Mariusz Sokol

Mar, 5, 2025

Writing a paper

Assingnment06

Github link: https://github.com/Mariusz-uw/Python110-Winter-2025

Introduction

This assignment documents Python code that handles user input and output. It helps user with error events. These operations processed using classes and functions.

Module import, menu string and constant value

A JSON module import. Constant string displaying a user-friendly menu. Constant string “FILE_NAME” containing the filename (Enrollments.json) stores the registration data.

```
import json

# 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"
```

Figure 1: JSON import, Menu , and FILE_NAME: str constant.

Input and output class and its functions

The IO class is responsible for handling all input and output operations in the program. It provides methods for displaying menus, collecting user input, showing student data, and managing error messages. The “output_error_messages” function displays error messages in case any occur. The “output_menu” function displays main menu options to the user. The “input_menu_choice" function retrieves the user’s selected menu option. The “output_student_courses” function displays a formatted list of registered students and their courses. The “input_student_data" function collects student details from the user (first name, last name and course),

validates input, and adds it to the data list, including basic error checking and validation (e.g., ensuring that names are alphabetic).

```python
class IO:
    """Handles all input and output operations."""

    @staticmethod
    def output_error_messages(message: str, error: Exception = None):
        """
        Displays an error message.

        :param message: The custom error message.
        :param error: The exception object (optional).
        """
        print(f"Error: {message}")
        if error:
            print("-- Technical Error Message --")
            print(error.__doc__)
            print(error.__str__())

    @staticmethod
    def output_menu():
        """
        Displays the program menu.
        """
        print(MENU)

    @staticmethod
    def input_menu_choice():
        """
        Gets the user's menu choice.

        :return: The menu choice as a string.
        """
        return input("What would you like to do: ")

    @staticmethod
    def output_student_courses(student_data: list):
        """
        Displays the list of registered students and their courses.

        :param student_data: List of dictionaries containing student details.
        """
        print("-" * 50)
        for student in student_data:
            print(f'Student {student["FirstName"]} '
                  f'{student["LastName"]} is enrolled in '
                  f'{student["CourseName"]}')
        print("-" * 50)
```

Figure 2: IO class containing functions for IO operations.

```python
    @staticmethod
    def input_student_data(student_data: list):
        """
        Collects student details from the user.

        :param student_data: List to store student records.
        """
        try:
            first_name = input("Enter the student's first name: ")
            if not first_name.isalpha():
                raise ValueError("The first name should not contain numbers.")
            last_name = input("Enter the student's last name: ")
            if not last_name.isalpha():
                raise ValueError("The last name should not contain numbers.")
            course_name = input("Please enter the name of the course: ")

            student = {"FirstName": first_name, "LastName": last_name,
                       "CourseName": course_name}
            student_data.append(student)

            print(f"You have registered {first_name} {last_name} for "
                  f"{course_name}.")
        except ValueError as e:
            IO.output_error_messages(message: "Invalid input provided.", e)
        except Exception as e:
            IO.output_error_messages(message: "An unexpected error occurred while "
                                              "entering student data.", e)
```

Figure 3: IO class continuing, input_student_data function with operations code for error handling.

File Processor class and its functions

The “read_data_from_file” function reads existing student-course registration data from a JSON file and loads it into a list . Includes error handling for scenarios such as file not found (FileNotFoundError) and malformed JSON data (JSONDecodeError). The “write_data_to_file” function writes the current registration data to JSON file, allowing persistent storage. It includes handling potential exceptions (file writing errors) and confirmation messages upon successful save.

```
class FileProcessor:
    """Handles reading from and writing to files."""

    @staticmethod
    def read_data_from_file(file_name: str, student_data: list):
        """
        Reads student data from a JSON file.

        :param file_name: The file name to read from.
        :param student_data: The list to store retrieved student records.
        """
        try:
            with open(file_name, "r") as file:
                student_data.extend(json.load(file))
        except FileNotFoundError:
            IO.output_error_messages("File not found. A new file will be "
                                     "created.")
        except json.JSONDecodeError:
            IO.output_error_messages("Error decoding JSON. Ensure the file "
                                     "is properly formatted.")
        except Exception as e:
            IO.output_error_messages(message:"An error occurred while reading "
                                     "the file.", e)

    @staticmethod
    def write_data_to_file(file_name: str, student_data: list):
        """
        Writes student data to a JSON file.

        :param file_name: The file name to write to.
        :param student_data: The list of student records to save.
        """
        try:
            with open(file_name, "w") as file:
                json.dump(student_data, file)
            print("The following data was saved to file!")
            IO.output_student_courses(student_data)
        except Exception as e:
            IO.output_error_messages(message:"An error occurred while writing "
                                     "to the file.", e)
```

Figure 4: FIleProcessor class with functions for data handling and error handling code.

Main Program Logic

Initializes an empty list “students” for storing student registration records. Loads existing data from JSON file (Enrollments.json) into the list at startup. Provides a continuous loop (while True) that presents the menu, captures user choices, and executes corresponding actions.

Option 1: Collects and validates new student-course registrations.

Option 2: Displays the current list of registrations.

Option 3: Saves the current registration data to a file (Enrollments.json).

Option 4: Exits the application.

Any other input: Prompts the user to provide a valid menu option.

```
# Main Program Logic
students = []
FileProcessor.read_data_from_file(FILE_NAME, students)

while True:
    IO.output_menu()
    choice = IO.input_menu_choice()

    if choice == "1":
        IO.input_student_data(students)
    elif choice == "2":
        IO.output_student_courses(students)
    elif choice == "3":
        FileProcessor.write_data_to_file(FILE_NAME, students)
    elif choice == "4":
        print("Program Ended")
        break
    else:
        print("Please only choose option 1, 2, 3, or 4.")
```

Figure 5: While loop connecting input values to respective classes and their functions.

Command Prompt Testing

Bellow, the script tested in command prompt. The script displays the same functionality as it does, when running in the PyCharm IDE console.

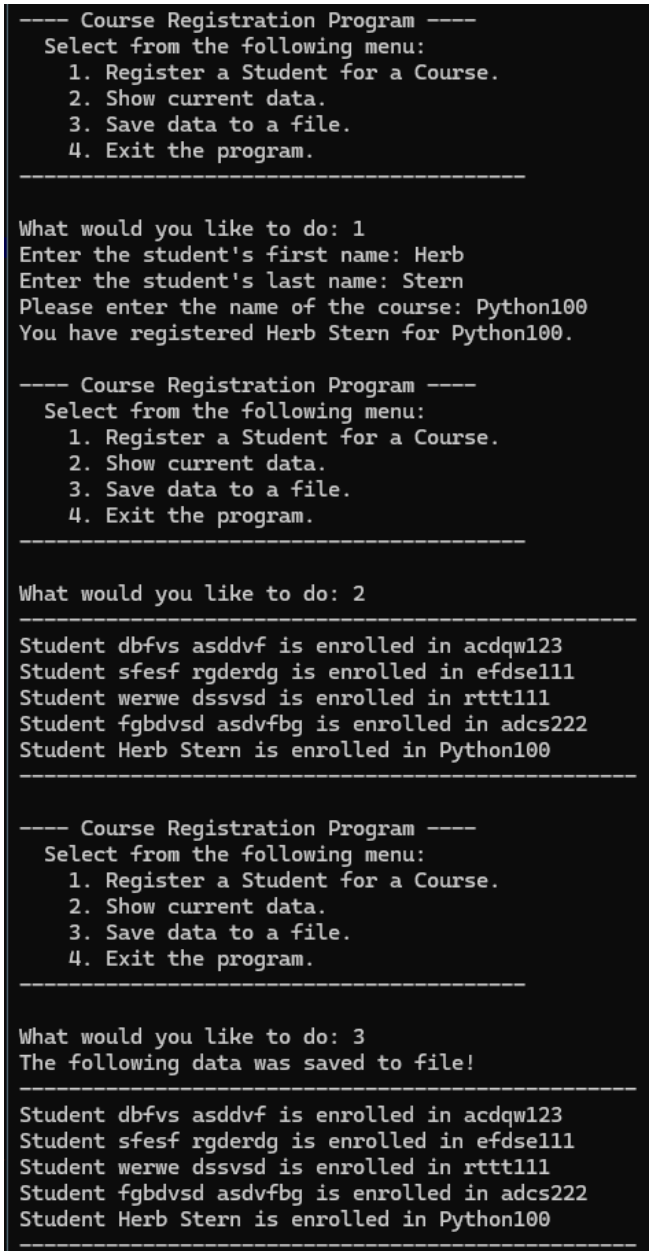

Figure 6: Option one, two, and tree outcomes.

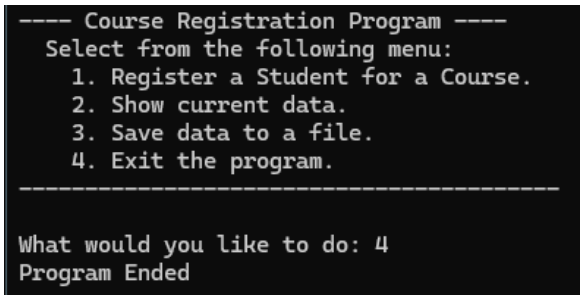

Figure 7: Option four outcome.

Summary

The program provides a clear, structured example of managing simple data registration operations with reliable persistence, thorough input validation, and consistent error handling practices.