Pythonization

# Theory

I started around 4/23/15 with the translation of the JavaScript into Python. This was done for the following reasons:

1. Python would be much faster for crunching data. I noticed that there was a ton of lag time with JavaScript on just short routes (as demonstrated with the timing done in version 3), and that expansion of routes would only work with a better language for speed. Python is infamous for its efficiency with loops and arrays, and since this program is essentially a big data analyzer, Python would be well adapted.
2. That has further carry over with the user experience, since the user does not have to wait nearly as long.
3. I needed a proxy server anyway (using Python) to get data using the APIs, and reduce time to get that data.
4. It would be a good opportunity to simplify the program down to its essential elements, and also experiment with slightly more object-oriented programming. Also, the States project was very disorganized in its structure, so rewriting the program with function flow in mind would help that problem.
5. I already am reasonably good at Python (certainly as good as JavaScript).

There are some drawbacks, though:

1. The urllib library is nowhere near as efficient as Ajax is with getting data from remote websites. It is likely that the decreased speed of getting the information (since Python is not asynchronous, like JavaScript) combined with the theoretical greater number of requests for the same amount of data would outweigh the time gained by getting to go at ten times the request speed.
2. There must be a way to relay data between the HTML application shell and the Python script.

# Structure

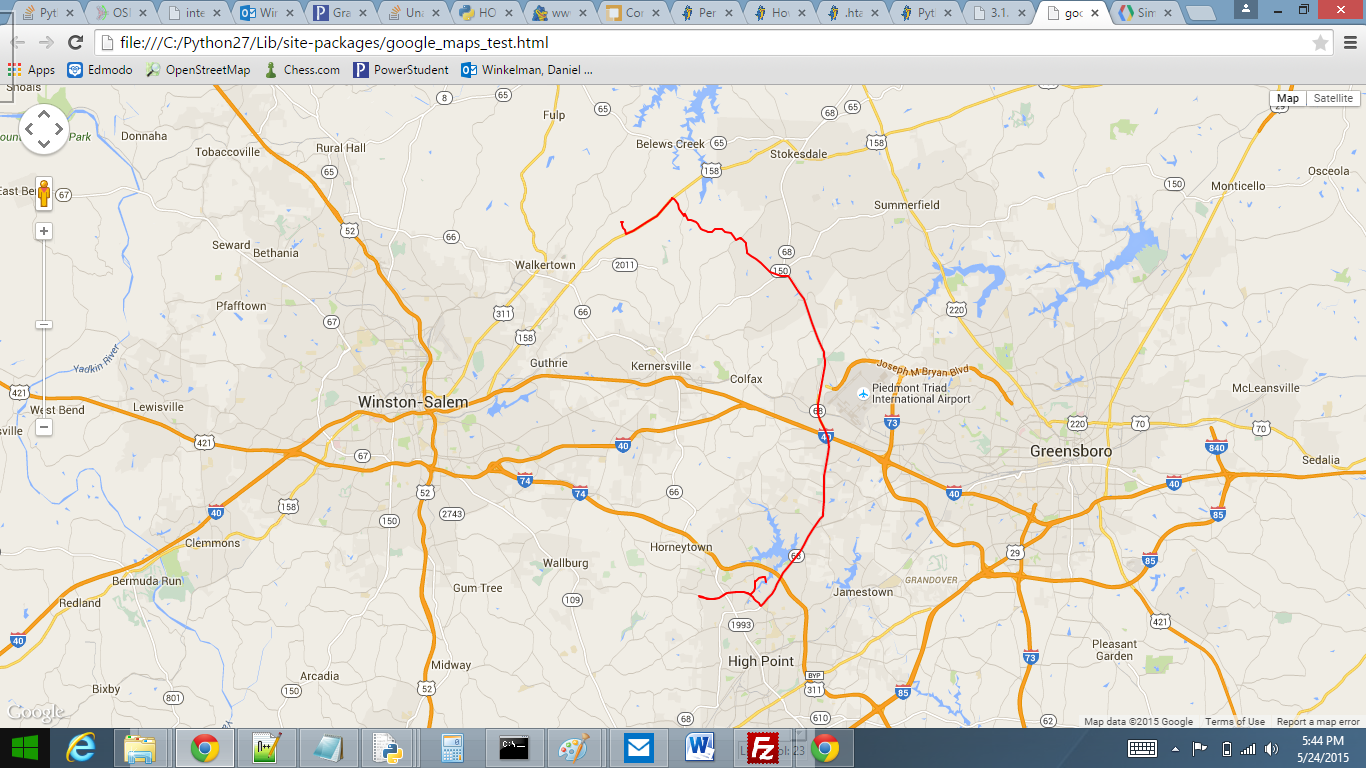
The program was built in this basic structure:

* Classes.py: contains class constructors that represent components of the project that have attributes and there are many of them, such as LatLngs, Pts, Intersections, Connections (to be nested inside intersections), and Bounds. They all have methods associated with them, like distanceTo for LatLngs.
* Util.py: had miscellaneous commands like array operations or formatting functions.
* Ecio.py: input/output stream, which writes console items to an HTML file, as well as parse final outputs into JSON, similarly to Google Directions API.
* Query.py: Classes which control the API requests. It handles errors and timing (to not go over query limits and repeat until success is achieved).
* Main.py: has all the commands for the program. It is essentially a collection of functions that all return (eventually) to the main function. There is not the daisy-chaining and global variables used in the JavaScript; everything is passed by arguments and returns. The functions could theoretically be unified into a single, very long function, but they are broken up to reduce cyclomatic complexity, and separate tasks for the benefit of the programmer.
* The program is not broken up into more than these submodules to simplify the imports needed.

# Observations

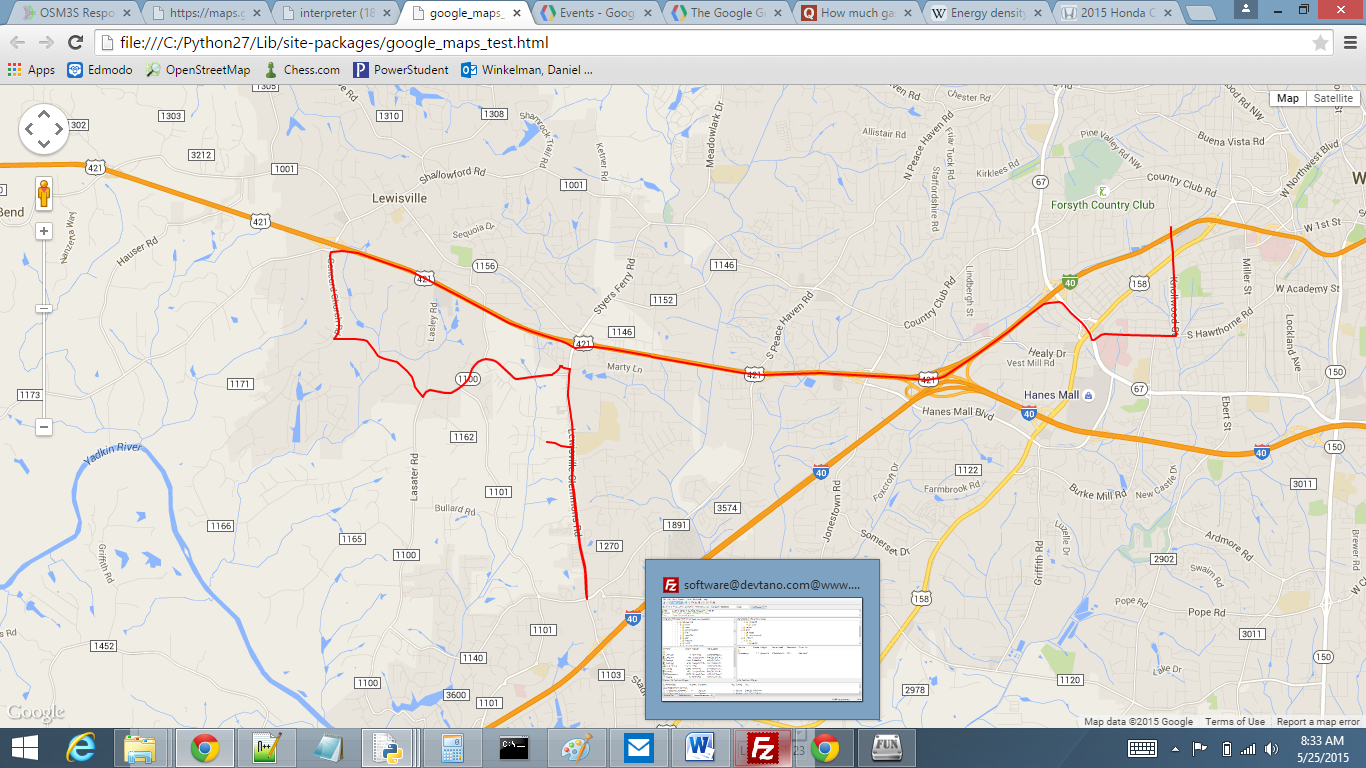
In general, it takes far less time to handle data. For one, using a 2 distance multiplier instead of 1.4 and no intersection count limit in Python, despite the seeming complications, has an average time of 0.02 to 0.03 seconds, whereas in JavaScript is was about 0.03 to 0.04 seconds. Also, the total time to compile intersections is considerably less than the JavaScript times, averaging about 0.2 to 0.3 seconds as opposed to 0.7 to 0.8 seconds.

Python is much easier to use than JavaScript, because the loops are easier to use with structures like tuples, and there are more useful and plentiful built-in functions, which drastically reduces run time. Additionally, this built-in nature of Python make the code faster to write and easier to read. It can also simplify to the point that I have difficulty understanding data structures, since I use almost exclusively arrays, and there is no good way to view internal data structure like the JavaScript console has. A major breakthrough was made on 5/10/15, when there were paths made out of intersections generated using a recursive function similar to Branch().

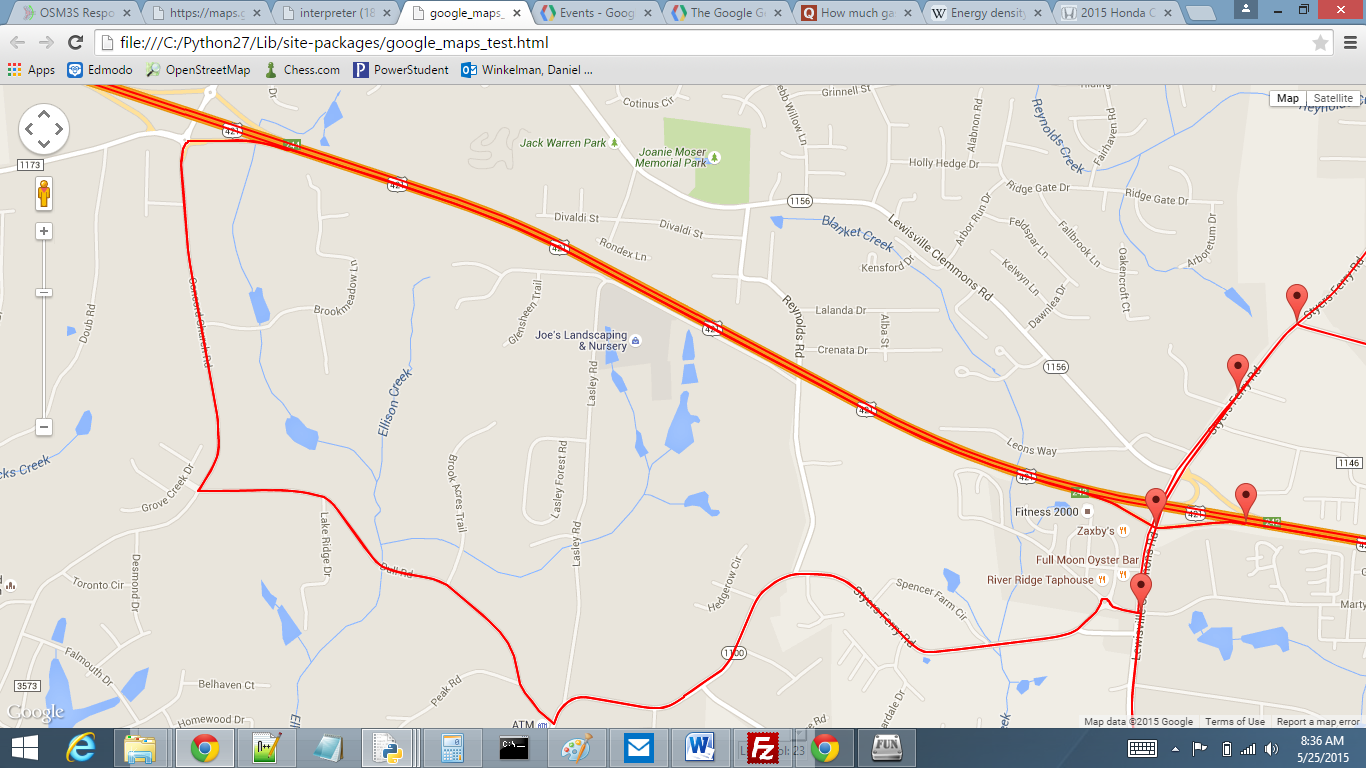
By 5/24/15, force was calculated and optimal routes were beginning to be found. 

This query is clearly significantly larger than the other one the old, Javascript algorithm could handle. It also did the calculations in a decent time.

On 5/25/2015, this route was generated:



When I overlaid the pts and intersections, I discovered that there were two possible ways to go from where the route deviates to where it rejoins with the expected route:



# Other Procedures

On Thursday, 5/7/14 during the TSA meeting, Andy Martinek helped us set up an account for the FTP server devtano.com. devtano.com is the website our advisor, Mr. Pantano, created to facilitate the TSA functions for our chapter. Subsequently, the states version of the project was uploaded, and the main page was edited to create a link to the project.

On Thursday, 5/14/15 during the TSA meeting, Collin and I drafted inquiries to auto companies requesting data on the engines.

On Friday, 5/15/15, all four team members met at Ali’s house, where we worked for 2 hours. The work done was to set up a shell for the website, understand how the website was going to be structured, and come up with ideas for implementing this. I did editing work on the HTML and CSS pages Akshat had prepared. We also came up with the following ideas for the website, which need to be implemented:

* To make a full-page shell for the Google Map interface, this would be given to me when it was finished to integrate the scripting.
* To make a single HTML file with the header and footer, and have an iframe be the content. This would remove redundancy with copying the header to each document, and therefore simplify the process of editing and also the uploading to the FTP server.
* To redo the End-User Documentation formatting, to match that of the templates Akshat had already created.
* Once preliminary versions of these documents were created, they would be uploaded to devtano.com, so that editing could be made conveniently be all members of the team.

On Thursday, 5/21/15, there was another meeting. Only Daniel, Ali and Akshat were present. Daniel attempted to find a way to accurately calculate force factoring in acceleration and deceleration. A plan was also formulated for reducing query sizes, with the circular net used to generate a Google Maps Directions API Query being changed to a diamond, since ellipses would be difficult to calculate. This would remove excess data that would likely not be used, and would especially speed up elevation calculation, since there would be significantly (arbitrary guess: 15-30%) fewer points to calculate elevation data for. It would also take less time to gather the Directions API data.

On Sunday, 5/24/15, I spent about 2.5 hours trying to find a way to have the Python script run from my Raspberry Pi, laptop or devtano.com so that it could be accessed remotely. I know how to do a local host, but that is irrelevant. I posted a question on Stack Overflow, asking for detailed instructions for setting up the script on the Host Gator site.

By Tuesday, 5/26/15, a preliminary version of the Python was finished. Some ideas for optimizations:

* May 9
  + make ellipse instead of sphere
  + chunk the google-optimized route and solve for several midpoints
  + make initial routes, choose most promising and remove unnecessary points (DONE: 4.1)
  + interpolate elevation data (DONE: 4.2)
  + calculate fuel saved, money saved, etc.
* May 23
  + create another module for classes to control query rates (improve readability, accuracy) (DONE: 4.1)
  + collect road data from returned query, not geocoding (DONE: 4.1)
  + make instructions for multiple routes, but choose drastically different (instead of slightly different, append-something-here routes)

# Next Versions

The next version will cut down on the amount of elevation data collected by using interpolation and compressed networks. This will additionally reduce time importing the Directions API data.

The version after that will improve pruning measures, and not calculate detailed forces, instructions, etc. for very similar routes. This will be done by comparing the number of shared consecutive intersections.