CMSC 201

Section 60

Fall 2022

Project #2: Files, Lists and Dictionaries

Project Value: 100 points

Due Date:

Completed Project: Monday, November 21, before midnight

Overview:

The purpose of this project is to gain expertise in working with multi-dimensional lists and with getting user input through the use of menus. You must also demonstrate good design practices and proper coding/testing practices through compliance with CMSC 201 Python coding standards. When you have successfully completed this project you will have demonstrated mastery in the use of complex data structures in Python.

Design and constraints:

You are given many fewer constraints on the design of this project than of project 1, and thus you have much more freedom to express your own coding style. Be wary, though, because that gives you the freedom to do things wrong. Hopefully we've taught you better than that!

You must use proper design and modularity. Implement a function where you need it. If you do something multiple times, it should probably be a function. If you do something that's separate from the rest of the code, you should probably isolate it by putting it in a function. If you need to interact with the user or the system in some way, in different parts of the program, you should probably implement a helper function to take care of that.

Data:

You will be reading in three files of (fake) student records. (No real records would be used in this assignment. 😊)

The files are called Fall_2025.csv, Spring_2026.csv and Fall_2026.csv. These files contain student grade records from future semesters. Each file contains two header lines, which you will read in and then discard. After that comes some number of student records. There is one record per line. The format of each line is:

Student last name, student first name, student ID, project 1 grade, project 2 grade, project 3 grade, test 1 grade, test 2 grade, test 3 grade.

Each value is separated by a comma from the next value, which is why these files are called “Comma-separated value” or “csv” files.

Your program:

You will write a program that does the following:

Step 1:

Read in and process the three files. Each file should be put into a separate list – one list for Fall_2025, one list for Spring_2026, and one list for Fall_2026.

Each list should have one student record per element.

Each element will be a dictionary. That is, a record for a student is a dictionary; the record for a class is the list of all student dictionaries.

Each dictionary must initially have the following elements:

Key: “last name”. Value: the student last name; a string

Key: “first name”. Value: the student first name; a string

Key: “student ID” Value: the student ID; a string

Key: “P1” Value: the student’s score on Project 1; an integer

Key: “P2” Value: the student’s score on Project 2; an integer

Key: “P3” Value: the student’s score on Project 3; an integer

Key: “T1” Value: the student’s score on Test 1; an integer

Key: “T2” Value: the student’s score on Test 2; an integer

Key: “T3” Value: the student’s score on Test 3; an integer

As a warning: some files *may* contain extra newline characters will result is extra rows with no data being added to the end of your list. It is your responsibility as the programmer to detect this and delete the extra rows, so that you number of students in each class is accurate.

Once you have successfully built these three lists, you may move on to the next step.

Step 2:

Insert three new key/value pairs into each dictionary in each list

Add an entry for the sum of the student’s thee Project scores. You should add a Key of “Project_Total” whose value is an integer that is the sum of the three values associated with keys “P1”, “P2” and “P3.”

Add an entry for the student’s total score. You should add a Key of “Total_Score” whose value,k and integer, is the sum of all three programs and all three tests.

Now add a pair for the student’s letter grade. The Key should be “Letter_grade.” The value will be a string.

. The total possible number of points is 640. To calculate the letter grade, divide the student’s total points, as calculated and stored above, by 640. If this is greater than or equal to 90%, enter an “A” for the value of “Letter_grade.”. Greater than or equal to 80% but less than 90% gets a “B”.. Greater than or equal to 70% but less than 80% gets a “C”; greater than or equal to 60% but less than 70% gets a “D”; and less than 60% gets an “F”as the value for this key. .

Once you have the three lists with their new entries correctly inserted, you may move on to the next step.

Step 3:

The three lists you have built constitute a “database” of student grades that can be queried by users who need information about student grades. Now you need to provide an interface to let users actually query your database and do their jobs.

Do this by displaying on the screen a menu of choices that the users can make. Then read in the choices that they type in, and perform the request operation.

The menu does not have to look exactly like this, but it should be very similar. In other words, you should offer the same options, in the same order, but your messages do not have to exactly match the samples.

Welcome to the Student Grade Information Service

What would you like to do?

To search by student last name, enter “L”

To search by student ID, enter “S”

To exit the system, enter “Q”

You must then read in the input the user types. All input from the keyboard is a string, so there’s no reason that your program should ever crash here, regardless of what the user types!

We’ll be a little forgiving of users here. If the user types either an uppercase “L” or a lowercase “l” move to Step 4. If the user enters either an uppercase “S” or a lowercase “s” also move to Step 4. If the user enters either an uppercase “Q” or a lowercase “q” move to Step 8, where the program terminates.

If the user enters anything other than those 6 values listed above, display an error message that says

I’m sorry, but that <echo the user input> is not a valid choice.

Replace <echo the user input> with whatever the user actually typed.

And then re-display the menu. Keep doing this until the user enters a valid value.

Step 4:

Next you must find out which semester the student to be searched for attended the class. Regardless of whether the user entered “L”, “l”, ”S”, or “s”, display the following menu:

In which semester was the student enrolled?

Enter “F” for Fall 2025

Enter “S” for Spring 2026

Enter “T” for Fall 2026

Enter “U” if you do not know.

Again, validate the input. This is a string, so there's no excuse for crashing. If the user enters anything other than "F", "f", "S","s", "T", "t", "U" or "u", print the same error message you displayed in Step 3 (ooh, sounds like a helper function!!) and re-display the menu. Keep doing this until you get a valid input. When you have a valid input, you may move on to Step 5.

Step 5:

Decide what to do next. If the input from Step 3 was "L" or "l", go to Step 6. When you call the function in Step 6, pass the value you got in Step 4.

If the input you got from Step 3 was "S" or "s", go to Step 7. When you call the function in Step 7, pass the value you got in Step 4.

Step 6:

Here, you process a request for a student grade, search by student last name. Prompt the user to enter the last name of the student for whom the user is searching.

The user will enter a string. You will now search for this string in your database.

Your code was passed a value, read in Step 4, that indicates which semester the student was purportedly enrolled in the class. If the value from Step 4 was "F" or "f" search only in the 2D list created from the Fall_2025 file. If the value from Step 4 was "S" or "s" search only in the 2D list created from the Spring_2026 file. If the value from Step 4 was "T" or "t" search only in the 2D list created from the Fall_2026 file. If the value was "U" or "u" you must potentially search in all 3 lists.

A match is returned regardless of case; that is, "Stark", "STARK", "sTaRk" and "stark" should all match the same name.

If the string entered by the user is NOT in the correct column in the list in which you are searching, (not in Any list if Step 4 provided "U" or "u') display an error message:

> Sorry, <echo user input> was not enrolled in that semester.
>
> Please enter a student last name.
>
> Enter "Q" to return to the main menu.

If the student presses either "Q" or "q" return to Step 3. Otherwise, stay in this loop until the user enters a student last name that is actually in the appropriate list.

If the user enters a last name that matches exactly one student in the list, display the following message:

Student Last Name Student First Name Student ID Total Points Letter Grade

<print the values from columns 0, 1, 2, 10, and 11 for the record that matched>

What If the user enters a last name that matches more than one student? E.g., does the user mean Eddard Stark or Catelyn Stark? Rebecca Mannion or Rupert Mannion? If the last name appears more than once in the list of dictionaries, you must ask the user to choose among the matches. Display a message similar to :

Please indicate the student for whom you are searching:

Enter 1 for: <student 1 first name> <Student 1 last name>

Enter 2 for: <student 2 first name><Student 2 last name>

Include all students with the matching last name. Here you will not worry about validating input. Presume the user will enter a valid value. Display the record as above for the selected student. That is, display

Student Last Name Student First Name Student ID Total Points Letter Grade

<print the values from columns 0, 1, 2, 10, and 11 for the record that matched>

Once you have displayed a valid record, go back to Step 3

Step 7:

Here, you process a request for a student grade, searching by student ID. This is actually easier than the last name search since an ID can match at most one student record. There is no disambiguation needed.

Prompt the user to enter the student ID of the student for whom the user is searching.

The user will enter a string. You will now search for this string in your database.

Your code was passed a value, read in Step 4, that indicates which semester the student was purportedly enrolled in the class. If the value from Step 4 was “F” or “f” search only in the 2D list created from the Fall_2020 file. If the value from Step 4 was “S” or “s” search only in the 2D list created from the Spring_2021 file. If the value was “U” or “u” you must potentially search in BOTH lists.

Just as with last name, a match is returned regardless of case.

If the string entered by the user is NOT in the correct column in the list in which you are searching, (not in EITHER list if Step 4 provided “U” or “u’) display an error message:

Sorry, <echo user input> was not enrolled in that semester.

Please enter a student ID..

Enter “Q” to return to the main menu.

If the student presses either “Q” or “q” return to Step 3. Otherwise, stay in this loop until the user enters a student ID that is actually in the appropriate list.

Once the user has entered a student ID that matches student in the list, display the following message:

Student Last Name Student First Name Student ID Total Points Letter Grade

<print the values from columns 0, 1, 2, 10, and 11 for the record that matched>

When you are finished displaying this message, go to Step 3.

Step 8:

Exit the program. Here, the user has chosen “Q” or “q” to exit the program. Print out a farewell message similar to the following. Again, you don’t have to use these exact words if you think you can be more poetic, but your message should convey the same idea.

Thank you for using the interactive student grade request service.

Hope to see you again soon.

Shutting down now.

Grading:

The 100 points for this project will be assigned as follows:

Factor	Points	Comments
Program Design	10	Program design follows good practice and is described by the student. Functions implement clear, logically unique parts of the program. Code is not repeated – e.g., the student does not have three separate code segments to read in the three

		files
Coding Standards	10	Particular emphasis on: - use of CONSTANTs, - no use of global variables in functions; - snake case for variable names; - meaningful function and variable names
Program Structure	40	***Main program structure.*** The main program should consist of setup and initialization code, and other than that mostly function calls ***Functions:*** At a minimum, the following should be implemented in functions: - reading in files. The students should have a function that takes a filename as a parameter, reads in the file, and returns the contents of the file. This function can either return the string containing the file contents less the two header lines in each file; OR it can call the function that creates the list; takes the list from that function; and then returns the list to the main program. - creating the list of dictionaries for each file. This should optimally be done in a separate function. It can be called by the function that reads files; OR it can be called by the main program. - Insert the new key/value pairs into the dictionaries. This can be done in a single function; though separate functions is acceptable. - Displaying the user menus. The students should call one or more functions to display the user menu; get the user input; and display error messages or further results. Additional functions are acceptable if they make sense in the overall structure of the program.

		Comments:. There should be a program header comment. Each function should be commented properly. Lines of code that collectively implement some operation should be set off by whitespace and commented.
Correct Output	40	The program must produce correct output for a variety of test cases, including: - Student not found - Student found by last name, including resolving conflicts - Student found by ID number - Searching in a specific semester - Searching over all semesters - Quitting the program