Fiorella Guerra
August 13, 2025
Foundations of Programming: Python
Assignment 5

Advanced Collections, JSON, and Error Handling

Introduction

Assignment 5 built upon the course registration program from prior assignments, but introduced advanced collections (dictionaries), JSON file handling, and structured error handling. This allowed for more meaningful data storage and improved program resilience by catching and responding to errors instead of crashing.

From the Module 5 Notes, I learned:

- **Dictionaries vs. Lists:** Lists store data by index, whereas dictionaries store data as key-value pairs.
- **Indexes vs. Keys:** Indexes are numeric positions; keys are descriptive identifiers for values.
- **JSON files:** A lightweight data-interchange format that stores data as key-value pairs, making it compatible with Python dictionaries.
- **The json module:** Provides functions like json.load() and json.dump() to easily read and write JSON data in Python.
- **Structured error handling:** Using try/except blocks to handle predictable errors gracefully, and finally to clean up resources like files.

Reading Data into Lists at Startup

When the program begins, it attempts to read an existing Enrollments.json file into the students variable, which is a **list of dictionaries**.

Steps taken:

- Opened the file in **read** mode.
- Used json.load() to convert the file contents into a Python list of dictionaries.
- Checked that the loaded data type was a list to ensure data integrity.
- Added except blocks to handle different potential problems:
    - FileNotFoundError: If the file didn’t exist, start with an empty list.
    - json.JSONDecodeError: If the file contained invalid JSON.

    - A general Exception to catch any other unexpected issues.
- Used finally to ensure the file closed properly.

```
try:
    file = open(FILE_NAME, "r")
    students = json.load(file)
    if not isinstance(students, list):
        raise ValueError("File does not contain a list of student records.")
except FileNotFoundError as e:
    print("Text file must exist before running this script! Starting with an empty 
list.\n")
    print(e, e.__doc__, type(e), sep='\n')
    students = []
except json.JSONDecodeError as e:
    print("Please check that the data is a valid JSON format\n")
    print(e, e.__doc__, type(e), sep='\n')
except Exception as e:
    print(e, e.__doc__, type(e), sep='\n')
finally:
    if file and not file.closed:
        file.close()
```

Registering a Student (Dictionaries with Error Handling)

For menu option 1:

- Prompted for **first name**, **last name**, and **course name**.
- Added **input validation** using .isalpha() for first and last names, raising ValueError if invalid.
- Created a dictionary for each registration
- Appended this dictionary to the students list.

```
student_data = {
    "FirstName": student_first_name,
    "LastName": student_last_name,
    "CourseName": course_name
}
```

This approach is more **descriptive and structured** than lists, as keys make the data self-explanatory.

Displaying Current Data

For menu option 2:

- Iterated through each dictionary in the students list.
- Used ",".join(student.values()) to produce a comma-separated display.

Example:

```
for student in students:
    print(",".join(student.values())
```

This kept the display format simple while ensuring the data remained tied to its keys internally.

Saving Data to JSON

For menu option 3:

- Opened the JSON file in **write** mode.
- Used json.dump() to serialize the list of dictionaries into JSON format.
- Added structured error handling:
    - PermissionError: If the file was open elsewhere.
    - TypeError: If the data contained non-serializable types.
    - OSError: For file system-related issues.
    - A general Exception for unexpected errors.

```
    try:
    file = open(FILE_NAME, 'w')
    json.dump(students, file)
    file.close()
    for student in students:
        print("Student", student['FirstName'], student['LastName'],
              "is enrolled in", student['CourseName'])
except PermissionError:
    print("Please close the file before saving.")
```

```
except TypeError:
    print("Data contains values that cannot be saved to JSON.")
except OSError as e:
    print("File system error while saving:", e)
except Exception as e:
    print("Unexpected error while saving:", e)
```

Summary

Through Assignment 5, I practiced:

- Using **dictionaries** to store structured data with key-value pairs.
- Reading and writing **JSON** files with Python’s json module.
- Distinguishing between **indexes** and **keys**.
- Implementing **structured error handling** for file operations and user input.
- Ensuring resources like files are properly closed with finally.
- Validating user input to prevent invalid data from entering the system.

This assignment moved the course registration program closer to a real-world application by improving **data structure design**, **data persistence format**, and **error resilience**.