Hao Yi Fei

11/9/2025

Introduction to Programming with Python

Assignment 05

Portter-fly/IntroToProg-Python-Mod05

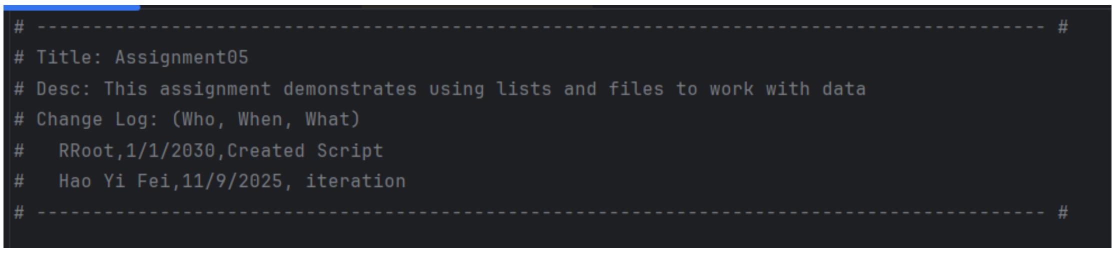
The Student Registration Program with Dictionaries, JSON, and Error Handling

Intro

This week, I built on Assignment04's student registration program and added some new, more advanced features. Assignment05 requires dictionaries for data storage, JSON files for data persistence, and structured error handling with try-except blocks. This assignment was a bit tricky at first, especially understanding how dictionaries work differently from lists, but by following the module resources and testing step by step, I got it working. Below is a detailed breakdown of how I wrote the program.

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)



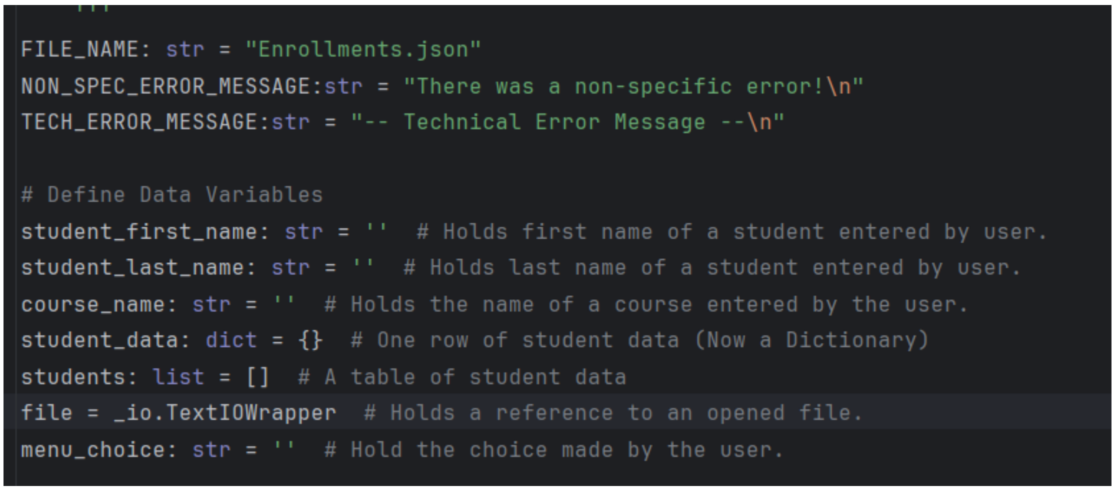
```
# ------------------------------------------------------------------------------------------ #
# Title: Assignment05
# Desc: This assignment demonstrates using lists and files to work with data
# Change Log: (Who, When, What)
#   RRoot,1/1/2030,Created Script
#   Hao Yi Fei,11/9/2025, iteration
# ------------------------------------------------------------------------------------------ #
```

Figure 1.1: Script header which has been updated.

Defining Constants

Next, I defined the required constants. The MENU constant was almost the same as

Assignment04, but I double-checked to make sure the formatting matched the assignment's exact requirement. I changed FILE_NAME to "Enrollments. json" (from CSV) and added two new constants for error messages, which are: NON_SPEC_ERROR_MESSAGE and TECH_ERROR_MESSAGE. (Figure 1.2)

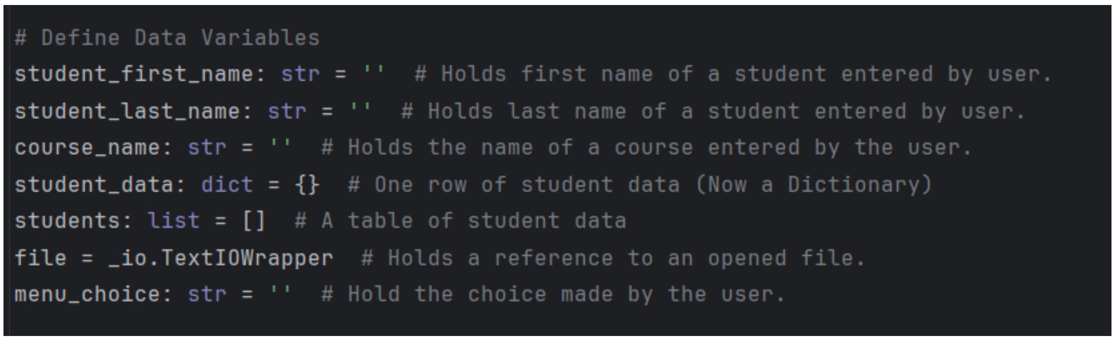


```
FILE_NAME: str = "Enrollments.json"
NON_SPEC_ERROR_MESSAGE:str = "There was a non-specific error!\n"
TECH_ERROR_MESSAGE:str = "-- Technical Error Message --\n"

# Define Data Variables
student_first_name: str = ''  # Holds first name of a student entered by user.
student_last_name: str = ''  # Holds last name of a student entered by user.
course_name: str = ''  # Holds the name of a course entered by the user.
student_data: dict = {}  # One row of student data (Now a Dictionary)
students: list = []  # A table of student data
file = _io.TextIOWrapper  # Holds a reference to an opened file.
menu_choice: str = ''  # Hold the choice made by the user.
```

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

Initializing Variables

This part was different from Assignment04 because I had to use a dictionary for student_data instead of a list. I initialized all variables as required: empty strings for names and course, student_data as an empty dictionary, students as an empty list, file set to _io.TextIOWrapper, and menu_choice as an empty string. (Figure 1.3)

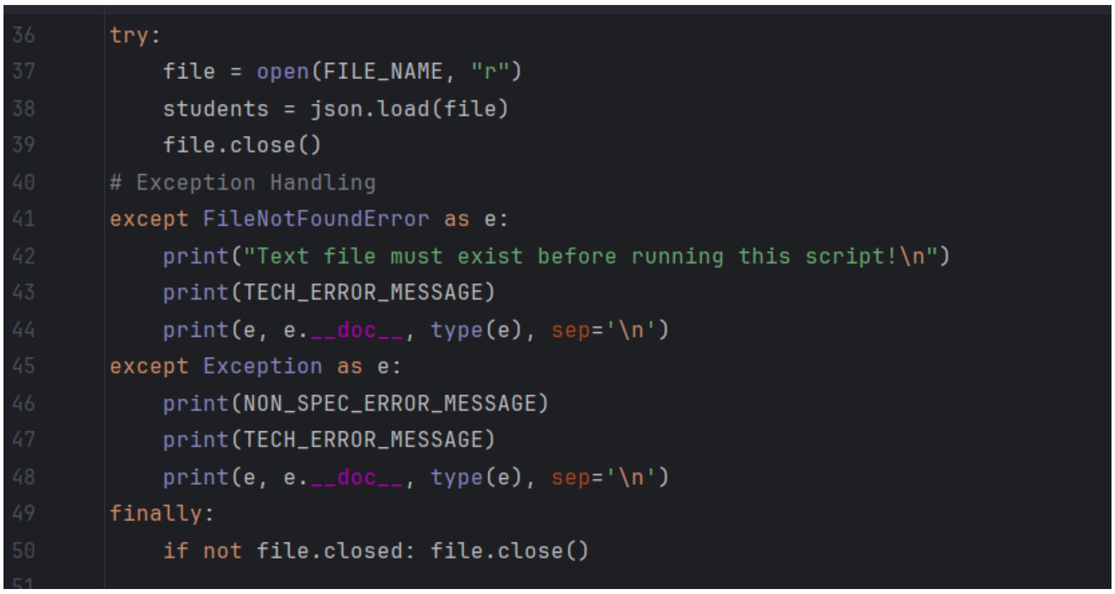


```
# Define Data Variables
student_first_name: str = ''  # Holds first name of a student entered by user.
student_last_name: str = ''  # Holds last name of a student entered by user.
course_name: str = ''  # Holds the name of a course entered by the user.
student_data: dict = {}  # One row of student data (Now a Dictionary)
students: list = []  # A table of student data
file = _io.TextIOWrapper  # Holds a reference to an opened file.
menu_choice: str = ''  # Hold the choice made by the user.
```

Figure 1.3: Variable initialization with dictionary and list setup

Loading Existing Data on Startup

A key new requirement was loading data from the JSON file when the program starts. In Assignment04, I used split(',') for CSV, but here I used json.load(file) to read the JSON data directly into the students list.   (Figure 1.4)

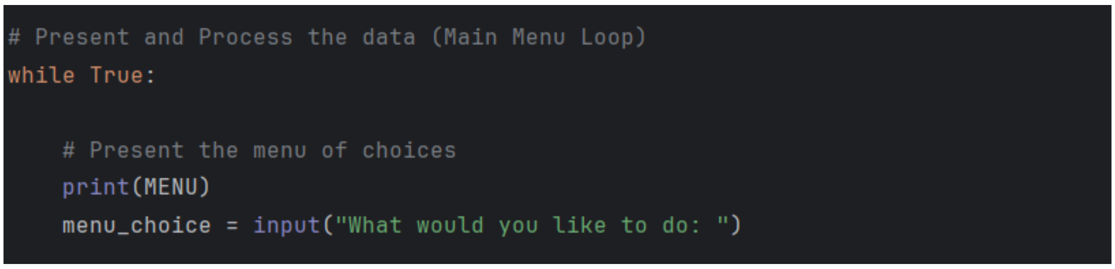


```python
try:
    file = open(FILE_NAME, "r")
    students = json.load(file)
    file.close()
# Exception Handling
except FileNotFoundError as e:
    print("Text file must exist before running this script!\n")
    print(TECH_ERROR_MESSAGE)
    print(e, e.__doc__, type(e), sep='\n')
except Exception as e:
    print(NON_SPEC_ERROR_MESSAGE)
    print(TECH_ERROR_MESSAGE)
    print(e, e.__doc__, type(e), sep='\n')
finally:
    if not file.closed: file.close()
```

Figure 1.4: Code to load JSON data into a list of dictionaries on startup

Implementing the Menu and While Loop

Just like the previous one, I remained the same function. (Figure 1.5)

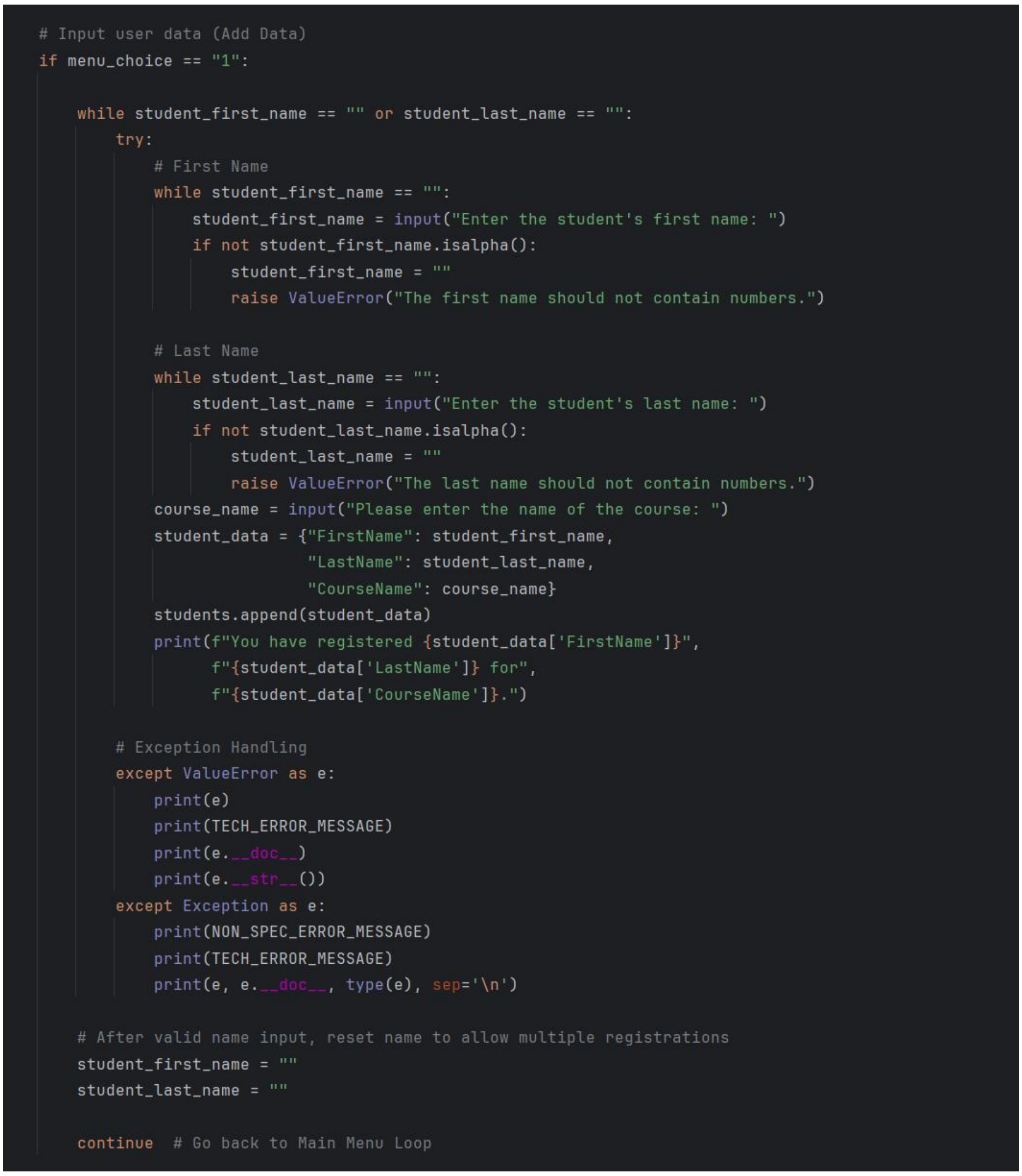


```python
# Present and Process the data (Main Menu Loop)
while True:

    # Present the menu of choices
    print(MENU)
    menu_choice = input("What would you like to do: ")
```

Figure 1.5: Use while loop for menu

Handling Menu Choice 1: Student Registration

This was the most complex part because I had to use dictionaries and add input validation with error handling. I added a while loop to make sure the user enters a first and last name (no empty strings). I also added a check using isalpha() to ensure names only contain letters. (Figure 1.6)

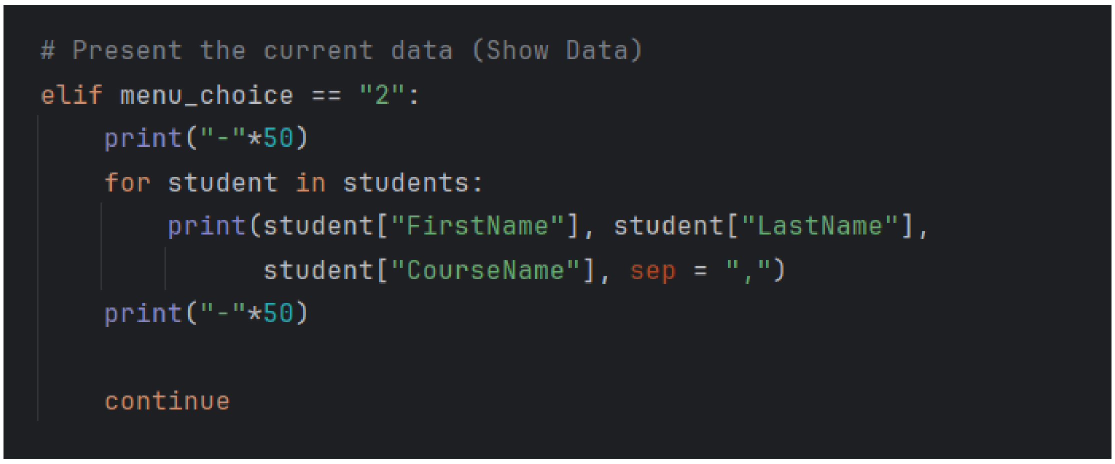


```python
# Input user data (Add Data)
if menu_choice == "1":

    while student_first_name == "" or student_last_name == "":
        try:
            # First Name
            while student_first_name == "":
                student_first_name = input("Enter the student's first name: ")
                if not student_first_name.isalpha():
                    student_first_name = ""
                    raise ValueError("The first name should not contain numbers.")

            # Last Name
            while student_last_name == "":
                student_last_name = input("Enter the student's last name: ")
                if not student_last_name.isalpha():
                    student_last_name = ""
                    raise ValueError("The last name should not contain numbers.")
            course_name = input("Please enter the name of the course: ")
            student_data = {"FirstName": student_first_name,
                            "LastName": student_last_name,
                            "CourseName": course_name}
            students.append(student_data)
            print(f"You have registered {student_data['FirstName']}",
                  f"{student_data['LastName']} for",
                  f"{student_data['CourseName']}.")

        # Exception Handling
        except ValueError as e:
            print(e)
            print(TECH_ERROR_MESSAGE)
            print(e.__doc__)
            print(e.__str__())
        except Exception as e:
            print(NON_SPEC_ERROR_MESSAGE)
            print(TECH_ERROR_MESSAGE)
            print(e, e.__doc__, type(e), sep='\n')

    # After valid name input, reset name to allow multiple registrations
    student_first_name = ""
    student_last_name = ""

    continue  # Go back to Main Menu Loop
```

Figure 1.6: Logic for registering students with input validation and dictionary storage (Menu Choice 1)

Handling Menu Choice 2: Show Current Data

Displaying data was simpler than the previous model. I looped through the students list and printed each dictionary's values with commas separating them (using sep=","). (Figure 1.7)

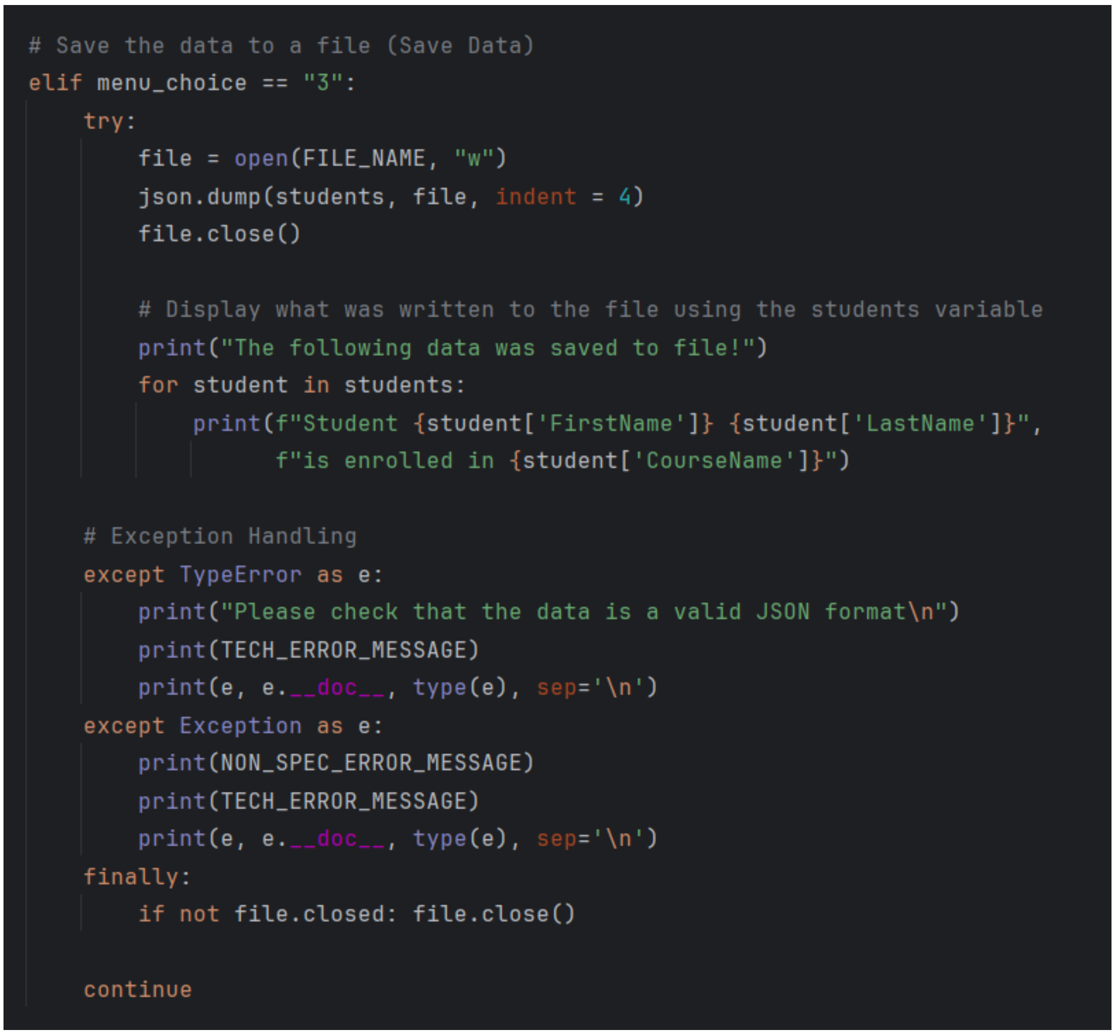


```python
# Present the current data (Show Data)
elif menu_choice == "2":
    print("-"*50)
    for student in students:
        print(student["FirstName"], student["LastName"],
              student["CourseName"], sep = ",")
    print("-"*50)

    continue
```

Figure 1.7: Logic to display all student data from the dictionaries (Menu Choice 2)

Handling Menu Choice 3: Save Data to a JSON File

I used json.dump to write the list of dictionaries to the file. The indent=4 made the JSON file easier to read. I also added error handling like what I have done before. (Figure 1.8)

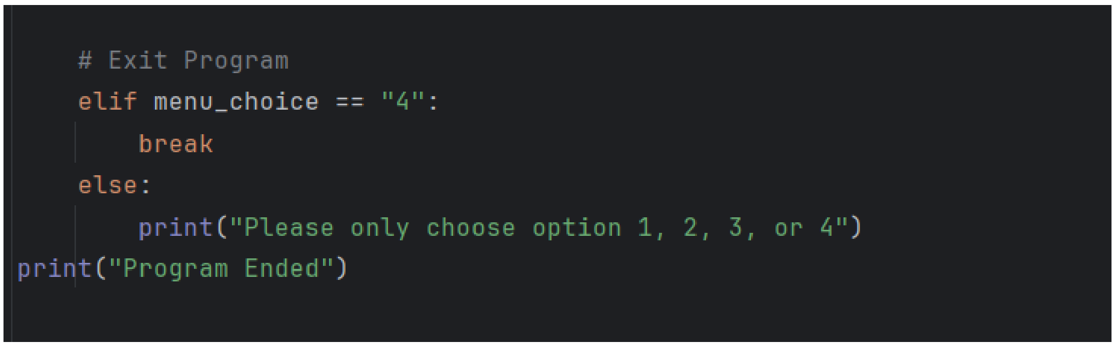


```python
# Save the data to a file (Save Data)
elif menu_choice == "3":
    try:
        file = open(FILE_NAME, "w")
        json.dump(students, file, indent = 4)
        file.close()

        # Display what was written to the file using the students variable
        print("The following data was saved to file!")
        for student in students:
            print(f"Student {student['FirstName']} {student['LastName']}",
                  f"is enrolled in {student['CourseName']}")

    # Exception Handling
    except TypeError as e:
        print("Please check that the data is a valid JSON format\n")
        print(TECH_ERROR_MESSAGE)
        print(e, e.__doc__, type(e), sep='\n')
    except Exception as e:
        print(NON_SPEC_ERROR_MESSAGE)
        print(TECH_ERROR_MESSAGE)
        print(e, e.__doc__, type(e), sep='\n')
    finally:
        if not file.closed: file.close()

    continue
```

Figure 1.8: Logic to save data to JSON file with error handling (Menu Choice 3)

Handling Menu Choice 4: Exit the Program

For menu choice 4, the loop is ended by using a break statement. (Figure 1.9)



```
    # Exit Program
    elif menu_choice == "4":
        break
    else:
        print("Please only choose option 1, 2, 3, or 4")
print("Program Ended")
```

Figure 1.9: Logic for exiting the program (Menu Choice 4)

Testing the Program

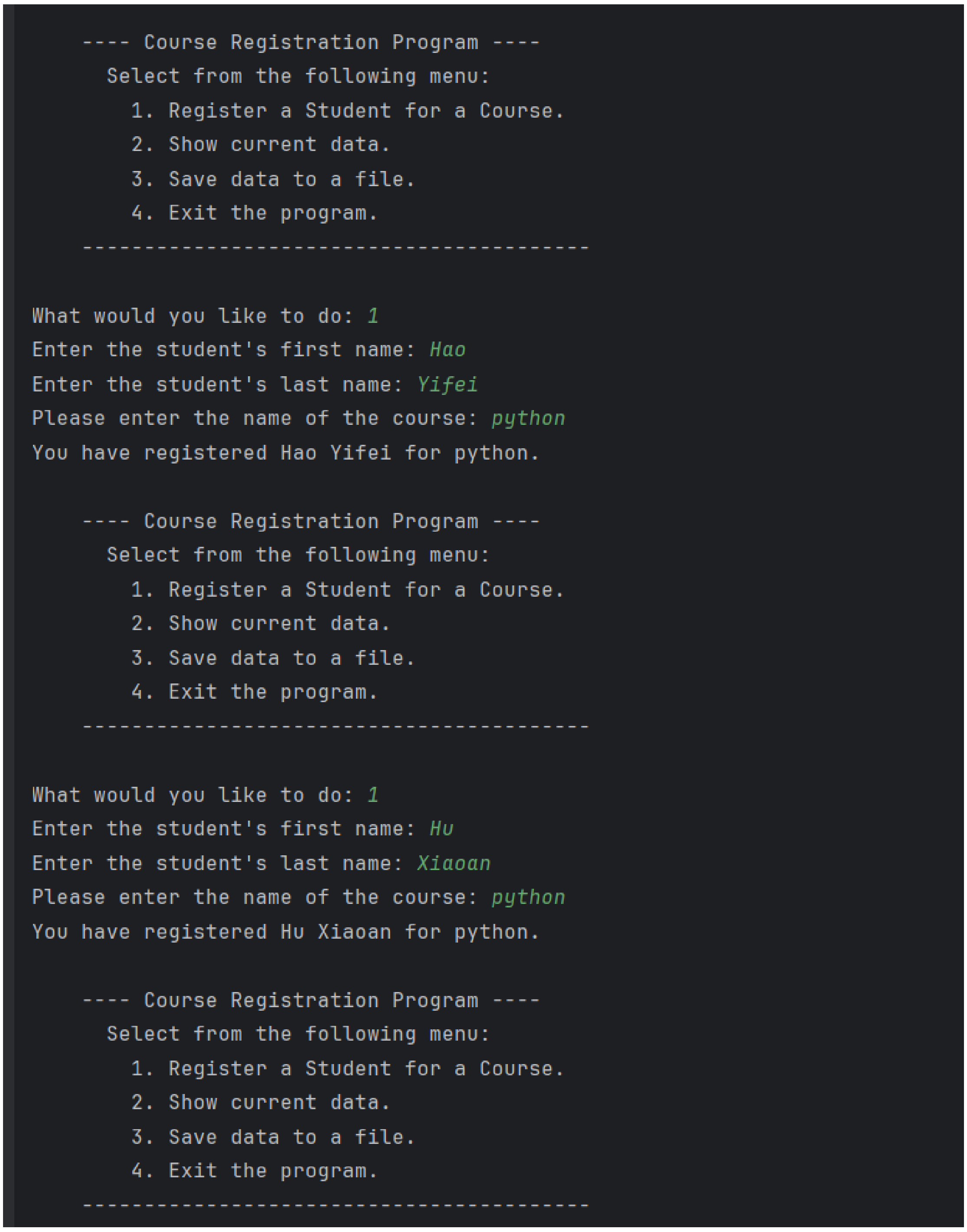
After writing the code, I tested it thoroughly to make sure all features worked. Here's what I did:

1 1. PyCharm Run Test

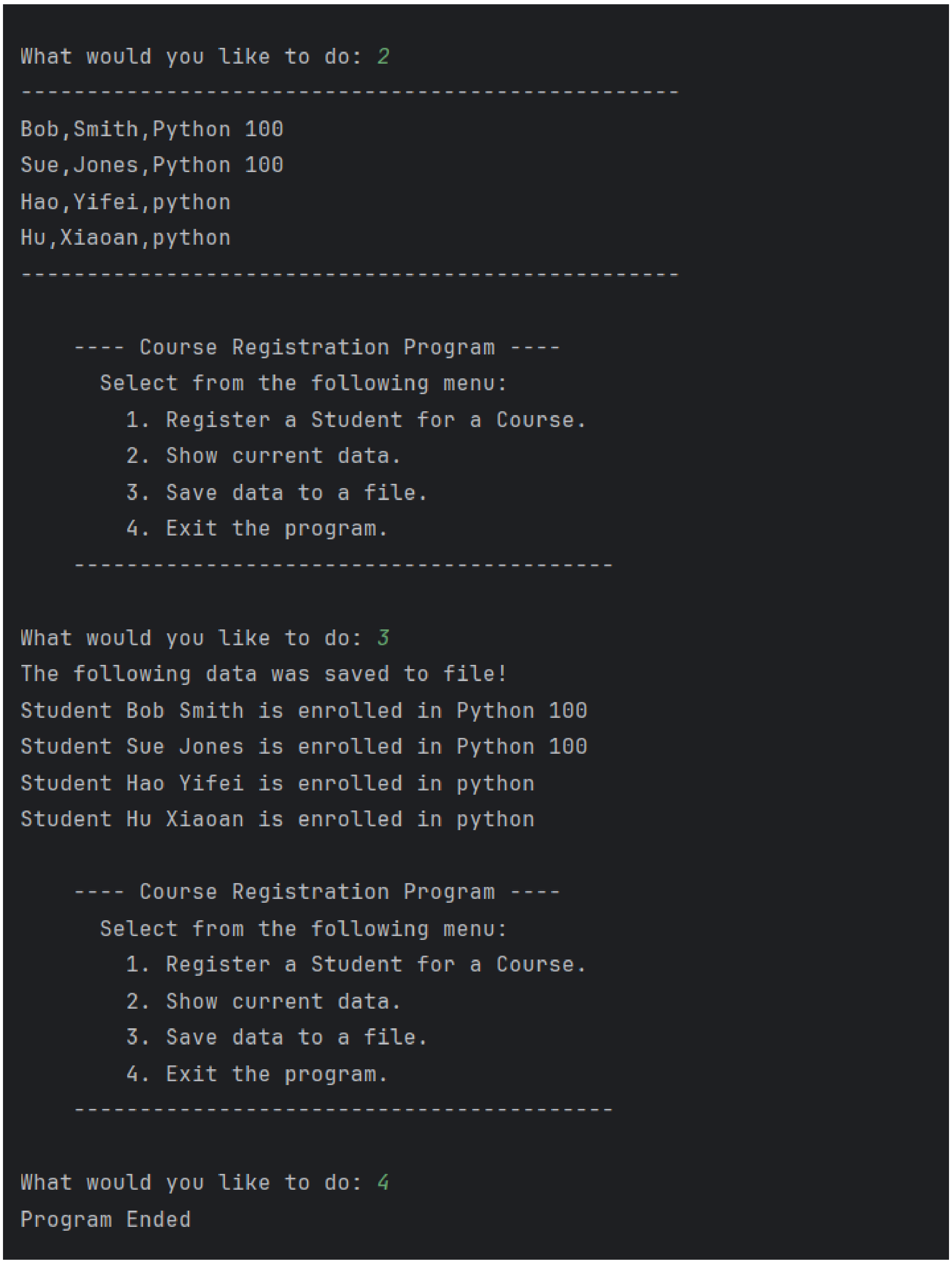
I ran the program in PyCharm and tested all menu options:

- Registered multiple students
- Tried entering names with numbers
- Showed the current data to confirm all registrations appeared
- Saved the data to the JSON file
- Exited and restarted the program to check if data loaded correctly

(Figure 2.1)



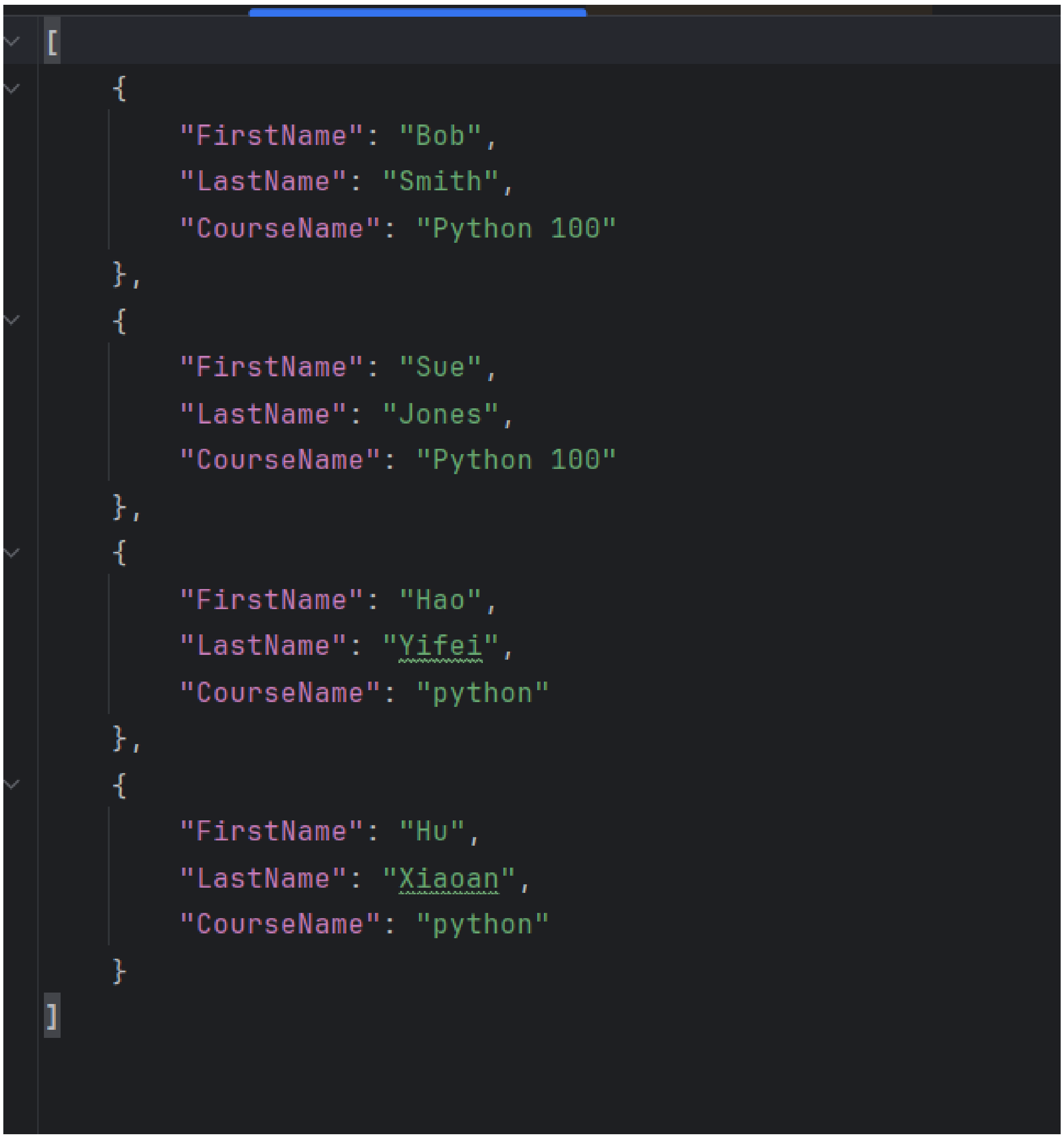
```
    ---- Course Registration Program ----
      Select from the following menu:
        1. Register a Student for a Course.
        2. Show current data.
        3. Save data to a file.
        4. Exit the program.
    -----------------------------------------

What would you like to do: 1
Enter the student's first name: Hao
Enter the student's last name: Yifei
Please enter the name of the course: python
You have registered Hao Yifei for 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.
    -----------------------------------------

What would you like to do: 1
Enter the student's first name: Hu
Enter the student's last name: Xiaoan
Please enter the name of the course: python
You have registered Hu Xiaoan for 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.
    -----------------------------------------
```



```
What would you like to do: 2
--------------------------------------------------
Bob,Smith,Python 100
Sue,Jones,Python 100
Hao,Yifei,python
Hu,Xiaoan,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.
    -----------------------------------------

What would you like to do: 3
The following data was saved to file!
Student Bob Smith is enrolled in Python 100
Student Sue Jones is enrolled in Python 100
Student Hao Yifei is enrolled in python
Student Hu Xiaoan is enrolled in 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.
    -----------------------------------------

What would you like to do: 4
Program Ended
```

Figure 2.1: PyCharm run with all functions

I then checked the JSON file. (Figure 2.2)



```
[
    {
        "FirstName": "Bob",
        "LastName": "Smith",
        "CourseName": "Python 100"
    },
    {
        "FirstName": "Sue",
        "LastName": "Jones",
        "CourseName": "Python 100"
    },
    {
        "FirstName": "Hao",
        "LastName": "Yifei",
        "CourseName": "python"
    },
    {
        "FirstName": "Hu",
        "LastName": "Xiaoan",
        "CourseName": "python"
    }
]
```

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

Summary

I now understand the difference between lists and dictionaries: lists use indexes and dictionaries use keys, which makes data easier to read and manage. I also learned how to work with JSON files. Also error handling with try-except blocks was a big new concept, but it made my program more robust. Overall, I think this exercise built a strong foundation for creating and managing Python scripts in the future.