Sam Hollister

5/21/25

IT FDN 110 A

Assignment 05

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

Advanced Collections and Error Handling

Introduction

In this week's module, our focus was on handling exceptions and custom error messages, as well as the dictionary data type and JSON files. We learned how to use alternative methods to accomplish similar tasks to previous modules, adding in error handling.

Objective

This week's assignment was to create a program which collects student enrollment information, writes it to a file in JSON format, and includes structured error handling for missing files, and incorrect values or formats. The program should present the user with an option menu, read data from an existing JSON file, collect data from the user, and write that data back to the JSON file.

This assignment was very similar to last week's; the major differences were working with dictionaries and JSON files, and adding in error collection.

Dictionaries & JSON File Format

Dictionaries are similar to lists, but they use keys to define data elements and are not necessarily ordered. They are well-suited to storing data in a JSON file. In order to work with the JSON file, I added a section to the program to import the "json" library.

```
# Import libraries
import json
```

Figure 1.1: Importing the JSON library

I also changed the file name to be ".json" instead of ".csv", and the student data variable to be a dictionary instead of a list.

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

# Define the Data Variables and constants
student_first_name: str = ''  # Holds the first name of a student entered by the user.
student_last_name: str = ''  # Holds the last name of a student entered by the user.
course_name: str = ''  # Holds the name of a course entered by the user.
student_data: dict = {}  # one row of student data
students: list = []  # a table of student data
file = None  # Holds a reference to an opened file.
menu_choice: str  # Hold the choice made by the user.
```

Figure 1.2: Changes to constants and variables

Working the JSON library allowed me to perform some tasks more efficiently and with fewer lines of code. For example, reading the data from the existing JSON and adding to the students list.

```
    file = open(FILE_NAME, "r")
    students = json.load(file)
    file.close()
```

Figure 1.3: Reading from a JSON file

I was also able to save the data from the students list to the JSON using the json.dump() method.

```
        file = open(FILE_NAME, "w")
        json.dump(students, file)
        file.close()
```

Figure 1.4: Writing to a JSON file

Overall, working with the JSON file and library was able to greatly simplify the program over previous methods using CSV files.

Error Handling

The other objective of this assignment was to add error handling with custom messages to account for common potential errors in the program. The three main checks to implement were: confirming that the JSON file exists, making sure that the user was entering reasonable data, and that the data was saved successfully in the correct file format.

To confirm that the file exists, I added code to collect the FileNotFoundError and print the message “File not found” if the JSON file did not already exist. I also added a general exception for unspecified errors. Lastly, it makes sure the file is closed.

```
try:
    file = open(FILE_NAME, "r")
    students = json.load(file)
    file.close()
except FileNotFoundError as e:
    print("File not found.")
    print("--Technical Error Documentation--")
    print(e, e.__doc__, type(e), sep = "\n")
except Exception as e:
    print("Unspecified error.")
    print("--Technical Error Documentation--")
    print(e, e.__doc__, type(e), sep = "\n")
finally:
    if file.closed == False:
        file.close()
```

Figure 2.1: Handling the FileNotFound error

Next, I added code to handle potential value errors when entering the student's first and last name. I have an "if" statement to check if the inputted data contains only letters. If it contains non-letter characters, it will return an error message informing the user of the error, and not save the data.

```
try:
    student_first_name = input("Enter the student's first name: ")
    if not student_first_name.isalpha():
        raise ValueError("First name should only contain letters.")
    student_last_name = input("Enter the student's last name: ")
    if not student_last_name.isalpha():
        raise ValueError("Last name should only contain letters.")
    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)
except ValueError as e:
    print(e)
    print("--Technical Error Documentation--")
    print(e.__doc__)
    print(e.__str__())
except Exception as e:
    print("Unspecified error.")
    print("--Technical Error Documentation--")
    print(e, e.__doc__, type(e), sep = "\n")
```

Figure 2.2: Handling Value Type errors

Lastly, I added code to check for type errors before writing the data to the JSON file. If the data is not in a valid JSON format such as a dictionary, it will print this message to the user.

```
try:
    file = open(FILE_NAME, "w")
    json.dump(students, file, indent=2)
    file.close()
    print("The following data was saved to file:")
    print("-"*50)
    for student in students:
        print(f"FirstName: {student["FirstName"]}, LastName: {student["LastName"]}, CourseName: {student["CourseName"]}")
    print("-"*50)
except TypeError as e:
    print("Please check that the data is in a valid JSON format.")
    print("--Technical Error Documentation--")
    print(e, e.__doc__, type(e), sep = "\n")
except Exception as e:
    print("Unspecified error.")
    print("--Technical Error Documentation--")
    print(e, e.__doc__, type(e), sep = "\n")
finally:
    if file.closed == False:
        file.close()
```

Figure 2.3: Handling JSON type errors

Summary

Overall, I found this module to be a good challenge. Last week, when we covered dictionaries, I was curious if we would be using that data type with JSON formats, and I was excited to be able to do so. Most of my day-to-day work is with table-structured data, so working with alternative formats is very interesting.

I found the error handling to be interesting as well. It can be tedious at times, but the value is immediately apparent when comparing custom error messages to the default Python documentation.