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 puzzle master at National Public Radio. Each Sunday morning he challenges listeners with a puzzle to solve by the following Thursday. Typically these are challenges that test one’s vocabulary, but, as we’ll see, we can frequently compute their solutions.

Here are some interesting problems (in no particular order):

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. 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 a disease 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?

p5: (Proposed November 22, 2020 by Alan Hochbaum of Duluth, Georgia) Name a marine animal in two words. Remove two consecutive letters in the name and read the resulting string of letters in order from left to right. You’ll name a major American city. What is it?

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

Getting Started. Before you begin, clone this week’s repository in the usual manner:

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

where your CS username replaces 22xyz3.
This Week's Tasks. The goal of this week is to build a toolbox or module of utilities, called wordTools for manipulating strings and 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. Review the wordTools.py script:
   - To leverage our previous work, we have incorporated several methods from our discussion sections. Some of these may be helpful to you in writing more powerful methods, while others may be generally helpful in answering puzzle questions. As you investigate these functions, think about how they might be used to solve more general problems.
   - Notice that when you run wordTools.py as a script, it calls the testmod() function from the doctest module. This method performs all of the interactive examples found in docstrings—called doctests—and verifies they produce the correct results.
   - Currently, one doctest associated with the canon function fails. What is the canon function supposed to do? Fix the doctest so that the script, when executed, runs quietly. Throughout this semester you will be required to use this testing process to demonstrate the methods you write run correctly.

2. Now, extend the toolbox:
   - Complete the function, isIsogram(string), that returns True when all the letters in the string are unique, and False otherwise. The strings 'Duane' and 'Lida' are isograms, but 'MoLlY' is not. Incorporate two doctests into the docstring associated with isIsogram that test other interesting strings.
     You may find that some of the code we have provided will help you avoid a complex solution written from scratch. Embrace these opportunities.
   - Complete a function, words(filename), that returns a list of words found one per line in a file whose name is specified by filename. A “word”, like 'New York', may include spaces internally, but not at its ends. Strip off any whitespace from the ends of lines as you read them in. You might use it this way:
     
     ```python
     >>> len(words('words/firstNames'))
     5166
     >>> words('words/bodyParts')[14]
     'belly button'
     ```
     
     Please incorporate two or more doctests (not the above) into the docstring associated with the words function.
   - 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:
     
     ```python
     >>> sized(8,words('words/italianCities'))
     ['Cagliari', 'Florence', 'Siracusa']
     ```
     
     Use two new doctests to help verify that your sized method works.
     
     You may develop other useful functions, as well. If you do, collect them in wordTools.py.
3. Review your wordTools toolkit, ensuring it 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:
  ```python
  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.
- As you add methods of your own design, make sure they're documented and tested, and think about whether someone might import them into their own scripts to solve problems.
- Define the global value `__all__` to be a list of strings of names that should be imported when you write:
  ```python
  from wordTools import *
  ```

4. 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`, `p4.py`, or `p5.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 b1 to b3 are challenges related to Spelling Bees:

b1. How many lowercase 7-letter isograms are in the word list 'words/dict'?

b2. (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?

b3. (September 22, 2020.) What are the longest lowercase words (there are 3!) from 'words/dict' that (1) include 'm' and (2) are spelled using only letters from the hive string 'mixcent'?
Grading Guidelines. We will also grade the original code you use to solve the various puzzles. 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. When possible, you should seek to make use of the tools imported from your wordTools module.

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, or b1.py. Failure to follow this pattern makes it more difficult for us to give you credit for your work.

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[^1] 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.