CS 1510: Intro to Computing - Fall 2017
Assignment 9: Tracking the Greats of the NBA

Code Due: Tuesday, November 28, 2017, by 11:59 p.m.

Assignment Overview

The goal of this project is to 1) gain experience using pieces of code written by someone other than yourself, 2) test your new top-down design skills in breaking a large problem down into steps and helper functions, and 3) gain more practice with file I/O, dictionaries, lists, files, and functions.

Background

A common element seen on web pages these days are tag clouds (http://en.wikipedia.org/wiki/Tag_cloud). A tag cloud is a visual representation of frequency of words, where more frequent words are represented in larger font. One can also use colors and placement. We are going to analyze a famous movie and create a tag cloud for each movie character based on the words they used, where the frequency of the words indicates the size of the font in the cloud.

The movie we will analyze is Montey Python and the Holy Grail. While watching the movie prior to starting this assignment is not a requirement, it's highly recommended if you have some extra time!

To help you with this assignment you are provided with the following documents:

- A transcript of the movie: holy-grail.txt
- A list of stop words: stoplist.txt
- Some helper functions: html-functions.py

Each of these files is explained below:

- **Transcript** - Open up the move transcript file . The transcript is in a particular format. Each time one of the move characters speak, that line is marked with the speaker's name followed by a colon (‘ARTHUR:’, ‘BLACK_KNIGHT:’, 'GALAHAD:', and others). Once encountered, all words are attributed to that character until another label occurs. Notice that this may not be for several lines.
- **Stopwords** - Not all words are worth counting. In the context of speeches, ‘a’, ‘the’, ‘was’, etc. are just junk. A list of such words is provided as stopSQL.txt (which is one of many stoplist files available online). Each line has a single word. No word in the stop word list should be counted in the tag cloud. This is the list distributed with MySQL 4.0.20 list with a few additions (mostly just duplication of contractions. That is both “can’t” and “cant” are now in the list)
- **Functions** - Three functions and an example are provided in htmlFunctions.py. Use them in your program. That file contains:
    - **makeHTMLword (word, cnt, high, low)**
      This function takes a word and wraps it in a font tag with a specific size. The function takes a word, how many times it occurred in the document, the highest word count and the lowest word count of words being processed (the highest count we are considering for this tag and the lowest). It returns a string that is the word wrapped in a tag that contains a style attribute that sets the font size between htmlBig and htmlLittle (two local vars in the function. You can change them to be whatever you like)

    - **makeHTMLbox(body)**
      This function takes a single string of all the font-wrapped words from makeHTMLword and places them in an html box to be displayed. It returns a string which is the html code for the box.

    - **printHTMLfile(body,title)**
      Takes the body returned from makeHTMLbox and wraps a standard html web page around it. The string title is used in the html. The title is also the file name with an ‘.html’ suffix. Play with this file for a few minutes until you see how it works. You do not need to understand all of the details, but you need to understand what each function does and how they work together.

Project Specifications

In a file called hw9.py, you will need to write a group of helper functions called by a master function (named **main()**) which will:

1. read through the transcript file,
2. create a dictionary of words spoken by a specific character,
3. remove stop words,
4. identify the 40 most frequently used words by that character,
5. and use that information to call the helper functions provided to in the file htmlFunctions.py to create the word cloud.

Your code should include **no less than 3 new helper functions (other than main())** to illustrate that you understand how to break large problems into small, managable steps. Of course, you can create more than 3 new helper functions if you like.

The master function main() will take in two parameters:

1. A string representing the name of the movie text file
2. A string representing the character that you wish to analyze. This string should be the exact name/label used in the debate transcript (without the colon. See below)

For example, if I invoke:

```
main("holy-grail.txt","ARTHUR")
```

my code should produce a file called ARTHUR.html which is the word cloud spoken by that character in the holy grail movie. That file should look like this one (ARTHUR.html)

CHECKING YOUR WORK. If you run the commands above on your finished code and the html files look different from mine, then something isn't quite right for one of us. It COULD be me who is wrong, but you should check your code first.

Helpful Hints

1. Create your design diagrams first. Break down all the steps that need to be performed, and figure out which functions they go into.
2. Parsing the movie file. You have to read in the file and separate the lines according to who said them. Use the file format to help you with this. Remember, once you see one of the character "tags" (like "GUARD_#1:"), all lines/words belong to that character until you see another character tag.
3. You have to remove the stop words. You have two choices. Either do it as you are reading the movie file or go back and remove them once you are done reading the entire movie file. Each has advantages. Pick one and go with it.
4. Also remember to remove punctuation from words: just because a word comes at the end of a sentence and has a period at the end of it doesn't make it a different word.
5. Capital letters are a potential problem. If we aren't careful, the word "Economic" at the start of a sentence will be counted separately from the word "economic" inside of a sentence. For simplicity, treat all words as lower case letters (but be careful when you decide to convert to lowercase letters since character speakers are labeled with all caps ("ARTHUR:") and you may want/need to use this information.
6. Count the word frequency in the character's words. Use a dictionary, where the key is the word and the value is the count.
7. Once you have dictionary for the character in question, you need to extract the 40 most frequently used non-stopwords and their counts. This should be familiar. You will need to use lists of lists as we have done before.
8. We also need to extract the biggest count and the smallest count from the words in this top-40 as that information is needed by makeHTMLword()
9. Once you have the forty most frequent words (and their counts) we want to alphabetize that list. That sounds like another set of lists in a list.
10. Finally, use the code presented to you in htmlFunctions.py to generate the html file with the appropriate name.

Final Submission

Submit the modified file **hw9.py** to eLearning.