CIS 5357 Fall 2020

Programming Assignment # 5

(20 points)

**Due Date: Before 11:59 pm on Friday, October 9, 2020**

**Caution:**

**The instructor has found many submissions with duplicate or similar code. The instructor expects individual effort on each and every submission. Assignments and exams assigned in this course are neither group projects nor any kind of group activity or collaboration are sanctioned. Such activities will be treated per the Academic Dishonesty policy as stated in the course syllabus. Each submission will be closely examined for plagiarism**. **A slightest hint of duplicate or similar code will be examined for academic integrity violation. Such submissions will automatically receive a grade of zero and reported to the Graduate College for further disciplinary action. To avoid such scrutiny, please do your own work. Please consult your instructor if you any questions or need clarification about assignments and examinations.**

1. **Requirements for Assignment 5:**
2. Name your Jupyter Notebook ‘YourName-Assignment5.ipynb’
3. Include your name and submission date as level 2 headings in the first cell of the notebook.
4. Insert a markdown cell with level 3 heading “Grading Comments – Totality of all points noted below resulted in a reduction of xx Points”
5. Insert a markdown cell with level 3 heading for the Program Name and its objective
6. The program design must be broken down into functions, with each function performing a specified task. The main driver program will also be coded as main() function. (see Hierarchy and IPO charts later)
7. Each function must include an appropriately written docstring.
8. Include functions defined for each program in their individual cells (in the same order as the IPOs described later) but together as a group.
9. Main() function should be the last one defined.
10. Upload your source code file (.ipynb) to the Assignments section of Canvas ***BEFORE*** 11:59 pm on Friday, October 9, 2020. **No other files need to be uploaded.**
11. Use the following paths for saving all text/CSV files being used in the program:
    1. PC: ‘C:/users/username/CIS5357Fall2020’
    2. Mac: ‘/Users/username/CISFall5357Fall2020’
12. **Specifications:**

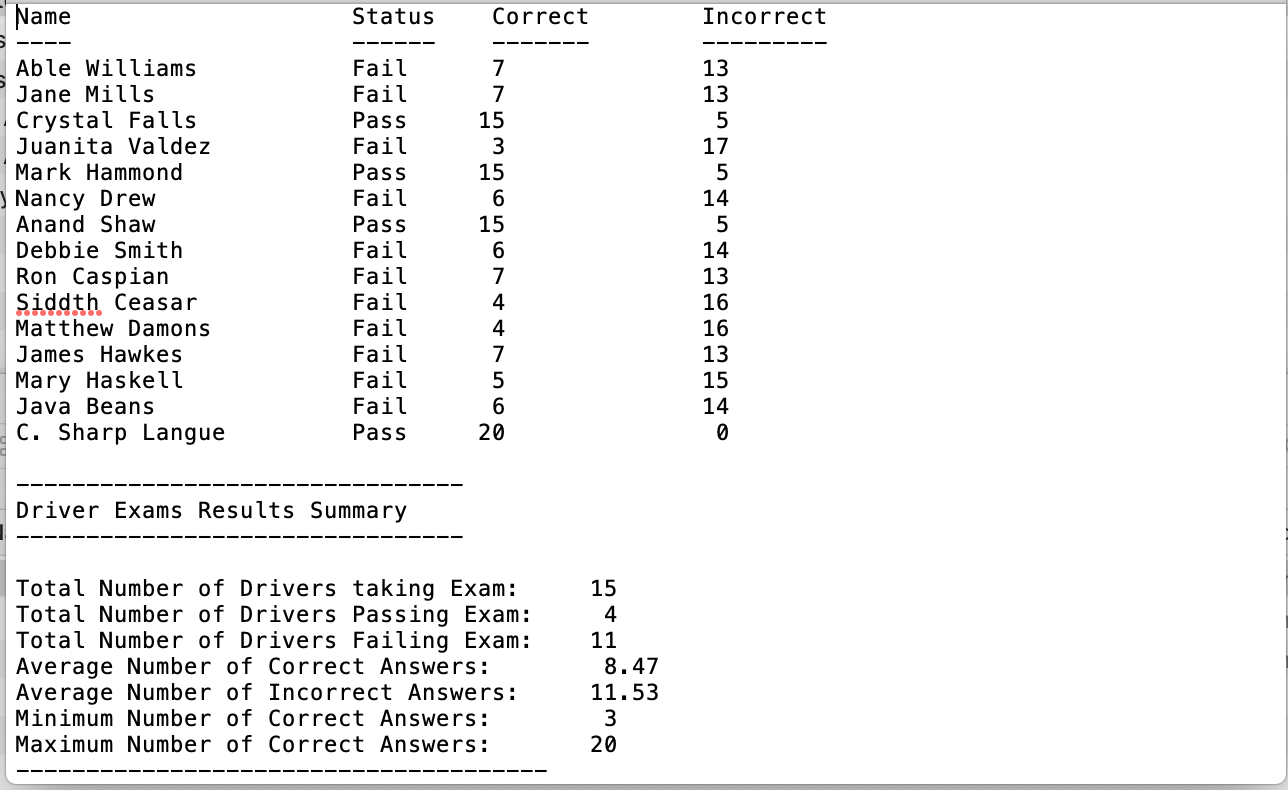
The local driver’s license office has asked you to create an application that grades the written portion of the driver’s license exam. The written portion has 20 multiple-choice questions. The correct answers to the 20 questions are listed below:

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Ques # & Correct Ans | | Ques # & Correct Ans | | Ques # & Correct Ans | | Ques # & Correct Ans | |
| 1. | A | 6. | B | 11. | A | 16. | C |
| 2. | C | 7. | C | 12. | D | 17. | B |
| 3. | A | 8. | A | 13. | C | 18. | B |
| 4. | A | 9. | C | 14. | A | 19. | D |
| 5. | D | 10. | B | 15. | D | 20. | A |

Your program should store these correct answers in a **tuple**, named **correct\_answers**. Remember, tuple is a sequence data structure. So, the first answer listed is associated with the first question, second answer to the second question, and so on. It is also immutable, meaning that once defined, the tuple cannot be changed in any way. Because the answers to the multiple-choice questions are not expected to change, tuple is the right data structure to use for this purpose.

The program should read the student’s answers for each of the 20 questions from a csv file named **PotentialDrivers.csv.** Each line in this file contains a record for one student driver’s answers. It includes driver’s name followed by answers to 20 questions, with each value separated by comma. This file can have any number of exams results for potential drivers. During any given time slot, there may be several potential drivers taking the written portion of the driver’s exam and so the number of records in the PotentialDrivers.csv file may vary.

For each driver, determine if the driver passed or failed the written portion. A driver must correctly answer a minimum of 15 of the 20 questions to pass the written portion. This will require comparing each answer read in to the corresponding answer in the tuple to see if there is a match and keeping a running count of correct and incorrect answers. Write the driver’s name, whether or not the driver passed or failed the test, number of questions answered correctly and the number of questions answered incorrectly to a text file named **DriversExamResults**.**txt**. This is an output file and will be used as a report file. Therefore, each value on a line must be separated by as many ‘tab’s (not comma) as are necessary to achieve properly aligned columns. The first line in this text file must be a header row. Once all drivers in the input file have been processed, write the total number of drivers who took the exam; the number of drivers who passed the exam, the number of drivers who did not pass the exam; the average number of questions answered correctly and incorrectly, minimum and maximum number of questions answered correctly based on all exams taken. This output file is illustrated below. The summary results should also be printed to the console via the print\_summary() function.



|  |  |  |
| --- | --- | --- |
| IPO – initialization() function | | |
| Input (Parameters) | Process | Outputs (returns) |
| None | Define a tuple, correct\_answers, to hold correct answers. | correct\_answers tuple |

|  |  |  |
| --- | --- | --- |
| IPO – read\_driver\_data() function | | |
| Input (Parameters) | Process | Outputs (returns) |
| correct\_answers tuple | Define variables and constants that you will to support this function. | **graded\_drivers\_data** list |
|  | Open PotentialDrivers.csv file in read mode |  |
|  | Load the driver’s data from the CSV file into a list named **driver\_responses** |  |
|  | Invoke process\_driver\_data function, passing in the correct\_answers tuple and driver\_responses list. Receive **graded\_drivers\_data** list from the process\_driver\_data function. |  |
|  | Return the **graded\_drivers\_data** list |  |
|  | | |
| **IPO – process\_driver\_data () function** | | |
| Input (Parameters) | Process | Outputs (returns) |
| correct\_answers tuple  driver\_responses list | Define any constants and variables you may need to support processing in this function. | **graded\_drivers\_data** list |
|  | For each driver in the driver\_responses list:   1. Extract driver’s name 2. For each question response in driver’s record, compare question’s response with the corresponding value in the correct\_answers tuple 3. If the driver’s response matches with the corresponding value in the correct\_answers tuple, add 1 to driver’s correct variable, else add 1 to driver’s incorrect variable 4. Once all responses have been checked, compare the total number of correct answers to the minimum required to pass the test. Set the status as ‘Passed” or “Failed” accordingly 5. Create a comma-separated driver’s result string comprising of driver’s name, number of correct answers, number of incorrect answers, and status. 6. Add the driver result string to a **graded\_drivers\_data** list.   Return graded\_drivers\_data list |  |

|  |  |  |
| --- | --- | --- |
| IPO – **write\_results\_file** function | | |
| Input (Parameters) | Process | Outputs (returns) |
| graded\_drivers\_data list  Total number of potential drivers  Total number passing the test  Total number failing the test  Average number correct answers  Average number of incorrect answers  Maximum number of correct answers  Minimum number of correct answers | For Each record in the graded\_drivers\_data list,   1. extract individual fields on the ‘comma’ separation. 2. Using the extracted fields, create another driver data string such that each driver’s field is separated by tabs so that the string has the appearance of the record shown in the sample output. This way, data in all records are properly aligned in columns. 3. Write the formatted string from (b) to the text file. Don’t forget that the first line written to this file is the header row of column titles.   Write the summary results per the sample output shown in specs. | None |
|  | Close the opened text file |  |

|  |  |  |
| --- | --- | --- |
| IPO – **compute\_summary()** function | | |
| Input (Parameters) | Process | Outputs (returns) |
| graded\_drivers\_data list | Define variables needed to hold summary results  Based on the data contained in the graded\_drivers\_data list:   1. Compute total number of drivers appearing for the exam. 2. Compute the total number of drivers passing the exam 3. Compute the total number of drivers failing the exam 4. Compute average number of correct answers per driver 5. Compute the average number of incorrect answers per driver 6. Compute the maximum number of correct answers 7. Compute the Minimum number of correct answers 8. All computed values must be ROUNDED to 2 decimal places.   Return all of the above computed summary results | Total number of potential drivers  Total number passing the test  Total number failing the test  Average number correct answers  Average number of incorrect answers  Maximum number of correct answers  Minimum number of correct answers |

|  |  |  |
| --- | --- | --- |
| IPO – **print\_summary()**  function | | |
| Input (Parameters) | Process | Outputs (returns) |
| Total number of potential drivers  Total number passing the test  Total number failing the test  Average number correct answers  Average number of incorrect answers  Maximum number of correct answers  Minimum number of correct answers | Print message “Summary of Drivers’ Exam Results”  Print each of the following with appropriate message. Format all numeric values to display to 2 decimal places, properly aligned:  Total Number of Drivers taking the exam  Total number passing test  Total number failing test  Average number of correct answers  Average number of incorrect answers  Maximum number of correct answers  Minimum number of correct answers | None |
|  | Print normal end of program message. |  |

|  |  |  |
| --- | --- | --- |
| IPO – **main()**  function | | |
| Input (Parameters) | Process | Outputs (returns) |
| None | Define any variables and constants that may need to receive data from and send data to other functions.  Invoke initialization() function, receiving the correct\_answers tuple  Invoke read\_driver\_data function, passing in the correct\_answers tuple and receiving graded\_drivers\_data list  Invoke compute\_summary() function, passing in the graded\_drivers\_data and receiving all computed summary results  Invoke write\_results\_file function, passing in the graded\_drivers\_data list and all computed summary results received above  Invoke print\_summary() function, passing in all computed summary results. | None |
|  | Print normal end of program message. |  |