Anna-Sofiia Mykhalevych

September 4, 2024

IT FDN 110 B

Assignment 06

https://github.com/AnnaSofiia/IntroToProg-Python

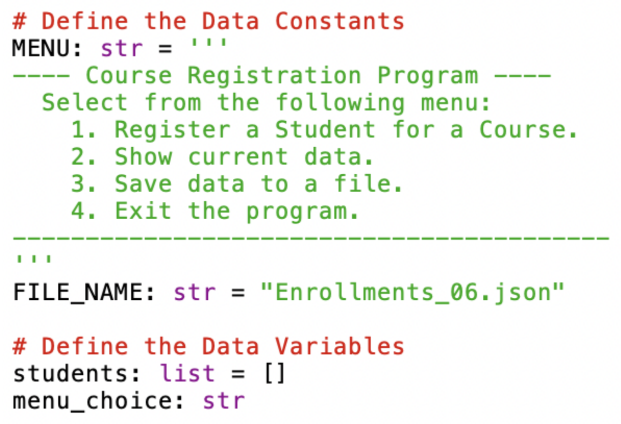
FUNCTIONS

Introduction

In this course registration assignment, I focused on learning three key programming techniques to enhance my Python script: utilizing functions, implementing classes, and adhering to the separation of concerns programming pattern. These methods were instrumental in improving the structure, maintainability, and efficiency of my course registration system.

1. Defining data constants and variables

The MENU constant outlines the interactive options available for the user, which include registering a student, showing current data, saving data to a file, and exiting the program. The students variable is initialized as an empty list to hold the registered student details, and menu_choiceis prepared to capture user input. This setup enhances the organization and flexibility of the registration system, allowing for more sophisticated data handling and retrieval.



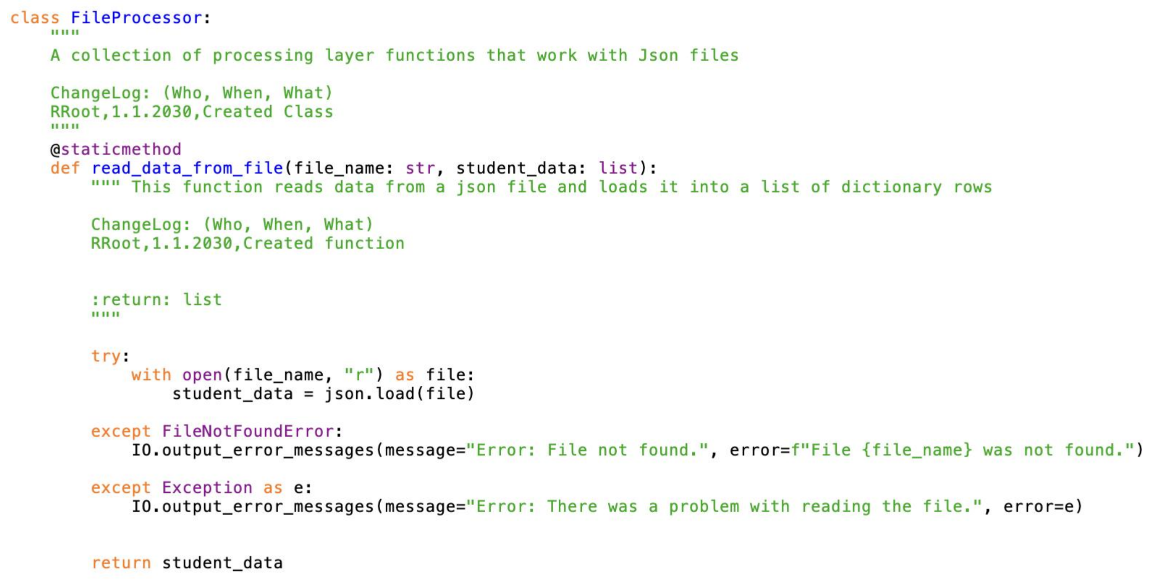
```
# Define the Data Constants
MENU: str = '''
---- Course Registration Program ----
  Select from the following menu:
    1. Register a Student for a Course.
    2. Show current data.
    3. Save data to a file.
    4. Exit the program.
-----------------------------------------
'''
FILE_NAME: str = "Enrollments_06.json"

# Define the Data Variables
students: list = []
menu_choice: str
```

Figure 1. *Defining data constants and variables*

2. Creating FileProcessor class

In this section of my Python project, I created a FileProcessor class designed to work with JSON files, enhancing the functionality of my course registration system. This class includes a read_data_from_file static method that opens a JSON file, reads the data, and loads it into a list of dictionaries representing student data. This method improves the script's ability to handle structured data efficiently.

I also implemented error handling within this method to manage common issues like file not being found or other reading errors. Using custom error messages helps in debugging and provides clear feedback to users when an error occurs.



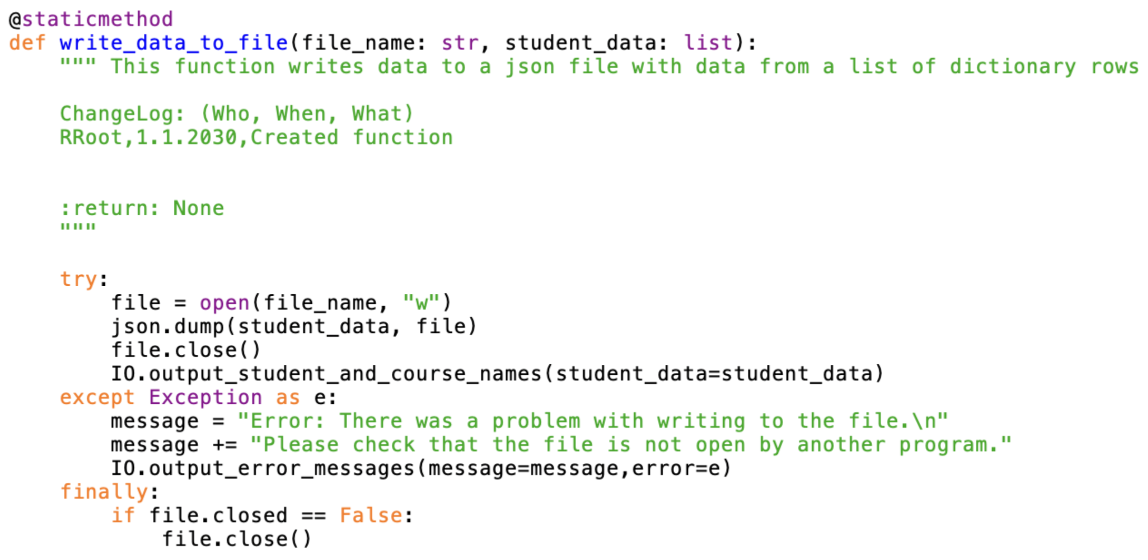
```
class FileProcessor:
    """
    A collection of processing layer functions that work with Json files

    ChangeLog: (Who, When, What)
    RRoot,1.1.2030,Created Class
    """
    @staticmethod
    def read_data_from_file(file_name: str, student_data: list):
        """ This function reads data from a json file and loads it into a list of dictionary rows

        ChangeLog: (Who, When, What)
        RRoot,1.1.2030,Created function


        :return: list
        """

        try:
            with open(file_name, "r") as file:
                student_data = json.load(file)

        except FileNotFoundError:
            IO.output_error_messages(message="Error: File not found.", error=f"File {file_name} was not found.")

        except Exception as e:
            IO.output_error_messages(message="Error: There was a problem with reading the file.", error=e)


        return student_data
```

Figure 2. *Creating FileProcessor class*

3. Static method

In this part of my Python project, I extended the FileProcessor class with a new static method, write_data_to_file, designed to handle writing data to a JSON file. This method opens the specified file in write mode, uses the json.dumpfunction to serialize a list of dictionaries (student data) into JSON format, and then closes the file. The method is structured to handle exceptions that might occur during the file operation process, providing detailed error messages to help diagnose issues.

Moreover, I included a finally block to ensure the file is properly closed even if an error occurs during the writing process.

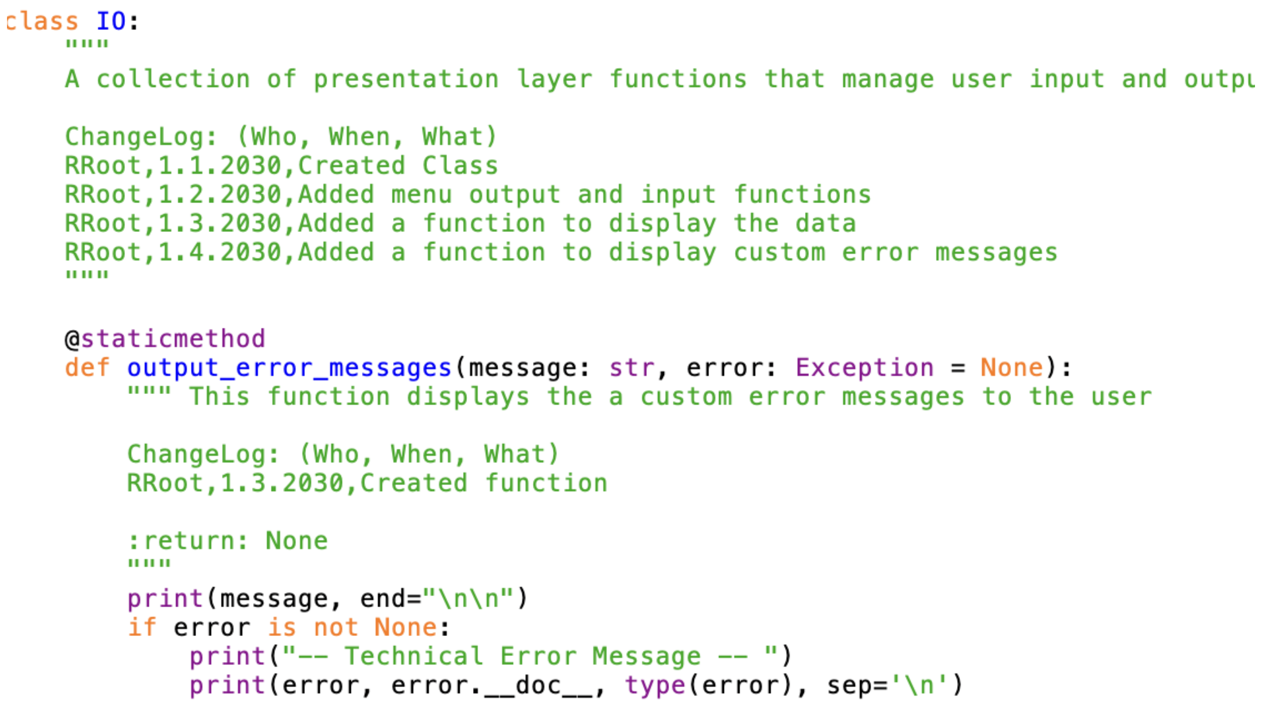


```
@staticmethod
def write_data_to_file(file_name: str, student_data: list):
    """ This function writes data to a json file with data from a list of dictionary rows

    ChangeLog: (Who, When, What)
    RRoot,1.1.2030,Created function


    :return: None
    """

    try:
        file = open(file_name, "w")
        json.dump(student_data, file)
        file.close()
        IO.output_student_and_course_names(student_data=student_data)
    except Exception as e:
        message = "Error: There was a problem with writing to the file.\n"
        message += "Please check that the file is not open by another program."
        IO.output_error_messages(message=message,error=e)
    finally:
        if file.closed == False:
            file.close()
```

Figure 3. *Static method*

4. IO class

In this segment of my Python project, I expanded the functionality of the IO class by adding a method output_error_messages. This method is crucial for providing clear and informative feedback to users when errors occur. It displays a custom message, and if an exception is provided, it further elaborates on the technical details of the error, including the exception's documentation and type.



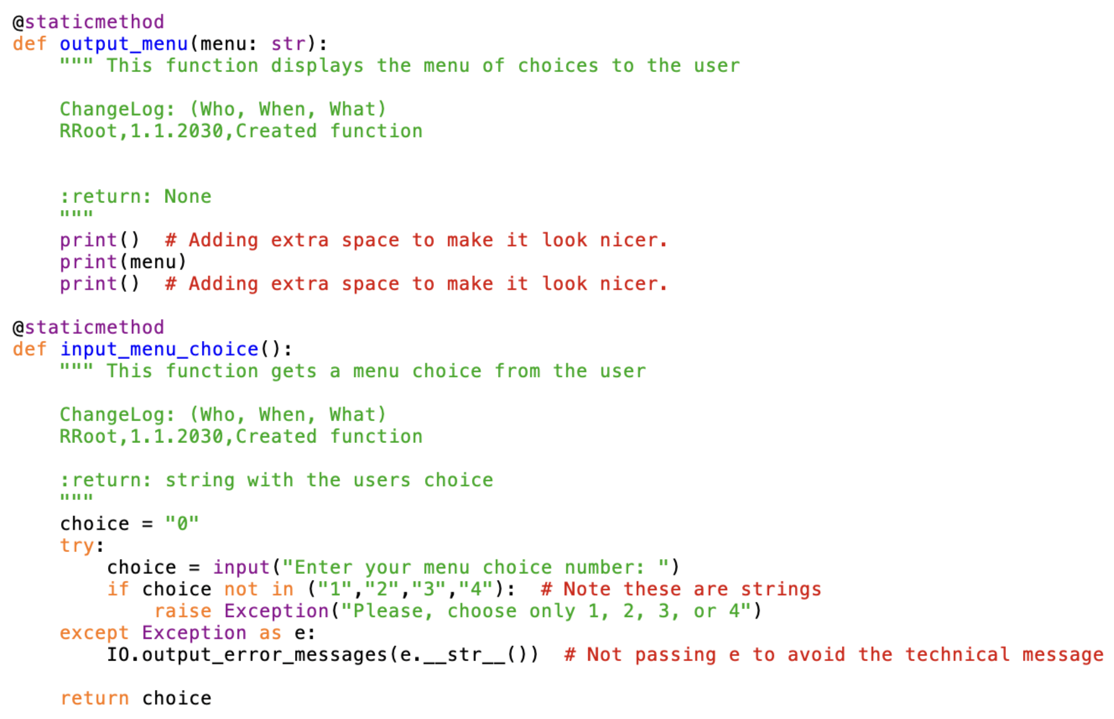
```
class IO:
    """
    A collection of presentation layer functions that manage user input and outpu

    ChangeLog: (Who, When, What)
    RRoot,1.1.2030,Created Class
    RRoot,1.2.2030,Added menu output and input functions
    RRoot,1.3.2030,Added a function to display the data
    RRoot,1.4.2030,Added a function to display custom error messages
    """

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

        ChangeLog: (Who, When, What)
        RRoot,1.3.2030,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 4. *IO class*

5. Enhancing IO class

In this part of my project, I enhanced the IO class by adding methods for displaying the menu and handling menu choice input from users, emphasizing user interaction within the application.

1. **Outputting the Menu:** The output_menu static method neatly presents the menu options to the user. I added extra print statements to insert blank lines before and after the menu, improving the visual layout and making it easier for users to read and make a selection.
2. **Handling User Input:** The input_menu_choice method prompts the user to enter their menu choice, ensuring that it is one of the valid options ("1", "2", "3", "4"). If an invalid option is entered, an exception is raised. I used error handling to catch this exception and employed the previously created output_error_messages method to provide a clear error message without exposing unnecessary technical details to the user.



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

    ChangeLog: (Who, When, What)
    RRoot,1.1.2030,Created function


    :return: None
    """
    print()  # Adding extra space to make it look nicer.
    print(menu)
    print()  # Adding extra space to make it look nicer.

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

    ChangeLog: (Who, When, What)
    RRoot,1.1.2030,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
```

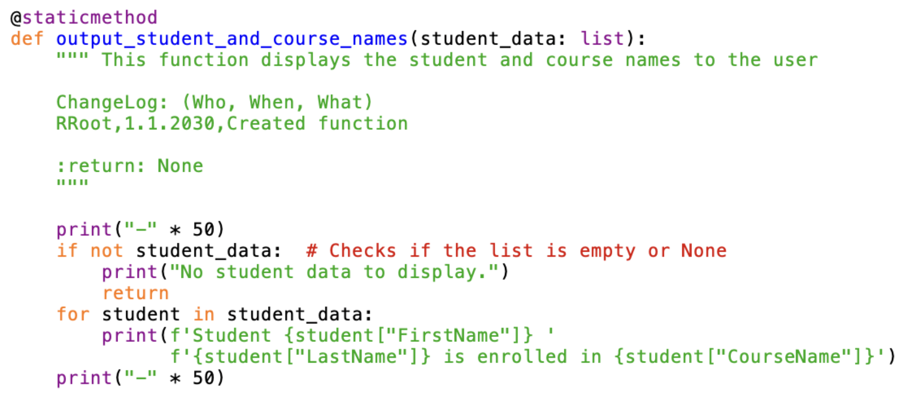
Figure 5. *Enhancing IO class*

6. Output_student_and_course_namesstatic method

In this part of the project, I further developed the IO class by implementing the output_student_and_course_namesstatic method. This method is tailored to display student and course information clearly to the user.

Here's how it works:

- **Formatting and Presentation:** The method starts and ends with a line of dashes to visually separate the student data output from other text in the console. This helps in keeping the user interface clean and easy to read.
- **Handling Empty Data:** Before attempting to display any student information, the method checks if the student_data list is empty. If it is, a message "No student data to display." is printed, informing the user that there are no records to show.
- **Data Output:** For each student in the student_data list, the method prints the student's first and last names along with the course name they are enrolled in, formatted in a clear and readable manner.



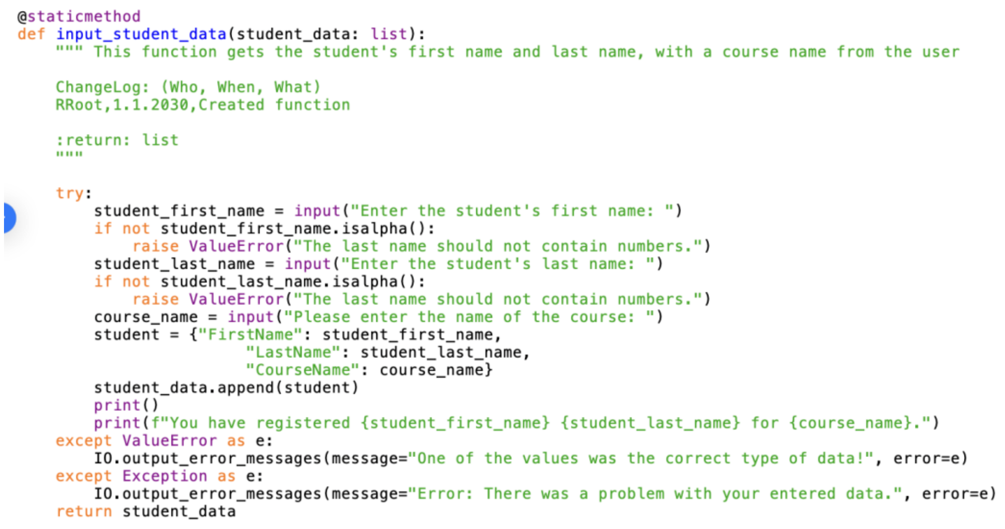
```
@staticmethod
def output_student_and_course_names(student_data: list):
    """ This function displays the student and course names to the user

    ChangeLog: (Who, When, What)
    RRoot,1.1.2030,Created function

    :return: None
    """

    print("-" * 50)
    if not student_data:  # Checks if the list is empty or None
        print("No student data to display.")
        return
    for student in student_data:
        print(f'Student {student["FirstName"]} '
              f'{student["LastName"]} is enrolled in {student["CourseName"]}')
    print("-" * 50)
```

Figure 6. *Output_student_and_course_namesstatic method*

7. Collect student details from the user

In this section of the project, I designed the input_student_data static method within the IO class to interactively collect student details from the user. The method gathers the student's first name, last name, and the course they wish to enroll in, incorporating several checks to ensure the data integrity:

- **Data Validation:** The method validates that the first and last names contain only alphabetical characters by using the isalpha() method. If the validation fails, it raises a ValueError with a message indicating that the names should not contain numbers or special characters.
- **Error Handling:** The method is wrapped in a try-except block to handle any exceptions that might arise during data input. This includes handling specific ValueErrors raised by validation failures and more general exceptions that could occur during the input process.
- **Feedback to User:** Upon successfully registering a student, it confirms the registration by printing a message. In case of an error, it utilizes the output_error_messages function from the same class to display appropriate error messages to the user.

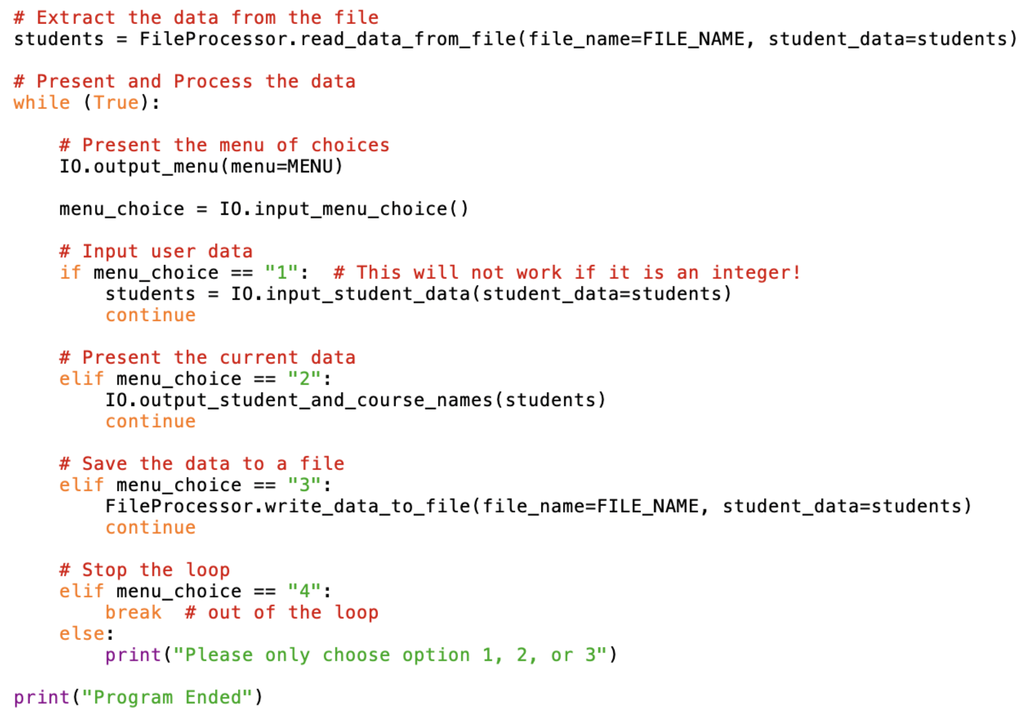


```
@staticmethod
def input_student_data(student_data: list):
    """ This function gets the student's first name and last name, with a course name from the user

    ChangeLog: (Who, When, What)
    RRoot,1.1.2030,Created function

    :return: list
    """

    try:
        student_first_name = input("Enter the student's first name: ")
        if not student_first_name.isalpha():
            raise ValueError("The last 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()
        print(f"You have registered {student_first_name} {student_last_name} for {course_name}.")
    except ValueError as e:
        IO.output_error_messages(message="One of the values was the correct type of data!", error=e)
    except Exception as e:
        IO.output_error_messages(message="Error: There was a problem with your entered data.", error=e)
    return student_data
```

Figure 7. *Collect student details from the user*

8. Main execution loop

In this segment of my Python script, I implemented the main execution loop for the course registration system, integrating various functionalities into a cohesive workflow. The script begins by loading existing student data from a JSON file. It then presents a menu to the user, captures their choice, and performs actions based on their selection, such as registering a new student, displaying current data, saving updates to a file, or exiting the program. Each menu option triggers a specific function that handles data collection, display, or storage, ensuring the application is interactive and user-friendly.



```python
# Extract the data from the file
students = FileProcessor.read_data_from_file(file_name=FILE_NAME, student_data=students)

# Present and Process the data
while (True):

    # Present the menu of choices
    IO.output_menu(menu=MENU)

    menu_choice = IO.input_menu_choice()

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

    # Present the current data
    elif menu_choice == "2":
        IO.output_student_and_course_names(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 8. *Main execution loop*

Summary

In my course registration system project, I designed several core functions to enhance the system's functionality and user interaction. These included functions for reading and writing JSON data to efficiently manage student information, ensuring data integrity and persistence.

I also implemented input validation to check the correctness of user-entered data and structured menu navigation to guide users through various application features. Robust error handling mechanisms were integrated to address potential issues like file access or invalid inputs, maintaining system stability.

Additionally, I developed functions to display registered students and their courses, providing clear and accessible data presentation. This suite of functions was crafted to ensure seamless interaction and a stable user experience, further honing my Python development skills and demonstrating my capability in creating practical, user-oriented applications.