15-112: Introduction to Programming and Computer Science, Spring 2020

Homework 4: Strings

Due: Wednesday, February 12, 2020 by 22:00

This assignment has 7 questions, for a total of 50 points.
To start this homework....

1. Create a folder named "week4"
2. Create a file called hw4.py and write all your code in that file.
3. When you have completed and fully tested hw4, submit hw4.py to Autolab. For this hw, you may submit up to 15 times, but only your last submission counts.

Some important notes:

1. You may not use sets, dictionaries, try/except, or classes on this homework.
2. After you submit to Autolab, make sure you check your score. If you aren't sure how to do this, then ask a CA or Professor.
3. There is no partial credit on Autolab testcases. Your Autolab score is your Autolab score.
4. Read the last bullet point again. Seriously, we won't go back later and increase your Autolab score for any reason. Even if you worked really hard and it was only a minor error...
5. Do not hardcode the test cases in your solutions.
6. Remember the course's academic integrity policy. Solving the homework yourself is your best preparation for exams and quizzes; cheating or short-cutting your learning process in order to improve your homework score will actually hurt your course grade long-term.
7. Your code will be graded for style. Check the style notes on the website for details.

1. [5 points] **consonantCount(s)**

   Write the function consonantCount(s), that takes a string s, and returns the number of consonants in s, ignoring case, so "B" and "b" are both consonants. The consonants are all of the English letters except "a", "e", "i", "o", and "u". So, for example:

   ```
   assert(consonantCount("Abc def!!! a? yzyzyz!") == 10)
   ```

2. [5 points] **topScorer**

   Write the function topScorer(data) that takes a multi-line string encoding scores as csv data for some kind of competition with players receiving scores, so each line has comma-separated values. The first value on each line is the name of the player (which you can assume has no integers in it), and each value after that is an individual score (which you can assume is a non-negative integer). You should add all the scores for that player, and then return the player with the highest total score. If there is a tie, return all the tied players in a comma-separated string with the names in the same order they appeared in the original data. If nobody wins (there is no data), return None (not the string "None"). So, for example:

   ```
   data = '''\
   Fred,10,20,30,40
   Wilma,10,20,30
   '''
   assert(topScorer(data) == 'Fred')

   data = '''\
   Fred,10,20,30
   Wilma,10,20,30,40
   '''
   assert(topScorer(data) == 'Wilma')

   data = '''\
   Fred,11,20,30
   Wilma,10,20,30,1
   '''
   assert(topScorer(data) == 'Fred,Wilma')

   assert(topScorer('') == None)
   ```

   Hint: you may want to use both splitlines() and split(',') here!

3. [6 points] **getQuizAverages**

   Write the function getQuizAverages(filename) that takes the name of a file containing quiz scores for a course in CSV format.

   The first value on each line is the name of the student (which you can assume has no integers in it), and each value after that is an individual score (which you can assume is a non-negative integer) out of 100. You should compute the average quiz score for each student and return a list containing tuples of (name, avg) pairs. The order of tuples in the returned list should match the order that students are found in the file.

   There are some important things to note:

   - If a score is the character `"-"`, then that means the student was exempted from that quiz and it should not be factored into their grade at all.
   - If a score is missing, then the student did not complete the quiz and it should count as a 0.
   - The lowest score for each student should be dropped prior to computing the average.
   - In your tuple containing the name and quiz average, the quiz average should be a string of the quiz average accurate to 2 decimal places.

   If the file contains no score data, then return None.

   So, for example:

```
# Imagine the file contains the following three lines and
# is named "whatever.txt".
'''
Fred,11,20,30,10,12
Wilma,10,20,,1,25
Barney,10,5,-,20,25
'''
assert(getQuizAverages("whatever.txt")
       == [('Fred', '18.25'), ('Wilma', '14.00'), ('Barney', '18.33')])
```

   Hint: You should probably look at the notes on string formatting to think about how to handle converting the average to a string.

4. [6 points] **applyCaesarCipher(message, shift)**

   A Caesar Cipher is a simple cipher that works by shifting each letter in the given message by a certain number. For example, if we shift the message "We Attack At Dawn" by 1 letter, it becomes "Xf Buubdl Bu Ebxo".

   Write the function applyCaesarCipher(message, shift) which shifts the given `message` by `shift` letters. You are guaranteed that message is a string, and that shift is an integer between -25 and 25. Capital letters should stay capital and lowercase letters should stay lowercase, and non-letter characters should not be changed. Note that "Z" wraps around to "A". So, for example:

```
assert(applyCaesarCipher("We Attack At Dawn", 1) == "Xf Buubdl Bu Ebxo")
assert(applyCaesarCipher("zodiac", -2) == "xmbgya")
```

5. [8 points] **isFloat Error Checking**

   We have seen in some lecture notes that if we want to read a float from the user, we use the function input() to read a string from the user and then use the function float() to convert that string to a float number. We also learned that if the user enters an invalid number, our program will crash. Now we don't like programs that crash. So, we would like to check if a string can be a float before we attempt to convert it to a float. This way, if the string is not float, we can print an error message and exit gracefully instead of crashing the program. If would be nice to have a function called isFloat that takes an input of string and returns back True if the input is a valid float and False if it is not. We can use this function in our programs as given below:

```
inp = input("Enter your QPA: ")
if( not isFloat(inp)):
    print ()"You entered a value that is not a valid QPA. Exiting gracefully :)")
    exit()
else:
    # we can safely decode the float now
    qpa = float(inp)
```

   Your task is to write the isFloat function that checks the string passed to it for validity of being a float number. If the input is a valid float number, it should return True, otherwise, it should return False. We will be calling the function as shown in the above example, so make sure the return values, function name, and input parameters are appropriate. You should NOT use a try/except structure for this function.

6. [10 points] **Oh So Many Units** Most questions of physics and chemistry require conversion between units to get the correct units needed for an equation. Einstein's famous equation $E = mc^2$ allows you to find Energy in Joules when you multiply mass in Kilograms by the square of the speed of light in meters per second. For this equation to work correctly, the units of each quantity have to be exact. If I know the mass of an object in pounds (pounds measure force but oh well), I would have to convert pounds to kilograms first and then apply the equation. In this task, we (that means you) will be performing conversions with length, mass, time and volume units.

   Write a function called convertUnits that takes 4 input arguments. These inputs are fromQuantity, fromUnit, toUnit, and category. "fromQuantity" is an amount that represents a quantity in "fromUnit" units. Your function should convert "fromQuantity" in "fromUnit" units, to a number in "toUnit" units. You should follow the tables below and only use the numbers in the table or else you will not get the points.

   For example, the function call convertUnits(1000.0,"mm","m","length") is asking you to convert 1000.0 millimeters to meters using length category. The return value of the above call should be 1.0.

   The function call convertUnits(156.5,"lb","kg","mass") converts 156.6 pounds to kilograms and returns 70.987148.

   Your function should be able to handle the following units - we will only ask you to convert between units that belong to the same category (for example, we will not ask you to convert between seconds and meters):

Unit	Description	Category	Conversion
g	grams	mass	1 g = 10^{-6} ton
g	grams	mass	1 g = 0.0022 lb
ton	tons	mass	1 ton = 10^{6} g
lb	pounds	mass	1 lb = 453.592 g
s	seconds	time	1 s = 0.0003 hr
s	seconds	time	1 s = 0.0167 min
hr	hours	time	1 hr = 3600.0 s
min	minutes	time	1 min = 60.0 s
m	meters	length	1 m = 1.0936 yard
m	meters	length	1 m = 3.2802 feet
m	meters	length	1 m = 39.3701 inch
yard	yards	length	1 yard = 0.9144 m
foot	feet	length	1 foot = 0.3048 m
inch	inches	length	1 inch = 0.0254 m
C	Celsius	temperature	Kelvin = Celsius + 273.15
K	Kelvin	temperature	Celsius = Kelvin - 273.15
C	Celsius	temperature	Fahrenheit = (9/5)*Celsius + 32
F	Fahrenheit	temperature	Celsius = (5/9)*(Fahrenheit - 32)

Your function should also be able to handle the following prefixes before each unit.

Prefix	Symbol in function	Meaning
Terra	T	10^{12}
Giga	G	10^{9}
Mega	M	10^{6}
Kilo	k	10^{3}
Deci	d	10^{-1}
Centi	c	10^{-2}
Milli	m	10^{-3}
Micro	u	10^{-6}
Nano	n	10^{-9}
Pico	p	10^{-12}

7. [10 points] **bestScrabbleScore(dictionary, letterScores, hand)**

Background: In a Scrabble-like game, players each have a hand, which is a list of lowercase letters. There is also a dictionary, which is a list of legal words (all in lowercase letters). And there is a list of letterScores, which is length 26, where letterScores[i] contains the point value for the ith character in the alphabet (so letterScores[0] contains the point value for 'a'). Players can use some or all of the tiles in their hand and arrange them in any order to form words. The point value for a word is 0 if it is not in the dictionary, otherwise it is the sum of the point values of each letter in the word, according to the letterScores list (pretty much as it works in actual Scrabble).

In case you are interested, here is a list of the actual letterScores for Scrabble:

```
letterScores = [
#  a, b, c, d, e, f, g, h, i, j, k, l, m
   1, 3, 3, 2, 1, 4, 2, 4, 1, 8, 5, 1, 3,
#  n, o, p, q, r, s, t, u, v, w, x, y, z
   1, 1, 3,10, 1, 1, 1, 1, 4, 4, 8, 4,10
]
```

Note that your function must work for any list of letterScores as is provided by the caller.

With this in mind, write the function bestScrabbleScore(dictionary, letterScores, hand) that takes 3 lists -- dictionary (a list of lowercase words), letterScores (a list of 26 integers), and hand (a list of lowercase characters) -- and returns a tuple of the highest-scoring word in the dictionary that can be formed by some arrangement of some subset of letters in the hand, followed by its score. In the case of a tie, the first element of the tuple should instead be a list of all such words in the order they appear in the dictionary. If no such words exist, return None.

Note: you should definitely write helper functions for this problem! In fact, try to think of at least two helper functions you could use before writing any code at all.

Another note: there is no fixed dictionary here. Each time we call the function, we may provide a different dictionary! It may contain 100 words or perhaps 100,000 words.