**Practical Project 01**

**Course: CST8002\_040**

**Due Date:** January 26, 2025

**Professor name:** Tyler DeLay

**Student Name:** Vaishali Jaiswal (#40999221)

Table of Contents

[**Evidence of Learning:** 3](#_Toc188796236)

[**Variables** 3](#_Toc188796237)

[**Methods** 3](#_Toc188796238)

[**Loop Structure** 4](#_Toc188796239)

[**File I/O** 5](#_Toc188796240)

[**Exception Handling** 6](#_Toc188796241)

[**Data Structure** 6](#_Toc188796242)

[**Program Demonstration via Screenshots:** 7](#_Toc188796243)

[**Source Code Commenting** 10](#_Toc188796244)

# **Evidence of Learning:**

In this section, I will explain how I have integrated Python’s key programming concepts in the project, including creation and implementation of variables, methods(functions), loops, file input/output (I/O), exception handling, and data structures.  
  
Lets have a look at each component and what’s its role in programming coding for this project.

## **Variables**

I have used 6 attributes (variables) to store the data for each vehicle record (as per the csv file provided). These variables are defined in the TrafficDataRecord class and are initialized for each object created. These variables are:

* **csduid**: Unique identifier for the census subdivision (e.g.,4805026 )
* **csd:** Name of the municipality (e.g., “Drumheller”)
* **period:** Year of the record (e.g., 2000)
* **indicator\_summary:** Description of the traffic data (e.g., “Daily Vehicles (per km of road)”)
* **unit\_of\_measure:** Measurement unit (empty/null)
* **original\_value:** The recorded traffic value (e.g., 2440.67151)

## **Methods**

I used a constructor method (\_\_init\_\_()) in the TrafficDataRecord class to initialize the attributes when an object is created. Additionally, the \_\_str\_\_() method is used to return a formatted string representation of the object, which helps in displaying the data in a readable format.

I used methods read\_csv\_data() and display\_records() in the main script to read from the csv file and display in the output. For example, read\_csv\_data() method:

|  |
| --- |
| def read\_csv\_data(filename):      """      Reads the CSV file and initializes TrafficDataRecord objects.      Parameters:      filename (str): The path to the CSV file.      Returns:      list: A list of TrafficDataRecord objects in the order they appear in the file.      """      records = []      try:          with open(filename, mode='r', encoding='utf-8') as file:              csv\_reader = csv.reader(file)              header = next(csv\_reader)  *# Read and skip the header row*                for idx, row in enumerate(csv\_reader, start=1):  *# Start from line 1 after header row*  *# Handle missing values by replacing empty cells with "N/A"*                  row = [value.strip() if value.strip() else "N/A" for value in row]  *# Ensure row has exactly 6 values before processing*                  if len(row) != 6:                      print(f"Skipping incomplete row {idx + 1}: {row}")                      continue                    try:                      record = TrafficDataRecord(                          int(row[0]),  *# Convert CSDUID to int*                          row[1],       *# Municipality name*                          int(row[2]),  *# Convert Year to int*                          row[3],       *# Indicator*                          row[4],       *# Unit (with N/A fallback)*                          float(row[5]) *# Convert Value to float*                      )                      records.append(record)                  except ValueError as e:                      print(f"Skipping row {idx + 1} due to data conversion error: {row} - {e}")          return records      except FileNotFoundError:          print(f"Error: The file '{filename}' was not found. Please check the file location and try again.")      except Exception as e:          print(f"An error occurred: {e}")      return [] |

This method reads data from a CSV file, creates a TrafficDataRecord object for each row, and stores these objects in the records list. Each field from the CSV is parsed, processed, and assigned to the appropriate attributes in the object before being appended to the list for further use.

## **Loop Structure**

To store each record, I used for loop in the display\_records function iterates over the list of TrafficDataRecord objects and prints each record in a formatted tabular structure. This loop allows sequential processing of each record stored in the list and ensures all traffic data entries are presented systematically.

|  |
| --- |
| for idx, row in enumerate(csv\_reader, start=1):  *# Start from line 1 after header row*  *# Handle missing values by replacing empty cells with "N/A"*                  row = [value.strip() if value.strip() else "N/A" for value in row]  *# Ensure row has exactly 6 values before processing*                  if len(row) != 6:                      print(f"Skipping incomplete row {idx + 1}: {row}")                      continue                    try:                      record = TrafficDataRecord(                          int(row[0]),  *# Convert CSDUID to int*                          row[1],       *# Municipality name*                          int(row[2]),  *# Convert Year to int*                          row[3],       *# Indicator*                          row[4],       *# Unit (with N/A fallback)*                          float(row[5]) *# Convert Value to float*                      )                      records.append(record)                  except ValueError as e:                      print(f"Skipping row {idx + 1} due to data conversion error: {row} - {e}") |

The for loop processes each row in the CSV file, starting after the header. It first replaces empty values with "N/A", then checks if the row has exactly six values. If not, it skips to the next row. If valid, a TrafficDataRecord object is created using the row values and added to the records list. If an error occurs during conversion, the row is skipped, and an error message is displayed. The loop runs until all rows are processed.

## **File I/O**

The open() function is used to read data from the CSV file in read mode ('r'). The csv.reader() API helps in parsing the CSV file efficiently, and the header row is skipped to start reading data entries directly.

|  |
| --- |
| with open(filename, mode='r', encoding='utf-8') as file:              csv\_reader = csv.reader(file)              header = next(csv\_reader)  *# Read and skip the header row* |

It handles opening the CSV file, reading its contents, and skipping the first row (header). The with open ensures the file is closed automatically after reading, and csv.reader processes the file line by line.

## **Exception Handling**

I used a try-except block to handle potential errors such as FileNotFoundError (when the CSV file is missing) and ValueError (if the data format is incorrect). This ensures the program continues running without crashing. For example,

|  |
| --- |
| records.append(record)                  except ValueError as e:                      print(f"Skipping row {idx + 1} due to data conversion error: {row} - {e}")          return records      except FileNotFoundError:          print(f"Error: The file '{filename}' was not found. Please check the file location and try again.")      except Exception as e:          print(f"An error occurred: {e}") |

This code uses exception handling to catch errors related to file access and data conversion. If the CSV file is missing, a meaningful error message is displayed to guide the user. It also handles data formatting issues by skipping problematic rows and printing an appropriate message, ensuring the program continues running without unexpected crashes.

## **Data Structure**

I have used a list data structure, records[ ] to store vehicle records. Each record is added to the list using the append() method. The list is later accessed by the display\_records() function to present the stored data in a structured format.

A screen shot of a computer

Description automatically generated

# **Program Demonstration via Screenshots:**

Please note that, my VS code terminal doesn’t show the header so I have run it through command prompt which shows the header too.

**Screenshot from command prompt output:**

A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated

**Screenshots from VS code terminal:**

A screenshot of a computer program

Description automatically generated

**A screen shot of a computer

Description automatically generated**

**A screenshot of a computer program

Description automatically generated**

The screenshots illustrate the successful execution of the first\_python\_project.py script through command prompt and Visual Studio Code. The script reads data from the dailyvehiclesdownload.csv file and outputs vehicle traffic records, including headings CSDUID, CSD, Period, IndicatorSummaryDescription, UnitOfMeasure and OriginalValue(it shows only on command prompt output).

The program reads and displays data correctly, showing that it can process and format records properly. The output also proves that error handling works well, helping to manage issues like missing files or wrong data formats without crashing.

# **Source Code Commenting**

I have used two types of commenting, triple quoted comments (“””) and single-line comments (#)

For the first\_python\_project.py file, I have commented at the beginning of the script and as well as at each method() level:

A screenshot of a computer

Description automatically generated

A screenshot of a computer program

Description automatically generated

Same way, for class TrafficDataRecord, I have commented at the beginning what this class is about, what are the each attribute is and method()

A screenshot of a computer program

Description automatically generated

A group of colorful squares

Description automatically generated with medium confidence