Practical Project\_1

Course: CST8333\_350

Due September 22, 2024

Professor name: Stanley Pieda

Student Name: Mohammad Dellawari 041094581

Table of Contents

[**Evidence of Learning:** 3](#_Toc177476300)

[**Variables** 3](#_Toc177476301)

[**Methods** 4](#_Toc177476302)

[**Loop** 5](#_Toc177476303)

[**File I/O** 5](#_Toc177476304)

[**Exception Handling** 6](#_Toc177476305)

[**Data Structure** 6](#_Toc177476306)

[**Program Demonstration via Screenshots:** 7](#_Toc177476307)

[**Source Code Commenting** 9](#_Toc177476308)

# **Evidence of Learning:**

In this section, I will explain how key programming concepts were applied in the project, including variables, methods, loops, file input/output (I/O), exception handling, and data structures.

## **Variables**

In my project, I used several attributes (variables) to store the data for each vehicle record. These variables are defined in the FuelConsumptionRecord class and are initialized for each object created. The following variables were used:

* **model\_year**: The year the vehicle model was manufactured.
* **make**: The manufacturer or brand of the vehicle (e.g., "Toyota," "Ford").
* **model**: The specific model name of the vehicle (e.g., "Corolla," "Mustang").
* **vehicle\_class**: The classification of the vehicle based on its size or type (e.g., "SUV," "Compact").
* **engine\_size**: The engine size of the vehicle in liters (e.g., "2.0," "3.5").
* **cylinders**: The number of engine cylinders (e.g., "4," "6").
* **transmission**: The type of transmission used in the vehicle (e.g., "Automatic," "Manual").
* **fuel\_type**: The type of fuel the vehicle uses (e.g., "Gasoline," "Diesel").
* **city**: The vehicle's fuel consumption in the city, measured in liters per 100 kilometers (L/100 km).
* **highway**: The vehicle's fuel consumption on highways, measured in liters per 100 kilometers (L/100 km).
* **combined**: The combined fuel consumption (city and highway) in liters per 100 kilometers (L/100 km).
* **co2\_emissions**: The vehicle's CO2 emissions, measured in grams per kilometer (g/km).
* **co2\_rating**: The CO2 emission rating of the vehicle (on a scale, e.g., 1 to 10).
* **smog\_rating**: The smog emission rating of the vehicle (on a scale, e.g., 1 to 10).

## **Methods**

I used a constructor method (\_\_init\_\_()) in the FuelConsumptionRecord class to initialize each record with data from the CSV file. The \_\_init\_\_() method automatically runs when a new object is created, allowing me to store the vehicle data into attributes like make, model, co2\_emissions, and combined. For example:

|  |
| --- |
|  |

This method ensures that each FuelConsumptionRecord object is initialized with the correct vehicle data.

## **Loop**

To print out the information for each vehicle, I used a for loop to iterate through the list of records. The loop goes through each FuelConsumptionRecord object stored in the records list and prints the relevant details (such as the model year, make, model, combined fuel consumption, and CO2 emissions). Here is the code for the loop:

|  |
| --- |
|  |

This loop ensures that all records from the dataset are displayed to the user.

## **File I/O**

To read the data from the CSV file, I used Python's csv module. This module allows me to open the CSV file, read each row, and convert it into a FuelConsumptionRecord object. The code uses the csv.DictReader to read the CSV file and extract the necessary columns:

|  |
| --- |
|  |

This code opens the CSV file, reads each row, and appends each record to the records list.

## **Exception Handling**

To handle potential errors, such as the file not being found, I used a try-except block. This allows the program to continue running even if an error occurs, and it prints an error message instead of crashing. Here’s an example of how I handled file-related errors:

|  |
| --- |
|  |

This code ensures that the program catches file-related errors and provides a meaningful error message to the user.

## **Data Structure**

The records list was used to store all the FuelConsumptionRecord objects created from the CSV file. Lists are useful in Python for storing collections of objects, and they allow for easy iteration and manipulation of data. Once each FuelConsumptionRecord was created, it was added to the records list using the append() method:

|  |
| --- |
|  |

By using a list to store the records, I was able to loop through them and print the details of each vehicle in a structured way.

# **Program Demonstration via Screenshots:**

|  |
| --- |
|  |
|  |
|  |

In these screenshots, we see the program successfully running in **Visual Studio Code**. The main.py script reads data from the my2024-fuel-consumption-ratings.csv file, and prints detailed fuel consumption information for each vehicle in the dataset.  
the program continues to process and display records from the CSV file. The data printed to the terminal includes detailed information for various vehicles, demonstrating how the program is correctly reading, processing, and outputting the information from the dataset.  
the script continues running and printing records from the dataset without errors. It also demonstrates that exception handling was implemented properly. Although no errors occurred here, if the file were missing or another issue arose, the program would display an appropriate message.

# **Source Code Commenting**

**import csv**

***# Class to represent a fuel consumption record***

**class FuelConsumptionRecord:**

**"""**

**A class to represent a fuel consumption record.**

**Attributes:**

**model\_year (int): The year of the vehicle model.**

**make (str): The manufacturer of the vehicle.**

**model (str): The model name of the vehicle.**

**vehicle\_class (str): The class of the vehicle.**

**engine\_size (float): The size of the vehicle's engine in liters.**

**cylinders (int): The number of cylinders in the engine.**

**transmission (str): The type of transmission.**

**fuel\_type (str): The type of fuel used by the vehicle.**

**city (float): Fuel consumption in the city (L/100 km).**

**highway (float): Fuel consumption on the highway (L/100 km).**

**combined (float): Combined fuel consumption (L/100 km).**

**co2\_emissions (int): CO2 emissions in grams per kilometer.**

**co2\_rating (int): The CO2 rating.**

**smog\_rating (int): The smog rating.**

**"""**

**def \_\_init\_\_(self, model\_year, make, model, vehicle\_class, engine\_size, cylinders, transmission, fuel\_type, city, highway, combined, co2\_emissions, co2\_rating, smog\_rating):**

**"""**

**Initialize a new FuelConsumptionRecord instance.**

**Parameters:**

**model\_year (int): The year of the vehicle model.**

**make (str): The manufacturer of the vehicle.**

**model (str): The model name of the vehicle.**

**vehicle\_class (str): The class of the vehicle.**

**engine\_size (float): The size of the vehicle's engine in liters.**

**cylinders (int): The number of cylinders in the engine.**

**transmission (str): The type of transmission.**

**fuel\_type (str): The type of fuel used by the vehicle.**

**city (float): Fuel consumption in the city (L/100 km).**

**highway (float): Fuel consumption on the highway (L/100 km).**

**combined (float): Combined fuel consumption (L/100 km).**

**co2\_emissions (int): CO2 emissions in grams per kilometer.**

**co2\_rating (int): The CO2 rating.**

**smog\_rating (int): The smog rating.**

**"""**

**self.model\_year = model\_year**

**self.make = make**

**self.model = model**

**self.vehicle\_class = vehicle\_class**

**self.engine\_size = engine\_size**

**self.cylinders = cylinders**

**self.transmission = transmission**

**self.fuel\_type = fuel\_type**

**self.city = city**

**self.highway = highway**

**self.combined = combined**

**self.co2\_emissions = co2\_emissions**

**self.co2\_rating = co2\_rating**

**self.smog\_rating = smog\_rating**

***# List to store records from the CSV file***

**records = []**

***# Try to open and read the CSV file***

**try:**

***# Open the CSV file using the csv module's DictReader***

**with open('my2024-fuel-consumption-ratings.csv', mode='r') as file:**

**csv\_reader = csv.DictReader(file)**

***# Loop through each row in the CSV and create a FuelConsumptionRecord***

**for row in csv\_reader:**

**record = FuelConsumptionRecord(**

**model\_year=row['Model year'],**

**make=row['Make'],**

**model=row['Model'],**

**vehicle\_class=row['Vehicle class'],**

**engine\_size=row['Engine size (L)'],**

**cylinders=row['Cylinders'],**

**transmission=row['Transmission'],**

**fuel\_type=row['Fuel type'],**

**city=row['City (L/100 km)'],**

**highway=row['Highway (L/100 km)'],**

**combined=row['Combined (L/100 km)'],**

**co2\_emissions=row['CO2 emissions (g/km)'],**

**co2\_rating=row['CO2 rating'],**

**smog\_rating=row['Smog rating']**

**)**

***# Add the created record to the list***

**records.append(record)**

***# Handle errors if the CSV file is not found***

**except FileNotFoundError:**

**print("The CSV file was not found.")**

***# Handle any other exceptions***

**except Exception as e:**

**print(f"An error occurred: {e}")**

***# Print the program author's name***

**print("Program by: Mohammad Dellawari")**

***# Loop through the records list and print each record's details***

**for record in records:**

**"""**

**Print each vehicle's information in a formatted string.**

**The details include model year, make, model, combined fuel consumption,**

**and CO2 emissions.**

**"""**

**print(f"{record.model\_year} {record.make} {record.model} - Fuel Consumption: {record.combined}L/100km, CO2 Emissions: {record.co2\_emissions}g/km")**