Project

Choose *only one* of the two following options:

Assignment Option 1: Your Own Design

- You are free to design and implement a Python program of your own design. If you choose this option, your project must meet the following requirements:
- The program must consist of at least two (2) .py files.
- The program must consist of at least one (1) class.
- The program must do some kind of file I/O.
- The program must make use of at least one (1) other Python module (e.g. os, csv, re, some kind of GUI, …)

Assignment Option 2: Scrabbler

If you'd prefer not to design your own project, you can choose to implement *Scrabbler* – a word game helper program.

Your *Scrabbler* should read its dictionary from the words.txt file used in Lab 4. It must then provide its users with at **least five (5)** of the following features:

- List all the anagrams for a set of tiles entered by the user.
- List all words that start with a "Q" but are not followed by a "u".
- Display all 2-letter words.
- List the 3-letter words containing a given input tile.
- Word verifier: Given an input word, verify that it exists within the Scrabble dictionary (words.txt).
- Find all words that end with one or more tiles entered by the user.
- Find all words that begin with one or more input tiles.
- Find all words containing the letters "X" or "Z" and an input tile.
- Display point values alongside all result words.
- Display a simple table showing Scrabble point values and frequencies for all 26 letters of the English alphabet plus blanks.
- ***[This one counts as 3 features.]*** Find all ways to play a given set of seven or fewer input tiles (the player's rack) that extend or cross an existing word on the board (disregarding the positioning of the word on the board – i.e. don't worry about running out of space, and don't worry about double- or triple-letter/word scores). List the results by category (extends the word, crosses the first letter of the word, crosses the second letter, and so on). Sort each play by its total point value within each category, and display the point value next to each play.

The point values and frequencies of the letters in Scrabble can be found at http://en.wikipedia.org/wiki/Scrabble_letter_distributions

Here are some recommendations for implementing Scrabbler:

Create a class to manage the Scrabble dictionary. Its __init__() method should read the words.txt file into a suitable data structure maintained as an attribute of the class. Its other methods should provide various ways of querying the dictionary, as outlined above.

For more-complex queries, it might make sense to encapsulate either the input or output data within another class. For example, the input class could contain the player's rack as a string, and the word to extend or cross as another string. The output class could contain lists of results for each category (extends, crosses first letter, crosses second letter, etc.) Each result could be a tuple containing the played word and its total point value.

Consider storing the words in your dictionary in more than one format, internally to your class, to make your query functions easier to write and/or more efficient at runtime. For example, you could maintain a list of all the "Q" words, a list of all the 2-letter words, and so on.

It's a good idea to separate your data from your user interface. This is known as the model-view paradigm. The data is the "model," and the UI is a "view" of that model. For Scrabbler, you should put your dictionary and query functions in one .py file, and your UI in a second .py file. The main entry point of the program could even be in a third .py file. The benefit of this design is that you can re-use your data (your "model") without having to bring the view along with it.

Your user interface can make use of one of the various GUI packages available for Python. Or it could start up an HTTP server and use a web browser as its UI. Be creative!

- **Grading – 100 points total**