Processing Data Using Files

This exercise will give you practice reading data from a file, processing the data, writing the results to another file, and defining and calling functions. You are are going to be developing an application that reads a file of exam scores, calculates a grade on a 4.0 scale, and writes the grades to a file. To practice incremental development, you will develop it in a series of steps. You may not use lists for this exercise.

Overview: Your program will read from the file formatted as follows: The first line will be a header of short exam names. The second line will give the maximum points for each exam. The remaining rows will contain a student PID and the points earned for each exam taken by the student. The file will use a fixed width format: the first column 7 characters wide and all exam columns 5 characters wide. We supply a sample exams.txt file, which contains 3 exams and scores for 5 students. But your program will work with any number of exams (columns) and students (rows).

Your program will determine the maximum points possible over all exams. For each student, it will calculate the grade earned and write the student PID and the grade earned to a new file, called grades.txt. It will use the following scale to calculate the grade earned.

90% – 100%: 4.0; 80% – 89%: 3.0; 70% – 79%: 2.0; 60% – 69%: 1.0; 0% – 59%: 0.0

For example, running your program in a directory that contains a copy of the exams.txt file from the course website will create a grades.txt file in the same directory with the contents shown to the right:

<table>
<thead>
<tr>
<th>Student ID</th>
<th>Score</th>
</tr>
</thead>
<tbody>
<tr>
<td>A100000</td>
<td>3.0</td>
</tr>
<tr>
<td>A314165</td>
<td>4.0</td>
</tr>
<tr>
<td>A650111</td>
<td>3.0</td>
</tr>
<tr>
<td>A203333</td>
<td>3.0</td>
</tr>
<tr>
<td>A333311</td>
<td>2.0</td>
</tr>
</tbody>
</table>

Program development: Develop your program in small steps, as described below.

Step 1: From the course website, download exams.txt and grader.py to a folder created for this week’s class. Open grader.py in Spyder. (It contains two function stubs, which you can ignore in this step. You will replace the stubs with full function definitions in later steps.)

Below the final comment in the file, start a main code section with code to

1. open exams.txt in read mode
2. read (and do nothing with) the first line of exams.txt
3. assign the second line of exams.txt to a variable
4. close exams.txt

Test your program by running it and then evaluating the variable (defined in sub step 3) in the shell.

Step 2: Replace the pass statement in the function stub for get_total with code to:

1. extract the scores from the input string passed in for score_str
2. convert the scores to int values
3. return the sum of the scores

(A design decision that you will need to make is what string the main program will pass to the get_total function.) Then, in the main code section, call get_total to calculate the total possible exam points and assign the result returned to a variable. Test your program by running it and then evaluating this new variable in the shell.
Step 3: Replace the pass statement in the function stub for get_grade with code to return the grade earned, assuming the first argument gives the total points a student earned and the second gives the maximum total exam points possible.

Test it by running the program and then calling get_grade in the shell with a variety of different arguments (use arguments that you can easily calculate the correct answers to).

Step 4: In the main code section, add code to iterate through the remaining lines and, for each:

1. call get_total to calculate the student’s total points
2. call get_grade to calculate the student’s grade
3. print the student’s PID followed by the grade earned

Test your program by running it and inspecting the output it displays.

Step 5: In the main code section, add code to:

1. open file grades.txt in write mode
2. print the student PIDs and grades earned to this file (modify the existing print statement so it prints to the file instead of the standard output display).
3. close file grades.txt

Step 6: Add a function that requests the name of the file containing the exam scores and, if the user enters a file that can be opened for input, returns a file object connected to the file which is open for input; or, if not, repeatedly prompts the user for the name of the exam scores file. Then modify the main code section to use your function instead of ‘hard coding’ the input file name.

Step 7: Modify the program to treat a missing score as 0 and a non-numeric score as invalidating the information in a row. If the information in a row containing a student’s scores is invalid, the program will write the student PID and an error message to grades.txt. If the information in the second row is invalid, the program will write an error message to grades.txt and terminate. For example, we show the contents of grades.txt produced by our solution when the user enters exams2.txt on the left and when the user enters exams3.txt on the right.

Step 8: Wrap the main code section in a function named main, which does not take any arguments. Add the call main() at the bottom of the file. Test that the program still works as expected.