CS 130R Fall 2020

Project 3: Anagrams

Assignment Overview
This project focuses again on strings as well as introducing lists. It also introduces a little file input/output. It is worth 75 points.

The Problem
An anagram, according to the wikipedia, is:

> a type of word play, the result of rearranging the letters of a word or phrase to produce a new word or phrase, using all the original letters exactly once (http://en.wikipedia.org/wiki/Anagram)

For example, the words canter, nectar, recant and trance are all anagrams of each other. That is, exactly the same letters are in each word but in a different order, and each ordering of the letters is a word in the English language.

You are going to write an interactive program that identifies anagrams in a text file. The program should support a number of features for counting and displaying anagrams, which are described below.

Program Specifications

1. Your program will initially prompt for the name of a text file for identifying anagrams.
2. After the text file has been specified, the user can repeatedly enter commands to query information about certain anagrams that appear in the text file. The following commands should be supported:
    a. `anagrams` *`letterstring`*: Display all unique anagrams of *`letterstring`* that appear in the specified text file (space separated on a single line)
    b. `count` *`letterstring`*: Display the number of occurrences of any anagram of *`letterstring`* that appears in the specified text file
    c. `top anagram`: Display the largest set of unique anagrams that appear in the specified file (space separate the anagrams on a single line of output)
    d. `all anagrams`: Display all anagram sets that contain more than one unique anagram. Display each set on its own line with space-separation between each unique anagram.

Deliverables
anagrams.py -- your source code solution (remember to comment your code).

1. Please be sure to use the specified file name, i.e. “anagrams.py”
2. Electronically submit a copy of the file.

Assignment Notes:

1. For this assignment, you can define a word as a space-separated string of symbols and letters. For example, in the text "Right now, I'm giving a definition. Here it is." the words are "Right", "now,", "I'm", "giving", "a", "definition.", "Here", "it", "is.".
2. To simplify the definition and processing of anagrams, remove all non-alphabetical symbols from each word and convert to lower case when checking anagrams. For example, we consider "Night's" to be an anagram of "things".
3. When displaying anagrams, you should display the symbol-less, lower-case version of words as described above. For example, if you identified "Night's" to be an anagram of "things" in the text file, you should display "nights" and "things".
4. The `count` command is the only command that does not deal with *unique* anagrams. That is, if the text file only contained the following text: "A rat got stuck in tar. Poor rat." then the count of "rat" anagrams would be 3 (two occurrences of "rat" and one of the anagram "tar"). However, the `anagrams` command would ignore duplicate instances of words, and display "rat tar".
5. Make sure your program doesn't ever crash, no matter what input is given by the user.
6. To handle user input that isn't formatted as one of the commands above, print a message informing the user and then re-prompt for a new command.

Assignment Hints:
There are a couple of problems here. Think about each one before you start to program.

1. How can I easily determine if two words are an anagram of each other? That is, what simple process can I apply to the words and, as a result, know if those words are anagrams? Note that the only characteristic that matters is what letters are in each word, irrespective of order. Is there an arrangement of the letters that would make it easier to check if words are anagrams? (Also, see the hint in hint #4 below!)
2. Once I have such a process of anagram detection, how can I apply the process to all the words in a word-list and then find all the words with the same anagram?
3. We haven't done file I/O yet, but it's quite easy.
    a. First, use the `open` function. It takes a single string argument, the name of the file you want to open, and returns a file descriptor. Make sure the file you want to open is in the same directory as the program you are running.
    b. Once you have the file descriptor, you can iterate over it using a `for` statement. Each iteration through the for loop gets another line from the file
    c. After you are done, use the `close` method to close the file.

```
fd = open('wordList.txt')
for ln in fd:
   print(ln)      # prints each line of the file
fd.close()
```

4. It will be useful to convert a string to a list and a list to a string. Here are some idioms for that:

```
myStr = 'abcd'
myList = list(myStr)      # produces list ['a','b','c','d']
myList.append('z')
```

```
newStr = ''.join(myList) # produces string 'abcdz'
```

The string method `join` takes each element from the argument (in this case, `myList`) and joins them together using the calling string. If the calling string is the empty string (which it is in this case), it just makes the string. Calling `join` with a different string, such as ':' puts that character between each element

```
':'.join(myList)     # produces 'a:b:c:d:z'
```

So here is the big clue! Ready? If I take a word (a string) and turn that word into a list, what operation can I apply with that list that I can't to the word as a string, and which would be most helpful for anagram representation?

Getting Started (suggested method)

1. Write a high-level outline of what you want to do. Think before you code.
2. Write functions for the parts you think you can accomplish and that you know will be useful later.
3. Write an interactive program that can respond to each of the 4 required commands, even if they don't do anything interesting yet.
4. Test all your code as you write it. Don't try to test everything at the end—it will likely be a disaster.
5. Start to "link up" your code by calling functions you wrote to implement the required commands. In an ideal world where you already wrote and tested lots of useful functions, this step should only take a couple minutes! In reality, you may need to define additional functions to implement logic you don't already have code for.

Below is a sample output for a correctly implemented program that was run using the same sampleText file you will be using to develop your project.