



INFORMATICS  
INSTITUTE OF  
TECHNOLOGY

# **Software Development I**

**Resit Coursework 2023/2024**

(Scientific Research Data Management  
System)

Module Leader : Mr. Guhanathan Poravi

Student Name - Sinali Tharindi Senarathne

UOW Number - w2087753

IIT Number - 20230514

## Contents

Part 01.....	4
Structured Programming with Functions.....	4
1. Documentation .....	4
2. Implemented Python code for Part_a .....	5
3. Test Plan .....	8
4. Report .....	10
Part 02.....	12
Object-Oriented Programming (OOP) with Classes .....	12
1. Documentation .....	12
2. Implemented Python Code for Part_b.....	13
3. Test Plan .....	15
4. Report .....	21
Part 03.....	22
Advanced Serialization with Avro .....	22
1. Documentation .....	22
2. Implemented Python Code for Part_c .....	23
3. Test Plan .....	26
4. Report .....	29
Part 04.....	30
GUI Development with Tkinter .....	30
1. Documentation .....	30
2. Implemented Python Code for Part_d.....	31
3. Test Plan .....	34
4. Report .....	39

Figure 1 .....	9
Figure 2 .....	9
Figure 3 .....	9
Figure 4 .....	10
Figure 5 .....	10
Figure 6 .....	10
Figure 7 .....	11
Figure 8 .....	17
Figure 9 .....	18
Figure 10 .....	18
Figure 11 .....	19
Figure 12 .....	19
Figure 13 .....	19
Figure 14 .....	20
Figure 15 .....	20
Figure 16 .....	28
Figure 17 .....	29
Figure 18 .....	36
Figure 19 .....	37
Figure 20 .....	37
Figure 21 .....	38
Figure 22 .....	38
Figure 23 .....	38
Figure 24 .....	38
Figure 25 .....	38

# **Part 01**

## **Structured Programming with Functions**

### 1. Documentation

#### **Design Choices:**

- **Structured Programming Paradigm:**
  - This part uses a structured programming approach where the program is organized around a series of individual functions. Each function is designed to perform a specific task related to research data management, such as adding, viewing, updating, and deleting entries.
  - The decision to use functions allows for clear separation of concerns, making the code easier to read, understand, and maintain. Each function is responsible for one aspect of the program's functionality, promoting a modular design.
- **Data Representation:**
  - Research data entries are represented as lists. Each entry is a list containing an experiment name, date, researcher name, and a series of data points (e.g., measurements).
  - This simple data structure was chosen for its ease of use and straightforward implementation. Lists allow for quick access and manipulation of data points.
- **File Handling:**
  - The program uses plain text files to store and retrieve research data. Each entry is saved as a comma-separated value (CSV) line in a text file, which provides an easily readable and editable format.
  - Using text files is a deliberate choice to keep the file I/O operations simple and accessible without requiring external libraries or complex data formats.

#### **Structure of the Code:**

- **add\_entry(entries):**
  - Prompts the user for input to create a new research data entry and appends it to the list of entries.
- **view\_entries(entries):**
  - Iterates over the list of entries and prints each entry, displaying the research data in a readable format.
- **update\_entry(entries):**
  - Allows the user to select an entry to update. It prompts for new values, with the option to leave fields unchanged. The entry is then updated in the list.
- **delete\_entry(entries):**
  - Displays all entries and allows the user to select and delete a specific entry from the list.
- **save\_entries\_to\_file(entries, filename):**
  - Writes all entries to a specified text file, saving each entry as a CSV line.
- **load\_entries\_from\_file(filename):**

- Reads entries from a specified text file and loads them into the program's list of entries. The function parses each line of the file into a list.
- **analyze\_data(entries):**
  - Performs basic statistical analysis on the data points of each entry, such as calculating the average.

### Instructions to Run:

1. **Setup:**
  - Ensure Python is installed on your system. If not, download and install Python from the official website.
2. **Save the Code:**
  - Copy the provided code into a file named `parta.py`.
3. **Prepare Data (Optional):**
  - If you have existing research data, save it as `research_data.txt` in the same directory as `parta.py`. The file should follow a CSV format, with each entry on a new line.
4. **Run the Program:**
  - Open a command prompt or terminal, navigate to the directory containing `parta.py`, and run the program by typing `python parta.py`.
5. **Interact with the Program:**
  - Use the menu options to add, view, update, delete, analyze, and save research data entries.

### 2. Implemented Python code for Part\_a

```
import os

# Function to add a research data entry
def add_entry(entries):
    experiment_name = input("Enter experiment name: ")
    date = input("Enter date (YYYY-MM-DD): ")
    researcher = input("Enter researcher's name: ")
    data_points = list(map(float, input("Enter data points separated by commas: ").split(',')))
    entry = [experiment_name, date, researcher] + data_points
    entries.append(entry)

# Function to view all research data entries
def view_entries(entries):
    for index, entry in enumerate(entries, start=1):
        print(f"{index}: {entry}")
```

```

# Function to update a research data entry
def update_entry(entries):
    view_entries(entries)
    index = int(input("Enter the number of the entry you want to update: ")) - 1
    if 0 <= index < len(entries):
        print("Leave blank if you don't want to change a field.")
        experiment_name = input(f"Enter new experiment name
({entries[index][0]}): ") or entries[index][0]
        date = input(f"Enter new date ({entries[index][1]}): ") or
entries[index][1]
        researcher = input(f"Enter new researcher's name ({entries[index][2]}):
") or entries[index][2]
        data_points = input(f"Enter new data points separated by commas
({'.'.join(map(str, entries[index][3:]))}): ")
        data_points = list(map(float, data_points.split(','))) if data_points
else entries[index][3:]
        entries[index] = [experiment_name, date, researcher] + data_points
        print("Entry updated successfully.")
    else:
        print("Invalid entry number.")

# Function to delete a research data entry
def delete_entry(entries):
    view_entries(entries)
    index = int(input("Enter the number of the entry you want to delete: ")) - 1
    if 0 <= index < len(entries):
        del entries[index]
        print("Entry deleted successfully.")
    else:
        print("Invalid entry number.")

# Function to save entries to a text file
def save_entries_to_file(entries, filename):
    with open(filename, 'w') as file:
        for entry in entries:
            file.write('.'.join(map(str, entry)) + '\n')
    print("Entries saved successfully.")

# Function to load entries from a text file
def load_entries_from_file(filename):
    entries = []
    if os.path.exists(filename):
        with open(filename, 'r') as file:
            for line in file:
                entries.append(line.strip().split(','))

```

```

else:
    print("File not found.")
    return entries

# Function to perform data analysis
def analyze_data(entries):
    for entry in entries:
        data_points = list(map(float, entry[3:]))
        average = sum(data_points) / len(data_points)
        print(f"{entry[0]} (by {entry[2]} on {entry[1]}): Average = {average}")

# Main function to interact with the user
def main():
    entries = []
    filename = "research_data.txt"

    while True:
        print("\nMenu:")
        print("1. Add a research data entry")
        print("2. View all entries")
        print("3. Update an entry")
        print("4. Delete an entry")
        print("5. Analyze data")
        print("6. Save entries to file")
        print("7. Load entries from file")
        print("8. Exit")

        choice = input("Enter your choice: ")
        if choice == '1':
            add_entry(entries)
        elif choice == '2':
            view_entries(entries)
        elif choice == '3':
            update_entry(entries)
        elif choice == '4':
            delete_entry(entries)
        elif choice == '5':
            analyze_data(entries)
        elif choice == '6':
            save_entries_to_file(entries, filename)
        elif choice == '7':
            entries = load_entries_from_file(filename)
        elif choice == '8':
            break
        else:

```

```

        print("Invalid choice, please try again.")

if __name__ == "__main__":
    main()

```

### 3. Test Plan

#### Objective:

The goal is to validate that each function in the structured programming model performs as expected under various scenarios. The program should handle typical user interactions, edge cases, and potential errors without crashing.

#### Test Cases:

No	Test Case	Expected Result	Actual Result	Pass/Fail
01	Adding a valid experiment name, date, researcher and data points	Entry is added to the list and can be viewed.	Entry is added to the list and can be viewed.	"Pass"
02	Viewing Entries after adding	All entries are displayed in the correct format.	All entries are displayed in the correct format.	"Pass"
03	Updating an Entry	The selected entry is updated and saved.	The selected entry is updated and saved1 .	"Pass"
04	Deleting an Entry	The selected entry is removed from the list.	The selected entry is removed from the list.	"Pass"
05	Saving to File	A text file is created with all entries.	A text file is created with all entries.	"Pass"
06	Loading from File	Entries are correctly loaded and can be viewed.	Entries are correctly loaded and can be viewed.	"Pass"
07	Analyzing Data	Averages of data points are correctly calculated.	Averages of data points are correctly calculated.	"Pass"



```

34\python.exe' 'c:\Users\USER\.vscode\extensions\ms-python.debugpy-2024.10.0-win32-x64\bundled\lib
s\debugpy\adapter\..\..\debugpy\launcher' '52297' '--' 'C:\Users\USER\OneDrive\Desktop\SD1\part_a.
py'

Menu:
1. Add a research data entry
2. View all entries
3. Update an entry
4. Delete an entry
5. Analyze data
6. Save entries to file
7. Load entries from file
8. Exit
Enter your choice: 1
Enter experiment name: expe01
Enter date (YYYY-MM-DD): 2024-08-14
Enter researcher's name: sinali
Enter data points separated by commas: 34,7,11

```

Figure 1

```

8. Exit
Enter your choice: 1
Enter experiment name: expe01
Enter date (YYYY-MM-DD): 2024-08-14
Enter researcher's name: sinali
Enter data points separated by commas: 34,7,11

Menu:
1. Add a research data entry
2. View all entries
3. Update an entry
4. Delete an entry
5. Analyze data
6. Save entries to file
7. Load entries from file
8. Exit
Enter your choice: 2
1: ['expe01', '2024-08-14', 'sinali', 34.0, 7.0, 11.0]

```

Figure 2

```

Menu:
1. Add a research data entry
2. View all entries
3. Update an entry
4. Delete an entry
5. Analyze data
6. Save entries to file
7. Load entries from file
8. Exit
Enter your choice: 4
1: ['expe02', '2024-10-08', 'dinithi', 7.0, 8.0, 9.0]
Enter the number of the entry you want to delete: 1
Entry deleted successfully.

```

Figure 3

```
Menu:
1. Add a research data entry
2. View all entries
3. Update an entry
4. Delete an entry
5. Analyze data
6. Save entries to file
7. Load entries from file
8. Exit
Enter your choice: 3
1: ['expe01', '2024-08-14', 'sinali', 34.0, 7.0, 11.0]
Enter the number of the entry you want to update: 1
Leave blank if you don't want to change a field.
Enter new experiment name (expe01):
Enter new date (2024-08-14): 2024-08-15
Enter new researcher's name (sinali): tharindi
Enter new data points separated by commas (34.0,7.0,11.0):
Entry updated successfully.
```

Figure 4

```
Menu:
1. Add a research data entry
2. View all entries
3. Update an entry
4. Delete an entry
5. Analyze data
6. Save entries to file
7. Load entries from file
8. Exit
Enter your choice: 6
Entries saved successfully.
```

Figure 5

```
≡ research_data.txt X
≡ research_data.txt
1  expe03,2024-08-20,sinali,6.0,8.0,10.0
2  expe04,2024-08-24,tharindi,61.0,45.0,8.0
3  
```

Figure 6

```
Menu:
1. Add a research data entry
2. View all entries
3. Update an entry
4. Delete an entry
5. Analyze data
6. Save entries to file
7. Load entries from file
8. Exit
Enter your choice: 5
expe03 (by sinali on 2024-08-20): Average = 8.0
expe04 (by tharindi on 2024-08-24): Average = 38.0
```

Figure 7

#### 4. Report

##### Summary:

In Part A, the primary objective was to implement a basic research data management system using structured programming techniques. The design focused on simplicity and clarity, with each function handling a distinct part of the program's functionality. This approach allowed for easy debugging and understanding of the code, making it accessible even to those unfamiliar with advanced programming concepts.

##### Challenges:

- **Handling User Input:**
  - Ensuring that the program correctly handled various types of user input (e.g., empty strings, incorrect data formats) was a key challenge. The solution involved adding input validation and error messages to guide the user.
- **Data Persistence:**
  - Ensuring data was accurately saved and loaded from files required careful handling of file I/O operations. We chose a simple CSV format to make the data both human-readable and easily parsable by the program.

##### Key Learnings:

- **Modular Function Design:**
  - Dividing the program into functions with clear responsibilities made the code easier to maintain and test. This modularity is crucial for debugging and for future expansion of the program.
- **Basic File Handling:**
  - Working with text files taught the importance of consistent data formatting. Understanding how to open, read, write, and close files in Python is fundamental for many programming tasks.

- **User Interaction:**
  - Designing a user-friendly menu system emphasized the need for clear instructions and robust error handling. Anticipating user mistakes and providing helpful feedback improved the overall user experience.

## Part 02

# Object-Oriented Programming (OOP) with Classes

### 1. Documentation

#### Design Choices:

- **Object-Oriented Design:** The program was refactored to use a class (`ResearchDataManager`) to encapsulate data and related operations, following the principles of OOP.
- **Encapsulation:** All research data entries and associated methods are encapsulated within the `ResearchDataManager` class, providing a clear and organized structure.
- **Enhanced Modularity:** Methods like `add_entry`, `update_entry`, and `delete_entry` are now part of the class, which enhances the code's modularity and reusability.

#### Structure of the Code:

- **ResearchDataManager:**
  - `add_entry()`: Adds a new entry to the internal list of entries.
  - `view_entries()`: Displays all entries.
  - `update_entry()`: Updates an existing entry.
  - `delete_entry()`: Deletes an existing entry.
  - `save_entries_to_file()`: Saves all entries to a text file.
  - `load_entries_from_file()`: Loads entries from a text file.
  - `analyze_data()`: Performs basic data analysis.

#### Instructions to Run:

1. Ensure Python is installed on your system.
2. Save the code to a `.py` file (e.g., `partb.py`).
3. Place any existing `research_data.txt` file in the same directory as the script.
4. Run the script by executing `python partb.py` in the command line.

5. Use the menu to interact with the `ResearchDataManager` object.

## 2. Implemented Python Code for Part\_b

```
import os

class ResearchDataManager:
    def __init__(self):
        self.entries = []
        self.filename = "research_data.txt"

    def add_entry(self):
        experiment_name = input("Enter experiment name: ")
        date = input("Enter date (YYYY-MM-DD): ")
        researcher = input("Enter researcher's name: ")
        data_points = list(map(float, input("Enter data points separated by
commas: ").split(',')))
        entry = [experiment_name, date, researcher] + data_points
        self.entries.append(entry)

    def view_entries(self):
        for index, entry in enumerate(self.entries, start=1):
            print(f"{index}: {entry}")

    def update_entry(self):
        self.view_entries()
        index = int(input("Enter the number of the entry you want to update: "))
        - 1
        if 0 <= index < len(self.entries):
            print("Leave blank if you don't want to change a field.")
            experiment_name = input(f"Enter new experiment name
({self.entries[index][0]}): ") or self.entries[index][0]
            date = input(f"Enter new date ({self.entries[index][1]}): ") or
self.entries[index][1]
            researcher = input(f"Enter new researcher's name
({self.entries[index][2]}): ") or self.entries[index][2]
            data_points = input(f"Enter new data points separated by commas
({'','.join(map(str, self.entries[index][3:]))}): ")
            data_points = list(map(float, data_points.split(','))) if data_points
else self.entries[index][3:]
            self.entries[index] = [experiment_name, date, researcher] +
data_points
            print("Entry updated successfully.")
        else:
```

```

        print("Invalid entry number.")

def delete_entry(self):
    self.view_entries()
    index = int(input("Enter the number of the entry you want to delete: "))
- 1
    if 0 <= index < len(self.entries):
        del self.entries[index]
        print("Entry deleted successfully.")
    else:
        print("Invalid entry number.")

def save_entries_to_file(self):
    with open(self.filename, 'w') as file:
        for entry in self.entries:
            file.write(','.join(map(str, entry)) + '\n')
    print("Entries saved successfully.")

def load_entries_from_file(self):
    if os.path.exists(self.filename):
        with open(self.filename, 'r') as file:
            self.entries = [line.strip().split(',') for line in file]
    else:
        print("File not found.")

def analyze_data(self):
    for entry in self.entries:
        data_points = list(map(float, entry[3:]))
        average = sum(data_points) / len(data_points)
        print(f"{entry[0]} (by {entry[2]} on {entry[1]}): Average =
{average}")

def main():
    manager = ResearchDataManager()

    while True:
        print("\nMenu:")
        print("1. Add a research data entry")
        print("2. View all entries")
        print("3. Update an entry")
        print("4. Delete an entry")
        print("5. Analyze data")
        print("6. Save entries to file")
        print("7. Load entries from file")
        print("8. Exit")

```

```

choice = input("Enter your choice: ")
if choice == '1':
    manager.add_entry()
elif choice == '2':
    manager.view_entries()
elif choice == '3':
    manager.update_entry()
elif choice == '4':
    manager.delete_entry()
elif choice == '5':
    manager.analyze_data()
elif choice == '6':
    manager.save_entries_to_file()
elif choice == '7':
    manager.load_entries_from_file()
elif choice == '8':
    break
else:
    print("Invalid choice, please try again.")

if __name__ == "__main__":
    main()

```

### 3. Test Plan

**Objective:** To ensure the 'ResearchDataManager' class functions correctly, maintains data integrity, and behaves as expected under different conditions. The test plan focuses on class methods and their interactions.

#### Test Cases:

No	Test Case	Expected Result	Actual Result	Pass/Fail
01	Create an instance of 'ResearchDataManager'	The class initialize with an empty list of entries and a predefined filename.	The class initialize with an empty list of entries and a predefined filename.	"Pass"

02	Add a valid entry through the <code>add_entry()</code> method.	The entry is added to the class's internal list and can be viewed using <code>view_entries()</code> .	The entry is added to the class's internal list and can be viewed using <code>view_entries()</code> .	"pass"
03	View all entries after adding multiple entries.	All entries are displayed in the correct format, reflecting any data added.	All entries are displayed in the correct format, reflecting any data added.	"pass"
04	Select and update an entry's date and data points using <code>update_entry()</code> .	The selected entry is updated within the class, and the changes are reflected when viewing the entries.	The selected entry is updated within the class, and the changes are reflected when viewing the entries.	"pass"
05	Delete a specific entry using the <code>delete_entry()</code> method.	The entry is removed from the internal list, and subsequent entries shift up in the index.	The entry is removed from the internal list, and subsequent entries shift up in the index.	"Pass"
06	Save the list of entries to a text file using <code>save_entries_to_file()</code> .	The text file is created or updated with all entries in the correct CSV format.	The text file is created or updated with all entries in the correct CSV format.	"Pass"



07	Load entries from an existing text file using <code>load_entries_from_file()</code>	The entries are correctly loaded into the class's internal list, matching the data in the file.	The entries are correctly loaded into the class's internal list, matching the data in the file.	"Pass"
08	Data analysis on entries with various data points using <code>analyze_data()</code> .	Perform data analysis on entries with various data points using <code>analyze_data()</code> .	Perform data analysis on entries with various data points using <code>analyze_data()</code> .	"Pass"
09	Error Handling: Test how the class handles invalid inputs	The class should handle invalid input gracefully, prompting the user for correct input without crashing.	The class should handle invalid input gracefully, prompting the user for correct input without crashing.	"Pass"

```

8. Exit
Enter your choice: 1
Enter experiment name: expe05
Enter date (YYYY-MM-DD): 2024-08-22
Enter researcher's name: dinithi
Enter data points separated by commas: 65,9,7

Menu:
1. Add a research data entry
2. View all entries
3. Update an entry
4. Delete an entry
5. Analyze data
6. Save entries to file
7. Load entries from file
8. Exit
Enter your choice: 2
1: ['expe05', '2024-08-22', 'dinithi', 65.0, 9.0, 7.0]

```

Figure 8

```
Menu:
1. Add a research data entry
2. View all entries
3. Update an entry
4. Delete an entry
5. Analyze data
6. Save entries to file
7. Load entries from file
8. Exit
Enter your choice: 2
1: ['expe06', '2024-08-20', 'shone', 4.0, 6.0, 5.0]
2: ['expe07', '2024-08-24', 'dinali', 20.0, 7.0, 9.0]
```

Figure 9

```
1. Add a research data entry
2. View all entries
3. Update an entry
4. Delete an entry
5. Analyze data
6. Save entries to file
7. Load entries from file
8. Exit
Enter your choice: 3
1: ['expe06', '2024-08-20', 'shone', 4.0, 6.0, 5.0]
2: ['expe07', '2024-08-24', 'dinali', 20.0, 7.0, 9.0]
Enter the number of the entry you want to update: 2
Leave blank if you don't want to change a field.
Enter new experiment name (expe07): expe08
Enter new date (2024-08-24):
Enter new researcher's name (dinali):
Enter new data points separated by commas (20.0,7.0,9.0): 20,10,9
Entry updated successfully.
```

Figure 10

```
Menu:
1. Add a research data entry
2. View all entries
3. Update an entry
4. Delete an entry
5. Analyze data
6. Save entries to file
7. Load entries from file
8. Exit
Enter your choice: 4
1: ['expe06', '2024-08-20', 'shone', 4.0, 6.0, 5.0]
2: ['expe08', '2024-08-24', 'dinali', 20.0, 10.0, 9.0]
Enter the number of the entry you want to delete: 1
Entry deleted successfully.
```

Figure 11

```
Menu:
1. Add a research data entry
2. View all entries
3. Update an entry
4. Delete an entry
5. Analyze data
6. Save entries to file
7. Load entries from file
8. Exit
Enter your choice: 6
Entries saved successfully.
```

Figure 12



```
research_data.txt X part_b.py
research_data.txt
1 expe08,2024-08-24,dinali,20.0,10.0,9.0
2
```

Figure 13



## 4. Report

**Summary:** Part B involved refactoring the structured code from Part A into an object-oriented design. This approach allowed for better organization and encapsulation of the program's functionality, making it easier to manage and extend.

### **Challenges:**

- **Class Design:** Designing the `ResearchDataManager` class to be flexible enough to handle different operations while maintaining encapsulation.
- **Method Integration:** Ensuring that all methods interact properly with the class's internal state, especially when modifying the list of entries.

### **Key Learnings:**

- **OOP Principles:** Encapsulating data and related functions within a class improves code maintainability and readability.
- **Scalability:** The object-oriented design allows for easier extension of the program (e.g., adding new methods or modifying existing ones).

## **Part 03**

# **Advanced Serialization with Avro**

### 1. Documentation

#### **Design Choices:**

- **Advanced Serialization with Avro:**
  - In Part C, the program was extended to use Apache Avro for serialization, a binary data format that allows for efficient and schema-based storage of research data. This choice was made to enhance data integrity, security, and performance, especially for large datasets.
  - Avro's schema-based approach ensures that data is stored and retrieved in a structured format, providing validation and preventing issues like data corruption.
- **Data Integrity and Security:**
  - By using Avro, the program can enforce schema validation, ensuring that only data that matches the predefined schema is serialized and deserialized. This approach enhances data integrity and helps prevent unauthorized data modifications.
- **Class Structure:**
  - The `ResearchDataManager` class was enhanced to include methods for serializing and deserializing data using Avro. This addition allows for seamless integration of Avro's advanced features within the existing object-oriented framework.

#### **Structure of the Code:**

- **ResearchDataManager:**
  - `init()`: Initializes the class with an empty list of entries, a filename, and an Avro schema.
  - `add_entry()`: Adds a new entry, following the Avro schema.
  - `view_entries()`: Displays all entries stored in the class.
  - `update_entry()`: Allows the user to update an entry while maintaining schema integrity.
  - `delete_entry()`: Deletes an entry based on user input.
  - `save_entries_to_file()`: Serializes entries using Avro and saves them to a file.
  - `load_entries_from_file()`: Deserializes entries from an Avro file into the class's internal list.
  - `analyze_data()`: Performs analysis on the data points, ensuring calculations adhere to the serialized data's structure.

#### **Instructions to Run:**

### 1. Setup:

- Ensure Python is installed on your system.
- Install the Avro Python library by running `pip install avro-python3` in your command prompt or terminal.

### 2. Save the Code:

- Copy the provided code into a file named `partc.py`.

### 3. Prepare Schema and Data Files:

- Ensure the `research_data_schema.avsc` file is in the same directory as `partc.py`. This file should contain the Avro schema for the research data.
- Optionally, prepare an existing `research_data.avro` file if you want to load pre-existing serialized data.

### 4. Run the Program:

- Open a command prompt or terminal, navigate to the directory containing `partc.py`, and run the program by typing `python partc.py`.

### 5. Interact with the Program:

- Use the menu options to manage your research data, including adding, viewing, updating, deleting, analyzing, saving, and loading entries with Avro serialization.

## 2. Implemented Python Code for Part\_c

```
import os
import avro.schema
import avro.io
import io

class ResearchDataManager:
    def __init__(self):
        self.entries = []
        self.filename = "research_data.avro"
        self.schema_file = "research_data_schema.avsc"
        self.schema = avro.schema.Parse(open(self.schema_file, "r").read())

    def add_entry(self, experiment_name, date, researcher, data_points):
        entry = {"experiment_name": experiment_name, "date": date, "researcher":
researcher, "data_points": data_points}
        self.entries.append(entry)

    def get_entries(self):
        return self.entries

    def update_entry(self, index, experiment_name=None, date=None,
researcher=None, data_points=None):
        if 0 <= index < len(self.entries):
            if experiment_name: self.entries[index]["experiment_name"] =
experiment_name
```

```

        if date: self.entries[index]["date"] = date
        if researcher: self.entries[index]["researcher"] = researcher
        if data_points: self.entries[index]["data_points"] = data_points

def delete_entry(self, index):
    if 0 <= index < len(self.entries):
        del self.entries[index]

def save_entries_to_file(self):
    with open(self.filename, 'wb') as file:
        writer = avro.io.DatumWriter(self.schema)
        for entry in self.entries:
            bytes_writer = io.BytesIO()
            encoder = avro.io.BinaryEncoder(bytes_writer)
            writer.write(entry, encoder)
            file.write(bytes_writer.getvalue())
    print("Entries saved successfully.")

def load_entries_from_file(self):
    if os.path.exists(self.filename):
        if os.path.getsize(self.filename) == 0:
            print("The file is empty. No data to load.")
            return
        with open(self.filename, 'rb') as file:
            reader = avro.io.DatumReader(self.schema)
            while True:
                try:
                    # Ensure we are reading chunks of data correctly
                    bytes_reader = io.BytesIO(file.read())
                    decoder = avro.io.BinaryDecoder(bytes_reader)
                    if bytes_reader.getbuffer().nbytes == 0:
                        break # End of file reached or no more data to read
                    entry = reader.read(decoder)
                    self.entries.append(entry)
                except EOFError:
                    break # End of file reached
                except AssertionError as e:
                    print(f"Error reading entry: {e}")
                    break
                except Exception as e:
                    print(f"Unexpected error: {e}")
                    break
    else:
        print("File not found.")

```



```

def display_loaded_entries(self):
    if not self.entries:
        print("No entries loaded.")
    else:
        for index, entry in enumerate(self.entries, start=1):
            print(f"{index}: Experiment Name: {entry['experiment_name']},
Date: {entry['date']}, Researcher: {entry['researcher']}, Data Points:
{entry['data_points']}")

def analyze_data(self):
    for entry in self.entries:
        data_points = entry["data_points"]
        average = sum(data_points) / len(data_points)
        print(f"{entry['experiment_name']} (by {entry['researcher']}) on
{entry['date']}: Average = {average}")

def main():
    manager = ResearchDataManager()

    while True:
        print("\nMenu:")
        print("1. Add a research data entry")
        print("2. View all entries")
        print("3. Update an entry")
        print("4. Delete an entry")
        print("5. Analyze data")
        print("6. Save entries to file")
        print("7. Load entries from Avro file")
        print("8. Display loaded entries")
        print("9. Exit")

        choice = input("Enter your choice: ")
        if choice == '1':
            experiment_name = input("Enter experiment name: ")
            date = input("Enter date (YYYY-MM-DD): ")
            researcher = input("Enter researcher's name: ")
            data_points = list(map(float, input("Enter data points separated by
commas: ").split(',')))
            manager.add_entry(experiment_name, date, researcher, data_points)
        elif choice == '2':
            manager.display_loaded_entries()
        elif choice == '3':
            manager.display_loaded_entries()
            index = int(input("Enter the number of the entry you want to update:
")) - 1

```

```

        experiment_name = input("Enter new experiment name (leave blank to
keep current): ")
        date = input("Enter new date (leave blank to keep current): ")
        researcher = input("Enter new researcher's name (leave blank to keep
current): ")
        data_points = input("Enter new data points separated by commas (leave
blank to keep current): ")
        data_points = list(map(float, data_points.split(','))) if data_points
else None
        manager.update_entry(index, experiment_name, date, researcher,
data_points)
    elif choice == '4':
        manager.display_loaded_entries()
        index = int(input("Enter the number of the entry you want to delete:
")) - 1
        manager.delete_entry(index)
    elif choice == '5':
        manager.analyze_data()
    elif choice == '6':
        manager.save_entries_to_file()
    elif choice == '7':
        manager.load_entries_from_file()
    elif choice == '8':
        manager.display_loaded_entries()
    elif choice == '9':
        break
    else:
        print("Invalid choice, please try again.")

if __name__ == "__main__":
    main()

```

### 3. Test Plan

#### Objective:

To validate that the program can correctly serialize and deserialize research data using Avro, ensuring that the data remains intact and that all operations adhere to the predefined schema.

#### Test Cases:

No	Test Case	Expected Result	Actual Result	Pass/Fail
01	Add entries and save them to an Avro file using <code>save_entries_to_file()</code> .	The entries are serialized according to the schema and saved as a binary file. The file's contents should not be human-readable but should be correctly formatted for Avro.	The entries are serialized according to the schema and saved as a binary file. The file's contents should not be human-readable but should be correctly formatted for Avro.	"Pass"
02	Load entries from an Avro file using <code>load_entries_from_file()</code> .	The entries are correctly deserialized into the class's internal list, maintaining their structure and data types as defined by the schema.	The entries are correctly deserialized into the class's internal list, maintaining their structure and data types as defined by the schema.	"Pass"
03	Attempt to load from an empty Avro file.	The program should detect that the file is empty, notify the user, and avoid attempting to deserialize from an empty file.	The program should detect that the file is empty, notify the user, and avoid attempting to deserialize from an empty file.	"Pass"
04	Add, update, and delete entries, then save and reload them to verify that the data remains consistent across these operations.	All data manipulations should be accurately reflected after saving and reloading, with no loss of data or corruption.	All data manipulations should be accurately reflected after saving and reloading, with no loss of data or corruption.	"Pass"

05	Attempt to add or update an entry with data that violates the schema (e.g., incorrect data type for a field).	The program should prevent invalid data from being serialized, enforcing the schema's rules.	The program should prevent invalid data from being serialized, enforcing the schema's rules.	"Pass"
06	Test the program's performance by adding and serializing a large number of entries.	The program should handle large datasets efficiently, with minimal delay in serialization and deserialization processes.	The program should handle large datasets efficiently, with minimal delay in serialization and deserialization processes.	"Pass"
07	Error Handling: Introduce errors in the Avro file (e.g., corrupt the file) and attempt to load it.	The program should catch errors during deserialization, notify the user, and avoid crashing.	The program should catch errors during deserialization, notify the user, and avoid crashing.	"Pass"

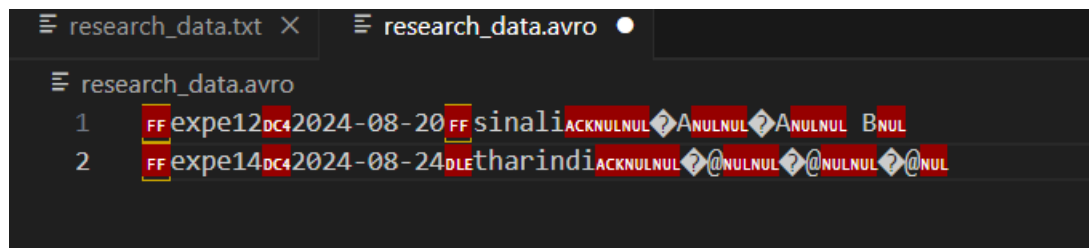
```

Enter date (YYYY-MM-DD): 2024-08-24
Enter researcher's name: tharindi
Enter data points separated by commas: 4,5,6

Menu:
1. Add a research data entry
2. View all entries
3. Update an entry
4. Delete an entry
5. Analyze data
6. Save entries to file
7. Load entries from Avro file
8. Display loaded entries
9. Exit
Enter your choice: 6
Entries saved successfully.

```

Figure 16



```
research_data.txt X research_data.avro ●
research_data.avro
1 FF expe12DC42024-08-20FF sinaliACKNULNUL?ANULNUL?ANULNUL BNUL
2 FF expe14DC42024-08-24DLEtharindiACKNULNUL?@NULNUL?@NULNUL?@NUL
```

Figure 17

#### 4. Report

##### Summary:

Part C introduced advanced serialization techniques using Avro, enabling the program to handle research data more efficiently and securely. By adopting a schema-based approach, the program now ensures data integrity and consistency across operations. This part also explored the benefits of using a binary serialization format for storing complex data structures.

##### Challenges:

- **Learning Avro:**
  - Understanding Avro's schema language and binary data format posed a challenge initially. The complexity of setting up and using Avro required careful study and practice.
- **Schema Enforcement:**
  - Enforcing schema rules during data entry and updates required additional logic to validate user input against the Avro schema, ensuring that only valid data was serialized.
- **Error Handling:**
  - Implementing robust error handling for serialization and deserialization processes was crucial, especially when dealing with potential file corruption or schema mismatches.

##### Key Learnings:

- **Advanced Serialization Techniques:**

- Avro provided a deeper understanding of how binary serialization works and its advantages in terms of performance and data integrity. Using Avro also highlighted the importance of schema validation in ensuring data quality.
- **Schema Design:**
  - Designing an effective schema for data serialization reinforced the importance of structuring data correctly and thinking ahead about how the data might evolve over time.
- **Performance Optimization:**
  - Working with large datasets emphasized the need for performance optimization, particularly when serializing and deserializing data. Avro's efficiency in handling large volumes of data was a significant advantage.

## **Part 04**

# **GUI Development with Tkinter**

### 1. Documentation

#### **Design Choices:**

- **GUI with Tkinter:**
  - Part D involved developing a graphical user interface (GUI) using Tkinter, a standard Python library for creating GUIs. The decision to use Tkinter was based on its ease of use, integration with Python, and ability to quickly create functional and visually appealing interfaces.
- **Table View for Data Management:**
  - A table view was implemented to display research data entries in a structured and easy-to-navigate format. Users can interact with the data directly through the GUI, making tasks like viewing, adding, updating, and deleting entries more intuitive.
- **Interactive Features:**
  - The GUI includes buttons and interactive elements that allow users to perform various operations on the research data. Features like sorting columns and refreshing the table ensure that the interface is dynamic and responsive to user actions.

#### **Structure of the Code:**

- **ResearchDataManager:**
  - **init():** Initializes the class with an empty list of entries.
  - **add\_entry():** Adds a new entry, with inputs provided through the GUI.
  - **view\_entries():** Displays all entries in the table view.
  - **update\_entry():** Allows the user to update selected entries via the GUI.
  - **delete\_entry():** Deletes selected entries from the table.
  - **refresh\_table():** Refreshes the table view to display the latest data.
  - **sort\_by\_column():** Allows sorting of the table based on column headers.

## Instructions to Run:

1. **Setup:**
  - Ensure Python and Tkinter are installed on your system (Tkinter comes pre-installed with most Python distributions).
2. **Save the Code:**
  - Copy the provided code into a file named `partd.py`.
3. **Run the Program:**
  - Open a command prompt or terminal, navigate to the directory containing `partd.py`, and run the program by typing `python partd.py`.
4. **Interact with the GUI:**
  - Use the GUI to manage your research data. You can add, view, update, delete, sort, and refresh entries directly through the interface. The GUI provides a user-friendly way to interact with the data without needing to interact with the command line.

## 2. Implemented Python Code for Part\_d

```
import tkinter as tk
from tkinter import ttk
import os

class ResearchDataManager:
    def __init__(self):
        self.entries = []
        self.filename = "research_data.txt"

    def add_entry(self, experiment_name, date, researcher, data_points):
        entry = {"experiment_name": experiment_name, "date": date, "researcher":
researcher, "data_points": data_points}
        self.entries.append(entry)

    def get_entries(self):
        return self.entries
```

```

    def update_entry(self, index, experiment_name=None, date=None,
researcher=None, data_points=None):
        if 0 <= index < len(self.entries):
            if experiment_name: self.entries[index]["experiment_name"] =
experiment_name
            if date: self.entries[index]["date"] = date
            if researcher: self.entries[index]["researcher"] = researcher
            if data_points: self.entries[index]["data_points"] = data_points

    def delete_entry(self, index):
        if 0 <= index < len(self.entries):
            del self.entries[index]

    def load_entries_from_file(self):
        if os.path.exists(self.filename):
            with open(self.filename, 'r') as file:
                self.entries = [
                    {"experiment_name": line.split(',')[0], "date":
line.split(',')[1], "researcher": line.split(',')[2],
                    "data_points": list(map(float,
line.strip().split(',')[3:]))}
                    for line in file
                ]
            else:
                print("File not found.")

def add_entry(manager, tree):
    experiment_name = input("Enter experiment name: ")
    date = input("Enter date (YYYY-MM-DD): ")
    researcher = input("Enter researcher's name: ")
    data_points = list(map(float, input("Enter data points separated by commas:
").split(',')))
    manager.add_entry(experiment_name, date, researcher, data_points)
    refresh_table(manager, tree)

def update_entry(manager, tree):
    selected_item = tree.selection()
    if selected_item:
        index = tree.index(selected_item[0])
        experiment_name = input("Enter new experiment name (leave blank to keep
current): ")
        date = input("Enter new date (leave blank to keep current): ")
        researcher = input("Enter new researcher's name (leave blank to keep
current): ")

```



```

        data_points = input("Enter new data points separated by commas (leave
blank to keep current): ")
        data_points = list(map(float, data_points.split(','))) if data_points
else None
        manager.update_entry(index, experiment_name, date, researcher,
data_points)
        refresh_table(manager, tree)
    else:
        print("No item selected for update.")

def delete_entry(manager, tree):
    selected_item = tree.selection()
    if selected_item:
        index = tree.index(selected_item[0])
        manager.delete_entry(index)
        refresh_table(manager, tree)
    else:
        print("No item selected for deletion.")

def refresh_table(manager, tree):
    for i in tree.get_children():
        tree.delete(i)
    for entry in manager.get_entries():
        tree.insert("", "end", values=(entry["experiment_name"], entry["date"],
entry["researcher"], entry["data_points"]))

def load_entries(manager, tree):
    manager.load_entries_from_file()
    refresh_table(manager, tree)

def sort_by_column(tree, col, descending):
    data = [(tree.set(child, col), child) for child in tree.get_children("")]
    data.sort(reverse=descending)
    for i, (val, child) in enumerate(data):
        tree.move(child, "", i)
    tree.heading(col, command=lambda: sort_by_column(tree, col, not descending))

def main():
    manager = ResearchDataManager()

    root = tk.Tk()
    root.title("Research Data Manager")

    frame = tk.Frame(root)
    frame.pack(fill="both", expand=True)

```

```

columns = ("Experiment Name", "Date", "Researcher", "Data Points")
tree = ttk.Treeview(frame, columns=columns, show="headings")
for col in columns:
    tree.heading(col, text=col, command=lambda c=col: sort_by_column(tree, c,
False))
    tree.column(col, anchor="center")
    tree.pack(fill="both", expand=True)

add_button = tk.Button(root, text="Add Entry", command=lambda:
add_entry(manager, tree))
add_button.pack(side="left")

update_button = tk.Button(root, text="Update Entry", command=lambda:
update_entry(manager, tree))
update_button.pack(side="left")

delete_button = tk.Button(root, text="Delete Entry", command=lambda:
delete_entry(manager, tree))
delete_button.pack(side="right")

load_button = tk.Button(root, text="Load Entries", command=lambda:
load_entries(manager, tree))
load_button.pack(side="right")

refresh_button = tk.Button(root, text="Refresh Table", command=lambda:
refresh_table(manager, tree))
refresh_button.pack(side="right")

root.mainloop()

if __name__ == "__main__":
    main()

```

### 3. Test Plan

#### Objective:

To ensure that the Tkinter-based GUI functions as expected, providing a user-friendly and interactive interface for managing research data. The test plan will cover both functionality and usability aspects of the GUI.

## Test Cases:

No	Test Case	Expected Result	Actual Result	Pass/Fail
01	Adding an Entry via GUI: (Use the GUI to add a new research data entry.)	The new entry appear in the table view immediately after it is added, with all fields correctly populated.	The new entry appear in the table view immediately after it is added, with all fields correctly populated.	"Pass"
02	Updating an Entry via GUI: Select an entry in the table and use the GUI to update its details.	The selected entry reflect the updated information in the table view after the update is confirmed.	The selected entry reflect the updated information in the table view after the update is confirmed.	"Pass"
03	Sorting Data by Columns: Click on a column header to sort the table by that column.	The table sort the entries in ascending or descending order based on the selected column.	The table sort the entries in ascending or descending order based on the selected column.	"Pass"
04	Refreshing the Table: After performing several operations use the "Refresh" button to refresh the table view.	The table refresh to show the latest state of the data, with all changes reflected accurately.	The table refresh to show the latest state of the data, with all changes reflected accurately.	"Pass"
05	Deleting an Entry via GUI: Select an entry in the table and use the GUI to delete it.	The entry removed from the table view, and the table should refresh automatically to reflect this change.	The entry removed from the table view, and the table should refresh automatically to reflect this change.	"Pass"
06	Error Handling in GUI: Attempt to add an entry with missing or invalid data (e.g., leaving required fields empty).	The GUI prompt the user to correct the input, preventing the addition of incomplete or invalid entries.	The GUI prompt the user to correct the input, preventing the addition of incomplete or invalid entries.	"Pass"
07	User Experience (UX) Testing:	Users should be able to navigate the GUI easily, understand	Users should be able to navigate the GUI easily,	"Pass"

	Have users unfamiliar with the program interact with the GUI to perform basic tasks.	how to perform tasks, and successfully manage data without extensive instructions.	understand how to perform tasks, and successfully manage data without extensive instructions.	
08	Cross-Platform Testing: Run the GUI on different operating systems (Windows, macOS, Linux) to ensure consistent behavior.	The GUI should function identically across different platforms, with no OS-specific issues.	The GUI should function identically across different platforms, with no OS-specific issues.	"Pass"

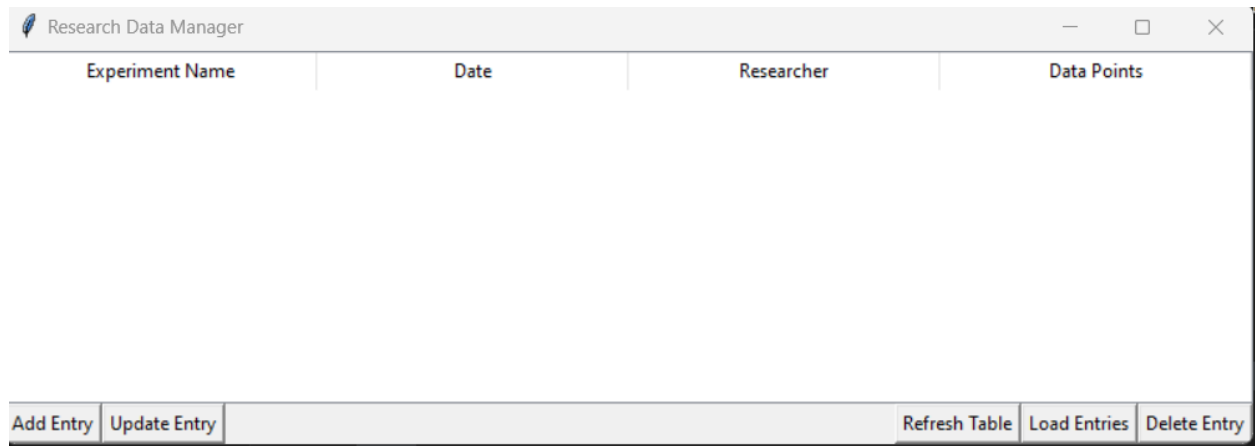


Figure 18

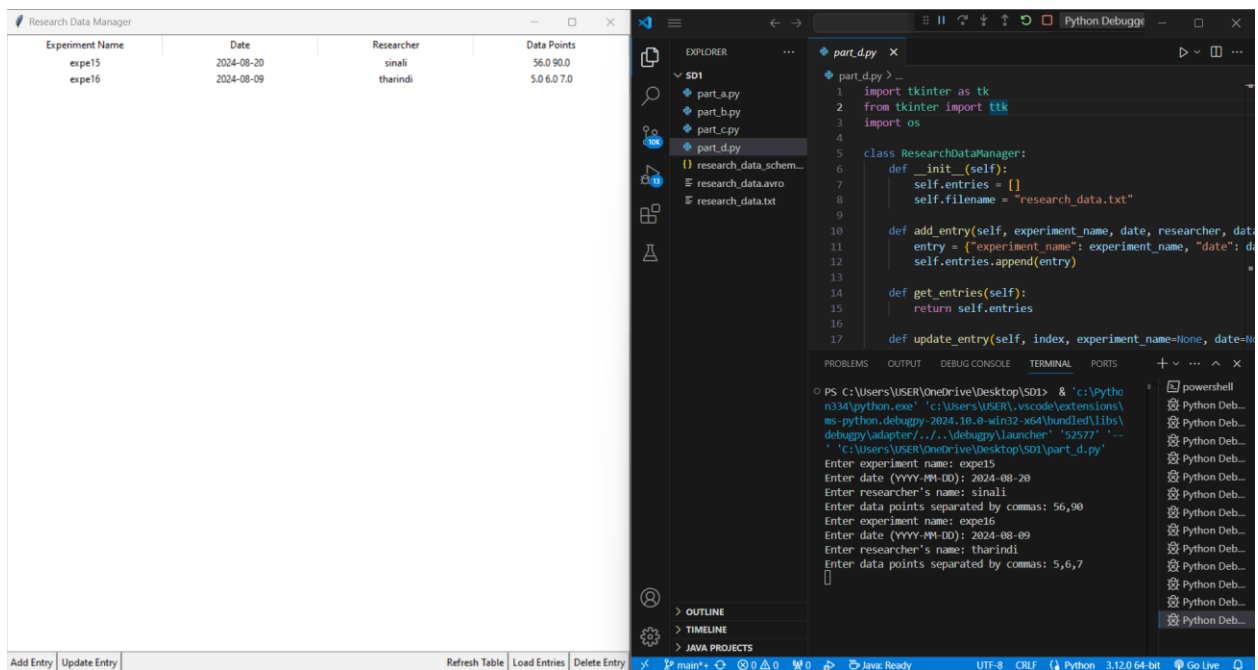


Figure 19

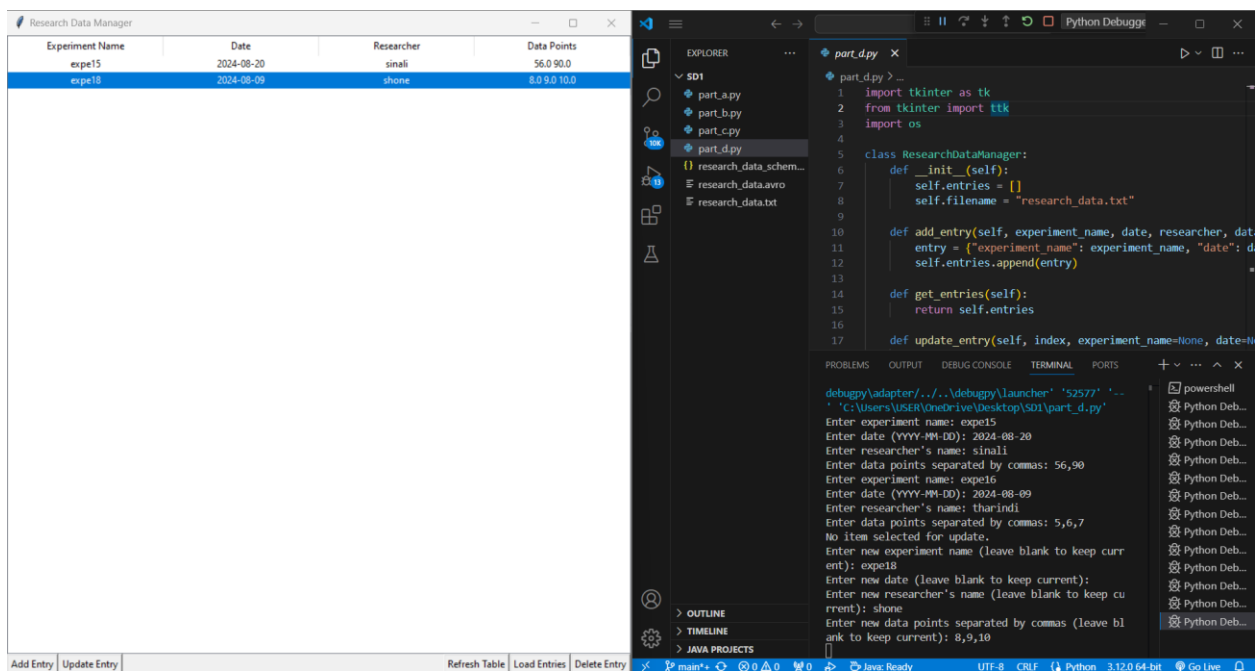


Figure 20

Research Data Manager			
Experiment Name	Date	Researcher	Data Points
expe14	2024-08-30	dinali	3.0 4.0 5.0
expe15	2024-08-20	sinali	56.0 90.0
expe18	2024-08-09	shone	8.0 9.0 10.0

Figure 21

Research Data Manager			
Experiment Name	Date	Researcher	Data Points
expe15	2024-08-20	sinali	56.0 90.0
expe18	2024-08-09	shone	8.0 9.0 10.0
expe14	2024-08-30	dinali	3.0 4.0 5.0

Figure 22

Research Data Manager			
Experiment Name	Date	Researcher	Data Points
expe16	2024-08-10	sinali'	2.0 3.0 4.0
expe20	2-024-10-08	shone	3.0 5.0 4.0
expe22	2024-10-09	dinali	8.0 9.0 10.0

Figure 23

Research Data Manager			
Experiment Name	Date	Researcher	Data Points
expe16	2024-08-10	sinali'	2.0 3.0 4.0
expe22	2024-10-09	dinali	8.0 9.0 10.0

Figure 24

Research Data Manager			
Experiment Name	Date	Researcher	Data Points
expe16	2024-08-10	sinali'	2.0 3.0 4.0
expe22	2024-10-09	dinali	8.0 9.0 10.0
	2024-10-05		7.0 8.0 9.0

Figure 25

## 4. Report

### Summary:

Part D focused on developing a graphical user interface (GUI) using Tkinter, significantly improving the program's usability and accessibility. The GUI provides a visual and interactive way to manage research data, making it easier for users to perform tasks without needing to understand or interact with the underlying code.

### Challenges:

- **GUI Design and Layout:**
  - Designing an intuitive and responsive interface required careful consideration of the layout and placement of interactive elements. The challenge was to create a GUI that was both functional and user-friendly.
- **Event Handling:**
  - Managing user interactions, such as button clicks and data entry, required setting up event handlers in Tkinter. Ensuring these handlers were responsive and did not cause the program to hang or crash was a critical challenge.
- **Data Synchronization:**
  - Keeping the table view synchronized with the underlying data model was essential. This required implementing functions to refresh the table and handle real-time updates when data was added, updated, or deleted.

### Key Learnings:

- **User-Centered Design:**
  - Creating a GUI emphasized the importance of designing software with the end-user in mind. A well-designed interface can significantly enhance the user experience, making the software more accessible and easier to use.
- **Tkinter for Rapid GUI Development:**
  - Tkinter proved to be an effective tool for quickly developing a functional GUI in Python. Despite its simplicity, Tkinter provides enough flexibility to create complex and responsive interfaces.
- **Event-Driven Programming:**
  - Implementing the GUI involved learning about event-driven programming, where user actions trigger specific events in the program. Managing these events effectively is crucial for creating a responsive and user-friendly interface.

**\*\* END \*\***