Seth Overbay

May 28th, 2025

IT FDN 110A

Assignment 06

GitHub URL:

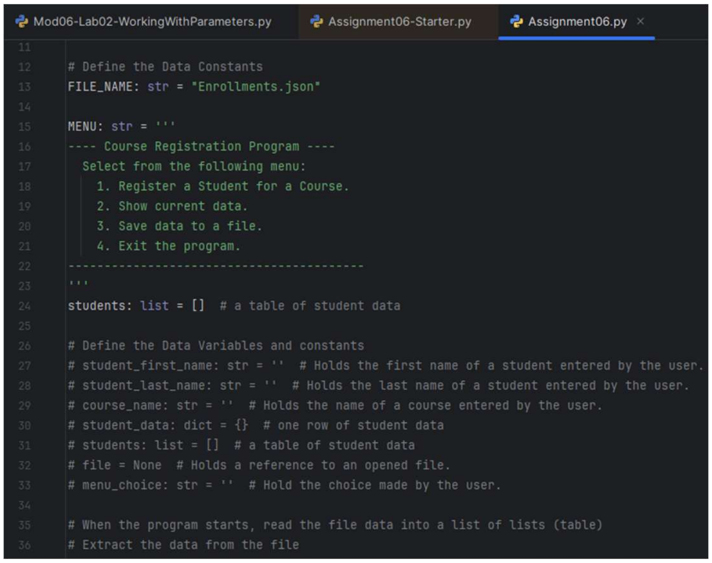
Introduction

In this week's module, I became more comfortable with structuring my scripts using classes and functions. Through the use of these fundamental concepts, I was able to gain a better understanding of how Separation of Concerns (SoC) is used to make lengthier scripts/programs more modular, scalable, reusable, and maintainable over time.

Assignment06

Step 1 *(Figure 6.1)*

- Defined my constants and variables, along with commenting out the global variables used in prior modules, as I will now use those variables locally within the new functions

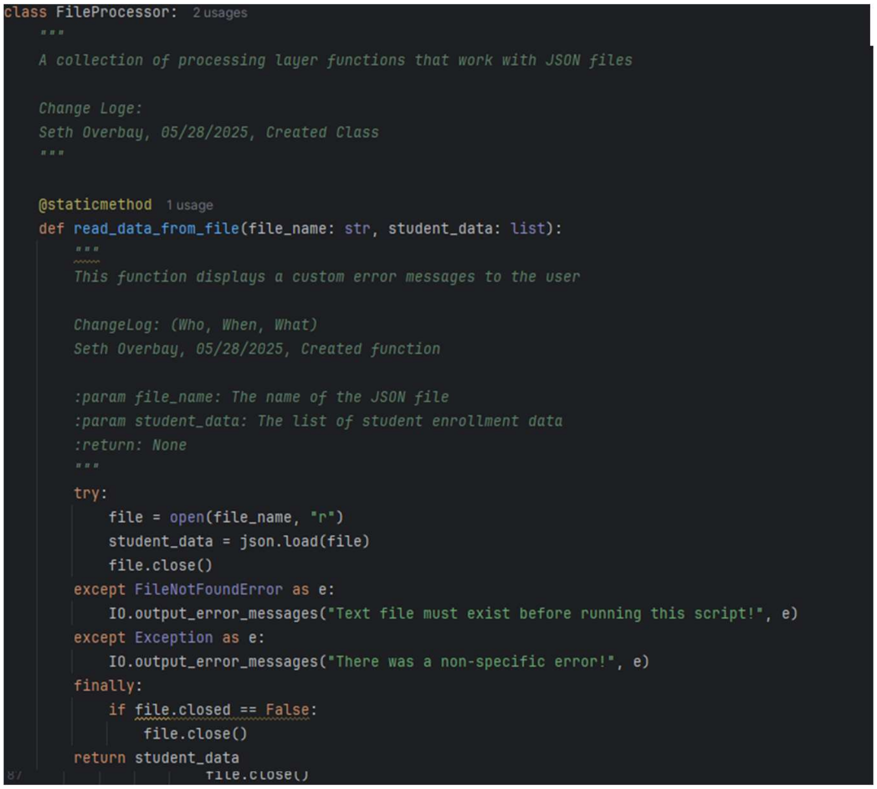


```

# Define the Data Constants
FILE_NAME: str = "Enrollments.json"

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.
-----------------------------------------
'''
students: list = []  # a table of student data

# 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.
# student_data: dict = {}  # one row of student data
# students: list = []  # a table of student data
# file = None  # Holds a reference to an opened file.
# menu_choice: str = ''  # Hold the choice made by the user.

# When the program starts, read the file data into a list of lists (table)
# Extract the data from the file
```

Figure 6.1

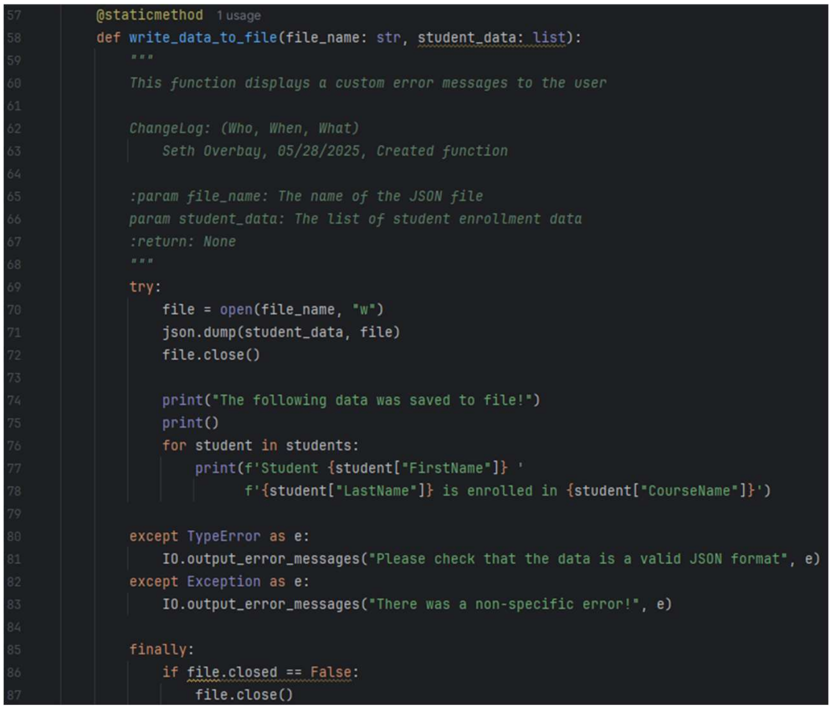
Step 2 *(Figure 6.2 & 6.3)*

- Created my first class (FileProcessor), which helps with separation of concerns (SoC) and separates code between a data processing layer and a presentation (UI) layer. Figure 6.2 shows the function created to read the current enrollment data from the “Enrollments” JSON file, while Figure 6.3 shows the function used to write existing and newly inputted user data back to the “Enrollments” JSON.



```
class FileProcessor:
    """
    A collection of processing layer functions that work with JSON files

    Change Loge:
    Seth Overbay, 05/28/2025, Created Class
    """

    @staticmethod
    def read_data_from_file(file_name: str, student_data: list):
        """
        This function displays a custom error messages to the user

        ChangeLog: (Who, When, What)
        Seth Overbay, 05/28/2025, Created function

        :param file_name: The name of the JSON file
        :param student_data: The list of student enrollment data
        :return: None
        """
        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
                file.close()
```

Figure 6.2

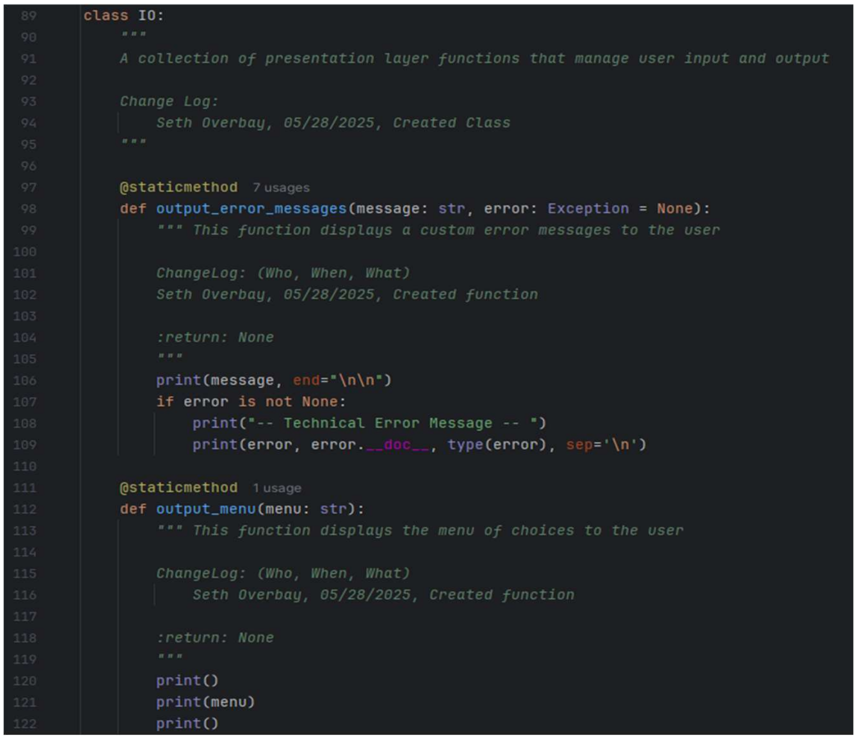


```
    @staticmethod
    def write_data_to_file(file_name: str, student_data: list):
        """
        This function displays a custom error messages to the user

        ChangeLog: (Who, When, What)
            Seth Overbay, 05/28/2025, Created function

        :param file_name: The name of the JSON file
        param student_data: The list of student enrollment data
        :return: None
        """
        try:
            file = open(file_name, "w")
            json.dump(student_data, file)
            file.close()

            print("The following data was saved to file!")
            print()
            for student in students:
                print(f'Student {student["FirstName"]} '
                      f'{student["LastName"]} is enrolled in {student["CourseName"]}')

        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 6.3

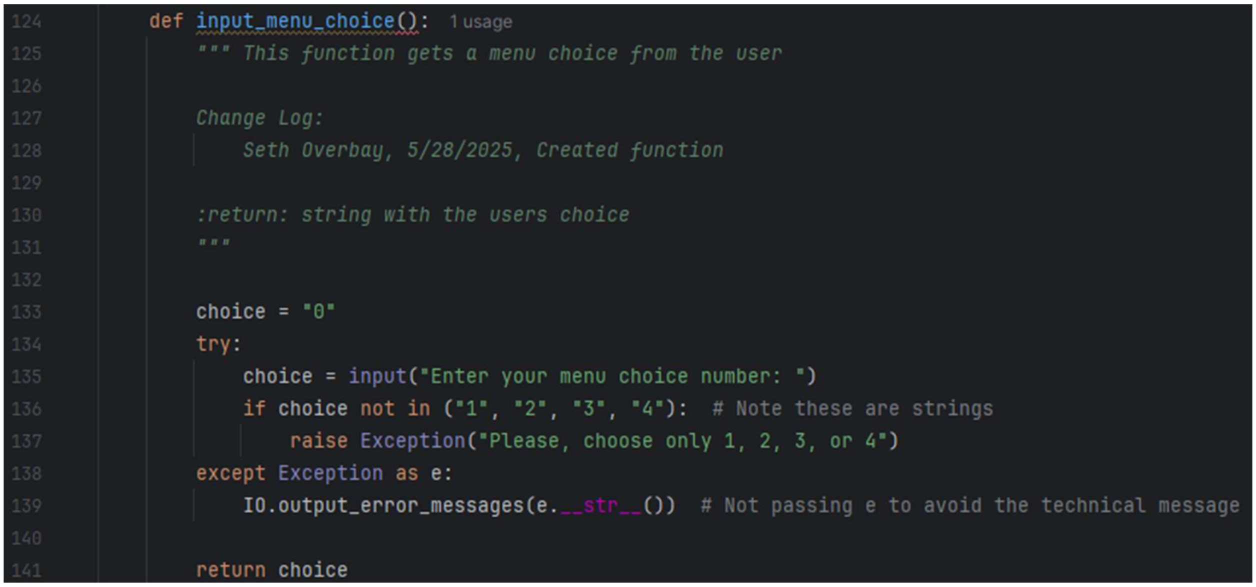
Step 3 *(Figure 6.4 – 6.7)*

- Created my second class (IO), which also satisfies the separation of concerns (SoC) principle and includes the functions used for presentation and user interfacing. Functions in this class include menu output *(Figure 6.4)*, user menu input *(Figure 6.5)*, student enrollment data input (*Figure 6.6)*, & student enrollment data output (Figure 6.7).



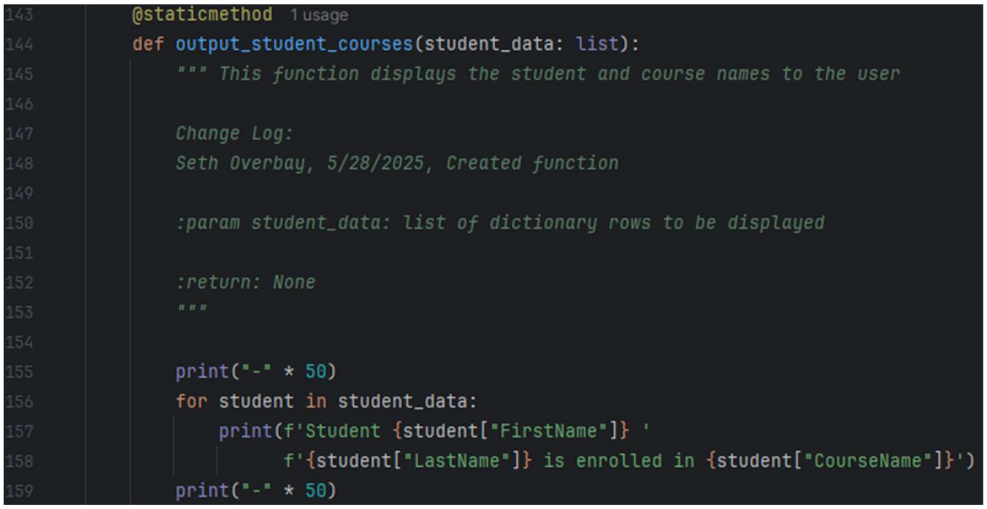
```
class IO:
    """
    A collection of presentation layer functions that manage user input and output

    Change Log:
        Seth Overbay, 05/28/2025, Created Class
    """

    @staticmethod  7 usages
    def output_error_messages(message: str, error: Exception = None):
        """ This function displays a custom error messages to the user

        ChangeLog: (Who, When, What)
        Seth Overbay, 05/28/2025, 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')

    @staticmethod  1 usage
    def output_menu(menu: str):
        """ This function displays the menu of choices to the user

        ChangeLog: (Who, When, What)
            Seth Overbay, 05/28/2025, Created function

        :return: None
        """
        print()
        print(menu)
        print()
```

Figure 6.4



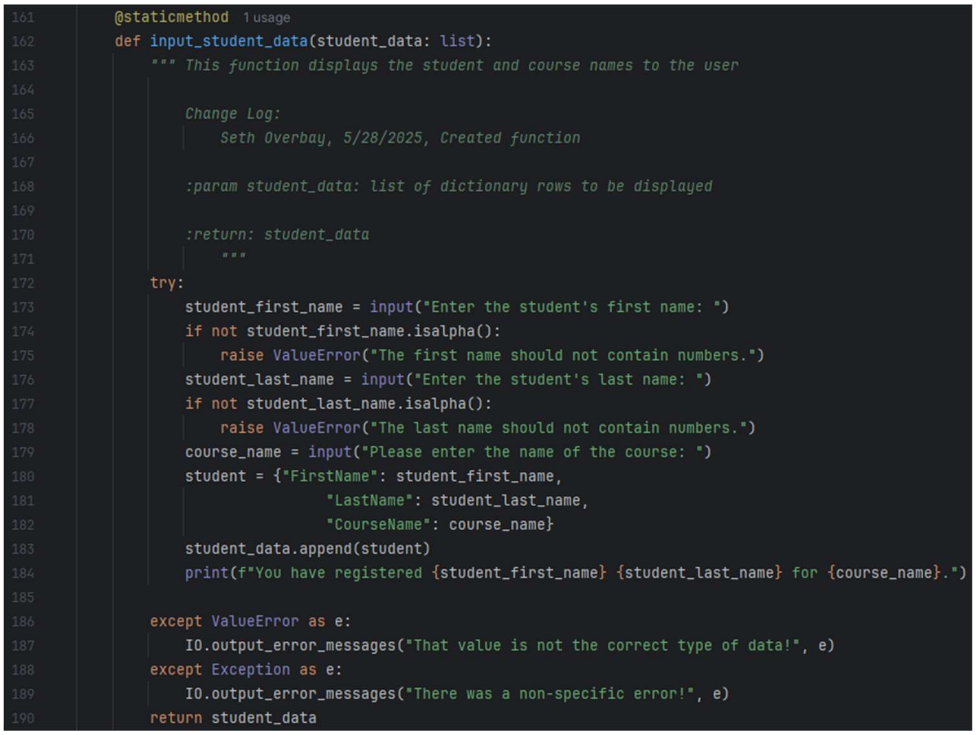
```
    def input_menu_choice():  1 usage
        """ This function gets a menu choice from the user

        Change Log:
            Seth Overbay, 5/28/2025, Created function

        :return: string with the users choice
        """

        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 e to avoid the technical message

        return choice
```

Figure 6.5



```python
    @staticmethod
    def output_student_courses(student_data: list):
        """ This function displays the student and course names to the user

        Change Log:
        Seth Overbay, 5/28/2025, Created function

        :param student_data: list of dictionary rows to be displayed

        :return: None
        """

        print("-" * 50)
        for student in student_data:
            print(f'Student {student["FirstName"]} '
                  f'{student["LastName"]} is enrolled in {student["CourseName"]}')
        print("-" * 50)
```

Figure 6.6

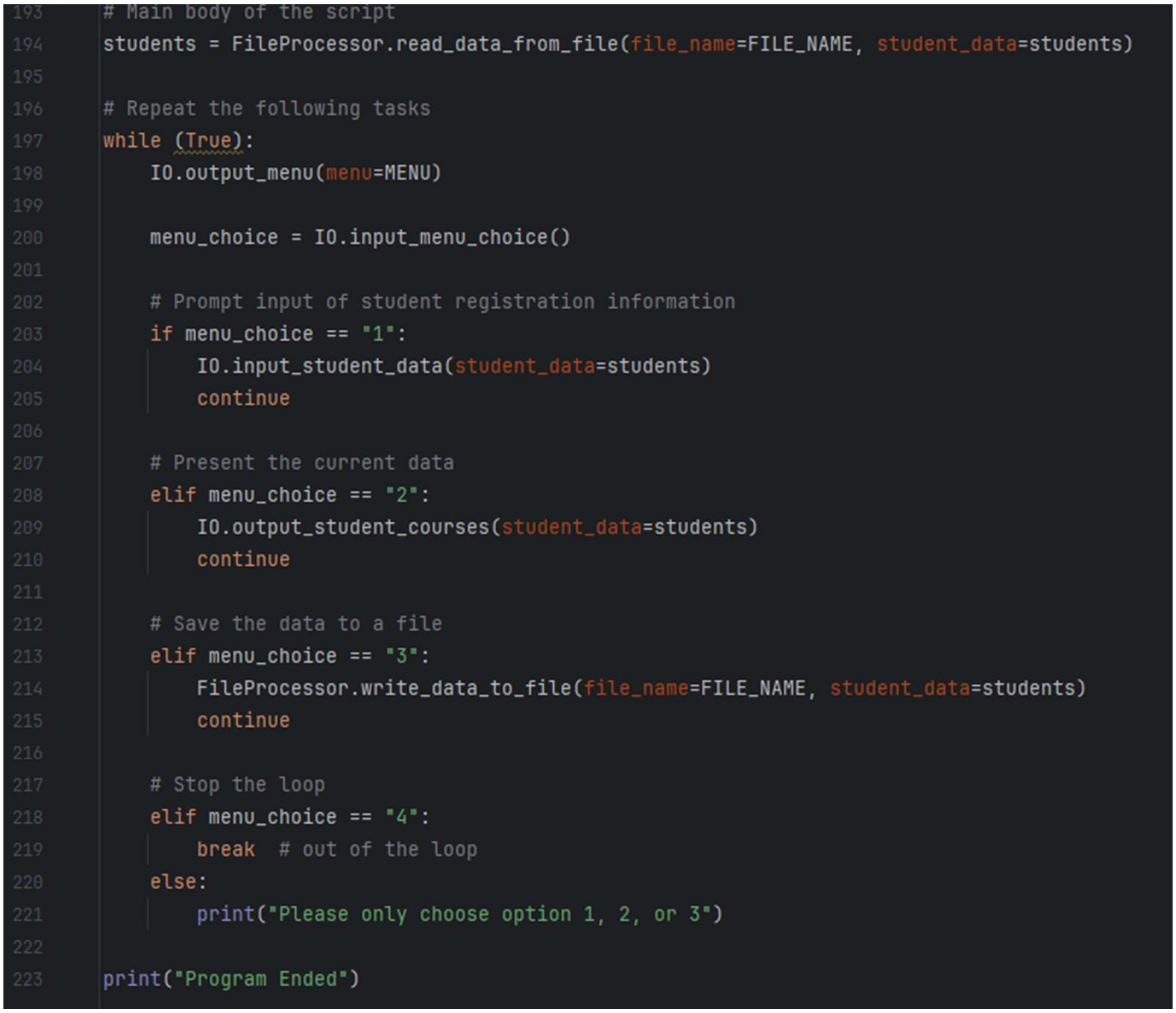


```python
    @staticmethod
    def input_student_data(student_data: list):
        """ This function displays the student and course names to the user

            Change Log:
                Seth Overbay, 5/28/2025, Created function

            :param student_data: list of dictionary rows to be displayed

            :return: student_data
                """
        try:
            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.")
            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: ")
            student = {"FirstName": student_first_name,
                            "LastName": student_last_name,
                            "CourseName": course_name}
            student_data.append(student)
            print(f"You have registered {student_first_name} {student_last_name} for {course_name}.")

        except ValueError as e:
            IO.output_error_messages("That value is not the correct type of data!", e)
        except Exception as e:
            IO.output_error_messages("There was a non-specific error!", e)
        return student_data
```

Figure 6.7

<u>Step 4 (*Figure 6.8*)</u>

- Used a “while” loop to print the menu, seek menu option inputs from the user and then repeat tasks based on user menu selections. Each if/elif statement calls on a different function from the two classes described above (FileProcessor & IO). Calling on these functions creates a much cleaner body of code and helps with code modularity & reusability. These functions are showing in *Figure 6.8 (next page)*.



```
# Main body of the script
students = FileProcessor.read_data_from_file(file_name=FILE_NAME, student_data=students)

# Repeat the following tasks
while (True):
    IO.output_menu(menu=MENU)

    menu_choice = IO.input_menu_choice()

    # Prompt input of student registration information
    if menu_choice == "1":
        IO.input_student_data(student_data=students)
        continue

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

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

    # 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 6.8.

Summary

In this week's module, I became more comfortable with structuring my scripts using classes and functions. Through the use of these fundamental concepts, I was able to gain a better understanding of how Separation of Concerns (SoC) is used to make lengthier scripts/programs more modular, scalable, reusable, and maintainable over time.