**Name: Session:**

**Programming I**

**A Scrabble Game**

**Lab Exercise 12.17.2019**

**INTRODUCTION**

In this problem, you'll implement a simple word game!

Let's begin by describing the word game: 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).

**GETTING STARTED**

Download and save the project files from the server,  scrabble.py,  test\_scrabble.py  and  words.txt - into the **same folder**.

Run the file scrabble.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 I 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.

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 filewords.txt (This will vary based on where you saved the file).

The file scrabble.py has a number of already implemented functions you can use. 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)

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

This problem 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.

I have provided several test functions to get you started. After you've written each new function, unit test by running the file  test\_scrabble.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\_scrabble.py now (before you modify the scrabble.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.

**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.

**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 scrabble.py and be sure you've passed the appropriate tests in test\_scrabble.py before moving on.

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

"""

points = 0

for i in range(len(word)):

wordKey = word[i]

points += SCRABBLE\_LETTER\_VALUES[wordKey]

points \*= len(word)

if n == len(word):

points += 50

return points

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 defined for you is getFrequencyDict, defined near the top of scrabble.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

{'l': 1, 'm': 1}

>>> displayHand(hand)

l m

(**NOTE:** In the above example, after the call to updateHand, it is also acceptable for the value of hand to be the dictionary {'a':0, 'q':0, 'l':1, 'm':1, 'u':0, 'i':0}. The value of hand will depend on your implementation; but the output of  displayHand() will be the same in either case.)

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\_scrabble.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)

"""

import copy

newHand = copy.copy(hand)

for i in range(len(word)):

wordKey = word[i]

newHand[wordKey] -= 1

if newHand[wordKey] == 0:

newHand.pop(wordKey)

return newHand

## 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 scrabble.py and be sure you've passed the appropriate tests in test\_scrabble.py.

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

"""

import copy

newHand = copy.copy(hand)

found = False

#check for empty word

if word == '':

return False

#check to see if word is in wordList

for i in range(len(wordList)):

if word == wordList[i]:

found = True

#check to see if all letters in the word are in hand

for i in range(len(word)):

if word[i] not in newHand:

found = False

else:

newHand[word[i]] -= 1

if newHand[word[i]] == 0:

newHand.pop(word[i])

return found

## 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 five lines of code.

def calculateHandlen(hand):

"""

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

hand: dictionary (string int)

returns: integer

"""

values = list(hand.values())

total = 0

for i in range(len(values)):

total += values[i]

return total

## Playing a Hand

In scrabble.py, there is a function playHand, here is some *pseudocode* for that function. This pseudocode is provided to help guide you in writing your function.

# Keep track of two numbers: the number of letters left in your hand and the total score

# As long as there are still letters left in the hand:

# Display the hand

# Ask user for input

# If the input is a single period:

# End the game (break out of the loop)

# Otherwise (the input is not a single period):

# If the word is not valid:

# Reject invalid word (print a message followed by a blank line)

# Otherwise (the word is valid):

# Tell the user how many points the word earned, and the updated total score, in one line followed by a blank line

# Update the hand

# Game is over (user entered a '.' or ran out of letters), so tell user the total score

**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 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.

Here is the playHand function:

def playHand(hand, wordList, n):

"""

Allows the user to play the given hand, as follows:

\* The hand is displayed.

\* The user may input a word or a single period (the string ".")

to indicate they're done playing

\* Invalid words are rejected, and a message is displayed asking

the user to choose another word until they enter a valid word or "."

\* When a valid word is entered, it uses up letters from the hand.

\* After every valid word: the score for that word is displayed,

the remaining letters in the hand are displayed, and the user

is asked to input another word.

\* The sum of the word scores is displayed when the hand finishes.

\* The hand finishes when there are no more unused letters or the user

inputs a "."

hand: dictionary (string -> int)

wordList: list of lowercase strings

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

"""

totalScore = 0

n = calculateHandlen(hand)

while calculateHandlen(hand) > 0:

print ("Current hand: ", end = ' ')

displayHand(hand)

word = input("Enter word, or a " + '"'+ "." + '"' + " to indicate that you are finished: ")

if word == '.':

break

else:

if not isValidWord(word, hand, wordList):

print ("Invalid word, please try again.")

print()

else:

score = getWordScore(word, n)

totalScore += score

print('"' + word +'"'+ " earned " + str(score) + " points. " + "Total: " + str(totalScore))

hand = updateHand(hand, word)

print()

if len(hand) > 0:

print("Goodbye! Total score: " + str(totalScore) + " points.")

else:

print("Ran out of letters. Total score: " + str(totalScore) + " points.")

## 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.

def playGame(wordList):

"""

Allow the user to play an arbitrary number of hands.

1) Asks the user to input 'n' or 'r' or 'e'.

\* If the user inputs 'n', let the user play a new (random) hand.

\* If the user inputs 'r', let the user play the last hand again.

\* If the user inputs 'e', exit the game.

\* If the user inputs anything else, tell them their input was invalid.

2) When done playing the hand, repeat from step 1

"""

hand = {}

answer = input("Enter n to deal a new hand, r to replay the last hand, or e to end game: ")

while answer != 'e':

if answer == 'n':

hand = dealHand(HAND\_SIZE)

playHand(hand, wordList, HAND\_SIZE)

elif answer == 'r':

if len(hand) == 0:

print("You have not played a hand yet. Please play a new hand first!")

else:

playHand(hand, wordList, HAND\_SIZE)

else:

print("Invalid command.")

answer = input("Enter n to deal a new hand, r to replay the last hand, or e to end game: ")

**When have your game working, print out the Python shell that shows a transcript of a round of game play and attach to this sheet.**