Crisandra Wilkie

3/5/25

Foundations of Programming: Python

Assignment 6

https://github.com/cris-wilkie/Python110Wi25.git

# Introduction

The focus of Module 6 was functions, classes and the separation of concerns. The objective was to implement a simple course registration program. The user can interact with it through a menu system, allowing them to register students, view current student registrations, save data to a file, or exit the program. Through the use of functions, classes and the separation of concerns I was able to create a modular and workable script.

# Key Components

Each part of the program was organized into the following categories data, processing, and presentation. This is what separation of concerns, mentioned above, is referring to. First, I started by inserting and updating the script header and I made sure to import the JSON module. Then I created the constants and variables.

**Data Constants:**

- MENU: A string containing the main menu options for the user.
- FILE_NAME: The name of the JSON file where student data is stored.

**Data Variables:**

- students: A list to hold student registration data.
- menu_choice: A variable to store the user's menu choice.

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

# Define the program's data
students: list = []  # a table of student data
menu_choice: str ='' # Hold the choice made by the user.
```

*Figure 1. Variables and Constants*

## Classes and functions

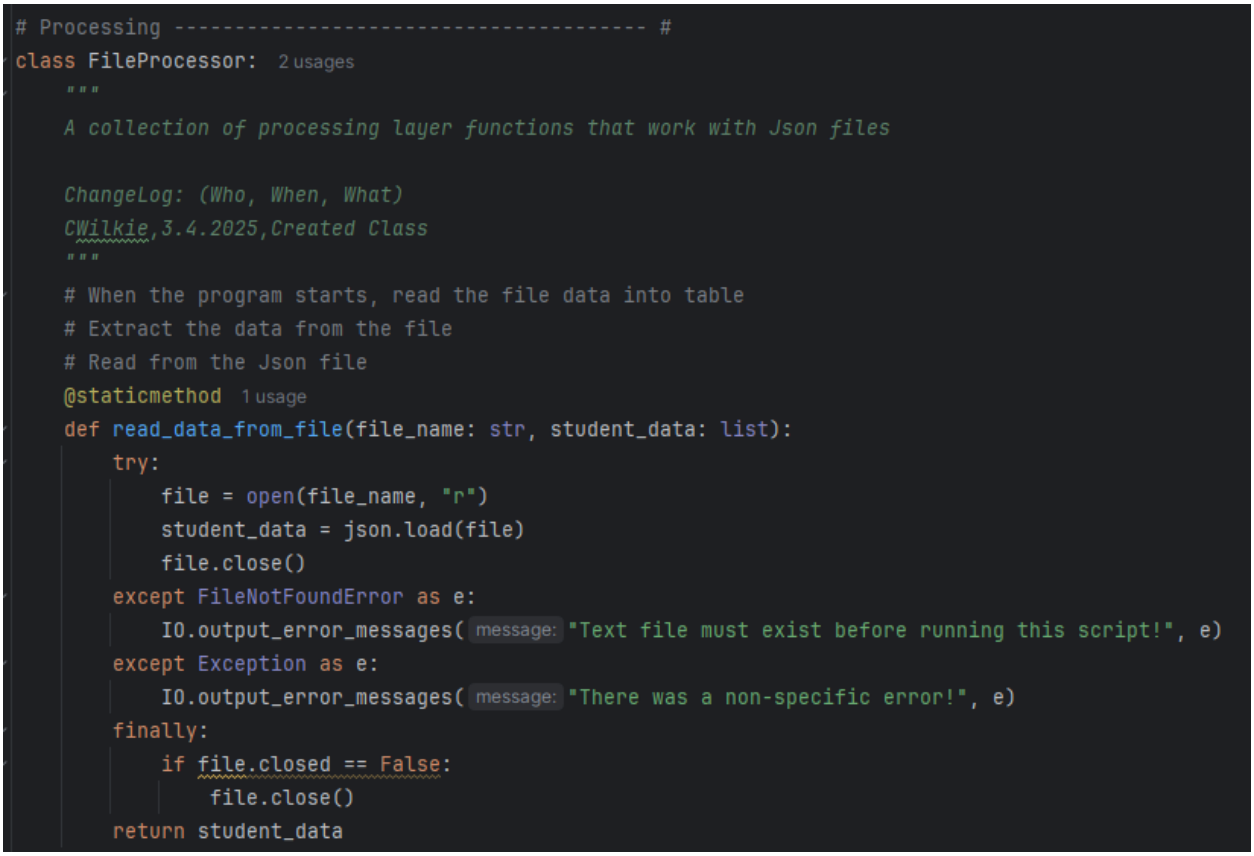
The next step was to create the classes. Classes help to organize the script by way of grouping functions, variables, and constants by the name of the class. A function is code that performs a specific task or set of tasks and is reusable. Each class and function included document strings, these simply allow space to offer notation on the class and function. The functions also included the @staticmethod decorator, this is indicative that the function code is static and does not change.

**FileProcessor:** Contains methods for handling file operations.

- read_data_from_file: Reads student data from a JSON file and loads it into the students list.
- write_data_to_file: Saves the current student data to a JSON file.

**IO:** Handles user input and output.

- output_error_messages: Displays custom error messages.
- output_menu: Displays the program's menu to the user.
- input_menu_choice: Prompts the user to input their menu choice.
- output_student_courses: Displays the current student registrations.
- input_student_data: Prompts the user to input a new student's information (first name, last name, and course name), and adds it to the students list.



```python
# Processing ------------------------------------- #
class FileProcessor:  2 usages
    """
    A collection of processing layer functions that work with Json files

    ChangeLog: (Who, When, What)
    CWilkie,3.4.2025,Created Class
    """
    # When the program starts, read the file data into table
    # Extract the data from the file
    # Read from the Json file
    @staticmethod  1 usage
    def read_data_from_file(file_name: str, student_data: list):
        try:
            file = open(file_name, "r")
            student_data = json.load(file)
            file.close()
        except FileNotFoundError as e:
            IO.output_error_messages( message: "Text file must exist before running this script!", e)
        except Exception as e:
            IO.output_error_messages( message: "There was a non-specific error!", e)
        finally:
            if file.closed == False:
                file.close()
        return student_data
```

*Figure 2. Class and Function*

# Main Loop

The program begins by reading any existing student data from the file (students).

A while loop presents the menu to the user repeatedly until they choose to exit.

- Option 1: Prompts the user to enter a new student's data.
- Option 2: Displays all current student registrations.
- Option 3: Saves the current data to the JSON file.
- Option 4: Exits the program.

```
# Beginning of the main body of this script
students = FileProcessor.read_data_from_file(file_name=FILE_NAME, student_data=students)
while True:
    IO.output_menu(menu=MENU)

    # Present the menu of choices
    menu_choice = IO.input_menu_choice()

    # Input user data
    if  menu_choice == "1":  # Get new data (and display the change)
        students = IO.input_student_data(student_data=students)
        IO.output_student_courses(student_data=students)  # Added this to improve user experience
        continue
```

*Figure 3. Start of the main body*

# Error Handling

The script uses try-except blocks to handle potential errors such as invalid input or file-related issues, and it displays relevant error messages.

**Workflow:**

1. The user is presented with a menu of options.
2. If they choose to register a student, the program prompts them for the student's first name, last name, and course.
3. If they choose to view current registrations, the program displays a list of all students enrolled in courses.
4. If they choose to save data, the program saves the student data to a JSON file.
5. The loop continues until the user decides to exit.

**Example of Program Flow:**

1. The user selects option 1 (register a student).
2. They input the student's details (first name, last name, and course).
3. The program displays the updated list of students.
4. The user can continue, view current data, save, or exit the program.

## Conclusion

As in previous weeks, this program focuses on managing student course registrations in a simple and user-friendly manner. Through the use of classes, functions and the separation of concerns, I was able to create a more organized and modular script. The use of document strings help me to keep the script annotated, so if I were to revisit or if someone else was to use my script, it should be easy to understand the purpose of each code block or statement.