Homework #4

Jay Laura

1 Introduction

This week's homework required the creation of 10 random points and four rectangles on a cartesian plane with origin (0,0) and bounded at (200,200). Using three geometries, we were then required to compute whether or not a point was located within one or more of the rectangles . We were instructed to reuse as much code from previous homeworks as possible. I have reused the `Point` class, but not the `Poly` class from the previous weeks homework. While I could have subclassed `Poly`, and added a `Rectangle` method, I believe it makes more sense to write a simple `Rectangle` class for now. I would extend this argument and suggest that writing a series of atomic functions, instead of a class hierarchy, provides the same functionality in a more concise and readable format. It is also not possible to pass class instances or class attributes to the Python MultiProccesing module, as they are not pickable. A functional programming architecture provides easier multiprocessing implementation without having to cast class attributes to local scope, pickable, variables.

2 Program Architecture

As described above, this program is designed in a largely functional style. Two classes exist to reuse previous code, the `Rectangle` class and `Point` class. The program flows as follows.

1. Randomly generate 10 points using the `Point` class. These are attributed with an x and y coordinate.

2. Randomly generate four rectangles.

   To generate a rectangle the upper left and lower right coordinates are determined. By definition, the upper right and lower left coordinates can be derived.

3. Use the ray algorithm to compute whether or not a point falls within a rectangle. This algorithm functions by counting the number of intersections between a rectangles side and a ray (vector) emanating from a point.

   Note that the determinant method for intersection detection was also implemented, but this method fails to function with line segments. Lines are extended an infinite distance and therefore interest in all cases except when parallel.

Point in polygon checking is a computationally slow process that could be greatly improved by first computing the bounding box around the vector emanating from the point and checking for intersection. If the bounding boxes overlap, then the computation could continue to check for the number of times the vector intersects the polygon.

Finally, the output is written. This is accomplished using `.join()` and `.format` to generate a well formatted output.