Name: Cari Williams

Date: 20 November 2025

Course: Foundations Of Programming with Python

Github: https://github.com/carilynette/IntroToProg-Python-Mod06

A06: Working with Functions, Classes & SoC

Introduction

This assignment builds on the previous labs by expanding the program into a more structured design using functions, classes, and Separation of Concerns (SoC). The main goal was to learn how to organize code into clear layers—processing, input/output, and the main script—while also using structured error handling and working with JSON files. The program still allows the user to register a student for a course, view existing registrations, and save data to a file, but the structure is cleaner and easier to follow than in earlier labs.

Starter file and initialization

I started by creating a new Python file named *Assignment06_CWilliams.py* in my course folder. I copied in the starter structure, updated the header, and continued using *Enrollments.json* to store the student registration data. The initial setup followed the same pattern as the earlier assignments but with space to add the new class structure.

Using Classes and Separation of Concerns

FileProcessor Class (Processing Layer)

The assignment required splitting program logic into two main classes – FileProcessor Class (Processing Layer) and *IO Class* (Presentation Layer).

FileProcessor

The *FIleProcessor* class handles reading from and writing to the JSON file:

read_data_from_file(): Opens the JSON file, loads the student data, it handles exceptions for missing files and unexpected issues with a friendly message, then returns the updated student list.

write_data_from_file(): Writes the current list of students to the JSON file, handles JSON formatting issues and other exceptions, uses a finally block to ensure files close properly

```python
@staticmethod
def read_data_from_file(file_name: str, student_data: list):
    """
    Reads student data from a JSON file and returns an updated list.

    :param file_name:  name of the JSON file to read from.
    :param student_data: the current list of students.
    :return: the list of students in the file already.
    """
    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
```

```python
@staticmethod
def write_data_to_file(file_name: str,student_data:list):
    """
    Writes the current list of students into the JSON file.
    :param file_name: the JSON file name being written.
    :param student_data: the list of students to be saved.
    :return:
    """

    try:
        file = open(file_name, "w")
        json.dump(student_data, file)
        file.close()
    except TypeError as e:
        IO.output_error_messages( message: "Please check that the data is a valid JSON format", e)
    except Exception as e:
        IO.output_error_messages( message: "There was a non-specific error!", e)
    finally:
        if file.closed == False:
            file.close()
```

Keeping all file-related logic inside this class makes the code simpler to reuse and keeps the main program clean.

IO Class (Presentation Layer)

The *IO Class* (Presentation Layer) manages input and output, including menus, error messages, and student data entry. Key functions include:

- output_menu() – Displays the main program menu

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

    Changelog (Who, When, What)
    CWilliams, 11/20/25, Added a function to display the menu text
    CWilliams,11/20/25,Added a function to display custom error messages

    :param menu: the menu being displayed
    :return: None
    """
    print()
    print(menu)
    print()
```

- input_menu_choice() – Gets the user's choice and validates it.

```python
@staticmethod  1 usage
def input_menu_choice():
    """This function displays a custom menu to the user
    Gets a menu choice from the user, with basic validation.

    Changelog (Who, When, What)
    CWilliams, 11/20/25, Added a function to display the data

    :return: the user's menu choice as a string
    """
    menu_choice: str = "0"
    try:
        menu_choice = input("Enter menu choice: ")
        if menu_choice not in ("1","2","3","4"):
            raise Exception("Invalid option. Please only choose 1, 2, 3 or 4.")
    except Exception as e:
            IO.output_error_messages(str(e), e)

    return menu_choice
```

- output_error_messages() – Prints friendly and technical error details

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

    Changelog (Who, When, What)
    CWilliams, 11/17/25, Created function , added function to display the message.

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

```python
@staticmethod
def input_student_data(student_data: list):
    """ This function gets the student's first and last name and a course name from the user
    :param student_data: the current list of students.
    :return: the list after adding the new student registration.
    """

    try:
        # Get the first name
        student_first_name = input("Enter the student's first name: ")
        if not student_first_name.isalpha():
            raise ValueError("Only alphabetic characters are allowed.")

        # Get the last name
        student_last_name = input("What is the student's last name? ")
        if not student_last_name.isalpha():
            raise ValueError("The last name should not contain numbers.")

        # Get the Course Name
        course_name = input("What is the name of the course? ")

        # Build a student dictionary
        new_student = {"FirstName": student_first_name,
                       "LastName": student_last_name,
                       "CourseName": course_name}

        # Add the new student to the existing list
        student_data.append(new_student)
    except ValueError as e:
        IO.output_error_messages( message: "That value is not the correct type of data!", e)
    except Exception as e:
        IO.output_error_messages( message: "There was a non-specific error!", e)
    return student_data
```

- input_student_data() – Gets and validates user input for first name, last name, and course name

```python
@staticmethod  2 usages
def output_student_courses(student_data: list):
    """This function displays each student's name and courses.

    Changelog (Who, When, What)
    CWilliams, 11/20/25, Added a function to display the data
    :param student_data: the list of students records (dicts).
    :return: None
    """
    for student in student_data:
        print(student["FirstName"], student["LastName"], student["CourseName"])
```

- output_student_courses() – Displays all registered students and their courses

Separating these functions makes the program easier to read, test, and maintain.

Main Script Logic

The main script coordinates the program by:

1. Loading existing data

```python
#  End of function definitions
# --- Main Script -------------------------------- #

# 1. Load any existing students from the JSON file into students
students = FileProcessor.read_data_from_file(file_name=FILE_NAME, student_data=students)
```

2. Running a loop that displays the menu and responds to the user's choices:

   Option 1: Register a new student. Calls IO.input_student_data() then displays the updated list.

   Option 2: Show current registrations. Calls IO.output_student_courses()

   Option 3: Save data to the JSON file. Calls *FileProcessor*.write_data_to_file()

Option 4: Exit the program

```
#2 Main program loop
while (True):

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

    # Register a new student, then display updated student_data
    if menu_choice == "1":
        students= IO.input_student_data(student_data=students)
        IO.output_student_courses(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
    else:
        print("Please only choose option 1, 2, 3 or 4")

print("Program Ended")
```

Through this assignment, I learned how to:

- Structure a program using classes and Separation of Concerns
- Pass data in and out of functions effectively
- Use structured error handling to deal with different types of problems

- Write clearer docstrings and explain the logic of each part of the program

I also gained more confidence in debugging and understanding how each component of the program communicates with the others.

Summary

Assignment 06 brought together everything from the earlier labs and introduced a more professional program structure. The script is easier to understand and maintain with by separating responsibilities into classes and functions. This assignment helped solidify my understanding of program architecture and error handling, preparing me for more complex Python programs.