Medha Sinha

November 12, 2025

Fundamentals of Python Course

Assignment 05

Working with Dictionaries and Error Handling

Introduction

Module 05 of the Fundamentals of Python course covers how to use dictionaries for collections of data, how to handle expected errors with the ‘try-except’ blocks, and how to use the GitHub repository for code management.

The problem statement given in Assignment 05 is similar to Assignment 03 and Assignment04. The task is to create a program that presents a menu of choices for registering a student. Based on the user’s selection, the program can request the student’s name and course name, output all students registered, add the new student’s information in an external file, or end the program. The program must also be able to register multiple students, and save all of the student data in the file. However, in this assignment, the operations of the program must be completed using dictionaries, JSON files, and ‘try-except’ blocks to handle expected errors. Finally, the program must run from PyCharm and from the computer’s command shell.

This report documents the design concept, code, and testing of the Assignment 05 program.

Design

Concept

To solve the problem statement defined in Assignment 04, the program first opens the input JSON file with data formatted with dictionary keys and reads the contents into a list of dictionaries that can be indexed. The code is captured in a ‘try-except’ block to handle errors like non-existent files, or files with invalid data.

Next, the program presents a menu of choices with a corresponding number and requests the user to input their choice.

The program offers the following choices in the menu:

1. Register a Student for a Course
2. Show current Data
3. Save Data to a File

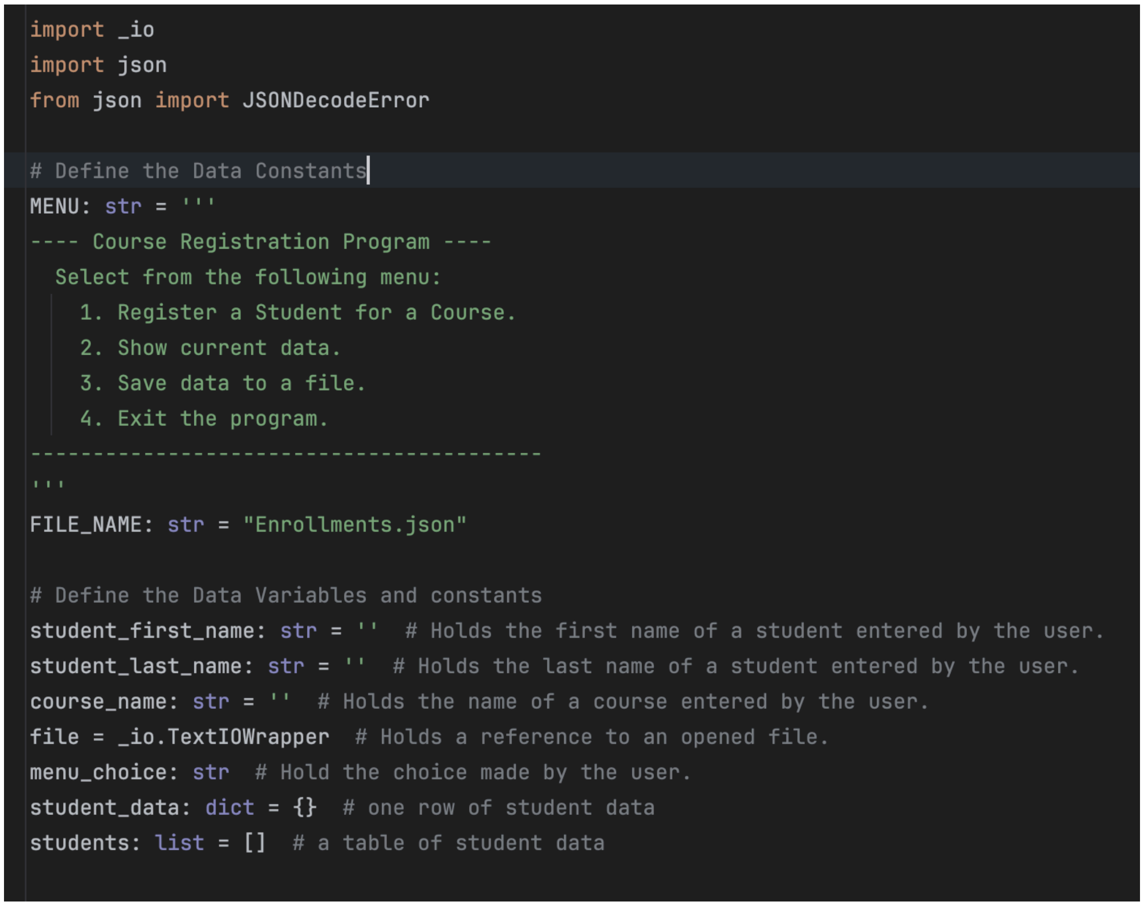
4. Exit the Program

The program uses a structure of 'if' and 'elif' statements to take actions corresponding to the user’s choice, all of which are nested within a while-loop that keeps the program running until the user selects Choice 4 to quit the program. Choice 1 and Choice 3 also use the ‘try-except’ blocks to verify user inputs are valid and the file opened to save the newly received data is valid.

Program Code

The program for Assignment 05 is written in one Python script file saved as Assignment05.py.

This program intends to store the collection of data as a dictionary. Therefore, the program expects to read and write to JSON files, which store data formatted for dictionaries in Python. To do so, the script begins with importing the json module. The program initializes various constants and variables, similar to the previous assignments, to collect input data from the user, store saved data in a list of dictionaries called ‘students’, and manage reading and writing to files (see Figure 1).



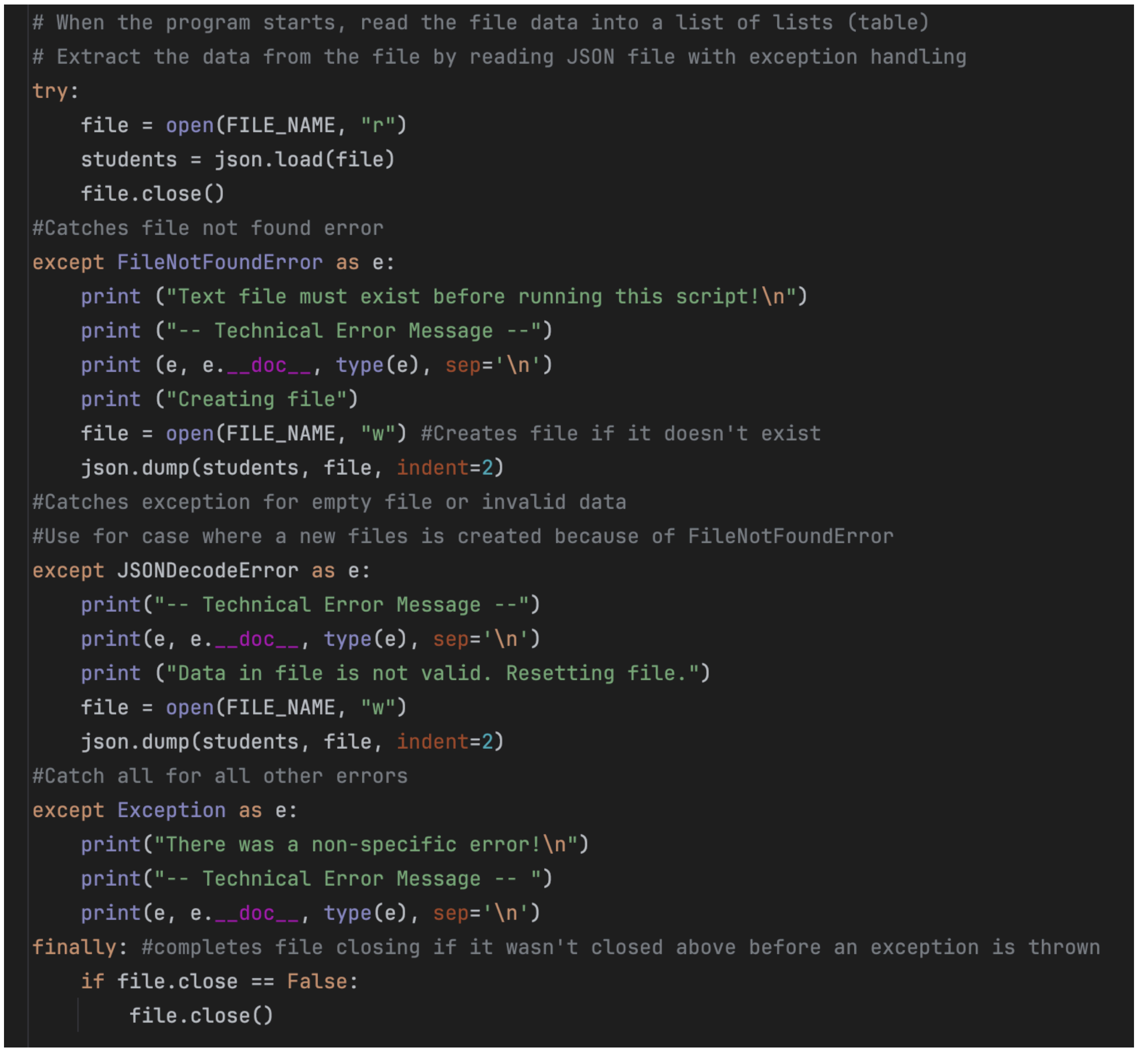
```python
import _io
import json
from json import JSONDecodeError

# 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 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 = _io.TextIOWrapper  # 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 1: Constant and Variable Definitions

After initializing the relevant constants and variables, the program reads the JSON file data into a list of dictionaries by calling the json.load() function. This is housed within a ‘try-except-finally’ block to anticipate a FileNotFoundError (see Figure 2).

If a FileNotFoundError occurs the file will create a new file with the assigned name. This can cause another error of invalid data, so another ‘except' block is added to capture this possible JSONDecodeError. This ‘except’ block then resets the data by writing the contents of the ‘students’ list into it. The last ‘except’ block captures all other exceptions. At the end, the ‘finally’ block closes the file if it is not already closed due to an exception. Due to this operation, it is important to initialize ‘file’ with _io.TextIOWrapper. If it is initialized as None or an empty string, the attempt to file.close() would throw an error.

The program imports the _io, json, and the JSONDecodeError modules to accomplish the tasks above.

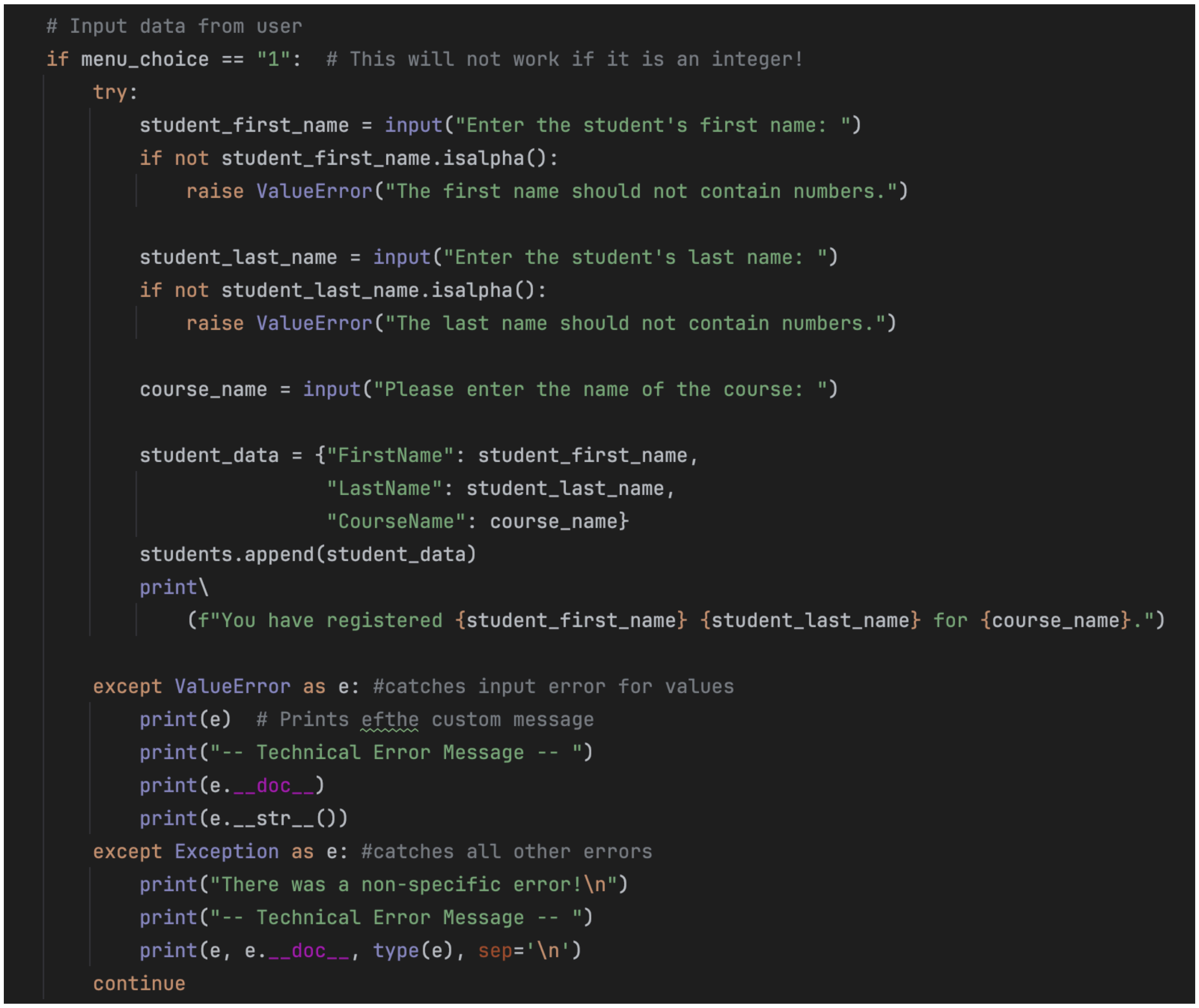


```
# When the program starts, read the file data into a list of lists (table)
# Extract the data from the file by reading JSON file with exception handling
try:
    file = open(FILE_NAME, "r")
    students = json.load(file)
    file.close()
#Catches file not found error
except FileNotFoundError as e:
    print ("Text file must exist before running this script!\n")
    print ("-- Technical Error Message --")
    print (e, e.__doc__, type(e), sep='\n')
    print ("Creating file")
    file = open(FILE_NAME, "w") #Creates file if it doesn't exist
    json.dump(students, file, indent=2)
#Catches exception for empty file or invalid data
#Use for case where a new files is created because of FileNotFoundError
except JSONDecodeError as e:
    print("-- Technical Error Message --")
    print(e, e.__doc__, type(e), sep='\n')
    print ("Data in file is not valid. Resetting file.")
    file = open(FILE_NAME, "w")
    json.dump(students, file, indent=2)
#Catch all for all other errors
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: #completes file closing if it wasn't closed above before an exception is thrown
    if file.close == False:
        file.close()
```

Figure 2: Reading data from the JSON file with Error Handling

Then, the program continues into a while-loop that will continue until the break function is called. Within the loop, the program prints the menu and prompts the user for a choice, so that this will be repeated after each selection is completed.

For menu choice 1, the program requests the student's first name, student's last name and the course name as input from the user. Then all the variables are stored in memory in the student_data dictionary, which is then appended to the master list of 'students'. This set of code is also housed in a 'try-except' block to capture instances where a user may enter invalid names into the program, like spaces or numbers for the name (see Figure 3).



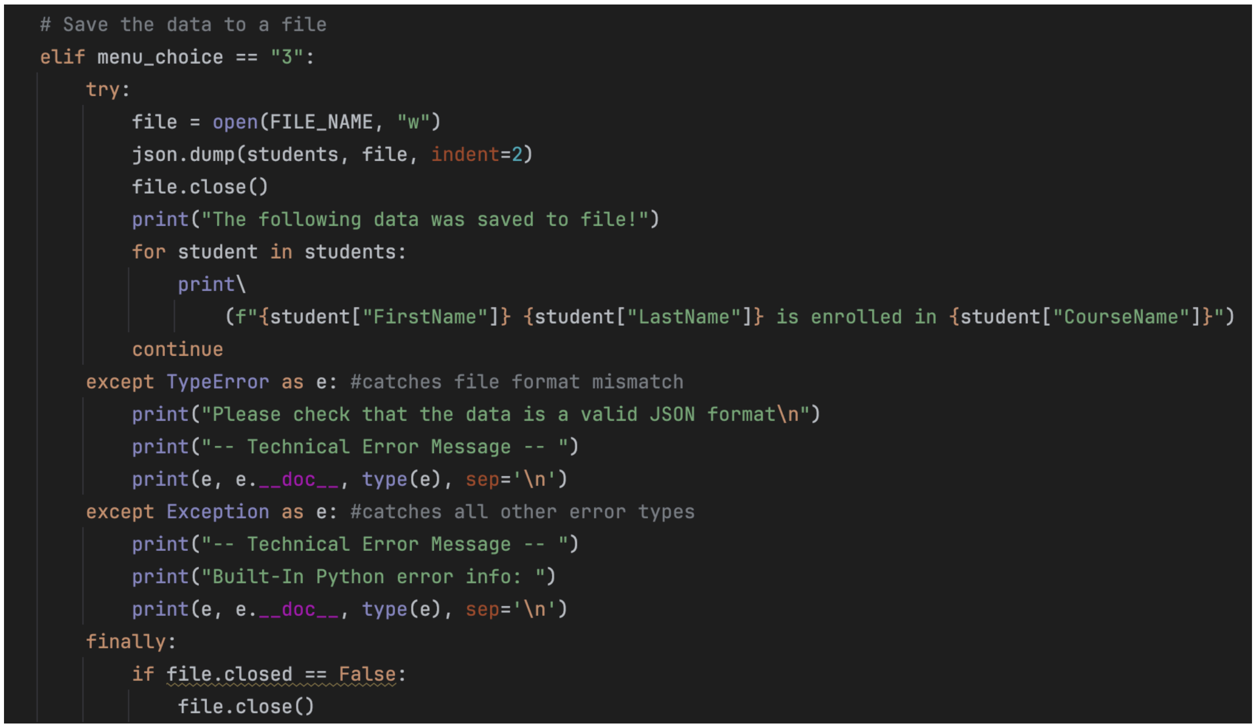
```
# Input data from user
if menu_choice == "1":  # This will not work if it is an integer!
    try:
        student_first_name = input("Enter the student's first name: ")
        if not student_first_name.isalpha():
            raise ValueError("The first name should not contain numbers.")

        student_last_name = input("Enter the student's last name: ")
        if not student_last_name.isalpha():
            raise ValueError("The last name should not contain numbers.")

        course_name = input("Please enter the name of the course: ")

        student_data = {"FirstName": student_first_name,
                        "LastName": student_last_name,
                        "CourseName": course_name}
        students.append(student_data)
        print\
            (f"You have registered {student_first_name} {student_last_name} for {course_name}.")

    except ValueError as e: #catches input error for values
        print(e)  # Prints efthe custom message
        print("-- Technical Error Message -- ")
        print(e.__doc__)
        print(e.__str__())
    except Exception as e: #catches all other errors
        print("There was a non-specific error!\n")
        print("-- Technical Error Message -- ")
        print(e, e.__doc__, type(e), sep='\n')
    continue
```

Figure 3: Error handling for user inputs in Choice 1.

For menu choice 2, the program uses a for-loop to iterate through 'students', the list of dictionaries stored in memory, and prints each entry.

If the menu choice is 3, the program opens the file object, in the ‘write’ mode, writes the contents of ‘students’ using the json.dump() function, closes the file, and then prints all the data that was saved to the file. This choice ensures that all the data is saved from memory to the RAM for persistence.

This section of code also uses the ‘try-except-finally’ structure to handle possible errors with writing to a file. A possible error this captures is the TypeError, where a file is not a JSON file. This also captures all other exceptions. The ‘finally’ block ensures that the file is closed even if an exception occurs (see Figure 4).



```python
# Save the data to a file
elif menu_choice == "3":
    try:
        file = open(FILE_NAME, "w")
        json.dump(students, file, indent=2)
        file.close()
        print("The following data was saved to file!")
        for student in students:
            print\
                (f"{student["FirstName"]} {student["LastName"]} is enrolled in {student["CourseName"]}")
        continue
    except TypeError as e: #catches file format mismatch
        print("Please check that the data is a valid JSON format\n")
        print("-- Technical Error Message -- ")
        print(e, e.__doc__, type(e), sep='\n')
    except Exception as e: #catches all other error types
        print("-- Technical Error Message -- ")
        print("Built-In Python error info: ")
        print(e, e.__doc__, type(e), sep='\n')
    finally:
        if file.closed == False:
            file.close()
```

Figure 4: Error handling for writing data into JSON file.

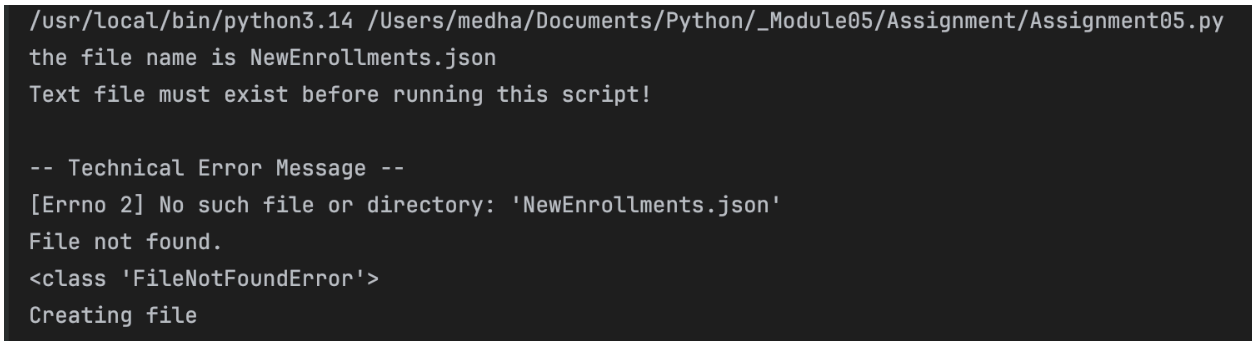
For menu choice 4, the program breaks out of the while-loop. Once outside of the while-loop, the program prints a statement acknowledging the program has ended.

Finally, the program has an 'else' statement to capture the condition of an invalid choice, which is any other character or string other than 1, 2, 3 or 4. This condition prints a statement that the choice was invalid. It then brings the program back to the beginning of the while-loop which displays the menu and prompts the user to make a choice again.

Testing

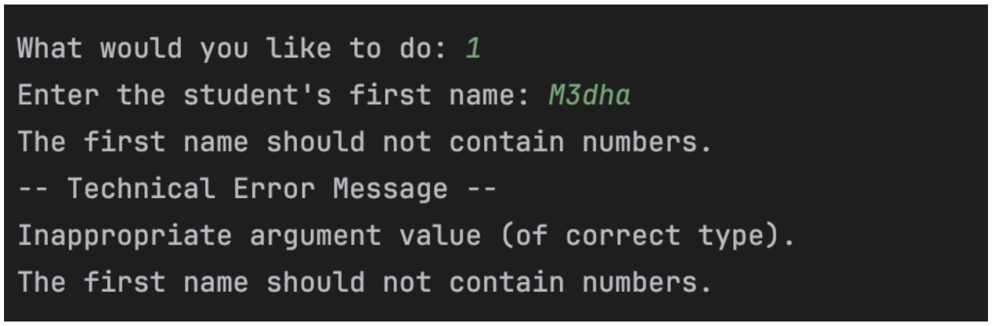
The program was tested by running it with the PyCharm shell and running it from a Mac operating system (OS) Terminal shell. The JSON file was also checked to confirm the file operations worked as intended.

For this assignment, it was important to test that the expected errors were handled with the 'try-except' blocks. As part of testing, the file names were changed to check if the FileNotFoundError was handled and if a new file was created. This also checked if the resulting JSON file was reset to hold valid data. The handling of non-alphanumeric inputs for new student names was also confirmed for Choice 1 (see Figures 5 & 6).



```
/usr/local/bin/python3.14 /Users/medha/Documents/Python/_Module05/Assignment/Assignment05.py
the file name is NewEnrollments.json
Text file must exist before running this script!

-- Technical Error Message --
[Errno 2] No such file or directory: 'NewEnrollments.json'
File not found.
<class 'FileNotFoundError'>
Creating file
```

Figure 5: Testing FileNotFoundError and JSONDecodeError



```
What would you like to do: 1
Enter the student's first name: M3dha
The first name should not contain numbers.
-- Technical Error Message --
Inappropriate argument value (of correct type).
The first name should not contain numbers.
```

Figure 6: Testing non-alphanumeric error handling for Choice 1

Summary

Dictionaries are an important data type that can store large sets of data and allow for easy recall. This assignment explores using dictionaries and JSON files to appropriately read data from a file, gather new data from the user, and write the data to a JSON file. Another important aspect of this assignment was to practice handling common errors like invalid input, non-existent files, or files with invalid data.