Paige Crowl

2 CR

Final Project Narrative

          This project changed shape somewhat from my original proposal, but in the end I created a functional program that performs a valuable function for my library. I had originally planned to make a program that would “take input CSV files from Alma, format them into specific templates depending on what the file represents (inventory data, usage statistics, etc), convert the text into HTML, and place these output files into a specific folder on the library’s web server, in a format that is readable and helpful to other library employees who use these statistics”. Upon speaking with my boss, I decided instead to make a program that would focus exclusively on one CSV report from Alma: the new acquisitions report. In my final version of the program, I process the CSV data, create separate lists of items based on call number, and finally join all these lists together into one formatted text file. I also decided to switch to using Markdown rather than HTML as my primary web content language. Running it through a free converter (i.e. <http://subdillinger.herokuapp.com/>) produces valid HTML output that can be uploaded to the library’s website.

    Much of the real work in the project lay in formatting the data, and deciding how to make formatting adjustments for data that wasn’t complete. Ultimately, I decided to not include new items in my lists that did not have titles listed (which should not happen very often, as it is a cataloging error), because I felt having a new item on the list without a title would prove too confusing for readers. In a more common case, items that are missing authors (i.e. the Bible), I simply did not include a line for “By:” and skipped directly to adding the call number. Additionally, some of the titles end in an / and some end in a period (and I could not find a rhyme or reason for why one or the other was the case), so I sorted those and cleaned them up to match in the final file. I found Markdown an ideal system to use in Python: it relieved me of the difficulty of correctly intenting the HTML code, and for the simple formatting I wanted to do the functionality was perfect.

Let’s follow a single item through the process, so we can take an in-depth look at how the program works. The very first item on the list is “The middle Platonists : 80 B.C. to A.D. 220 /”, with no author listed, and a call number of “B505 .D55 1977”. It’s read into csv\_infile and then appended to listofitems[]. Since its title ends in “ /”, it’s run through a small section that cuts those two characters off. Next it is diverted: since it is a B-BP call number, it goes through six if/elif checks before it finds its “assigned” method. Here it goes into the method BtoBP() and its item formatting process begins. Since it has no author, its item description will consist of only the title and call number. When it finishes, it will be appended to the btobplist. When the btobp list is finally called on to be added to the text file, the item description is printed to the final output file, neatly formatted. Everything is ordered and in its place, formatted to be converted to HTML!

    That all said, I do have criticisms of my code: it’s somewhat clunky and brute-force to enumerate methods and blocks of code for every call number range, and I believe that I could more elegantly reduce it to one method by expanding the parameters so that the range details are entirely filled in every time the method is called, instead of spelling them out explicitly within the method. The code is long, and in my opinion there are certainly places its length could be reduced by as much as half with a little finagling. As it is, the code functions well, and I tried to safety-proof it as much as possible. Big, reliable, and ugly is better than efficient, small, and sometimes glitchy. I know I’ll certainly be spending some time over the break to keep tinkering with it and cleaning it up before I present it to my boss in the next few weeks!