Assignment 2 - Wordle

Due Date: Friday, Nov 17, 2023 at 23:55 (Edmonton time)
Percentage overall grade: 6.66%
Penalties: No late assignments allowed
Maximum marks: 9 (A perfect score is 9.)

Goals

- 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.
- Get familiar with using custom errors, raise, try, and catch.
- Realize the importance of user input validation.

Task 1 - Word List Class

Your task will be to write a class: **WordleWords**, which **must inherit from the abstract base class MutableSet.**
This will allow you to use instances of WordleWords like Python sets. The Python set ADT is described here You will not need to implement every method in the ADT. Python will automatically fill in most of them for you if you implement the methods **`__contains__, __iter__, __len__, add, and discard`**.
WordleWords should store the list of words in the set which is referred to by its attribute, _words. This makes WordleWords a “proxy”, which is a class A that acts like another class B but also uses class B to do most of the work by having an instance of class B as an attribute and calling class B’s methods from class A’s method. This allows you to put extra functionality and data checking into class A which enhances how class B works.
Most importantly, **`__contains__, __iter__, __len__, add, and discard`** should call and return the same value as those methods of self._words.
You should not store a second copy of the words in any other attributes inside WordleWords, they should only be stored in self._words!

- **`__contains__(self, word)`** - returns True if the word is in the set, returns False otherwise. This should be done by using the in operator on self._words, like the following code (but add the appropriate comments!).

```python
def __contains__(self, word):
    return word in self._words
```

- **`__iter__(self)`** - returns an iterator over all the words in the set. This should be done by using the iter built-in function described here on self._words, like the following code (but add the appropriate comments!).

```python
    def __iter__(self):
        return iter(self._words)
```

- **`__len__(self)`** - return the number of words in the set. This should be done by using the len built-in function described here on self._words, like the following code (but add the appropriate comments!).

```python
    def __len__(self):
        return len(self._words)
```

- **`add(self, word)`** - add the word to the set. Raises an error if the word is too short, or too long, or doesn't contain only capital letters A-Z. This should be done by using the add method of Python's set data structure, like the following code (but add the appropriate checking and comments!)

```python
    def add(self, word):
        ''' some code to check the word '''
        self._words.add(word)
```

- **`discard(self, word)`** - remove a word from the set. This should be done by using the discard method of Python's set data structure, like the following code (but add the appropriate checking and comments!) You should not use discard outside of this method. Use remove everywhere else instead, which will check first if the word is already in the set.

```python
    def discard(self, word):
        self._words.discard(word)
```

You should also add the following methods to WordleWords:

- `__init__(self, letters)` - initializes an empty set of words. letters specifies how long they should all be.
    - Init should create a _words attribute (at least, you can have more), which should be a Python set. For example;

```python
def __init__(self, letters):
    ''' some comment about the method '''
    self._words = set()
    ... more code here ...
```

- `load_file(self, filename)` - Add words to the set using the content of the file specified by the filename. The file consists of a list of words of varying lengths, you can download it below. You should skip all words that don't have exactly letters number of letters when reading the file. You should make sure to convert all the words to capital letters.
- `check_word(self, word)` - Takes a word and makes sure that it consists only of the capital letters A-Z (no accents) and is the correct length.
    - If the word is too short it should **raise a TooShortError**. You will need to create the TooShortError class yourself!
    - If the word is too long it should **raise a TooLongError**. You will need to create the TooLongError class yourself!
    - If the word has anything in it other than the capital letters A-Z, it should **raise a NotLettersError**. You will need to create the NotLettersError class yourself.
    - TooShortError, TooLongError, and NotLEttersError errors should use the base class ValueError.
    - Every piece of code that needs to check these things should use this method, to prevent code duplication!
- `letters(self)` - Returns the number of letters in every word. (Which should be the same for every word!)
- `copy(self)` - Returns a second **WordleWords instance** that contains the same words. It should not re-load the file!
- **Do not store the list of words in any other attribute other than `._words`.**

Task 2 - Guess Class

Create a Guess class that will represent the results of a guess.

- `__init__(self, guess, answer)` - should take two parameters, the guess the player made and the correct answer.
- `guess(self)` - should return the guess that the player made.
- `correct(self)` - should return a string that is the same length as the answer. It should consist of underscores, except for where the player guessed correctly. For example, Guess("NOISE", "BONUS").correct() should return "_O___".
- `misplaced(self)` - should return a sorted string that contains every letter that the player guessed that is also in the answer, but not at the same position. For example, Guess("NOISE", "BONUS").misplaced() should return "NS".
- `wrong(self)` - should return a sorted string that contains every letter that the player guessed that was not in the answer. For example, Guess("NOISE", "BONUS").wrong() should return "EI".
- `is_win(self)` - should return True if the guess was the same as the answer.

Important: in cases where words have more than one of the same letter, the second, third, etc. copies of the letter must work correctly. If you take the letters returned by correct(), misplaced(), and wrong() you should get the exact same letters and the same number of each letter as in guess().
For example, (these examples are included in check1)

Python
```
assert Guess("MOOSE", "BONUS").correct() == "_O___"
assert Guess("MOOSE", "BONUS").misplaced() == "S"
assert Guess("MOOSE", "BONUS").wrong() == "EMO"

assert Guess("MOOSE", "BOONS").correct() == "_OO__"
assert Guess("MOOSE", "BOONS").misplaced() == "S"
assert Guess("MOOSE", "BOONS").wrong() == "EM"

assert Guess("YOYOS", "BOONS").correct() == "_O__S"
assert Guess("YOYOS", "BOONS").misplaced() == "O"
assert Guess("YOYOS", "BOONS").wrong() == "YY"

assert Guess("YOYOS", "BONUS").correct() == "_O__S"
assert Guess("YOYOS", "BONUS").misplaced() == ""
assert Guess("YOYOS", "BONUS").wrong() == "OYY"
```

Task 3 - Wordle Class

Create a **Wordle** class that will represent a game of Wordle.

- `__init__(self, words)` - should take one parameter, which is a **WordleWords instance object**. It should choose a random word for the game.
- `guesses(self)` - should return the number of guesses the player has made so far.
- `guess(self, guessed)` - should take a string guessed and return a Guess instance object that represents the results of the guess.

Can I use … in my solution?

You should use the following imports in your solution:

```python
from random import choice
from collections.abc import MutableSet
```

Do not import anything else in your a2.py.
Be sure to comment on any code that comes from things you found online. Remember, if you take large pieces of code from online, or if you use small pieces of code that you wouldn't be able to fully understand and write yourself, it is considered plagiarism.

Do not copy from check1.py, etc., console_wordle.py, auto_wordle.py, into your a2.py.

Running and main()

To mark your code, we will run a different Python file in the same folder. **Your python file must be named a2.py or it will not work!** You can test your own code by using **check1.py** from the download folder or by creating your own test cases. You should also test your code using the **console_wordle.py** and **auto_wordle.py** from the download folder.
The download folder also contains the word list, words.txt which is the official Scrabble words, and is copyright Collins, 2019.
When you run one of the checker scripts, it will check that your code is working correctly. If it stops with an error, your code is not working correctly. If your code is working correctly, it will output ALL OK.
You do not have to include a main(), but if you do, you must call it using the format:

```python
if __name__=="__main__":
    main()
```

Hints

- Extra newline \n characters can be removed by using the rstrip() method of strings.
- Check the discussion forum for more hints!

Submission

- **Submit only your a2.py. Make sure it is named a2.py.**
- Late submissions will not be accepted.

Assessment

It is worth 6.66% of your final grade in the course.
The assignment will be marked out of 9. (A perfect score is 9.)

Rubric

- 1 point: Runs without errors, main is called correctly (or no main)
    - Must not run any code at the global scope (no indentation)
- 1 point: Required classes are present
- 1 point: Required methods are present
- 1 point: WordleWords works correctly
- 1 point: Guess works correctly
- 1 point: Wordle works correctly
- 1 point: Meets software quality requirements
- 1 point: Classes meet the descriptions and ADT requirements
- 1 point: Works correctly with console_wordle.py and auto_wordle.py

Penalties:

- -3 points will be removed if a2.py code uses imports other than the two that are allowed.
- -1 point if TA has to fix anything to get the code working.
- -1 point if it's not called a2.py.

Partial points and penalties can be earned.

REMINDER: Plagiarism will be checked for

Just a reminder that, as with all submitted assessments in this course, we use automated tools to search for plagiarism. In case there is any doubt, you CANNOT post this assignment (in whole or in part) on a website like Chegg, Coursehero, StackOverflow or something similar and ask for someone else to solve this problem (in whole or in part) for you. Similarly, you cannot search for and copy answers that you find already posted on the Internet. You cannot copy someone else's solution, regardless of whether you found that solution online, or if it was provided to you by a person you know. **YOU MUST SUBMIT YOUR OWN WORK**.