the sub-class")

    def does_the_shape_overlap(self, test_shape):
        x1_min, x1_max, y1_min, y1_max = self.get_bounds()
        x2_min, x2_max, y2_min, y2_max = test_shape.get_bounds()

        if (
            self.x_min <= test_shape.x_max and\
            test_shape.x_min <= self.x_max and\
            self.y_min <= test_shape.y_max and\
            test_shape.y_min <= self.y_max
        ):
            return True
        else:
            return False
#================================================================================
#= RECTANGLE␣
 ↪=============================================================================
class Rectangle(common_base):
    def __init__(self, length, width, x, y):
        super(Rectangle, self).__init__(x, y)
        self.__length=length
        self.__width=width
        self.x_min = self._x
        self.y_min = self._y
        self.y_max = self._y + self.__width
        self.x_max = self._x + self.__length
# Computations ========================
    def compute_area(self):
        self.__area = (self.__length * self.__width)
        return self.__area
```

```
    def compute_perimeter(self):
        self.__perimeter = (2 * (self.__length + self.__width))
        return self.__perimeter

# Points along the perimeter ================================================
    def gather_points(self, limit=16):
        # Note that length and width are the Domain and Range limits!
        # eg) D in the set of [0,length]
        # eg) R in the set of [0,width]
        self.__limit=limit
        self.__point_list=list()


        # my thoughts:
        # perimiter has to follow a linear path.
        # so, we need to have points go along the lines
        perimeter=[]
        self.line_list=["left", "right", "top", "bottom"]
        while len(perimeter)<self.__limit:
            line = rand.choice(self.line_list)
            if line == "left":
                pt = (self.x_min, rand.randint(self.y_min, self.y_max))
                perimeter.append(pt)

            elif line == "right":
                pt = (self.x_max, rand.randint(self.y_min, self.y_max))
                perimeter.append(pt)

            elif line == "top":
                pt = (rand.randint(self.x_min, self.x_max), self.y_max)
                perimeter.append(pt)

            elif line == "bottom":
                pt = (rand.randint(self.x_min, self.x_max), self.y_min)
                perimeter.append(pt)


        self.__point_list = perimeter
        return self.__point_list
# Point checker ============================================================
    def check_if_inside(self, test_x, test_y):
        self.test_point=(test_x, test_y)
        self.Domain = range(self.x_min, self.x_max)
        self.Range = range(self.y_min, self.y_max)
        print("Test Point: ", self.test_point)
```

```
        # Note, we are working with discrete variables.
        # Continuous are not allowed because it would look like spaghetti.
        if self.test_point[0] == self.x_min or\
        self.test_point[0] == self.x_max or\
        self.test_point[1] == self.y_min or\
        self.test_point[1] == self.y_max:
            raise ValueError(self.test_point, " is a point in the Perimeter!␣
 ↪(Reached a bound) Try another point.")

        elif self.test_point[0] not in self.Domain:
            raise ValueError(self.test_point, " is outside the Perimeter!␣
 ↪(Domain error) Try another point.")

        elif self.test_point[1] not in self.Range:
            raise ValueError(self.test_point, " is outside the Perimeter!␣
 ↪(Range error) Try another point.")

        elif self.x_min < self.test_point[0] < self.x_max and\
        self.y_min < self.test_point[1] < self.y_max:
            print(self.test_point, " is in the sample space.")
        else:
            raise ValueError("Perimeter points are not set! Must call that␣
 ↪first.")

# Helper Method for overlapping ======================
    def get_bounds(self):
        return (self.x_min, self.x_max, self.y_min, self.y_max)
#=============================================================================================␣
 ↪
#= CIRCLE␣
 ↪=============================================================================
class Circle(common_base):
    def __init__(self, radius, x, y):
        super(Circle, self).__init__(x, y)
        self._radius=radius
        self.h=x
        self.k=y
        self.x_min=self._x - self._radius
        self.x_max=self._x + self._radius
        self.y_min=self._y - self._radius
        self.y_max=self._y + self._radius
# Computations ========================
    def compute_area(self):
        # Area = pi* r^2
        self.__area = round((math.pi)*(self._radius**2))
        return self.__area
```

```
    def compute_circumference(self):
        # Circumference = 2* pi* r
        self.__circumference = round(2 * (math.pi) * self._radius)
        return self.__circumference

# Points along the perimeter ================================================
    def gather_points(self, limit=16):
        # Note that x,y are the center of the circle!
        # eg) D in the set of [x-r, x+r]
        # eg) R in the set of [y-r, y+r]
        self.__limit=limit
        self.__point_list=list()

        # This should get a point in the Domain and Range intervals.
        for i in range(self.__limit):
            theta = (2*math.pi*i)/self.__limit
            x=round((self.h + self._radius)*math.cos(theta))
            y=round((self.k + self._radius)*math.sin(theta))
            self.__point_list.append((x,y))
        return self.__point_list

# Point checker ============================================================
    def check_if_inside(self, test_x, test_y):
        x=test_x
        y=test_y
        test_point=(x,y)

        dx = (x - self.h)
        dy = (y - self.k)
        distance_squared = (dx**2 + dy**2)
        radius_squared = self._radius**2
        if self.__point_list:
            circumference_set = set(self.__point_list)

        if test_point in circumference_set:
            raise ValueError("Point on circumference")

        if distance_squared<radius_squared:
            print(test_point, " is inside the circle")
        else:
            print(test_point, " is outside the circle")

# Helper Method for overlapping ======================
    def get_bounds(self):
        return (self.x_min, self.x_max, self.y_min, self.y_max)
```

```
#=============================================================================================
#= TRIANGLE
#=============================================================================
class Triangle(common_base):
    def __init__(self, a,base,c, height, x, y):
        super(Triangle, self).__init__(x, y)
        self.__base=base
        self.__height=height
        self.__a=a
        self.__b=self.__base
        self.__c=c
        self.x_min=self._x
        self.x_max=self._x + self.__base
        self.y_min=self._y
        self.y_max=self._y + self.__height

# Computations ========================
    def compute_area(self):
        # Area = .5* base* height
        self.__area=(.5)*(self.__base)*(self.__height)
        return self.__area

    def compute_perimeter(self):
        self.__perimeter = (self.__a + self.__b + self.__c)
        return self.__perimeter

# Points along the perimeter ==================================================
    def gather_points(self, limit=16):
        # Note that base and height are the Domain and Range limits!
        # eg) D in the set of [0,base]
        # eg) R in the set of [0,height]
        self.__limit=limit
        self.__point_list=list()

        A=(self._x, self._y)
        B=(self._x + self.__base, self._y)
        C=(self._x, self._y + self.__height)

        # This should get a point along the interval.
        edge_points=[(A,B),(B,C),(C,A)]
        points=set()

        while len(points) < limit:
            # 3) Choose a random edge
            p1, p2 = rand.choice(edge_points)
```

```
            # 4) Pick a random uniform, u
            u = rand.random()
            x = round(p1[0] + u * (p2[0] - p1[0]))
            y = round(p1[1] + u * (p2[1] - p1[1]))

            points.add((x, y))
        self.__point_list = list(points)
        return self.__point_list

# Point checker ================================================================
    def check_if_inside(self, test_x, test_y):
        x=test_x
        y=test_y
        test_point=(x,y)
        A=(self._x, self._y)
        B=(self._x + self.__base, self._y)
        C=(self._x, self._y + self.__height)

        if self.__point_list:
            perimeter_set = set(self.__point_list)

        if test_point in perimeter_set:
            print(test_point, " in the Triangle.")

        # Recursion ============================
        def cross(P1, P2, P3):
            return (P1[0] - P3[0]) * (P2[1] - P3[1]) - (P2[0] - P3[0]) * (P1[1]␣
 ↪- P3[1])

        D1=cross(test_point, A, B)
        D2=cross(test_point, B, C)
        D3=cross(test_point, C, A)

        # Ccheck if signs are non-negative
        neg = (D1<0) or (D2<0) or (D3<0)
        pos = (D1>0) or (D2>0) or (D3>0)

        if not (neg and pos):
            print(test_point, " in the Triangle.")
        else:
            print(test_point, " not in the Triangle.")
# Helper Method for overlapping ======================
    def get_bounds(self):
        return (self.x_min, self.x_max, self.y_min, self.y_max)
```

```
#␣
 ↪==========================================================================================
# Testing func␣
 ↪==========================================================================================
# make these into functions later.
def testing():
    # Note that x,y are the starting points.
    rectangle = Rectangle(length=18, width=10, x=0, y=0)
    print("Rectangle Area:", rectangle.compute_area())
    print("Rectangle Perimeter:", rectangle.compute_perimeter())
    print("Periemeter Points:\n", rectangle.gather_points())
    print()
    print(rectangle.check_if_inside(test_x=2, test_y=9))
    print("=========================================================")
    circle = Circle(radius=5, x=0, y=0)
    print("Circle Area:", circle.compute_area())
    print("Circle Circumference:", circle.compute_circumference())
    print("Circumference Points:\n", circle.gather_points())
    print()
    print(circle.check_if_inside(test_x=2, test_y=3))
    print("=========================================================")
    triangle = Triangle(a=12, base=19, c=13, height=18, x=1, y=2)
    print("Triangle Area:", triangle.compute_area())
    print("Triangle Perimeter:", triangle.compute_perimeter())
    print("Perimeter Points:\n", triangle.gather_points())
    print()
    print(triangle.check_if_inside(test_x=2, test_y=3))
    print("=========================================================")
    print("Rect Overlap Checker (Circle): ", rectangle.
 ↪does_the_shape_overlap(circle))
    print("Rect Overlap Checker (Triangle): ", rectangle.
 ↪does_the_shape_overlap(triangle))

    print("Circle Overlap Checker (Rect): ", circle.
 ↪does_the_shape_overlap(rectangle))
    print("Circle Overlap Checker (Triangle): ", circle.
 ↪does_the_shape_overlap(triangle))

    print("Triangle Overlap Checker (Circle): ", triangle.
 ↪does_the_shape_overlap(circle))
    print("Triangle Overlap Checker (Rect): ", triangle.
 ↪does_the_shape_overlap(rectangle))
testing()
# Testing func␣
 ↪==========================================================================================
```

```
Rectangle Area: 180
```

```
Rectangle Perimeter: 56
Periemeter Points:
 [(0, 4), (0, 8), (0, 7), (5, 0), (18, 0), (9, 0), (3, 10), (17, 10), (6, 0),
(12, 0), (0, 9), (8, 10), (18, 8), (18, 4), (12, 10), (18, 9)]

Test Point:  (2, 9)
(2, 9)  is in the sample space.
None
============================================================
Circle Area: 79
Circle Circumference: 31
Circumference Points:
 [(5, 0), (5, 2), (4, 4), (2, 5), (0, 5), (-2, 5), (-4, 4), (-5, 2), (-5, 0),
(-5, -2), (-4, -4), (-2, -5), (0, -5), (2, -5), (4, -4), (5, -2)]

(2, 3)  is inside the circle
None
============================================================
Triangle Area: 171.0
Triangle Perimeter: 44
Perimeter Points:
 [(13, 8), (1, 2), (12, 10), (1, 4), (4, 2), (1, 17), (8, 13), (1, 7), (1, 13),
(7, 2), (2, 2), (17, 5), (3, 18), (14, 2), (1, 10), (11, 11)]

(2, 3)  in the Triangle.
None
============================================================
Rect Overlap Checker (Circle):  True
Rect Overlap Checker (Triangle):  True
Circle Overlap Checker (Rect):  True
Circle Overlap Checker (Triangle):  True
Triangle Overlap Checker (Circle):  True
Triangle Overlap Checker (Rect):  True
```

10. Use the `Canvas` class from lecture to 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. (10 Points)

[77]:
```python
import math
import random as rand
class Canvas:
    def __init__(self, width, height):
        self.width = width
        self.height = height
        # Empty canvas is a matrix with element being the "space" character
        self.data = [[' '] * width for i in range(height)]

    def set_pixel(self, row, col, char='*'):
```

```
        self.data[row][col] = char

    def get_pixel(self, row, col):
        return self.data[row][col]

    def clear_canvas(self):
        self.data = [[' '] * self.width for i in range(self.height)]

    def v_line(self, x, y, w, char="*"):
        for i in range(x,x+w):
            self.set_pixel(i,y, char)

    def h_line(self, x, y, h, char="*"):
        for i in range(y,y+h):
            self.set_pixel(x,i, char)

    def line(self, x1, y1, x2, y2, char="*"):
        slope = (y2-y1) / (x2-x1)
        for y in range(y1,y2):
            x= int(slope * y)
            self.set_pixel(x,y, char)

    def display(self):
        print("\n".join(["".join(row) for row in self.data]))

class common_base:
    # if it is a sub class of common base, we need area implemented.
    # Basically, reference the Lecture using INHERITANCE.
    # We can try to use super here to call from the commmon base class, that␣
 ↪way we don't have to init every variable.
    def __init__(self, x=0, y=0):
        self._x=x
        self._y=y
    def compute_area(self):
        raise("Area for Shape is not defined.")
    def compute_perimeter(self):
        raise("Perimeter for Shape is not defined.")
    def compute_circumference(self):
        raise("Circumference for Shape not defined.")
    def gather_points(self, limit=16):
        raise("Domain or Range err. Maybe the limit was not set. Try␣
 ↪limit=number less than or eq to 16.")
    def check_if_inside(self):
        raise("Error, not implemented in the sub-class")

    def does_the_shape_overlap(self, test_shape):
        x1_min, x1_max, y1_min, y1_max = self.get_bounds()
```

```
        x2_min, x2_max, y2_min, y2_max = test_shape.get_bounds()

        if (
            self.x_min <= test_shape.x_max and\
            test_shape.x_min <= self.x_max and\
            self.y_min <= test_shape.y_max and\
            test_shape.y_min <= self.y_max
        ):
            return True
        else:
            return False
    def paint(self,canvas): pass
#=============================================================================================
#= RECTANGLE
 ↪==========================================================================
class Rectangle(common_base):
    def __init__(self, length, width, x, y, char='*'):
        super(Rectangle, self).__init__(x, y)
        self.__length=length
        self.__width=width
        self.x_min = self._x
        self.y_min = self._y
        self.y_max = self._y + self.__width
        self.x_max = self._x + self.__length
        self.char=char
# Computations ========================
    def compute_area(self):
        self.__area = (self.__length * self.__width)
        return self.__area

    def compute_perimeter(self):
        self.__perimeter = (2 * (self.__length + self.__width))
        return self.__perimeter
# Points along the perimeter ==========================================
    def gather_points(self, limit=16):
        # Note that length and width are the Domain and Range limits!
        # eg) D in the set of [0,length]
        # eg) R in the set of [0,width]
        self.__limit=limit
        self.__point_list=list()


        # my thoughts:
        # perimiter has to follow a linear path.
        # so, we need to have points go along the lines
        perimeter=[]
        self.line_list=["left", "right", "top", "bottom"]
```

```
        while len(perimeter)<self.__limit:
            line = rand.choice(self.line_list)
            if line == "left":
                pt = (self.x_min, rand.randint(self.y_min, self.y_max))
                perimeter.append(pt)

            elif line == "right":
                pt = (self.x_max, rand.randint(self.y_min, self.y_max))
                perimeter.append(pt)

            elif line == "top":
                pt = (rand.randint(self.x_min, self.x_max), self.y_max)
                perimeter.append(pt)

            elif line == "bottom":
                pt = (rand.randint(self.x_min, self.x_max), self.y_min)
                perimeter.append(pt)


        self.__point_list = perimeter
        return self.__point_list
# Point checker ================================================================
    def check_if_inside(self, test_x, test_y):
        self.test_point=(test_x, test_y)
        self.Domain = range(self.x_min, self.x_max)
        self.Range = range(self.y_min, self.y_max)
        print("Test Point: ", self.test_point)

        # Note, we are working with discrete variables.
        # Continuous are not allowed because it would look like spaghetti.
        if self.test_point[0] == self.x_min or\
        self.test_point[0] == self.x_max or\
        self.test_point[1] == self.y_min or\
        self.test_point[1] == self.y_max:
            raise ValueError(self.test_point, " is a point in the Perimeter!␣
 ↪(Reached a bound) Try another point.")

        elif self.test_point[0] not in self.Domain:
            raise ValueError(self.test_point, " is outside the Perimeter!␣
 ↪(Domain error) Try another point.")

        elif self.test_point[1] not in self.Range:
            raise ValueError(self.test_point, " is outside the Perimeter!␣
 ↪(Range error) Try another point.")

        elif self.x_min < self.test_point[0] < self.x_max and\
        self.y_min < self.test_point[1] < self.y_max:
```

```
            print(self.test_point, " is in the sample space.")
        else:
            raise ValueError("Perimeter points are not set! Must call that␣
 ↪first.")

# Helper Method for overlapping ======================
    def get_bounds(self):
        return (self.x_min, self.x_max, self.y_min, self.y_max)
# Paint checker ------------------------------------------------------ Need to␣
 ↪use object dimensions here-------------------
    def paint(self, canvas):
        # draw outline
        canvas.h_line(self.y_min, self.x_min, self.__length, self.char)
        canvas.h_line(self.y_max, self.x_min, self.__length, self.char)
        canvas.v_line(self.y_min, self.x_min, self.__width, self.char)
        canvas.v_line(self.y_min, self.x_max, self.__width, self.char)
#=============================================================================================␣
 ↪
#= CIRCLE␣
 ↪============================================================================
class Circle(common_base):
    def __init__(self, radius, x, y, char="*"):
        super(Circle, self).__init__(x, y)
        self._radius=radius
        self.h=x
        self.k=y
        self.x_min=self._x - self._radius
        self.x_max=self._x + self._radius
        self.y_min=self._y - self._radius
        self.y_max=self._y + self._radius
        self.char=char
# Computations ========================
    def compute_area(self):
        # Area = pi* r^2
        self.__area = round((math.pi)*(self._radius**2))
        return self.__area

    def compute_circumference(self):
        # Circumference = 2* pi* r
        self.__circumference = round(2 * (math.pi) * self._radius)
        return self.__circumference

# Points along the perimeter ==========================================
    def gather_points(self, limit=16):
        # Note that x,y are the center of the circle!
        # eg) D in the set of [x-r, x+r]
        # eg) R in the set of [y-r, y+r]
```

```
        self.__limit=limit
        self.__point_list=list()

        # This should get a point in the Domain and Range intervals.
        for i in range(self.__limit):
            theta = (2*math.pi*i)/self.__limit
            x=int(round(self.h + self._radius*math.cos(theta)))
            y=int(round(self.k - self._radius*math.sin(theta)))
            self.__point_list.append((x,y))
        return self.__point_list

# Point checker ================================================================
    def check_if_inside(self, test_x, test_y):
        x=test_x
        y=test_y
        test_point=(x,y)

        dx = (x - self.h)
        dy = (y - self.k)
        distance_squared = (dx**2 + dy**2)
        radius_squared = self._radius**2
        if self.__point_list:
            circumference_set = set(self.__point_list)

        if test_point in circumference_set:
            raise ValueError("Point on circumference")

        if distance_squared<radius_squared:
            print(test_point, " is inside the circle")
        else:
            print(test_point, " is outside the circle")

# Helper Method for overlapping ======================
    def get_bounds(self):
        return (self.x_min, self.x_max, self.y_min, self.y_max)

    def paint(self, canvas):
        points = self.gather_points()
        for x,y in points:
            canvas.set_pixel(y, x, self.char)
#=============================================================================
#= TRIANGLE
#============================================================================
class Triangle(common_base):
    def __init__(self, a,base,c, height, x, y, char="*"):
        super(Triangle, self).__init__(x, y)
```

```
        self.__base=base
        self.__height=height
        self.__a=a
        self.__b=self.__base
        self.__c=c
        self.x_min=self._x
        self.x_max=self._x + self.__base
        self.y_min=self._y
        self.y_max=self._y + self.__height
        self.char=char

# Computations ========================
    def compute_area(self):
        # Area = .5* base* height
        self.__area=(.5)*(self.__base)*(self.__height)
        return self.__area

    def compute_perimeter(self):
        self.__perimeter = (self.__a + self.__b + self.__c)
        return self.__perimeter

# Points along the perimeter ==========================================
    def gather_points(self, limit=16):
        # Note that base and height are the Domain and Range limits!
        # eg) D in the set of [0,base]
        # eg) R in the set of [0,height]
        self.__limit=limit
        self.__point_list=list()

        A=(self._x, self._y)
        B=(self._x + self.__base, self._y)
        C=(self._x, self._y + self.__height)

        # This should get a point along the interval.
        edge_points=[(A,B),(B,C),(C,A)]
        points=set()

        while len(points) < limit:
            # 3) Choose a random edge
            p1, p2 = rand.choice(edge_points)

            # 4) Pick a random uniform, u
            u = rand.random()
            x = round(p1[0] + u * (p2[0] - p1[0]))
            y = round(p1[1] + u * (p2[1] - p1[1]))

            points.add((x, y))
```

```
        self.__point_list = list(points)
        return self.__point_list

# Point checker ================================================================
    def check_if_inside(self, test_x, test_y):
        x=test_x
        y=test_y
        test_point=(x,y)
        A=(self._x, self._y)
        B=(self._x + self.__base, self._y)
        C=(self._x, self._y + self.__height)

        if self.__point_list:
            perimeter_set = set(self.__point_list)

        if test_point in perimeter_set:
            print(test_point, " in the Triangle.")

        # Recursion ============================
        def cross(P1, P2, P3):
            return (P1[0] - P3[0]) * (P2[1] - P3[1]) - (P2[0] - P3[0]) * (P1[1]␣
 ↪- P3[1])

        D1=cross(test_point, A, B)
        D2=cross(test_point, B, C)
        D3=cross(test_point, C, A)

        # Ccheck if signs are non-negative
        neg = (D1<0) or (D2<0) or (D3<0)
        pos = (D1>0) or (D2>0) or (D3>0)

        if not (neg and pos):
            print(test_point, " in the Triangle.")
        else:
            print(test_point, " not in the Triangle.")
# Helper Method for overlapping ======================
    def get_bounds(self):
        return (self.x_min, self.x_max, self.y_min, self.y_max)

    def paint(self, canvas):
        #B=(self._x + self.__base, self._y)
        points = self.gather_points()
        for x,y in points:
            canvas.set_pixel(y,x, self.char)

#class CompundShape(Canvas):
#    def __init__():
```

```
def testing():
    # Note that x,y are the starting points.
    rectangle = Rectangle(length=18, width=10, x=0, y=0)
    print("Rectangle Area:", rectangle.compute_area())
    print("Rectangle Perimeter:", rectangle.compute_perimeter())
    print("Periemeter Points:\n", rectangle.gather_points())
    print()
    print(rectangle.check_if_inside(test_x=2, test_y=9))
    print("=========================================================")
    circle = Circle(radius=5, x=0, y=0)
    print("Circle Area:", circle.compute_area())
    print("Circle Circumference:", circle.compute_circumference())
    print("Circumference Points:\n", circle.gather_points())
    print()
    print(circle.check_if_inside(test_x=2, test_y=3))
    print("=========================================================")
    triangle = Triangle(a=12, base=19, c=13, height=18, x=1, y=2)
    print("Triangle Area:", triangle.compute_area())
    print("Triangle Perimeter:", triangle.compute_perimeter())
    print("Perimeter Points:\n", triangle.gather_points())
    print()
    print(triangle.check_if_inside(test_x=2, test_y=3))
    print("=========================================================")
    print("Rect Overlap Checker (Circle): ", rectangle.
 ↪does_the_shape_overlap(circle))
    print("Rect Overlap Checker (Triangle): ", rectangle.
 ↪does_the_shape_overlap(triangle))

    print("Circle Overlap Checker (Rect): ", circle.
 ↪does_the_shape_overlap(rectangle))
    print("Circle Overlap Checker (Triangle): ", circle.
 ↪does_the_shape_overlap(triangle))

    print("Triangle Overlap Checker (Circle): ", triangle.
 ↪does_the_shape_overlap(circle))
    print("Triangle Overlap Checker (Rect): ", triangle.
 ↪does_the_shape_overlap(rectangle))
testing()
```

```
Rectangle Area: 180
Rectangle Perimeter: 56
Periemeter Points:
 [(18, 10), (18, 4), (18, 9), (18, 3), (0, 5), (5, 10), (15, 0), (10, 0), (18,
9), (8, 10), (18, 5), (0, 10), (5, 10), (0, 10), (0, 2), (0, 2)]

Test Point:  (2, 9)
(2, 9)  is in the sample space.
```

```
None
============================================================
Circle Area: 79
Circle Circumference: 31
Circumference Points:
 [(5, 0), (5, -2), (4, -4), (2, -5), (0, -5), (-2, -5), (-4, -4), (-5, -2), (-5,
0), (-5, 2), (-4, 4), (-2, 5), (0, 5), (2, 5), (4, 4), (5, 2)]

(2, 3)  is inside the circle
None
============================================================
Triangle Area: 171.0
Triangle Perimeter: 44
Perimeter Points:
 [(6, 2), (1, 15), (1, 8), (1, 20), (4, 2), (1, 4), (1, 17), (16, 2), (17, 2),
(18, 2), (2, 2), (7, 15), (3, 18), (3, 2), (1, 19), (1, 9)]

(2, 3)  in the Triangle.
None
============================================================
Rect Overlap Checker (Circle):  True
Rect Overlap Checker (Triangle):  True
Circle Overlap Checker (Rect):  True
Circle Overlap Checker (Triangle):  True
Triangle Overlap Checker (Circle):  True
Triangle Overlap Checker (Rect):  True
```

[86]:
```
my_canvas=Canvas(30,30)
rectangle = Rectangle(length=10, width=9, x=0, y=0, char="*")
circle = Circle(radius=3, x=5, y=6, char="*")
triangle = Triangle(a=12, base=19, c=15, height=12, x=5, y=0, char="*")
print("Rectangle Bounds: ", rectangle.get_bounds())
rectangle.gather_points()
rectangle.paint(my_canvas)
my_canvas.display()
print("Circle Bounds: ", circle.get_bounds())
my_canvas.clear_canvas()
circle.gather_points()
circle.paint(my_canvas)
my_canvas.display()
print("Triangle Bounds", triangle.get_bounds())
my_canvas.clear_canvas()
triangle.gather_points()
triangle.paint(my_canvas)
my_canvas.display()
```

```
Rectangle Bounds:  (0, 10, 0, 9)
***********
```

```
*         *
*         *
*         *
*         *
*         *
*         *
*         *
*         *
**********
```

```
Circle Bounds:  (2, 8, 3, 9)
```

```
    ***
   *   *
  *     *
  *     *
  *     *
   *   *
    ***
```

```
Triangle Bounds (5, 24, 0, 12)
      *   * *    *    *


     *

                 *
              *
     *        *
             *
     *
        *
      **
     *
```

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. (10 Points)
12. Implement the ability to load/save raster drawings and demonstate that your method works. One way to implement this ability:(20 Points)

- 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:

[1]:
```
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
```

[2]:
```
# Test
print(repr(foo(1,"hello")))
```

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

[3]:
```
# Create an object and save it
ff=foo(1,"hello")
ff.save("Test.foo")
```

[4]:
```
# Check contents of the saved file
!cat Test.foo
```

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

[5]:
```
# Load the object
ff_reloaded=foo_loader("Test.foo")
ff_reloaded
```

[5]:
```
foo(1,'hello')
```

[ ]:

[ ]: