proj07: Word Game

This game is a lot like Scrabble or Text Twist. Letters are dealt to players, who then construct one or more words out of their letters. Each valid word receives a score, based on the length of the word and the letters in that word. The rules of the game are as follows:

Dealing

- A player is dealt a hand of n letters chosen at random (assume n=7 for now).
- The player arranges the hand into as many words as they want out of the letters, but using each letter at most once.
- Some letters may remain unused (these won't be scored).

Scoring

- The score for the hand is the sum of the score for the words times the length of the word.
- The score for a word is the sum of the points for letters in the word, multiplied by the length of the word, plus 50 points if all n letters are used on the first go.
- Letters are scored as in Scrabble; A is worth 1, B is worth 3, C is worth 3, D is worth 2, E is worth 1, and so on. We have defined the dictionary SCRABBLE_LETTER_VALUES that maps each lowercase letter to its Scrabble letter value.
- For example, 'weed' would be worth 32 points ((4+1+1+2)*4=32), as long as the hand actually has 1 'w', 2 'e's, and 1 'd'.
- As another example, if n=7 and you get 'waybill' on the first go, it would be worth 155 points ((4+1+4+3+1+1+1)*7=105, +50) for the bonus of using all seven letters).

Problem 1. Word scores

The first step is to implement some code that allows us to calculate the score for a single word.

The function get_word_score should accept a string of lowercase letters as input (a word) and return the integer score for that word, using the game's scoring rules.
Fill in the code for get_word_score.

You may assume that the input word is always either a string of lowercase letters, or the empty string "". You will want to use the SCRABBLE_LETTER_VALUES dictionary defined at the top of the file. You should not change its value.

Do not assume that there are always 7 letters in a hand! The parameter n is the number of letters required for a bonus score (the maximum number of letters in the hand).

Testing: If this function is implemented properly, and you run test_proj07.py, you should see that the test_get_word_score() tests pass.
Also test your implementation of get_word_score, using some reasonable English words.

Problem 2. Dealing with hands
Please read problem 2 entirely before you begin coding the solution to problem 2

Representing hands
A hand is the set of letters held by a player during the game. The player is initially dealt a set of random letters. For example, the player could start out with the following hand:
a, q, l, m, u, i, l

In our program, a hand will be represented as a dictionary: the keys are (lowercase) letters and the values are the number of times the particular letter is repeated in that hand.

For example, the above hand would be represented as:

```
hand = {'a':1, 'q':1, 'l':2, 'm':1, 'u':1, 'i':1}
```

Notice how the repeated letter 'l' is represented.

Notice that with a dictionary representation, the usual way to access a value is hand['a'], where 'a' is the key we want to find. However, this only works if the key is in the dictionary; otherwise, we get a KeyError. To avoid this, we can use the call hand.get('a',0). This is the "safe" way to access a value if we are not sure the key is in the dictionary.

d.get(key,default) returns the value for key if key is in the dictionary d, else default. If default is not given, it returns None, so that this method never raises a KeyError.

Converting words into dictionary representation
One useful function we've defined for you is get_frequency_dict, defined near the top of proj07.py. When given a string of letters as an input, it returns a dictionary where the keys are letters and the values are the number of times that letter is represented in the input string.

For example:
```
>> get_frequency_dict("hello")
{'h': 1, 'e': 1, 'l': 2, 'o': 1}
```
As you can see, this is the same kind of dictionary we use to represent hands.

Displaying a hand
Given a hand represented as a dictionary, we want to display it in a user-friendly way. We have provided the implementation for this in the display_hand function. Make sure you read through this carefully and understand what it does and how it works.

Generating a random hand
The hand a player is dealt is a set of letters chosen at random. We provide you with the implementation of a function that generates this random hand, deal_hand. The function

takes as input a positive integer n, and returns a new object, hand containing n lowercase letters.

Removing letters from a hand (you implement this)
The player starts with a hand, a set of letters. As the player spells out words, letters from this set are used up. For example, the player could start out with the following hand: a, q, l, m, u, i, l

The player could choose to spell the word quail. This would leave the following letters in the player's hand:
l, m

You will now write a function that takes a hand and a word as inputs, uses letters from that hand to spell the word, and returns the remaining letters in the hand.

For example:

```
>> hand = {'a':1, 'q':1, 'l':2, 'm':1, 'u':1, 'i':1}
>> display_hand(hand)
a q l l m u i
>> hand = update_hand(hand, 'quail')
>> hand
{'l': 1, 'm': 1}
>> display_hand(hand)
l m
```

(NOTE: alternatively, in the above example, after the call to update_hand the value of hand could be the dictionary {'a':0, 'q':0, 'l':1, 'm':1, 'u':0, 'i':0}. The exact value depends on your implementation; but the output of display_hand() should be the same in either case.)

Implement the update_hand function. Make sure this function has no side effects; i.e., it cannot mutate the hand passed in.

Testing: Make sure the test_update_hand() tests pass. You may also want to test your implementation of update_hand with some reasonable inputs.

Problem 3. Valid words
At this point, we have written code to generate a random hand and display that hand to the user.

We can also ask the user for a word (Python's raw_input) and score the word (using your get_word_score). However, at this point we have not written any code to verify that a word given by a player obeys the rules of the game.

A valid word is in the word list; and it is composed entirely of letters from the current hand.

Testing: Make sure the test_is_valid_word tests pass. In particular, you may want to test your implementation by calling it multiple times on the same hand – what should the correct behavior be?

Problem 4. Playing a hand
We are now ready to begin writing the code that interacts with the player.

Implement the play_hand function. This function allows the user to play out a single hand.

Testing: Try out your implementation as if you were playing the game.

Note: Do not assume that there will always be 7 letters in a hand! The global variable HAND_SIZE represents this value. Here is some example output of play_hand (your output may differ, depending on what messages you print out):

Case #1

```
Current Hand: a c i h m m z
Enter word, or a "." to indicate that you are finished: him
"him" earned 24 points. Total: 24 points

Current Hand: a c m z
Enter word, or a "." to indicate that you are finished: cam
"cam" earned 21 points. Total: 45 points

Current Hand: z
Enter word, or a "." to indicate that you are finished: .

Total score: 45 points.
```

Case #2

```
Current Hand: a s t t w f o
Enter word, or a "." to indicate that you are finished: tow
"tow" earned 18 points. Total: 18 points

Current Hand: a s t f
Enter word, or a "." to indicate that you are finished: tasf
Invalid word, please try again.

Current Hand: a s t f
Enter word, or a "." to indicate that you are finished: fast
"fast" earned 28 points. Total: 46 points.

Total score: 46 points.
```

Problem 5. Playing a game
A game consists of playing multiple hands. We need to implement one final function to complete our word-game program.

Write the code that implements the play_game function. You should remove the code that is currently uncommented in the play_game body. Read through the specification and make sure you understand what this function accomplishes.

For the game, you should use the HAND_SIZE constant to determine the number of cards in a hand. If you like, you can try out different values for HAND_SIZE with your program.

Extension

You decide to teach your computer to play the game you just built so that you can prove once and for all that computers are inferior to human intellect. Modify the play_hand function from the original program.

Problem 6A: Computer Word Choose

First we must create a function that allows the computer to choose a word.
We have provided the function get_perms(hand, n) (defined in perm.py, but usable by simply calling get_perms(hand, n)).

You are not required to know how get_perms works.
It is your responsibility to create the function comp_choose_word(hand, word_list)

Hint: First try to make a legal player, and then worry about making the computer player better (if you have time).

Problem 6B: Computer's turn to play a hand

Now you need to write a function similar to the play_hand function.
Implement the comp_play_hand function. This function should allow the computer to play the game through completion.

Problem 6C: U & Ur Computer

Now that your computer can choose a word, you need to give the computer the option to play.

Write the code that re-implements the play_game function. You will modify the function to behave as described below in the function's comments.

As before, you should use the HAND_SIZE constant to determine the number of cards in a hand. If you like, you can try out different values for HAND_SIZE with your program.