Final Project

Goals

- Complete a project that demonstrates problem solving skills and logical thinking.
- Demonstrate knowledge of Python programming concepts learned throughout the semester.
- Choose one of the two options below.

Option 1: Design Your Own

Description

- You are free to design and implement a Python program of your own design.
- If you choose this option, you must discuss your idea with the professor and **before** starting your project.

Requirements

- The program must consist of at least two (2) classes (each in its own file).
- The program must consist of at least three (3) .py files.
- The program must do some kind of file reading / writing.
- The program must make use of at least one (1) other Python module such as csv, tkinter, os, re, or PyGame.
- The program must be well-designed as we have made in class. Use a main function, have no global variables, use loops / conditionals, use functions and classes to simplify program and reuse code, use clear variable names, and have comments.

Option 2: Scrabble Helper (Scrabbler)

Description

- If you'd prefer not to design your own project, you can choose to implement a text-based Scrabbler – a word game helper program. The program must provide for taking user input (which options they want to execute, entering tiles, etc.). Your Scrabbler should read its dictionary from the **words.txt** file used in a previous assignment.
- You are implementing a word helper such as could be used to help play the game Scrabble. You are not implementing the game Scrabble, but if you are unfamiliar with Scrabble, here is additional information: https://en.wikipedia.org/wiki/Scrabble

Requirements for Scrabbler

- The program should begin with a menu of options and should continue looping until the user exits.
- The program must consist of at least two (2) .py files.
- The program must consist of at least one (1) class (in its own file).
- The program must complete at least eight (8) features from the list below.
- The program must be well-designed as we have made in class. Use a main function, have no global variables, use functions and classes to simplify program and reuse code, use clear variable names, and have comments.
- The program must perform error-checking (e.g. if the user chooses an invalid option from a menu, the program must display a message and ask them to enter again).
- Even though the user interface is text-based, make it look nice and readable. Add tab and blank lines when needed.

Features for Scrabbler

IMPORTANT NOTE: All these outputs for these features must be determined when your program runs. You may not simply save the answers in a string and print them.

- Ask user for set of tiles and then show them the word that they can make that is worth the most points.
- Ask user for set of tiles and then show all words that can be made with any of the letters.
- Ask user for set of tiles and then show the anagrams (i.e. words made using all the letters).
- List all words that start with a "Q" but are not followed by a "u".
- Display all 2-letter words.
- Ask user for a letter and then show all the 3-letter words containing that given input tile.
- Word verifier: Ask user for input and then verify that it exists within the Scrabble dictionary.
- Ask user to enter one or more letters and then show all words that end with that group of letters.
- Ask user to enter one or more letters and then show all words that begin with that group of letters.
- Ask user to enter a letter and then show all words containing the letters "X" or "Z" and the input tile.
- Whenever you display possible words to the user, also display the point values.
- Display a simple table showing Scrabble point values and frequencies for all 26 letters of the English alphabet plus blanks (*the table may be save ahead of time in a string and simply displayed).

- **[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.
- **[For a challenge and extra credit]** Implement a GUI-based Scrabbler Helper. Program must provide the same functionality as above, but provide an interactive, visual interface with buttons, text entries, etc.

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 list 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.

For extra credit, you can ALSO design a GUI for your Scrabbler:

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!

Deliverables and Submission Instructions

- Create a folder on your computer called
  `ITP115_project_lastname_firstname`
  (replace # with this lab number)
- Inside the folder, include your python source code
- Compress the folder (make a zip file) called
  `ITP115_project_lastname_firstname.zip`
  (replace # with this assignment number)
- Upload zip file to Blackboard site for our course

Grading (100 points)

- **10 points** Read Me (write-up)
  - Inside your project file, you must include a file called "readme.txt"
  - This will describe how to run your program (e.g. any required libraries with links, version of python, etc.)
  - You should also give detailed explanation of HOW your program works (e.g. what classes did you use, how are they accessed)
  - If you used any additional online resources or libraries (e.g. PyGame), you must clearly explain which resources you used and which code is your own work

- **15 points** Comments and Style
  - Code should be formatted as we discussed in class
  - Variables, classes, and functions should be appropriately named
  - Code should be broken up into functions where appropriate
  - All functions should have comments describing input, output, and description
  - All classes should have descriptions of instance variables and methods

- **75 points** Project runs and fulfills requirements
  - Projects that do not run will receive an automatic 50% penalty
  - Submit your project on Blackboard

** Points will be deducted for poor code style, or improper submission.*