Name: Cathy D.
Date: July 30, 2024
Course: IT FDN 110 Su 24 Foundations of Programming Python

Assignment 05 – Advanced Collections and Error Handling

Introduction

In this assignment, I created a Python course registration program that demonstrates the use of dictionaries and structured error handling while performing file operations with JSON. Dictionaries in Python are useful because they allow for fast retrieval of values based on unique keys, which are better for storing and accessing data with a clear key-value relationship, unlike the lists from last week which are ordered collections accessed by index. The focus of this document are on these new topics for this week.

Topic – JSON reading and writing

Figure 1 includes the following parts. The file_name is opened and loaded into a Python data structure by reading the data using 'r'.

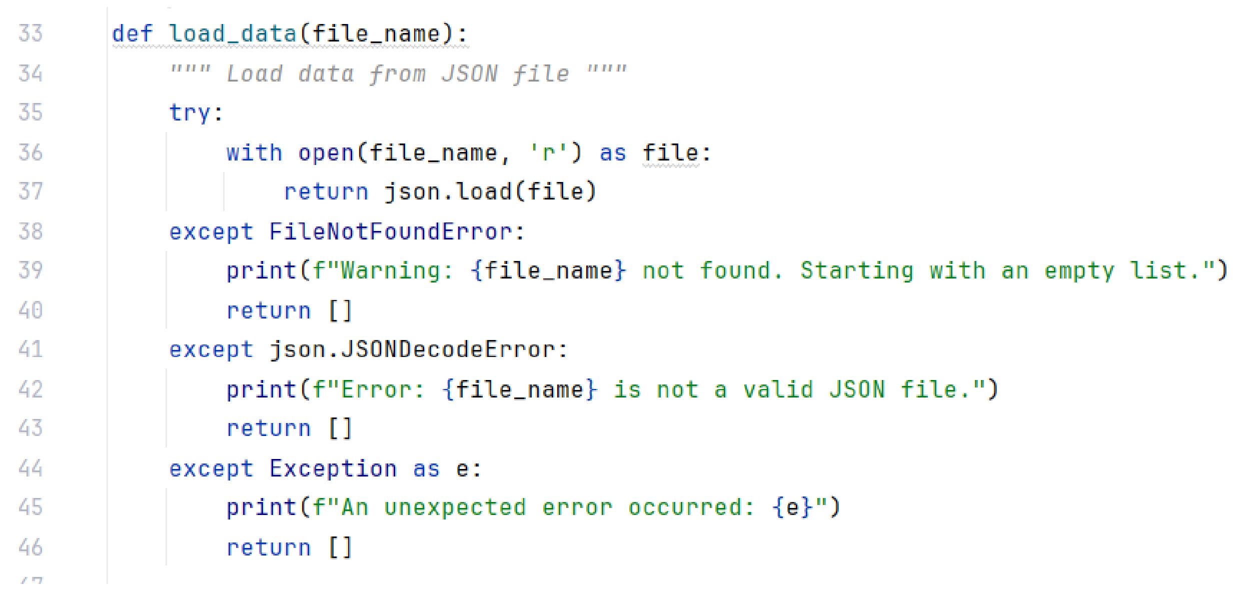
Load JSON Data: The json.load(file) function is used to take the JSON data and convert it into a Python dictionary.

Error Handling (important so program doesn't just crash):

FileNotFoundError: If a file is not found, then a warning message is printed, and an empty list is returned.

JSONDecodeError: If the file contains invalid JSON, an error message is printed, and an empty list is returned.

Any other issues: If there are any other unexpected errors that are caught, then an error message is printed, and an empty list is returned.



```python
def load_data(file_name):
    """ Load data from JSON file """
    try:
        with open(file_name, 'r') as file:
            return json.load(file)
    except FileNotFoundError:
        print(f"Warning: {file_name} not found. Starting with an empty list.")
        return []
    except json.JSONDecodeError:
        print(f"Error: {file_name} is not a valid JSON file.")
        return []
    except Exception as e:
        print(f"An unexpected error occurred: {e}")
        return []
```

Figure 1. Screen shot of JSON reading to open and give error messages if the file isn't found in PyCharm

Writing data to a JSON file involves opening the file, writing the data in JSON format, and handling any potential errors that may occur during this process. The save_data function in the script is responsible for writing data to the JSON file.

The save_data function follows these steps: (all Figure 2 below):

Open the File: The file is opened in write mode ('w'). If the file does not exist, it is created.

Write JSON Data: The json.dump(data, file, indent=4) method is used to convert the Python data structure into JSON format and write it to the file. The indent=4 parameter makes the JSON output more readable by adding indentation.

Error Handling:

TypeError: Catches errors related to incompatible data types during the conversion to JSON.

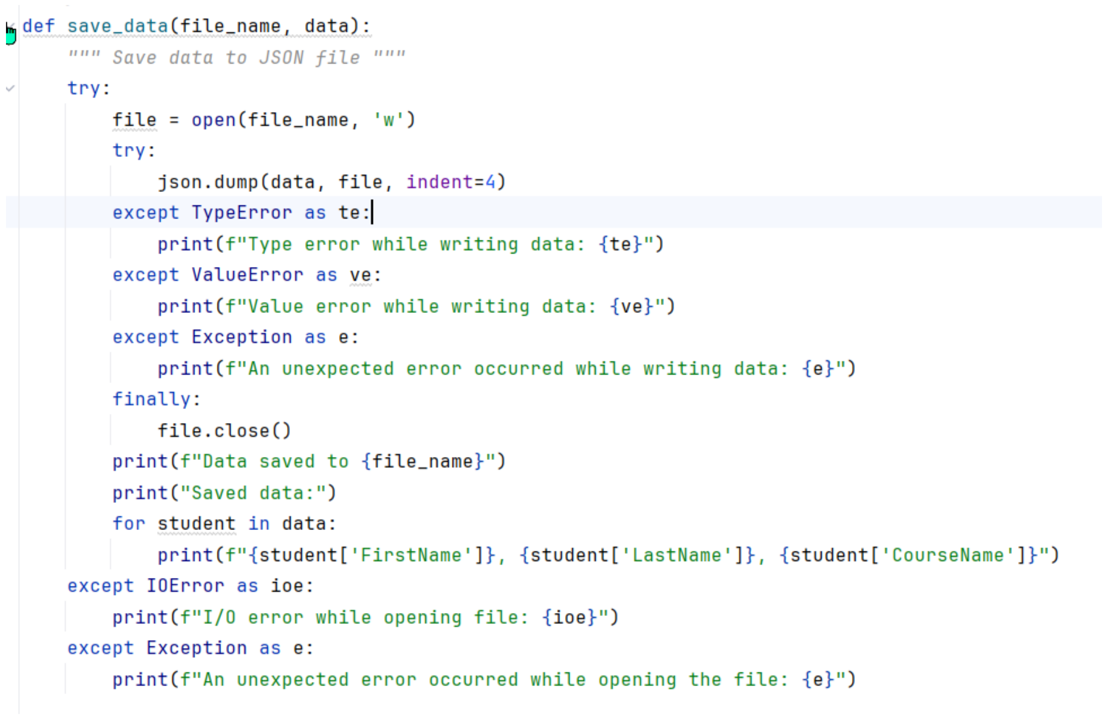
ValueError: Catches errors related to invalid values during the conversion to JSON.

General Exception: Catches any other unexpected errors during writing process.

Finally Block: Ensures the file is closed after writing, regardless of whether an error occurred.

IOError: Catches errors related to input/output operations when opening the file.

General Exception: Catches any other unexpected errors during the opening process.



```python
def save_data(file_name, data):
    """ Save data to JSON file """
    try:
        file = open(file_name, 'w')
        try:
            json.dump(data, file, indent=4)
        except TypeError as te:
            print(f"Type error while writing data: {te}")
        except ValueError as ve:
            print(f"Value error while writing data: {ve}")
        except Exception as e:
            print(f"An unexpected error occurred while writing data: {e}")
        finally:
            file.close()
        print(f"Data saved to {file_name}")
        print("Saved data:")
        for student in data:
            print(f"{student['FirstName']}, {student['LastName']}, {student['CourseName']}")
    except IOError as ioe:
        print(f"I/O error while opening file: {ioe}")
    except Exception as e:
        print(f"An unexpected error occurred while opening the file: {e}")
```

Figure 2. Screen shot of code to write data into the JSON file format type and handle any writing errors.

Topic – Structured Error Handling

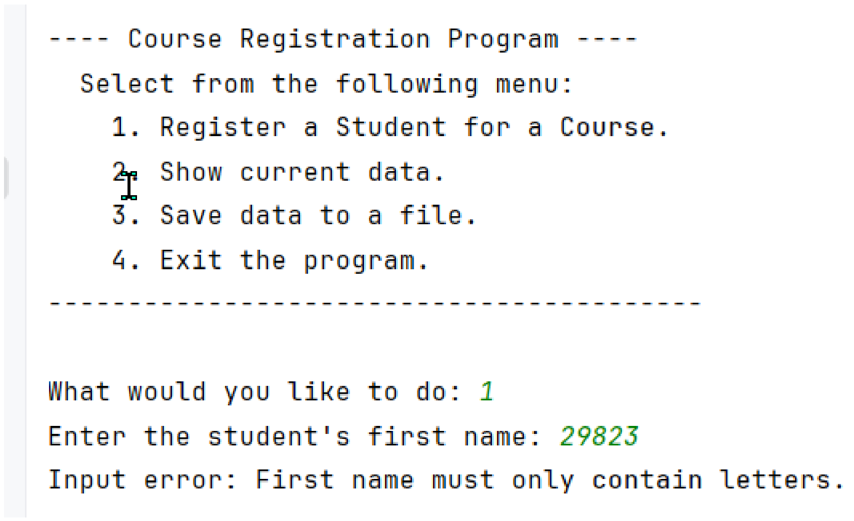
In the script, structured error handling is implemented using try, except, and finally blocks as in Figure 2 above. This ensures that specific errors are caught and handled appropriately, and the file is managed for a better user experience.

Error Handling in load_data function uses multiple except blocks to handle specific errors like FileNotFoundError: Handles the case where the JSON file does not exist, JSONDecodeError: Handles errors related to invalid JSON content and General Exception: Catches any other unexpected errors.

Error Handling in save_data function function uses nested try blocks to handle errors during both the file opening and writing processes like TypeError and ValueError which handle errors during the conversion of data to JSON format. The General Exception: Catches any other unexpected errors during the writing process. Finally Block: Ensures the file is closed after writing. IOError: Handles errors during the file opening process.

General Exception: Catches any other unexpected errors during the file opening process.

I used the raise custom error function to prevent the user from entering numbers for their first and last name (Figure 3).

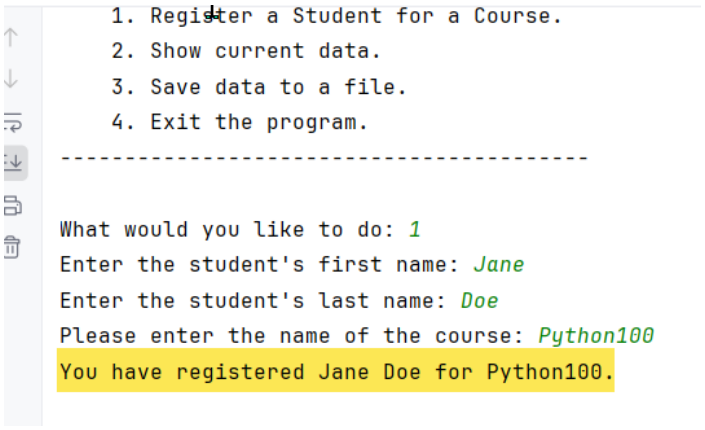


```
---- Course Registration Program ----
  Select from the following menu:
    1. Register a Student for a Course.
    2. Show current data.
    3. Save data to a file.
    4. Exit the program.
-----------------------------------------

What would you like to do: 1
Enter the student's first name: 29823
Input error: First name must only contain letters.
```

Figure 3. User can't input numbers into the name field because it causes an error.

Topic – Testing

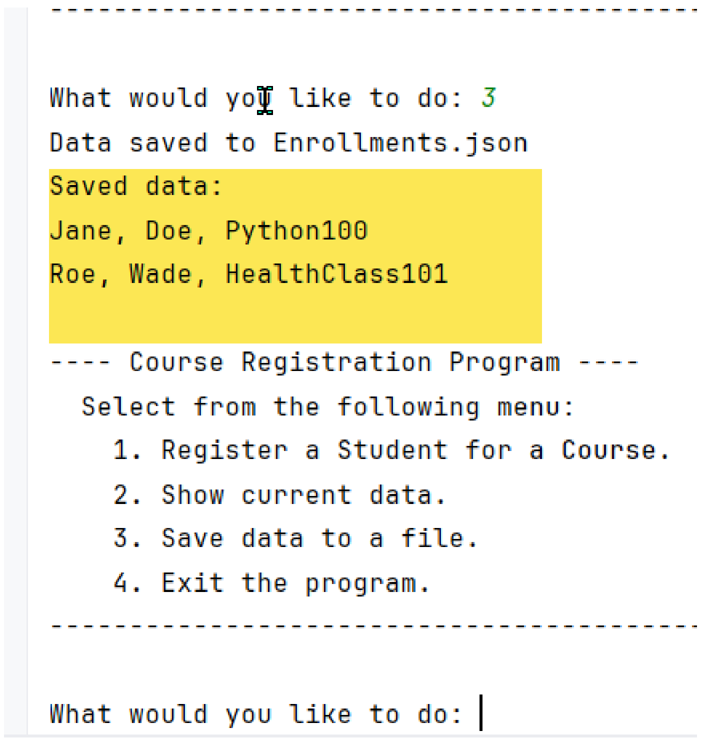
After inputting student registration data, the program then lists the first and last name and course name. (See Figure 4)



```
    1. Register a Student for a Course.
    2. Show current data.
    3. Save data to a file.
    4. Exit the program.
-----------------------------------------

What would you like to do: 1
Enter the student's first name: Jane
Enter the student's last name: Doe
Please enter the name of the course: Python100
You have registered Jane Doe for Python100.
```

Figure 4. Course details displayed after entering data for Menu choice 1.

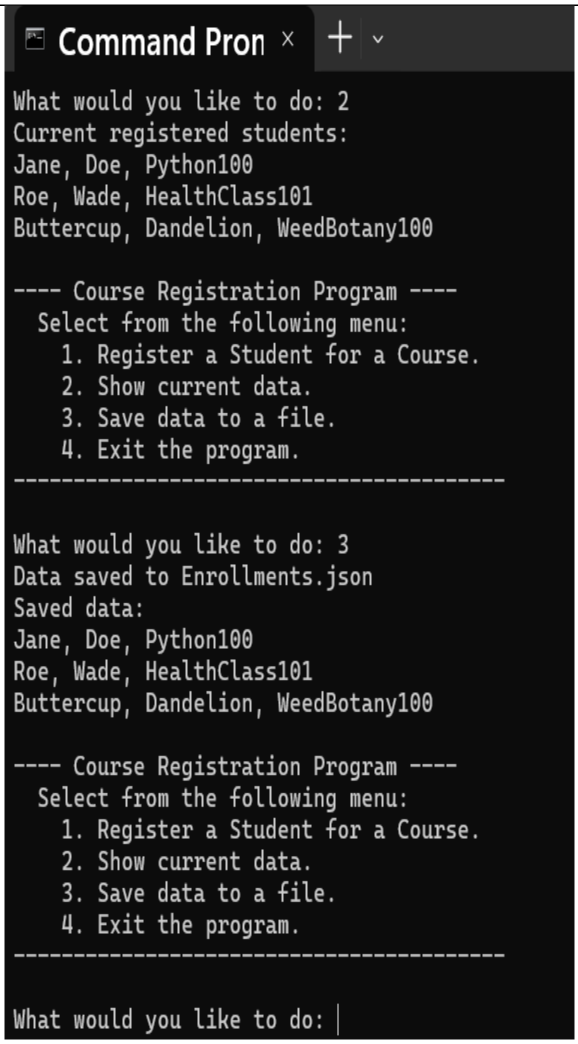
For Menu choice 3, the data gets saved to a file named Enrollments.json and shows the saved data (Figure 5).



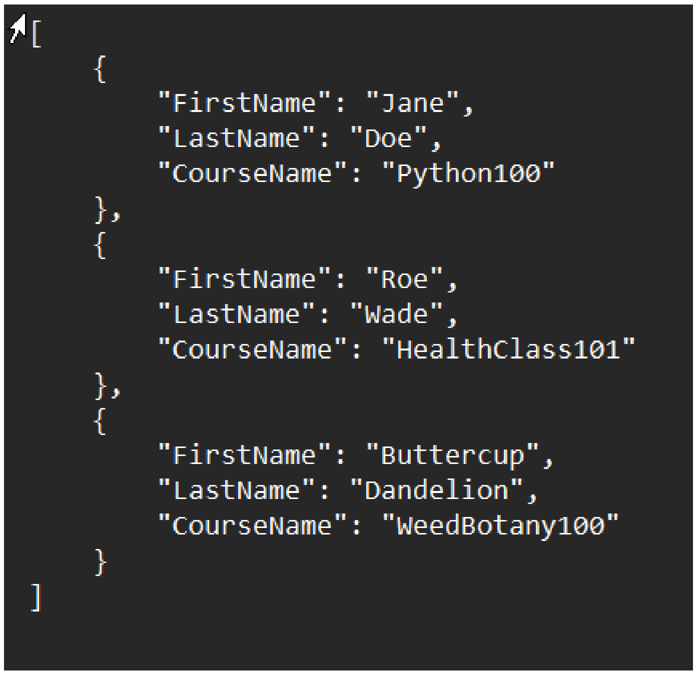
```
-----------------------------------------

What would you like to do: 3
Data saved to Enrollments.json
Saved data:
Jane, Doe, Python100
Roe, Wade, HealthClass101

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

What would you like to do: 
```

Figure 5. Course details displayed after saving data for Menu choice 3.

Per the assignment, I also tested in the MS Command Prompt and the program ran successfully in that environments (Figure 6). I verified that the program can register students, display current data, save data to a file, and handle multiple registrations and that it can display the dictionary data in Notepad with the key and individual string values.



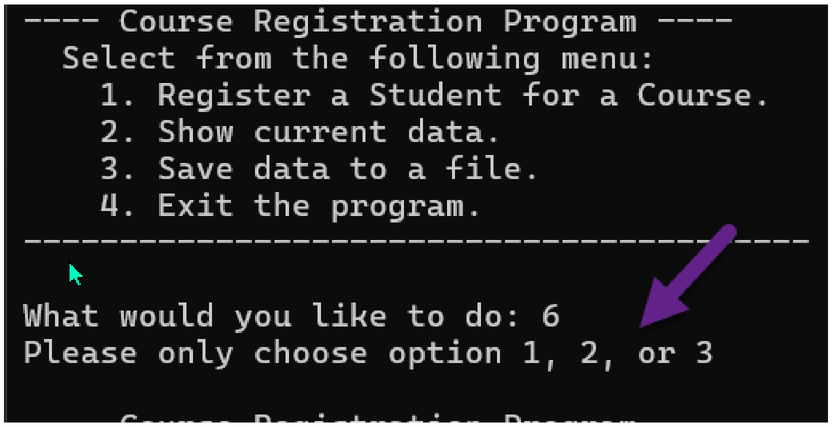
```
What would you like to do: 2
Current registered students:
Jane, Doe, Python100
Roe, Wade, HealthClass101
Buttercup, Dandelion, WeedBotany100

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

What would you like to do: 3
Data saved to Enrollments.json
Saved data:
Jane, Doe, Python100
Roe, Wade, HealthClass101
Buttercup, Dandelion, WeedBotany100

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

What would you like to do:
```



```
[
    {
        "FirstName": "Jane",
        "LastName": "Doe",
        "CourseName": "Python100"
    },
    {
        "FirstName": "Roe",
        "LastName": "Wade",
        "CourseName": "HealthClass101"
    },
    {
        "FirstName": "Buttercup",
        "LastName": "Dandelion",
        "CourseName": "WeedBotany100"
    }
]
```

Figure 6. Command prompt and Enrollments.json in notepad screen shot

The program successfully ensures the user must enter a 1, 2 or 3 (Figure 7).



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

What would you like to do: 6
Please only choose option 1, 2, or 3
```

Figure 7. User must enter 1, 2, or 3 or else they get an error message.

Summary

The script successfully demonstrates the use of opening, closing functions and adding comma separated data to lists in a JSON dictionary and key file structure. It allows users to register students for courses, display the current registrations, save the data to a JSON file, and exit the program while giving error messages if there are problems like entering numbers for names or course menu items beyond the actual choices. The program was tested in PyCharm and Command Prompt and executed without errors, providing a JSON solution using dictionaries and keys for managing course registrations along with some guard rails around user inputs and file handling.