Programming Project 08

This assignment is worth 50 points (5% of the course grade) and must be **completed and turned in before 11:59 on Monday, November 11, 2019.**

Assignment Overview

(learning objectives)
This assignment will give you more experience on the use of:

1. Lists
2. Sets
3. Tuples
4. Functions
5. Dictionaries
6. Input/output

The goal of this project is to develop an algorithm to generate all possible approved words from a given string and optional list of strings for a game of Scrabble with corresponding points. This will be determined by two elements: 1. Rack (str - letters in hand) 2. Segments (List[str] - letters or groups on letters you see on the board).

https://scrabblewordfinder.org → this link gives a good idea of what you will be creating

Assignment Background

Scrabble came out in 1938. Each player is given 7 tiles in their rack. The game is about laying those letters on a board creating words (possibly with preexisting words on the board as well). These words can be assembled vertically or horizontally. Each letter in the word you create corresponds to a value. In addition, there are certain places on the board that give double or triple scores for words or letters. The player who accumulates the most points wins the game.

If you would like to know more about the rules, visit here: https://scrabble.hasbro.com/en-us/rules

However, the Scrabble game itself is not pertinent knowledge for this project. We will only focus on how to generate the best possible word scores based on one's rack and the preexisting tiles on a board. Our tool will be an aide that will make suggestions, but it will not be perfect in that some suggestions may not be feasible because we do not incorporate the position of the preexisting tiles (incorporating the positions of preexisting tiles would have made the project too complicated for this point in the semester).

Assume that you have the following scrabble board and that your rack has tiles: t,b,y,r,e,t,t

Consider this portion of a scrabble board that has GULP played:

				G				
				U				
				L				
				P				

One choice would be to use letters ‘b’,’t’,’t’,’e’ and ‘r’ in combination with the existing ‘U’ to create the word “butter” (Note that there is no meaning to lower case and upper case in this example. Case is shown simply to differentiate the existing tiles from the new tiles.)

				G				
			b	U	t	t	e	**r**
				L				
				P				

For simplicity, our project only considers one row of the scrabble board at a time and only one tile placed on the board. We then loop through the other single tiles: G, U, L, P. For simplicity, we do not consider two columns (or more) of tiles to connect to.

An actual scrabble game includes a blank tile that can be used as any letter but is worth no points by itself. For simplicity our project does not have such a tile.

Project Description

Your program will prompt for two strings:

1. The *rack* of up to seven characters such as the `tbyrett` used in the example above.
2. The *placed tiles* such as the `GULP` used in the example above.

You will determine the best combination of characters to form a valid word (a word in the *common_3000.txt* file) with the highest score. You will output the top five words by score (longest words first for ties) and output the top five longest words (highest scored words first for ties).

We provide a file named *common_3000.txt* of the 3000 most commonly used English words for testing (the official Scrabble file has over 37,000 entries, but using that file takes too long when testing on Mimir).

Each tile (letter) has a value for scoring according to this table (commonly used letters have lower values):

- **(1 point) : A, E, I, O, U, L, N, S, T, R**
- **(2 points): D, G**
- **(3 points): B, C, M, P**
- **(4 points): F, H, V, W, Y**
- **(5 points): K**
- **(8 points): J, X**

- **(10 points): Q, Z**

To score a word simply add up the value of each letter. If all seven tiles in a rack are used in the word, a bonus of 50 points is added to the score. For simplicity, we do not consider Scrabble's bonus squares that can double or triple the score of a letter or word.

Data Structure

Create a constant data structure named `SCORE_DICT`. As a constant it appears at the top of your program immediately below any `import` statements, and the name is *capitalized* (to indicate that it is a constant). `SCORE_DICT` is a dictionary whose keys are letters and whose values are the letter scores from the table above. This data structure is used when calculating the score of a word.

Functions

`open_file() → fp (file pointer)`

This function repeatedly prompts for a file name until the file is successfully opened; it returns a file pointer. You likely can copy this from a previous project.

`read_file(fp) → dictionary`

This function reads a file and returns a dictionary. Each line of the file is one word. Ignore words smaller than three characters and words with a hyphen or apostrophe (tiles are only letters and we won't bother with small words). Make every word lower case. Each dictionary entry uses a word as the key and the integer 1 as its value.

Commentary: we build a dictionary with all values as 1, but we will never use the values. Why? Dictionary lookup is fast so we want a dictionary. We will only look for membership of keys, i.e. we will use the dictionary to check if a collection of characters, a potential "word", is a valid English word from the *common_3000.txt* file. The dictionary we return is what appears as `scrabble_words_dict` in functions below.

`calculate_score(rack: str, word: str): → int`

This function calculates and returns the score of a word. If all 7 tiles in the rack are used, there will be a 50-point bonus (no bonus, if the rack has fewer than 7 tiles). Use the `SCORE_DICT` data structure to get the score for a letter. Returns the score of the word.

Hint: When checking if all seven tiles in the rack had been used I used the string `replace` method to remove characters. One problem is that by default it replaces all occurrences, but it takes a third parameter so it removes only one occurrence, e.g.
`word.replace(ch,'',1)`will replace exactly one occurrence of `ch` in `word`.

`generate_combinations(rack: str, placed_tile: str): → set`

Finds all combinations of a string (combined `rack` and `placed_tile`) with length of at least 3 (what is the longest possible string you can generate?) Collect combinations in a set, but do **not** include combinations that do not have the `placed_tile`. For example, consider the example above with `GULP` and `butter`. The placed tile is U so you only want combinations with U in them, i.e. you want "butter" (it has a U) but not "bet" (it does not have a U). However, to further complicate your life, it is possible that `placed_tile` will be the empty string in which case you want to generate all combinations of your `rack` (this situation can happen at the start of a Scrabble game when no tiles have been placed on the board yet).

Use the itertools.combinations method to generate the combinations of a string of a particular length.

`itertools.combinations(str, length)`

For example, to generate the combinations of 'abcd' of length 3, here is a shell session. Notice that `itertools.combinations()` is an iterator so you need to iterate through it using `for`. Also, notice that it generates tuples of characters. For this function we will leave them as tuples of characters (we will combine the characters in the tuple into a word in another function).

```
>>> for x in itertools.combinations('abcd',3):
...     print(x)
...
('a', 'b', 'c')
('a', 'b', 'd')
('a', 'c', 'd')
('b', 'c', 'd')
>>>
```

Also, note that `itertools.combinations` only takes one length as an argument whereas you need to generate combinations of *at least 3* so you need another `for` loop.

Commentary: Collecting combinations in a set has two advantages: (1) redundant combinations are implicitly eliminated and (2) for Mimir testing order will not matter.

`generate_words(combo,scrabble_words_dict): → set`

This function takes a list of characters (`combo`) and finds all permutations of the characters. However, only those permutations that are valid English words are added to a set (the second parameter, `scrabble_words_dict`, is the dictionary of valid English words). That set of English words is returned.

You will use the permutation function provided by python.

`itertools.permutations(List[str])`

As with combinations, the permutations are generated as tuples of characters. You will need to combine the characters into words in order to check if the word is a valid English word. You can use `"".join(list)`. For example, here are the permutations of `('a','e','t')`.

```
>>> for w in itertools.permutations(('a','e','t')):
...     word = ''.join(w)
...     print(word)
...
aet
ate
eat
eta
tae
tea
>>>
```

In this example, you can see three English words: `ate`, `eat`, and `tea`. Only those permutations are to be added to the set of English words that you return.

Commentary: since the combinations that we are finding permuations of came from a set, redundancies have already been eliminated. There are multiple good choices for data structures to collect English words. We use sets here for Mimir testing—sets have no order.

`generate_words_with_scores(rack,placed_tile,scrabble_words_dict):→ dictionary{str:int}`

This function takes in the `rack` (string), the `placed_tile` (one-character string), and the `scrabble_words_dict`. It returns a dictionary whose keys are valid English words and whose values are the word's score.

1. First generate all the combinations of your tiles using **`generate_combinations`**
2. Then take that set of combinations and for each combination find the permutations that are valid English words using **`generate_words.`**
3. Take the resulting set of valid English words, find each word's score (using **`calculate_score`**) and add the word to a dictionary with its score (that is, the key:value pair is word:score).
4. Return that dictionary.

`sort_words(word_dic: Dictionary{str:int}): → tuple(List[tuple(str, int, int), List[tuple(str, int, int))`

The function takes in the dictionary of words and their scores (word:score) and returns two lists. Each is a list of tuples: (word, score, len(word)). The first list sorts the tuples by score, with ties sorted by length and alphabetically. This will be referenced as score sorting. The second list will

be sorted by length, with ties sorted by score and alphabetically. This will be referenced as the length sorting. Both lists should be sorted largest to smallest values for score/length and ascending order alphabetically.

You can use `itemgetter` for this. Within a `sort` (or `sorted`) function include `key=itemgetter(1,2)` to sort the tuples first by index 1 and then by index 2.

Since the `sorted` function and the `sort` method are stable sorts, you can first sort the words alphabetically, and then sort them by score/length. You do the two sorts in that order because you do the primary key last, score/length are the primary key in this case.

`display_words(word_list: List[tuple(str, int, int): specifier: str)` → `void`

Prints a display of the top 5 elements of the list with number first, then word. You may assume that the list is already sorted from highest to lowest (i.e. assume that the programmer called `sort_words` before calling this function). If the list length is less than 5, display the full list. If `specifier == 'score'`, display scores; if `specifier == 'length'`, display lengths. Note that the header of the display includes the specifier. Use the following format string for printing the word,value pairs:

```
print("{:>6d}  -  {:s}".format())
```

main() → void

This function is the starting point of the program. After you display the banner, you need to prompt the user if he wants to try an example, if the answer is 'y', you need to prompt for a word file name and open it, then read the data into a dictionary, and close the file. Then the program will proceed with the game:

1. Prompt for rack
   Check that input is 2 to 7 letters; remember to check that they are letters.
   Re-prompt until rack input is correct.
2. Prompt for placed tiles
   Check that the input is letters.
   Re-prompt until placed tiles are correct.
3. For each tile in the placed tiles, generate words into one dictionary to find all possible words by combining each placed tile with the rack.
   You may want to combine dictionaries; if so, use the dictionary `update()` method.
4. Display the top 5 choices. You should use the two functions `sort_words()` and `display_words()`in this step.
5. Ask to continue. If the user answers 'y', repeat from step 1.

Assignment Deliverable

The deliverable for this assignment is the following file: `proj08.py` – the source code for your Python program

Be sure to use the specified file name and to submit it for grading via the **Mimir system** before the project deadline.

Assignment Notes

1. Imports and Global Variables
    a. *itertools* – necessary for importing permutations as well as combinations
    b. *operator* – uses itemgetter for sorting
    c. *SCORE_DICT* – dictionary that relates letters to corresponding Scrabble values
2. Notice how the function tests include other functions within the function you are testing. Try working from the inside out. Get your _get_count to pass then _all_combinations and so on.
3. This project is by no means an optimized generation of words for Scrabble, but a simpler way to understand how an application may go about finding the words itself.

Sample Output

Test 1

```
Scrabble Tool
Would you like to enter an example (y/n): y
Input word file: common_3000.txt
Input the rack (2-7chars): tbyrett
Input tiles on board (enter for none): gulp

Word choices sorted by Score
 Score  -  Word
    11  -  pretty
    10  -  buyer
     9  -  bury
     9  -  type
     8  -  butter

Word choices sorted by Length
Length  -  Word
     6  -  pretty
     6  -  butter
     5  -  buyer
     4  -  bury
     4  -  type

Do you want to enter another example (y/n): n
Thank you for playing the game
```

Test 2

```
Scrabble Tool
Would you like to enter an example (y/n): y
Input word file: common_3000.txt
Input the rack (2-7chars): tbyrett
Input tiles on board (enter for none):

Word choices sorted by Score
 Score  -  Word
     6  -  try
     6  -  yet
     5  -  bet

Word choices sorted by Length
Length  -  Word
     3  -  try
     3  -  yet
     3  -  bet

Do you want to enter another example (y/n): n
Thank you for playing the game
```

Test 3

```
Scrabble Tool
Would you like to enter an example (y/n): y
Input word file: common_3000.txt
Input the rack (2-7chars): aabsti
Input tiles on board (enter for none): xya

Word choices sorted by Score
 Score  -  Word
    10  -  six
    10  -  tax
     7  -  stay
     6  -  say

Word choices sorted by Length
Length  -  Word
     4  -  stay
     3  -  six
     3  -  tax
     3  -  say

Do you want to enter another example (y/n): n
Thank you for playing the game
```

Test 4

```
Scrabble Tool
Would you like to enter an example (y/n): y
Input word file: common_3000.txt
Input the rack (2-7chars): increas
Input tiles on board (enter for none): e

Word choices sorted by Score
 Score  -  Word
    60  -  increase
     8  -  screen
     7  -  scene
     7  -  since
     6  -  care

Word choices sorted by Length
Length  -  Word
     8  -  increase
     6  -  screen
     5  -  scene
     5  -  since
     5  -  arise

Do you want to enter another example (y/n): y
Input word file: common_3000.txt
Input the rack (2-7chars): uccesss
Input tiles on board (enter for none):

Word choices sorted by Score
 Score  -  Word
    61  -  success
     3  -  sue
     3  -  use

Word choices sorted by Length
Length  -  Word
     7  -  success
     3  -  sue
     3  -  use

Do you want to enter another example (y/n): n
Thank you for playing the game
```

Grading Rubric

```
General Requirements:
  (4 pts) Coding Standard 1-9
(descriptive comments, source code Headers, function headers, etc...)

Function Tests:
  (3 pts) read_file
  (4 pts) calculate_score
```

```
  (5 pts) generate_combinations
  (5 pts) generate_words
  (5 pts) generate_words_with_scores
  (5 pts) sort_words

Program Tests
  (4 pts) Test1
  (5 pts) Test2
  (5 pts) Test3
  (5 pts) Test4
```