Sam Hollister

5/28/25

IT FDN 110 A

Assignment 06

https://github.com/samhollister/IntroToProg-Python-Mod06

Functions

Introduction

In this module, we learned new techniques to improve the reusability and organization of scripts: functions, classes, and the separation of concerns pattern. Functions and classes are ways of storing and organizing actions we want the program to be able to perform repeatedly. The separation of concerns is a framework for organizing the script, which improves readability and makes troubleshooting easier.

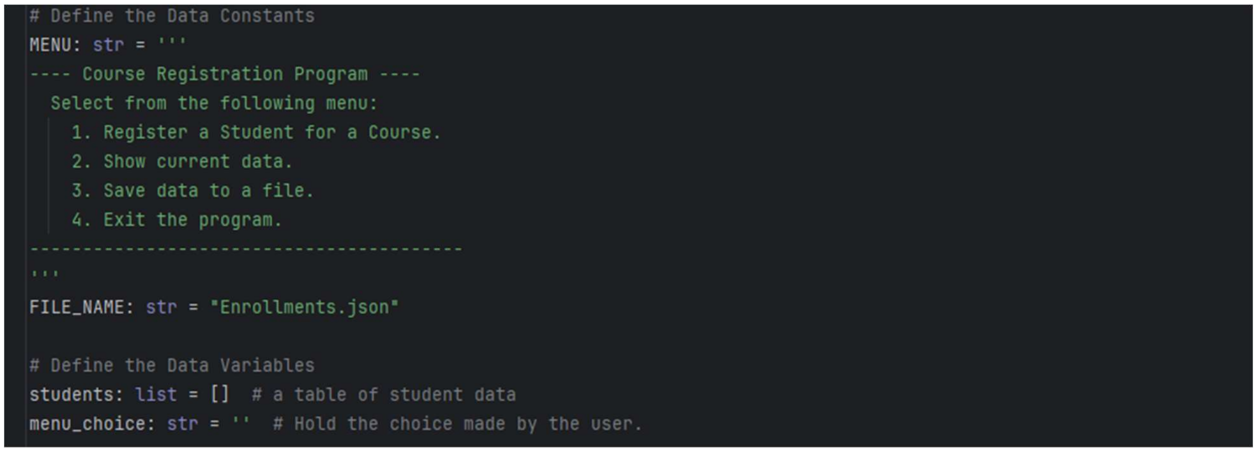
Objective

In this week's assignment, the objective is to use the techniques we learned to build upon a script that is like those used in previous assignments. Instead of handling the file processing and input/output in the body of the script, in this assignment we are meant to use functions and classes to pre-define the actions of the program and use the separation of concerns pattern to organize the script.

The script should be organized in the following sections:

- Script header
- Set up (import libraries & define variables and constants)
- Define classes and functions according to the separation of concerns pattern
- Main script body

The script header and set up are standard and have been covered in previous assignments. The most notable change is the use of functions allows us to define fewer variables outside of functions. In this case, the only variables and constants defined outside of functions are the MENU string, the FILE_NAME, the students list, and the menu_choice string.



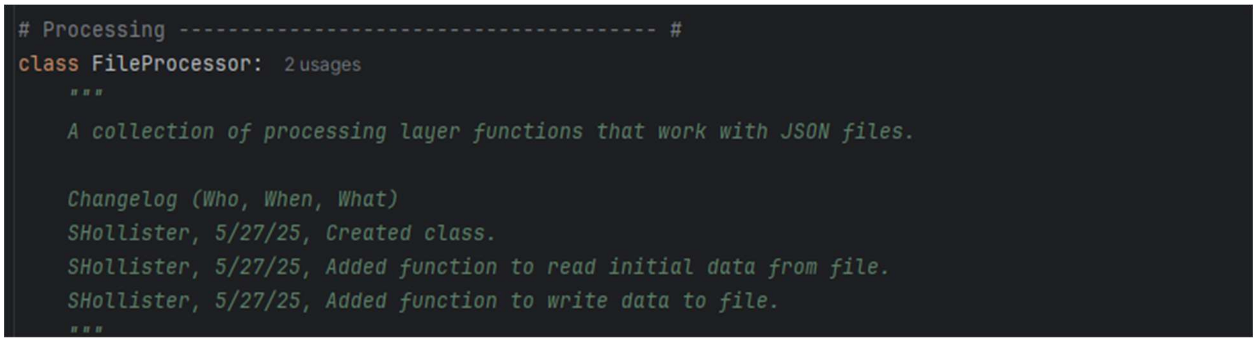
```
# Define the Data Constants
MENU: str = '''
---- Course Registration Program ----
  Select from the following menu:
    1. Register a Student for a Course.
    2. Show current data.
    3. Save data to a file.
    4. Exit the program.
-----------------------------------------
'''
FILE_NAME: str =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

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

Figure 1: The reduced list of constants and variables

Defining Functions and Classes

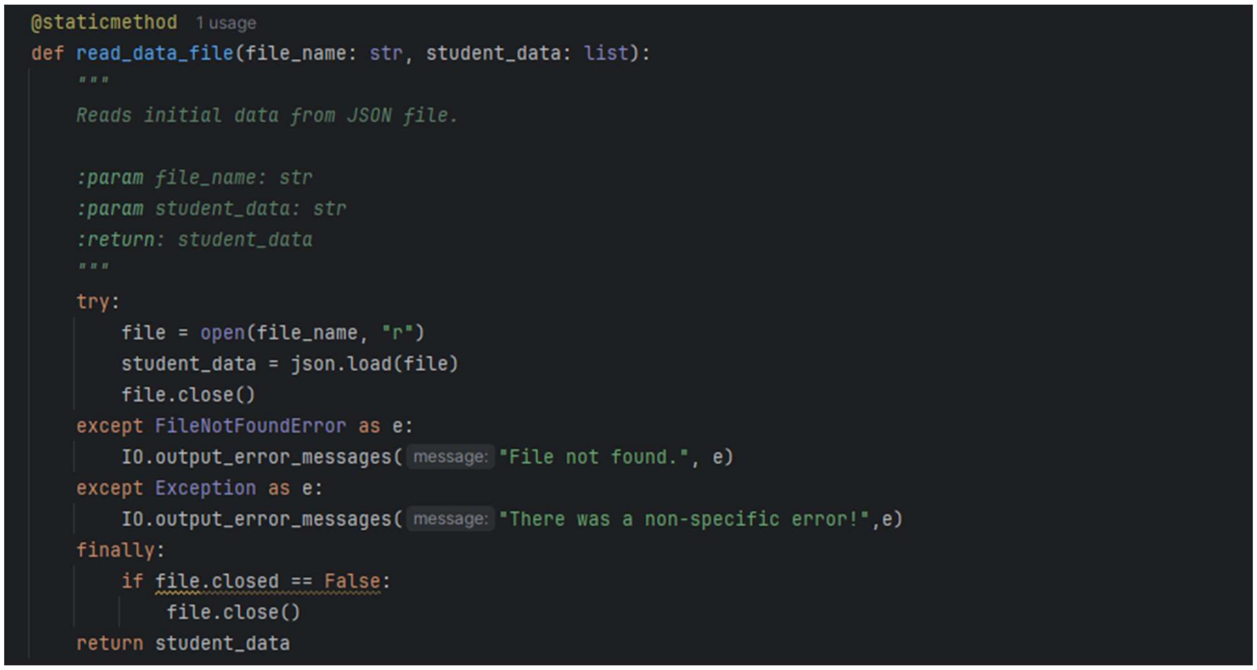
While defining the functions and classes, we follow the separation of concerns pattern and split them into two subsections: Processing and Input/Ouput (I/O).

In the Processing section, we define functions to read the data file and write data to the file. We assign them to the class FileProcessor, and add docstrings to inform the user what each class and function does. One interesting thing in these functions is that they call a function we will define later on in the script, which handles error messages. This is a good example of when order does not matter in a Python script.



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

    Changelog (Who, When, What)
    SHollister, 5/27/25, Created class.
    SHollister, 5/27/25, Added function to read initial data from file.
    SHollister, 5/27/25, Added function to write data to file.
    """
```

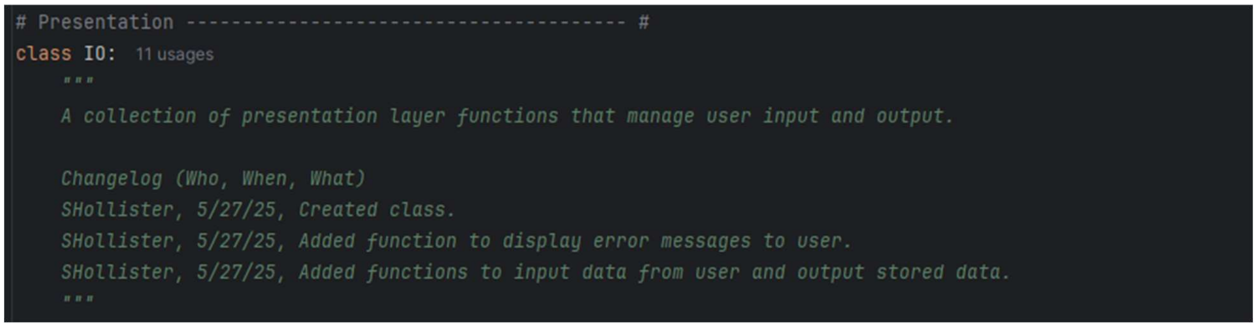
Figure 2.1: Defining the FileProcessor class



```
@staticmethod  1 usage
def read_data_file(file_name: str, student_data: list):
    """
    Reads initial data from JSON file.

    :param file_name: str
    :param student_data: str
    :return: student_data
    """
    try:
        file = open(file_name, "r")
        student_data = json.load(file)
        file.close()
    except FileNotFoundError as e:
        IO.output_error_messages( message: "File not found.", 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
```

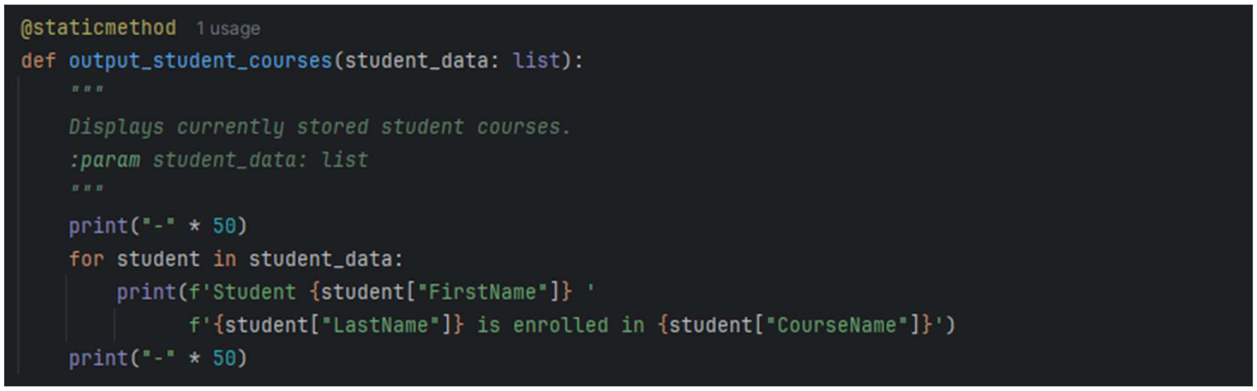
Figure 2.2: Defining the read_data_file function

Next, we handle the I/O functions. We define the class IO, and within it define functions to: output error messages, output menu, input menu choice, output student courses, and input student data.



```
# Presentation -------------------------------------- #
class IO:  11 usages
    """
    A collection of presentation layer functions that manage user input and output.

    Changelog (Who, When, What)
    SHollister, 5/27/25, Created class.
    SHollister, 5/27/25, Added function to display error messages to user.
    SHollister, 5/27/25, Added functions to input data from user and output stored data.
    """
```

Figure 2.3: Defining the IO class



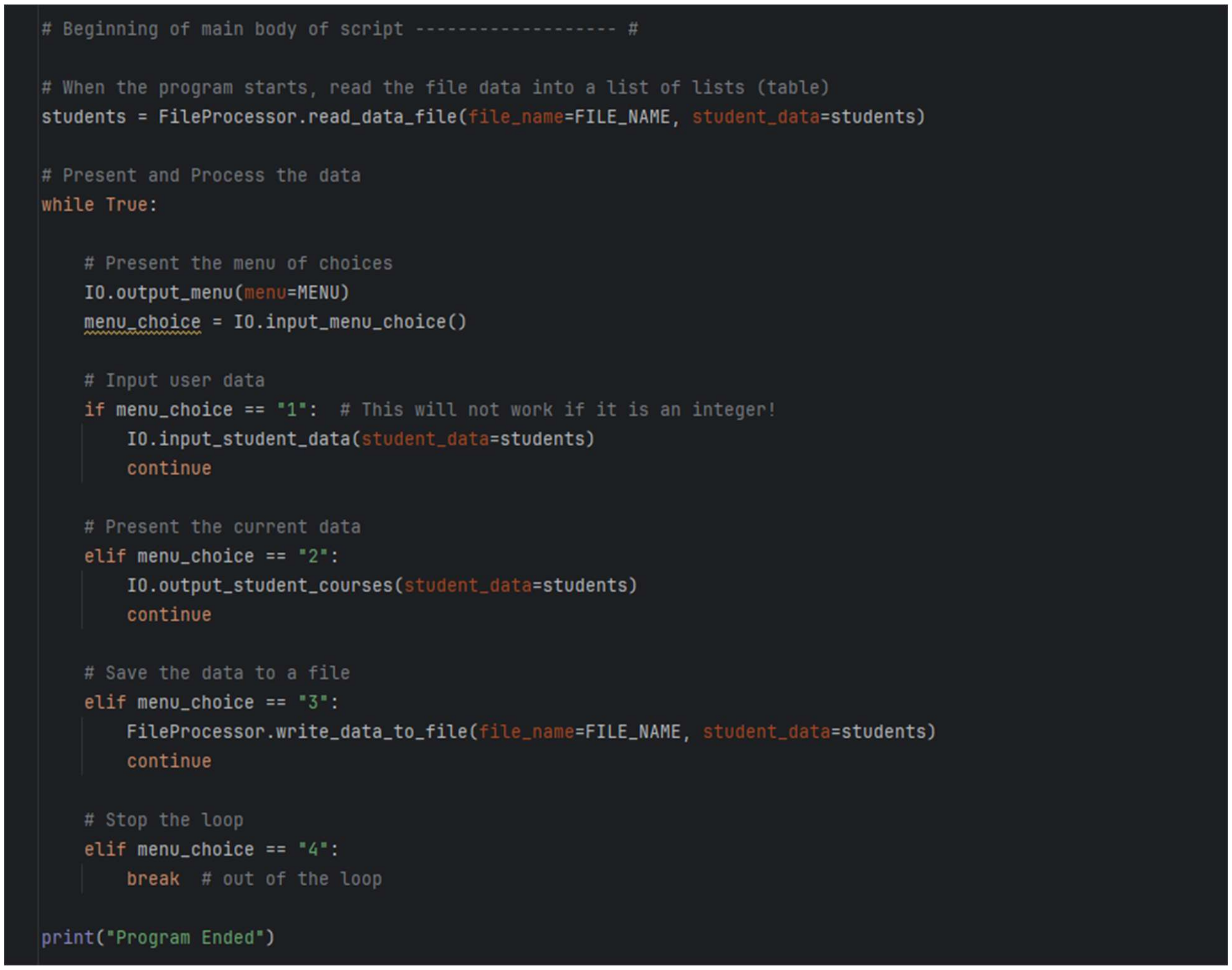
```
@staticmethod  1 usage
def output_student_courses(student_data: list):
    """
    Displays currently stored student courses.
    :param student_data: list
    """
    print("-" * 50)
    for student in student_data:
        print(f'Student {student["FirstName"]} '
              f'{student["LastName"]} is enrolled in {student["CourseName"]}')
    print("-" * 50)
```

Figure 2.4: Defining the output_student_courses function

These functions perform similar actions to code we have included in previous assignments. This time, they can be called as-needed later in the main script body.

Script Body

With all classes and functions defined, the main script body becomes much shorter and readable. Like with previous assignments, it reads in a data file, displays a menu, prompts the user for a selection, and then inputs, displays, or writes data to a file, depending on the selection. Now, instead of long sections written for each option, we simply call the functions that were defined previously.



```
# Beginning of main body of script ------------------- #

# When the program starts, read the file data into a list of lists (table)
students = FileProcessor.read_data_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!
        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

print("Program Ended")
```

Figure 3: The script body

Summary

Each week we learn a new technique that improves our code, but this week felt like a significant step forward with functions and script organization. The improvements to readability alone were massive, and I can see why concepts like the separation of concerns are best practices. I feel like I could hand my code off to someone who was unfamiliar with it, and they would be able to interpret what the program does more effectively, and vice versa.