CSc 110, Autumn 2017
Programming Assignment #11: Sentiment Analysis (20 points)
Due: Tuesday, November 21, 2017, 7:00 PM

Thanks to Eric D. Manley and Timothy M. Urness from Drake University and Marty Stepp from Stanford.

This assignment focuses on using dictionaries. Turn in a file named `sentiment.py`. You will need `training.txt` to write this program.

Program Description:

Sentiment Analysis is a Big Data problem which seeks to determine the general attitude of a writer given some text they have written. For instance, we would like to have a program that could look at the text "The film was a breath of fresh air" and realize that it is a positive statement while "It made me want to poke out my eye balls" is negative.

One algorithm that we can use for this is to assign a numeric value to any given word based on how positive or negative that word is and then score the statement based on the values of the words.
But, how do we come up with our word scores in the first place?

That's the problem that we'll solve in this assignment using machine learning. Your program will search through a file containing movie reviews from the *Rotten Tomatoes* website which have both a numeric score as well as text. You'll use this to learn which words are positive and which are negative.

The file displayed below contains two pieces of data on each line: the positivity rating and the sentence it is for.

The ratings have the following meanings:

- 0 : negative
- 1 : somewhat negative
- 2 : neutral
- 3 : somewhat positive
- 4 : positive

```
1 A series of escapades demonstrating the adage that what is good for the goo
4 This quiet , introspective and entertaining independent is worth seeking .
1 Even fans of Ismail Merchant 's work , I suspect , would have a hard time s
3 A positively thrilling combination of ethnography and all the intrigue , be
1 Aggressive self-glorification and a manipulative whitewash .
4 A comedy-drama of nearly epic proportions rooted in a sincere performance t
1 Narratively , Trouble Every Day is a plodding mess .
3 The Importance of Being Earnest , so thick with wit it plays like a reading
1 But it does n't leave you with much .
1 You could hate it for the same reason .
1 There 's little to recommend Snow Dogs , unless one considers cliched dialo
1 Kung Pow is Oedekerk 's realization of his childhood dream to be in a marti
4 The performances are an absolute joy .
3 Fresnadillo has something serious to say about the ways in which extravagar
3 I still like Moonlight Mile , better judgment be damned .
3 A welcome relief from baseball movies that try too hard to be mythic , this
3 a bilingual charmer , just like the woman who inspired it
2 Like a less dizzily gorgeous companion to Mr. Wong 's In the Mood for Love
1 As inept as big-screen remakes of The Avengers and The Wild Wild West .
2 It 's everything you 'd expect -- but nothing more .
```

When your program starts it should prompt the user for a file name of data to learn from. After that, and after it completes each task, it should prompt the user with five options. An example:

```
Learning data file name? training.txt
What would you like to do?
1: Get the score of a word
2: Get the average score of words in a file
3: Find the highest / lowest scoring words in a file
4: Sort the words in a file into positive.txt and negative.txt
5: Exit the program
Enter a number 1 - 5:
```

1. When the program starts

As soon as the user enters in a learning data file name your program should read in all of this file's data. You may assume it is in the format described above. Using this data you should create a dictionary that contains each word mapped to the sum of the scores for each line that the word occurred in divided by the total number of times it occurred. For example, if the file contained only the following four lines:

```
1 it wasn't the best
4 Best thing ever.
4 oh wow it was so great!!!! The BEST!
2 it might be better but I really don't know
```

Than for the word "best" your dictionary should store 3. Note this was calculated by (4 + 4 + 1) / 3 where 4, 4, and 1 were the scores of lines containing the word "best" and 3 was the total number of lines that contained the word "best".

Note that this ignores capitalization and punctuation. You must get rid of all characters that aren't letters or numbers. We suggest that you use the `ord` function to do this. Convert each character to its numeric value using `ord` and then test if the characters value is between that of `a` and `z`.

It will be easiest to keep track of the necessary data if you keep track of data in two dictionaries, one with a count of the times you see each word and one with the sum of the scores for each word. You will want to keep these dictionaries throughout your program so that it can learn from user reactions.

2. Computing the average score

To get the average score for words in a file, first read in that file, normalize the case and strip it of all punctuation, as described above, and split it on whitespace. Then, compute the average of the scores for all words in the file. Output this to the user and output whether it is generally positive or negative. You can consider an average score above 2.01 as positive and one below 1.99 as negative. Anything in between should be considered neutral.

After the program outputs this data it should ask the user if it identified the sentiment correctly. If the user types "`yes`" it should do nothing. If the user types "`no`" it should prompt the user for the correct score and add that, for the words in the sentence, to the dictionaries.

3. Highest / Lowest scores

If the user asks for the highest / lowest scoring words you should do the same thing as above except output the highest and lowest scoring words of all words in the file. You may assume that there is at least one word in the file.

4. Output positive and negative lists

If the user selects the option to output lists of positive and negative words, the program should output all positive words from the dictionary into a file called `positives.txt` and all negatives into a file called `negatives.txt`. It should overwrite whatever was already in those files.

Style Guidelines:

No specific constants are required for this assignment but you should use them anywhere they will help make your code clearer

We will grade your function structure strictly on this assignment. Use at least **seven nontrivial functions** besides `main`. These functions should use parameters and returns, including dictionaries, as appropriate. The functions should be well-structured and avoid redundancy. No one function should do too large a share of the overall task. You may not nest these functions inside each other or in main.

Your `main` function should be a concise summary of the overall program. It is okay for `main` to contain some code but `main` should not perform too large a share of the overall work itself, such as traversing over dictionaries. Also avoid "chaining," when many functions call each other without ever returning to `main`.

We will check strictly for redundancy on this assignment. If you have a very similar piece of code that is repeated several times in your program, eliminate the redundancy such as by creating a function, by using `for` loops over the elements of lists and/or dictionaries, and/or by factoring `if/else` code.

Since dictionaries are a key component of this assignment, part of your grade comes from using lists dictionaries properly. Carefully consider how they should be passed as parameters and/or returned from functions as you are decomposing your program.

You are limited to features in lectures 1 - 31. Follow past style guidelines such as names, variables, line lengths, and comments (at the beginning of your program, on each function, and on complex sections of code). You may not have any global variables (except constants) or code outside of functions.