Assignment 2

Assignment 2 - Class, data structure and encapsulation

Due Date: February 19th 2022 at 23:55

Percentage overall grade: 7%
Penalties: No late assignments allowed
Maximum Marks: 100

Goal: refresher of Python and hands-on experience with class building and encapsulation. Understand how classes can be used to represent complex objects, and use methods to interact with the class state. Get familiar with reusing code. realize the importance of user input validation.

Assignment Specifications

Problem : Wordle175.

You have probably heard of the **Wordle Game**, a word guessing game that is becoming a craze on the internet these days. It is not the Wordle as the famous application to draw a word cloud from text. But it is a new popular online word game that hundreds of thousands of people are playing daily online. See here an article about it.

Somebody knowing that you have studied how to program in python dared you to implement this on your computer. The popular game is actually simple and you take on this challenge. You call the game **Wordle175**.

Tasks to do

Task 1

One initial challenge you have is to make sure people playing your game would enter valid English words but you certainly do not know all the words in the English dictionary. So, to be able to verify that a word entered by a player is a correct English word, you obtain from a friend a text file with 5 letter words acceptable in Scrabble. The file, word5Dict.txt is herein attached. Unfortunately, when you checked the file you found it is corrupt. Instead of having one word per line as you expected, the file has words separated by "#" a few per line. You need to fix it and write a program that reads this corrupt file and generates a new file with one word per line. This new file will be your dictionary to verify input.

Therefore, your first task is to write a program that reads word5Dict.txt and generates a new file called scrabble5.txt in which each line is a word from the input file.

Task 2

Your task will be to write a class: **ScrabbleDict,** which must implement the following methods:

- __init__(self, size, filename): initializes a dictionary using the content of the file specified by *filename*. Eventually the file will be the file that you generated in Task1. The keys in the dictionary would be the first element in a line of the file so that its length is exactly the specified *size*. Any line in the file which has a first element with a length different then *size* would be ignored. The value in the dictionary would be the full line as read from the file. This would allow the dictionary to have a definition as a value in case the file has a word followed by a definition per line. Of course, your current file from Task 1 doesn't have definitions for word. But you anticipate another version later with definitions.

- check(self, word): returns True if *word* is in the dictionary and False otherwise.

- getSize(self): returns the number of words in the dictionary.

- getWords(self, letter): returns a sorted list of words in the dictionary starting with the character *letter*.

- getWordSize(self): returns the length of the words stored in the dictionary. This is the value *size* provided when building the dictionary.

Do not forget to test your class in isolation before you write the code of the game itself. Test your class methods, one by one. Include these tests (along with comments, as appropriate) under `if __name__ == "__main__":` at the bottom of your **python** file. **You will lose marks if you do not include these tests for us to see.**

Task 3

Your task to design and implement the Wordle175 game while using the ScrabbleDict class.

The rules are as follows: You get 6 tries to guess a selected and hidden word of 5 letters. Let's call it a target word. So you have to guess the target word in 6 attempts or less. A guess consists of typing a word of 5 letters. After each attempt, you get a feedback consisting on three lists of letters:

RED: a list of letters in the guessed word that are not in the target word; ORANGE: a list of letters in the guessed word that are in the target word but in the wrong position; and GREEN: a list of letters in the guessed word that appear in the right position in the target word.

Letters can appear more than ones in the target word, and therefore you can use letters more than ones in your guesses. If your guess has for instance two As, they would be indicated as A1 and A2.

For example assuming the hidden target word is TIMER and your first guess is TREES, the feedback would be GREEN={T, E2}; ORANGE={R}; RED={E1,S}. This means the T and the second E are in the right place, R is in the wrong place, and S as well as the first E are wrong. If in the second guess you type TIGER, GREEN will be {T,I,E,R} and RED will be {G}.

You need to do input validation for any words that are entered as attempts.

- A word cannot be longer or shorter than 5 letters. It has to be exactly 5 letters long.
- A word has to exist as a word in English. The *ScrabbleDict* instance you initialed in Task 2 will be you English dictionary.
- A word cannot be a word that has already been entered as an attempt for the same target word.

Here is an example of a game round where the word was guessed:

```
Attempt 1: Please enter a 5 five-letter word: computer
COMPUTER is too long
Attempt 1: Please enter a 5 five-letter word: five
FIVE is too short
Attempt 1: Please enter a 5 five-letter word: txtic
TXTIC is not a recognized word
Attempt 1: Please enter a 5 five-letter word: 23783
23783 is not a recognized word
Attempt 1: Please enter a 5 five-letter word: trial
TRIAL Green={T} – Orange={I, R} – Red={A, L}
```

```
Attempt 2: Please enter a 5 five-letter word: Tried
TRIAL Green={T} - Orange={I, R} - Red={A, L}
TRIED Green={E, T} - Orange={I, R} - Red={D}
Attempt 3: Please enter a 5 five-letter word: tiGER
TRIAL Green={T} - Orange={I, R} - Red={A, L}
TRIED Green={E, T} - Orange={I, R} - Red={D}
TIGER Green={E, I, R, T} - Orange={} - Red={G}
Attempt 4: Please enter a 5 five-letter word: timer
TRIAL Green={T} - Orange={I, R} - Red={A, L}
TRIED Green={E, T} - Orange={I, R} - Red={D}
TIGER Green={E, I, R, T} - Orange={} - Red={G}
TIMER Green={E, I, M, R, T} - Orange={} - Red={}
Found in 4 attempts. Well done. The Word is TIMER
```

Here is an example of a game round where the word was not guessed:

```
Attempt 1: Please enter a 5 five-letter word: trial
TRIAL Green={T} - Orange={I, R} - Red={A, L}
Attempt 2: Please enter a 5 five-letter word: Tried
TRIAL Green={T} - Orange={I, R} - Red={A, L}
TRIED Green={E, T} - Orange={I, R} - Red={D}
Attempt 3: Please enter a 5 five-letter word: trees
TRIAL Green={T} - Orange={I, R} - Red={A, L}
TRIED Green={E, T} - Orange={I, R} - Red={D}
TREES Green={E2, T} - Orange={R} - Red={E1, S}
Attempt 4: Please enter a 5 five-letter word: tomatoes
TOMATOES is too long
Attempt 4: Please enter a 5 five-letter word: trees
TREES was already entered
Attempt 4: Please enter a 5 five-letter word: tuber
TRIAL Green={T} - Orange={I, R} - Red={A, L}
TRIED Green={E, T} - Orange={I, R} - Red={D}
TREES Green={E2, T} - Orange={R} - Red={E1, S}
TUBER Green={E, R, T} - Orange={} - Red={B, U}
Attempt 5: Please enter a 5 five-letter word: tyler
TRIAL Green={T} - Orange={I, R} - Red={A, L}
TRIED Green={E, T} - Orange={I, R} - Red={D}
TREES Green={E2, T} - Orange={R} - Red={E1, S}
TUBER Green={E, R, T} - Orange={} - Red={B, U}
TYLER Green={E, R, T} - Orange={} - Red={L, Y}
Attempt 6: Please enter a 5 five-letter word: tiger
TRIAL; Green={T} - Orange={I, R} - Red={A, L}
TRIED Green={E, T} - Orange={I, R} - Red={D}
TREES Green={E2, T} - Orange={R} - Red={E1, S}
TUBER Green={E, R, T} - Orange={} - Red={B, U}
TYLER Green={E, R, T} - Orange={} - Red={L, Y}
TIGER Green={E, I, R, T} - Orange={} - Red={G}
Sorry you lose. The Word is TIMER
```

Notice that when the feedback is given, the word attempted is converted to upper case and all the previous feedbacks are also given in sequence as a reminder.

Notice also that letters in the sets are ordered alphabetically.

Task 4

In Tasks 1 and 2 you cleaned a text file with words in English and then a class that can handle it in memory. Having access to this file with English words of size 5, and a python class that reads this file in memory. you decide to write a program that can provide you with hints when you know some well positioned letters while trying to guess the target word in Wordle175. This program uses an extension of your previous class so that you can get a template of a five-letter word and accessing the dictionary will provide a list of all valid words that are compatible with the template. The template is simply a list of 5 letters with possible wildcards (*). For example T**ER is a template stating that the word has to start with T and end with ER. The program would return:.

```
taber
taker
taler
tamer
taper
taser
tater
taver
tawer
taxer
theer
Tiber
tiger
tiler
timer
titer
tiver
tober
toker
toner
toper
toter
tower
toyer
trier
truer
tryer
tuber
tuner
tuyer
tweer
twier
twoer
twyer
tyger
tyler
```

To do so you add the following method in the class:

- getMaskedWords(self, template): returns the list of words that follow the *template* provided. Remember the template cannot have a size different then the size of words in the dictionary

You improve the program to weed out unnecessary candidates by providing additionally in the input the possible letters for the wildcards in the template. For instance, in addition to the template T**ER you also provide the letter I. This means that I has to be in the 2nd or 3rd position. Notice that the number of candidate letters should not be more than the number of wildcards (*) in the template. Your new program would return:

```
tiber
tiger
tiler
timer
```

```
titer
tiver
trier
twier
```

To do so you add the following method in the class:
- getConstrainedWords(self, template,letters): returns the list of words that follow the *template* provided and where the wildcards can be replaced by a letter from the *letters*. *letters* is a list of characters. Remember the *template* cannot have a size different then the size of words in the dictionary and *letters* cannot have more elements than the number of wildcards in the *template*.

Write a program in a file called *hint.py* that reads a template from the user and validates it. When valid, it asks to enter additional letters. These are optional, meaning that the user your enter nothing. When this input is entered and validated the program provides the words from the dictionary that abide by this template and the letters entered. The program should use your extended class.

Task 5

Having access to the list of words of size 5, you decide to do some statistics counting the frequency of each letter of the alphabet in these words. This information could help you make guesses while playing the game. To do so, write a small program that traverses the list of words you collected and counts the appearance of each letter. A python dictionary would be handy. The program should display the letters in alphabetical order (one letter per line), and for each letter display the number of appearance as well as the percentage this letter represents. The percentage is the number of appearances of the given letter divided by the total number of appearances of all letters. This percentage is displayed with 2 digits after the decimal point. To display the data in the form of a histogram, print on the same line for each letter, a star for each percentage after rounding to the closes integer. The output could look like this:

```
A: 3684  8.28%  ********
B:  668  1.23%  *
C: 1246  2.81%  ***
...
```

Notice how the numbers are aligned. The numbers above are not correct numbers from the statistics of the given file. They are only to illustrate the required format.

General Guidelines

In addition to making sure that your code runs properly, we will also check that you follow good programming practices. For example, divide the problem into smaller sub-problems, and write functions to solve those sub-problems so that each function has a single purpose; use the most appropriate data structures for your algorithm; use concise but descriptive variable names; define constants instead of hardcoding literal values throughout your code; include meaningful comments to document your code, as well as docstrings for all functions; and be sure to acknowledge any collaborators/references in a header comment at the top of your Python file(s).

Restrictions for this assignment: Do **NOT** change the method signatures for each class - implement your classes exactly as described in the ADTs. Do not use break/continue; do not import anything other than the described class.

Assignment Deliverables

- You are to submit the implemented **Wordle175.py** class file with the necessary classes, a main.py that plays the Wordle game and **hint.py** which creates instances of the ScrabbleDict class and can provide hits based on templates.

Submission Instructions

Please follow these instructions to correctly submit your solution.

- Name your files as indicated above
- Submit your program at the end of this page
- Note that late submissions ***will not be accepted.*** You are able to submit your work as often as you like and only the last submission will be marked. So submit early and submit often.

word5Dict.txt ✚ 16 January 2022, 5:45 PM

Submission status

Attempt number	This is attempt 1.
Submission status	No attempt
Grading status	Not graded
Due date	Saturday, 19 February 2022, 11:55 PM
Time remaining	19 days 3 hours
Last modified	-
Submission comments	▸ Comments (0)

Add submission

You have not made a submission yet.