Dallen Teruel

February 25, 2026

IT FDN 110 GP

Assignment 05

https://github.com/Dallenalexander/IntroToProg-Python-Mod05

Advanced Collections and Error Handling

Intro

In Module 05, I learned how to manage more structured data in Python using dictionaries and JSON files to store and persist in information. This module introduced reading from and writing to JSON files, handling user input more safely with structured error handling, and using try-except blocks to prevent program crashes. I also learned how to use GitHub for source control and how PyCharm integrates with GitHub to store and share code. The following sections describe how these concepts were applied to build this program.

Script Header

The program begins with a script header that documents the file's purpose and revision history. The header includes the script title, a brief description of the program's functionality, and a change log identifying the author, date, and nature of the modification. The file is named Assignment05.py to align with the assignment requirements. These header details are included as comments at the top of the script to improve readability and maintainability (Figure 1.1).

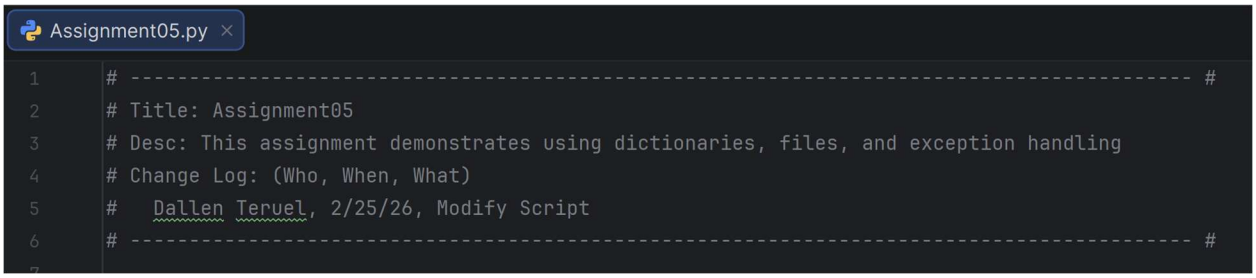


Figure 1.1: Script header comments containing the program title, description, author name, and current date.

Imports

Next, the script includes import statements required to support file handling and JSON processing. First, the statement import json imports Python's built-in JSON module, which enables the program to read data from a JSON file using json.load() and write Python objects to a file using json.dump().

Additionally, the statement from io import TextIOWrapper imports the TextIOWrapper class from Python's io module. This class is used for type hinting and improves code readability when working with file objects (Figure 1.2).

Figure 1.2: Python import statements used to load standard library modules and classes so their functionality can be accessed within the script.

Constants

Two constants are defined to store values that remain unchanged throughout the execution of the program. The first constant, MENU, is a multi-line string that displays the program's menu options to the user:

"'---- Course Registration Program ----

Select from the following menu:

1. Register a Student for a Course
2. Show current data
3. Save data to a file

4. Exit the program

---------- '''

And Triple quotes are used to support formatting across multiple lines. The second constant, FILE_NAME, is set to "Enrollments.json" and specifies the name of the file used to store student registration data. These constants improve maintainability by centralizing configuration values (Figure 1.3).

```
# Define the Data Constants
MENU: str = '''
---- Course Registration Program ----
  Select from the following menu:
    1. Register a Student for a Course.
    2. Show current data.
    3. Save data to a file.
    4. Exit the program.
----------------------------------------
'''
# Define the Data Constants
FILE_NAME: str = "Enrollments.json"
```

Figure 1.3: Constant string values for the program menu and the JSON file name used to store enrollment data.

Variables

Several variables are initialized at the beginning of the script to store user input and program data. The variables student_first_name, student_last_name, course_name, and menu_choice are initialized as empty strings. The variable student_data is initialized as an empty dictionary and is used to store individual student records. The variable students is initialized as an empty list and serves as a collection of all enrollment records.

Additionally, the variable file, typed as TextIOWrapper, is initialized to None. This allows the program to safely call file.close() within a finally block when performing file operations. These variables support data storage and manipulation throughout the program's execution (Figure 1.4).

```
# Define the Data Variables and constants
student_first_name: str = ''  # Holds the first name of a student entered by the user.
student_last_name: str = ''  # Holds the last name of a student entered by the user.
course_name: str = ''  # Holds the name of a course entered by the user.
file: TextIOWrapper = None  # Holds a reference to an opened file. (Changed to use _io.TextIOWrapper instead of None)
menu_choice: str = ''  # Hold the choice made by the user.
student_data: dict = {}  # one row of student data (Changed from list to dictionary)
students: list = []  # a table of student data
```

Figure 1.4: Variable initialization for student names, course name, menu choice, individual student records, the list of all students, and the file object.

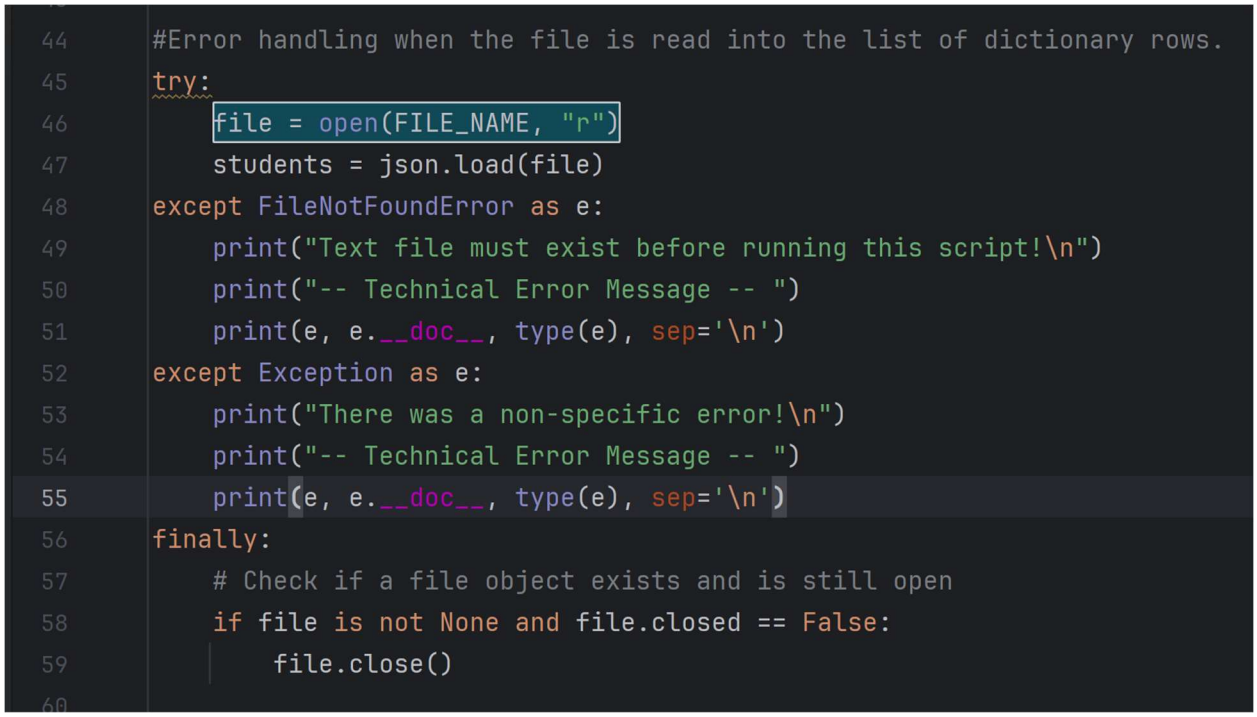
Processing/Error Handling

Next, the program includes structured error handling to safely load existing enrollment data from the JSON file. A try block attempts to open **Enrollments.json** in read mode and load its contents into the students list using json.load().

The program includes exception handling for the following cases:

- **FileNotFoundError**: Executes if the file does not exist and notifies the user.
- **Exception**: Catches any unexpected errors and displays technical details for debugging purposes.
- **Finally**: Executes regardless of whether an exception occurs and ensures that the file is properly closed if it was opened.

This approach prevents the program from crashing due to missing or inaccessible files and ensures proper resource management (Figure 1.5).



```
#Error handling when the file is read into the list of dictionary rows.
try:
    file = open(FILE_NAME, "r")
    students = json.load(file)
except FileNotFoundError as e:
    print("Text file must exist before running this script!\n")
    print("-- Technical Error Message -- ")
    print(e, e.__doc__, type(e), sep='\n')
except Exception as e:
    print("There was a non-specific error!\n")
    print("-- Technical Error Message -- ")
    print(e, e.__doc__, type(e), sep='\n')
finally:
    # Check if a file object exists and is still open
    if file is not None and file.closed == False:
        file.close()
```

Figure 1.5: File input processing with structured exception handling to safely load enrollment data from a JSON file.

Input/Error Handling

A while loop is used to repeatedly display the menu and prompt the user for a selection until the program is exited. When the user selects option 1, the program prompts for the student's first name, last name, and course name.

Validation is applied to user input to ensure data integrity:

- The first and last names are validated using the .isalpha() method to confirm that they contain only alphabetic characters.
- The course name is validated using .isalnum() to ensure that it contains both letters and numbers.

If invalid input is detected, a ValueError is raised with a descriptive message. Once valid input is received, the program creates a dictionary named student_data containing the keys FirstName, LastName, and CourseName. This dictionary represents a single enrollment record and is appended to the students list.

The program includes exception handling to catch validation errors (ValueError) and unexpected runtime errors (Exception), displaying appropriate error messages to the user

(Figure 1.6).

```
# Present and Process the data
while (True):

    # Present the menu of choices
    print(MENU)
    menu_choice = input("What would you like to do: ")

    # Input user data
    # Error handling when the user enters a first name.
    if menu_choice == "1":  # This will not work if it is an integer!
        try:
            # Input the data
            student_first_name = input("Enter the student's first name: ")
            if not student_first_name.isalpha():
                raise ValueError("The first name should not contain numbers.")
    # Error handling when the user enters a last name.
            student_last_name = input("Enter the student's last name: ")
            if not student_last_name.isalpha():
                raise ValueError("The last name should not contain numbers.")

            course_name = input("Please enter the name of the course: ")
            if course_name.isalnum():
                raise ValueError("Course name should contain alpha and numeric values")

            student_data = {"FirstName": student_first_name,
                            "LastName": student_last_name,
                            "CourseName": course_name}
            students.append(student_data)
        except ValueError as e:
            print(e)  # Prints the custom message
            print("-- Technical Error Message -- ")
            print(e.__doc__)
            print(e.__str__())
        except Exception as e:
            print("There was a non-specific error!\n")
            print("-- Technical Error Message -- ")
            print(e, e.__doc__, type(e), sep='\n')
        continue  # the loop
```

Figure 1.6: Infinite loop that processes menu input, validates user data, creates student records, and handles exceptions.

Output

When the user selects option 2, the program displays the current student registration data. For readability, a line of 50 dashes is printed before and after the output. A for loop iterates through the students list and prints each enrollment record in a formatted, comma-separated format that includes the student's first name, last name, and course name

(Figure 1.7).

```
    # Present the current data
    elif menu_choice == "2":

        # Process the data to create and display a custom message
        print("-" * 50)
        for student in students:
            print(f"Student {student['FirstName']},{student['LastName']} is enrolled in {student['CourseName']}")
        print("-" * 50)
        continue
```

Figure 1.7: Output logic that displays all registered students using a loop and formatted print statements.

Processing

When option 3 is selected, the program opens Enrollments.json in write mode. If the file does not exist, it is created; if it does exist, it is overwritten. The students list is then written to the file using json.dump().

The program includes exception handling for:

- TypeError: Catches errors related to invalid JSON data.
- Exception: Catches unexpected file or write-related errors.

After successfully saving the data, the file is closed and a confirmation message is displayed. Selecting option 4 exits the program by breaking out of the loop. An else statement handles invalid menu selections by prompting the user to choose a valid option. Upon exit, a message is displayed indicating that the program has ended (Figure 1.8).

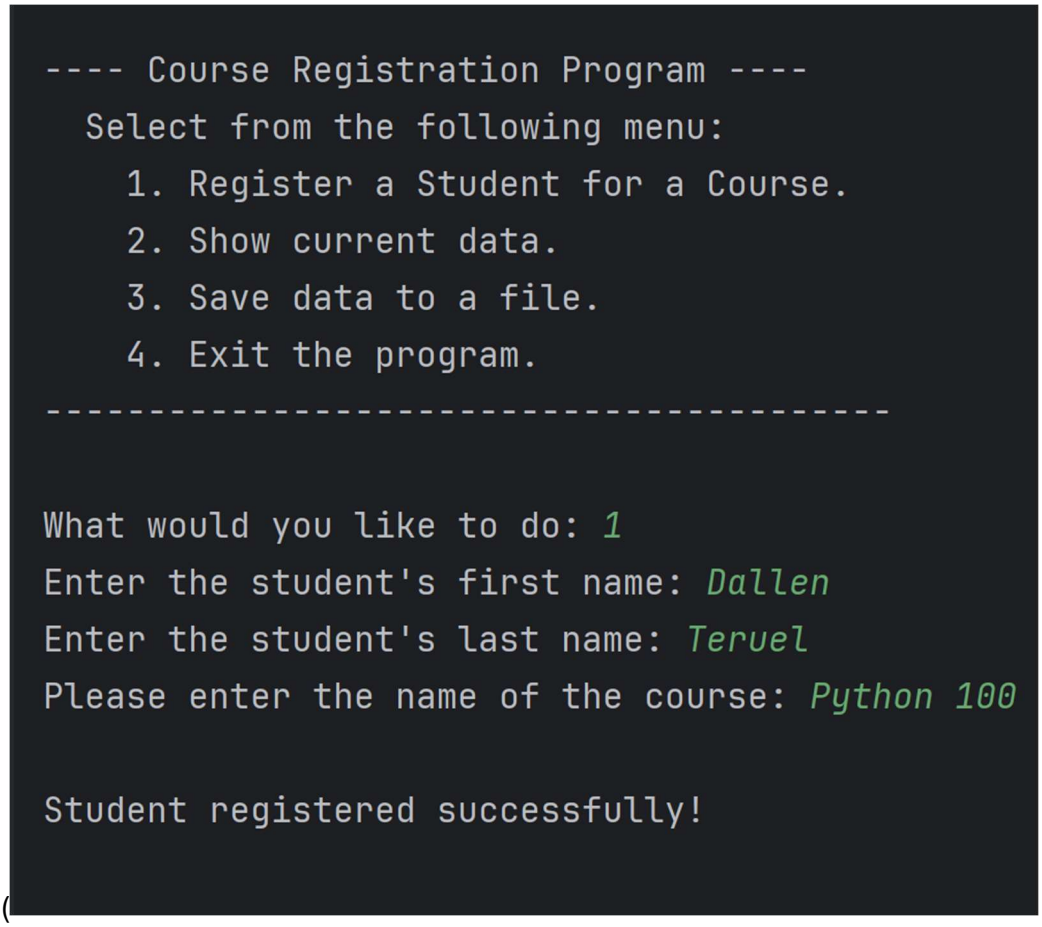
```
112
113     # Save the data to a file
114     # Error handling when the dictionary rows are written to the file.
115     elif menu_choice == "3":
116         try:
117             file = open(FILE_NAME, "w")
118             json.dump(students, file)
119         except TypeError as e:
120             print("Please check that the data is a valid JSON format\n")
121             print("-- Technical Error Message -- ")
122             print(e, e.__doc__, type(e), sep='\n')
123         except Exception as e:
124             print("-- Technical Error Message -- ")
125             print("Built-In Python error info: ")
126             print(e, e.__doc__, type(e), sep='\n')
127         finally:
128             if file is not None and file.closed == False:
129                 file.close()
130         continue # the loop
131
132     elif menu_choice == "4":
133         break  # out of the while loop
```

Figure 1.8: Menu processing logic for saving data, exiting the program, and handling invalid selections.

Test

After implementing constants, variables, input handling, output formatting, processing logic, and error handling, the program was tested to verify correct functionality. When executed, the menu is displayed, allowing the user to register one or more students by selecting option 1. Option 2 correctly displays all registered students. Option 3 successfully saves the enrollment data to Enrollments.json, and option 4 exits the program.

The program was tested using both IDLE and the Windows Command Prompt and executed successfully without errors (Figures 1.9–1.13).



(

```
---- Course Registration Program ----
  Select from the following menu:
    1. Register a Student for a Course.
    2. Show current data.
    3. Save data to a file.
    4. Exit the program.
----------------------------------------

What would you like to do: 1
Enter the student's first name: Vic
Enter the student's last name: Vu
Please enter the name of the course: Python 100

Student registered successfully!
```

```
C:\Users\chq-dallent\Documents\Python\PythonCourse\Module04>python Assignment04.py

---- Course Registration Program ----
  Select from the following menu:
    1. Register a Student for a Course.
    2. Show current data.
    3. Save data to a file.
    4. Exit the program.
----------------------------------------

What would you like to do: 1
Enter the student's first name: Dallen
Enter the student's last name: Teruel
Please enter the name of the course: Python 100

Student registered successfully!
```

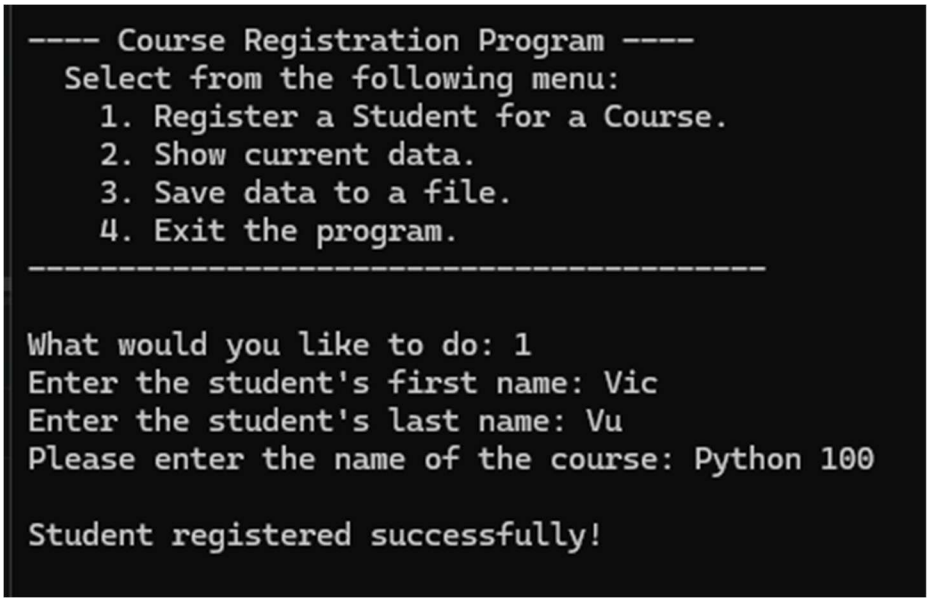
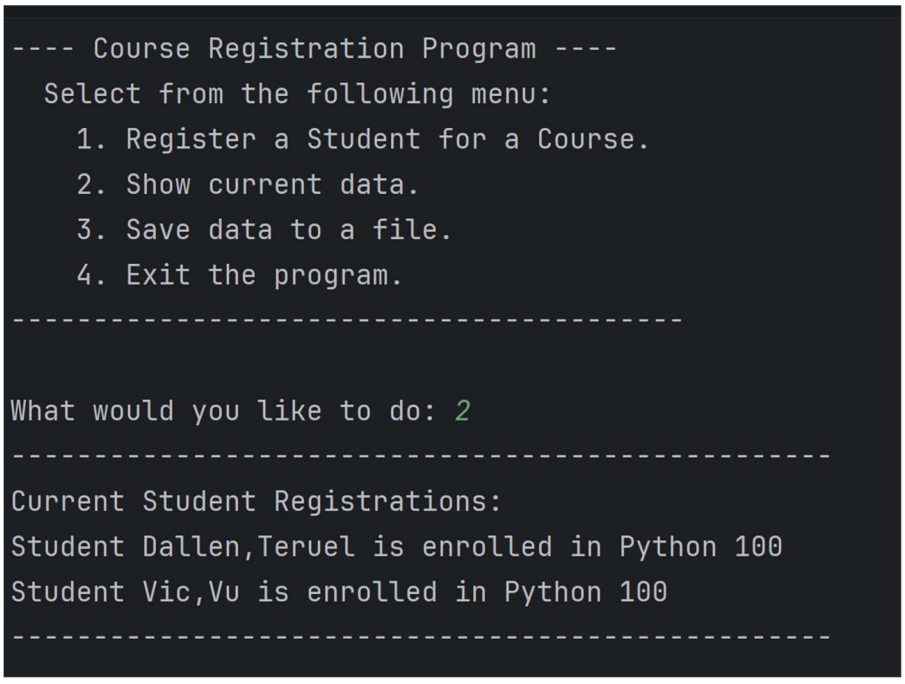


Figure 1.9: The user registers two students for a course. The user first selects menu option 1 and enters each student's first name, last name, and course name to create individual enrollment records.



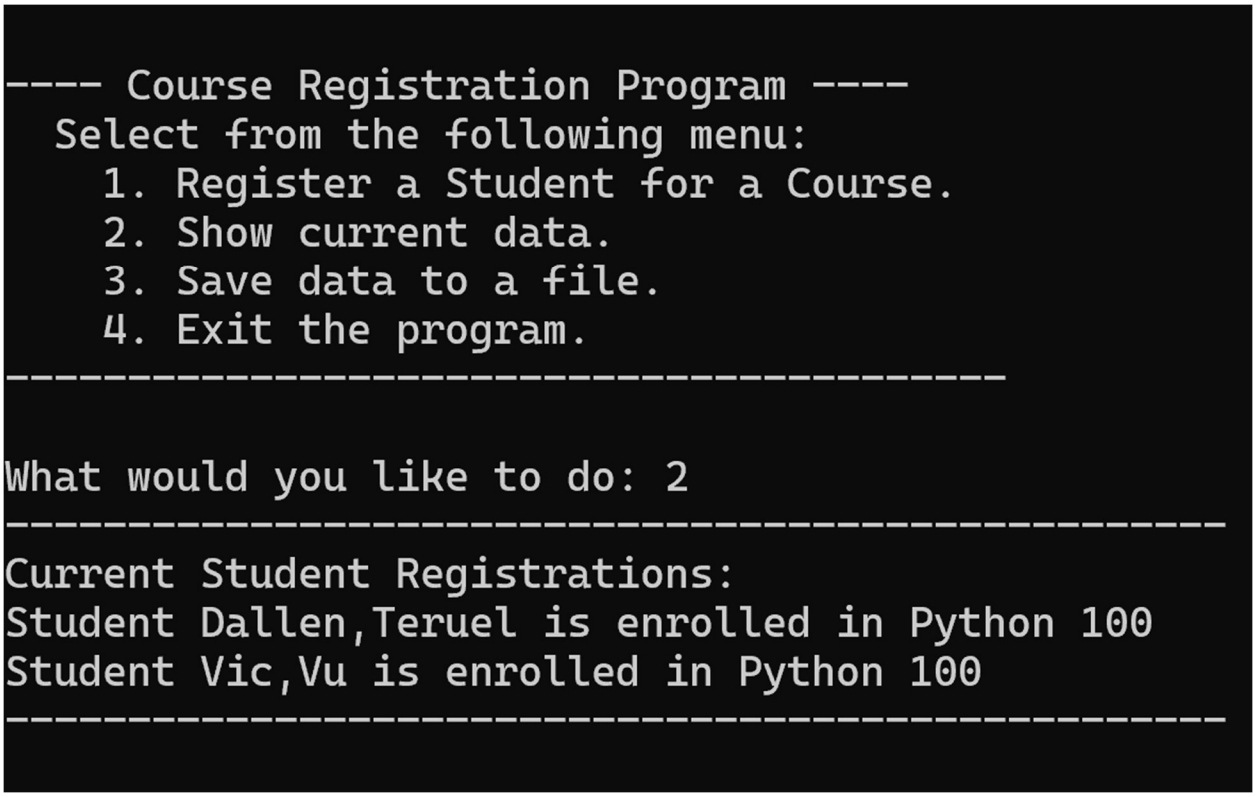


Figure 1.10: The user selects menu option 2 to display the current registration data. The program outputs both student records in a formatted, comma-separated format that includes each student's first name, last name, and course name.

```
---- Course Registration Program ----
  Select from the following menu:
    1. Register a Student for a Course.
    2. Show current data.
    3. Save data to a file.
    4. Exit the program.
----------------------------------------

What would you like to do: 3
--------------------------------------------------
Data Saved!
--------------------------------------------------
```

```
---- Course Registration Program ----
  Select from the following menu:
    1. Register a Student for a Course.
    2. Show current data.
    3. Save data to a file.
    4. Exit the program.
----------------------------------------

What would you like to do: 3
--------------------------------------------------
Data Saved!
--------------------------------------------------
```

Figure 1.11: The user selects menu option 3, which saves the registration data to the Enrollments.json file. A confirmation message indicating that the data was saved successfully is displayed.

```
---- Course Registration Program ----
  Select from the following menu:
    1. Register a Student for a Course.
    2. Show current data.
    3. Save data to a file.
    4. Exit the program.
----------------------------------------

What would you like to do: 5
Please only choose option 1, 2, 3, or 4
```


```
---- Course Registration Program ----
  Select from the following menu:
    1. Register a Student for a Course.
    2. Show current data.
    3. Save data to a file.
    4. Exit the program.
----------------------------------------

What would you like to do: 5
Please only choose option 1, 2, 3, or 4
```


Figure 1.12: The user enters an invalid menu option (5), and the program displays a message instructing the user to choose an option between 1 and 4.

```
---- Course Registration Program ----
  Select from the following menu:
    1. Register a Student for a Course.
    2. Show current data.
    3. Save data to a file.
    4. Exit the program.
----------------------------------------

What would you like to do: 4
Program Ended

Process finished with exit code 0
```

```
---- Course Registration Program ----
  Select from the following menu:
    1. Register a Student for a Course.
    2. Show current data.
    3. Save data to a file.
    4. Exit the program.
----------------------------------------

What would you like to do: 4
Program Ended
```

Figure 1.13: The user selects menu option 4, which exits the program and displays a program termination message.

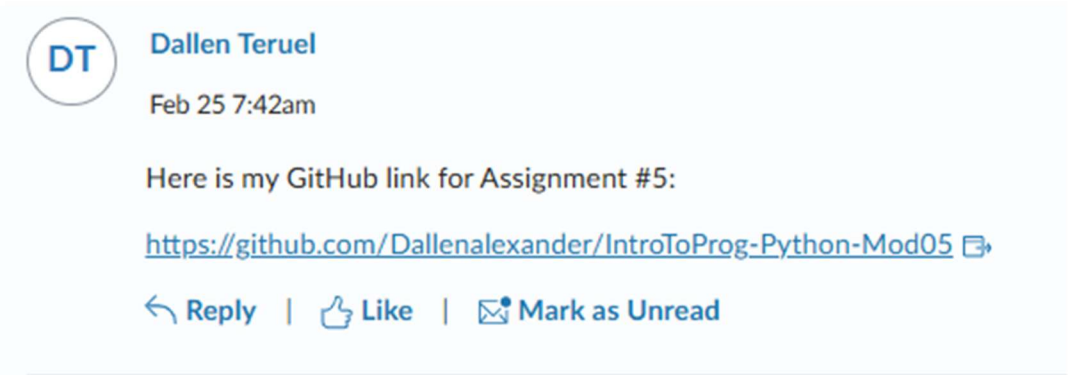
Assignment05.py {} Enrollments.json

```
[
  {
    "FirstName": "Dallen",
    "LastName": "Teruel",
    "CourseName": "Python 100"
  },
  {
    "FirstName": "Vic",
    "LastName": "Vu",
    "CourseName": "Python 100"
  }
]
```

Figure 1.14: The Enrollments.json file displaying both student registration records stored in valid JSON format.

Source Control

- Link to GitHub repository
- Screenshot of link to GitHub repository in GitHub links forum



Summary

In Module 05, I learned how to work with more advanced data handling techniques in Python using dictionaries and JSON files to store, organize, and persist information. I learned how to read from and write to JSON files using the json module, and how to manage program data using key-value pairs instead of lists. This module also introduced structured error handling with try-except-finally blocks, allowing programs to safely handle invalid user input and file errors without crashing. I practiced building programs that load data at startup, process user input, and save updated data back to a file. Overall, this module helped me create more reliable and professional Python programs that handle data safely and support real-world use cases.