Assignment-6&7
PROG8421-24-Sec2
Spring 2024
Due Date: Tuesday, July 16, 2024

Important Notes:

- This is a group assignment with maximum 4 students in a group
- Make sure you follow the specifications for each question.

Question-1. Modify the Miles Per Gallon program so it stores the data for each calculation in a CSV file.

1. In PyCharm, open the **mpg_write.py** file (available in eConestoga with this assignment)
2. Review the code and run the program so you remember how it works.
3. Enhance the program so it stores the data for each calculation, or trip, in a two-dimensional list. For each calculation, these values should be put in the list: miles driven, gallons of gas used, and the calculated MPG value.
4. Enhance the program so it saves the data in the list to a file named trips.csv when the user wants to exit from the program.
5. Test the program to make sure it works. To do that, you can open the CSV file with a spreadsheet program like Excel.
6. Take a screen shot of the spreadsheet

Question-2. Modify the Miles Per Gallon program so it adds to the data in the file that you created in Question-1 above. This program should display the data for each trip that's entered in a CSV file as shown below:

```
The Miles Per Gallon program
Distance        Gallons         MPG
225             17              13.24
1374            64              21.47
2514            79              31.82


Enter miles driven:      274
Enter gallons of gas:    18.5
Miles Per Gallon:        14.81

Distance        Gallons         MPG
225             17              13.24
1374            64              21.47
2514            79              31.82
274.0           18.5            14.81

More entries? (y or n):
```

1. In Pycharm, open the **mpg.py** file (available on eConestoga with this assignment)
2. Add a write_trips() function that writes the data from a two-dimensional list named trips that's passed to it as an argument. This list contains the data for each trip that's entered, and it should be written to a CSV file named trips.csv. As the console above shows, the data for each trip consists of miles driven, gallons of gas used, and the calculated MPG value.
3. Add a read_trips() function that reads the data from the trips.csv file and returns the data for the trips in a two-dimensional list named trips.
4. Add a list_trips() function that displays the data in the trips list on the console, as shown above.

5. Enhance the main() function so it starts by getting the data from the CSV file and listing it as shown above.
6. Enhance the main() function so it adds the last trip that's entered to the trips list after it calculates the MPG. Then, display the data for the updated trips list.
7. Test all aspects of the program until you're sure that it works correctly.

Question-3. Modify the programs that you created Question-1 and Question-2 so they create and use a binary file instead of a CSV file. Otherwise, everything should work the same.

Modify the CSV version of the write program

1. Open the **mpg_write.py** file that you created in exercise Question-l. Then, save it as **mpg_write_binary.py** in the same directory.
2. Modify this program so it saves the list as a binary file instead of a CSV file. The file should be named trips.bin.
3. Test the program to make sure it works. To do that, add statements that read the file at the end of the program and display the list that has been read.

Modify the CSV version of the trip program

4. Open the **mpg.py** file that you created in exercise Question-2. Then, save it as **mpg_binary.py.**
5. Modify this program so it works the same as it did with the CSV file.
6. Test this program to make sure it works.

Question-4. Modify the Future Value program so the user can't cause the program to crash by entering an invalid int or float value.

1. In Pycharm, open the **future_value.py** (available in eConestoga with this assignment)
2. Review the code and study the get_number() and get_integer() functions. Note that they receive three arguments: the prompt for a user entry, the low value that the entry must be greater than, and the high value that the entry must be less than or equal to. Then, review the calling statements in the main() function and note how these functions are used.
3. Test the program. Note that you can cause the program to crash by entering values that can't be converted to float and int values.
4. Add exception handling to the get_number() and get_integer() functions so the user has to enter valid float and int values. Then, test these changes to make sure the exception handling and the data validation work correctly.

Question-5. In this exercise, you'll modify the Movies List 2.0 program so it does more exception handling. You'll also use a raise statement to test for exceptions.

1. In Pycharm, open **movies2.py** (available in eConestoga with this assignment)
2. Add data validation to the **add_movie()** function so the year entry is a valid integer that's greater than zero. Then, test this change.
3. Modify the **write_movies()** function so it also handles any OSError exceptions by displaying the class name and error message of the exception object and exiting the program
4. Test this by using a raise statement in the try block that raises a BlockingIOError. This is one of the child classes of the OSError. Then, comment out the raise statement.
5. In the read_movies() function, comment out the two statements in the except clause for the FileNotFoundError. Instead, use this except clause to return the empty movies list that's initialized in the try block. This should cause