Computer Science CS134 (Fall 2020)

Daniel Aalberts, Duane Bailey, & Molly Feldman

Laboratory 3

Building a Python Toolbox

Objective. To construct a toolbox of functions for manipulating words.

This week we'll construct a small toolbox or *module* of tools for manipulating words and word lists. When finished, we'll be able to answer some trivia questions.

The NPR Puzzle. Will Shortz is the editor of the New York Times Crossword and the Puzzlemaster at National Public Radio. Each Sunday morning he challenges listeners with a puzzle to solve by the following Thursday. Typically these are language-based challenges, but, as we'll see, their solutions can frequently be computed.

Here are some interesting problems:

p1: (Proposed February 11, 2018.) Name part of the human body in six letters. Add an 'r' and rearrange the result to name a part of the body in seven letters. What is it?

p2: (Proposed August 16, 2020.) Think of a major city in France whose name is an anagram of a major city in Italy. What cities are they?

p3: (Proposed April 2, 2017 by David Edelheit of Oyster Bay, N.Y.) Think of four 4-letter proper names that are all anagrams of each other. Two of them are first names—one male and one female. The other two are well-known geographical names. What names are these?

p4: (Proposed September 23, 2018 by Jim Levering of San Antonio) Think of an affliction in five letters. Shift each letter three spaces later in the alphabet—for example, 'a' would become 'd', 'b' would become 'e', etc. The result will be a prominent name from the Bible. Who is it?

Are you up for solving one or more of these challenges?

Getting Started. Before you begin, clone this week's repository:

```
git clone https://evolene.cs.williams.edu/cs134-labs/22xyz3/lab03.git lab03/
```

where your CS username replaces 22xyz3.

This Week's Tasks. The goal of this week is to build a "toolbox" of utilities for manipulating lists of words. Our hope is to help people who wish to solve puzzles like those we've just seen. As we complete the tasks, we might think about other methods that might be helpful in solving general word problems.

1. Complete a small toolbox of string-related functions, `wordTools.py`:

   ◇ Write a function, `words(filename)`, that returns a `list` of words found one per line (one word may include spaces) in a file whose name is specified by `filename`. Strip off any unnecessary whitespace from the words as you read them in. You might use it this way:

   ```
   >>> len(words('words/firstNames'))
   5166
   >>> words('words/bodyParts')[124]
   'skeleton'
   ```

- ◇ Write a function, `sized(n,wordList)`, that takes a word list `wordList` and a word length, `n`. It returns a list of the words that are exactly length `n`. For example:

```
>>> sized(8,words('words/italianCities'))
['Cagliari', 'Florence', 'Siracusa']
```

- ◇ Write a function, `canon(s)`, that returns a lower-cased, space-free, and sorted rearrangement of the letters of `s`, as a string. For example:

```
>>> canon('Lot A')
'alot'
>>> canon('aim')
'aim'
>>> canon('Mia')
'aim'
```

You may develop other useful functions, as well. If you do, collect them in `wordTools.py`.

2. Make sure your `wordTools` toolkit is a solidly built module:

   - ◇ Complete the triple-quoted docstring at the top of the file. This helps users understand the purpose of this module. You can check all your documentation with:

     ```
     pydoc3 wordTools
     ```

   - ◇ Make sure that every method is documented with a helpful document string.
   - ◇ Thoroughly test each method. You might, for example, import the particular method into interactive Python and make sure it works as you expect.
   - ◇ Include, in each docstring, at least two doctests (>>>) for each method in `wordTools`.
   - ◇ Define the global value `__all__` to be a list of strings of names that should be imported when you write:

     ```
     from wordTools import *
     ```

     We've suggested some, but you should add more as you complete your work.

3. Now, solve one of the puzzles. For example: to solve puzzle 3, write a new Python script, `p3.py` that prints the solution as directly as possible. Where there are multiple choices, the output should be short enough for a reader to identify the correct solution. The word lists found in the `words` folder will be useful. (The `words/README.txt` file describes the contents of these word lists.)

Good luck! Do not forget to add, commit, and push your work as it progresses!

Submitting Your Work. Certify that your work is your own, by signing the Honor Code statement in the `honorcode.txt` file. When you're finished, add, commit, and push all of your work to the server. This will include the `honorcode.txt`, three completed methods in `wordTools.py`, and at least one problem-solving script, `p1.py`, `p2.py`, `p3.py`, or `p4.py`.

Extra credit. One way to get more credit is to solve more than one of the problems described above.

The Spelling Bee puzzle from the New York Times is also the source of interesting word problems. These words are spelled with an alphabet (a "hive") of at most seven letters. Problems **p5** to **p8** are challenges related to Spelling Bees:

p5. An n-letter isogram is a word spelled with n unique letters. The `isIsogram(word)` method in `wordTools` uses a `set` to determine if `word` is an isogram. Rewrite `isIsogram` so that it identifies an isogram using a single `for` loop instead of a `set`.

p6. How many lowercase 7-letter isograms are in the word list `'words/dict'`?

p7. (September 22, 2020.) Suppose you have a seven letter hive, `'mixcent'`. How many 4-letter lowercase words in `'words/dict'` (1) include `'m'` and (2) are spelled only using (possibly repeated) letters from the hive string?

p8. (September 22, 2020.) What are the *longest* lowercase words from `'words/dict'` that (1) includes `'m'` and (2) is spelled using only letters from the hive string `'mixcent'`?

Grading Guidelines. Some of the specific functional requirements we will test for include:

1. Your code for the problems must compute each answer as directly as possible.
2. We're looking for solutions that use only a few loops. For example, `p1` can be solved using 2 loops. If you find yourself writing more than 4 loops, it may be best to review your strategy with a TA.
3. Make sure you implement the methods of `wordTools` carefully. Do not modify method names or interpret parameters differently. Make sure your methods return the results described. This document serves, in some way, as a *contract* between you and your users. Deviating from this contract makes it hard for potential users to adopt your implementation!
4. Name any Python script after the problem it solves. Use names like `p1.py`, `p2.py`, etc. Failure to follow this pattern makes it more difficult for us to give you credit for your work. (Problem 5 is just a modification of `wordTools.py`.)

Functionality and programming style are important, just as both the content and the writing style are important when writing an essay. We expect to see code that makes your logic as clear and easy to follow as possible. *A Python Style Guide* is available on the course website to help you with stylistic decisions.

The file `GradeSheet.txt`—the basis for our grading feedback—documents our expectations.

⋆