Hao Yi Fei

11/12/2025

Introduction to Programming with Python

Assignment 06

https://github.com/Portter-fly/IntroToProg-Python-Mod06.git

The Student Registration Program with Functions, Classes, and Separation of Concerns

Intro

This week's assignment built on Assignment05's dictionary and JSON functionality, but added a big new challenge: organizing code with functions, classes, and the separation of concerns pattern. This assignment also required writing descriptive document strings for every class and function, which forced me to think clearly about what each part does. Below is a step-by-step breakdown of how I wrote the program, including the mistakes I made and how I fixed them.

Creating the Program

With the instruction of the assignment, I began iterating the Student Registration program. The first step was updated the script header. I followed the assignment's template, updating the date, and what change I've done to the script. (Figure 1.1)

```
# ------------------------------------------------------------------------------------------ #
# Title: Assignment06
# Desc: This assignment demonstrates using functions
# with structured error handling
# Change Log: (Who, When, What)
#   RRoot,1/1/2030,Created Script
#   Hao Yi Fei,11/12/2025, iteration
# ------------------------------------------------------------------------------------------ #
```

Figure 1.1: Script header which has been updated.

Defining Constants

The constants were almost the same as Assignment05, and I just removed the TECH_ERROR_MESSAGE from Assignment05 (Figure 1.2)

```
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.json"
NON_SPEC_ERROR_MESSAGE: str = "There was a non-specific error!"
```

Figure 1.2: Constant definitions including menu, file name, and error messages

Initializing Variables

I initialized the required variables as specified in the assignment: menu_choice as an empty string, students as an empty list (to hold dictionaries of student data), and success as a boolean. (Figure 1.3)

```
# Define the Data Variables
students: list = []  # a table of student data
menu_choice: str = ''  # Hold the choice made by the user.
success: bool = False  # Holds success boolean of file write operation
```

Figure 1.3: Variable initialization for menu choice, student data, and save status

Creating the FileProcessor Class

This class was really new. And I had made a mistake, which was: in read_data_from_file, I used student_data = json.load(file) instead of student_data.extend(json.load(file)). This meant existing data was lost when reading from the file. And I fixed it by switching to extend(), which keeps old registrations. (Figure 1.4)

```python
class FileProcessor:  2 用法
    """
    A collection of functions that perform File Processing
    """

    @staticmethod  1 个用法
    def read_data_from_file(file_name: str, student_data: list) -> list:
        """ This function reads data from file
            param file_name: string of the file name
            param student_data: list of student data
            return: list of student data"""

        file = _io.TextIOWrapper
        try:
            file = open(file_name, "r")
            student_data.extend(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(NON_SPEC_ERROR_MESSAGE, e)
        finally:
            if file.closed == False:
                file.close()
        return student_data

    @staticmethod  1 个用法
    def write_data_to_file(file_name: str, student_data: list) -> bool:
        """ This function write data to file
            param file_name: string of the file name
            param student_data: list of student data
            return: boolean status of successful write operation"""
```

```python
        file = _io.TextIOWrapper

        try:
            file = open(file_name, "w")
            json.dump(student_data, file, indent=4)
            file.close()
            return True
        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(NON_SPEC_ERROR_MESSAGE, e)
        finally:
            if file.closed == False:
                file.close()
        return False
```

Figure 1.4: File Processor class
with read_data_from_file and write_data_to_file methods

Creating the IO Class

The IO class handles everything related to the user: showing the menu, getting choices, collecting student data, displaying registrations, and showing errors. I added five static

methods as required. Each method has a document string explaining its purpose, parameters, and return value (Figure 1.5)

```
class IO:  12 用法
    """
    A collection of functions that present or gather user input and output"""

    @staticmethod  7 用法
    def output_error_messages(message: str, error: Exception = None) -> None:
        """ This function displays a custom error messages to the user
            param message: Custom Error Message
            param error: Exception
            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 个用法
    def output_menu(menu: str) -> None:
        """ This function displays the menu of choices to the user
        param menu: Menu string
        return: None"""

        print()
        print(menu)

    @staticmethod  1 个用法
    def input_menu_choice() -> str:
        """ This function gets a menu choice from the user
        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 the exception object to avoid the technical message
        return choice
```

```python
@staticmethod  1 个用法
def input_student_data(student_data: list) -> list:
    """ This function gets the student data from the user
    param student_data: list of student data
    return: list of 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: ")
        new_student = {"FirstName": student_first_name,"LastName": student_last_name,"CourseName": course_name
        student_data.append(new_student)
    except ValueError as e:
        IO.output_error_messages(e.__str__())

    except Exception as e:
        IO.output_error_messages( message: "Error: There was a problem with "
                                  "your entered data.", e)
    return student_data

@staticmethod  2 用法
def output_student_courses(student_data: list) -> None:
    """ This function displays the student courses
    param student_data: list of student data
    return: None"""
    print("-" * 50)
    for student_row in student_data:
        print(student_row["FirstName"], student_row["LastName"],
              student_row["CourseName"], sep=",")
    print("-" * 50)
```

Figure 1.5: IO class with all input/output and error-handling functions

Loading Existing Data on Startup

Just like Assignment05, the program needs to load data from Enrollments. json when it starts. But now, instead of writing the file-reading code directly in the main script, I called File Processor. read_data_from_file. This kept the main code clean and followed separation of concerns. (Figure 1.6)

```python
# Start by reading JSON file of Enrollments
students = FileProcessor.read_data_from_file(file_name=FILE_NAME, student_data=students)
```

Figure 1.6: Code to load JSON data)

Implementing the Main Menu Loop

Instead of having all the logic directly in the loop, I just called methods from the IO and File Processor classes. (Figure 1.7)

```
# Present and Process the data
while (True):

    IO.output_menu(menu=MENU)
    menu_choice = IO.input_menu_choice()
```

Figure 1.7: Main menu loop calling class methods for all functionality

Handling Menu Choice

Menu choice 1, 2, 3 are all much simpler. (Figure 1.7)

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

    elif menu_choice == "2":
        IO.output_student_courses(student_data=students)
        continue

    elif menu_choice == "3":
        success = FileProcessor.write_data_to_file(file_name=FILE_NAME, student_data=students)
        if success:
            print("Written to File:")
            IO.output_student_courses(student_data=students)
        continue

    elif menu_choice == "4":
        break  # out of the loop

print("Program Ended")
```

Figure 1.9: Logic for menu choice

Testing the Program

After writing the code, I tested it thoroughly to make sure all features worked. Here's what I did:

PyCharm Run Test

I ran the program in PyCharm and tested all menu options:

- Registered multiple students
- Tried entering names with numbers
- Showed current data to confirm all registrations appeared

- Saved data to Enrollments. json
- Exited and restarted the program to check if data loaded correctly (Figure 2.1)

```
---- 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.
----------------------------------------

Enter your menu choice number: 1
Enter the student's first name: Hao
Enter the student's last name: Yifei
Please enter the name of the course: python


---- 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.
----------------------------------------
```

```
Enter your menu choice number: 2
-------------------------------------------------
Bob,Smith,Python 100
Sue,Jones,Python 100
Hao,Yifei,python
-------------------------------------------------


---- 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.
----------------------------------------

Enter your menu choice number: 3
Written to File:
-------------------------------------------------
Bob,Smith,Python 100
Sue,Jones,Python 100
Hao,Yifei,python
-------------------------------------------------


---- 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.
----------------------------------------
```

```
Enter your menu choice number: 4
Program Ended
```

Figure 2.1: PyCharm run with all functions

I then checked the JSON file. (Figure 2.2)

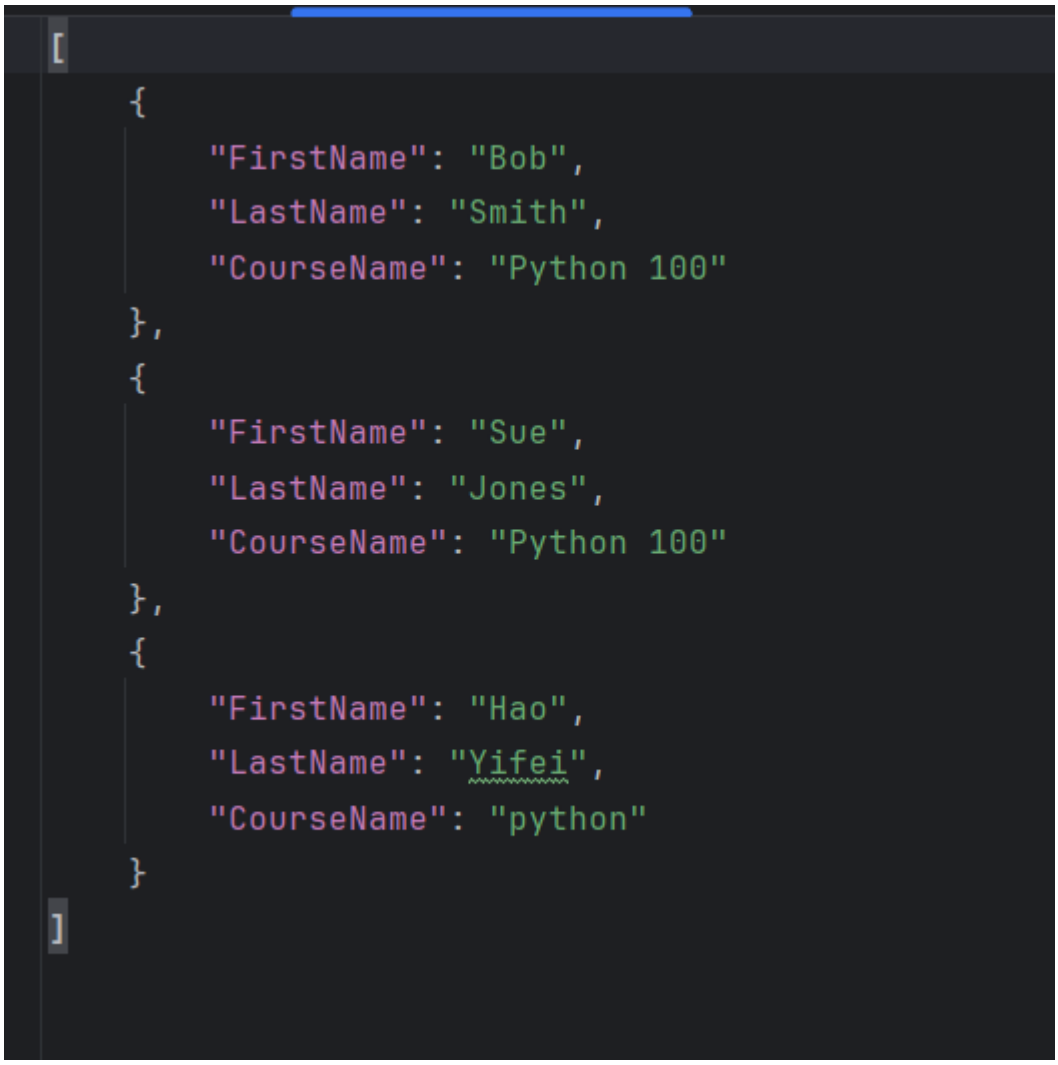

Figure 2.2: Contents of Enrollments.json showing saved student data

Summary

This assignment taught me so much about organizing code! I now understand that classes are just ways to group related functions. The separation of concerns pattern made my code easier to read and fix. I also learned the importance of document strings, which helped me catch mistakes early. Overall, I think this exercise built a strong foundation for creating and managing Python scripts in the future.