Sophia Brooks

05/21/2025

IT FDN 110 A

Assignment #5

https://github.com/SEBrooks25/IntroToProg-Python-Mod05

# Advanced Collections & Error Handling

## Introduction

Module 5 provides essential skills for working with more advanced data structures and tools in Python programming. The focus begins with exploring the use of dictionaries, how they store data as key-value pairs and can be used to represent structured data, like rows in a table. This allows for the use of dictionaries to read from and write to files, making data storage more dynamic and organized. Building on this, JSON files are introduced as a standard format for exchanging data. The module demonstrates how Python's json module simplifies the process of converting dictionaries and lists into JSON-formatted files and vice versa. Building on the tools developed for troubleshooting coding issues the module delves deeper into error handling with structured try-except blocks showing how to gracefully manage common issues such as input errors or missing files. The use of custom exceptions and specific error types helps reinforce writing more reliable and user-friendly programs, and the importance of preventing errors by validating input and ensuring files are properly closed. The criticality of saving and sharing code was highlighted through the use of GitHub. Concepts like repositories, commits, and the GitHub web interface emphasized collaboration and version control. The tools and techniques covered in this module support more professional and maintainable code development.

## Dictionaries

Dictionaries are powerful data structures used to store and manage information through key-value pairs. Each key in a dictionary is case-sensitive, meaning that "Name" and "name" are treated as entirely different ID’s. This makes it important to consistently match key names when accessing or modifying data. Dictionaries come with several built-in methods such as .keys(), .values(), and .items(), which are useful for retrieving specific parts of the data or looping through its contents. Adding new data involves creating a new dictionary and appending it to a list, often representing a table of records. Removing data usually involves identifying a key or value in a specific dictionary within the list and using methods like .remove() based on a specific condition. Dictionaries can be written to and read from text files in a structured way, making them ideal for storing and retrieving organized data efficiently. By converting dictionaries to strings, or using Python’s json module, entire tables of dictionary-based data can be stored or retrieved efficiently. This is especially useful when dealing with tabular or form-like data that benefits from labeled fields. Overall, dictionaries offer a flexible and readable way to organize complex data in Python programs.

While both lists and dictionaries are used to store collections of data, their structure and use cases differ significantly. Lists store data in a linear, indexed order, making them ideal for ordered sequences where position matters. Dictionaries, on the other hand, use named keys to access data, which is more intuitive when working with records or labeled fields. When working with file data, lists are simpler and efficient for flat, line-by-line storage, while dictionaries are better suited for structured data where specific labels (like "FirstName" or "GPA") are important. Choosing between them depends on whether the data needs to be accessed by an index position (use a list) or by label (use a dictionary). In most real-world applications that involve readable or relational data, dictionaries tend provide more clarity and functionality.

## JSON Files

JSON (JavaScript Object Notation) files are a lightweight, human-readable format used to store and exchange structured data. They are built on key-value pairs, where keys are always enclosed in double quotes “”, and values can be strings, numbers, booleans, null, arrays, or other objects. Curly braces {} are used to define individual objects, while square brackets [] are used to define arrays, or lists of objects. This structure makes JSON a flexible way to store complex or nested data, such as a list of student records where each record contains personal details and additional attributes. In Python, the json module allows data to be easily converted between dictionaries and JSON format using json.dump() to write data to a file and json.load() to read it. JSON files are especially useful when working with web APIs, configuration settings, or datasets that have hierarchical or nested elements. Compared to CSV files, which store data in flat, row-and-column format, JSON supports nesting and varied structures within a single file. CSV is generally better suited for simple, tabular data such as spreadsheets, while JSON is ideal for more complex data relationships. If a project involves predictable columns and straightforward data, CSV is often the right choice. However, if the data includes arrays, objects within objects, or optional fields, JSON provides the flexibility to handle that structure cleanly. Understanding the formatting rules of JSON is essential to using it effectively, especially the requirement for double-quoted keys and correctly matched brackets and braces.

## Error Handling

Structured error handling is a crucial part of writing reliable and professional Python code. It allows programs to respond quickly and efficiently to unexpected situations rather than crashing or producing confusing results. The try-except block is the core structure used for catching and managing errors in Python. Within this block, code that might raise an error is placed inside the try section, and any exceptions that occur are handled in the except section. Using the Exception class, developers can capture detailed error information, including technical descriptions, error types, and helpful documentation, which can be printed for advanced users or logged for debugging. Specific exceptions, like ZeroDivisionError or FileNotFoundError, can be caught individually to provide tailored messages and recovery steps. Custom exceptions can also be raised to enforce program rules, such as ensuring a user does not enter numbers in a name field. This improves user experience by providing clear, targeted feedback. Structured error handling also supports error prevention by validating inputs and confirming whether files are open or closed. Through these methods many common issues can be avoided entirely. Using these techniques not only keeps programs from crashing but also makes them easier to troubleshoot and maintain. In larger or user-facing applications, structured error handling is essential for delivering dependable functionality and protecting the program from bad input or system issues. When used correctly, structured error handling turns unpredictable errors into manageable events.

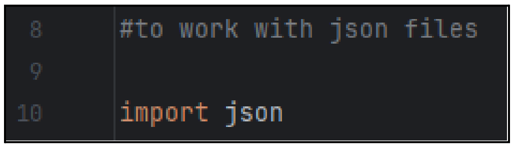
## Code Management & GitHub

Effective code file management is a foundational skill in software development, ensuring that work is organized, protected, and easy to access or share. This means saving and naming files consistently, maintaining backups, and knowing where code is stored. In team environments, developers often need to share files, and this can be done through network file sharing, where folders are hosted on a local server and made accessible to multiple users. This method is still used in many organizations and allows for collaboration within a secure internal system. Cloud file sharing has become a more modern solution, offering similar functionality over the internet through platforms like Google Drive, Dropbox, or Microsoft OneDrive. These services allow files to be accessed from anywhere, making remote collaboration easier and more flexible. However, it's important to consider privacy and compliance, especially when storing code that may include sensitive or proprietary information. Both methods help prevent data loss by providing backup options and ensure that multiple contributors can work on the same files without confusion. As projects grow and collaboration becomes more complex, these basic methods often give way to more advanced tools like GitHub, which offer additional control and history tracking.

GitHub is a cloud-based platform designed for managing and sharing code using version control. It allows developers to store code in repositories, track changes over time, and collaborate with others through features like pull requests, issue tracking, and commit histories. Users can access GitHub through its web interface, which makes it easy to create repositories, upload files, and manage updates. A GitHub repository acts like a folder that holds all project files and their version history, providing a centralized place to store and update collaborative work. Files can be public or private, and access can be controlled to ensure security or open collaboration. While being built for code, GitHub also supports documentation, images, and other file types. As part of a larger strategy for code management, GitHub offers a professional and scalable solution that integrates file storage, sharing, and version control into one platform.

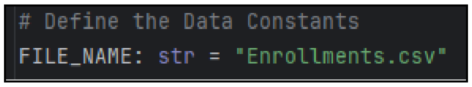
## The Assignment

To create the enhanced course registration interface using dictionaries, JSON files, and error handling, the first step involved adjusting the program's structure from the previous version. The program begins by importing the json module (Figure 1), which is essential for handling the reading and writing of data in JSON format—this allows the application to easily convert between Python dictionaries and JSON files for efficient data storage and retrieval.The file name was changed from "Enrollments.csv" (Figure 2) to "Enrollments.json" (Figure 3) and stored as a constant, along with a MENU constant to control what is displayed to the user each time they are prompted for input. The variables to hold the student's first name, last name, and course name remained similar to before, but the way that student data is stored is changed from lists to dictionaries. Instead of using position-based indexes, each registration is saved using descriptive key-value pairs; "FirstName", "LastName", and "CourseName". This makes the data more structured and easier to understand. All student dictionaries are stored inside a list called students, which allows multiple entries to be held in the memory at once. To begin the program, the code attempted to open the JSON file using the open() and json.load() functions. If the file existed and was valid, it would load all prior registrations into the students list. This ensured the data was ready as soon as the user launched the program.



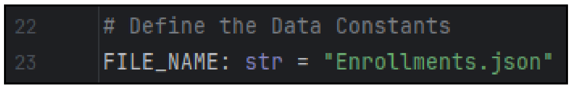
```
#to work with json files

import json
```

Figure 1: Shows import of the json module.



```
# Define the Data Constants
FILE_NAME: str = "Enrollments.csv"
```

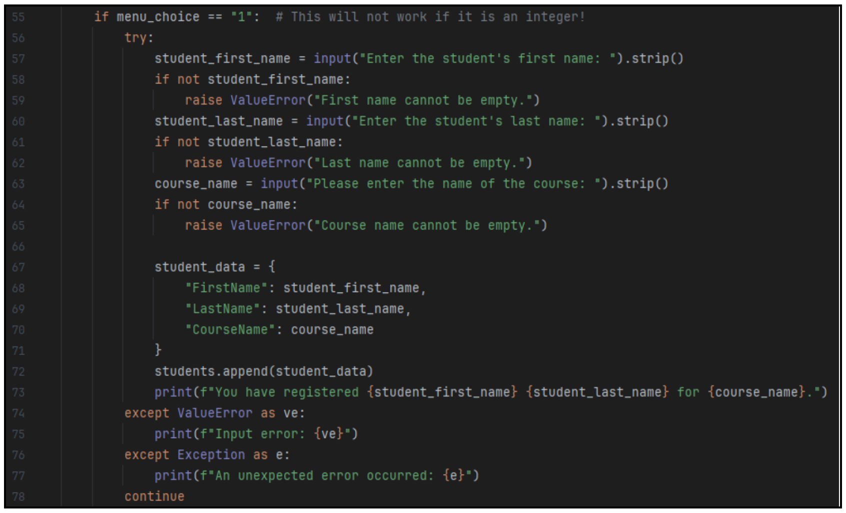
Figure 2: Shows initial code saving to csv.



```
# Define the Data Constants
FILE_NAME: str = "Enrollments.json"
```

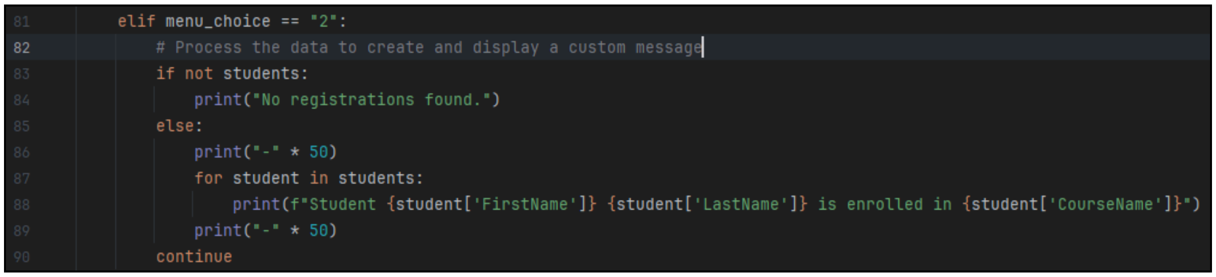
Figure 3: Shows updated code saving to json.

To collect new registration data, a while loop was used to present a menu with four options: register a student, view data, save data, or exit. When the user selects option 1 (Figure 4), they are prompted for the student's first name, last name, and course name. These values are used to create a dictionary representing a single student. The dictionary is then appended to the students list. This process allows for any number of new registrations to be added during a single session. If the user selects option 2 (Figure 5), the program loops through the students list and prints each student's full name and course in a readable sentence format. Because dictionaries are used instead of lists, values can be accessed by key names, which makes the code more intuitive. The output clearly shows what has been entered so far, making it easy to review before saving. The use of dictionary keys also removes the need to remember index positions, reducing the chance for mistakes during display. This structure also keeps each row of data organized and clean for future operations.



```
        if menu_choice == "1":  # This will not work if it is an integer!
            try:
                student_first_name = input("Enter the student's first name: ").strip()
                if not student_first_name:
                    raise ValueError("First name cannot be empty.")
                student_last_name = input("Enter the student's last name: ").strip()
                if not student_last_name:
                    raise ValueError("Last name cannot be empty.")
                course_name = input("Please enter the name of the course: ").strip()
                if not course_name:
                    raise ValueError("Course name cannot be empty.")

                student_data = {
                    "FirstName": student_first_name,
                    "LastName": student_last_name,
                    "CourseName": course_name
                }
                students.append(student_data)
                print(f"You have registered {student_first_name} {student_last_name} for {course_name}.")
            except ValueError as ve:
                print(f"Input error: {ve}")
            except Exception as e:
                print(f"An unexpected error occurred: {e}")
            continue
```

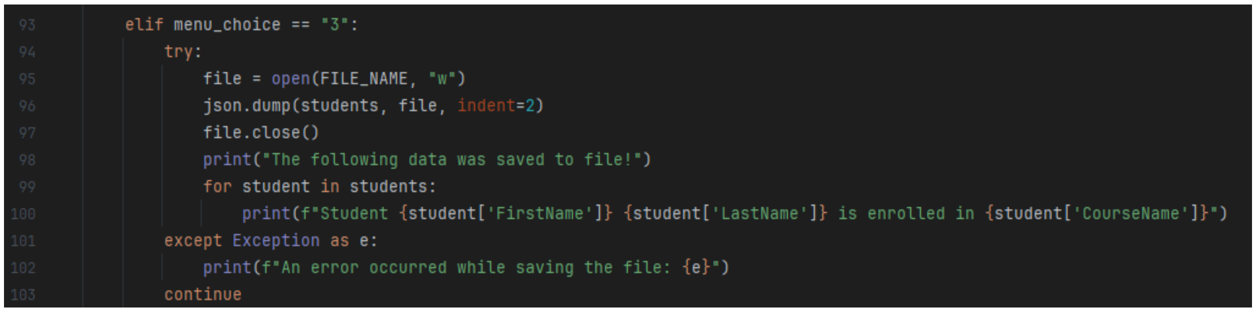
Figure 4: Shows the try-except block for handling option 1.



```
        elif menu_choice == "2":
            # Process the data to create and display a custom message
            if not students:
                print("No registrations found.")
            else:
                print("-" * 50)
                for student in students:
                    print(f"Student {student['FirstName']} {student['LastName']} is enrolled in {student['CourseName']}")
                print("-" * 50)
            continue
```

Figure 5: Shows the try-except block for handling option 2.

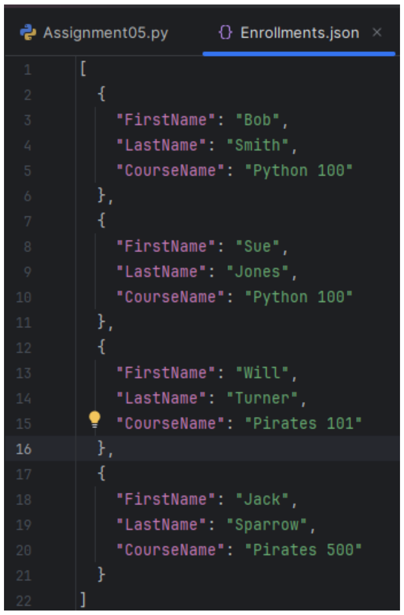
When the user chooses option 3 (Figure 6) to save the data, the program uses json.dump() to convert the students list into properly formatted JSON and write it to the file. This replaced the previous version's method of constructing CSV strings manually. A try-except block was used to handle any potential errors when reading or writing the file, ensuring the program would not crash if the file was missing or unreadable. Additional error handling was added for user input. If a user leaves a required field blank, a ValueError will be raised, and the program will notify the user without stopping. These changes made the application more user-friendly and less likely to fail unexpectedly.



```
        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!")
                for student in students:
                    print(f"Student {student['FirstName']} {student['LastName']} is enrolled in {student['CourseName']}")
            except Exception as e:
                print(f"An error occurred while saving the file: {e}")
            continue
```

Figure 6: Shows the try-except block for handling option 3.

As a result of storing each registration as a dictionary with clearly labeled keys, the saved JSON file displays the data in a clean, organized structure (Figure 7). Each student's entry appears as a separate dictionary within a list, making it easy to distinguish one registration from another. This format reflects the way the data was handled in the program—using descriptive keys like "FirstName", "LastName", and "CourseName" rather than relying on index positions. By using json.dump() with indentation, the output is not only machine-readable but also human-friendly, which is useful for reviewing or sharing the file. The structure mirrors how the data was entered during the session, providing a direct and readable link between the program's output and the contents of the saved file.



```json
[
  {
    "FirstName": "Bob",
    "LastName": "Smith",
    "CourseName": "Python 100"
  },
  {
    "FirstName": "Sue",
    "LastName": "Jones",
    "CourseName": "Python 100"
  },
  {
    "FirstName": "Will",
    "LastName": "Turner",
    "CourseName": "Pirates 101"
  },
  {
    "FirstName": "Jack",
    "LastName": "Sparrow",
    "CourseName": "Pirates 500"
  }
]
```

Figure 7: Shows the data in the “Enrollments.json” file.

## Summary

Mastering the use of dictionaries, structured error handling, and GitHub lays the groundwork for writing clean, reliable, and collaborative Python code. Dictionaries offer a powerful way to store and manage structured data through key-value pairs, making them ideal for representing real-world information such as records, forms, or database entries. Their flexibility and readability, especially when combined with tools like the json module, make them essential for working with dynamic files and complex datasets. Understanding the distinctions between dictionaries and lists allows for better design decisions, depending on whether the data is indexed or labeled. Alongside data management, structured error handling ensures programs remain stable and user-friendly, even when unexpected issues arise. Using try-except blocks and specific exceptions helps prevent crashes, guide users with clear messages, and streamline debugging. Validating inputs and safely managing files further reduces risk and improves program reliability. Incorporating custom exceptions reinforces program rules and supports clean, maintainable code. As projects grow in complexity and collaboration becomes necessary, GitHub provides a centralized platform for storing, sharing, and versioning code. Its tools for managing repositories, tracking changes, and supporting teamwork promote efficient and professional development practices. Together, these skills contribute to producing scalable, dependable, and well-structured applications that meet modern programming standards.