Problem Set 4:

Introduction: In this problem set, you'll implement *two* versions of a wordgame!
Don't be intimidated by the length of this problem set. There is a lot of reading, but it can be done with a reasonable amount of thinking and coding. It'll be helpful if you start this problem set a few days before it is due!
Let's begin by describing the 6.00 wordgame: This game is a lot like Scrabble or Words With Friends, if you've played those. 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, 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 scores for each word formed.
- 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 word created.
- 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) for the four letters, then multiply by `len('weed')` to get (4+1+1+2)*4 =

32). Be sure to check that the hand actually has 1 'w', 2 'e's, and 1 'd' before scoring the word!

- As another example, if *n*=7 and you make the word 'waybill' on the first try, it would be worth 155 points (the base score for 'waybill' is (4+1+4+3+1+1+1)*7=105, plus an additional 50 point bonus for using all *n* letters).

Sample Output

Here is how the game output will look!

```
Loading word list from file...
83667 words loaded.
#--------------------------
Enter n to deal a new hand, r to replay the last hand, or e to
end game: n
Current Hand: p z u t t t o
Enter word, or a "." to indicate that you are finished: tot
"tot" earned 9 points. Total: 9 points
Current Hand: p z u t
Enter word, or a "." to indicate that you are finished: .
Total score: 9 points.

Enter n to deal a new hand, r to replay the last hand, or e to
end game: r
Current Hand: p z u t t t o
Enter word, or a "." to indicate that you are finished: top
"top" earned 15 points. Total: 15 points
Current Hand: z u t t
Enter word, or a "." to indicate that you are finished: tu
Invalid word, please try again.
Current Hand: z u t t
Enter word, or a "." to indicate that you are finished: .
Total score: 15 points.

Enter n to deal a new hand, r to replay the last hand, or e to
end game: n
Current Hand: a q w f f i p
Enter word, or a "." to indicate that you are finished: paw
"paw" earned 24 points. Total: 24 points
Current Hand: q f f i
```

```
Enter word, or a "." to indicate that you are finished: qi
"qi" earned 22 points. Total: 46 points
Current Hand: f f
Enter word, or a "." to indicate that you are finished: .
Total score: 46 points.

Enter n to deal a new hand, r to replay the last hand, or e to
end game: n
Current Hand: a r e t i i n
Enter word, or a "." to indicate that you are finished:
inertia
"inertia" earned 99 points. Total: 99 points
Run out of letters. Total score: 99 points.

Enter n to deal a new hand, r to replay the last hand, or e to
end game: e
#---------------------------
```

Run the file `pset4_game.py`, without making any modifications to it, in order to ensure that everything is set up correctly (this means, open the file in IDLE, and use the Run command to load the file into the interpreter). The code we have given you loads a list of valid words from a file and then calls the `playGame` function. You will implement the functions it needs in order to work. If everything is okay, after a small delay, you should see the following printed out:

```
Loading word list from file...
      83667 words loaded.
playGame not yet implemented.
```

1.

    If you see an `IOError` instead (e.g., "No such file or directory"), you should change the value of the `WORDLIST_FILENAME` constant (defined near the top of the file) to the **complete pathname** for the file `words.txt` (This will vary based on where you saved the files).
    For example, if you saved all the files including this `words.txt` in the directory "C:/Users/Ana/6001x/PS4" change the line:
    `WORDLIST_FILENAME = "words.txt"` to something like
    `WORDLIST_FILENAME = "C:/Users/Ana/6001x/PS4/words.txt"`

Windows users, if you are copying the file path from Windows Explorer, you will have to change the backslashes to forward slashes.

The file `pset4_game.py` has a number of already implemented functions you can use while writing up your solution. You can ignore the code between the following comments, though you should read and understand how to use each helper function by reading the docstrings:

```
# ----------------------------------
# Helper code
# You don't need to understand this helper code,
# but you will have to know how to use the functions
# (so be sure to read the docstrings!)
    .
    .
    .
# (end of helper code)
```

2. `# ----------------------------------`

3. This problem set is structured so that you will write a number of modular functions and then glue them together to form the complete word playing game. Instead of waiting until the entire game is *ready*, you should test each function you write, individually, before moving on. This approach is known as *unit testing*, and it will help you debug your code.

We have provided several test functions to get you started. After you've written each new function, unit test by running the file `test_pset4_game.py` to check your work.

If your code passes the unit tests you will see a `SUCCESS` message; otherwise you will see a `FAILURE` message. These tests aren't exhaustive. You will want to test your code in other ways too.

Try running `test_pset4_game.py` now (before you modify the `pset4_game.py` skeleton). You should see that all the tests fail, because nothing has been implemented yet.

These are the provided test functions:

`test_getWordScore()`

Test the `getWordScore()` implementation.

`test_updateHand()`

Test the `updateHand()` implementation.

`test_isValidWord()`

Test the `isValidWord()` implementation.

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 `getWordScore` should accept as input a string of lowercase letters (a *word*) and return the integer score for that word, using the game's scoring rules.

A Reminder of the Scoring Rules

Scoring

- The score for the hand is the sum of the scores for each word formed.
- 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 word created.
- 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) for the four letters, then multiply by `len('weed')` to get (4+1+1+2)*4 = 32). Be sure to check that the hand actually has 1 'w', 2 'e's, and 1 'd' before scoring the word!
- As another example, if *n*=7 and you make the word 'waybill' on the first try, it would be worth 155 points (the base score for 'waybill' is (4+1+4+3+1+1+1)*7=105, plus an additional 50 point bonus for using all *n* letters).

Hints

- 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 `pset4_game.py`. 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). Our goal is to keep the code modular - if you want to try playing your word game with *n=10* or *n=4*, you will be able to do it by simply changing the value of `HAND_SIZE`!
- **Testing:** If this function is implemented properly, and you run `test_pset4_game.py`, you should see that the `test_getWordScore()` tests pass. Also test your implementation of `getWordScore`, using some reasonable English words.

Fill in the code for `getWordScore` in `pset4_game.py` and be sure you've passed the appropriate tests in `test_pset4_game.py` before pasting your function definition here.

Problem 2 - Dealing with Hands

****Please read this problem entirely!!**** The majority of this problem consists of learning how to read code, which is an incredibly useful and important skill. At the end, you will implement a short function. Be sure to take your time on this problem - it may seem easy, but reading someone else's code can be challenging and this is an important exercise.

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. Remember that with a dictionary, 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`. For example:

```
>>> hand['e']
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
KeyError: 'e'
>>> hand.get('e', 0)
0
```

Converting words into dictionary representation

One useful function we've defined for you is `getFrequencyDict`, defined near the top of `pset4_game.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:

```
>>> getFrequencyDict("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 `displayHand` function. Take a few minutes right now to read through this function 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, `dealHand`. The function takes as input a positive integer `n`, and returns a new object, a hand containing `n` lowercase letters. Again, take a few minutes (right now!) to read through this function carefully and understand what it does and how it works.

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`**. Your task is to implement the function `updateHand`, which takes in two inputs - a `hand` and a `word`

(string). `updateHand` uses letters from the hand to spell the word, and then returns a copy of the `hand`, containing only the letters remaining. For example:

```
>>> hand = {'a':1, 'q':1, 'l':2, 'm':1, 'u':1, 'i':1}
>>> displayHand(hand) # Implemented for you
a q l l m u i
>>> hand = updateHand(hand, 'quail') # You implement this function!
>>> hand
{'a':0, 'q':0, 'l':1, 'm':1, 'u':0, 'i':0}
>>> displayHand(hand)
l m
```

Implement the `updateHand` function. Make sure this function has no side effects: i.e., it must not mutate the hand passed in. Before pasting your function definition here, be sure you've passed the appropriate tests in `test_pset4_game.py`.

Hints

Testing

Testing: Make sure the `test_updateHand()` tests pass. You will also want to test your implementation of `updateHand` with some reasonable inputs.

Copying Dictionaries

You may wish to review the ".copy" method of Python dictionaries (review this and other Python dictionary methods here).

Your implementation of updateHand should be short (ours is 4 lines of code). It does not need to call any helper functions.

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 `input`) and score the word (using your `getWordScore`). 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. Implement the `isValidWord` function.

Testing: Make sure the `test_isValidWord` tests pass. In addition, you will want to test your implementation by calling it multiple times on the same hand - what should the correct behavior be? Additionally, the empty string (`''`) is not a valid word - if you code this function correctly, you shouldn't need an additional check for this condition.

Fill in the code for `isValidWord` in `pset4_game.py` and be sure you've passed the appropriate tests in `test_pset4_game.py` before pasting your function definition here.

Problem 4 - Hand Length

We are now ready to begin writing the code that interacts with the player. We'll be implementing the `playHand` function. This function allows the user to play out a single hand. First, though, you'll need to implement the helper `calculateHandlen` function, which can be done in under five lines of code.

Problem 5 - Playing a Hand

In `pset4_game.py`, note that in the function `playHand`, there is a bunch of *pseudocode*. This pseudocode is provided to help guide you in writing your function. Check out the Why Pseudocode? resource to learn more about the What and Why of Pseudocode before you start coding your solution.

Note: Do **not** assume that there will always be 7 letters in a hand! The parameter `n` represents the size of the hand.

Testing: Before testing your code, try out your implementation as if you were playing the game. Here is some example output of `playHand`:

Test Cases

Case #1

Function Call:

```
wordList = loadWords()
playHand({'h':1, 'i':1, 'c':1, 'z':1, 'm':2, 'a':1}, wordList, 7)
```

Output:

```
 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: .
 Goodbye! Total score: 45 points.
```

Case #2

Function Call:

```
wordList = loadWords()
playHand({'w':1, 's':1, 't':2, 'a':1, 'o':1, 'f':1}, wordList, 7)
```

Output:

```
 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

  Run out of letters. Total score: 46 points.
```

Case #3

Function Call:

```
wordList = loadWords()
playHand({'n':1, 'e':1, 't':1, 'a':1, 'r':1, 'i':2}, wordList, 7)
```

Output:

```
 Current Hand: a r e t i i n
  Enter word, or a "." to indicate that you are finished: inertia
  "inertia" earned 99 points. Total: 99 points

  Run out of letters. Total score: 99 points.
```

Additional Testing

Be sure that, in addition to the listed tests, you test the same basic test conditions with varying values of `n`. `n` will never be smaller than the number of letters in the hand.

Problem 6 - 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 `playGame`

function. You should remove the code that is currently uncommented in the `playGame` 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.

Testing: Try out this implementation as if you were playing the game. Try out different values for `HAND_SIZE` with your program, and be sure that you can play the wordgame with different hand sizes by modifying *only* the variable `HAND_SIZE`.

Sample Output

Here is how the game output should look...

```
Loading word list from file...
   83667 words loaded.
Enter n to deal a new hand, r to replay the last hand, or e to end game: r
You have not played a hand yet. Please play a new hand first!

Enter n to deal a new hand, r to replay the last hand, or e to end game: n
Current Hand: p z u t t t o
Enter word, or a "." to indicate that you are finished: tot
"tot" earned 9 points. Total: 9 points

Current Hand: p z u t
Enter word, or a "." to indicate that you are finished: .
Goodbye! Total score: 9 points.

Enter n to deal a new hand, r to replay the last hand, or e to end game: r
Current Hand: p z u t t t o
Enter word, or a "." to indicate that you are finished: top
"top" earned 15 points. Total: 15 points

Current Hand: z u t t
Enter word, or a "." to indicate that you are finished: tu
Invalid word, please try again.

Current Hand: z u t t
Enter word, or a "." to indicate that you are finished: .
Goodbye! Total score: 15 points.

Enter n to deal a new hand, r to replay the last hand, or e to end game: n
Current Hand: a q w f f i p
Enter word, or a "." to indicate that you are finished: paw
"paw" earned 24 points. Total: 24 points

Current Hand: q f f i
Enter word, or a "." to indicate that you are finished: qi
"qi" earned 22 points. Total: 46 points

Current Hand: f f
```

```
Enter word, or a "." to indicate that you are finished: .
Goodbye! Total score: 46 points.

Enter n to deal a new hand, r to replay the last hand, or e to end game: n
Current Hand: a r e t i i n
Enter word, or a "." to indicate that you are finished: inertia
"inertia" earned 99 points. Total: 99 points.

Run out of letters. Total score: 99 points.

Enter n to deal a new hand, r to replay the last hand, or e to end game: x
Invalid command.
Enter n to deal a new hand, r to replay the last hand, or e to end game: e
```

Hints about the output

Be sure to inspect the above sample output carefully - very little is actually printed out in this function specifically. Most of the printed output actually comes from the code you wrote in `playHand` - be sure that your code is modular and uses function calls to the `playHand` helper function!

You should also make calls to the `dealHand` helper function. You shouldn't make calls to any other helper function that we've written so far - in fact, this function can be written in about 15-20 lines of code.

Here is the above output, with the output from `playHand` obscured:

```
Loading word list from file...
   83667 words loaded.
Enter n to deal a new hand, r to replay the last hand, or e to end game: r
You have not played a hand yet. Please play a new hand first!

Enter n to deal a new hand, r to replay the last hand, or e to end game: n
<call to playHand>

Enter n to deal a new hand, r to replay the last hand, or e to end game: n
<call to playHand>

Enter n to deal a new hand, r to replay the last hand, or e to end game: n
<call to playHand>

Enter n to deal a new hand, r to replay the last hand, or e to end game: x
Invalid command.
Enter n to deal a new hand, r to replay the last hand, or e to end game: e
```

Hopefully this hint makes the problem seem a bit more approachable.

Entering Your Code

Be sure to only paste your definition for `playGame` below. Do not include any other function definitions.

A Cool Trick about 'print'

A cool trick about `print`: you can make two or more print statements print to the same line! Try out the following code. It will separate the first and second line with a space, and the second and third line with a "?" rather than putting each on a new line.

```
print('Hello', end = " ")
print('world', end="?")
print('!')
```

Part B is dependent on your functions from `pset4_game.py`, so be sure to complete `pset4_game.py` before working on `pset4_skynet.py`

Now that you have completed your word game code, you decide that you would like to enable your computer (SkyNet) to play the game (your hidden agenda is to prove once and for all that computers are inferior to human intellect!) In this part, you will be able to compare how you as a user succeed in the game compared to the computer's performance.

You should look at the following two functions: `compChooseWord` and `compPlayHand`, before moving on to Problem 7.

compChooseWord

If you follow the pseudocode for `compChooseWord`, you'll see that the code creates a computer player that is legal, but not always the best. Try to walk through and understand our implementation.

A Note On Runtime: You may notice that things run a bit slowly when the computer plays. This is to be expected - the `wordList` has 83667 words, after all!

Test Cases to Understand the Code:

```
>>> compChooseWord({'a': 1, 'p': 2, 's': 1, 'e': 1, 'l': 1}, wordList, 6)
appels
>>> compChooseWord({'a': 2, 'c': 1, 'b': 1, 't': 1}, wordList, 5)
acta
>>> compChooseWord({'a': 2, 'e': 2, 'i': 2, 'm': 2, 'n': 2, 't': 2}, wordList, 12)
immanent
>>> compChooseWord({'x': 2, 'z': 2, 'q': 2, 'n': 2, 't': 2}, wordList, 12)

None
```

compPlayHand

Now that we have the ability to let the computer choose a word, we need to set up a function to allow the computer to play a hand - in a manner very similar to Part A's `playHand` function. This function allows the computer to play a given hand and is very similar to the earlier version in which a user selected the word, although deciding when it is done playing a particular hand is different.

Test Cases to Understand the Code:

```
compPlayHand({'a': 1, 'p': 2, 's': 1, 'e': 1, 'l': 1}, wordList, 6)

Current Hand: a p p s e l

"appels" earned 110 points. Total: 110 points

Total score: 110 points.
```

```
compPlayHand({'a': 2, 'c': 1, 'b': 1, 't': 1}, wordList, 5)

Current Hand: a a c b t "acta"

earned 24 points. Total: 24 points
```

```
Current Hand: b Total score: 24 points.
```

```
compPlayHand({'a': 2, 'e': 2, 'i': 2, 'm': 2, 'n': 2, 't': 2},
wordList, 12)

Current Hand: a a e e i i m m n n t t

"immanent" earned 96 points. Total: 96 points

Current Hand: a e t i

"ait" earned 9 points. Total: 105 points

Current Hand: e

Total score: 105 points.
```

Problem 7 - You and your 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 `playGame` 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. Be sure to try out different values for `HAND_SIZE` with your program.

Sample Output and Hints

Here is how the game output should look...

Enter n to deal a new hand, r to replay the last hand, or e to end game: n

Enter u to have yourself play, c to have the computer play: u

Current Hand: a s r e t t t
Enter word, or a "." to indicate that you are finished: tatters
"tatters" earned 99 points. Total: 99 points

Run out of letters. Total score: 99 points.

Enter n to deal a new hand, r to replay the last hand, or e to end game: r

Enter u to have yourself play, c to have the computer play: c

Current Hand: a s r e t t t
"stretta" earned 99 points. Total: 99 points

Total score: 99 points.

Enter n to deal a new hand, r to replay the last hand, or e to end game: x
Invalid command.

Enter n to deal a new hand, r to replay the last hand, or e to end game: n

Enter u to have yourself play, c to have the computer play: me
Invalid command.

Enter u to have yourself play, c to have the computer play: you
Invalid command.

Enter u to have yourself play, c to have the computer play: c

Current Hand: a c e d x l n
"axled" earned 65 points. Total: 65 points

Current Hand: c n
Total score: 65 points.

Enter n to deal a new hand, r to replay the last hand, or e to end game: n

Enter u to have yourself play, c to have the computer play: u

Current Hand: a p y h h z o
Enter word, or a "." to indicate that you are finished: zap
"zap" earned 42 points. Total: 42 points

Current Hand: y h h o
Enter word, or a "." to indicate that you are finished: oy
"oy" earned 10 points. Total: 52 points

Current Hand: h h
Enter word, or a "." to indicate that you are finished: .
Goodbye! Total score: 52 points.

Enter n to deal a new hand, r to replay the last hand, or e to end game: r

Enter u to have yourself play, c to have the computer play: c

Current Hand: a p y h h z o
"hypha" earned 80 points. Total: 80 points

```
Current Hand:  z o
Total score: 80 points.

Enter n to deal a new hand, r to replay the last hand, or e to end game: e
```

Hints about the output

Be sure to inspect the above sample output carefully - very little is actually printed out in this function specifically. Most of the printed output actually comes from the code you wrote in `playHand` and `compPlayHand` - be sure that your code is modular and uses function calls to these helper functions!

You should also make calls to the `dealHand` helper function. You shouldn't make calls to any other helper function that we've written so far - in fact, this function can be written in about 15-20 lines of code.

Here is the above output, with the output from `playHand` and `compPlayHand` obscured:

```
Enter n to deal a new hand, r to replay the last hand, or e to end game: r
You have not played a hand yet. Please play a new hand first!

Enter n to deal a new hand, r to replay the last hand, or e to end game: n

Enter u to have yourself play, c to have the computer play: u

<call to playHand>

Enter n to deal a new hand, r to replay the last hand, or e to end game: r

Enter u to have yourself play, c to have the computer play: c

<call to compPlayHand>

Enter n to deal a new hand, r to replay the last hand, or e to end game: x
Invalid command.

Enter n to deal a new hand, r to replay the last hand, or e to end game: n

Enter u to have yourself play, c to have the computer play: me
Invalid command.

Enter u to have yourself play, c to have the computer play: you
Invalid command.

Enter u to have yourself play, c to have the computer play: c

<call to compPlayHand>
```

`Enter n to deal a new hand, r to replay the last hand, or e to end game: n`

`Enter u to have yourself play, c to have the computer play: u`

`<call to playHand>`

`Enter n to deal a new hand, r to replay the last hand, or e to end game: r`

`Enter u to have yourself play, c to have the computer play: c`

`<call to compPlayHand>`

`Enter n to deal a new hand, r to replay the last hand, or e to end game: e`

Hopefully this hint makes the problem seem a bit more approachable.

A Note On Runtime

You may notice that things run slowly when the computer plays. This is to be expected. If you want (totally optional!), feel free to investigate ways of making the computer's turn go faster - one way is to preprocess the word list into a dictionary (string -> int) so looking up the score of a word becomes much faster in the `compChooseWord` function.

Be careful though - you only want to do this preprocessing *one time* - probably right after we generate the wordList for you (at the bottom of the file). If you choose to do this, you'll have to modify what inputs your functions take (they'll probably take a word dictionary instead of a word list, for example).

IMPORTANT: Don't worry about this issue when running your code in the checker below! We load a very small sample wordList (*much* smaller than 83667 words!) to avoid having your code time out. Your code will work even if you don't implement a form of pre-processing as described.

Entering Your Code

Be sure to only paste your definition for `playGame` from `pset4_skynet.py` below. Do not include any other function definitions.

Problem 7 - Computer Chooses a Word

Part B is dependent on your functions from `pset4_game.py`, so be sure to complete `pset4_game.py` before working on `pset4_skynet.py`

Now that you have completed your word game code, you decide that you would like to enable your computer (SkyNet) to play the game (your hidden agenda is to prove once and for all that computers are inferior to human intellect!) In Part B you will make a modification to the `playHand` function from part A that will enable this to happen. The idea is that you will be able to compare how you as a user succeed in the game compared to the computer's performance.

It is your responsibility to create the function `compChooseWord(hand, wordList, n)`. Pseudocode is provided in the file `pset4_skynet.py`.

If you follow the pseudocode, you'll create a computer player that is legal, but not always the best. Once you've implemented it following our approach, feel free to try your own approach! As much as we'd love to give you credit for making an improved `compChooseWord` function, we hope you can understand our automatic grading facilities are limited in their ability to accept differing solutions.

Hints and Sample Output

A Note On Runtime

You may notice that things run a bit slowly when the computer plays. This is to be expected - the `wordList` has 83667 words, after all!

However, don't worry about this issue when running your code in the checker below! We load a very small sample wordList (*much* smaller than 83667 words!) to avoid having your code time out.

Test Cases

Some test cases to look at:

```
>>> compChooseWord({'a': 1, 'p': 2, 's': 1, 'e': 1, 'l': 1}, wordList, 6)
appels
>>> compChooseWord({'a': 2, 'c': 1, 'b': 1, 't': 1}, wordList, 5)
acta
>>> compChooseWord({'a': 2, 'e': 2, 'i': 2, 'm': 2, 'n': 2, 't': 2}, wordList, 12)
immanent
>>> compChooseWord({'x': 2, 'z': 2, 'q': 2, 'n': 2, 't': 2}, wordList, 12)
None
```

For the first test case your code might also find `apples`, depending on how you code your solution. This is okay and will check as correct.

Problem 8 - Computer Plays a Hand

Now that we have the ability to let the computer choose a word, we need to set up a function to allow the computer to play a hand - in a manner very similar to Part A's `playHand` function (get the hint?).

Implement the `compPlayHand` function. This function should allow the computer to play a given hand, using the procedure you just wrote in the previous part. This should be very similar to the earlier version in which a user selected the word, although deciding when it is done playing a particular hand will be different.

Be sure to test your function on some randomly generated hands using `dealHand`.

Test Cases

Test Cases

Some test cases to look at. Note it is okay if your code finds a different word, as long as the point values are the same.

`compPlayHand({'a': 1, 'p': 2, 's': 1, 'e': 1, 'l': 1}, wordList, 6)`
Current Hand: a p p s e l
"appels" earned 110 points. Total: 110 points
Total score: 110 points.
`compPlayHand({'a': 2, 'c': 1, 'b': 1, 't': 1}, wordList, 5)`
Current Hand: a a c b t
"acta" earned 24 points. Total: 24 points

Current Hand: b
Total score: 24 points.
`compPlayHand({'a': 2, 'e': 2, 'i': 2, 'm': 2, 'n': 2, 't': 2}, wordList, 12)`
Current Hand: a a e e i i m m n n t t
"immanent" earned 96 points. Total: 96 points

Current Hand: a e t i
"ait" earned 9 points. Total: 105 points

Current Hand: e
Total score: 105 points.

Important: For your code to be graded correctly, you must surround the computer's word with single or double quotes. So when displaying what word the computer chooses your line should look like:

"immanent" earned 96 points. Total: 96 points

or

'immanent' earned 96 points. Total: 96 points

Paste your definition of `compChooseWord`, in addition to your definition of `compPlayHand`, below.