I used the following links to get all of my formulas:

<https://vvvv.org/blog/polar-spherical-and-geographic-coordinates>

<https://en.wikipedia.org/wiki/Geographic_coordinate_system#Earth-centered,_Earth-fixed>

<https://en.wikipedia.org/wiki/Geographic_coordinate_conversion#From_geodetic_to_ECEF_coordinates>

<https://en.wikipedia.org/wiki/World_Geodetic_System#A_new_World_Geodetic_System:_WGS_84>

<https://gssc.esa.int/navipedia/index.php/Ellipsoidal_and_Cartesian_Coordinates_Conversion>

<https://en.wikipedia.org/wiki/Vincenty%27s_formulae>

<http://earth-info.nga.mil/GandG/publications/tr8350.2/wgs84fin.pdf>

I assume an **elliptical** earth, although I do have another simpler solution for a spherical earth.

The POLAR\_SEMI\_MAJOR\_AXIS is the distance radius at the poles

The EQUITORIAL\_SEMI\_MINOR\_AXIS is the radius at the equator

I imported the math module because I needed access to **sin** and **cos**, and conversely the radians functions.

**sin** and **cos** take in radians, so I needed to convert from degrees to radians

The radius always changes based on the latitude (since the earth is an ellipsis). I have named the variables in all lower case and the constants in all upper case. Steps:

* I imported the csv module
* I had to convert the excel files to csv files, I have included both of them although my program only uses the first one.
* I added a newline='' in the csv file
* I only used my first method of determining X, Y, and Z

I assume the csv files are named "%N%\_earthwgs84.csv" with %N% being a positive number, starting from 1 up until the last file.

I assume the number of files to work with is 3.

I have provided the base files so you don't need to make your own, but if you wish to change the files used please be sure to use the right naming scheme.

I import matplotlib.pyplot to be able to graph the lines nicely:

<https://matplotlib.org/tutorials/introductory/pyplot.html>

Please ensure you have installed matplotlib when running the code <https://matplotlib.org/users/installing.html>

matplotlib draw graphs.

Looking in the code:

The function **make\_xyzcoords()** is a function that takes in no parameters but uses the csv wgs84 data file. This function creates files with the naming scheme "%N%xyzcoords.csv", where %N% is the same number as from the earthwgs84 file.

I used string formatting for the file names, to "change" a character in the middle of a string as strings are immutable. None of my opens need to be closed manually because I use with my make\_graph function takes the csv xyzcoords files and outputs a single graph using pyplot. I create a dictionary of values, the key being which number file the values came from, and the values being a list of the distance between the first and the last point (all other points are ignored). This is the distance between 2 points formula.

<https://www.mathplanet.com/education/algebra-2/conic-sections/distance-between-two-points-and-the-midpoint>

I used a stack overflow answer to sort my two lists together (since sorting them individually may not keep the x and y pairs together, it uses the zip function twice. Once to zip() and once to unzip zip(\*)

<https://docs.python.org/3.3/library/functions.html#zip>

<https://stackoverflow.com/questions/9764298/is-it-possible-to-sort-two-listswhich-reference-each-other-in-the-exact-same-w>

I used a stackoverflow answer to sort my dictionary of

<https://stackoverflow.com/questions/613183/how-do-i-sort-a-dictionary-by-value>

I plot 3 lines, starting from the largest line to the smallest, so the larger lines do not cover up the smaller ones.

The x and y lim ensures the bottom left hand corner is the origin, this limit must be placed after the graph is plotted.

The functions make\_xyzcoords needs to only be ran once to create the csv files, so comment that out once you ran it for the first time.

The make\_graph function is ran to create the graph, which will be in a new window.

I first had to make sure I knew how to plot using the library so I experimented myself on making the graph, and overlaying similar lines to see which lines would and would not be displayed.

Then I had to find a way to get the spans. The spans always start from the origin point and go up until the distance of the line, but where the lines are do not matter. The direction or slope does not matter, only the length of the line, so I calculated that using the first and the last points in the csv.

I found a way to make the lists sorted so the x and y pairs do not "break up", so the first point and the last point will be the left and rightmost points. I ignored all the other points in this equation.

I tried to use numpys polyfit and polynomial fit to get the equation for the line, but I had trouble getting the correct values, and I also realized I did not need to look at the whole line, just the two endpoints instead.

I then had to find a way to sort my dictionary, since my experimentations showed that if a long line is above a short one, you can't see the short one.

I found a way to sort the dictionary based on its values, then I reversed it so it was sorted largest to smallest

I would plot the largest value first, then move on to smaller and smaller ones (like stacking a tower of hanoi, a larger piece cannot be on top of a smaller piece).

Finally I outputted the graph and ran my main 2 functions.