Alexander Clark

06Nov2023

IT FDN 110 B

Assignment05

https://github.com/arclark360/IntroToProg-Python/

MOD 5 Advance Collections and Error Handling

Introduction

This week in module 5 kicked off with an introduction to a new collection type: dictionaries. These versatile data structures demonstrate how assigning "keys" can significantly enhance the organization of our collections. We also delved into the world of JSON (Java Script Object Notation) files, a new data storage type that will be replacing CSV in our assignment.

Diving further into our exploration, we rounded up the week with a comprehensive overview of error handling and the utilization of GitHub, a popular online repository for code. These newfound skills were used in upgrading the student registration program for this week's assignment.

Dictionaries

This week the new collection type we learned was called dictionaries. Like list they are mutable data structures that use **"key values"** to organize and access the data. For instance, instead of utilizing index numbers in a list you would use the key value to locate the information you were trying to access. As seen in **(Figure 1.)** one key difference between list and dictionaries is the use of **("{}")** in the syntax to identify the dictionary. Also each value in the dictionary must follow the **("key_name" : value)** format. The key is what you will call in replacement of the index number. Unlike list dictionaries are unordered so when sequence is important a list is preferable.

```
1  student = {
2      "name": "Alice",
3      "age": 20,
4      "grade": "A"
5  }
6
7  print(student["name"])  # Output: "Alice"
8  print(student["age"])   # Output: 20
9
10 student2 = [
11     "Alice",
12     "20",
13     "A"
14 ]
15
16 print(student2[0])  # Output: "Alice"
17 print(student2[1])   # Output: 20
```

Figure 1. Depicts a dictionary vs a list syntax.

JSON Files

Another new concept we tackled this week was JSON files. JSON, an acronym for "JavaScript Object Notation," serves as a means to store and transmit structured data in text format. My prior exposure to JSON mostly revolved around using it minimally within certain Power Automate features at my workplace, so I was eager to explore this concept in our class.

One noticeable distinction between CSV and JSON files is their opening symbols: **{}** or **[]**. When a file begins with {}, it signifies a **JSON Object**, while [] denotes a **JSON Array**. A JSON Object is akin to a collection of key-value pairs (dictionary), whereas a JSON Array is akin to a list.

One of the key advantages of using JSON files is their ease of reading and writing to the files. When working with CSV files, extracting information often necessitates using the **.readlines()** (unless importing) command to iterate through the file. In contrast, reading a JSON file simply involves importing the json module and employing **json.loads()**. This process instantly stores the data as a list without the need to format the information in the file. The illustration in **Figure 2** highlights the differences between reading a CSV file and a JSON file. When writing to a JSON file the syntax is also easy as you'll just use the **json.dump()** command to write to the file. The benefit here is that you don't have to format the list into a string and you can "dump" the list directly into the file.

```
import json

# Reading a JSON file
file =  open('data.json', 'r')
    data = json.load(file)
    print(data)
    file.close()

# Reading a CSV file
file =  open('data.csv', 'r')
    for row in file.readlines():
        data = row.strip.split(",")
        datacompiled += [data]
    file.close()
```

Figure 2. Depicts a reading to a JSON File vs CSV File

Exceptions

After learning to read and write JSON files we moved on to learning about how we can write python code that can handle exceptions/errors in the code using four new commands:

Try:

The try statement in Python is a crucial part of its exception handling mechanism. It serves as the initial segment of code that try's to execute the given instructions. It functions like an if statement, yet distinct in that it anticipates potential errors during its execution. Should an error surface during the attempt, the code swiftly transitions to the subsequent segment **"except"** for resolution. This distinctive structure

enables Python to gracefully handle unforeseen errors, ensuring a smoother and more controlled flow within the program.

Except:

The except statement stands as the complementary counterpart to the try statement. In the event that the code enclosed in the preceding try statement encounters an error, the program transitions to execute the instructions embedded within the except block. Here, you can craft tailored messages or specific operations designed to address and resolve the encountered exceptions. Furthermore, the except statement allows for the crafting of specialized responses tailored to distinct exceptions based on their "**raised"** names. An example of the Try and Except concept can be seen in **Figure 3**.

Raise:

The raise function can be used to throw an exception that the system normally wouldn't catch. This is usually code that would still have ran but not how you would of liked it to function.

Finally:

The finally section can be used to run anything that happens at the end of all the exception handling. In class we saw where you could reduce the code by having the file_obj the was opened to be read to get closed in the finally block at the end of the code instead of having written in all the Try and Except areas.

```
while True:
    try:
        file_obj = open(FILE_NAME, "r")
        students = json.load(file_obj)
        break
    except FileNotFoundError as e:
        print("JSON File not found, creating JSON file")
        file_obj = open(FILE_NAME, "w")
        json.dump(students, file_obj)
    except json.JSONDecodeError as e:
        print("File is not in right format and can't load students, please review file for error")
        break
    except Exception as e:
        print("Undefined Error")
        print(e, e.__doc__, type(e), sep="\n")
    finally:
        file_obj.close()
```

Figure 3. Depiction of Try, Except, and Finally

Github

This week, we concluded by establishing our GitHub accounts for use in our class. GitHub, an online code repository, serves as a platform for storing code and managing projects. It boasts distinctive features such as version control, allowing meticulous tracking of every file upload and changes made within those files. Additionally, GitHub offers robust collaboration tools, positioning it as a premier choice within the software development industry.

I'm particularly eager to explore this platform further as my analytics department at work also leverages GitHub. We will be utilizing GitHub going forward so that we can share our assignments throughout the weeks.

Assignment

Overview

This week we updated the student registration app to be able to read and write to JSON files instead of the previously used CSV files. This involves changing how we collect the data and how we handle exceptions within the code.

Constants and Variables:

Constants and variables retain their values from the previous week, with the exception of some adjustments made to **'student_data'** and '**students**'. This week, the variable **'student_data'** is redefined as a dictionary type, and **'students'** is now a list of dictionaries. An empty list is assigned to 'students'. This ensures that if a file isn't found, an empty list will be written to the JSON file upon its creation. **Figure 4** depicts all constants and variables used in the code.

```
# Define the Data Constants
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"

# Define the Data Variables
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.
csv_data: str = ""  # Holds combined string data separated by a comma.
file_obj = None  # Holds a reference to an opened file.
menu_choice: str = ""  # Hold the choice made by the user.
student_data: dict[str, str, str]  # Holds a dictionary value of student information for the JSON
students: list[dict[str, str, str]] = []  # Holds the list of dictionary's for the JSON
```

Figure 4. Constants and Variables

Startup

At the beginning, we needed to tweak last week's code to enable reading a JSON file. First, we incorporated the '**import json'** command, facilitating the loading of the json module into Python. Instead of utilizing the '**readlines()'** command and a for loop, as done with the CSV file, we streamlined the process by employing '**json.load(file_obj)'.** This efficiently loads the existing data into the 'students' list.

Within my code as seen in **Figure 5**, I implemented handling for three distinct exception types, ensuring seamless management in case of errors while attempting to load the file. If no file is found, the code writes and creates a JSON file housing an empty '**[]'**. In case of a file format issue, it prompts the user to verify the file for syntax problems. Lastly, a general exception clause is incorporated to address any other unexpected issues that might arise.

Additionally, I employed the approach of including '**the .close()'** command within the '**finally'** block for optimal code execution. All this code is nested within a while loop, offering various break-out points.

```
while True:
    try:
        file_obj = open(FILE_NAME, "r")
        students = json.load(file_obj)
        break
    except FileNotFoundError as e:
        print("JSON File not found, creating JSON file")
        file_obj = open(FILE_NAME, "w")
        json.dump(students, file_obj)
    except json.JSONDecodeError as e:
        print("File is not in right format and can't load students, please review file for error")
        break
    except Exception as e:
        print("Undefined Error")
        print(e, e.__doc__, type(e), sep="\n")
        break
    finally:
        file_obj.close()
```

Figure 5. Loading of the JSON File

Choice 1

This week, several adjustments were made to choice one. Firstly, I introduced a while loop and a try-except sequence to each name input, ensuring an error is raised if the user inputs a number as the name. I used the '**.isalpha()'** command to test for the string. This setup guarantees the input comprises a genuine name.

Additionally, I revamped the storage of student data. It now involves storing the input within a dictionary, featuring three keys **(first_name, last_name, course_name**), which is then appended to the students list. This marks a departure from last week's method, where we stored data as a list within a list. **Figure 6** depicts the modifications to choice 1.

```python
# Option 1 Input user data
if menu_choice == "1":  # This will not work if it is an integer!
    # While loop and error handling to ensure user enters an accurate first name
    while True:
        try:
            student_first_name = input("Enter the student's first name: ")
            if not student_first_name.isalpha():
                raise ValueError("first name must be alphabetic")
            break
        except ValueError as e:
            print(e)
    # While loop and error handling to ensure user enters an accurate last name
    while True:
        try:
            student_last_name = input("Enter the student's last name: ")
            if not student_last_name.isalpha():
                raise ValueError("last name must be alphabetic")
            break
        except ValueError as e:
            print(e)
    course_name = input("Please enter the name of the course: ")
    # Stores user input as a dictionary item in the students list
    student_data = {"first_name": student_first_name, "last_name": student_last_name, "course_name": course_name}
    students.append(student_data)
    continue
```

Figure 6. Using try and except to catch incorrection user input in choice 1

Choice 2 & 3

Only changes needed for choice 2 this week was replacing all the index numbers with the **'key's'** for the values since we were now utilizing dictionary to store the student data which are unorganized data structures. **Figure 7** depicts the modification.

```python
# Option 2 Present the current data of both the list and a string version of the list
elif menu_choice == "2":
    print("\nThe current data is:")
    print(f"List Data: {students}")
    print("String Format:")
    for single_student in students:
        print(f"{single_student["first_name"]},{single_student["last_name"]},{single_student["course_name"]}")
    continue
```

Figure 7. Changed index to keys in choice 2

In choice 3 I added another exception to handle any errors that might occur during the writing of the JSON file. I also switched to the '**.dump()'** command to write to the JSON file since we can now just write students directly to file without it needing to be a string. Then like choice two the index positions were changed to the associate keys. **(Figure 8.)**

```python
# Option 3 Save the data to a file
elif menu_choice == "3":
    # Error handling setup to catch any errors that might occur when writing to the file that will notify the user
    # an error occurred and the data wasn't saved
    try:
        file_obj = open(FILE_NAME, "w")
        json.dump(students, file_obj)
        file_obj.close()
        for single_student in students:
            print(
                f"You have registered {single_student["first_name"]} {single_student["last_name"]} for {single_student["course_name"]}.")
        continue
    except Exception as e:
        print("Undefined Error Data was not saved")
        print(e, e.__doc__, type(e), sep="\n")
```

Figure 8. Choice 3 modifications

Choice 4&5

No modifications were needed for choice 4 or 5.

Test Output

```
C:\Python\Python3.12\python.exe: can't open file 'C:\\Users\\arcla\\OneDrive\\Desktop\\Python Mods\\_Module05\\_Mod
PS C:\Users\arcla\OneDrive\Desktop\Python Mods\_Module05\_Module05\Assignment> python.exe Assignment05.py

---- 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: Shelly
Enter the student's last name: Lim
Please enter the name of the course: Python 4

---- 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

The current data is:
List Data: [{'first_name': 'Alex ', 'last_name': 'Clark', 'course_name': 'Python 100'}, {'first_name': 'Jeff', 'las
e': 'Python 4'}]
String Format:
Alex ,Clark,Python 100
Jeff,Shu,Python 200
Feon,Shu,Python 3
Shelly,Lim,Python 4

---- 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.
----------------------------------------
```

Figure 9. Terminal Test part A

```
What would you like to do: 3
You have registered Alex  Clark for Python 100.
You have registered Jeff Shu for Python 200.
You have registered Feon Shu for Python 3.
You have registered Shelly Lim for Python 4.

---- 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
PS C:\Users\arcla\OneDrive\Desktop\Python Mods\_Module05\_Module05\Assignment>
```

Figure 10. Terminal Test part B

```
C:\Python\Python3.12\python.exe "C:\Users\arcla\OneDrive\Desktop\Python Mods\_Module05\_Module05\Assignment\Assignment05.py"

---- 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: Feon
Enter the student's last name: Shu
Please enter the name of the course: Python 3

---- 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

The current data is:
List Data: [{'first_name': 'Alex ', 'last_name': 'Clark', 'course_name': 'Python 100'}, {'first_name': 'Jeff', 'last_name': 'Shu', 'course_name': 'Python 200'}, {'first_name': 'Feon', 'last_name': 'Shu', 'course_name': 'Python 3'}]
String Format:
Alex ,Clark,Python 100
Jeff,Shu,Python 200
Feon,Shu,Python 3

---- 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
You have registered Alex  Clark for Python 100.
```

Figure 11. PyCharm Test part a

```
You have registered Jeff Shu for Python 200.
You have registered Feon Shu for Python 3.

---- 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

Process finished with exit code 0
```

Figure 12. PyCharm Test part b