1. Download and save [Problem Set 4](https://prod-edxapp.edx-cdn.org/assets/courseware/v1/1e89542cc136bc1e496aac7130cac1ba/asset-v1:MITx+6.00.1x+2T2019a+type@asset+block/ProblemSet4.zip), a zip file of all the skeleton code you'll be filling in. Extract the files from the zip folder and make sure to save *all* the files  - ps4a.py, ps4b.py, test\_ps4a.py and words.txt - in the **same folder**. We recommend creating a folder in your Documents folder called 6001x, and inside the 6001x folder, creating a separate folder for each problem set. If you don't follow this instruction, you may end up with issues because the files for this problem set depend on one another.
2. Run the file ps4a.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:
3. Loading word list from file...
4. 83667 words loaded.
5. playGame not yet implemented.

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.

1. The file ps4a.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:
2. # -----------------------------------
3. # Helper code
4. # You don't need to understand this helper code,
5. # but you will have to know how to use the functions
6. # (so be sure to read the docstrings!)
7. .
8. .
9. .
10. # (end of helper code)

# -----------------------------------

1. 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\_ps4a.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\_ps4a.py now (before you modify the ps4a.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**

 Bookmark this page

Problem 1 - Word Scores

0.0/10.0 points (graded)

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](https://courses.edx.org/courses/course-v1:MITx+6.00.1x+2T2019a/courseware/f0a19f0a8c2d49f3aa78ef3823845271/64f4d344ecdc48d2bef514882e6236ab/3?activate_block_id=block-v1%3AMITx%2B6.00.1x%2B2T2019a%2Btype%40vertical%2Bblock%40544d6c6b87ad4aea8034d5e0802aa37d)

**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 ps4a.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\_ps4a.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 ps4a.py and be sure you've passed the appropriate tests in test\_ps4a.py before pasting your function definition here.

## Problem 2 - Dealing with Hands

 Bookmark this page

### Problem 2 - Dealing with Hands

0.0/10.0 points (graded)

**\*\*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 ps4a.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\_ps4a.py.

### Hints

[Testing](https://courses.edx.org/courses/course-v1:MITx+6.00.1x+2T2019a/courseware/f0a19f0a8c2d49f3aa78ef3823845271/64f4d344ecdc48d2bef514882e6236ab/3?activate_block_id=block-v1%3AMITx%2B6.00.1x%2B2T2019a%2Btype%40vertical%2Bblock%40544d6c6b87ad4aea8034d5e0802aa37d)

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

[Copying Dictionaries](https://courses.edx.org/courses/course-v1:MITx+6.00.1x+2T2019a/courseware/f0a19f0a8c2d49f3aa78ef3823845271/64f4d344ecdc48d2bef514882e6236ab/3?activate_block_id=block-v1%3AMITx%2B6.00.1x%2B2T2019a%2Btype%40vertical%2Bblock%40544d6c6b87ad4aea8034d5e0802aa37d)

You may wish to review the ".copy" method of Python dictionaries (review this and other Python dictionary methods [here](http://docs.python.org/library/stdtypes.html#mapping-types-dict)).

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

 Bookmark this page

### Problem 3 - Valid Words

0.0/10.0 points (graded)

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 ps4a.py and be sure you've passed the appropriate tests in test\_ps4a.py before pasting your function definition here.

## Problem 4 - Hand Length

 Bookmark this page

### Problem 4 - Hand Length

0.0/10.0 points (graded)

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

 Bookmark this page

### Problem 5 - Playing a Hand

0.0/10.0 points (graded)

In ps4a.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?](https://prod-edxapp.edx-cdn.org/assets/courseware/v1/85721a1199ca98dda55d8992bc93658d/asset-v1:MITx+6.00.1x+2T2019a+type@asset+block/WhyPseudocode.pdf) 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 in the answer box, try out your implementation as if you were playing the game. Here is some example output of playHand:

### Test Cases

[Case #1](https://courses.edx.org/courses/course-v1:MITx+6.00.1x+2T2019a/courseware/f0a19f0a8c2d49f3aa78ef3823845271/64f4d344ecdc48d2bef514882e6236ab/3?activate_block_id=block-v1%3AMITx%2B6.00.1x%2B2T2019a%2Btype%40vertical%2Bblock%40544d6c6b87ad4aea8034d5e0802aa37d)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](https://courses.edx.org/courses/course-v1:MITx+6.00.1x+2T2019a/courseware/f0a19f0a8c2d49f3aa78ef3823845271/64f4d344ecdc48d2bef514882e6236ab/3?activate_block_id=block-v1%3AMITx%2B6.00.1x%2B2T2019a%2Btype%40vertical%2Bblock%40544d6c6b87ad4aea8034d5e0802aa37d)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](https://courses.edx.org/courses/course-v1:MITx+6.00.1x+2T2019a/courseware/f0a19f0a8c2d49f3aa78ef3823845271/64f4d344ecdc48d2bef514882e6236ab/3?activate_block_id=block-v1%3AMITx%2B6.00.1x%2B2T2019a%2Btype%40vertical%2Bblock%40544d6c6b87ad4aea8034d5e0802aa37d)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](https://courses.edx.org/courses/course-v1:MITx+6.00.1x+2T2019a/courseware/f0a19f0a8c2d49f3aa78ef3823845271/64f4d344ecdc48d2bef514882e6236ab/3?activate_block_id=block-v1%3AMITx%2B6.00.1x%2B2T2019a%2Btype%40vertical%2Bblock%40544d6c6b87ad4aea8034d5e0802aa37d)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

 Bookmark this page

### Problem 6 - Playing a Game

0.0/15.0 points (graded)

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...](https://courses.edx.org/courses/course-v1:MITx+6.00.1x+2T2019a/courseware/f0a19f0a8c2d49f3aa78ef3823845271/64f4d344ecdc48d2bef514882e6236ab/3?activate_block_id=block-v1%3AMITx%2B6.00.1x%2B2T2019a%2Btype%40vertical%2Bblock%40544d6c6b87ad4aea8034d5e0802aa37d)

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](https://courses.edx.org/courses/course-v1:MITx+6.00.1x+2T2019a/courseware/f0a19f0a8c2d49f3aa78ef3823845271/64f4d344ecdc48d2bef514882e6236ab/3?activate_block_id=block-v1%3AMITx%2B6.00.1x%2B2T2019a%2Btype%40vertical%2Bblock%40544d6c6b87ad4aea8034d5e0802aa37d)

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](https://courses.edx.org/courses/course-v1:MITx+6.00.1x+2T2019a/courseware/f0a19f0a8c2d49f3aa78ef3823845271/64f4d344ecdc48d2bef514882e6236ab/3?activate_block_id=block-v1%3AMITx%2B6.00.1x%2B2T2019a%2Btype%40vertical%2Bblock%40544d6c6b87ad4aea8034d5e0802aa37d)

Be sure to only paste your definition for playGame in the following box. Do not include any other function definitions.

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('!')