

Rob Armstrong

August 1, 2024

IT FDN 110 A

Assignment05

GitHub URL: <https://github.com/robstrong517/IntroToProg-Python-Mod05>

Intro to Programming with Python

Module 5 – Advanced Collections and Error Handling

Introduction

This week allowed for more of a personal touch with code. Learning to create the lists last week helped build to the use of JSON files this week, but the personalization of the code is what I particularly enjoyed this week. As in weeks prior, adding a personal touch to the print statements gives a sense of accomplishment. Being able to build in exception handling with try statements took this personalization to a new level.

Writing the Code

As it has been each week, the first step has been to define the constants, variables, and determine the output file. This week, the use of lists has expanded into the use of dictionaries by using JSON (JavaScript Object Notation). Just like the weeks prior, defining the constants is the first step.

```
# Define the Data Constants
import json

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.
-----
'''

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

FIGURE 1: CONSTANTS, INCLUDING IMPORT JSON FEATURE.

While the variables and constants are largely the same as every other week, two new features can be noted within the lines of script. The new features used this week are dictionaries and JSON files.

```
# 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.
file: None # Holds a reference to an opened file.
menu_choice: str = '' # Hold the choice made by the user.
student_data: dict = {} # one row of student data
students: list = [] # a table of student data
```

FIGURE 2: VARIABLES WITH NEW TYPE, DICTIONARIES.

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

except FileNotFoundError as e:
    print("Text file must exist before running this script!\n")
    print("-- Technical Error Message -- ")
    print("Creating new file...")
    file = open(FILE_NAME, "w")
    print(e, e.__doc__, type(e), sep='\n')

except Exception as e:
    print("There was a non-specific error!\n")
    print("-- Technical Error Message -- ")
    print("Resetting session data...")
    students = []
    print(e, e.__doc__, type(e), sep='\n')

finally:
    if file.closed == False:
        file.close()
```

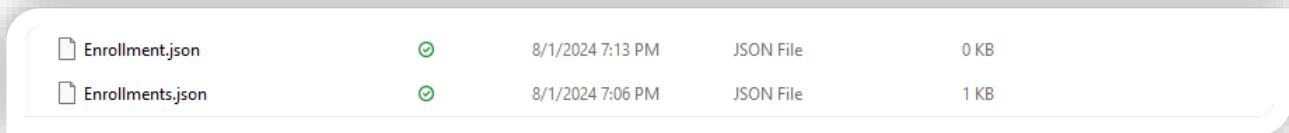
FIGURE 3: READ-FILE FOR NEW OUTPUT, JSON, FOLLOWED BY ANOTHER NEW FEATURE, TRY-EXCEPT.

JSON Files

At this level of familiarity with JSON and CSV files, they appear similar so far. The text describes JSON as being more suitable for web-based configuration while CSV is more suitable for spreadsheets, databases, and tables. JSONs have more flexibility than CSVs and can support the use of strings, integers, and Boolean operators without additional coding framework. One of the benefits of using JSON files is that indices are not required to decode the information within, or to write to file. So long as the “Keys”, or item types what would represent a column header, are well-defined, the contents can more easily be transcribed.

As can be seen in *Figure 3*, there is additional code that follows the read command for the JSON file. This code provides an immediate correction should there be an error locating the correct JSON file. The correction

provided above for “FileNotFoundError” is to create a file as defined in the constants. *Figure 4* shows a successful creation of a new JSON file, following the defined naming convention.

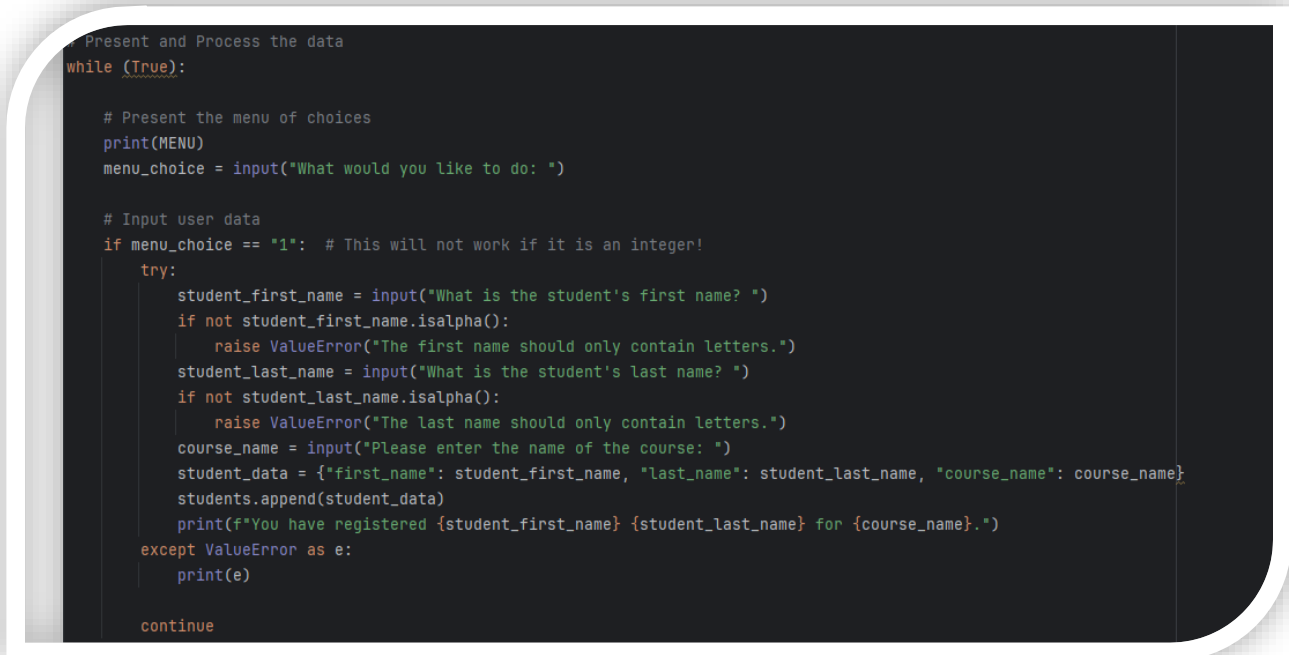


Enrollment.json	✓	8/1/2024 7:13 PM	JSON File	0 KB
Enrollments.json	✓	8/1/2024 7:06 PM	JSON File	1 KB

FIGURE 4: NEW JSON FILE CREATED IN DESIRED FOLDER BY USING TRY-EXCEPT METHOD.

Error Handling

As seen in the figures above, another new feature this week is the use of error handling. You begin error handling by providing a “Try” statement, followed by any number of “Except” statements. To break this error handling loop, a “Finally” statement must be used to indicate the last step before resuming script execution.



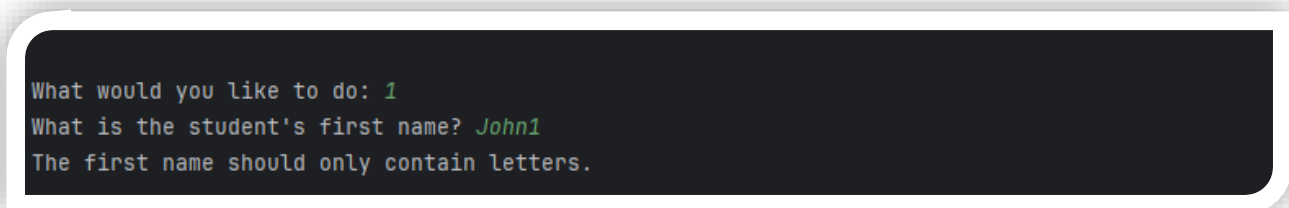
```
Present and Process the data
while (True):

    # Present the menu of choices
    print(MENU)
    menu_choice = input("What would you like to do: ")

    # Input user data
    if menu_choice == "1": # This will not work if it is an integer!
        try:
            student_first_name = input("What is the student's first name? ")
            if not student_first_name.isalpha():
                raise ValueError("The first name should only contain letters.")
            student_last_name = input("What is the student's last name? ")
            if not student_last_name.isalpha():
                raise ValueError("The last name should only contain letters.")
            course_name = input("Please enter the name of the course: ")
            student_data = {"first_name": student_first_name, "last_name": student_last_name, "course_name": course_name}
            students.append(student_data)
            print(f"You have registered {student_first_name} {student_last_name} for {course_name}.")
        except ValueError as e:
            print(e)

        continue
```

FIGURE 5: MENU CHOICE 1 INPUT WITH ERROR HANDLING.



```
What would you like to do: 1
What is the student's first name? John1
The first name should only contain letters.
```

FIGURE 6: MENU CHOICE 1 WITH TRIGGERED ERROR HANDLING.

```

Save the data to a file
elif menu_choice == "3":

    try:
        file = open(FILE_NAME, "w")
        json.dump(students, file)
        for student in students:
            #item IS THE VARIABLE YOU CHOSE FOR EACH INDIVIDUAL DICTIONARY, NOT student
            print(f"Student {student["first_name"]} {student["last_name"]} is enrolled in {student["course_name"]}!")
        file.close()
    except Exception as e:
        print("Error saving data to file.")
        print(e)
    finally:
        if file.closed == False:
            file.close()
    continue

```

FIGURE 7: MENU CHOICE 3 WITH ERROR HANDLING.

```

try:
    file = open(FILE_NAME, "w")
    json.dump(students, file_obj)
    for student in students:
        #item IS THE VARIABLE YOU CHOSE FOR EACH INDIVIDUAL DICTIONARY, NOT student
        print(f"Student {student["first_name"]} {student["last_name"]} is enrolled in {student["course_name"]}!")
    file.close()
except Exception as e:
    print("Error saving data to file.")
    print(e)

```

FIGURE 8: MENU CHOICE 3 WITH INTENTIONAL ERROR TO TRIGGER ERROR HANDLING.

```

What would you like to do: 3
Error saving data to file.
name 'file_obj' is not defined

```

FIGURE 9: MENU CHOICE 3 OUTPUT WITH TRIGGERED ERROR HANDLING.

Summary

This was a difficult assignment because, while similar in this week's data type, CSVs and JSONs have quite a different approach to gain functionality. I personally prefer CSVs at the moment and can conceive of more opportunities where I might use CSVs, yet see the scale that JSONs could be valuable. In my workplace I have seen instances of both types of data but have seen key-type data more often within the code of the processes that I use daily. With that, I believe that proficiency using JSONs is must.