Matt Fevergeon
8/17/2025
IT FDN 110 A
Assignment 06
My Git Link

Assignment06 Summary

Program Overview

The program Assignment06.py demonstrates the use of constants, variables, functions, classes, and structured error handling to implement a Course Registration Program. The program allows users to register students for courses, display current registrations, and save the data to a JSON file named Enrollments.json. This implementation builds upon previous assignments by introducing separation of concerns through dedicated classes (FileProcessor and IO) to handle processing and input/output tasks.

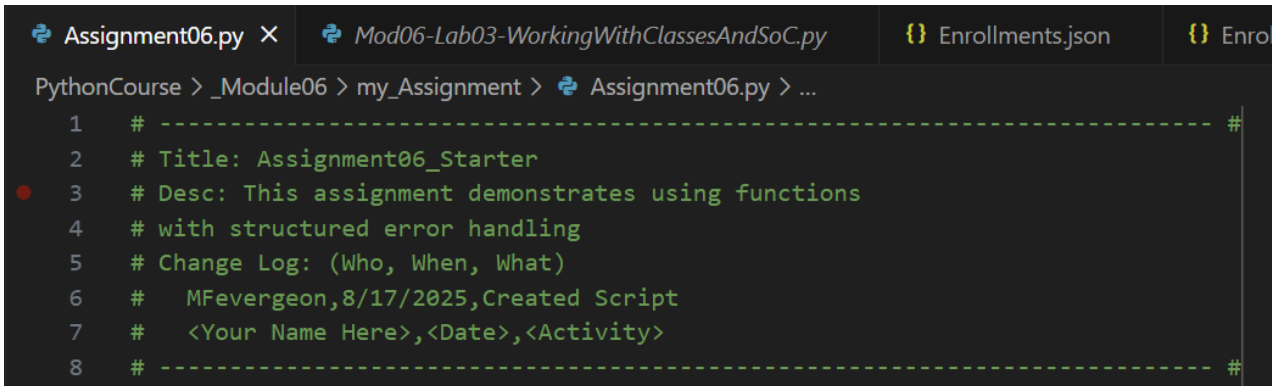
This week felt like a lot of new information, and if I'm honest, I still feel as though I'm struggling with some of the concepts for the separation of concerns and how we're implementing error handling to use common messages. In theory, my program is working well, but I need to do more work and study on implementing this into my own programs.

For Module 5 and 6 I've moved back to the use of VScode. I use this IDE at work, and decided I'd like to get more experience with it in practice. I like the built in copilot functionality, as well as file management. However, I struggled for a decent amount of time getting my scripts to run locally to the same directory as my script. I eventually found the correct settings to get functioning correctly, but it took much longer than I would have preferred.

Summary of how my program meets the acceptance criteria will follow.

Script Header

The script includes the required header with the title, description, changelog, author, and date:

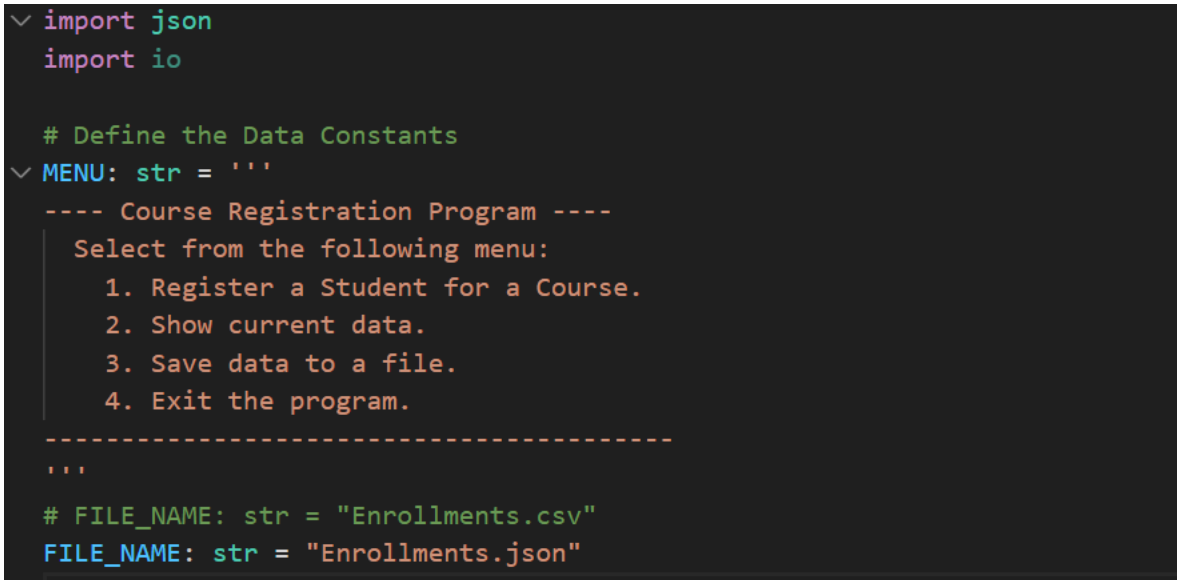

Assignment06.py
Mod06-Lab03-WorkingWithClassesAndSoC.py
Enrollments.json
Enro
PythonCourse > _Module06 > my_Assignment > Assignment06.py > ...

```
# ------------------------------------------------------------------------------------------ #
# Title: Assignment06_Starter
# Desc: This assignment demonstrates using functions
# with structured error handling
# Change Log: (Who, When, What)
#   MFevergeon,8/17/2025,Created Script
#   <Your Name Here>,<Date>,<Activity>
# ------------------------------------------------------------------------------------------ #
```

Figure 1: Script Header

Constants and Variables

The program defines two constants for the MENU and FILE_NAME, as well as the variables as outlined in the Assignment06 document outline. We also define our imports. See figure 2.

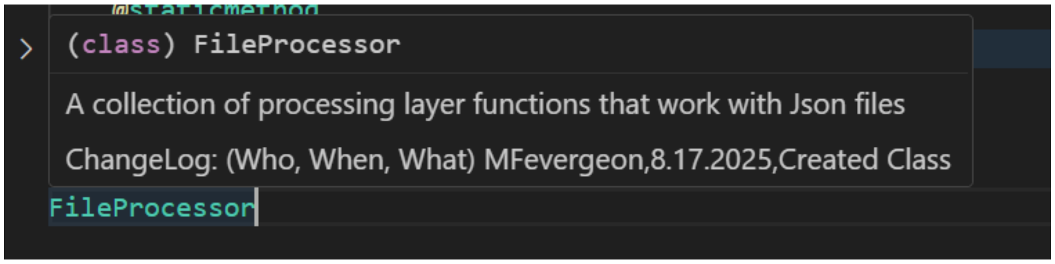


```
import json
import io

# 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.
-----------------------------------------
'''
# FILE_NAME: str = "Enrollments.csv"
FILE_NAME: str = "Enrollments.json"
```

Figure 2: Constants, Variables, and imports

Classes

This week we added the use of classes containing methods (functions) to use in our script to similar tasks to previous weeks. Specifically we define two classes: FileProcessor and IO. These have also been separated out in the script via comments under “Processing” and “Presentation”. The classes include descriptive document strings to support clarity when implementing the classes. Figure 3 shows and example of the popup when calling the Class in the script.



```
(class) FileProcessor

A collection of processing layer functions that work with Json files
ChangeLog: (Who, When, What) MFevergeon,8.17.2025,Created Class
FileProcessor
```

Figure 3: Class Descriptor popup

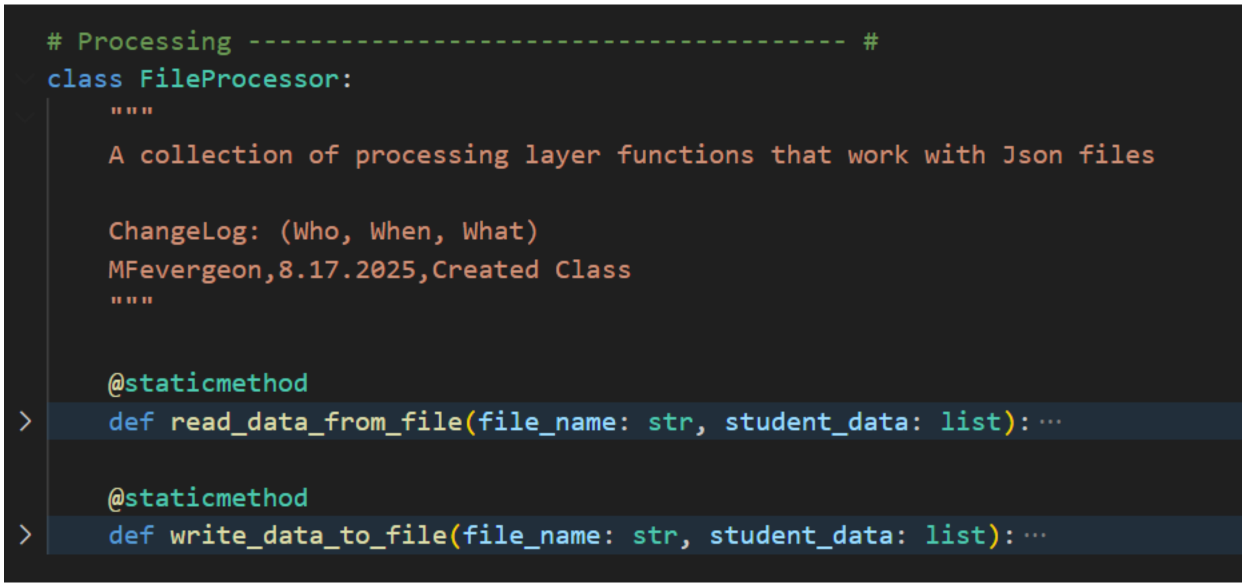
Class: FileProcessor

The FileProcessor class manages reading and writing student registration data to/from the JSON file via two separate functions/methods:

- read_data_from_file(file_name: str, student_data: list):
    - Reads existing student data from Enrollments.json.

    - Handles file-not-found and other exceptions gracefully.
- write_data_to_file(file_name: str, student_data: list)
    - Saves all student registration data back to Enrollments.json.
    - Uses structured error handling for data and file errors.

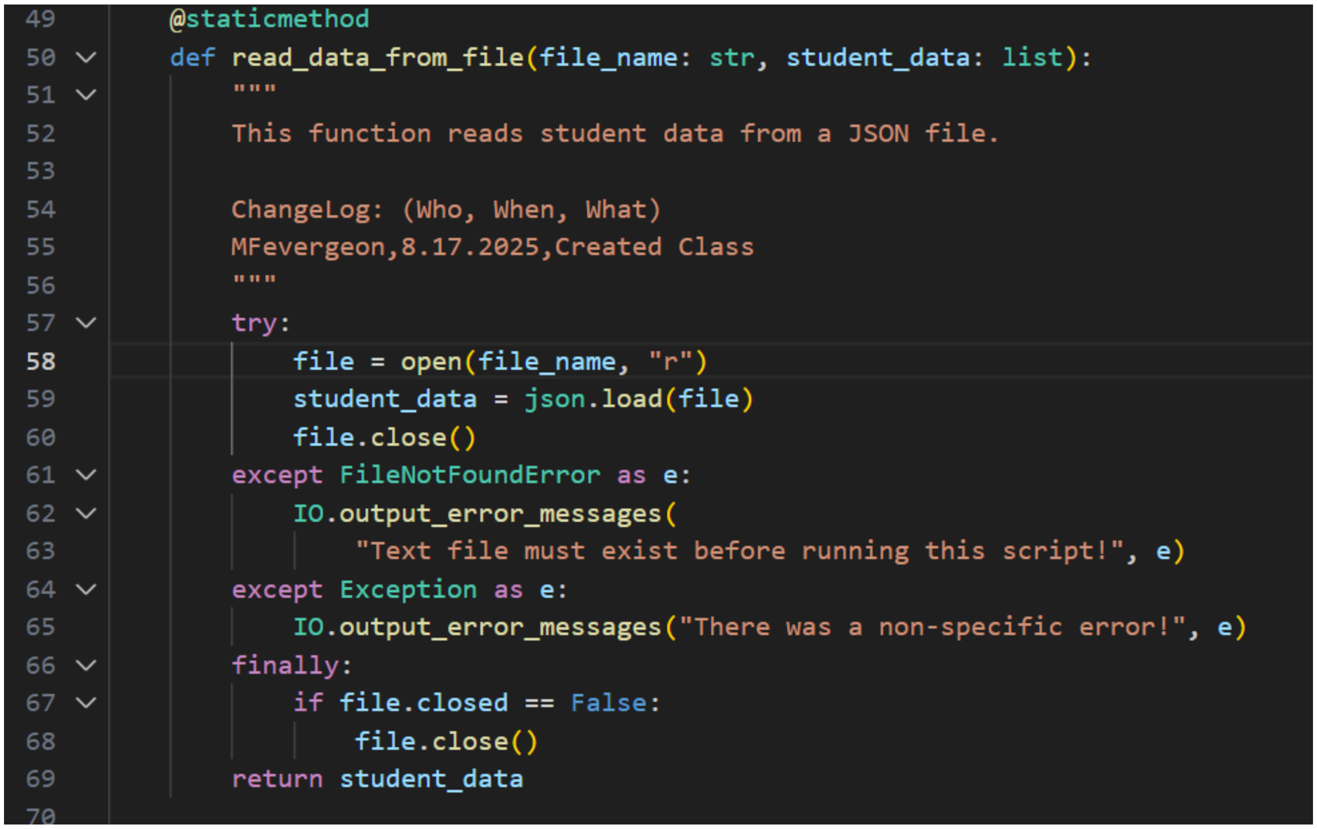
See figure 4 for the collapsed definitions, noting the call for @staticmethod before each definition. The functions also define the type of expected parameters for the functions in ( ). Both functions contain error handling (for both reading and writing to the JSON file per the acceptance criteria. This meets all acceptance criteria as pertaining to the FileProcessor Class.



```
# Processing --------------------------------------- #
class FileProcessor:
    """
    A collection of processing layer functions that work with Json files

    ChangeLog: (Who, When, What)
    MFevergeon,8.17.2025,Created Class
    """

    @staticmethod
    def read_data_from_file(file_name: str, student_data: list):···

    @staticmethod
    def write_data_to_file(file_name: str, student_data: list):···
```

Figure 4: FileProcessor Class definition

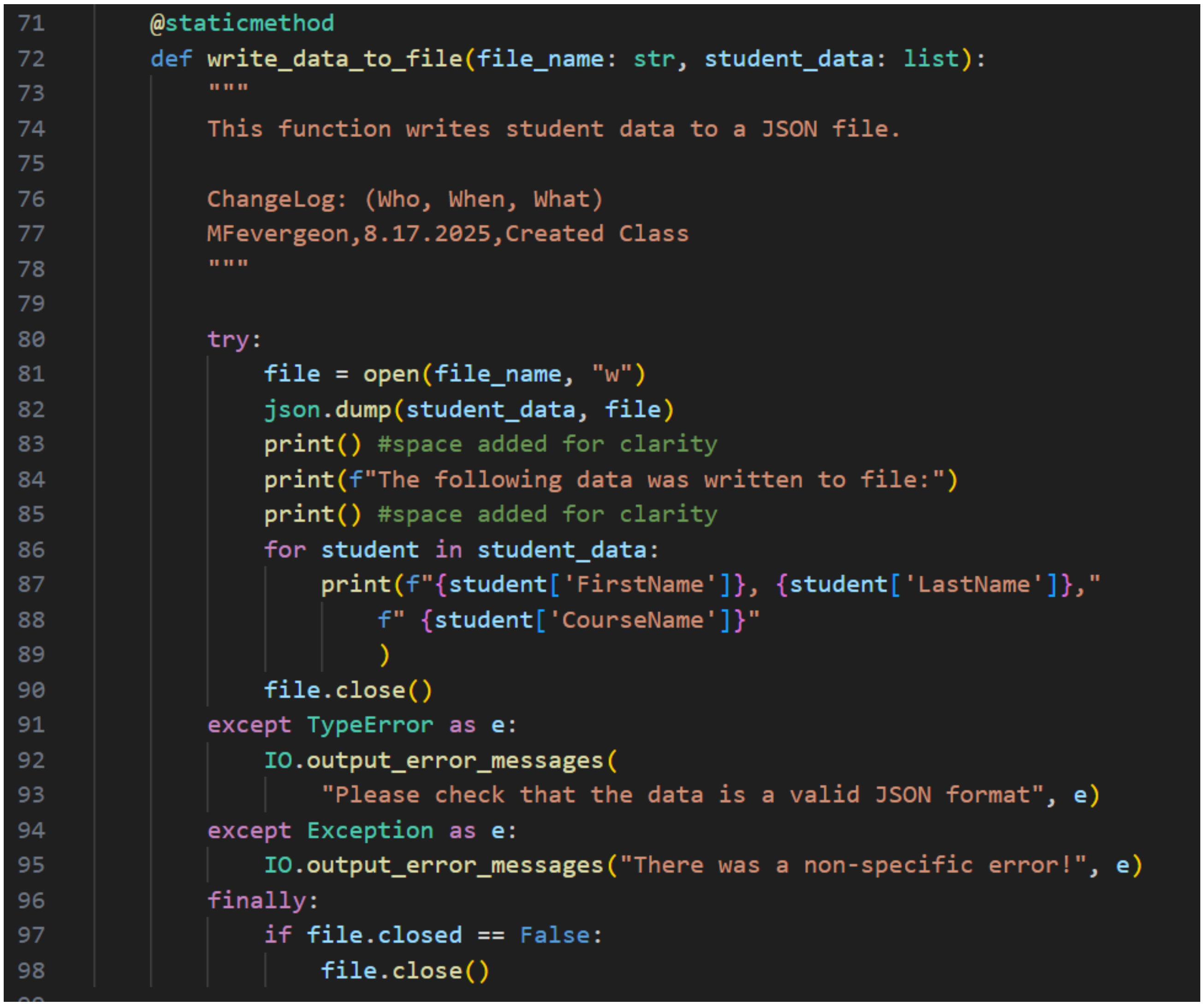
The read_data_from_file static method in the FileProcessor class is responsible for loading student registration data from a specified JSON file into a Python list. It attempts to open the file in read mode, uses json.load() to parse the file contents into the student_data list, and then closes the file. If the file does not exist, it calls a custom error message function to inform the user. Any other exceptions are also caught and reported. The method ensures the file is closed properly in all cases and returns the loaded student data for use elsewhere in the program.



```
    @staticmethod
    def read_data_from_file(file_name: str, student_data: list):
        """
        This function reads student data from a JSON file.

        ChangeLog: (Who, When, What)
        MFevergeon,8.17.2025,Created Class
        """
        try:
            file = open(file_name, "r")
            student_data = json.load(file)
            file.close()
        except FileNotFoundError as e:
            IO.output_error_messages(
                "Text file must exist before running this script!", e)
        except Exception as e:
            IO.output_error_messages("There was a non-specific error!", e)
        finally:
            if file.closed == False:
                file.close()
        return student_data
```

Figure 4.1: read_data_from_file method

The write_data_to_file static method in the FileProcessor class handles saving the current list of student registration data to a specified JSON file. It opens the file in write mode and uses json.dump() to serialize and write the list of student dictionaries to the file. After writing, it prints a formatted message to the user, displaying each student's first name, last name, and course name on a new line. The method includes structured error handling for type errors and other exceptions, providing user-friendly feedback, and ensures the file is closed properly even if an error occurs.



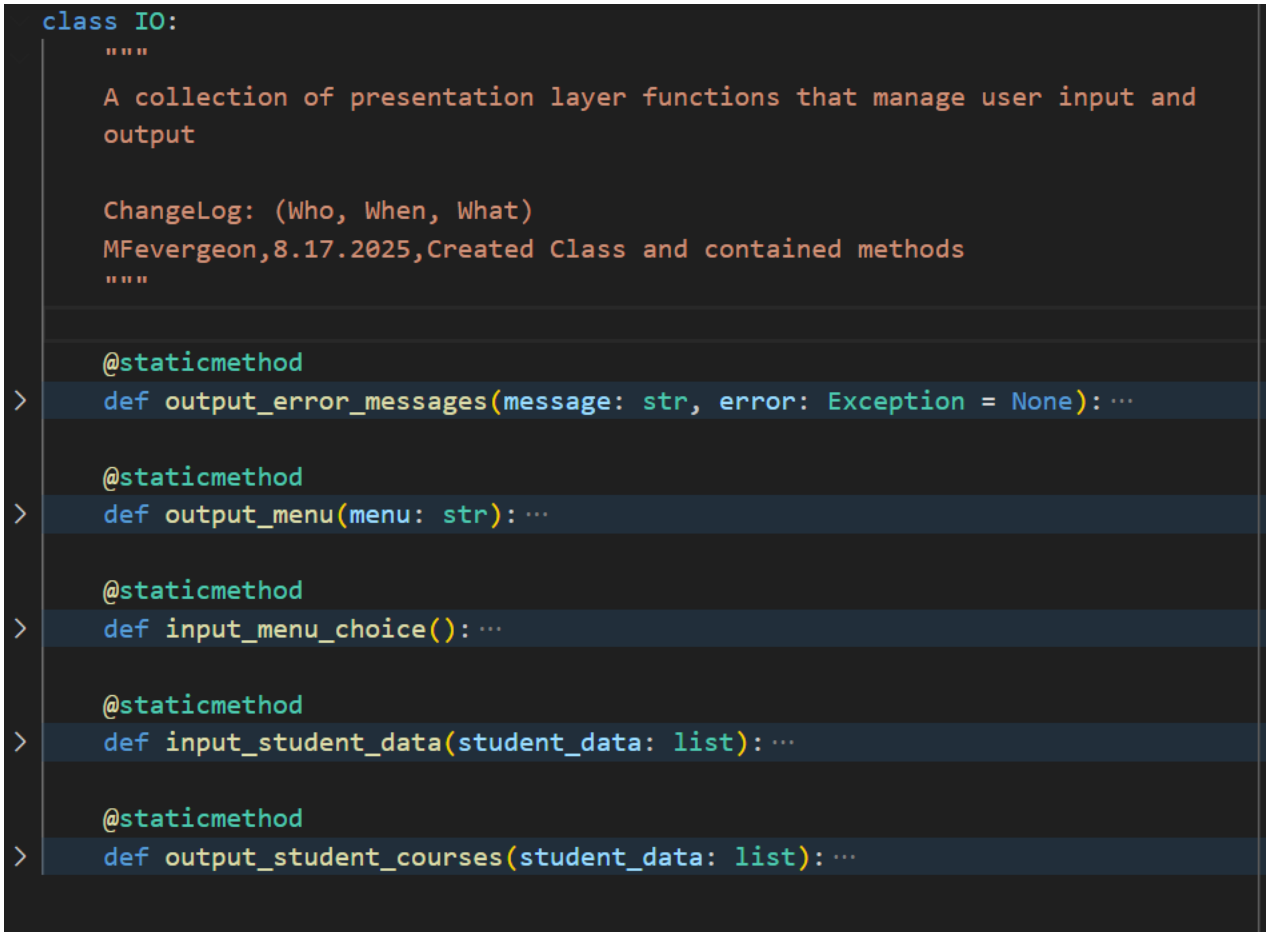
```
    @staticmethod
    def write_data_to_file(file_name: str, student_data: list):
        """
        This function writes student data to a JSON file.

        ChangeLog: (Who, When, What)
        MFevergeon,8.17.2025,Created Class
        """

        try:
            file = open(file_name, "w")
            json.dump(student_data, file)
            print() #space added for clarity
            print(f"The following data was written to file:")
            print() #space added for clarity
            for student in student_data:
                print(f"{student['FirstName']}, {student['LastName']},"
                    f" {student['CourseName']}"
                    )
            file.close()
        except TypeError as e:
            IO.output_error_messages(
                "Please check that the data is a valid JSON format", e)
        except Exception as e:
            IO.output_error_messages("There was a non-specific error!", e)
        finally:
            if file.closed == False:
                file.close()
```

Figure 4.2: write_data_to_file method

Class: IO

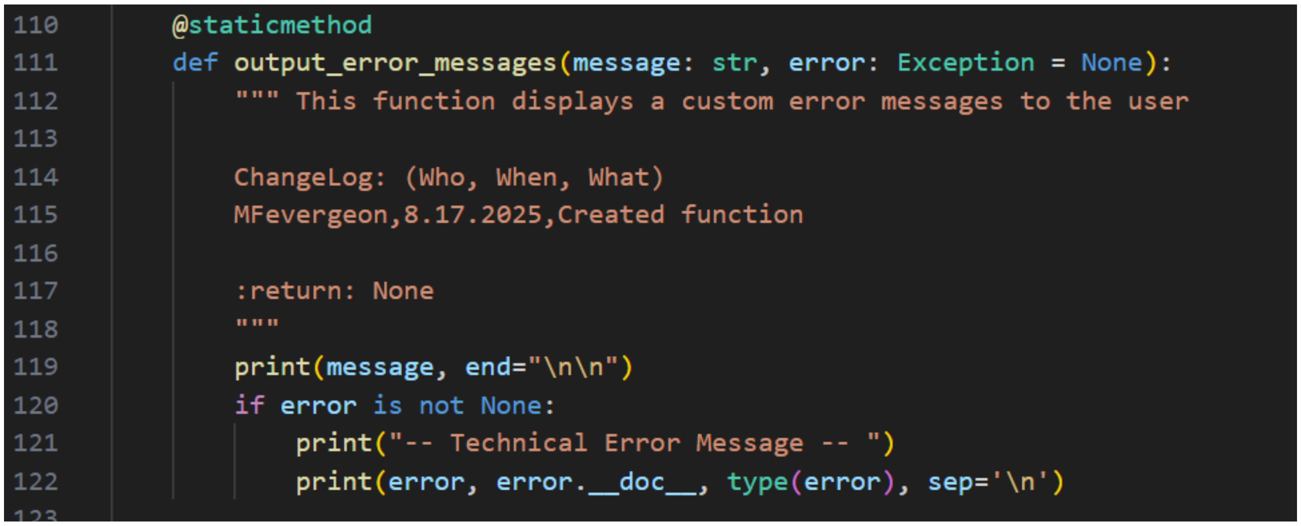
The IO class handles all user interactions and defines the following methods per the acceptance criteria (see figure 5):

- output_error_messages(message: str, error: Exception = None): Displays descriptive error messages.
- output_menu(menu: str): Displays the program menu.
- input_menu_choice(): Accepts validated user input for menu selection.
- input_student_data(student_data: list): Collects and validates student registration info.
- output_student_courses(student_data: list): Displays student registration data.



```python
class IO:
    """
    A collection of presentation layer functions that manage user input and
    output

    ChangeLog: (Who, When, What)
    MFevergeon,8.17.2025,Created Class and contained methods
    """

    @staticmethod
    def output_error_messages(message: str, error: Exception = None):...

    @staticmethod
    def output_menu(menu: str):...

    @staticmethod
    def input_menu_choice():...

    @staticmethod
    def input_student_data(student_data: list):...

    @staticmethod
    def output_student_courses(student_data: list):...
```

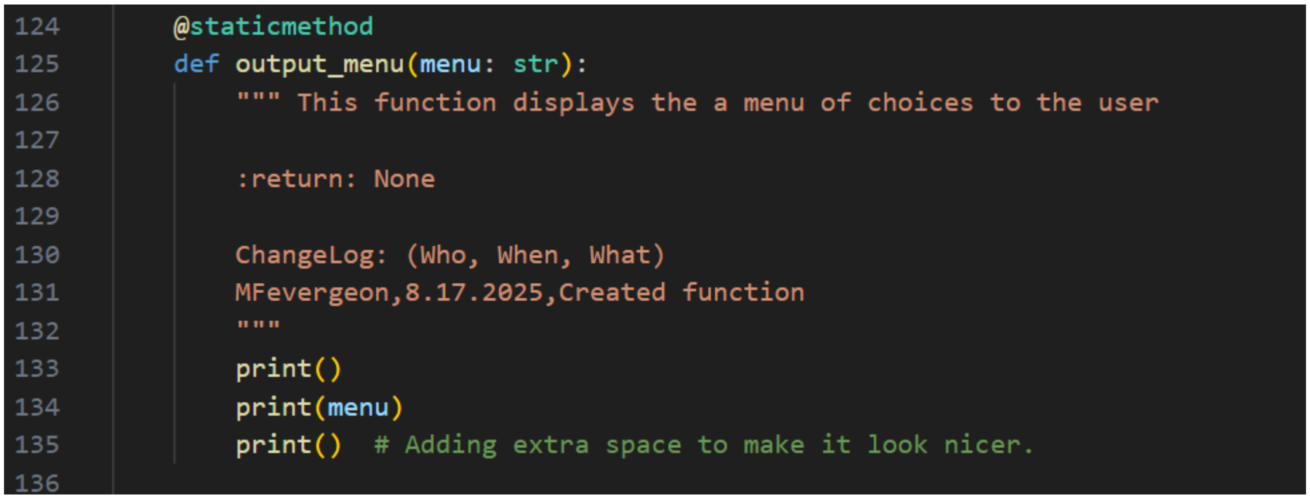
Figure 5: IO Class definition



```python
    @staticmethod
    def output_error_messages(message: str, error: Exception = None):
        """ This function displays a custom error messages to the user

        ChangeLog: (Who, When, What)
        MFevergeon,8.17.2025,Created function

        :return: None
        """
        print(message, end="\n\n")
        if error is not None:
            print("-- Technical Error Message -- ")
            print(error, error.__doc__, type(error), sep='\n')
```

Figure 5.1: output_error_message

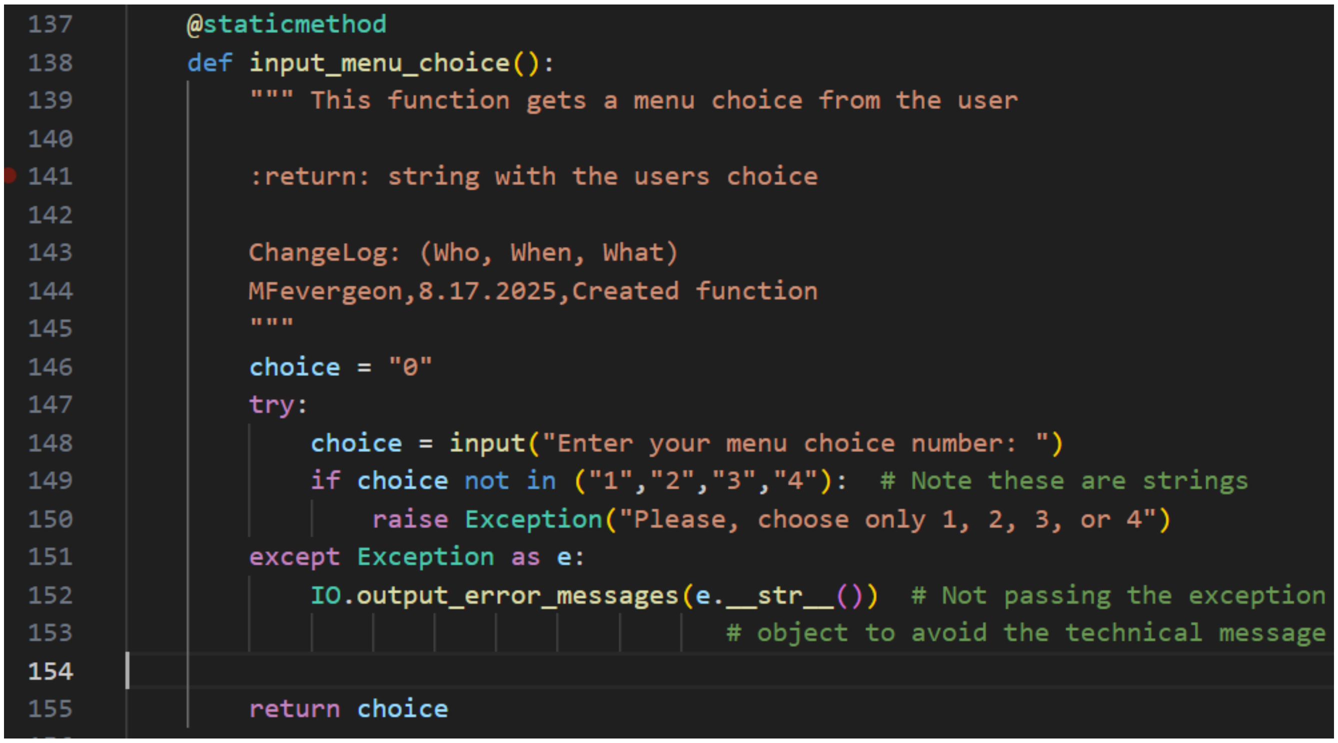
The output_error_messages static method in the IO class is designed to display custom error messages to the user. It prints the provided message and, if an exception object is supplied, also prints technical details about the error, including the exception message, documentation, and type. This helps users and developers understand what went wrong during program execution.



```python
    @staticmethod
    def output_menu(menu: str):
        """ This function displays the a menu of choices to the user

        :return: None

        ChangeLog: (Who, When, What)
        MFevergeon,8.17.2025,Created function
        """
        print()
        print(menu)
        print()  # Adding extra space to make it look nicer.

```

Figure 5.2: output_menu

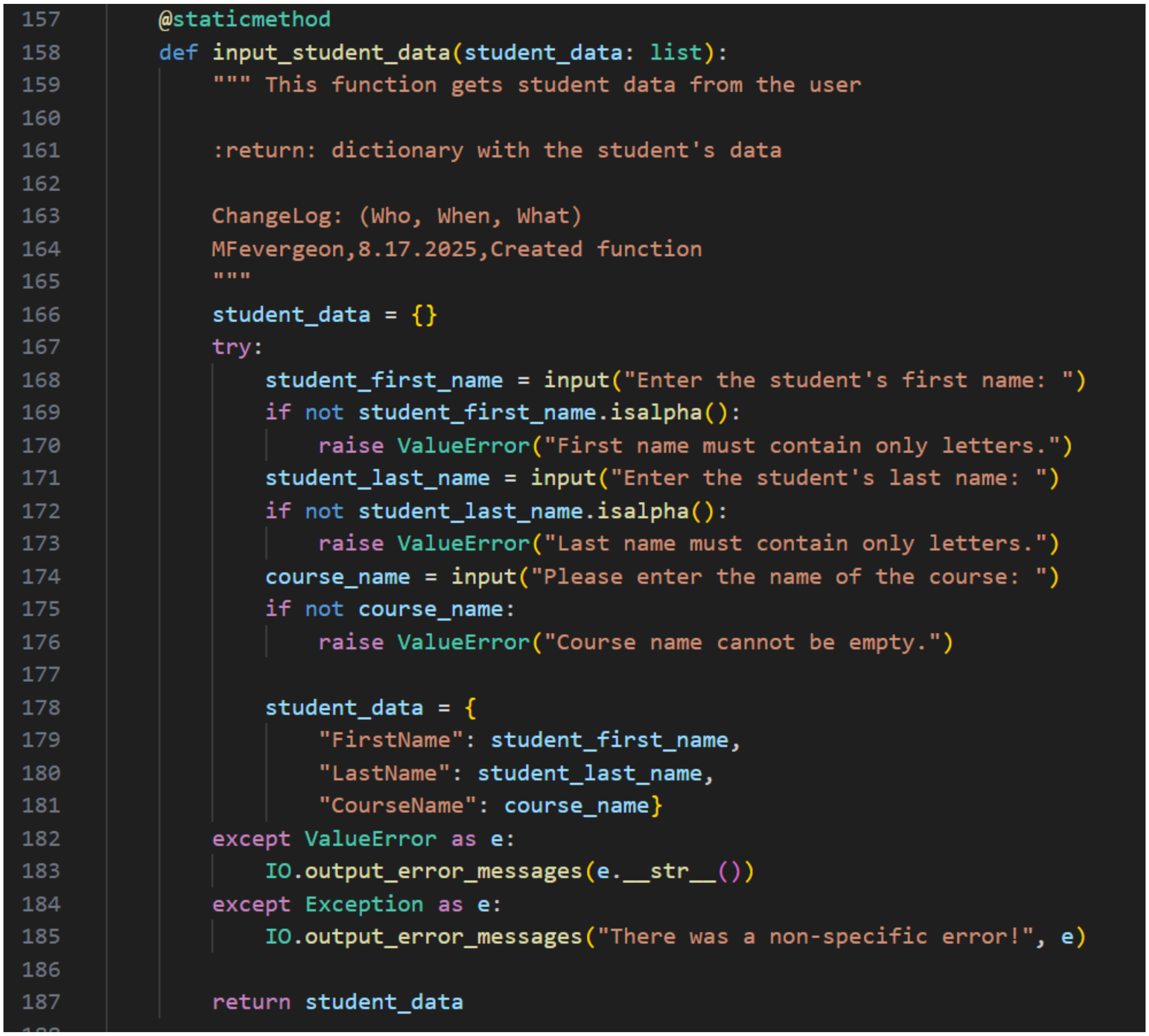
The output_menu static method in the IO class is responsible for displaying the main menu of choices to the user. It prints the menu string, adding extra spacing for readability, so users can easily see their options when interacting with the program.



```python
    @staticmethod
    def input_menu_choice():
        """ This function gets a menu choice from the user

        :return: string with the users choice

        ChangeLog: (Who, When, What)
        MFevergeon,8.17.2025,Created function
        """
        choice = "0"
        try:
            choice = input("Enter your menu choice number: ")
            if choice not in ("1","2","3","4"):  # Note these are strings
                raise Exception("Please, choose only 1, 2, 3, or 4")
        except Exception as e:
            IO.output_error_messages(e.__str__())  # Not passing the exception
                                                   # object to avoid the technical message

        return choice
```

Figure 5.3: input_menu_choice

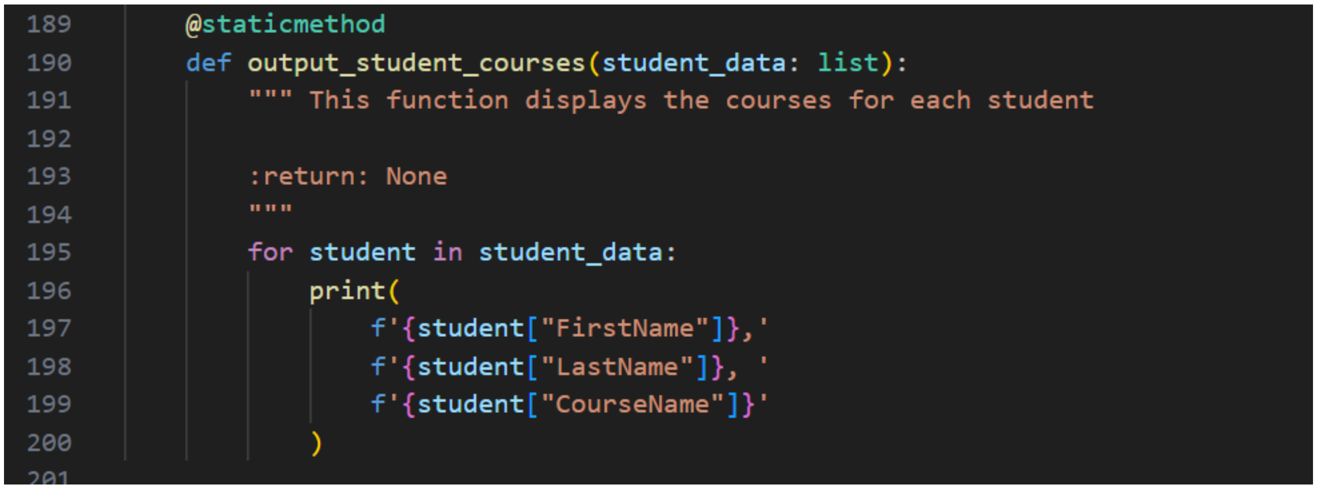
The input_menu_choice static method in the IO class prompts the user to enter a menu choice and validates the input. It ensures that the user's input is one of the valid options ("1", "2", "3", or "4"). If the input is invalid, it raises and handles an exception, displaying a user-friendly error message. The method then returns the user's choice as a string for further processing.



```python
    @staticmethod
    def input_student_data(student_data: list):
        """ This function gets student data from the user

        :return: dictionary with the student's data

        ChangeLog: (Who, When, What)
        MFevergeon,8.17.2025,Created function
        """
        student_data = {}
        try:
            student_first_name = input("Enter the student's first name: ")
            if not student_first_name.isalpha():
                raise ValueError("First name must contain only letters.")
            student_last_name = input("Enter the student's last name: ")
            if not student_last_name.isalpha():
                raise ValueError("Last name must contain only letters.")
            course_name = input("Please enter the name of the course: ")
            if not course_name:
                raise ValueError("Course name cannot be empty.")

            student_data = {
                "FirstName": student_first_name,
                "LastName": student_last_name,
                "CourseName": course_name}
        except ValueError as e:
            IO.output_error_messages(e.__str__())
        except Exception as e:
            IO.output_error_messages("There was a non-specific error!", e)

        return student_data
```

Figure 5.4: input_student_data

The input_student_data static method in the IO class collects student information from the user, including first name, last name, and course name. It validates that the names contain only letters and that the course name is not empty. If the input is invalid, it raises and handles exceptions, providing clear feedback to the user. The method returns a dictionary containing the validated student data.



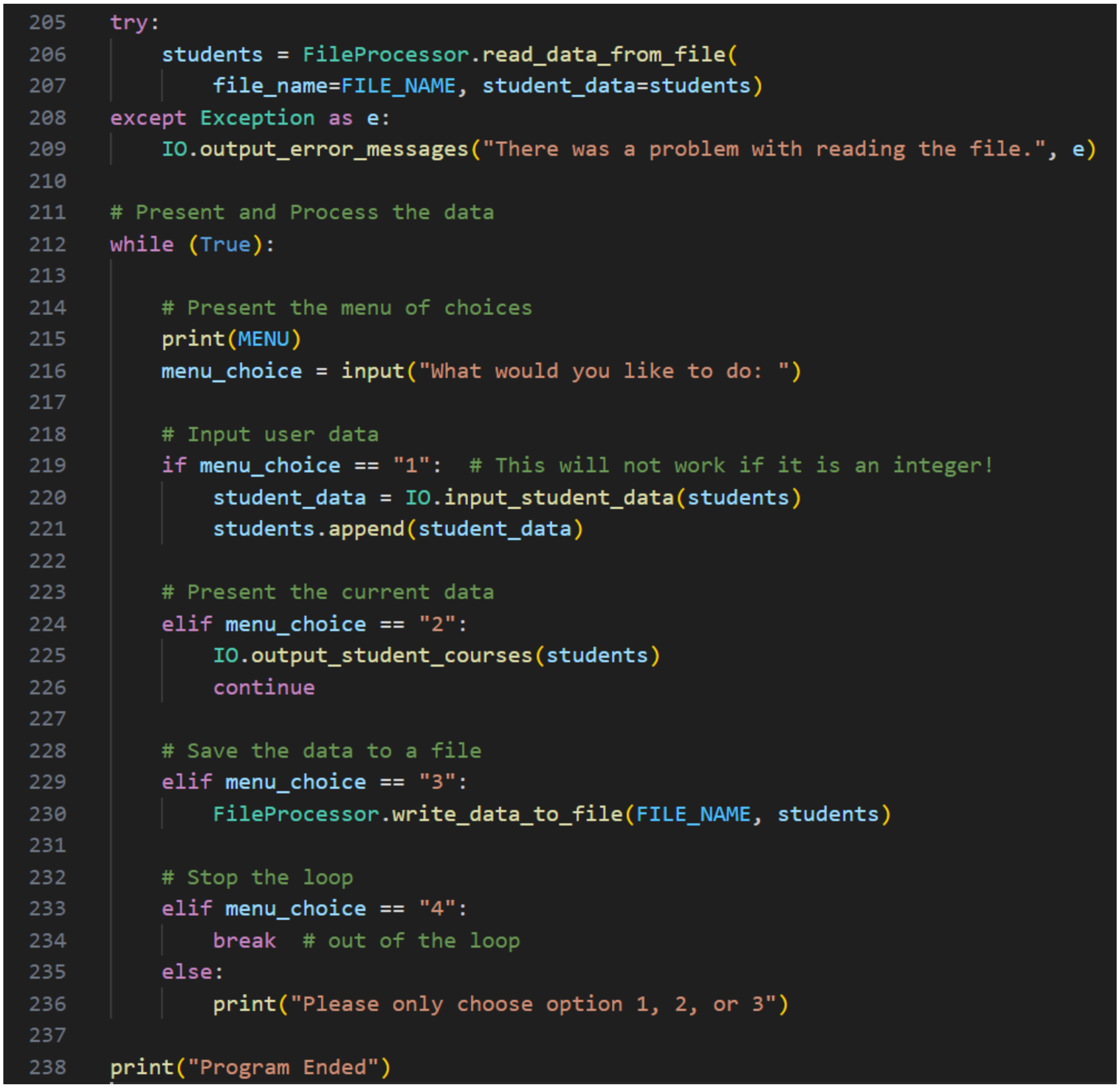
```
    @staticmethod
    def output_student_courses(student_data: list):
        """ This function displays the courses for each student

        :return: None
        """
        for student in student_data:
            print(
                f'{student["FirstName"]},'
                f'{student["LastName"]}, '
                f'{student["CourseName"]}'
            )
```

Figure 5.5: output_student_courses

Finally, the output_student_courses static method in the IO class displays all registered students and their courses. It iterates through the list of student data and prints each student's first name, last name, and course name in a formatted manner, making it easy for users to review the current registrations.

Main Script Body

The main script body begins by attempting to load existing student registration data from a JSON file using the FileProcessor.read_data_from_file method. If an error occurs during this process, a user-friendly error message is displayed. The script then enters an infinite loop that serves as the main menu-driven interface for the program. Each cycle, the menu is displayed and the user is prompted for a choice. If the user selects option 1, they are prompted to enter a new student's information, which is then added to the list of students. Option 2 displays all current student registrations currently stored in memory. Option 3 saves the current list of students to the JSON file using the FileProcessor.write_data_to_file method. Option 4 exits the loop and ends the program. If the user enters an invalid option, an error message prompts them to choose a valid menu option. The script ensures that data is loaded, displayed, added, and saved in a structured and user-friendly manner.



```python
try:
    students = FileProcessor.read_data_from_file(
        file_name=FILE_NAME, student_data=students)
except Exception as e:
    IO.output_error_messages("There was a problem with reading the file.", e)

# 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
    if menu_choice == "1":  # This will not work if it is an integer!
        student_data = IO.input_student_data(students)
        students.append(student_data)

    # Present the current data
    elif menu_choice == "2":
        IO.output_student_courses(students)
        continue

    # Save the data to a file
    elif menu_choice == "3":
        FileProcessor.write_data_to_file(FILE_NAME, students)

    # Stop the loop
    elif menu_choice == "4":
        break  # out of the loop
    else:
        print("Please only choose option 1, 2, or 3")

print("Program Ended")
```

Figure 6: Main Script Body

Testing Results

Please find testing results for the following in the Appendix.

- Appendix A – Shows the program function in VSCode Terminal window. After program start, the program reads data from the Enrollments.json file. Option 2 shows the currently loaded data. Option 1 adds student data (multiple), and then option 2 displays the updated data in memory. Option three then writes to the file and option 4 closes.

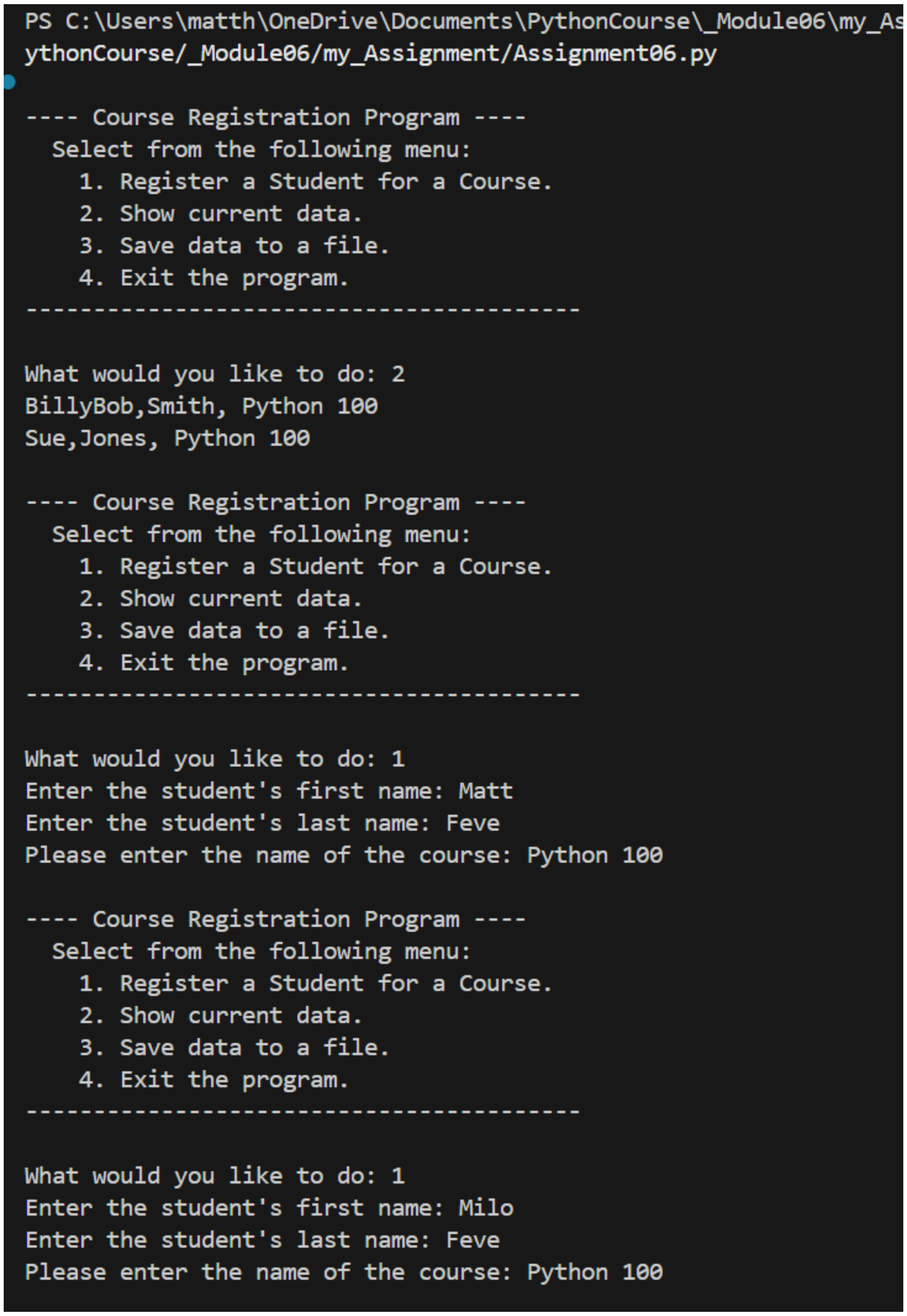
- Appendix B – Shows the .json file before and after program runs.

- Appendix C – Shows program results in command terminal

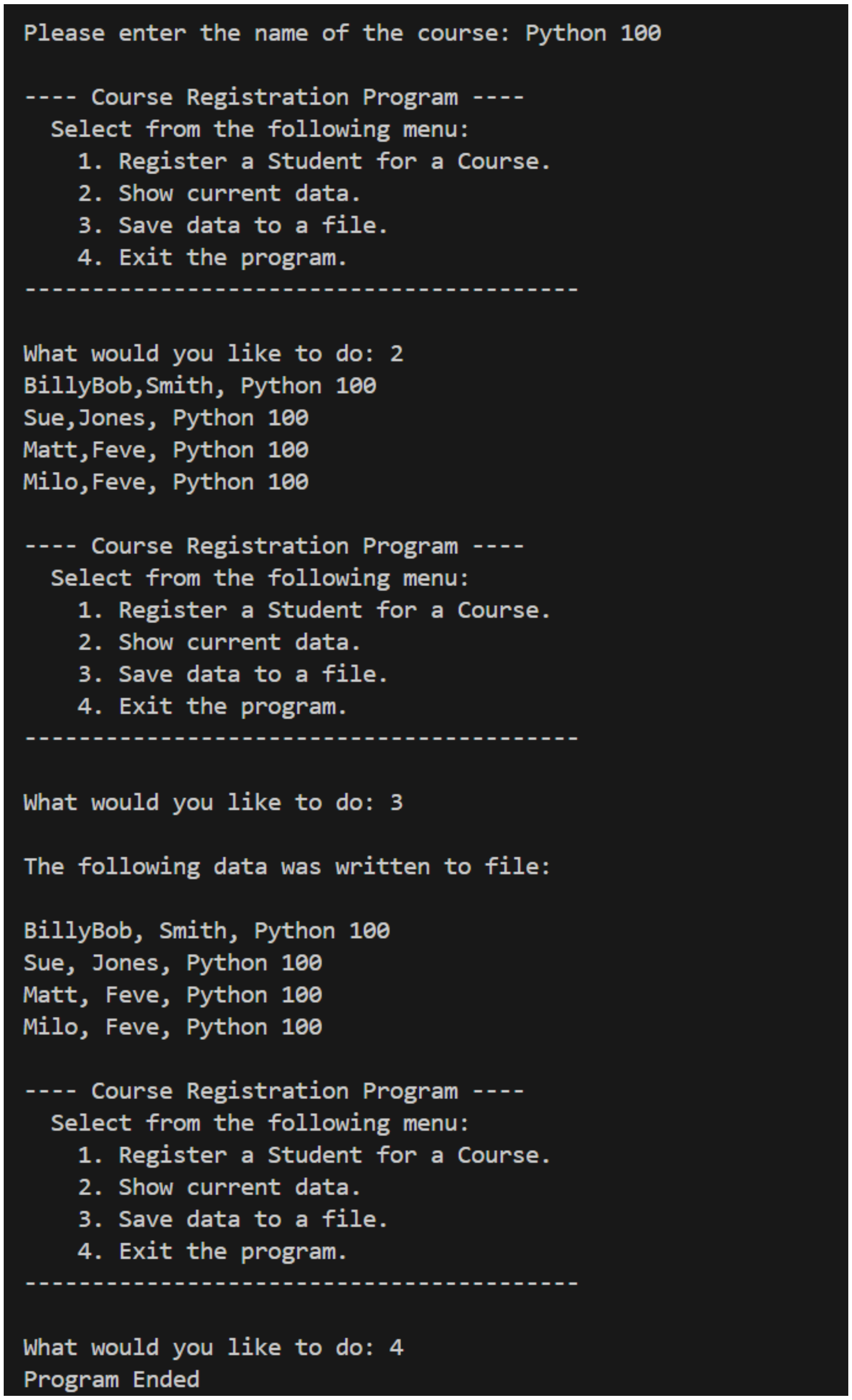
Conclusion

The program Assignment06.py successfully meets all acceptance criteria. It demonstrates constants, variables, functions, classes, separation of concerns, structured error handling, file persistence, and user interaction. The program is complete and fully functional for course registration tasks.

Appendix A – Terminal results



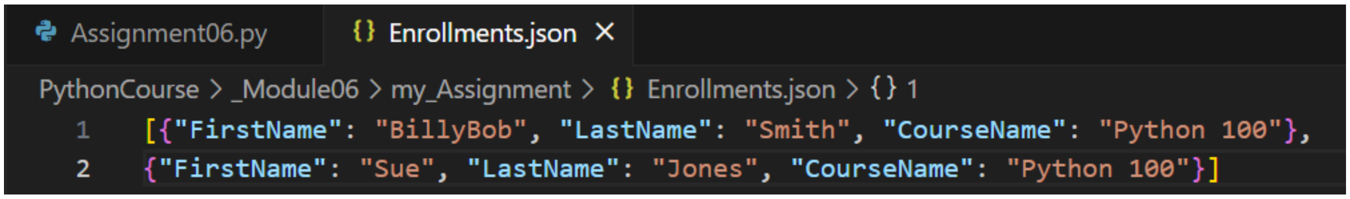
```
PS C:\Users\matth\OneDrive\Documents\PythonCourse\_Module06\my_As
ythonCourse/_Module06/my_Assignment/Assignment06.py

---- 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: 2
BillyBob,Smith, Python 100
Sue,Jones, Python 100

---- 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: Matt
Enter the student's last name: Feve
Please enter the name of the course: Python 100

---- 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: Milo
Enter the student's last name: Feve
Please enter the name of the course: Python 100
```



```
Please enter the name of the course: Python 100

---- 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: 2
BillyBob,Smith, Python 100
Sue,Jones, Python 100
Matt,Feve, Python 100
Milo,Feve, Python 100

---- 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

The following data was written to file:

BillyBob, Smith, Python 100
Sue, Jones, Python 100
Matt, Feve, Python 100
Milo, Feve, Python 100

---- 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
```

Appendix B – Enrollments.json file results

Initial Data

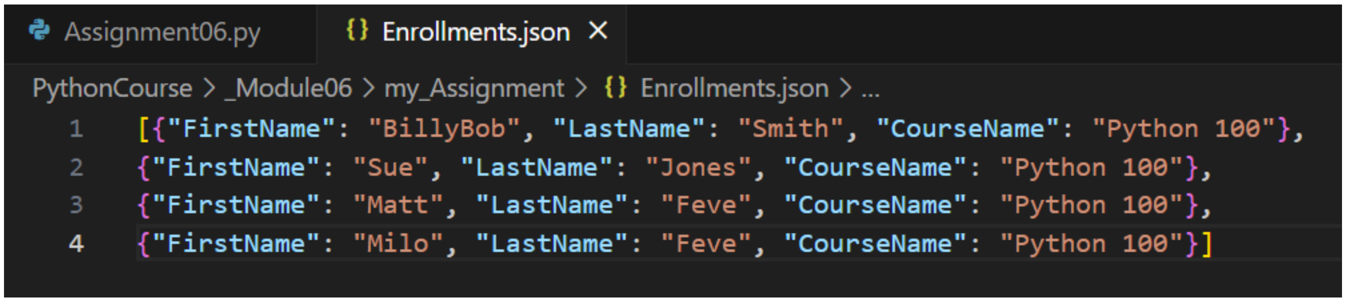


Assignment06.py   {} Enrollments.json ×

PythonCourse > _Module06 > my_Assignment > {} Enrollments.json > {} 1

```
[{"FirstName": "BillyBob", "LastName": "Smith", "CourseName": "Python 100"},
{"FirstName": "Sue", "LastName": "Jones", "CourseName": "Python 100"}]
```

After Program run (new lines added for clarity)

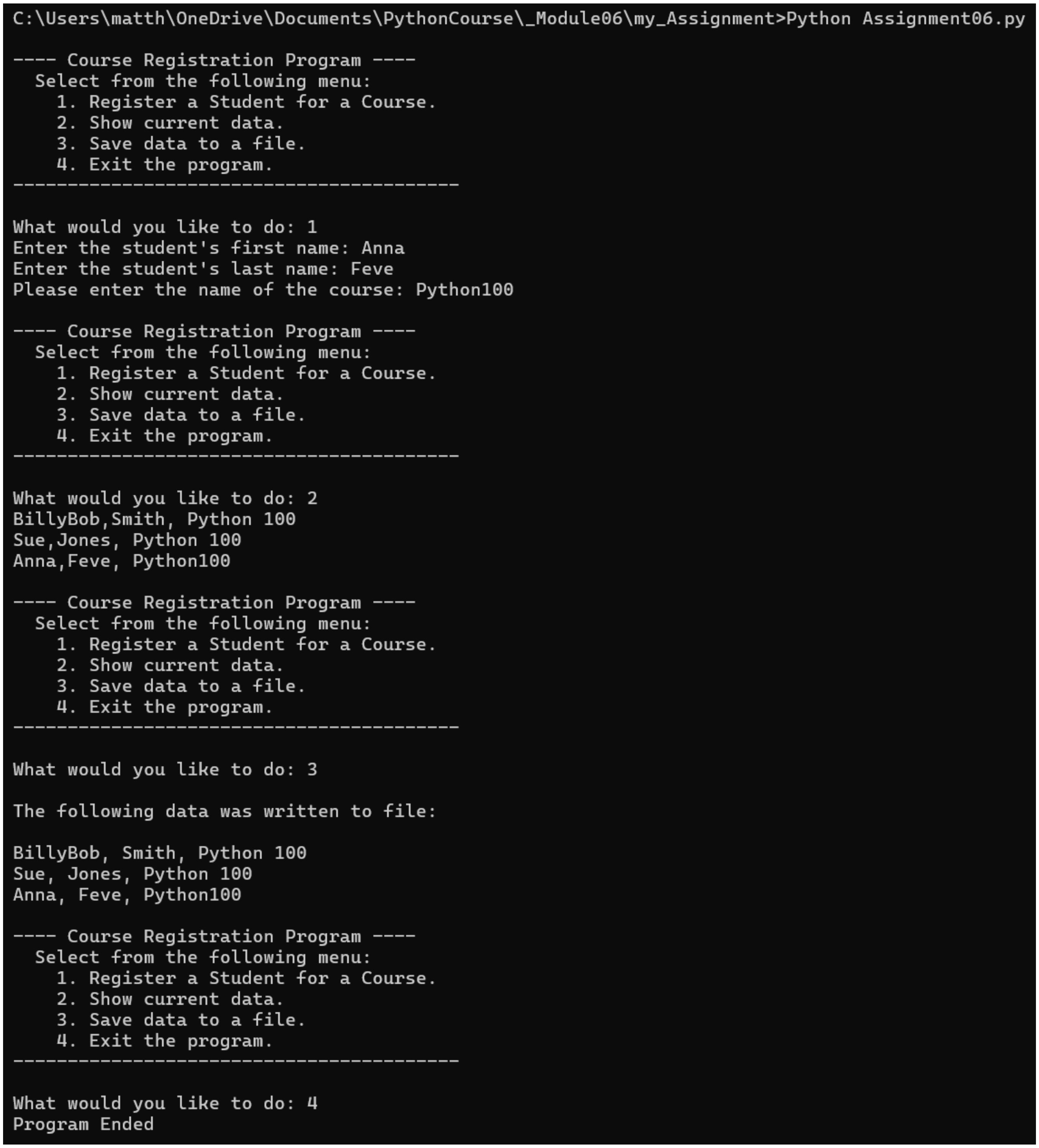


Assignment06.py   {} Enrollments.json ×

PythonCourse > _Module06 > my_Assignment > {} Enrollments.json > ...

```
[{"FirstName": "BillyBob", "LastName": "Smith", "CourseName": "Python 100"},
{"FirstName": "Sue", "LastName": "Jones", "CourseName": "Python 100"},
{"FirstName": "Matt", "LastName": "Feve", "CourseName": "Python 100"},
{"FirstName": "Milo", "LastName": "Feve", "CourseName": "Python 100"}]
```

Appendix C – Command Prompt Run Results



```
C:\Users\matth\OneDrive\Documents\PythonCourse\_Module06\my_Assignment>Python Assignment06.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: Anna
Enter the student's last name: Feve
Please enter the name of the course: Python100

---- 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: 2
BillyBob,Smith, Python 100
Sue,Jones, Python 100
Anna,Feve, Python100

---- 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

The following data was written to file:

BillyBob, Smith, Python 100
Sue, Jones, Python 100
Anna, Feve, Python100

---- 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
```