Phuong Tran (Eric)

November 17, 2024

IT FDN 110 A

Assignment 05

GitHubURL: https://github.com/erictran03/IntroToProg-Python-Mod05

Working with Dictionaries, JSON Files, and Exception Handling

Introduction

In Assignment 05, I extended the functionality of my previous Course Registration program by incorporating dictionaries, JSON file handling, and structured error handling. The focus of this assignment was to improve data organization using dictionaries, implement persistent data storage using JSON files, and handle errors gracefully with try-except blocks. This document explains the steps I took to complete the assignment, what I learned during the process, and how the new features enhanced the overall program.

Enhancements Made to the Previous Version

In the previous assignment (Assignment 04), I used lists to store student data and saved the information in a CSV file format. This time, I made significant changes:

1. **Using Dictionaries for Data Storage:**

   Instead of storing student information in a list, I used dictionaries (Figure 1). This allowed me to use key-value pairs for storing the first name, last name, and course name of each student, making the data structure more descriptive and easier to manage.

```
student_data = {
    "FirstName": student_first_name,
    "LastName": student_last_name,
    "CourseName": course_name
}  # Create a dictionary for the student's data.
```

Figure 1: Create Dictionaries for Data Storage

2. **Switching from CSV to JSON for Data Storage:**

   I replaced the CSV file with a JSON file (Enrollments.json). Using JSON provided a more structured format for storing complex data (i.e., a list of dictionaries). JSON is also more versatile for data interchange between different systems.

3. **Implementing Structured Error Handling:**

I added try-except blocks throughout the program to handle various types of errors, such as file not found (FileNotFoundError), invalid JSON data (JSONDecodeError), and input validation errors (ValueError). This made the program more robust and user-friendly.

Loading Data from JSON File

At the start of the program, I included code to read existing student data from the JSON file. If the file does not exist, it creates a new empty file. If the file contains invalid JSON data, it resets the student list. As shown in Figure 2, I used FileNotFoundError to handle cases where the file is missing. Also, I used json.JSONDecodeError to handle cases where the JSON data is corrupted or incorrectly formatted.

```
try:
    file = open(FILE_NAME, "r")       # Open the JSON file in read mode.
    json_data = file.read()           # Read the entire file content as a string.
    students = json.loads(json_data)  # Parse the JSON string into a list of dictionaries.
except FileNotFoundError:
    print('File not found. Creating a new file...')  # If file does not exist, print a message.
    open(FILE_NAME, 'w').close()                     # Create an empty file if it does not exist.
    students = []                                    # Initialize the students list as empty.
except json.JSONDecodeError:
    print('JSON decoding error. Resetting the data...')  # If JSON is invalid, reset the data.
    students = []                                        # Initialize the students list as empty.
```

Figure 2: Error Handling

Main Program Loop and Menu Options

I structured the program into a main loop with four menu options, allowing users to register a student, display the current data, save the data to a file, or exit the program.

1. **Register a Student (Option 1):**

I prompted the user to enter the first name, last name, and course name. The inputs were validated to ensure the names contain only alphabetic characters (Figure 3).

```
# Option 1: Register a student
if menu_choice == "1":
    try:
        student_first_name = input("Enter the student's first name: ").strip()  # Get the student's first name.
        if not student_first_name.isalpha():                                    # Check if the name is alphabetic.
            raise ValueError('First name must be alphabetic')                   # Raise an error if invalid.

        student_last_name = input("Enter the student's last name: ").strip()    # Get the student's last name.
        if not student_last_name.isalpha():                                     # Check if the name is alphabetic.
            raise ValueError('Last name must be alphabetic')                    # Raise an error if invalid.

        course_name = input("Please enter the name of the course: ").strip()    # Get the course name.
        student_data = {
            "FirstName": student_first_name,
            "LastName": student_last_name,
            "CourseName": course_name
        }  # Create a dictionary for the student's data.
        students.append(student_data)  # Add the student's data to the list.
        print(f"Registered {student_first_name} {student_last_name} for {course_name}.")  # Confirmation message.
    except ValueError as e:
        print(e)  # Print the error message.
```

Figure 3: Option 1 - Register a student

2. **Show Current Data (Option 2):**

This option iterates through the list of student dictionaries and prints each student's information. I used a try-except block to handle missing keys in the dictionary, which could occur if the data structure is modified or corrupted (Figure 4).

```
# Option 2: Display current data
elif menu_choice == "2":
    print("-" * 50)               # Print a separator line.
    if students:                  # Check if there are any students registered.
        for student in students:  # Loop through each student dictionary.
            try:
                print(
                    f"Student {student['FirstName']} {student['LastName']} is enrolled in {student['CourseName']}")
            except KeyError as e:
                print(f"Missing data: {e}")            # Handle missing keys in the dictionary.
    else:
        print("No students are currently registered.")  # Inform the user if the list is empty.
    print("-" * 50)                                    # Print a closing separator line.
```

Figure 4: Option 2 - Display Current Data

3. **Save Data to JSON File (Option 3):**

I used the json.dump() method to save the list of dictionaries to the JSON file. This approach made it easy to serialize the Python data structure into a JSON format. Additionally, I wrapped the file writing code in a try-except-finally block to handle any potential errors and ensure the file is always closed properly (Figure 5).

```
# Option 3: Save the data to a JSON file
elif menu_choice == "3":
    try:
        file = open(FILE_NAME, "w")                # Open the JSON file in write mode.
        json_data = json.dumps(students, indent=4)  # Convert the list of dictionaries to a JSON string.
        file.write(json_data)                      # Write the JSON string to the file.
        print("Data successfully saved to file.")  # Confirmation message.
    except Exception as e:
        print('Error saving data to file:', e)     # Print an error message if saving fails.
    finally:
        if file and not file.closed:               # Ensure the file is closed.
            file.close()                           # Close the file.
```

Figure 5: Option 3 - Save Data to JSON File

4. **Exit the Program (Option 4) and Handle Invalid Menu Choice:**

This option ends the main loop and exits the program. It provides a clear and user-friendly way to close the application. Also, in the main loop, I included a check for invalid menu choices. If the user enters something that is not "1", "2", "3", or "4", the program displays an error message informing them of the valid options. This ensures that the user is guided back to the menu instead of causing the program to behave unexpectedly (Figure 6).

```
    # Option 4: Exit the program
    elif menu_choice == "4":
        print("Exiting the program")                    # Exit message.
        break                                           # Exit the loop and end the program.

    # Invalid menu choice
    else:
        print("Please only choose option 1, 2, 3, or 4") # Inform the user of an invalid choice.

print("Program Ended")                                  # Final message when the program ends.
```

Figure 6: Option 4 - Exit the Program and Handle Invalid Menu Choice

Testing the Program

I tested the program by running it in both PyCharm and Terminal and entering multiple student records. I checked for the following scenarios:

- Registering students with valid and invalid input.
- Displaying the list of registered students, including cases where no students are registered.
- Saving the data to the JSON file and verifying its contents in a text editor.
- Handling file errors, such as missing files or corrupted JSON data.

All tests were successful, and the program handled errors gracefully, providing helpful feedback to the user.

Summary

This assignment allowed me to build upon the previous module by incorporating dictionaries for better data organization, using JSON for persistent data storage, and implementing structured error handling to enhance program reliability. I learned how to use the json module for serializing and deserializing data, how to validate user input more effectively, and how to handle exceptions in a Python program. These skills are essential for creating robust and user-friendly applications.