Jesse Huminski

11/12/2024

IT FDN 100 A

Assignment 05

[GitHub URL](https://github.com/JMH1707/IntroToProg-Python-Mod05)

Dictionaries and JSON Files

# Introduction

Module 05 introduces us to dictionaries as a data structure, which allow for data in our code to be categorized better for easier readability. By utilizing dictionaries, we can classify our collection of data by using keys for our data values. Furthermore, we are taught how to add dictionaries to a list (list of dictionaries) and then call back to that list so the data can be printed and displayed to the user. This module also introduces us to JSON files in a similar way to how our code can work with CSV files. While JSON files can work similarly to CSV files regarding writing and accessing our data, JSONs allow for more complex data classification and human readability and allow us to manipulate and store more robust, hierarchical data.

Lastly, this module introduces the concept of Structured Error Handling and how we can prepare our program for potential issues. Error handling is essential as it informs the user of the error and can allow them to correct the error moving forward. Structured error handling allows the programmer to plan for potential errors that may arise in their code whether from incorrect user input or other errors that may arise when the program is running. This principle ensure that, should your code encounter an error, it is able to process the error and attempt to work around it. Or, it can inform the user of the issue it encountered, and allow them to relate that information to programmer for potential correction.

# Creating the Script

The requested script for Assignment05 requests that we gather data from the user in terms of their name and the course they wish to sign up for. Instead of writing the data to a CSV file using a list, we are told to write the data to a JSON using the keys and values of dictionaries. Additionally, the code should be able to process errors the code may encounter and perform operations that will inform the user of any encountered errors.

As always, to start off, we have the script header. Additionally, since I wanted to use functions of the JSON code, I imported that as well. (Image 1.1)

A screen shot of a computer

Description automatically generated

Image 1.1: Script header with necessary information.

Afterward, I define the constants and variables I wish to use and ensure that their type hints are included for others who may wish to work with this code. (Image 1.2)

A screenshot of a computer program

Description automatically generated

Image 1.2: Defining the constants and variables

We are requested to load information from a JSON file, “Enrollments.csv,” and assign it to the list variable “students.” Initially, I had a bit of an issue with this as I tried to it to the dictionary, “student\_data.” I didn’t document my thought process, which in hindsight, I obviously should have. My thoughts were that the data would come from the JSON file in that dictionary format, so I should assign it to the dictionary variable. Makes sense, right? I also attempted to try and convert the data from dictionary form to list form using indices. All in all, the way I did it was wrong, and I spent quite a while troubleshooting why. Eventually, I did get it to work properly. I set up structured error handling for this as well. If the file does not exist, the code will inform the user and exit the program. Should any other issue arise, it will do the same. At this point, I wanted the code to quit upon reaching an error. (Image 1.3)

A computer screen shot of a program code

Description automatically generated

Image 1.3: Opening and reading the data from the JSON and included error handling

I start the loop here as this is the point where we want the code to start over after each case has been completed. From there, we print the menu of options for the user and request their input. This is where I start the Match/Case flow statement. I prefer to use match-and-case as I feel it give me more control over the program for this type of user input over the if-elif statements. (Image 1.4)

A screen shot of a computer

Description automatically generatedImage 1.4: Present menu of choices, gather input from user, and start Match/Case flow.

For case 1, I want to gather data from the user in the form of their first and last name, and the course they wish to enroll in. For first and last name, I introduce structured error handling to only allow alphabet values (A-Z). Additionally, I use a general exception statement to raise an error should any other issues arise. (Image 1.5)

A screen shot of a computer program

Description automatically generated

Image 1.5: Gathering user input with structured exception handling

If an error arose from the user’s input containing a numeric value when entering either their first or last name, I did not want the code to move on to the next step. I utilized a nested While loop here to force a repeat of the input until it is entered correctly. Otherwise, the data would not be assigned to the variable and written to the JSON file correctly further on in the code. I did have some indenting issues here when incorporating the nest While loop. I had written the code already and, when testing, noticed that the code moved on after encountering the exception. I ended up having to rewrite the entire input of the first and last name because I could not get the indenting right. Something I know I need to work on!

I moved to gather input from the user in the form of “course\_name” and printed a message to them indicating that they have successfully enrolled in their course. I assigned the collected input data to the dictionary variable “student\_data” and appended that dictionary information to the list “students.” After the dictionary has been written to the list properly, I loop back to the beginning of the code where the menu of options is printed and the process repeats. (Image 1.6)

A computer screen shot of a program code

Description automatically generated

Image 1.6: Finish gathering data from user. Print a message displaying gathered information, assign to dictionary variable and append to list variable.

In case 2, the data collected from previous program iterations can be printed and displayed to the user upon request. Additionally, if data was gathered from users from case 1 prior to case 2 being chosen, that data will also be displayed to the user. I implemented a FileNotFoundError exception here as data can’t be properly displayed from a file that doesn’t exist. But this seems like a redundancy to me as, if the file doesn’t exist, the code that reads the data from the file would close the program before we’re even able to get to this point. (Image 1.7)

A computer screen shot of a program code

Description automatically generated

Image 1.7: Displaying collected data to the user in a readable format.

In case 3, we want to save the previously collected data and any data that was gathered from the user when the program runs to the JSON file, “Enrollments.csv.” When this case is chosen, the code will inform the user that their data was saved to the file and include its name. I use the json.dump() function to dump all the information in the variable “students,” and ensured that data was written to the JSON in a readable format for later use.(Image 1.8)

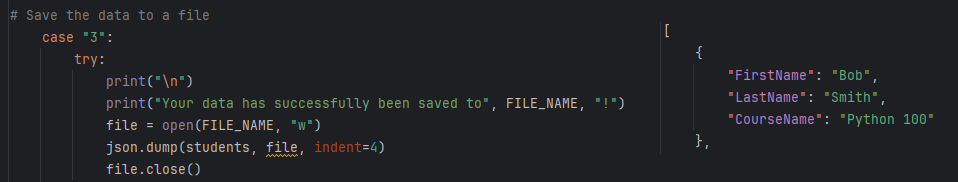
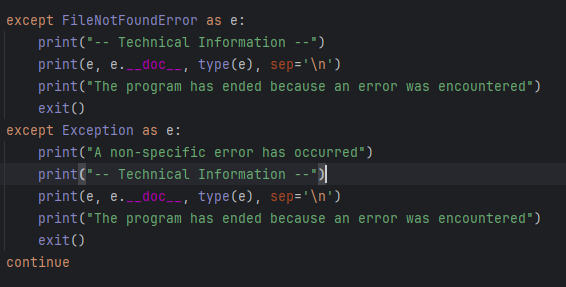


Image 1.8: Written code displaying information that is saved to the file (left). An example of the data that was written to the JSON file. This is repeated for each student (right).

I also included error handling here for any exception that may occur as well as the FileNotFoundError. Again, I feel this is a redundancy. But, it is nice to include. (Image 1.9)



**Image 1.9: Exception handling for saving data.**

Case 4 will break from the loop and end the program. As in the past, I included a selection should the user not enter a number that corresponds to the menu that was printed at the beginning of the program. (Image 1.10)

A screenshot of a computer program

Description automatically generated

**Image 1.10: Break from loop. Error handling incorrect selection from user input**

# Running the Program

Now, it is time to run the program. Once selected to run in PyCharm, the data is loaded in the “students” variable. We can test this by inputting option 2 when the program starts. (Image 2.1)

A screenshot of a computer screen

Description automatically generated

Image 2.1: Showing information that was loaded from the JSON file to ensure it is working correctly.

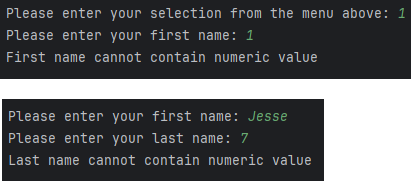
The program starts and displays the menu of options to the user and requests their input. (Image 2.2)

A screen shot of a computer program

Description automatically generated

Image 2.2: Showing the user their options and requesting their input.

When we press case 1, the program asks us for our name. I want to test the error handling here to see if it works as intended. So, I will enter a numeric value for both First Name & Last Name. (Image 2.3)



**Image 2.3: Error handling working as intended.**

You see that after encountering the exception from the First Name, the code asks me to enter my first name again. Upon receiving an appropriate response, the code break from the nested While loop and move to the next step, asking me for my last name. This happens again if an exception is encountered when asking for a last name.

Let’s enter our data correctly to see what happens. (Image 2.4)

A screenshot of a computer program

Description automatically generated

**Image 2.4: Data collected from user correctly. A success message is displayed to the user with the collected data.**

Let’s check to see that the data was appended to the dictionary and added to the list of dictionaries by pressing 2. (Image 2.5)

A screenshot of a computer screen

Description automatically generated

Image 2.5: Collected data displayed back to user to ensure it is working as intended

Let’s save the collected data to the JSON file. I’m going to add multiple students to ensure that the program can collect multiple lines of data from different users, append it to the list, and save it to the file. Before saving to the JSON, I display the collected data to confirm it is working as intended. (Image 2.6)

A black background with white text

Description automatically generated

Image 2.6: Displaying the collected data from previous iterations of the program as well as the newly collected data from students.

I then press option 3 to save the data and option 4 to quit the program. (Image 2.7)

A screen shot of a computer program

Description automatically generated

Image 2.7: Success message for saving data. Option 4 quits the program and informs the user of doing so.

Now, I want to look at the JSON file to see if my data collected and saved properly. (Image 2.8)

A screen shot of a computer program

Description automatically generated

Image 2.8: The data contained in the JSON file saved properly and in a format that is easily readable for later use.

The program appears to be working as intended! Let’s run it quickly through the command shell. (Image 2.9)

A screenshot of a computer program

Description automatically generated

Image 2.9: Testing in command shell is successful. Data written to JSON correctly.

# Summary

We expand our knowledge of the Python language by working with dictionary data collections and JSON files. Dictionaries allow us to compartmentalize data in a readable format by using categories (keys) for data (values). JSON file allow for information to be readable by humans who will be looking at the data later on. But, can be significantly larger than CSV files due to the expanded information collected. We are introduced to importing additional Python modules that allow us to use other functions and classes so that we can use them in our own code. Lastly, we also are introduced to GitHub as a place where we can share our data with other users. I am a bit familiar with GitHub already, which is why I have really mentioned it. So, sorry about that!