Problem 1 - Word Scores

(10/10 points)

The first step is to implement some code that allows us to calculate the score for a single word. The functiongetWordScore 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+2T2016/courseware/f0a19f0a8c2d49f3aa78ef3823845271/64f4d344ecdc48d2bef514882e6236ab/)

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 thetest\_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 intest\_ps4a.py before pasting your function definition here.

def getWordScore(word, n):

"""

Returns the score for a word. Assumes the word is a valid 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 turn.

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 (see SCRABBLE\_LETTER\_VALUES)

word: string (lowercase letters)

n: integer (HAND\_SIZE; i.e., hand size required for additional points)

returns: int >= 0

"""

# TO DO ... <-- Remove this comment when you code this function

score = 0

bonus = 50

# retrieve scrabble for each letter and add up to get score

for letter in word:

try:

score += SCRABBLE\_LETTER\_VALUES[letter]

except KeyError:

return 0

# get length of word and add up to score.

word\_len = len(word)

score \*= word\_len

# if user guessed entire length of word provide bonus point for user

if (word\_len == n):

score += bonus

return score

Problem 2 - Dealing with Hands

(10/10 points)

**\*\*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 forkey 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 intest\_ps4a.py.

def updateHand(hand, word):

"""

Assumes that 'hand' has all the letters in word.

In other words, this assumes that however many times

a letter appears in 'word', 'hand' has at least as

many of that letter in it.

Updates the hand: uses up the letters in the given word

and returns the new hand, without those letters in it.

Has no side effects: does not modify hand.

word: string

hand: dictionary (string -> int)

returns: dictionary (string -> int)

"""

# TO DO ... <-- Remove this comment when you code this function

new\_hand = hand.copy()

for letter in word:

new\_hand[letter] = new\_hand.get(letter) - 1

return new\_hand

Problem 3 - Valid Words

(10/10 points)

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 theisValidWord 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.pybefore pasting your function definition here.

def isValidWord(word, hand, wordList):

"""

Returns True if word is in the wordList and is entirely

composed of letters in the hand. Otherwise, returns False.

Does not mutate hand or wordList.

word: string

hand: dictionary (string -> int)

wordList: list of lowercase strings

"""

# TO DO ... <-- Remove this comment when you code this function

new\_hand = hand.copy()

if word in wordList:

for letter in word:

try:

if (letter not in new\_hand) or (new\_hand.get(letter) == 0):

return False

else:

new\_hand[letter] = new\_hand.get(letter) - 1

except KeyError:

return False

return True

else:

return False

Problem 4 - Hand Length

(10/10 points)

We are now ready to begin writing the code that interacts with the player. We'll be implementing the playHandfunction. 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.

def calculateHandlen(hand):

"""

Returns the length (number of letters) in the current hand.

hand: dictionary (string int)

returns: integer

"""

# TO DO... <-- Remove this comment when you code this function

length = 0

for letter in hand:

if hand[letter] != 0:

length += hand[letter]

return length

Problem 5 - Playing a Hand

(10 points possible)

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://d37djvu3ytnwxt.cloudfront.net/assets/courseware/v1/85721a1199ca98dda55d8992bc93658d/asset-v1:MITx+6.00.1x+2T2016+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:

#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.