CS5001
Fall 2019
Homework 6
Assigned: November 7, 2019
Deadline: November 21, 2019 at 9:05pm

To submit your solution, compress all files together into one .zip file. Login to `handins.ccs.neu.edu` with your CCIS account, click on the appropriate assignment, and upload that single zip file. You may submit multiple times right up until the deadline; we will grade only the most recent submission.

Your solution will be evaluated according to our CS 5001 Grading Rubric. This is a programming-only assignment! There are two programs; both count for 50% of your HW6 score.

Programming #1: Drug-Drug Interaction Assistant
Drug-drug interaction (DDI)refers to the changes in a drug's efficacy and effect on the body when the drug is taken together with a second (or third) drug. This is a serious problem when patients have multiple chronic illnesses (known as multiple morbidities). For example, if a patient has hypertension AIDS and diabetes, they may be using a multitude of single drugs or drug "cocktails" that work fine by themselves but may have adverse (or different) indications when taken together. A drug-drug interaction can delay, decrease, or enhance absorption of either drug. This interaction decreases or increases the effectiveness of the drugs and can cause adverse effects for the patient. My former colleagues at IBM Research Africa worked quite extensively on this problem, and we're going to do a simple version of a DDI Assistant here.
This part of the assignment will utilize your knowledge of classes, exception handling and files.

Copy and save the following two files as `rxcui_drugs.txt`:

```
Amantadine|831103|Treats Influenza A and Parkinson's|Phenelzine,Rizatriptan
Aluminum Hydroxide|153329|Used as an antacid|*
Desipramine|1099289|Anti-depressant|Rizatriptan
Ciprofloxin|197511|Antibiotic for treating infections|Isocarboxazid
Dextromethorphan Polistirex|1373045|Cough medicine|*
Phenelzine|8123|Anti-depressant|Amantadine
Rizatriptan|88014|Treats migraine headaches|Desipramine,Amantadine
Moclobemide|30121|Treatment of social anxiety|*
Cortisone|329229|Lessen inflammation|*
Melatonin|1362753|Sleep aid|*
Acetaminophen|198439|Fever reducer|Ibuprofen
Isocarboxazid|6011|Treats severe depression|Ciprofloxin
```

And `prescriptions.txt`:

```
Clark Kent|Back pain|Cortisone
Peter Parker|Depression and Migraines|Desipramine,Rizatriptan
Bruce Wayne|Insomnia|Melatonin
Barry Allen|Scraped knee and depression|Ciprofloxin,Isocarboxazid
Diana Prince|Cough and sore throat|Dextromethorphan Polistirex
Natasha Romanoff|Flu and fever|Amantadine,Acetaminophen
```

Here's what you need to do:
1. Create a RxDrug class. Save your class in the file rxdrug.py. This will be your user-defined type that helps us track the drugs and DDI for this assignment. The class specification is below. Your class should implement at least the 5 methods shown below (and I've given you set_description() as an example). Your class may implement *more* than the required 5, but to get full credit for this assignment, you need to build your class with at least these 5.

```
class RxDrug:
    def __init__(self, name, rx_ID):

    def add_interaction(self, other_drug):
        # if other_drug is not in our interactions list
            # add other_drug to our interactions list

    def check_interaction(self, other_drugs):
       # given a list of drugs that possibly interact with us
       # return a list containing the names of the drugs that we
       # actually match with from the list passed in

    def set_description(self, description):
        self.description = description

    def __str__(self):
```

2. Create an rxdrug_driver.py file. Write functions (or classes if you wish) to:

- Read the drug name, RCUIS # and manufacturer from `rxcui_drugs.txt`. Each line of the file is a RCUIS entry with 4 elements, each element separated by | The elements are:
    1. Item name
    2. RX_ID
    3. Description
    4. List of drugs this drug interacts with (comma separated, no spaces)
- Create an RxDrug instance for each RCUIS entry in the file, initializing the object with the data you read
- Open and read the Prescriptions file `prescriptions.txt`. Remember our EHR from a previous lab? Our superheroes are back again, but this time we're using persistent data (file based) to retrieve their information. This file contains the patient name, ailment description, and prescription list separated by |
- For each patient, determine if a DDI exists due to their prescribed medication. If so, print a warning. If not, print a simple message that shows there is no dangerous interactions.

Mandatory: You need at least 1 class: RxDrug. You can use more classes for your solution if you want (e.g. Patient, etc.). You must also have at least 1 try/except block in your code.

Amazing points: Write to an output file and log the interactions so we can save and inspect them after the program finishes running. (this file can have the same output as what we see on the console)

Here is a sample run:

```
 RESTART: /Users/keithbagley/Dropbox/NORTHEASTERN UNIVERSITY/CS5001/Fall 2019/Homework/HW

Checking prescriptions for  Clark Kent  who suffers from  Back pain
Current Rx:  Cortisone
No serious drug interactions found in this Rx
--------------------
Checking prescriptions for  Peter Parker  who suffers from  Depression and Migraines
Current Rx:  Desipramine,Rizatriptan
Warning: drug-drug interaction between  Desipramine  and  Rizatriptan
--------------------
Checking prescriptions for  Bruce Wayne  who suffers from  Insomnia
Current Rx:  Melatonin
No serious drug interactions found in this Rx
--------------------
Checking prescriptions for  Barry Allen  who suffers from  Scraped knee and depression
Current Rx:  Ciprofloxin,Isocarboxazid
Warning: drug-drug interaction between  Ciprofloxin  and  Isocarboxazid
--------------------
Checking prescriptions for  Diana Prince  who suffers from  Cough and sore throat
Current Rx:  Dextromethorphan Polistirex
No serious drug interactions found in this Rx
--------------------
Checking prescriptions for  Natasha Romanoff  who suffers from  Flu and fever
Current Rx:  Amantadine,Acetaminophen
No serious drug interactions found in this Rx
--------------------
>>>
```

Interesting Background Information for DDI:

The **RxNorm** concept unique identifier (RXCUI) for the clinical drug or substance is shared amongst organizations and published by the NIH. It's a unique ID for drugs used in the U.S. and helps disambiguate the core substance from variations in brand, generics, etc. The NIH also has an API Application Programming Interface you can interact with (remember in HW2 when we talked about Service Oriented Architecture – here's an open API you can use to pull useful information from a back-end service). The drugs and IDs you're using are real; the link below gives you a sense of the data you can pull using the API if you want to explore more:
https://rxnav.nlm.nih.gov/InteractionAPIs.html#uLink=Interaction_REST_findInteractionsFromList

Programming #2: Word Game

- Starter code:
  - `wordlist.txt` (download this into the same directory as your .py files, but don't submit it with your solution)
  - `wordlist.py` (to process the wordlist)
  - `test_scrabble.py` (to test the functions you're required to write as part of hw5)
- Your files:
  - `wordgame.py` (the driver)
  - `scrabble_points.py` (functions)

This is a word game that's sort of a cross between Scrabble and Countdown. It's a one-player game. The player attempts to create words from 7 randomly-chosen letters, and they are awarded points according to the value of each letter.

Points	**Letters**
1	A, E, I, O, U, L, N, S, T, R
2	D, G
3	B, C, M, P
4	F, H, V, W, Y
5	K
8	J, X
10	Q, Z

Seven letters are in play at a time, and they're chosen at random from a big bag of 100 letters. Most letters have more than one instance in the bag, following these frequencies:

Frequency	**Letters**
1	J, K, Q, X, Z
2	B, C, F, H, M, P, V, W, Y, blank
3	G
4	D, L, S, U
6	N, R, T
8	O
9	A, I
12	E

Play proceeds as follows:

- We begin with a bag of letters, following the table above -- there are 12 "E" letters in the bag, 9 "A" letters, 9 "I" letters, and so on.
- The user is repeatedly presented with a menu of options:

    - **D** - Draw 7 letters from the bag
    - **W** - Make a word from the letters in play
    - **P** - Print all words played so far
    - **Q** - Quit
- The game is over when the player enters Q to quit, or there are no more letters in the bag or in play.

Player Chooses D to Draw

Every time the user chooses D to draw, 7 letters are removed at random from the bag and put into play.

When the user chooses D, if any letters are already in play, they are discarded. They don't go back into the bag; they're just gone.

Player Chooses W to Make a Word from the Letters in Play

Every time the user chooses W to make a word, they are prompted to enter a word that can be formed with the 7 letters currently in play.

- If they enter a word that is not in the wordlist, it is invalid and they get no points. The letters in play remain the same.
- If they enter a word that cannot be made from the letters currently in play, it is invalid and they get no points. The letters in play remain the same.
- If they enter a word that they've played it already, it is invalid and they get no points. The letters in play remain the same.
- If they enter a valid word, they are awarded points for it based on the letters used. The letters used for the word are no longer in play and replaced with new letters.

For a valid word, count up the total points based on the letter values described above. Add the word's value to the player's overall points.

Player Chooses P to Print

Every time the user chooses P to print, they see all the valid words they've played in the game so far, along with the points awarded for each one, something like this:

```
You have a total of 12 points so far.
HEN -- 6 points
BLUE -- 6 points
```

Player Chooses Q to Quit

The game is over when the user enters Q to quit, or there are no more letters in the bag or in play.

Note that the D-draw option discards all the letters currently in play and replaces them with 7 new ones. But when the user plays a valid word, only the letters used in the word are replaced. The letters-in-play get kind of "refreshed", if you will, but not completely replaced unless the word actually used all 7 letters.

Starter Code

- We've written code that reads from the file **`wordlist.txt`**, and returns a list of strings to represent the dictionary. The wordlist.txt file, which you need to download into the same directory as your Python files, is the built-in Unix wordlist. We do some clean-up, such as converting the words to all uppercase and removing spaces. There are still some words with punctuation, and some words I'm surprised are words ('snee'?) but we'll consider it a valid wordlist and let those eccentricities be Alex Trebek's problem.

- We've also provided a test suite to test two functions that you are required to write. Beyond these, any functions you choose to define are up to you as long as they meet our usual standards of "good" functions. You must write functions that pass the tests included for:
    - `bag_of_letters`
        - Params: dictionary where *key* = letter and *value* = frequency of that letter.
        - Returns: a list of letters, with each one repeated according to its frequency.
        - Example Input: `{'A':1, 'B':2}`. Expected output: `['A', 'B', 'B']`
    - `get_word_value`.
        - Params: a word to evaluate (string), and a dictionary where *key* = letter and *value* = points that letter is worth.
        - Returns: the total point value of the word (an int). If any letter in the word does not appear in the letter-value mapping, then return 0 points.
        - Example Inputs: `'HI', {'H':4, 'I':1}`. Expected output: `5`
        - Example Inputs: `'HELLO', {'H':4, 'I':1}`. Expected output: `0`

Requirements:

- Write the two functions as described above such that all the tests in our provided test suite pass.
- For additional functions you write, you are not required to submit a test suite, but you need to be confident that they work for any reasonable input.
- Point values and letter frequencies must be allocated using the distributions above.
- Allow the user to enter upper or lowercase letters, both for their choice from the menu and the words they want to play in the game.
- If the user enters an invalid menu option, repeatedly re-prompt them until they enter a good one.
- When the user plays a valid word, they are awarded points, which is added to their total.
- When the user attempts to play an **invalid** word (not in the dictionary from wordlist.txt, not possible from the letters in play, and/or they already got points for the word), they get no points and the letters in play stay the same.
- If the user chooses D for draw, the 7 letters currently in play are discarded and 7 new letters are drawn.
- If a blank tile is in play, it can take the place of any single letter.
- Unlike in scrabble, we count the blank tiles as whatever the letters are worth.
    - For example, suppose the letters in play are `'A', 'E', 'C', '_', 'W', 'Z',` and `'L'`. The user can play the word `CAKE`, using the blank tile as a K. The total points for this point are: 3(C) + 1(A) + 5(K) + 1(E) = 10.
    - In "real" scrabble, the blank would count as zero points. We're nicer.
    - The user could *not*, however, make the word `CABLE`, because the blank tile can count for one letter but not two.
- The game ends when the user enters Q or there are no more letters in the bag or in play.

Example of game play (as always, your wording can vary as long as your program is friendly and informative!). Laney came up with this problem and made a video of the gameplay rather than screen shots. Find it here: https://youtu.be/fE6HwVy8HTE

AMAZING points:

- Use constants for any variable whose value doesn't change once it's defined.
- Write a test suite for every function you add. Maybe there's one function that you don't test because it prompts the user, but it's really tiny. Everything else is tested, and you and I can both be confident that it works great.