

Scott Duncan

11/01/2024

IT FDN 110 A Au 24: Foundations Of Programming: Python

Module 05 - Advanced Collections and Error Handling

GitHub URL: <https://github.com/gsdunca/Python110-Fall2024.git>

Introduction

In Module Five I was exposed to advanced data collections, how to work with program errors, and how to develop custom error handling routines. As with all Python learning modules thus far, we were able to begin our exploration by using a “Starter File” as a template to build upon. In this learning module the student was introduced to the differentness between a Coma Separated Variable (CSV) and a JavaScript Object Notation (JSON) file extensions. Learning the differences between lists, and dictionaries became a foundation for development of Python programming that incorporated json files to read and store data sets. Although I personally find the csv file is much easier to read, manipulate and work with overall, I hope to fine the general love for json files by the end of my Foundations of Programming in Python Couse end.

Lists verses Dictionaries

In the development of our knowledge of programming Python to work with .json file extensions it was important to understand the differences between a Python “List” and a Python “Dictionary”. In module Four we worked with creating lists by working with data files in the format of Coma Separated Variable (CSV) files. In this module Four it was shown that a csv file could be loaded into memory with the “readlines” command creating a list in the computer’s memory. Where in working with JavaScript files Python needed to have a special module loaded into memory to allow the program language the ability to work with .json file types. At the beginning of each program, it was necessary to instruct or load the json module by adding the command “import json” to the Python program. This simple command allowed python to load a JavaScript module that brought into play a set of specialized commands developed to open, save and write to json files. Once this module was loaded Python had the ability to work with Dictionaries. Much like a table in a csv file the dictionary in a json file was viewed as a row of data. But instead of using an index to locate sections of data, the dictionary used “Keys and Value” pairs to locate data locations. Unfortunately, I have not learned how to make the json file more humanly readable, as its data is all on a single line unlike a typical csv. **(Figure 1)**

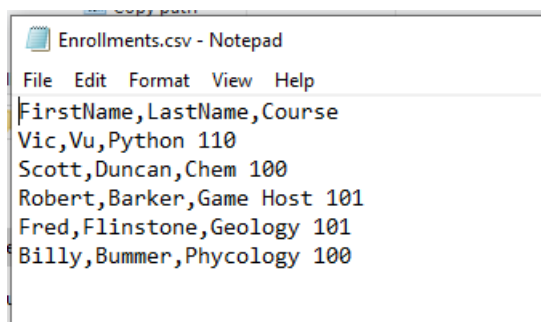


Figure 1: Typical CSV viewed in a text editor

The json file is all on a single line making it much more difficult to read. **(Figure 2)**

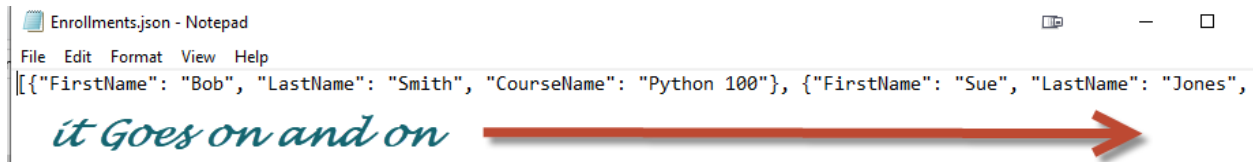


Figure 2: Typical JSON file as viewed in a text editor

Moreover, the syntax for working with json files versus lists became important due to the fact the list used square brackets [] in the Python programming where the json dictionary uses braces { } to denote data sets. **(Figure 4 & 5)**

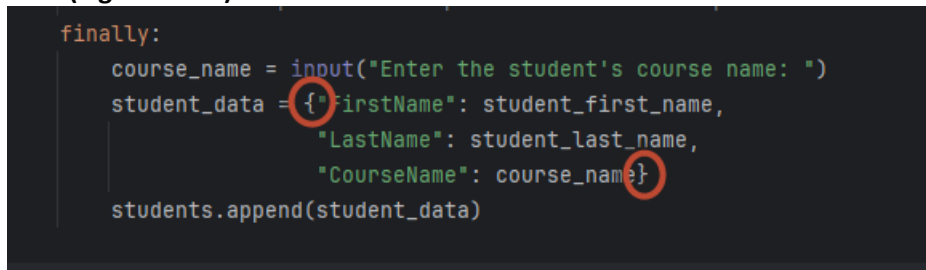


Figure 4: Braces used in the Python code

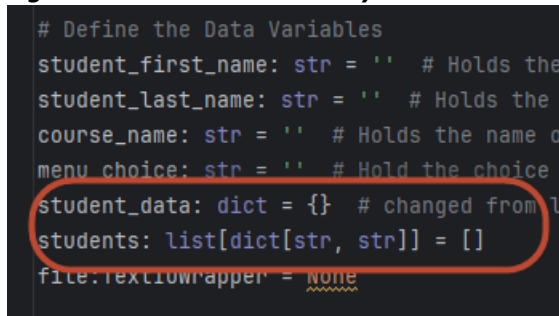


Figure 5: Braces used in variable definitions

In Module Five's reading material, it was discussed the differences between a csv and json file including their structure, readability, complexity, use cases and more. I can see that json files have their usefulness and adaptability to various programming languages, but at this point in my programming journey, I prefer to work with the simpler csv file format.

Introduction to structured Error Handling

One of the more useful sections of Module Five was the introduction to customizing errors that may come up during the development of a program. The more I work in Python I want to include areas of my code that will trap misconfigured sections of my program and display errors that will assist me in quickly resolving issues. Until now I have needed to research what an error code maybe in order to find the misconfigured code and correct the issue. Also, with custom error handling, I can include areas of my

program that can trap potential user input errors that will inhibit the users from successfully using my program.

In this section of the learning material, it was learned that using an exception class we can configure a “try-except” block of code to gracefully handle errors. In creating an exception object Python will fill the exception object with information about the error and what may have caused this error. In this assignment it was requested error handling was used for missing or malformed files upon the programs initial loading of the data. With this error handling, it was discovered that misconfigured json key and index values can be pointed out allowing the user to double check the file for potential issues. **(Figure 6)**

```
''' # Error Handling for a missing file '''
except FileNotFoundError as e:
    print("")
    print(f" ERROR: \n The file {FILE_NAME:} does not exist")
    print("")
    print(e, e.__doc__, type(e), sep='\n')
    print("")
    print("")

''' # Error Handling for m2
missing or syntax errored file keys '''
except KeyError as e:
    print("")
    print(f" ERROR: \n The Category (Key) {e.args[0]:} does not exist or is malformed")
    print("")
    print(e, e.__doc__, type(e), sep='\n')
    print("Exiting the Program")
    exit()
```

Figure 6: Error handling with file loading issues

Additional error handling was incorporated in this assignment to alert the user to data input errors. One of these error handling processes was to ensure that the user input the students name as an alphabetic character only, eliminating the potential for incorrectly entered username data. **(Figure 7)**

```

# Input user data
if menu_choice == "1": # This will not work if it is an integer!
    ''' this Error Handling was taken from Module 5 course notes
        I attempted to create my own but failed '''
    try:
        student_first_name = input("Enter the student's first name: ")
        if not student_first_name.isalpha():
            raise ValueError("--> Please enter the student's first name in Alpha format ONLY <--")
    except ValueError as e:
        print(e) # Prints the custom message
        print("-- Technical Error Message -- ")
        print(e.__doc__)
        # print(e.__str__())
        print("")
        print("Lets start again")
        continue # upon user's input error it will loop back to the main menu
    try:

```

Figure 7: Typical Alphabetic error handler

Unfortunately, I was not able to create this section of code from memory, or from scratch, I had to resort to using the code Professor Root provided in the learning materials. Testing this code I did prove that the input of a user's name "Scott" was accepted while the username "Sc0tt" provided an error message, that I was able to customize. **(Figure 8)**

```

---- 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: Sc0tt
--> Please enter the student's first name in Alpha format ONLY <--
-- Technical Error Message --
Inappropriate argument value (of correct type).

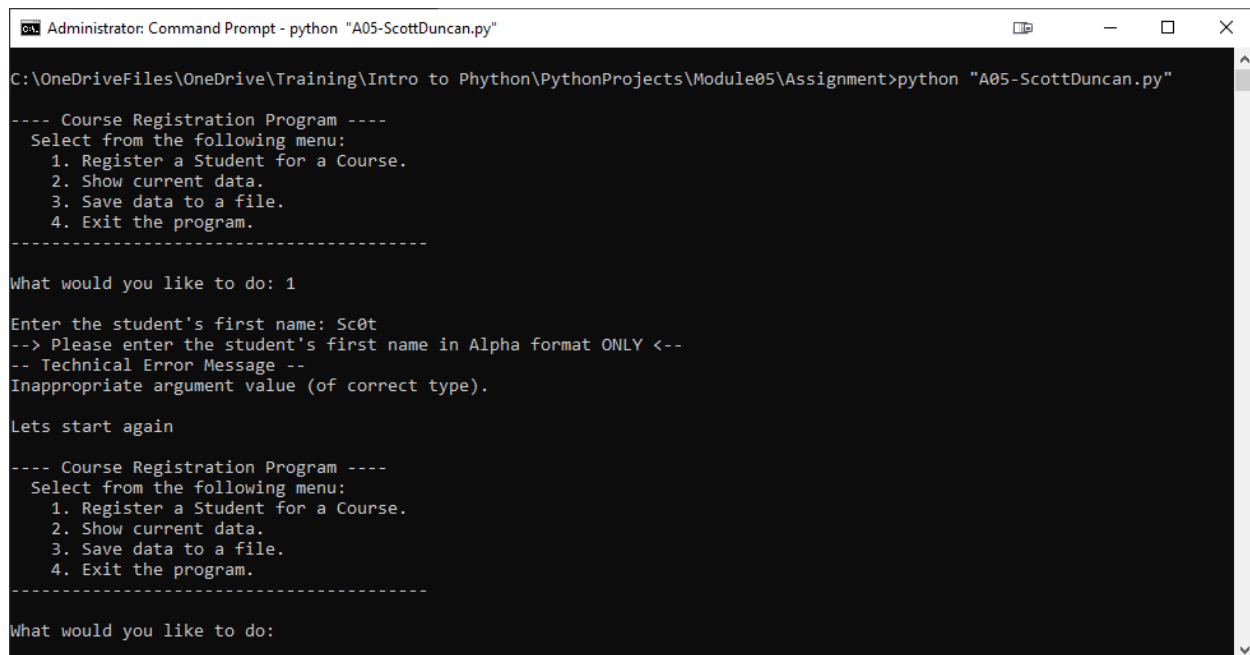
Lets start again

---- Course Registration Program ----
Select from the following menu:

```

Figure 8: Username Input Error

This error handling worked in PyCharm as well as windows command window. **(Figure 9)**



```
Administrator: Command Prompt - python "A05-ScottDuncan.py"

C:\OneDriveFiles\OneDrive\Training\Intro to Python\PythonProjects\Module05\Assignment>python "A05-ScottDuncan.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: Sc0t
--> Please enter the student's first name in Alpha format ONLY <--
-- Technical Error Message --
Inappropriate argument value (of correct type).

Lets start again

---- 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:
```

Figure 9: Error handling in Command Window

More technical information was shared in the error handling module that will allow me to dive deeper into my error handling configurations going forward. At this point in my Python journey it is pretty much “smoke and mirrors” to me, but I hope to understand how I can use the built-in attribute codes such as “e.__doc__” and “e.__str__()” and the various exception classes to my advantage.

Sharing and Managing Code Files

As an exercise in Module Five it was requested that we learn how to use a file sharing service such as GitHub. Following the instructions Professor Root had in our Module Five reading I was able to create a profile, and upload my files as requested. These files can be viewed at the following URL:

<https://github.com/gsdunca/Python110-Fall2024.git>

Summary

In module five I learned I need to learn more about json files, and their potential uses. Once again, I leave this module as I have with other modules reviewed in the past few weeks with the desire and need to know more about the various aspects of the Python programming language. I hope in future lessons we will learn how to provide the users with a more friendly interface such as a GUI that will be easier for the users to navigate.

Although the information in this module was a surface view of what Python can do with Data Sets and Error Handling, I can see many applicable uses in my professional life. As an example, I hope to configure Python programming that will allow an Junior Network Engineer to use a custom Python program that will provide a more user-friendly interface to aid them in resolving simpler network infrastructure configurations and issue isolation.