# PROG8421 - Programming for Big Data

**Assignment 6&7**

**Group 1 Members:**

|  |  |
| --- | --- |
| **Name: Sudhan Shrestha** | **Name: Jose Encarnacion** |
| **Student ID: 8889436** | **Student ID: 8982860** |

**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.**
2. **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.**
3. **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.**
4. **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**

**Code**

import csv

from typing import \*

# from os import system

# 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

def get\_miles\_driven()->float:

"""

This function get data from user miles driven

Parameters:

None

Returns:

float

"""

while True:

try:

miles\_driven:float = float(input("Enter miles driven:\t"))

if miles\_driven > 0:

return miles\_driven

else:

print("Entry must be greater than zero. Please try again.\n")

except ValueError:

print("Invalid input. Please enter a numeric value.\n")

def get\_gallons\_used()->float:

"""

This function get data from user gallons used

Parameters:

None

Returns:

float

"""

while True:

try:

gallons\_used:float = float(input("Enter gallons of gas:\t"))

if gallons\_used > 0:

return gallons\_used

else:

print("Entry must be greater than zero. Please try again.\n")

except ValueError:

print("Invalid input. Please enter a numeric value.\n")

def save\_trips\_to\_csv(trips:List)->None:

"""

This function save data to a csv file named 'trips.csv'

Parameters:

trips (list of float)

Returns:

None

"""

with open('trips.csv', mode='w', newline='') as file:

writer = csv.writer(file)

writer.writerow(["Miles Driven", "Gallons Used", "MPG"])

writer.writerows(trips)

def main():

print()

print("The Miles Per Gallon program\n")

trips = [] # List to store trip data

while True:

miles\_driven = get\_miles\_driven()

gallons\_used = get\_gallons\_used()

mpg = round((miles\_driven / gallons\_used), 2)

print(f"Miles Per Gallon:\t{mpg}\n")

# Store the trip data

trips.append([miles\_driven, gallons\_used, mpg])

while True: # Validation for more entries

more = input("More entries? (y or n): ").strip().lower()

if more in ["y", "yes"]:

break

elif more in ["n", "no"]:

# Save the trip data to a CSV file

save\_trips\_to\_csv(trips)

print("Bye! The trip data has been saved to trips.csv.")

return

else:

print("Invalid input. Please enter 'y' or 'n'.\n")

if \_\_name\_\_ == "\_\_main\_\_":

main()

**Comment**

The get\_miles\_driven() function is used to get the miles driven from the user, get\_gallon\_used() is used to get the gallon of gas used, the function save\_trips\_to\_csv() get the list of values of miles, gallon and mpg that was calculated and saves it into a *trips.csv* files in proper order. Input validation is done for each user input along with yes or no question during the prompt if user wants to continue.

**Screenshot**

**A screen shot of a computer

Description automatically generated**

**Data saved in csv file**

**A screenshot of a computer

Description automatically generated**

**Input validation**

**A screen shot of a computer

Description automatically generated**

**A screen shot of a computer

Description automatically generated**

**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:**

**A number of gas prices

Description automatically generated with medium confidence**

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**

**Code**

#!/env/bin python3

import csv

from typing import \*

from os import system

def get\_positive\_float(prompt: str)->float:

"""

This function get data from user and convert to float value

Parameters:

prompt: str

Returns:

float

"""

while True:

try:

value = float(input(prompt))

if value > 0:

return value

else:

print("Entry must be greater than zero. Please try again.\n")

except ValueError:

print("Invalid input. Please enter a numeric value.\n")

def write\_trips(trips:List)->None:

"""

This function write data in a csv file named 'trips.csv'

Parameters:

trips:List

Returns:

None

"""

with open('trips.csv', 'w', newline='') as file:

writer = csv.writer(file)

writer.writerow(["Distance", "Gallons", "MPG"])

writer.writerows(trips)

def read\_trips()->List:

"""

This function read data from csv file named 'trips.csv'

Parameters:

trips:List

Returns:

List

"""

trips = []

try:

with open('trips.csv', 'r') as file:

reader = csv.reader(file)

next(reader) # Skip the header

for row in reader:

trips.append([float(row[0]), float(row[1]), float(row[2])])

except FileNotFoundError:

pass

return trips

def list\_trips(trips:List)->None:

"""

This function show a list for console

Parameters:

trips:List

Returns:

None

"""

print("Distance Gallons MPG")

for trip in trips:

print(f"{trip[0]} {trip[1]} {trip[2]}")

def main():

# display a welcome message

print("The Miles Per Gallon program")

print()

trips = read\_trips()

if trips:

list\_trips(trips)

print()

while True:

miles\_driven = get\_positive\_float("Enter miles driven:\t")

gallons\_used = get\_positive\_float("Enter gallons of gas:\t")

mpg = round((miles\_driven / gallons\_used), 2)

print(f"Miles Per Gallon:\t{mpg}")

print()

trips.append([miles\_driven, gallons\_used, mpg])

list\_trips(trips)

print()

while True:

more = input("More entries? (y or n): ").strip().lower()

if more in ["y", "yes"]:

break

elif more in ["n", "no"]:

write\_trips(trips)

print("Bye!")

return # Exit the main function

else:

print("Invalid input. Please try again.\n")

if \_\_name\_\_ == "\_\_main\_\_":

main()

**Comment**

The function get\_postiive\_float() get the user entered miles and gallon and converts it into float data type further more data validation is done with addition to checking if the value entered is greater than 0. The function write\_trips() writes the data in a csv file named *trips.csv*. read\_trips() reads the data from the csv file to show the entered data in the program. The function list\_trips() displays the data of trips that the users have entered. The program was enhance as the main() function strats with reading the data from the csv file using the function read\_trips() and displays it using list\_trips() also the last trip that’s entered to the trips list after the calculation of mgp is completed.

**Screenshot**

**A screen shot of a computer

Description automatically generated**

**The CSV file**

**A screenshot of a computer

Description automatically generated**

**Input validation**

**A screen shot of a computer

Description automatically generated**

**A screenshot of a computer

Description automatically generated**

**A screen shot of a computer

Description automatically generated**

**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.**

**Code**

import pickle

from typing import List

def get\_miles\_driven() -> float:

"""

This function get data from user miles driven

Parameters:

None

Returns:

float

"""

while True:

try:

miles\_driven = float(input("Enter miles driven:\t"))

if miles\_driven > 0:

return miles\_driven

else:

print("Entry must be greater than zero. Please try again.\n")

except ValueError:

print("Invalid input. Please enter a numeric value.\n")

def get\_gallons\_used() -> float:

"""

This function get data from user gallons used

Parameters:

None

Returns:

float

"""

while True:

try:

gallons\_used = float(input("Enter gallons of gas:\t"))

if gallons\_used > 0:

return gallons\_used

else:

print("Entry must be greater than zero. Please try again.\n")

except ValueError:

print("Invalid input. Please enter a numeric value.\n")

def save\_trips\_to\_binary(trips: List) -> None:

'''

Save data to a binary file named 'trips.bin'

Parameters:

trips (list of float)

Returns:

None

'''

with open('trips.bin', 'wb') as file:

pickle.dump(trips, file)

def read\_trips\_from\_binary() -> List:

'''

Read data from a binary file named 'trips.bin'

Parameters:

None

Returns:

list of float

'''

try:

with open('trips.bin', 'rb') as file:

trips\_from\_file = pickle.load(file)

print("\nTrips read from binary file:")

for trip in trips\_from\_file:

print(f"Miles Driven: {trip[0]}, Gallons Used: {trip[1]}, MPG: {trip[2]}")

except (FileNotFoundError, EOFError):

return []

def main():

print()

print("The Miles Per Gallon program\n")

trips = [] # list of trips

while True:

miles\_driven = get\_miles\_driven()

gallons\_used = get\_gallons\_used()

mpg = round((miles\_driven / gallons\_used), 2)

print(f"Miles Per Gallon:\t{mpg}\n")

trips.append([miles\_driven, gallons\_used, mpg])

while True:

more = input("More entries? (y or n): ").strip().lower()

if more in ["y", "yes"]:

break

elif more in ["n", "no"]:

save\_trips\_to\_binary(trips)

read\_trips\_from\_binary() # read data from binary file

print("Bye!")

return

else:

print("Invalid input. Please enter 'y' or 'n'.\n")

if \_\_name\_\_ == "\_\_main\_\_":

main()

**Comment**

The code from Question 1 was modified as instructed, modification to the function save\_trips\_to\_csv() which used to save the files in csv was modified to save\_trips\_to\_binary() which saves the files in *trips.bin* as a binary files instead using the *pickle* library. The function read\_trips\_from\_binary() reads the binary file *trips.bin* and reads the file and displays all the data and shows at the end of execution of the program as instructed.

**Screenshot**

**A screen shot of a computer

Description automatically generated**

**Input Validation**

**A screenshot of a computer

Description automatically generated**

**Modify the CSV version of the trip program**

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

**Code**

import pickle

from typing import List

def get\_positive\_float(prompt: str) -> float:

"""

This function get data from user and convert to float value

Parameters:

prompt: str

Returns:

float

"""

while True:

try:

value = float(input(prompt))

if value > 0:

return value

else:

print("Entry must be greater than zero. Please try again.\n")

except ValueError:

print("Invalid input. Please enter a numeric value.\n")

def write\_trips(trips: List) -> None:

'''

This function write data in a binary file named 'trips.bin'

Parameters:

trips (list of float)

Returns:

None

'''

with open('trips.bin', 'wb') as file:

pickle.dump(trips, file)

def read\_trips() -> List:

'''

This function read data from a binary file named 'trips.bin'

Parameters:

None

Returns:

list of float

'''

try:

with open('trips.bin', 'rb') as file:

return pickle.load(file)

except (FileNotFoundError, EOFError):

return []

def list\_trips(trips: List) -> None:

"""

This function show a list for console

Parameters:

trips:List

Returns:

None

"""

print("Distance Gallons MPG")

for trip in trips:

print(f"{trip[0]} {trip[1]} {trip[2]}")

def main():

print("The Miles Per Gallon program\n")

trips = read\_trips()

if trips:

list\_trips(trips)

print()

while True:

miles\_driven = get\_positive\_float("Enter miles driven:\t")

gallons\_used = get\_positive\_float("Enter gallons of gas:\t")

mpg = round((miles\_driven / gallons\_used), 2)

print(f"Miles Per Gallon:\t{mpg}\n")

trips.append([miles\_driven, gallons\_used, mpg])

list\_trips(trips)

print()

while True:

more = input("More entries? (y or n): ").strip().lower()

if more in ["y", "yes"]:

break

elif more in ["n", "no"]:

write\_trips(trips)

print("Bye!")

return

else:

print("Invalid input. Please try again.\n")

if \_\_name\_\_ == "\_\_main\_\_":

main()

**Comment**

Similarly, the code of Question 2 was modified as well, the write\_trips() function now saves the files in binary format as *trips.bin* and the read\_trips() function here reads the *trips.bin* file to display the output.

**Screenshot**

**A screen shot of a computer

Description automatically generated**

**Input Validation**

**A screenshot of a computer

Description automatically generated**

**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.**

**Code**

#!/usr/bin/env python3

def get\_number(prompt, low, high):

'''

Get a float value from the user between low and high

Parameters:

prompt:str

low: float

hight: float

Returns:

number :float

'''

while True:

try:

number = float(input(prompt))

if number > low and number <= high:

is\_valid = True

return number

else:

print(f"Entry must be greater than {low} "

f"and less than or equal to {high}.")

continue

except ValueError:

print("Invalid input. Please enter a float value.")

def get\_integer(prompt, low, high):

'''

Get an integer value from the user between low and high

Parameters:

prompt:str

low: float

hight: float

Returns:

number:float

'''

while True:

try:

number = int(input(prompt))

if number > low and number <= high:

is\_valid = True

return number

else:

print(f"Entry must be greater than {low} "

f"and less than or equal to {high}.")

continue

except ValueError:

print("Invalid input. Please enter an integer value.")

def calculate\_future\_value(monthly\_investment, yearly\_interest, years):

'''

Calcuate the future value of an investment

Parameters:

monthly\_investment: float

yearly\_interest: float

years: int

Returns:

future\_value: float

'''

# convert yearly values to monthly values

monthly\_interest\_rate = yearly\_interest / 12 / 100

months = years \* 12

# calculate future value

future\_value = 0.0

for i in range(months):

future\_value += monthly\_investment

monthly\_interest = future\_value \* monthly\_interest\_rate

future\_value += monthly\_interest

return future\_value

def main():

while True:

# get input from the user

print()

print("Future Value Calculator\n")

monthly\_investment = get\_number("Enter monthly investment:\t", 0, 1000)

yearly\_interest\_rate = get\_number("Enter yearly interest rate:\t", 0, 15)

years = get\_integer("Enter number of years:\t\t", 0, 50)

# get and display future value

future\_value = calculate\_future\_value(

monthly\_investment, yearly\_interest\_rate, years)

print()

print(f"Future value:\t\t\t{round(future\_value, 2)}")

print()

# see if the user wants to continue

while True:

choice = input("Continue? (y/n): ")

if choice.lower() == "y":

break

elif choice.lower() == "n":

print("Bye!")

exit()

else:

print("Invalid input. Please enter 'y' or 'n'.")

if \_\_name\_\_ == "\_\_main\_\_":

main()

**Comment**

The get\_number() and get\_integer() functions both were modified to include a try and except block to catch any errors that may arise if user try to input invalid values for both of these functions. Furthermore, validation on the prompt the user to continue was also included to check if proper values ‘y’ or ‘n’ were entered.

**Screenshot**

**A screenshot of a computer screen

Description automatically generated**

**Validation of each input**

**A screenshot of a computer screen

Description automatically generated**

**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**

**Code**

#!/env/bin python3

from typing import \*

from os import system

import csv

import sys

FILENAME = "movies.csv"

def exit\_program()->None:

print("Terminating program.")

sys.exit()

def read\_movies():

try:

movies = []

with open(FILENAME, newline="") as file:

reader = csv.reader(file)

for row in reader:

movies.append(row)

return movies

except FileNotFoundError as e:

print(f"Could not find {FILENAME} file.")

return movies # Return empty list if file is not found

except Exception as e:

print(type(e), e)

exit\_program()

def write\_movies(movies)->None:

try:

with open(FILENAME, "w", newline="") as file:

writer = csv.writer(file)

writer.writerows(movies)

# Raise a BlockingIOError to test the exception handling

# raise BlockingIOError("Testing exception handling") # Uncomment to test

except OSError as e:

print(f"OSError: {type(e).\_\_name\_\_} - {e}")

exit\_program()

except Exception as e:

print(type(e), e)

exit\_program()

def list\_movies(movies):

for i, movie in enumerate(movies, start=1):

print(f"{i}. {movie[0]} ({movie[1]})")

print()

def add\_movie(movies):

name = input("Name: ")

while True:

year = input("Year: ")

try:

year = int(year)

if year <= 0:

raise ValueError("Year must be greater than zero.")

break

except ValueError as e:

print(f"Invalid year: {e}. Please try again.")

movie = [name, year]

movies.append(movie)

write\_movies(movies)

print(f"{name} was added.\n")

def delete\_movie(movies)->None:

while True:

try:

number = int(input("Number: "))

except ValueError:

print("Invalid integer. Please try again.")

continue

if number < 1 or number > len(movies):

print("There is no movie with that number. Please try again.")

else:

break

movie = movies.pop(number - 1)

write\_movies(movies)

print(f"{movie[0]} was deleted.\n")

def display\_menu()->None:

print()

print("The Movie List program")

print()

print("COMMAND MENU")

print("list - List all movies")

print("add - Add a movie")

print("del - Delete a movie")

print("exit - Exit program")

print()

def main():

display\_menu()

movies = read\_movies()

while True:

command = input("Command: ")

if command.lower() == "list":

list\_movies(movies)

elif command.lower() == "add":

add\_movie(movies)

elif command.lower() == "del":

delete\_movie(movies)

elif command.lower() == "exit":

break

else:

print("Not a valid command. Please try again.\n")

print("Bye!")

if \_\_name\_\_ == "\_\_main\_\_":

main()

**Comment**

The add\_movie() was modified to include a validation using try and expect block furthermore validation was done to check if the year was greater than zero for a movie to be added in case of error the error type with proper error message is prompted to the user. The function write\_movies() was modified to include OSError along with Exception error along with a message of error being prompted to the user, also to test and error a BlockingIOError was included as a raise statement this code line is comment by default, to test this error we need to uncomment the code first and run the program. In the read\_movies() function we return a empty movies list if no *movies.csv* file found.

**Screenshot**

**A computer screen shot of a computer program

Description automatically generated**

**Validation and Exception**

**add\_movies() Error Handling**

**A screen shot of a computer program

Description automatically generated**

A screen shot of a computer

Description automatically generated

**write\_movies() Error Handling**

**OSERROR**

A screen shot of a computer

Description automatically generated

*To replicate this error, we need to have the csv file open during program execution causing this error during writing the movie.*

**Testing BlockingIOError**

***A screen shot of a computer program

Description automatically generated***

*To simulate this error again, we need to uncomment the code line, then it will execute when we try to add a movie.*

**Read\_movies FileNotFoundError:**

**A screen shot of a computer

Description automatically generated**

*When we delete the movies.cvs and try to run the program it prints the error message and when we try to list the movies it returns the empty movies. If we proceed to add new movies it creates a new movies.csv file and adds the data to it.*

**Additional Command Input Validation:**

A screen shot of a computer

Description automatically generated