Programming assignment 6: Text-based Wordle

Handing in the assignment: Students hand in a single archive (**ZIP**/**RAR**/**7Z**) containing all their code (*.py*) and **storage files** on *Canvas*. Make sure all files are placed so that the program runs correctly when your main python file is run directly, and that the python file that should be run is named descriptively.
This assignment is **not** tested with automatic tests but run by humans and is mainly judged on working functionality. Points may nonetheless be deducted for:

- redundant or unnecessarily repeated code,
- messy code,
- code that is difficult to understand and not explained in comments.

Wordle
Wordle is a game of words where one player knows a 5 letter word while the other player (or group of players) guesses what it is. They may guess whole 5 letter words. Each time a guess is wrong the guessing player gets a hint. The hint includes which letters were correct and in the correct location and which letters were correct but misplaced. The game is won if the correct word is guessed. The game is lost after 5 wrong guesses.

Your assignment is to program a text based version of Wordle. The computer will always play as the game master, who knows the correct word, and the user will always play as the guesser. **You can use any built-in data structure in python** and structures made by yourself, co-students or teachers in previous assignments.

The program should only accept a five letter guess, otherwise letting the user know and asking for a new guess. The user should be allowed uppercase and lowercase letters and the program should regard them as equivalent (not case sensitive). After a correctly formed guess, the program outputs a line for each guess in that round, one line after the first guess, but five lines after the last guess, repeating the previous lines so the user has an overview over their round. Each line has the guess, then a bit of empty space, then a five letter code for that guess. Each letter in the code is an uppercase 'C' for a correct letter in the correct place, lower case 'c' for a correct letter in the wrong place and a dash, '-' for a wrong letter.

The program should have access to a text file containing a ***word bank*** to choose words from. It should be possible to add words to the file and remove them between running the program. It is recommended that you read this word bank into a data structure when the program is started to have quick access to it throughout the run. Each time a new game is started a word should be randomly selected from the word bank.

Once a game is either won or lost the program should let the user know and offer them the possibility of either quitting or playing another game.

See the next page for a breakdown of features and grading.

You can implement the following parts in any order you see fit but it is recommended to finish each group before proceeding to the next. Each part is graded on both **functionality** and **program structure**, but mainly on *functionality*.

1. **BASIC GAME - 50%**
    - Display previous guesses in the round - **10%**
    - Display code with each guess, -c-C- - **10%**
    - Lets user know if wrong format and doesn't crash - **5%**
    - Handles uppercase and lowercase without complaint - **5%**
    - Detect loss when guesses are finished - **10%**
    - Detect victory when a guess is correct - **10%**

2. **MORE REFINED SINGLE GAME - 30%**
    - User can input or select number of letters and guesses before the game begins - **5%**
        - Extends the "5 letters, 5 guesses" requirement
    - After finishing a game the user can select to quit or start a new game - **5%**
    - Program stores word bank in a data structure - **5%**
    - Program randomly selects word from word bank - **5%**
    - The word bank is stored in and read from a file - **10%**

3. **CONNECTED SERIES OF GAMES - 30%** (*that makes* **110%**)
    - Keep track of wins and losses throughout the run (store in classes/variables) - **5%**
    - Find a way to score series of games and keep track of high scores - **5%**
        - Total scores can for example be affected by (*but not limited by*):
            - # of wrong guesses per word
            - length of word
            - # of games before loss (*or total score of those games*), etc.
    - Scores/highscores stored so that they live between runs of the program - **5%**
    - Allow words to be added to the word bank (and file) through the program itself - **5%**
    - Allow user to see their history of games/scores - **5%**
    - Allow save/profile for multiple users - **5%**
        - Students figure out how best to accomplish this
        - Don't need password login, but switching/selecting users