Name: Mohiuddin Khan
Date: November 12, 2025
Course: Python 100 (Foundations of Programming: Python)

Assignment 05 – Working with dictionaries, files (JSON), and exception handling to manage course registrations

Introduction

For Assignment 05, I created a Python console program called Assignment05.py that manages course registrations using a menu, a list of dictionaries, a JSON file, and structured error handling. The goal was to extend the work from the earlier assignment (which used a list of lists) and apply the new concepts from Module 05: dictionaries, JSON files, and try–except blocks. I believe the finished program lets the user:

- Register a student for a course
- Display all current registrations
- Save the registrations to a JSON file (Enrollments.json)
- Load existing registrations from the JSON file at startup
- Handle common errors (bad name input, file problems, JSON issues) in a friendly way

The script follows the provided starter structure and acceptance criteria and includes a professional header, constants, typed variables, and clearly labeled sections.

Steps I Took to Complete the Assignment

1. Reviewing the Assignment Instructions and Notes

I started reading the Module 05 Notes and the Assignment 05 instructions. These explained:

- How lists and dictionaries can be combined to create a “table” of data (a list of dictionary rows).
- How to read and write structured data using JSON and the json module.
- Why structured error handling with try–except is recommended, especially for file operations and user input.

I also skimmed the textbook (Python Programming for the Absolute Beginner- 3rd Ed-MDawson) sections on variables, comments, and simple I/O to reinforce good coding style and clear naming.

2. Setting Up the Script Header, Constants, and Variables

Next, I opened the starter file and created my own script, **Assignment05.py**, using the same structure. I updated the header with my name, the date, and a short description:

```
# ------------------------------------------------------------------------------------------

# Title: Assignment05

# Desc: This assignment demonstrates using dictionaries, files (JSON), and exception

#           handling to manage course registrations.

# Change Log: (Who, When, What)

#   Mohiuddin Khan,11/12/2025, Completed Assignment05 with dictionaries, JSON, and
#    exceptions

# ------------------------------------------------------------------------------------------
```

Then I defined the **constants** exactly as required in the assignment:

```
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"
```

I also created the typed variables that the assignment specified. These hold user input, the current student record, the list of registrations, the file handle, and the menu choice:

```
student_first_name: str = ''

student_last_name: str = ''

course_name: str = ''
```

```
student_data: dict = {}

students: list = []

file = _io.TextIOWrapper

menu_choice: str = ''
```

Using clear names and type hints helped keep the logic organized and match the examples from the module labs.

3. Loading Existing Data from Enrollments.json with Error Handling

The next step was to implement startup processing so that the program automatically loads any existing registrations from Enrollments.json. I wrapped this logic in a try–except–finally block to prevent the program from crashing if the file is missing or the JSON is invalid, following the style from Mod05-Lab02 and Mod05-Lab03.

```
try:

    file = open(FILE_NAME, "r")

    students = json.load(file)          # Expecting a list of dictionaries.

    file.close()

except FileNotFoundError as e:

    print("The enrollment file must exist before running this script!\n")

    print("-- Technical Error Message --")

    print(e, e.__doc__, type(e), sep="\n")

except json.JSONDecodeError as e:

    print("The enrollment file contains invalid JSON data.\n")

    print("-- Technical Error Message --")

    print(e, e.__doc__, type(e), sep="\n")

except Exception as e:

    print("There was a non-specific error while reading the file!\n")

    print("-- Technical Error Message --")

    print(e, e.__doc__, type(e), sep="\n")

finally:
    try:
        if file.closed is False:
            file.close()
    except Exception:
        pass
```

This code mirrors the structured error handling pattern shown in the notes and labs: specific exceptions first (FileNotFoundError, JSONDecodeError), followed by a general Exception.

4. Building the Main Menu Loop

After the startup logic, I created the main program loop using while True:. In each iteration, the program:

1. Prints the MENU constant
2. Asks the user for a menu option
3. Uses if/elif/else to call the correct block of code

```
while True:
    print(MENU)
    menu_choice = input("What would you like to do: ").strip()
    print()

    if menu_choice == "1":
        # register a student
    elif menu_choice == "2":
        # show current data
    elif menu_choice == "3":
        # save data to file
    elif menu_choice == "4":
        print("Program Ended.Thank you for your input. Have a Great Day!")
        break
    else:
        print("Please only choose option 1, 2, 3, or 4.\n")
```

This structure follows the menu logic pattern described in the assignment and earlier modules: a loop, a menu printout, and a branch for each menu option.

5. Implementing Menu Choice 1 – Registering a Student

For Menu Choice 1, I needed to:

- Prompt the user for first name, last name, and course name
- Validate the name inputs using structured error handling
- Store the data in a dictionary
- Append that dictionary to the students list

I modeled this after the list/dictionary examples in the Module 05 notes.

```
if menu_choice == "1":
    try:
        student_first_name = input("Enter the student's first name: ").strip()
        if not student_first_name.isalpha():
            raise ValueError("First name must contain only letters (no numbers or symbols).")

        student_last_name = input("Enter the student's last name: ").strip()
        if not student_last_name.isalpha():
            raise ValueError("Last name must contain only letters (no numbers or symbols).")

        course_name = input("Please enter the name of the course: ").strip()

        student_data = {
            "FirstName": student_first_name,
            "LastName": student_last_name,

            "CourseName": course_name

        }


        students.append(student_data.copy())


        print(f"You have registered {student_first_name} {student_last_name} for
{course_name}.\n")


    except ValueError as e:

        print(e)

        print("-- Technical Error Message --")

        print(e.__doc__)

        print(e.__str__())

    except Exception as e:

        print("There was a non-specific error while entering student data!\n")

        print("-- Technical Error Message --")

        print(e, e.__doc__, type(e), sep="\n")


    continue
```

Key design choices:

- isalpha() ensures first and last names do not contain digits or symbols.
- Custom ValueError messages give clear feedback if the input is invalid.
- A new dictionary is created for each student and appended to the students list as a row, which matches the idea of a “table” built from a list of dictionaries.

6. Implementing Menu Choice 2 – Showing Current Data

For Menu Choice 2, I needed to display all the registrations in two ways:

1. A friendly sentence for each student
2. A comma-separated (CSV-style) line for each row

```
elif menu_choice == "2":
    if not students:
        print("No registrations to display yet.\n")
    else:
        print("-" * 50)
        for student_data in students:
            first = student_data.get("FirstName", "")
            last = student_data.get("LastName", "")
            course = student_data.get("CourseName", "")

            print(f"Student {first} {last} is enrolled in {course}")
            print(f"{first},{last},{course}")
        print("-" * 50)
        print()
    continue
```

Using .get() on the dictionary keys makes the code more robust, in case any key is missing. Looping through the list-of-dictionaries demonstrates using dictionaries as "rows" in a table, which was a major concept for this module.

7. Implementing Menu Choice 3 – Saving Data to JSON

For **Menu Choice 3**, I wrote the students list to Enrollments.json using json.dump, again with structured error handling:

```
elif menu_choice == "3":
```

```
try:
        file = open(FILE_NAME, "w")
        json.dump(students, file, indent=2)
        file.close()

        print("The following data was saved to file!\n")
        if not students:
            print("(There were no registrations to save.)\n")
        else:
            for student_data in students:
                first = student_data.get("FirstName", "")
                last = student_data.get("LastName", "")
                course = student_data.get("CourseName", "")
                print(f"{first},{last},{course}")
            print()
    except TypeError as e:
        print("Please check that the data can be converted into valid JSON.\n")
        print("-- Technical Error Message --")
        print(e, e.__doc__, type(e), sep="\n")
    except Exception as e:
        print("There was an error while trying to save the data!\n")
        print("-- Technical Error Message --")
        print(e, e.__doc__, type(e), sep="\n")
    finally:
        try:
            if file.closed is False:
                file.close()
        except Exception:
            pass


    continue
```

This follows the JSON example pattern from the notes and labs: open → dump → close → error handling.

8. Tested Major Assignment Requirements

To make sure I met all the requirements listed in Mod05-Assignment, I performed the following tests:

a. **Input and storage of first name, last name, and course name**

- I chose menu option 1 and entered several different student names and course names.
- After each entry, I confirmed the success message (e.g., "You have registered …").

b. **Multiple registrations using a list of dictionaries**

- I added several students in a row.
- I selected menu option 2 and checked that each registration appeared once and correctly.

c. **Display of CSV-style output**

- In option 2, I verified that every student also printed as FirstName,LastName,CourseName on its own line.

d. **Saving data to the JSON file**

- I selected option 3 to save.
- I opened Enrollments.json in a text editor and confirmed that the file contained a valid JSON list of student dictionaries with the expected data.

e. **Reloading data on startup**

- I closed the program.
- I re-ran it, went straight to option 2, and checked that the data loaded from the file appeared correctly.

f. **Error handling for names and files**

- I deliberately typed invalid first and last names containing numbers to trigger the ValueError and confirm the error messages.

- I tested with an empty or incorrect JSON file to confirm the startup error messages for file and JSON issues.

9. Adding Comments and Code Clarity

Throughout the script, I focused on readability and clarity, which is emphasized both in the module content and in the textbook.

Specific things I did:

- Used a clear header block at the top with title, description, and change log.
- Added section comments (e.g., Startup Processing, Main Program Loop, and menu headers) so someone reading the code can quickly see the structure.
- Chose descriptive variable names like students, student_data, course_name, and menu_choice instead of single letters.
- Used consistent casing for dictionary keys ("FirstName", "LastName", "CourseName") to match the guidance in the notes about key names and JSON compatibility.
- Printed user-friendly messages for both normal operations and errors, keeping technical info in a clearly labeled "Technical Error Message" section.

This matches the general best practices: comment "why", use readable names, and structure the code with logical sections.

10. Documentation and Submission

After finishing and testing the program, I completed the documentation and submission steps described in Mod05-Assignment:

a. **Knowledge Document (this file)**

- Wrote this document following the professional template and section headings.
- Described what I did, showed key code sections, and explained how the program meets the requirements.

b. **GitHub Repository**

- Created a public repository named **IntroToProg-Python-Mod05**.
- Uploaded:
    - Assignment05.py
    - Enrollments.json with sample data

    - This knowledge document (as a PDF).
    - Several knowledge based discussions in a separate file in PDF.
- Committed the changes so the instructor and classmates can review the work.

c. **Canvas Submission**

- Placed the Python file, the JSON file, and this document into a folder called A05.
- Zipped the folder and uploaded the .zip file to the Canvas Assignment 05 page.

d. **Discussion Board (for review)**

- Posted the GitHub link to the "Assignment 05 Documents for Review!" discussion board so peers can access it for informal review.

Summary

In this assignment, I moved from simple list-based programs to a more realistic data management script that combines:

- A menu-driven loop for user interaction
- A list of dictionaries to store student registration records
- A JSON file for persistent storage
- Structured error handling to gracefully handle invalid input and file issues

By following the Module 05 notes, labs, and assignment steps, I practiced working with dictionaries, JSON, and try–except blocks in a practical context. Overall, this assignment helped me see how separate pieces—variables, lists, dictionaries, files, loops, and exceptions—fit together into a small but complete application for course registration.

Citations

- Assignment05.py – Course Registration Program (M. Khan, 2025).
- Introduction to Programming with Python – Module 05: Advanced Collections and Error Handling, course notes.
- Module 05 – Assignment Instructions: Working with Lists, Files, Loops, and Menu Logic, course assignment handout.
- Dawson, M. (2010). *Python Programming for the Absolute Beginner* (3rd ed.). Course Technology PTR.