Disassembling Dracula: The Light and Dark Hours

The Dracula Project helped hone my ability to "get at things" in a text document and think about how to go about getting at things using various patterns and stacks of patterns and/or various modifications of patterns. This project consisted of 3 main components. I needed to do the following:

- isolate and grab out the chapters from the body of the text *Dracula*
- isolate and grab out the chapter names from the table of contents to then transform into the required format to be used as file names

- write out the chapters individually to newly created files, named with corresponding file names in the designated format

  At first, I had to play with reading in the file and figuring out when and why to loop and when and why to work with contents as is (read in – as string or list). I don't know if that is a common problem but there came a time when I could finally visualize using what I would think of as a "frozen-retained chunk" of information (performing a read operation on a file object and storing in a variable) vs a "moving-disappearing chunk" of information (loop over information). Of course, there are many reasons for looping and types of loops (reference, repetition, filtering), but when it comes to grasping "doing things with files," there was a distinct cross over that accompanied making the distinction between manipulating "frozen-retained chunk" vs manipulating "moving-disappearing chunk." Once I made that leap, I could get more into the nitty gritty about what I needed to do and how I needed to do it.

  Grab File Names from Table of Contents

  A couple points: I knew I needed to search and filter for certain items and my intent was to build a list of the info I needed for the file names and ignore the rest of the text. Because I needed chapter names which existed on certain lines within the target substring (the table of contents), for this particular objective, lines had meaning. I would need to loop over them. Table of contents already builds meaning into lines. This made intuitive sense. So, start the looping.

  Secondly, I understood what command allowed me to read in material as a string vs a list. When getting out my table of contents, I needed to think about a "frozen-retained chunk" so I read in *Dracula* and saved it as one big string. Then I used search methods to save

key positions so I could slice out a chunk of that big string and create a target substring to operate on. The target sub-string I needed was the table of contents. So, I considered contenders for getting my substring saved. Any unique strings I could search for and save? I noticed “CONTENTS” is unique and starts off my target substring.

“338” was also unique and ends my target substring. I ran these to find the indices and did some slicing. Since I used the [string.find(“substring”)] method and that returns the first position of the string, I needed to add back in the positions for “338.” First, I just added 3 to the string and saved that position to a variable. I thought there was a better way to rely less on the specific code. Eventually, I thought to operate on len(“338”) instead. That method would work no matter what the string was. Save the length of the string instead of manually counting the length and adding in the number. That seemed like an improvement toward abstraction.

I then saved the target positions returned and used those as my beginning and end slicing index positions to get out the table of contents.

Once I had that target string isolated, I could move into the looping phase to search for chapter names to build up file names. I needed to identify certain lines: lines that had a length (not empty lines) and of those lines, only the ones that ended in digits (the lines that had pages numbers which meant they represented the lines with the chapter names.

Once these lines were identified, I needed to split the lines into words (split on the spaces) to get out the words and digits. I then needed to build a list of the words with page numbers dropped, so a list of each of these lines, filtering out the last “word,” in each line, which represented the digits/pages numbers.

From this point, I called a cleanline function on each filename in the filename list which replaced undesired punctuation with desired punctuation and then called a dressline function on each cleaned filename in the filename list which added the rest of the needed string pieces to build the final string names for the chapter filenames.

Target Chapter Start and End Points:

Start point: I decided to use “DRACULA” as my target string to search for as a start point. I knew that I needed to use the first occurrence after the Table of Contents since “Dracula” appears plenty of times before this point. I used former code and targeted the end of the Table of Contents. I then saved that position and passed that point in as the second parameter to the Find method, which will then search for the first occurrence of given string in first parameter position (“DRACULA” in this case) after the string passed in the second parameter.

End Point: Since “THE END” was a unique target string that identified the place I needed to identify, I used this string and my previous technique of adding the length of this string to target my needed precise position. I saved this position to get at the end of the chapters.

Next, I sliced out the chapters using string slicing methods.

I returned a list of chapters by splitting on the string “CHAPTER”.

When it came to the isolating my target substrings (ie; chapters) as to send to separate files, I initially forgot that the split method collapsed down my “CHAPTER” string and I hadn’t added that back in yet. That was a correction I needed to make which helped drive home the notion of “collapsing.”

I built two lists: one for file names and one for chapter content.

The last step was assembling code that would reference the items in my chapters list and send that content to created outfiles (the items in my file names list).

I used a reference range loop to accomplish this and indirectly looped through the chapters list, grabbing each item. I created a file io object, opening the filenames in the list (referenced through the range loop) in write mode and saving to this objecting one by one in the loop. I then directed the items in my chapter list (the chapter content) one by one in the loop to the newly created files, looping through the filename list to create them with the desired names one by one. I then printed out the content.

THE END