Files Problem

Exercise 1 (1 / 8)

Your task is to write a Python script which calculates and stores several aspects of grade scores of fictitious college students, given two different files containing the relevant information.

The things you will calculate include the **total number of available student scores**, the **sum of all these scores**, and the **average score** (sum / total = average).

Once you have made these calculations, you will **save** all the data to a new file – including the student's name and student ID. You will do this for each student in the students.txt file.

Exercise 1 (2 / 8)

Consider the data in the following file (students.txt):

Student ID, Name
123456, John Smith
654321, John Smith
246810, Trevor Smith
135791, Sally Smith

Exercise 1 (3 / 8)

The first row of the file can be considered to contain column headers called **attributes**. In this case, **Student ID** and **Name**. These attributes describe the data contained in each **tuple** (row) of data in the **relation** (table).

Notice, that the first attribute in this relation, **Student ID**, only makes sense if it is *unique*. Meaning – each entry in this column must be different than all the other entries in the column. It would not make sense to have two different students with the same **Student ID**. However, it *does* make sense to have *multiple* students with the same **Name** value. Thus, the **Student ID** attribute has a property that makes it *different* from the **Name** attribute. This property of necessary uniqueness makes the **Student ID** attribute the **Primary Key** in this case.

For more information on Primary Keys, please see the following article:
https://en.wikipedia.org/wiki/Primary_key

For more information on Relations, please see the following article:
https://en.wikipedia.org/wiki/Relation_(database)

Exercise 1 (4 / 8)

Consider the data in the following file (scores.txt):

Student ID, Assignment, Score
123456, Zany Text, 100
123456, Magic 9 Ball, 60
123456, Nim Grab, 80
123456, Dungeon Crawl, 78
123456, Ultimate TODO List, 90
654321, Zany Text, 48
654321, Ultimate TODO List, 82
654321, Nim Grab, 17
246810, Zany Text, 100
246810, Ultimate TODO List, 24
246810, Nim Grab, 98
135791, Zany Text, 84
135791, Ultimate TODO List, 3
135791, Nim Grab, 89
135791, Dungeon Crawl, 0

Exercise 1 (5 / 8)

Notice that the first row of data in this file contains this table's **attributes**, like the first file did. Notice further, that this relation contains a **Student ID** attribute like the first table did. In fact, the values used in the both files are the same.

However, notice that there can be more than one instance of a **Student ID** in the second table. Why is this? The answer is because Student ID in the second table is what is known as a **foreign key**. A **foreign key** is a **primary key** in *another table*. In this case, the table in students.txt. The primary key in this case, would be the *combination* of the **Student ID** and **Assignment** attributes. Each student can only have one instance of a *specific* assignment, but a single student can have multiple *different* assignments ('Zany Text,' 'Magic 9 Ball,' etc.).

We need *both* the **Student ID** attribute and the **Assignment** attribute to make each row unique. If we tried to use *just* the **Assignment** attribute as the primary key, there could only be one instance of 'Zany Text' in the entire table… only one instance of 'Magic 9 Ball' in the entire table… etc. Meaning – only one student in the entire class could do one of these assignments! However, by using more than one **attribute** in the **primary key**, multiple students can complete the same assignments, but each row will be unique. For example, doing things this way there will ever only be *one* row with **Student ID** 123456 *and* **Assignment** 'Zany Text.'

For more information about Foreign Keys, please see the following article:
https://en.wikipedia.org/wiki/Foreign_key

Exercise 1 (6 / 8)

By iterating through each **Student ID** in the students.txt table and comparing these unique values to the **Student IDs** found in the scores.txt table, we can calculate the **total number of scores** a student has, what the **sum of all those scores** is, and the **average of these scores** (average = sum / total). We can then append these values to a 'calculated grade' list, starting with the student's **Student ID** and **Name**.

This newly created 'calculated grade' list can then be **appended** to a 'master grades' list. This 'master grades' list will start with a sub-list of all the column attributes. This sub-list of attributes will then be followed by additional sub-lists containing all the data for all the students.

The 'master grades' list can initially appear in the code as follows:

```
grades = [["Student ID", "Name", "Total Scores", "Sum of All Scores", "Score Average"]]
```

Exercise 1 (7 / 8)

Once the ‘master grades’ list of lists has been constructed, we can finally **output** this list of lists to a file called **grades.txt**. The final output should appear *exactly* as it is on the next slide/ the example file **grades_example.txt** (provided).

Notice, all the entries in each tuple are delineated by commas and there are **no spaces before** or **after** each entry. The only spaces that appear in the output data are between a student’s first and last name, and between words in the column attributes.

Further, there is no ‘newline’ character after the very last line of data.

There should be a total of five (5) lines in the file – no more.

Exercise 1 (8 / 8)

Example Output (grades.txt):

Student ID,Name,Total Scores,Sum of All Scores,Score Average

123456,John Smith,5,408,81.6

654321,John Smith,3,147,49.0

246810,Trevor Smith,3,222,74.0

135791,Sally Smith,4,176,44.0

HINTS

- While this problem *may* seem daunting at first, thinking through how to solve it in a step-by-step manner will reveal that it is not so hard after all.
  - Ex: For each ____ you must do something to each ____ .
    - What sort of structure does the above sentence indicate? How would you make something that, for each item, something happens to all the other items in another set?
  - Determine an algorithm *before* you start coding.
- Think about all the programming components we have learned so far – how could you combine these concepts together?