Jason Becker

MIDS INFO-W18: Project 1

**Narrative:**

My project for this course was to use object oriented programming to model a fantasy Formula One league. I selected such a niche topic because I’m in a fantasy F1 league, and I wanted to spend my time making something that I could use and enhance after the project is over. The 2016 season started last weekend, and this should help cut down on the time that I spend manually pulling results and updating charts and graphs throughout the year.

The first challenge I encountered was establishing a workflow for the program. Since this isn’t a game with strictly ordered steps, I had a lot of freedom in designing the flow of user interactions. Part of the challenge was making it fit in with data sources and steps that are completed entirely outside of the script. Several changes were made to the Google Form that is used to collect a player’s driver selections, and that caused problems downstream.

One of the larger components of the project involves opening various csv files, reading the data, modifying it, and then rewriting it back to the same file. There were several instances of whitespace sneaking into the output, or newline characters modifying the structure of the file, but I was able to use the csvwriter and reader to help keep each iteration of the file uniform.

At the outset I assumed the race results would be coming from a handmade csv file, but the prospect of setting up all of those files was very daunting. I found that the Wikipedia pages for each race were kept very standardized all season, so I chose to add a web scraping feature so that this next season would be easy to keep up with. Unfortunately, some table entries included references to footnotes, so I had to use an HTML cleaner object to sanitize all of the superscript elements. The upside is that after completing that portion of my module, I used the same methods to build two webscraping scripts at work and saved some interns hours of crawling through pages to copy and paste out location information. I’ve already tested the 2016 links, and the script is executing properly.

One of the reasons why I enjoy Formula One is its international scope, with many of the drivers coming from European and South American countries. Unfortunately, that means several drivers with non ASCII characters in their names. I began switching over to the Unicode representations of their names, but still needed the user to be able to pass in “Raikonnen” instead of “Kimi Räikkönen”. I added in a few dictionaries to allow the script to move between the two representations of the same driver. This ended up paying off when trying to annotate scatter plots with shortened names.

When the module reads in data from a google form, all of the names have the driver’s cost wrapped in parentheses as a suffix. Although I could have used string splicing and indexing to trim the names down, I chose to use RegEx. The extra time spent practicing with RegEx ended up being a great way to reinforce a topic that we covered very quickly in class.

I was not able to complete every feature that I have in store for this module, but the core functionality will make it very easy to extend the scope. In order to communicate standings, I would like to switch out the matplotlib graphs out for PowerPoint slides. If there are several slides with different visualizations, it would be feasible to build an XML parser that can update the slides in real time to create a fantasy dashboard that could be viewed alongside the races.

**Usage:**

From the command line, navigate to this project folder and execute: “Python Jason\_Becker\_Project\_1.py pull Australia”

You will be prompted to download the results from Wikipedia. Currently the Results folder is empty, but if you select “Y”, two files will be generated in that folder. In case of any problems, copies of all race results are stored inside a backup folder.

At this point the script will instantiate a Team object and a Race object, in this case specific to Australia. The Team attributes are pulled from the corresponding team.csv in the Teams folder. These are all copy and pasted results from a Google form that users fill out before every race. You will see the team selections printed out on the screen. All teams must be submitted before qualifying begins, so there isn’t typically a need to update the teams but you have the option if necessary. If you select “Y”, you can either add, drop, or replace drivers. Adding and dropping require one team and one driver, while replacing requires two drivers. Each element must be separated by a space. For example, try: “replace Jason Rosberg Maldonado”. This will drop Nico Rosberg from Jason’s team, and then add Pastor Maldonado. If you make a spelling mistake, you should receive an error message and the opportunity to retry. Every change invokes the push() method, causing the csv file to be saved with your changes. If necessary, copies of the Team.csv files are saved in a backup folder.

Next, the script will print out qualifying results, and prompt you to make updates if necessary. Drivers earn points for qualifying higher, but typically the Wikipedia results are very accurate. Select yes, you can pass in the driver and their new qualifying position, such as “Massa 4”. Just make sure to set Massa back to 3 before continuing. Every change that you make will be pushed to the Qualifying.csv file in the results folder, and you can always re-execute this script and download a new file from Wikipedia if you want a fresh start.

Next the script will print out the grid results. Here’s where we typically need to make updates in order to match scoring. As an example, Bottas had a grid position of “—“ according to Wikipedia. It turns out he had an injury and didn’t actually line up on the grid. The script didn’t know how to score “—“ so he got dumped to the back of the grid at 19. Go ahead and update him to “DNS” (did not start). You can loop through this to make as many updates as are necessary.

You have the option to see the race results, because you don’t want to spoil the results if you haven’t seen it yet. Typically I would run this script at the beginning of a race so I can make sure the reported grid positions have lined up with where they started the race at.

Lastly, you can update the race positions if there was an error, but this is typically very accurate. The csv files are automatically updated to include any changes you implemented.

Next, it’s time to push this race into the season. Results are tracked in Season\_Teams.csv and Season Drivers.csv. If you execute “python Jason\_Becker\_Project\_1.py push Australia” it will push these results into the season documents. If you open Season\_Drivers, you’ll see the each driver was scored according to their performance. For details on how the scoring takes places, see the Project Proposal document saved in this folder. Also, the cost for the next race (Malaysia) is recorded as well. The drivers and their costs can be copied into a Google form for polling participants for the next race. Each player also has their drivers and scores appended to the Season\_Teams.csv file. There is a total line stored as well. The total may be different from the sum of the drivers because of bonus points for not spending your entire budget. See the Project Proposal for more details.

Lastly, you can chose to view season results and then select one of four graphs. The first is the points earned by each driver. The second is the fantasy points that each driver can earn. Third is the total points for each fantasy team. Lastly, you can see each drivers average fantasy points compared to their current cost. This is a great way to find drivers that are currently undervalued.