GPH573 – HW #2
Jay Laura

This weeks homework required that we code a point factory class, generate 4 random points, compute the euclidean distance between all point pairs, find the closest two points, and finally generate a report as a text file. What follows is a description of how my code accomplishes this task.

Lines 6 – 60 encompass the class Point. This class has three methods: __init__, generateGCS, and generatePCS. At create this class requires one argument, the number of points to be generated. Optionally, the user can supply an argument to generate geographic coordinates (GCS) or pixel coordinates(PCS). Pixel coordinates could easily be extended to support UTM Zones or State Plane coordinate reference systems. At initialization, the class checks to ensure that the user has provided an integer number of points, and calls either generateGCS or generatePCS (the default) . Note that generateGCS exits after having printed the generated points as only euclidean distance computation is currently implemented. Also note that generatePCS is constrained to a 1000x1000 integer plane.

Line 62-79, the euclidean_distance function, computes the distance between two points using:

$$d(x, y) = \sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2}$$

Lines 81 -99 are a generator that recursively yields all pairs of points. The main function, lines 101 – 131 parses in the input collected in lines 134 – 137 and controls program flow as follows:

1. Instantiate a Point object (line 105) with a number of points equal to the user supplied integer and set a local (to the class) scope variable as the coordinates attribute.
2. Create an empty dictionary to store coordinate pairs and their distances. (line 108)
3. Call the generator to return a pair of coordinates. (line 109) The generator iterates over all pairs of points. It is designed to allow for sets of size, n. In this case we pass 2, as we need sets containing two points.
   (a) compute the distance between the two points (line 110)
   (b) add the distance to the distance dict with key equals coordinates (string) and value equals distance (line 111)
4. Set the minimum distance to infinity. (line 113) Given the low number of points we perform a brute force check of all points in the dict to get the minimum value. This is not suitable for large numbers of points.
5. Iterate through the dict and grab the key, value pair that has the minimum distance. (lines 114-118) We do not check for identical distances.
6. Open an output file, with name supplied by user as a sys.argv in append mode. This is to avoid overwriting (mode='w'). (line 121)
7. Write variables and headings to the text files. (lines 122 – 131)

Potential improvements could include support for different distance measures, the inclusion of different geometry types, and a less naïve check for minimum distance.