The assessment was to be completed within two files that is the main\_from\_file.py and the main\_from\_user.py. Below lies the process of thought and implementation in completing the task assigned.

**Main From File**

The solution to he task started with the main\_from\_file.py. A Polygon class was created having some attributes (data, path\_to\_data and testing\_data) and methods (read\_polygon\_data, testing\_points, create\_mbr\_multiple\_points, create\_point\_mbr, create\_rca\_category, and write\_category\_to\_csv). The plotter module and the rca module which was created was imported into the main\_from\_file.py file.

**Attributes**

The data was to hold the data read from the csv.

The path\_to\_data is the path to the csv to be read by the script.

The testing\_data was the input data that was read and used to test the script.

Code:

def \_\_init\_\_(self, path\_to\_data):

self.path\_to\_data = path\_to\_data

self.data = []

self.testing\_data = []

**Methods**

The **read\_polygon\_data** method was used to to read the csv file and to create the polygon data which was held in the data variable.

Code:

def read\_polygon\_data(self):

with open(self.path\_to\_data) as f:

rd = f.readlines()

for a in rd[1:]:

a = a.replace('\n', '').split(',')[:]

a = tuple(map(float, (a)))

self.data.append(a)

return self.data

The testing\_points was used to read and create the points to be used in testing the script. It was held in the testing\_data variable.

Code:

def **testing\_points**(self, path\_to\_testing\_data):

with open(path\_to\_testing\_data) as f:

rd = f.readlines()

for a in rd[1:]:

a = a.replace('\n', '').split(',')[:]

a = tuple(map(float, (a)))

self.testing\_data.append(a)

return self.testing\_data

The **create\_mbr\_multiple\_points** was used to categorize the points using the minimum bounding rectangle algorithm. This script was to handle a collection of points.

Code:

def create\_mbr\_multiple\_points(self, category\_data):

categorize = []

xs = [point[1] for point in self.data]

ys = [point[2] for point in self.data]

maxX = max(xs)

minX = min(xs)

maxY = max(ys)

minY = min(ys)

mbr = [[maxX, maxY], [minX, minY]]

for ac in category\_data:

if (minX <= ac[0] <= maxX ) and (minY <= ac[1] <= maxY):

acs = list(ac)+ ['inside']

categorize.append(acs)

else:

ac = list(ac) + ['outside']

categorize.append(ac)

return categorize

The **create\_point\_mbr** was used to categorize a point using the minimum bounding rectangle algorithm.

Code:

def create\_point\_mbr(self, point):

xs = [point[1] for point in self.data]

ys = [point[2] for point in self.data]

maxX = max(xs)

minX = min(xs)

maxY = max(ys)

minY = min(ys)

mbr = [[maxX, maxY], [minX, minY]]

if (minX <= point[0] <= maxX ) and (minY <= point[1] <= maxY):

point = point + ['inside']

else:

point = point + ['outside']

return point

The **create\_rca\_category** was used to categorize a point using the ray casting algorithm. It is not fully optimized as it does not give the optimum output. The method here imported the functions from the rca.py file where the ray casting functions have been developed. The ray casting function will be pasted below. The script was referenced from <https://www.geeksforgeeks.org/how-to-check-if-a-given-point-lies-inside-a-polygon/> .

Code:

def create\_rca\_category(self, point):

polys = self.data[:]

# Function call

if checkInside(polys, len(polys), point):

point = list(point) + ['inside']

else:

point = list(point) + ['outside']

return point

**Ray Cating Algorithm**

def onLine(line ,point):

# Check whether p is on the line or not

if (point[0] <= max(line[0][0], line[1][0]) and point[0] <= min(line[0][0], line[1][0]) and

point[1] <= max(line[0][1], line[1][1]) and point[1] <= min(line[0][1], line[1][1])):

return True;

return False;

def direction(Point\_a, Point\_b, Point\_c):

val = (Point\_b[1] - Point\_a[1]) \* (Point\_c[0] - Point\_b[0]) - (Point\_b[0] -

Point\_a[0]) \* (Point\_c[1] - Point\_b[1])

if val == 0:

# Colinear

return 0;

elif val < 0:

# Anti-clockwise direction

return 2;

# Clockwise direction

return 1;

def isIntersect(line\_l1, line\_l2):

#Four direction for two lines and points of other line

dir1 = direction(line\_l1[0], line\_l1[1], line\_l2[0])

dir2 = direction(line\_l1[0], line\_l1[1], line\_l2[1])

dir3 = direction(line\_l2[0], line\_l2[1], line\_l1[0]);

dir4 = direction(line\_l2[0], line\_l2[1], line\_l1[1]);

# When intersecting

if dir1 != dir2 and dir3 != dir4:

return True;

# When p2 of line2 are on the line1

if dir1 == 0 and onLine(line\_l1, line\_l2[0]):

return True;

# When p1 of line2 are on the line1

if dir2 == 0 and onLine(line\_l1, line\_l2[1]):

return True;

# When p2 of line1 are on the line2

if dir3 == 0 and onLine(line\_l2, line\_l1[0]):

return True;

# When p1 of line1 are on the line2

if dir4 == 0 and onLine(line\_l2, line\_l2[1]):

return True;

return False;

def checkInside(poly, n, p):

# When polygon has less than 3 edge, it is not polygon

if n < 3:

return False;

# Create a point at infinity, y is same as point p

exline = [p, (9999, p[1])]

count = 0;

i, start = 0, 0;

while start < len(poly) :

# Forming a line from two consecutive points of polygon

side = [poly[i], poly[(i + 1) % n]];

if isIntersect(side, exline):

# If side is intersects exline

if direction(side[0], p, side[1]) == 0:

return onLine(side, p)

count+=1

i = (i + 1) % n

start += 1

# When count is odd

return count & 1;

The **write\_category\_to\_csv** was used to write out the output of the categorization done using either the minimum bounding or ray casting algorithm.

Code:

def write\_category\_to\_csv(self, category\_data):

with open('n\_output2.csv', 'w') as f:

writer = csv.writer(f)

writer.writerow(['x', 'y', 'category'])

writer.writerows(category\_data)

Code of Ploygon Class:

class Polygon():

def \_\_init\_\_(self, path\_to\_data):

self.path\_to\_data = path\_to\_data

self.data = []

self.testing\_data = []

""" read polygon.csv """

def read\_polygon\_data(self):

with open(self.path\_to\_data) as f:

rd = f.readlines()

for a in rd[1:]:

a = a.replace('\n', '').split(',')[:]

a = tuple(map(float, (a)))

self.data.append(a)

return self.data

""" read input.csv """

def testing\_points(self, path\_to\_testing\_data):

with open(path\_to\_testing\_data) as f:

rd = f.readlines()

for a in rd[1:]:

a = a.replace('\n', '').split(',')[:]

a = tuple(map(float, (a)))

self.testing\_data.append(a)

return self.testing\_data

""" categorize points using minimum bounding rectangle """

def create\_mbr\_multiple\_points(self, category\_data):

categorize = []

xs = [point[1] for point in self.data]

ys = [point[2] for point in self.data]

maxX = max(xs)

minX = min(xs)

maxY = max(ys)

minY = min(ys)

mbr = [[maxX, maxY], [minX, minY]]

for ac in category\_data:

if (minX <= ac[0] <= maxX ) and (minY <= ac[1] <= maxY):

acs = list(ac)+ ['inside']

categorize.append(acs)

else:

ac = list(ac) + ['outside']

categorize.append(ac)

return categorize

""" categorize points using minimum bounding rectangle """

def create\_point\_mbr(self, point):

xs = [point[1] for point in self.data]

ys = [point[2] for point in self.data]

maxX = max(xs)

minX = min(xs)

maxY = max(ys)

minY = min(ys)

mbr = [[maxX, maxY], [minX, minY]]

if (minX <= point[0] <= maxX ) and (minY <= point[1] <= maxY):

point = point + ['inside']

else:

point = point + ['outside']

return point

""" categorize points using ray casting """

def create\_rca\_category(self, point):

polys = self.data[:]

# Function call

if checkInside(polys, len(polys), point):

point = list(point) + ['inside']

else:

point = list(point) + ['outside']

return point

""" write output.csv """

def write\_category\_to\_csv(self, category\_data):

with open('n\_output2.csv', 'w') as f:

writer = csv.writer(f)

writer.writerow(['x', 'y', 'category'])

writer.writerows(category\_data)

**Main From User**

The solution then later progressed to the main\_from\_user.py file. Since the class for creating the Polygon and its needed methods had already been created in the main\_from\_file.py, we then imported it content (class and methods) into the main\_from\_user.py file. The DRY principle and reusabiltiy of code was appreciated here.

We then went ahea to answer the questions and follow the instructions from the questions.

We read the read polygon.csv below.

Code:

poly = Polygon('polygon.csv')

read\_data = poly.read\_polygon\_data()

Insert point information using user inputs.

Code:

x = float(input('x coordinate: '))

y = float(input('y coordinate: '))

Categorizing the points were implemented below using both the MBR and RCA methods from the main\_from\_file module.

Code:

categorize\_mbr = poly.create\_point\_mbr([x, y])

categorize\_rca = poly.create\_rca\_category([x, y])

The plotting of the polygon and the points for visualization was implemented the calling the functions defined within the plotter module and running them.

Code:

plotter.add\_point(categorize\_mbr[0], categorize\_mbr[1], categorize\_mbr[2])

xs = [point[1] for point in read\_data]

ys = [point[2] for point in read\_data]

plotter.add\_polygon(xs, ys)

plotter.show()

Code:

from plotter import Plotter

from main\_from\_file import Polygon

def main():

plotter = Plotter()

""" read polygon.csv """

poly = Polygon('polygon.csv')

read\_data = poly.read\_polygon\_data()

""" Insert point information """

x = float(input('x coordinate: '))

y = float(input('y coordinate: '))

""" categorize point """

categorize\_mbr = poly.create\_point\_mbr([x, y])

categorize\_rca = poly.create\_rca\_category([x, y])

""" plot polygon and point """

plotter.add\_point(categorize\_mbr[0], categorize\_mbr[1], categorize\_mbr[2])

xs = [point[1] for point in read\_data]

ys = [point[2] for point in read\_data]

plotter.add\_polygon(xs, ys)

plotter.show()

if \_\_name\_\_ == '\_\_main\_\_':

main()