Practical Project Part 2

Full Name: Meet Maheta

Course Name: CST8333

Assignment Title: Programming Language Research Project - Practical Project Part 2

Submission Date: June 16, 2024

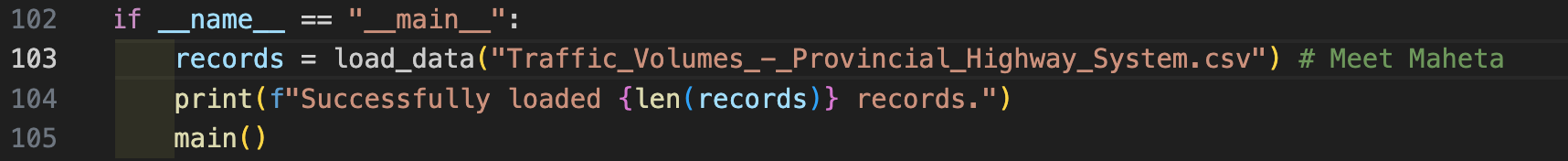
Evidence of Learning

**1. Variables:**

* filename (line 11): This variable holds the name of the CSV file to be processed.



* records (line 103): This variable is a list that will store TrafficRecord objects created from the CSV data.



* \_init\_ (lines 8): The constructor method for the TrafficRecord class, which initializes the record with data from individual attributes.

A screen shot of a computer program

Description automatically generated

* save\_data (lines 41): This function saves the list of TrafficRecord objects to a CSV file.

**A screenshot of a computer program

Description automatically generated**

* display\_records (lines 36): This function loops over the list of TrafficRecord objects and prints each one.

**A screen shot of a computer program

Description automatically generated**

**2. Loop Structure:**

* For record in records[:num\_records] (lines 47): This loop iterates over each TrafficRecord object in the records list and prints it.

**A screen shot of a computer program

Description automatically generated**

**3. File-IO Reading from the Dataset:**

* with open(filename, newline='') as csvfile (line 6): This line opens the CSV file for reading.

A screen shot of a computer program

Description automatically generated

**4. Exception Handling:**

* try-except block (lines 22): This block handles exceptions that may occur while reading the CSV file.

**A computer screen shot of a program code

Description automatically generated**

**5. Use of an API Library:**

* pandas (line 3): The pandas library is used for reading the CSV file.

**A black background with colorful text

Description automatically generated**

**6. Array (or similar data structure):**

* records (line 24): This list stores instances of TrafficRecord objects.

A screen shot of a computer code

Description automatically generated

Program Architecture

The design of this project follows a multi-layered architecture, which organizes the code into distinct layers to separate concerns and improve maintainability. The main layers in this project are:

1. Presentation Layer
2. Business Logic Layer
3. Persistence Layer

**1. Presentation Layer**

This layer is responsible for interacting with the user. It handles input and output operations, displaying menus, and getting user choices. It primarily consists of functions that print messages to the console and gather input from the user.

**Files:**

1. main.py
2. presentation.py

**Responsibilities:**

* Displaying the main menu to the user.
* Handling user input and output.
* Calling functions from the Business Logic Layer based on user input.

**2. Business Logic Layer**

This layer contains the core functionality and logic of the application. It handles operations such as adding, editing, deleting, and managing records. It acts as an intermediary between the Presentation Layer and the Persistence Layer.

**Files:**

1. business.py

**Responsibilities:**

* Managing records (add, edit, delete).
* Validating user input.
* Processing and manipulating data before passing it to the Persistence Layer.

**3. Persistence Layer**

This layer is responsible for data storage and retrieval. It interacts with the file system to load and save data. This layer ensures that data is correctly read from and written to CSV files.

**Files:**

1. persistence.py

**Responsibilities:**

* Loading data from the CSV file.
* Saving data to the CSV file.
* Converting data between file format and application format.

**UML Diagram:**

In this project, the main classes are TrafficRecord and TrafficManager.

Here's a brief description of the classes and their relationships:

TrafficRecord: Represents a single traffic record with attributes such as section ID, highway, section, etc.

TrafficManager: Manages a collection of TrafficRecord objects and provides methods to add, edit, delete, and retrieve records.

Below is a UML class diagram representing the relationship between TrafficRecord and TrafficManager:

A screenshot of a computer

Description automatically generated

### **Explanation of the UML Diagram**

* **TrafficManager** has a one-to-many relationship with **TrafficRecord**.
* **TrafficManager** contains a list of **TrafficRecord** objects.
* **TrafficManager** provides methods to manipulate the list of **TrafficRecord** objects, including adding, editing, deleting, and retrieving records.

### **Explanation of Class Methods**

* **TrafficRecord**:
  + \_\_init\_\_(self, \*args): Initializes the traffic record with the provided attributes.
  + \_\_repr\_\_(self): Returns a string representation of the traffic record.
* **TrafficManager**:
  + \_\_init\_\_(self, filename: str): Initializes the traffic manager and loads data from the specified file.
  + add\_record(self, record\_data: dict): Adds a new record to the list.
  + edit\_record(self, index: int, new\_data: dict): Edits an existing record at the specified index.
  + delete\_record(self, index: int): Deletes a record at the specified index.
  + get\_record(self, index: int) -> TrafficRecord: Retrieves a record at the specified index.
  + load\_data(self, filename: str) -> list<TrafficRecord>: Loads data from the specified file and returns a list of records.
  + save\_data(self, filename: str): Saves the current list of records to the specified file.

This UML class diagram and explanation provide a clear view of the relationships and responsibilities of the classes within this project.

Program Demonstration via Screen Shots

### **Introduction:**

This section includes screenshots of the running program. The screenshots display records from the dataset and ensure that my full name, Meet Maheta, is visible on the screen.

### **Description:**

**Screenshot 1: Initial Data Load**

A screen shot of a computer

Description automatically generated

* This screenshot shows the program reading the CSV file and displaying the records.
* Each record is displayed with its details, including fields like SECTION ID, HIGHWAY, SECTION LENGTH, and more.
* My full name, Meet Maheta, is visible at the bottom of the output to ensure it always remains on the screen.

**Screenshot 2: Reload Data**

A screenshot of a computer

Description automatically generated

* This screenshot demonstrates the functionality to reload data from the CSV file.
* The program successfully reloads and displays the total number of records.
* My full name, Meet Maheta, is visible at the bottom of the output.

**Screenshot 3: Save Data**

A screen shot of a computer

Description automatically generated

* This screenshot illustrates the functionality to save the current data to an output CSV file.
* The program confirms that the data has been saved successfully.
* My full name, Meet Maheta, is visible at the bottom of the output.

**Screenshot 4: Display Records**

A screenshot of a computer

Description automatically generated

* This screenshot shows the functionality to display a specified number of records.
* The user inputs the number of records to display, and the program outputs the requested records.
* My full name, Meet Maheta, is visible at the bottom of the output.

**Screenshot 5: Add a New Record**

A screenshot of a computer program

Description automatically generated

A screen shot of a computer program

Description automatically generated

A screenshot of a computer

Description automatically generated

* This screenshot displays the functionality to add a new record to the dataset.
* The user inputs the details for the new record, and the program confirms that the record has been added successfully.
* My full name, Meet Maheta, is visible at the bottom of the output.

**Screenshot 6: Edit an Existing Record (Changed the new added Section ID from 1081 to 1111)**

A screenshot of a computer program

Description automatically generated

A screenshot of a computer

Description automatically generated

* This screenshot demonstrates the functionality to edit an existing record in the dataset.
* The user inputs the SECTION ID of the record to edit, along with the updated details. The program confirms that the record has been edited successfully.
* My full name, Meet Maheta, is visible at the bottom of the output.

**Screenshot 7: Delete a Record (Deleted an already existing record at line 10618)**

A screenshot of a computer program

Description automatically generated

A screenshot of a computer

Description automatically generated

* This screenshot shows the functionality to delete a record from the dataset.
* The user inputs the SECTION ID of the record to delete, and the program confirms that the record has been deleted successfully.
* My full name, Meet Maheta, is visible at the bottom of the output.

These descriptions provide a clear understanding of the various functionalities implemented in the program, as evidenced by the screenshots. Each step ensures that my full name, Meet Maheta, is visible to maintain the authenticity and integrity of the project.

Unit Testing Demonstration via Screen Shots

**Introduction:**

This section includes screenshots of the unit tests for the program. The screenshots show the unit tests being executed either within an IDE or from a console, ensuring that my full name, Meet Maheta, is visible in the screenshots. These tests validate the functionality of the key features of the application, such as adding, editing, and deleting records.

**Running Unit Tests from Console**

**A screen shot of a computer

Description automatically generated**

**Details:**

The screenshot includes the console output, showing the results of the tests. It confirms that test of all functions for records have passed or if there are any failures. My full name, Meet Maheta, is included in the console prompt or output to ensure it is always visible.

Source Code Commenting Example

#### **Main Program File:**

# Author: Meet Maheta

# main.py

from business import TrafficManager

from presentation import display\_records, display\_menu, get\_user\_choice, get\_record\_details, display\_message

def main():

"""

The main function to run the traffic data management program.

It initializes the TrafficManager and provides a menu-driven interface for user interaction.

"""

filename = "Traffic\_Volumes\_-\_Provincial\_Highway\_System.csv"

manager = TrafficManager(filename) # Initialize the TrafficManager with the given filename

while True:

display\_menu() # Display the main menu

choice = get\_user\_choice() # Get the user's choice from the menu

if choice is None:

continue

if choice == 1:

display\_message("Reloading data...")

manager.reload\_data() # Reload data from the CSV file

display\_message(f"Successfully loaded {len(manager.records)} records.")

elif choice == 2:

display\_message("Saving data...")

manager.save\_data() # Save data to the CSV file

display\_message("Data saved successfully.")

elif choice == 3:

num\_records = int(input("Enter the number of records to display: "))

display\_records(manager.records, num\_records) # Display the specified number of records

elif choice == 4:

display\_message("Adding a new record...")

details = get\_record\_details() # Get new record details from the user

manager.add\_record({

'section\_id': details[0],

'highway': details[1],

'section': details[2],

'section\_length': details[3],

'section\_description': details[4],

'date': details[5],

'description': details[6],

'group': details[7],

'type\_': details[8],

'county': details[9],

'ptrucks': details[10],

'adt': details[11],

'aadt': details[12],

'direction': details[13],

'pct85': details[14],

'priority\_points': details[15]

}) # Add the new record to the manager

display\_message("Record added successfully.")

elif choice == 5:

display\_message("Editing a record...")

section\_id = input("Enter the SECTION ID of the record to edit: ")

details = get\_record\_details() # Get updated record details from the user

index = next((i for i, r in enumerate(manager.records) if r.section\_id == section\_id), None)

if index is not None and manager.edit\_record(index, {

'section\_id': details[0],

'highway': details[1],

'section': details[2],

'section\_length': details[3],

'section\_description': details[4],

'date': details[5],

'description': details[6],

'group': details[7],

'type\_': details[8],

'county': details[9],

'ptrucks': details[10],

'adt': details[11],

'aadt': details[12],

'direction': details[13],

'pct85': details[14],

'priority\_points': details[15]

}):

display\_message("Record edited successfully.")

else:

display\_message("Record not found.")

elif choice == 6:

display\_message("Deleting a record...")

section\_id = input("Enter the SECTION ID of the record to delete: ")

index = next((i for i, r in enumerate(manager.records) if r.section\_id == section\_id), None)

if index is not None and manager.delete\_record(index):

display\_message("Record deleted successfully.")

else:

display\_message("Record not found.")

elif choice == 0:

display\_message("Exiting program.")

break

else:

display\_message("Invalid choice. Please try again.")

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

main()

#### **Unit Test File:**

# Author: Meet Maheta

# test\_business.py

import unittest

from business import TrafficManager

class TestTrafficManager(unittest.TestCase):

"""

Unit tests for the TrafficManager class.

"""

def setUp(self):

"""

Set up the test environment by initializing a TrafficManager instance.

"""

self.manager = TrafficManager('Traffic\_Volumes\_-\_Provincial\_Highway\_System.csv')

def test\_add\_record(self):

"""

Test the add\_record method to ensure it correctly adds a new record.

"""

initial\_count = len(self.manager.records)

self.manager.add\_record({

'section\_id': '9999', 'highway': 'H', 'section': 'S', 'section\_length': '1',

'section\_description': 'Desc', 'date': '2023-01-01', 'description': 'Desc',

'group': 'G', 'type\_': 'T', 'county': 'C', 'ptrucks': 'P', 'adt': 'A',

'aadt': 'A', 'direction': 'D', 'pct85': '85', 'priority\_points': 'PP'

})

self.assertEqual(len(self.manager.records), initial\_count + 1)

def test\_reload\_data(self):

"""

Test the reload\_data method to ensure it correctly reloads the data.

"""

self.manager.reload\_data()

self.assertGreater(len(self.manager.records), 0)

def test\_edit\_record(self):

"""

Test the edit\_record method to ensure it correctly edits an existing record.

"""

self.manager.add\_record({

'section\_id': '9999', 'highway': 'H', 'section': 'S', 'section\_length': '1',

'section\_description': 'Desc', 'date': '2023-01-01', 'description': 'Desc',

'group': 'G', 'type\_': 'T', 'county': 'C', 'ptrucks': 'P', 'adt': 'A',

'aadt': 'A', 'direction': 'D', 'pct85': '85', 'priority\_points': 'PP'

})

index = len(self.manager.records) - 1

self.manager.edit\_record(index, {'adt': '2000'})

self.assertEqual(self.manager.get\_record(index).adt, '2000')

def test\_delete\_record(self):

"""

Test the delete\_record method to ensure it correctly deletes a record.

"""

self.manager.add\_record({

'section\_id': '9999', 'highway': 'H', 'section': 'S', 'section\_length': '1',

'section\_description': 'Desc', 'date': '2023-01-01', 'description': 'Desc',

'group': 'G', 'type\_': 'T', 'county': 'C', 'ptrucks': 'P', 'adt': 'A',

'aadt': 'A', 'direction': 'D', 'pct85': '85', 'priority\_points': 'PP'

})

index = len(self.manager.records) - 1

self.manager.delete\_record(index)

self.assertIsNone(self.manager.get\_record(index))

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

unittest.main()