Week 13 Practice Problems

Week 13 is a "review" week. Here are some questions that should exercise a bunch of the stuff we've studied in the course. Some of these problems may be covered in the lectures during Week 13.

#1. Create your own text file with TextEdit or NotePad. Save it as a .txt file! The file should have multiple lines - at least 10 of them to make it interesting. Make sure your code works with both an odd and an even number of lines.

Write a function that takes a string as a parameter that will be the name of the file. Your function should do the following:

- open the file whose name is passed in for reading
- open a second file whose name is the same as the one passed in with a "_rev" before the .txt extension. So, if the parameter passed in is "my_text.txt", you should create a new file called "my_text_rev.txt".
- write the contents of the first file to the second file but swap the order of each pair of consecutive lines. For example, if this is your input file:

```
Bicycle
We are the champions
Bohemian rhapsody
We will rock you
Another one bites the dust
```

then this should be the output file:

```
We are the champions
Bicycle
We will rock you
Bohemian rhapsody
Another one bites the dust
```

a) Write the function using a list. (Easy)

b) Write the function without using any collections (no lists, tuples, sets, dictionaries). You cannot write your own collection. Why? What if your textfile is 800 Gb? In part a) you will try to read the whole thing into memory and your program will either crash or take a very long time. Here, you should write the code so that you only ever had two lines in memory at any one time. (Intermediate)

#2. Write a program (you can choose to use functions or not) to create a "fake" marks file as follows.

- the course has the following assessments: Assignment1, Assignment2, Midterm1, Midterm2, Final
    - Assignment1 has 1 part A
    - Assignment2 has 2 parts A, B
    - Midterm1 has 4 parts (A-D)
    - Midterm2 has 5 parts (A-E)
    - Final has 5 parts (A-E)
- each assessment is marked out of 100 and so the sum of the values of each part must be 100. Each part of a given assessment is worth the same. So if there are 4 parts, each is worth 25 marks.
- you have 30 students in the class and each one has a unique 3-digit ID number. For each student, generate a random 3-digit number ID.
- For each student, for each part of each assessment, generate the student's mark. The mark must be between 0 and the value of that part of that assessment.
- Write out the marks file as a CSV with the following format:

```
id, assessment_name, mark_part1, mark_part2, ...
```

(up to the number of parts of the assignment). For example, the first three lines of the file might be:

```
333, Assignment1, 67
333, Assignment2, 44, 20
333, Midterm1, 20, 21, 18, 22
```

This file will be used in Q3.

One option is to create data structures using classes and/or dictionaries. You could do it this way. However, the main purpose of this file is to be used in Q3, so you can feel free to be "quick and dirty". My code takes about 20 lines.

#3. The purpose of this problem is to give you practice with nested dictionaries.

a) Write a function that takes in a filename, reads in the CSV file you created in Q2 from a file of that name, and returns a dictionary of the following form:

```
{id : {assessment : total_mark}}
```

This is a nested dictionary. The outer dictionary has keys corresponding to student IDs. For each student ID, the value is an inner dictionary whose keys are the 5 assessments and whose values are the total score on the corresponding assessment. For example, from the partial CSV file listed in Q2, you would create the following partial dictionary.

```
{333 : {'Assignment1' : 67,
        'Assignment2' : 64,
        'Midterm1' : 81, ...},...}
```

Then write a function that takes the dictionary and returns a list of tuples - one for each student. Each tuple should be of the form `(id, avg_mark)` where `id` is an ID number and `avg_mark` is the average mark from all assessments of that student.

b) Write a function that takes in a filename, reads in the CSV file you created in Q2 from a file of that name, and returns a dictionary of the following form:

```
{id : {assessment : {assessment_part : mark}}}
```

This is a triply nested dictionary. The outer dictionary has keys corresponding to student IDs. For each student ID, the value is an inner dictionary whose keys are the 5 assessments and whose values are another dictionary. This inner-most dictionary should have keys corresponding to the parts of the assessment (i.e., A,B,C,D,E). For example, from the partial CSV file listed in Q2, you would create the following partial dictionary.

```
{333 : {'Assignment1' : {'A' : 67},
        'Assignment2' : {'A' : 44, 'B' : 20},
        'Midterm1' : {'A' : 44, 'B' : 20, 'C' : 18, 'D' : 22},
        ...},
...}
```

Then write a function that takes the dictionary and returns a list of tuples - one for each student. Each tuple should be of the form `(id, avg_mark)` where `id` is an ID number and `avg_mark` is the average mark from all assessments of that student. Note that this function is different from the one in part a) because the dictionary has a different form.

#4. Using the same data and set-up as Q3 do the following.

a) Redo Q3a using a `Student` class.

Create a `Student` class. The class should contain the student ID and a dictionary that is the same as the inner dictionary in Q3a. That is, each Student object should have a dictionary of the form: `{assessment : total_mark}`

For example:
```
{'Assignment1' : 67,'Assignment2' : 64,'Midterm1' : 81,
...}
```

Create a dictionary of the form: `{id : Student-object}`

Then write a function that takes the dictionary and returns a list of tuples - one for each student. Each tuple should be of the form `(id, avg_mark)` where `id` is an ID number and `avg_mark` is the average mark from all assessments of that student.

b) Redo Q3b using a `Student` class.

Create a `Student` class. The class should contain the student ID and a dictionary that is the same as the inner dictionary in Q3b. That is, each Student object should have a dictionary of the form: `{assessment : {assessment_part : mark}}`

For example:
```
{'Assignment1' : {'A' : 67},
'Assignment2' : {'A' : 44, 'B' : 20},
'Midterm1' : {'A' : 44, 'B' : 20, 'C' : 18, 'D' : 22},...}
```

Create a dictionary of the form: `{id : Student-object}`

Then write a function that takes the dictionary and returns a list of tuples - one for each student. Each tuple should be of the form `(id, avg_mark)` where `id` is an ID number and `avg_mark` is the average mark from all assessments of that student.

#5. This problem will give you some practice with algorithms.

The Collatz Conjecture (https://en.wikipedia.org/wiki/Collatz_conjecture) is the following conjecture about a sequence of numbers (c_1, c_2, ...). Set c_1 to any positive integer. The next term, c_{i+1}, in the sequence is found by using the following rules:

- if c_i is even, $c_{i+1} = c_i/2$
- if c_i is odd, $c_{i+1} = 3c_i + 1$

The Collatz Conjecture says that regardless of the starting number, the sequence will always reach 1.

Test the Collatz Conjecture as follows:

- write a function that takes a positive integer as input and follows the rules of the Collatz Conjecture until it reaches one. Return the number of elements in the sequence.
- write code to call the function you've just written for all numbers from 1 to 1000, print out each sequence length, and then print out the starting number of the maximum sequence found.