Steve Freedman

2/25/2026

IT FDN 110 GP Intro to Python Prog

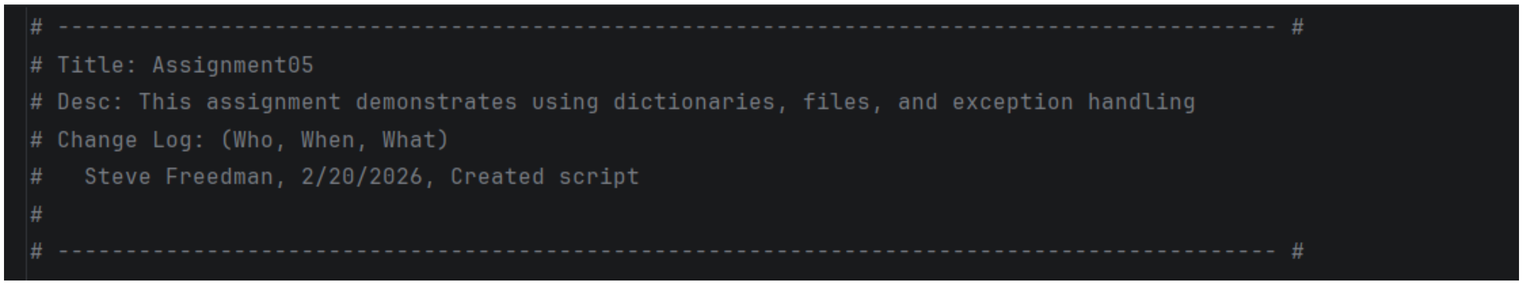
Assignment05

Introduction

The assignment: "Create a Python program that demonstrates using constants, variables, and print statements to display a message about a student's registration for a Python course. This program is very similar to Assignment04, but **It adds the use of data processing using dictionaries and exception handling."**

Topic 1

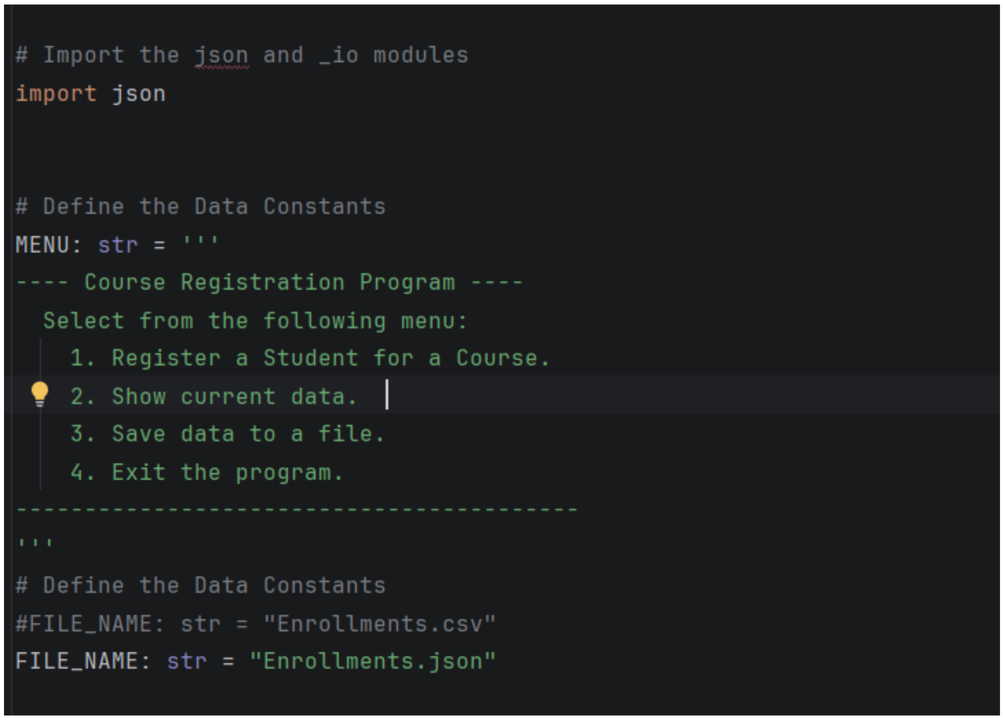
We start with the starter file, as usual:



```
# ------------------------------------------------------------------------------------------ #
# Title: Assignment05
# Desc: This assignment demonstrates using dictionaries, files, and exception handling
# Change Log: (Who, When, What)
#   Steve Freedman, 2/20/2026, Created script
#
# ------------------------------------------------------------------------------------------ #
```

Figure 1 Showing the starter file with appropriate edits.

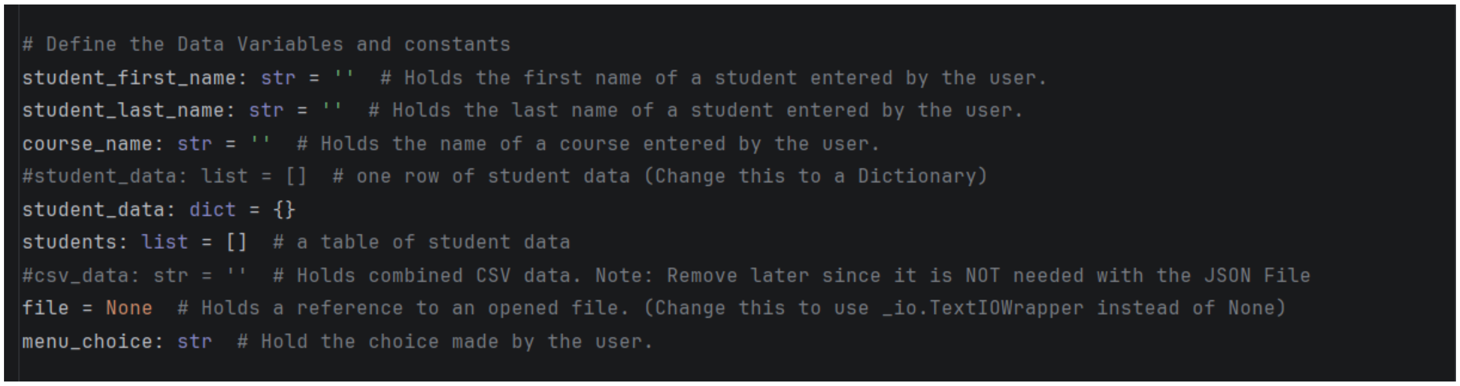
We are told to "Import the json module" and input the CONSTANTS:



```
# Import the json and _io modules
import json


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

Figure 2 Showing the import statement and the CONSTANTS being declared

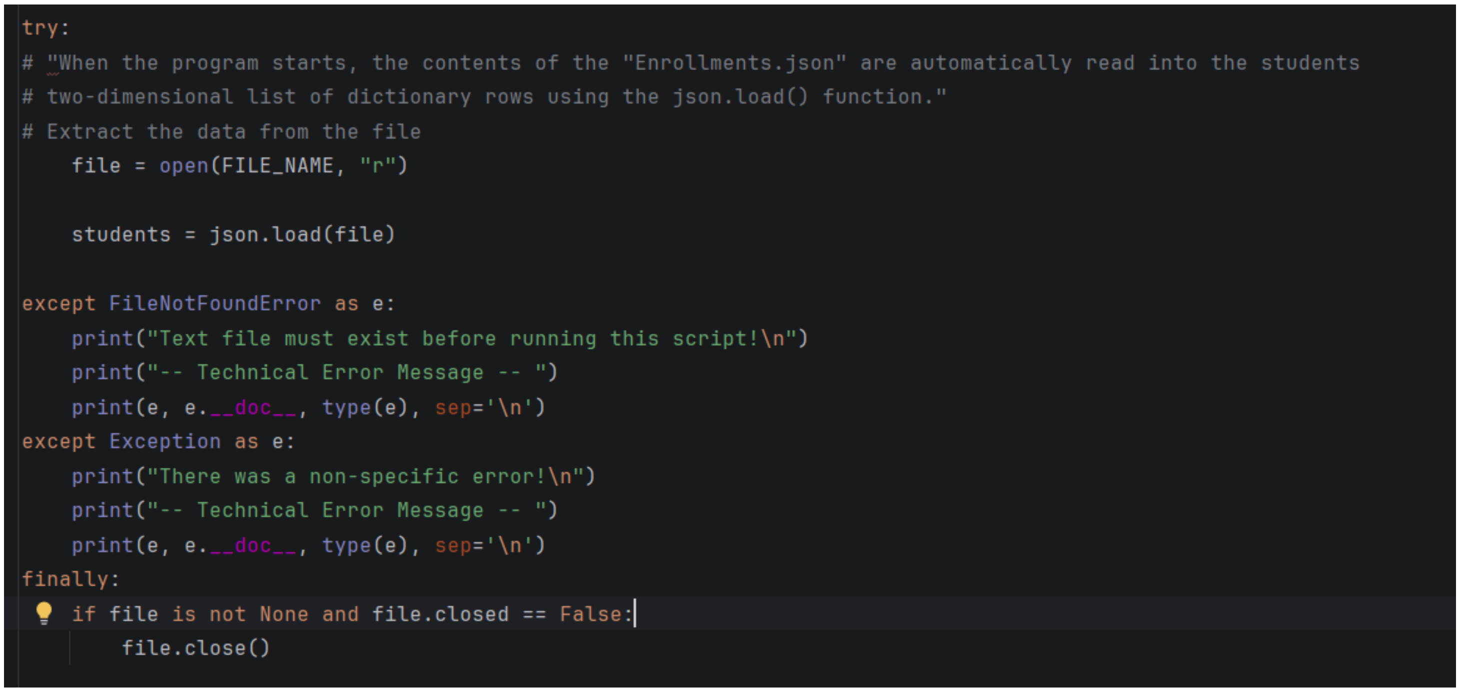
Then we declare the variables to be used:



```python
# 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: list = []  # one row of student data (Change this to a Dictionary)
student_data: dict = {}
students: list = []  # a table of student data
#csv_data: str = ''  # Holds combined CSV data. Note: Remove later since it is NOT needed with the JSON File
file = None  # Holds a reference to an opened file. (Change this to use _io.TextIOWrapper instead of None)
menu_choice: str  # Hold the choice made by the user.
```

Figure 3 Showing the variables being declared

Our next instruction was: “"When the program starts, the contents of the "Enrollments.json" are automatically read into the students two-dimensional list of dictionary rows using the json.load() function."



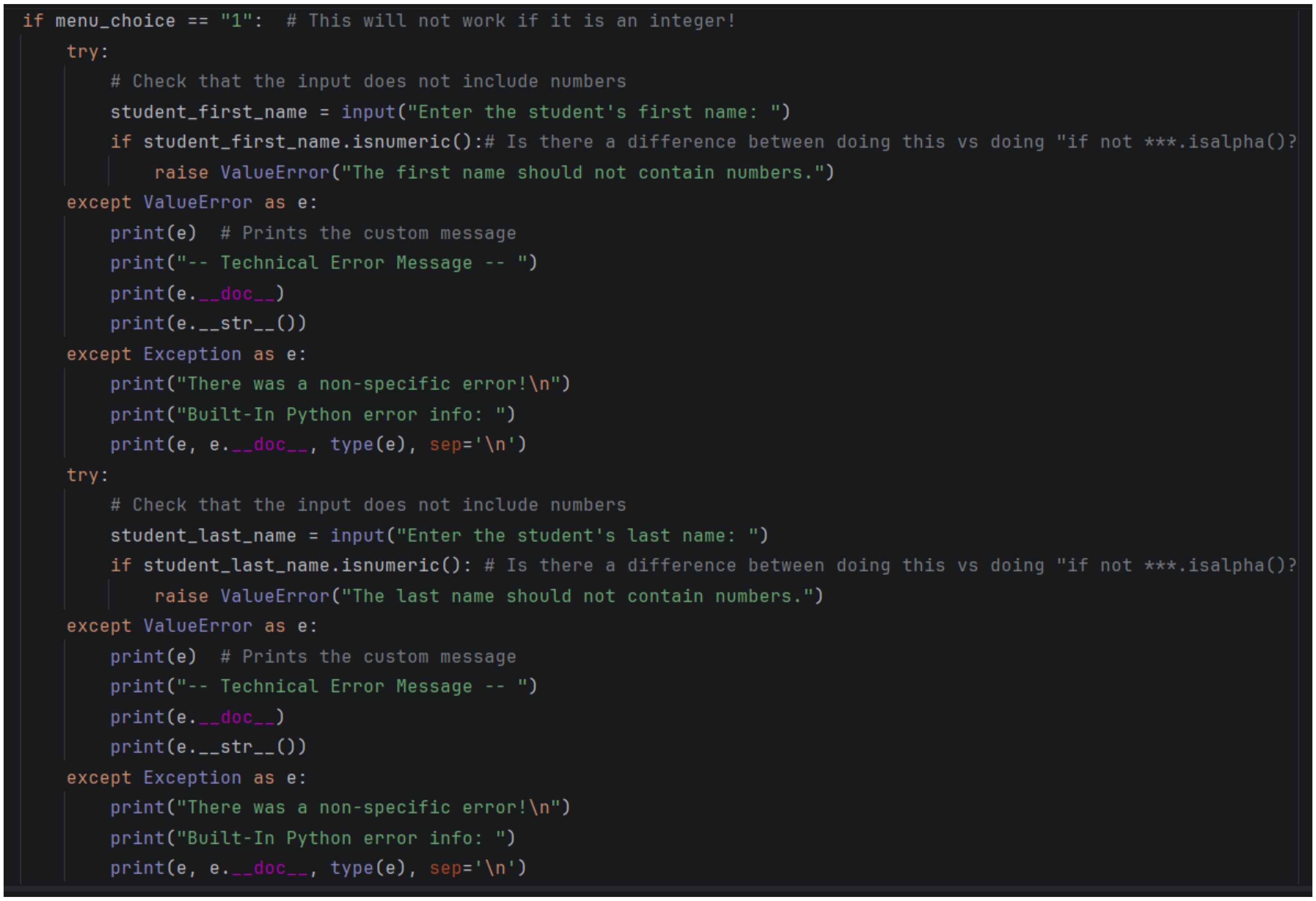
```python
try:
# "When the program starts, the contents of the "Enrollments.json" are automatically read into the students
# two-dimensional list of dictionary rows using the json.load() function."
# Extract the data from the file
    file = open(FILE_NAME, "r")

    students = json.load(file)

except FileNotFoundError as e:
    print("Text file must exist before running this script!\n")
    print("-- Technical Error Message -- ")
    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(e, e.__doc__, type(e), sep='\n')
finally:
    if file is not None and file.closed == False:
        file.close()
```

Figure 4 Showing the opening of the Enrollments.json file in read mode and the loading of the data into memory (via the “students” variable

Regarding I/O, we are told:

“On menu choice 1, the program prompts the user to enter the student's first name and last name, followed by the course name, using the input() function and stores the inputs in the respective variables.

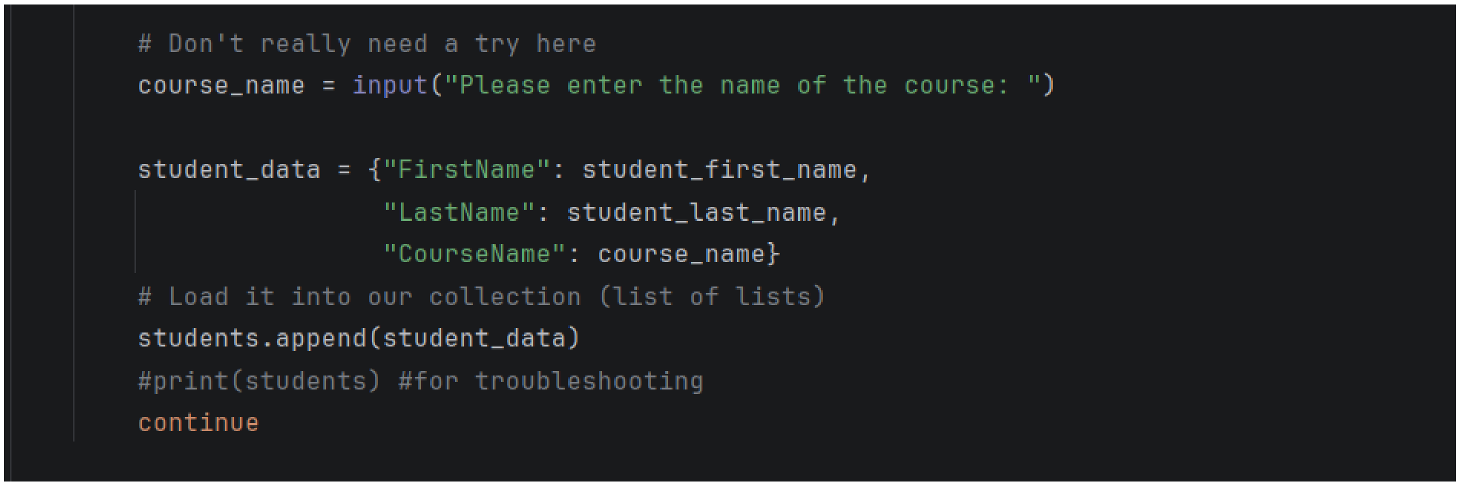
Data collected for menu choice 1 is added to a dictionary named student_data. Next, student_data is added to the **students** two-dimensional list of dictionaries rows**.”**



```python
if menu_choice == "1":  # This will not work if it is an integer!
    try:
        # Check that the input does not include numbers
        student_first_name = input("Enter the student's first name: ")
        if student_first_name.isnumeric():# Is there a difference between doing this vs doing "if not ***.isalpha()?
            raise ValueError("The first name should not contain numbers.")
    except ValueError as e:
        print(e)  # Prints the custom message
        print("-- Technical Error Message -- ")
        print(e.__doc__)
        print(e.__str__())
    except Exception as e:
        print("There was a non-specific error!\n")
        print("Built-In Python error info: ")
        print(e, e.__doc__, type(e), sep='\n')
    try:
        # Check that the input does not include numbers
        student_last_name = input("Enter the student's last name: ")
        if student_last_name.isnumeric(): # Is there a difference between doing this vs doing "if not ***.isalpha()?
            raise ValueError("The last name should not contain numbers.")
    except ValueError as e:
        print(e)  # Prints the custom message
        print("-- Technical Error Message -- ")
        print(e.__doc__)
        print(e.__str__())
    except Exception as e:
        print("There was a non-specific error!\n")
        print("Built-In Python error info: ")
        print(e, e.__doc__, type(e), sep='\n')
```

Figure 5 Showing the try/except blocks for Choice 1. The input() is used to get the user's input, stored in the appropriate variables

Figure 6 is a continuation of Figure 5, for Choice 1:



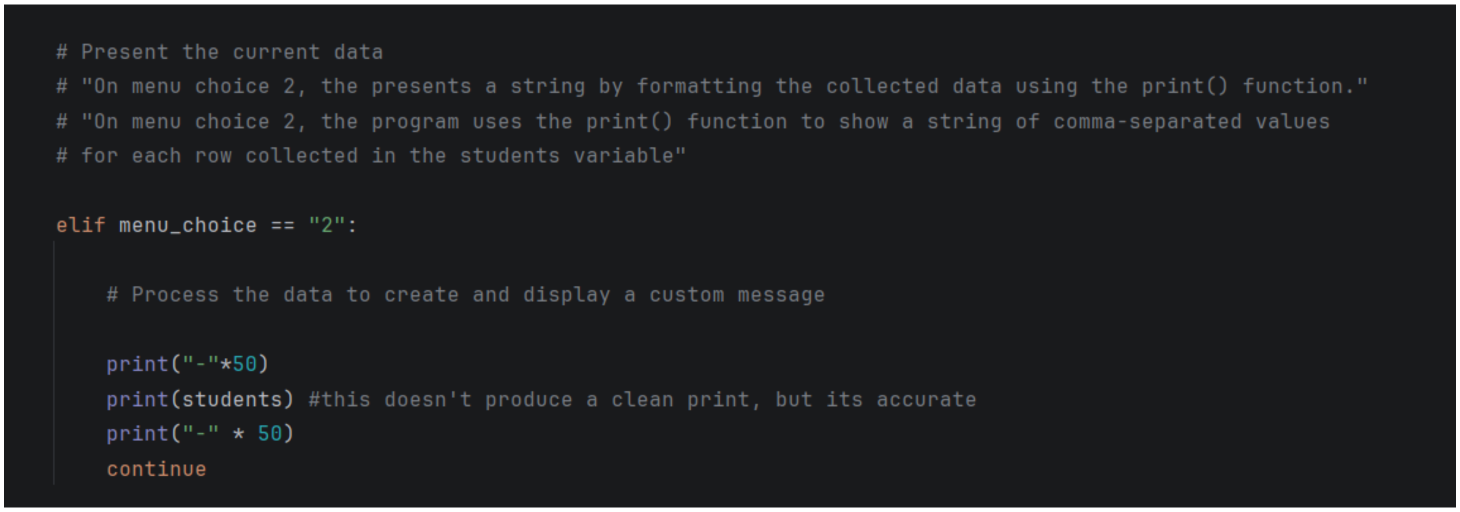
```python
    # Don't really need a try here
    course_name = input("Please enter the name of the course: ")

    student_data = {"FirstName": student_first_name,
                    "LastName": student_last_name,
                    "CourseName": course_name}
    # Load it into our collection (list of lists)
    students.append(student_data)
    #print(students) #for troubleshooting
    continue
```

Figure 6 Showing the creation of the student_data dictionary and the adding of the student_data to the students two-dimensional list of dictionaries rows. **<Author's Note: This is where the error we can't figure out occurs>**

For choice 2, we are told:

"On menu choice 2, the presents a string by formatting the collected data using the print() function.

On menu choice 2, the program uses the print() function to show a string of comma-separated values for each row collected in the **students** variable."

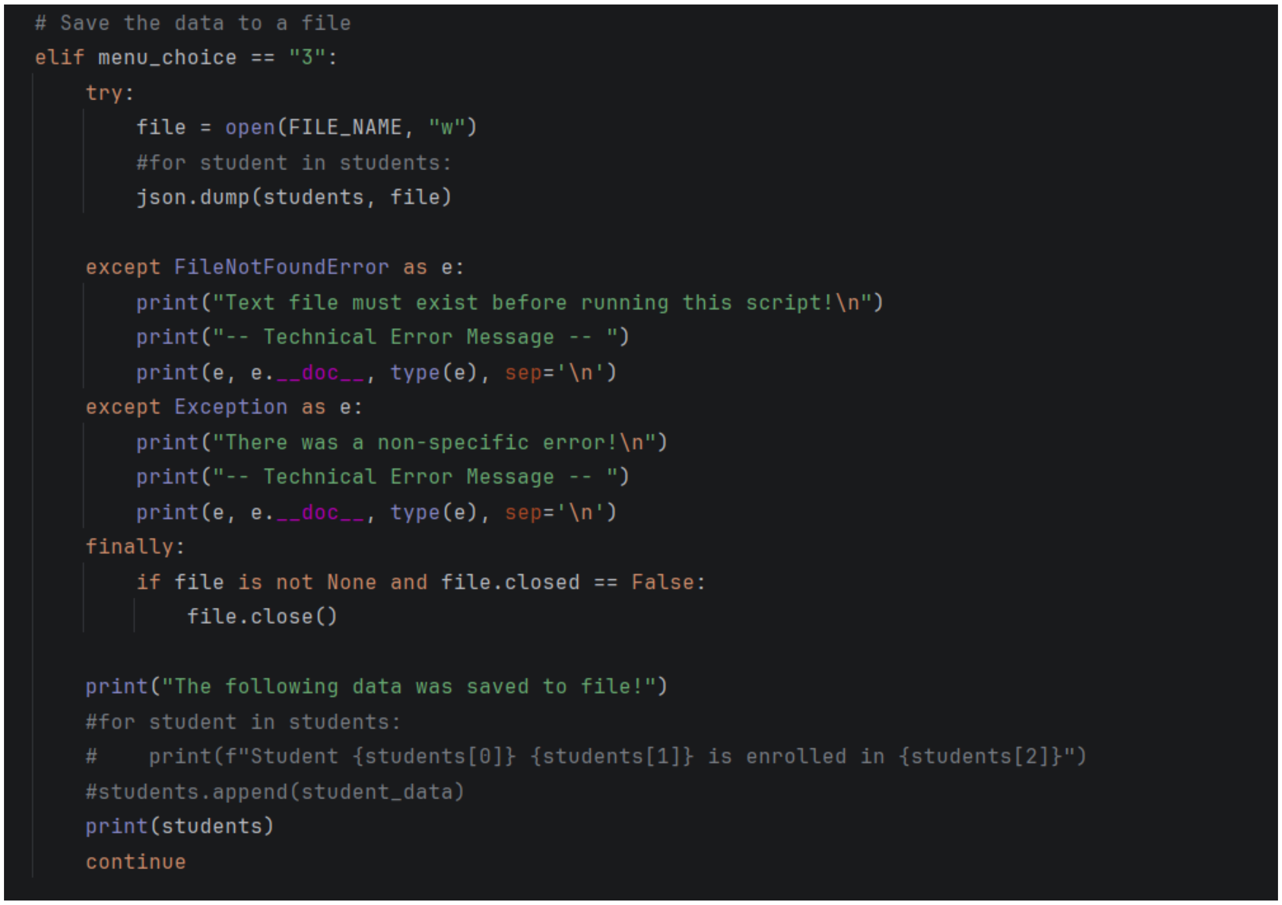


```
# Present the current data
# "On menu choice 2, the presents a string by formatting the collected data using the print() function."
# "On menu choice 2, the program uses the print() function to show a string of comma-separated values
# for each row collected in the students variable"

elif menu_choice == "2":

    # Process the data to create and display a custom message

    print("-"*50)
    print(students) #this doesn't produce a clean print, but its accurate
    print("-" * 50)
    continue
```

Figure 7 Showing the print()

Our next task is:

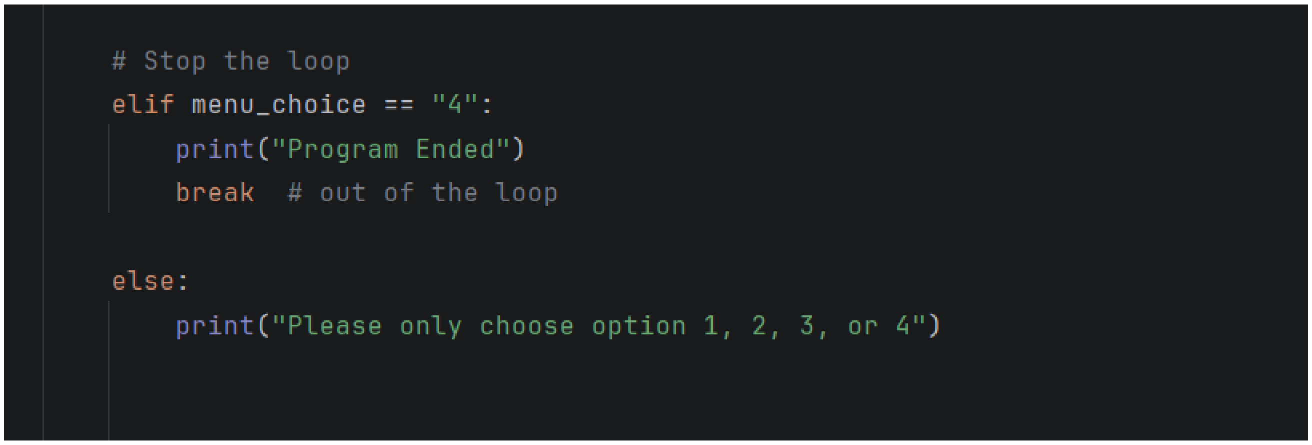
"On menu choice 3, the program opens a file named "Enrollments.json" in write mode using the open() function. It writes the contents of the **students** variable to the file using the json.dump() function. Next, the file is closed using the close() method. Finally, the program displays what was written to the file using the **students** variable."



```
# Save the data to a file
elif menu_choice == "3":
    try:
        file = open(FILE_NAME, "w")
        #for student in students:
        json.dump(students, file)

    except FileNotFoundError as e:
        print("Text file must exist before running this script!\n")
        print("-- Technical Error Message -- ")
        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(e, e.__doc__, type(e), sep='\n')
    finally:
        if file is not None and file.closed == False:
            file.close()

    print("The following data was saved to file!")
    #for student in students:
    #    print(f"Student {students[0]} {students[1]} is enrolled in {students[2]}")
    #students.append(student_data)
    print(students)
    continue
```

Figure 8 Showing Choice 3, the opening of the .json file, the use of json.dump(), the file being closed and the print of the students variable

The final task: "On menu choice 4, the program ends."



```
    # Stop the loop
    elif menu_choice == "4":
        print("Program Ended")
        break  # out of the loop

    else:
        print("Please only choose option 1, 2, 3, or 4")
```

Figure 9 Showing Choice 4, the printing of the message "Program Ended" and the ELSE block, in case the user tries something other than options 1, 2, 3 or 4.

Regarding Error Handling:

- The program provides structured error handling when the file is read into the list of dictionary rows. See Figure 4
- The program provides structured error handling when the user enters a first name. See Figure 5
- The program provides structured error handling when the user enters a last name. See Figure 5
- The program provides structured error handling when the dictionary rows are written to the file. See Figure 8

For testing:

- The program takes the user's input for a student's first, last name, and course name.
- The program displays the user's input for a student's first, last name, and course name.
- The program saves the user's input for a student's first, last name, and course name to a JSON file. (check this in a PyCharm or a simple text editor like Notepad or TextEdit.)
- The program allows users to enter multiple registrations (first name, last name, course name).
- The program allows users to display multiple registrations (first name, last name, course name).
- The program allows users to save multiple registrations to a file (first name, last name, course name).
- The program runs correctly in both PyCharm and from the console or terminal.

It should be noted here that testing was not completed.

Finally, for Source Control, this is the repository on GitHub:

sfreedman860/IntroToProg-Python-Mod05:

Summary

This assignment had us use Structured Error Handling for the first time and .json files for the first time. We learned that .json files are an industry standard for the collection and retrieval of data. Structured Error Handling gives us a chance to prevent an application crash and to handle errors in a more graceful fashion.