Jeisson Guerrero Estrada

02/21/24

Python 100

Assignment 06

Functions

Introduction

I developed a Python code for a course registration program with functions and error handling. It allows users to register students, view enrollment data, and save information to a JSON file. The program includes important data constants and two classes for managing file operations and user interactions. Functions are used to handle tasks like reading and writing to a JSON file, displaying menus, managing user input, and handling errors. The main loop continuously presents the menu and adapts to user choices. The accompanying documentation provides an overview, usage instructions, details on constants, and explanations of classes and functions. The goal is to make course registration accessible and user-friendly.

Data Constants & File Name Definition

The script starts by defining important data constants. The MENU constant contains a well-formatted menu, which gives users clear options to register students, view data, save to a file, or exit the program. The FILE_NAME constant defines the name of the

JSON file as "Enrollments.json," which highlights the program's ability to store data persistently.

```
import json

# Define the Data Constants
MENU = '''
---- 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 = "Enrollments.json"

# Define the Data Variables and constants
students = []  # a table of student data
```

FIGURE 1

Class: FileProcessor for JSON File Handling

The following script introduces the FileProcessor class, which is specifically designed to facilitate the interaction with JSON files. This class defines two crucial functions. The first one is the read_data_from_file function which reads data from a specified file, making it easier to initialize the students list. On the other hand, the write_to_file function writes the current student data to the designated JSON file. This structured file processing ensures seamless data retrieval and storage.

```
class FileProcessor:
    """
    Processing function that works with JSON Files

    ChangeLog: Jeisson Guerrero Estrada 02/21/24
    """
    @staticmethod
    def read_data_from_file(file_name, student_data):
        """
        Read data from JSON Files
        :param file_name: string data from file name to read from
        :param student_data: list of dictionary with student data
        :return: list of dictionaries with student data
        """
        try:
            # Open the file for reading
            with open(file_name, "r") as file:
                # Load the JSON data from the file into the student_data list
                student_data = json.load(file)
```

```
        except Exception as e:
            # Handle exceptions, output an error message
            IO.output_error_messages(message="Error: There was a problem with
reading the file.", error=e)

        return student_data

    @staticmethod
    def write_to_file(file_name, student_data):
        """
        Write data to JSON Files
        :param file_name: String data from file name to write to
        :param student_data: list of dictionaries with student data
        :return: none
        """
        try:
            # Open the file for writing
            with open(file_name, "w") as file:
                # Write the student_data list as JSON to the file
                json.dump(student_data, file)
            # Output student and course names after writing to the file
            IO.output_student_and_course_names(student_data=student_data)
            print("Data successfully saved to the file.")
        except Exception as e:
            # Handle exceptions, output an error message
            message = "Error: There was a problem with writing to the file.
\n"
            message += 'Please check that the file is not open by another
program.'
            IO.output_error_messages(message=message, error=e)
```

FIGURE 2

Class: IO for User Interaction & Error Handling

The IO class is a crucial component that is responsible for various user interactions, such as menu display, input validation, and error handling. This class includes significant functions such as "output_error_messages" for displaying error messages, "output_menu" for presenting the program menu, "get_menu_choice" for obtaining and validating user preferences, "output_student_and_course_names" for showing student names and enrolled courses, and "input_student_data" for acquiring and validating student registration data. Adopting this modular approach enhances code readability and maintainability, making it easy to understand and maintain the program over time.

```
class IO:
    @staticmethod
    def output_error_messages(message, error=None):
        """
        Print error
        :param message: string with message
        :param error: Technical error message
        :return: None
        """
        # Output general error message and technical error details
        print(message, end='\n\n')
        if error is not None:
            print('-- Technical Error Message --')
            print(error, str(error), type(error), sep='\n')
```

```
    @staticmethod
    def output_menu():
        """
        Print menu
        :return: None
        """
        # Output the program menu
        print(MENU)

    @staticmethod
    def get_menu_choice():
        """
        Get user input for menu choice
        :return: User's Choice
        """
        try:
            # Get user input for menu choice and validate it
            choice = input("Enter your menu choice number: ")
            if choice not in ('1', '2', '3', '4'):
                raise Exception('Please, choose from the options of 1, 2, 3,
or 4')
        except Exception as e:
            # Handle exceptions related to user input
            IO.output_error_messages(str(e), e.__doc__)
            return IO.get_menu_choice()  # Ask for input again if there's an
error
        return choice

    @staticmethod
    def output_student_and_course_names(student_data):
        """
        Print student and course name
        :param student_data: List of dictionaries containing rows
```

```
        :return: none
        """
        # Output student names and their enrolled courses
        print('-' * 50)
        for student in student_data:
            print(f'student {student["FirstName"]} '
                  f'{student["LastName"]} is enrolled in
{student["CourseName"]}')
        print('-' * 50)


    @staticmethod
    def input_student_data(student_data):
        """
        Print student
        :param student_data: List of dictionaries containing rows of input
data
        :return: list
        """
        try:
            # Get user input for student data and validate it
            student_first_name = input("Enter your first name: ")
            if not student_first_name.isalpha():
                raise ValueError('The first name should not contain
numbers.')
            student_last_name = input("Enter your last name: ")
            if not student_last_name.isalpha():
                raise ValueError('The last name should not contain numbers')
            course_name = input("Enter your course name: ")
            # Create a dictionary for the new student and add it to the
student_data list
            student = {'FirstName': student_first_name, 'LastName':
student_last_name, 'CourseName': course_name}
            student_data.append(student)
            print(f'You have registered {student_first_name}
{student_last_name} for {course_name}.')
```

```
        except (ValueError, Exception) as e:
            # Handle exceptions related to incorrect data input
            IO.output_error_messages(message='One of the values was not the
correct type of data!', error=e)
        return student_data
```

FIGURE 3

Main Program Logic

The main program loop serves as the conductor of the script's overall flow. It is triggered by reading initial data from the JSON file into the students list. This loop maintains the interactive nature of the program by continuously displaying the menu, collecting user choices, and directing execution based on the selected option. This dynamic loop structure guarantees a responsive and user-friendly experience.

```
# Read initial data from the file into the students list
students = FileProcessor.read_data_from_file(file_name=FILE_NAME,
student_data=students)

# Main program loop
while True:
```

FIGURE 4

User Registration, Data Display, & File Saving

The main program loop comprises specific conditional blocks that handle user choices. The "Register a Student for a Course" option enables users to input and register new students by invoking the input_student_data function. The "Show current data" option calls the output_student_and_course_names function to display the names of students and the courses they are enrolled in. The "Save data to a file" option utilizes the write_to_file function from the FileProcessor class to store the current data persistently in the specified JSON file. Finally, when selecting the "Exit the program" option, the script gracefully concludes.

```
# Main program loop
while True:
    # Display the program menu
    IO.output_menu()
    # Get user input for menu choice
    menu_choice = IO.get_menu_choice()

    if menu_choice == "1":
        # Register a new student for a course
        students = IO.input_student_data(student_data=students)
        continue

    elif menu_choice == "2":
        # Show current student data
        IO.output_student_and_course_names(students)
        continue
```

```
    elif menu_choice == "3":
        # Save data to a file
        FileProcessor.write_to_file(file_name=FILE_NAME,
student_data=students)
        continue


    elif menu_choice == "4":
        # Exit the program
        print("Program Ended, Goodbye!")
        break


    else:
        print("Please only choose option 1, 2, 3, or 4”)
```

FIGURE 5

Summary

The developed Python code offers a functional and user-friendly course registration program. It allows users to register students, manage enrollment data, and save information to a JSON file. The code's well-defined classes enhance organization and readability. Comprehensive documentation offers usage instructions and explanations of key components. The aim is to provide an efficient tool for handling course registrations with ease of understanding and flexibility for future enhancements.