Instructions for use
Make a copy of this template each time you need to write a Design Document. Then ***delete all italicized instructions*** *and fill out the rest of the document.*

Alex Weirauch
CS256
Lab Assignment 6
11/12/2023

Lab Assignment 6 Design Document

Introduction *(1-5 sentences, starting with lab 2)*
The Phantom Tollbooth Info Counter will feature a couple of functions that can be used to dissect the information and more specifically the words of the book “The Phantom Tollbooth”. Likely the Info Counter will feature a list of words contained therein, their frequency, and any other notable statistics. Pronouns, articles, prepositions, and conjunctions will be considered separately, based on the users need.

Functional Requirements *(starting with lab 3)*

1. Inputs
   All input in this program will be received from the book the phantom tollbooth. Any other user input will only narrow or broaden the scope of analysis respectively. The book itself will be obtained by generating a copy of it as a Python file inside of Github. For any given book to be analyzed, the actual words should be stored within a function definition inside of a Python file.

2. Program:
   The book itself will be decomposed in a seemingly endless list of strings, that will then be analyzed. Before analysis is conducted, the string count will be reduced based on user input: The idea is to write individual functions looking at all conjunctions for example within the strings and removing them optionally, possibly through a “Do you want to remove all conjunctions before analyzing this text” prompt or similar.

   Then all individual strings are counted and compared. This will likely require a nested loop that creates a counter for each string passed and then outputs the top fifty strings by frequency.

3. Outputs:
   The program will output some generic statistics, depending on how I feel when building the functions. At the very least those statistics will include word frequency, overall word counts, maybe something like “words-per-paragraph” count.

4. Variables/Branches:
   The only branching will occur based on whether the user wants to remove certain word categories such as conjunctions or pronouns, and with that information analyze a different set of strings when beginning the statistical analysis.

Design Requirements *(starting with lab 4)*

The to-be-analyzed text (which should be stored inside a function in a .py file) will be called on by a function get_strings(text), which considers each space in the text as separators and converts all individually grouped letters into new strings. I am not entirely sure if it makes more sense to first build remove_wordtype() functions (before passing them through a get_strings() environment. Either way, here or thereafter will be a few remove_wordtype(text) functions (example: remove_conjunctions(text)), that pass either a list or the entire text through themselves and remove every instance of them to output a new text.

Brainstorming this makes me want to structure it like this:
remove_wordtype(text) functions stand at the beginning. Those are optional and require user input to be executed. They are all nestled loops that count individual instances of the words they are removing. Those loops need to understand how to separate the initial text and therefore contain a line of code that will be similar to the next function (get_strings()). Output is the same text as before.

get_strings(text) passes whatever text is leftover after the remove_wordtype() selections are made into a function that decomposes the gigantic string fed in, into much smaller ones that only contain individual words and build an equally gigantic list. Again, depending on “live coding” experience this may be changed with the remove_wordtype() funciton.

analyze_strings(string_list) takes the decomposed text and analyzed string frequency by comparing strings to one another. I am not exactly sure how to build the nestled loop functions required for this yet, but the idea is to pass each element of the list created in get_strings, and compare it to a variable output variable that contains all the previously analyzed elements and adds a count if necessary. Since we’d like to see at least 50 words, that may end up including a whole lot of lines. Maybe something containing [:49] tokens will work. Admittedly most unsure about this technical aspect of the program, and based on how things go, this part of the program may change most significantly.

output_strings(analyzed_strings) may be the final part of the previous function or a standalone function, that utilizes mainly print functions to generate a somewhat readable table (would love a dropdown table type thing but not sure I know how to build that yet) showcasing the fifty most common words and their frequency. This part should also reemphasize which wordtypes have been removed for better comparison. The table would probably look cute with some general information in the first few lines, such as word or character count, depending on what the analyze_strings() function ultimately outputs.

Testing Predictions Results *(starting with Lab 6 and on)*

My best bet here will be to evaluate the decomposed string list without any of the remove_wordtype() functions active and compare the output with a function that uses the raw_text() before it is converted into strings and see if I obtain similar results. In an ideal world both are exactly equal and I can introduce a remove_wordtype() function to both that do the exact same thing to their respective sets of data and still output the same thing.
I predict there will be an awful lot of “junk data” in my first iteration that will have to be removed, for example strings containing special signs such as “.,]\$%” etc.. Removing those will be somewhat time consuming, but maybe I can introduce a function right at the beginning of the program that only considers those binary values that are tied to letters and ignore all others.
Stay tuned for actual results…

Detail your testing strategy to ensure your program runs correctly. Be sure to include the following:

1. *What variations of input did you decide to test?*
    a. *(Variations in testing data should be used to ensure different parts of the program are run.)*
2. *What did you predict would happen with your tests?*
3. *What were the actual results?*

Reflection and Questions

Reflect about your experience designing this program. Be sure to answer all the following questions:

1. *How did you first try to decompose the program (by input->processing->output? By sequence? Randomly?)*
2. *What was the first thing you figured out?*
3. *What questions did you uncover during your design process?*
4. *Were you able to answer all your questions? Do you have any questions you don't have the answers to yet? If so, what are they?*
5. *Did you make a 2nd attempt to decompose the problem? What changes did you make to your strategy from the 1st strategy?*
6. *How complete of a design do you think you achieved (25%? 50%? 100%).*
7. *What part do you think you understand the most? What part do you understand the least?*

Collaboration *(starting with lab 3– optional)*

Include a paragraph for each collaborator and what ideas of theirs you used. If you contributed to others' work, list what you contributed. Example (non-code example, but true to life):