American University
CSC 280: Fall 2019
PYTHON PROJECT: Text Analyser
Instructor: Donna Dietz

In this project, you will write some functions to demonstrate your mastery of a variety of topics covered in class. You will probably need to go back to some of the notes from class. Remember to only use techniques shown in class, unless you see me ahead of time for exemptions. Several text files are included in this repo, all of which are available from Project Gutenburg. You may add one or two extra files to your repo as long as they are not too large and they are not copyrighted (except by you perhaps).

This code (when it's finished) can be run through the command line and will prompt the user to choose one of the files in the same directory as the code. The user should only choose text files, or the code may crash. The code will then read the file one line at a time and count the letter frequencies. Capital and lowercase letters will be treated as the same letter, and all other characters will be discarded. Please understand that this will make counts for non-English text inaccurate, as accented characters will be discarded by this process. A frequency report will be presented to the user, followed by a *distance* measure from English. The user will then be prompted to continue or not. My code easily fits on two sheets of standard paper and is not very efficient, so the description of this project is actually longer than the code it describes.

Before you start work on this project, take some time to play with the **runme.py** and associated files which are included with this project.

For this project, the following functions are required:

def readfile(fyle): This function takes in a string which gives the name of the local file to be read and analysed. A list is returned with each line of the file as one list element. However, if you wish to remove the "\n" from the end of each line, you may do so.

def data2dict(L): This function takes in the list created by *readfile* and counts all the letters. A dictionary is returned which has letters (either all lowercase or all capital) as the keys and their frequencies (integers) as the values.

def sortDict(d): This function takes in the dictionary created by *data2dict* and creates a list of sorted tuples which reflects the same information as the dictionary but in a different order. Each tuple will contain two items: an integer and a letter. The integer gives the frequency of the letter, and the tuples are sorted highest to lowest by the value of the integer.

def analyseFreq(TupleList): This function takes in a list of tuples which was created by *sortDict* and creates a report for the user. It does not return anything. The report must list the letters in order of highest to lowest frequency, in two

columns, so that the entire first column is read and then the second column. The program must be able to handle missing letters properly, both an even and odd number of missing letters. All columns must line up precisely, and percents must be expressed with 2 digits after the decimal. Unused letters must be presented in alphabetical order after the list is produced, or if no letters are missing, a different message must appear expressing this fact. (Note: This is about halfway through the coding, but reaching this point successfully will be worth 75% of the credit for the project.)

def distanceEnglish(d): This is a distance measurement created just for this project. This function takes in the dictionary produced by *data*2*dict* and returns a floating point value. In order to calculate the distance, first visit **https://en.wikipedia.org/wiki/Letter_frequency** and grab all the letter frequencies from the chart at the top of this page. For example, the letter "A" accounts for 8.167% of all letters used in English. You can use a list or dictionary or other structure to save these values in your code. For each letter, compare the absolute difference in frequency (as a percent) between the table and your calculations. Multiply that difference by the value in the table expressed as a decimal. Then, add that up for all 26 letters. For example, "E" is supposed to account for 12.702% but let's say that in your selected file, it only accounts for 8%. For the letter "E", you would subtract 8 from 12.702 to get 4.702, then multiply this by 0.12702 to get 0.59724804. Each letter contributes to the total which is then returned.

def pickFile(): This function takes no input, but questions the user one time only about which file is to be analysed. It does however, handle poor responses from the user, forcing the user to choose from the listed files. The function lists the files which are returned by os.listdir() in the order given, starting with 1 (not zero!) and going up to the number of files in the directory. It does not handle the case where the user unwisely chooses a file that's not a text file. (The program will crash in that case, and that's fine!) The function should not "hard code" the number of files in the directory. The program should know how many files are in the directory based on the length of the list returned by os.listdir(). Once a valid file has been selected, the function calls *readfile* and passes that result back to the calling code.

def main(): This function takes no input. This function contains the loop that repeatedly allows the user to continue to analyse files. It greets the user and presumes the user wants to analyse at least one file. A file is selected by *pickFile*, and this result is passed to *data*2*dict* which in turn has its output fed into *sortDict* whose output goes to *analyseFreq*. The distance is then calculated and expressed to the user with two digits after the decimal. Then the user has the choice to continue if desired.

At the end of your "hw7.py" file, you must include the following two lines of code so your program may be run from the command prompt:

```
if __name__=='__main__':
    main()
```