Geoff DuBuque

July 29, 2024

IT FDN 110 A Su 24

Assignment 05

https://github.com/gdubuque/IntroToProg-Python-Mod05.git

Assignment 05 – Advanced Collections and Error Handling

Introduction

This module's programming assignment continues to build upon all that we learned in the previous assignments to create a program to collect user data and display messages about registering a student for a class. This assignment introduces using a new Python collection type, dictionaries, and a new data file type, .json. This program also introduces exception handling to handle program errors that can come up in runtime and display built-in and custom messages about the errors. The following sections describe the steps I used to complete the programming task of this assignment.

Creating the Python script

Script Header, Constants and Variables

The beginning of the script starts with our standard script header and the program's constants and variables, with the names, data types and initial values given to us in the assignments' starter Python file. I removed the *json_data* variable to simplify and optimize the code because it is not needed to perform the programs tasks. I also added my initials in parentheses, (GD), at the end of comments I added to keep track of what I changes I made to the script. See **Figure 1** on the next page.

To work with .json files to read and write data to them in the JSON (JavaScript Object Notation) format, we start the script by importing the *json* module in Python. The *json* module includes functions to make it easy to work with .json files. The JSON format is an easy-to-read text-based format to store data that is similar to Python's syntax for dictionaries and other collections.

```python
# ------------------------------------------------------------------------------------------ #
# Title: Assignment05
# Desc: This assignment demonstrates using dictionaries, files, and exception handling
# Change Log: (Who, When, What)
#   R.Root, 2030/01/01, Created Script
#   G.DuBuque, 2030/07/29, Updated script for Assignment 5 to use json files, dictionaries
#                          and exception handling. Removed unused variable json_data: str
# ------------------------------------------------------------------------------------------ #

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.
----------------------------------------
'''

FILE_NAME: str = "Enrollments.json"

# Define the Data Variables
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 = {}         # Holds data for one student in a dictionary (GD)
students: list = []             # List to hold all student dictionaries (json format) (GD)
```

Figure 1: Script Header, Constants and Variables

Initial Data Processing and File Error Handling

Because the intent of the program is to add data to an existing data file, the program needs to read the existing data and save it in the program so it can be written back to the file with the new data. We collect the data by saving it to the list of dictionaries. Each item in the list is a dictionary of data for one student, with the following keys and values:

{"FirstName": student_first_name, "LastName": student_last_name, "CourseName": course_name}

The program starts by opening the .json file and using the *json* module *load()* method to extract the data into a list of dictionaries and saves is to the *students* variable. The program also starts using structured error handling when working with the file using the *try, except, finally* syntax. The error handling syntax and functions are explained in the module labs and are re-used and modified for this assignment. We first try to open, read and close the file in the *try:* block, and if

any errors occur one of *except:* code blocks will run. We first check for a *FileNotFoundError* and then print a custom message and the built-in error messages, then check if there is any other kind of error and print a custom message and the built-in error messages. Then the *finally:* code block always runs where we check and close the file if it is not already closed in the case an error occurred after the file was opened. See **Figure 2** below.

```
# When the program starts read the contents of the json file into the students list (GD)
# Each dictionary has the following keys: "FirstName", "LastName", "CourseName" (GD)
# Extract the data from the file
try:
    file = open(FILE_NAME, "r")
    students = json.load(file)
    file.close()
except FileNotFoundError as e:  # Check if file does not exist (GD)
    print(f"Text file {FILE_NAME} must exist before running this script!\n")  # Added FILE_NAME (GD)
    print("-- Technical Error Message -- ")
    print(e, e.__doc__, type(e), sep='\n')
except Exception as e:          # Check for other errors (GD)
    print("There was a non-specific error!\n")
    print("-- Technical Error Message -- ")
    print(e, e.__doc__, type(e), sep='\n')
finally:  # Close file if still opened (GD)
    if not file.closed:
        file.close()
```

Figure 2: Initial Data Processing and File Error Handling

Program Menu and While Loop

This program presents the user with a menu of choices and then asks for their input the same way as in the previous assignment. This is done by using a *while loop*, displaying the *MENU* constant and then prompting the user for their input and saving it to the *menu_choice* variable. See **Figure 3** below.

```
# Present and Process the data
while True:

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

Figure 3: Start of Program Loop

Menu Choice 1 and Input Error Handling

The program then runs certain tasks based on the user's choice, the same as the previous assignment, by using a series of *if, else* statements comparing the user's input. For choice 1 we prompt the user for the student's first name, last name, and course name and save the values to their respective variables. This program adds structured error handling to the user's input by

checking to make sure the name inputs do not contain numbers using the *isalpha()* method and then raising a *ValueError* with a custom message if it does.

We put the student data variables as *values* in a dictionary with the given *keys* saved to the *student_data* variable. Because we want to save the new data along with the existing data in the file, we *append* the *student_data* to the existing data collection in the *students* list. We then print a formatted string displaying the data that was entered.

If a *ValueError* was raised we then print the custom and built-in error messages, and then check for any other kind of errors. See **Figure 4** below.

```
    # Input user data
    if menu_choice == "1":
        try:
            student_first_name = input("Enter the student's first name: ")
            # Check to make sure name does not include numbers (GD)
            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: ")
            # Check to make sure name does not include numbers (GD)
            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: ")

            # Add student data to dictionary, then add dictionary to students list (GD)
            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:
            print(e)  # Print invalid name entry message (GD)
            print("-- Technical Error Message -- ")
            print(e.__doc__)
            print(e.__str__())
        except Exception as e:  # Print other error message (GD)
            print("There was a non-specific error!\n")
            print("-- Technical Error Message -- ")
            print(e, e.__doc__, type(e), sep='\n')
        continue
```

Figure 4: Menu Choice 1 and Input Error Handling

Menu Choice 2

For choice 2 the program displays all the data in the *students* list as formatted strings. Because we are using a dictionary instead of a list to store each student's data we need to access the student data with their *keys* instead of indexes. To display the data, I use a *for loop* to iterate through each dictionary in the *students* collection and print the data for each student using an *f-string* to format the data into a sentence. See **Figure 5** on the next page.

```
    # Present the current data
    elif menu_choice == "2":
        # Process the data to create and display a custom message
        # Updated with student data in dictionary format (GD)
        print("-" * 50)
        for student in students:
            print(f"Student {student["FirstName"]} {student["LastName"]} "
                  f"is enrolled in {student["CourseName"]}")
        print("-" * 50)
        continue
```

Figure 5: Menu Choice 2

Menu Choice 3 and File Error Handling

For choice 3 the program saves the data to the Enrollments.json file and then displays the data in the file. We also provide structured error handling when using the file by checking for a *TypeError,* in case the data or file in not in the JSON format, checking for any other type of error, and finally closing the file if it's still opened. The data is saved to the file in the JSON format by using the *dump()* method to save the *students* list to the file. I chose to display the data as it would be seen in the .json file, as dictionaries, and used use a *for loop* to iterate over every dictionary in the *students* list to print one dictionary per line to make it easier to read. See **Figure 6** below.

```
    # Save the data to a file
    elif menu_choice == "3":
        try:
            file = open(FILE_NAME, "w")
            json.dump(students, file)
            file.close()
            print("The following data was saved to file!")
            # Changed print format to show data as is (dictionaries in json format) (GD)
            for student in students:
                print(student)
            continue
        # Check to make sure student data is in json format (GD)
        except TypeError as e:
            print("Please check that the data is a valid JSON format\n")
            print("-- Technical Error Message -- ")
            print(e, e.__doc__, type(e), sep='\n')
        # Check for other errors (GD)
        except Exception as e:
            print("-- Technical Error Message -- ")
            print("Built-In Python error info: ")
            print(e, e.__doc__, type(e), sep='\n')
        finally:  # Close file if still opened (GD)
            if not file.closed:
                file.close()
```

Figure 6: Menu Choice 3 and File Error Handling

Menu Choice 4 and Invalid Inputs

For choice 4 the program ends by using the *break* command to exit the *while loop*. For any other choice the (the last *else* statement) the user is prompted to only choose 1, 2, 3 or 4, and the loop starts over again. After choice 4 the loop ends, and the last print statement displays "Program Ended" before the program ends. See **Figure 7** below.

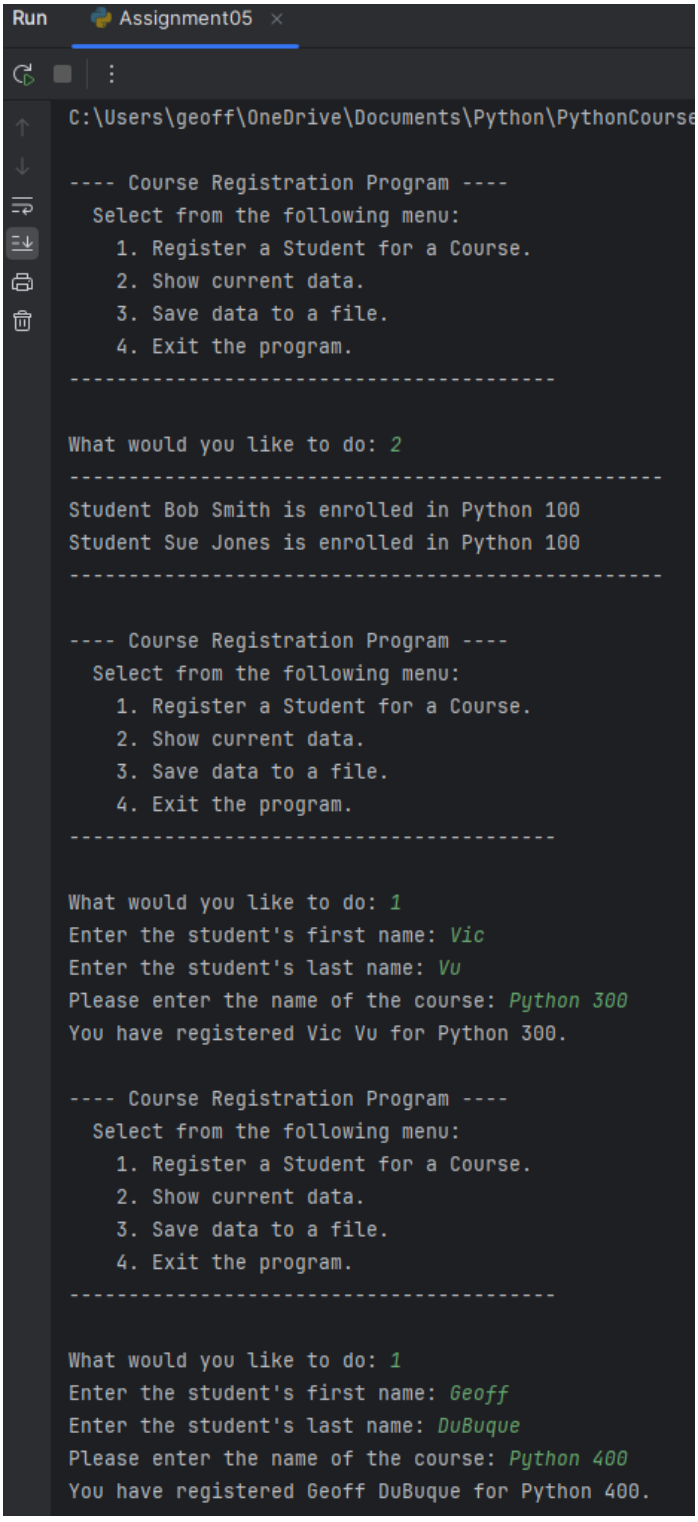
```
128         # Stop the loop
129         elif menu_choice == "4":
130             break  # out of the loop
131         else:
132             print("Please only choose option 1, 2, 3 or 4")     # Added option 4 (GD)
133
134     print("Program Ended")
```

Figure 7: End of Program

Testing the Program

The program first needs an existing Enrollments.json file with some data in it. The Enrollments.json file given in the module had one of the keys as "Email" so I corrected it to "LastName" so it would work with this program, as this program assumes all the keys will be correct. I tested the program by first choosing menu choice 2 to display the existing data in the file and make sure it is displayed in the correct format. Then I use menu choice 1 a couple times to add some new data. I used choice 2 again to make sure the new data was added. I then use choice 3 to save the data and make sure it is displayed correctly. I ended the program with choice 4.

I tested the error handling by changing the file name in the FILE_NAME variable and entering numbers in the first and last name inputs to make sure the corresponding errors are displayed. The program is run in both PyCharm and Windows PowerShell. The results of the program running correctly are shown on the next pages.



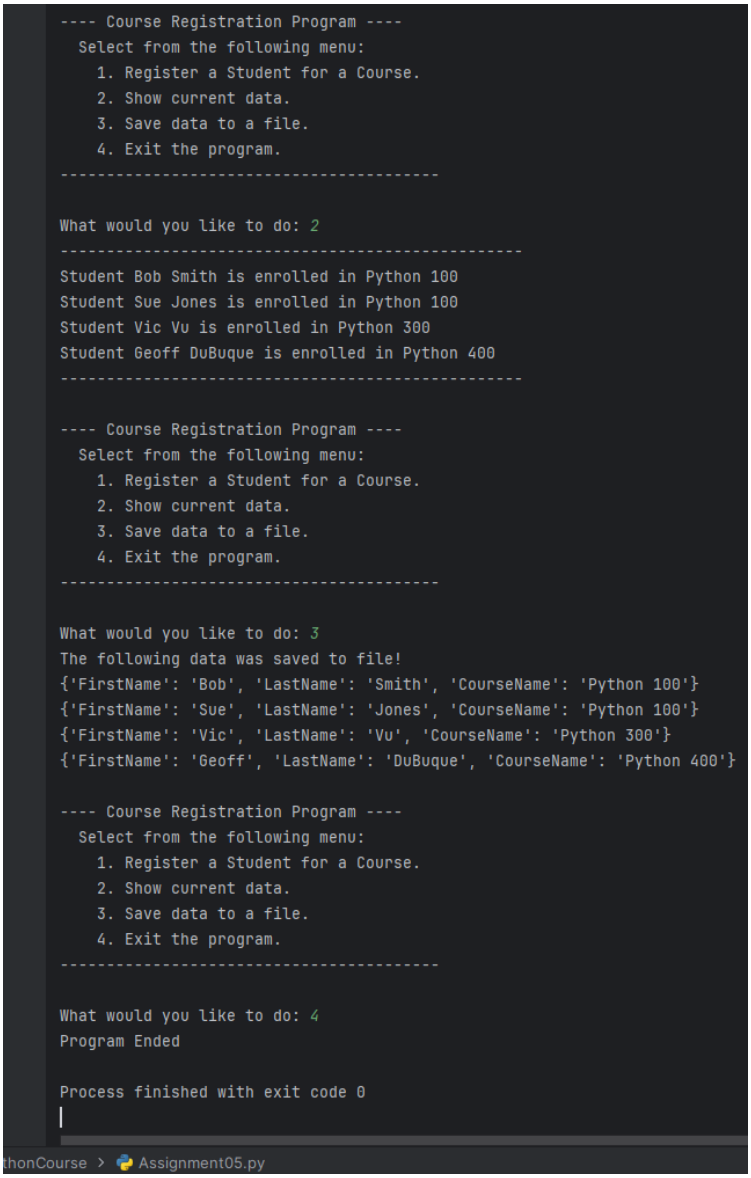
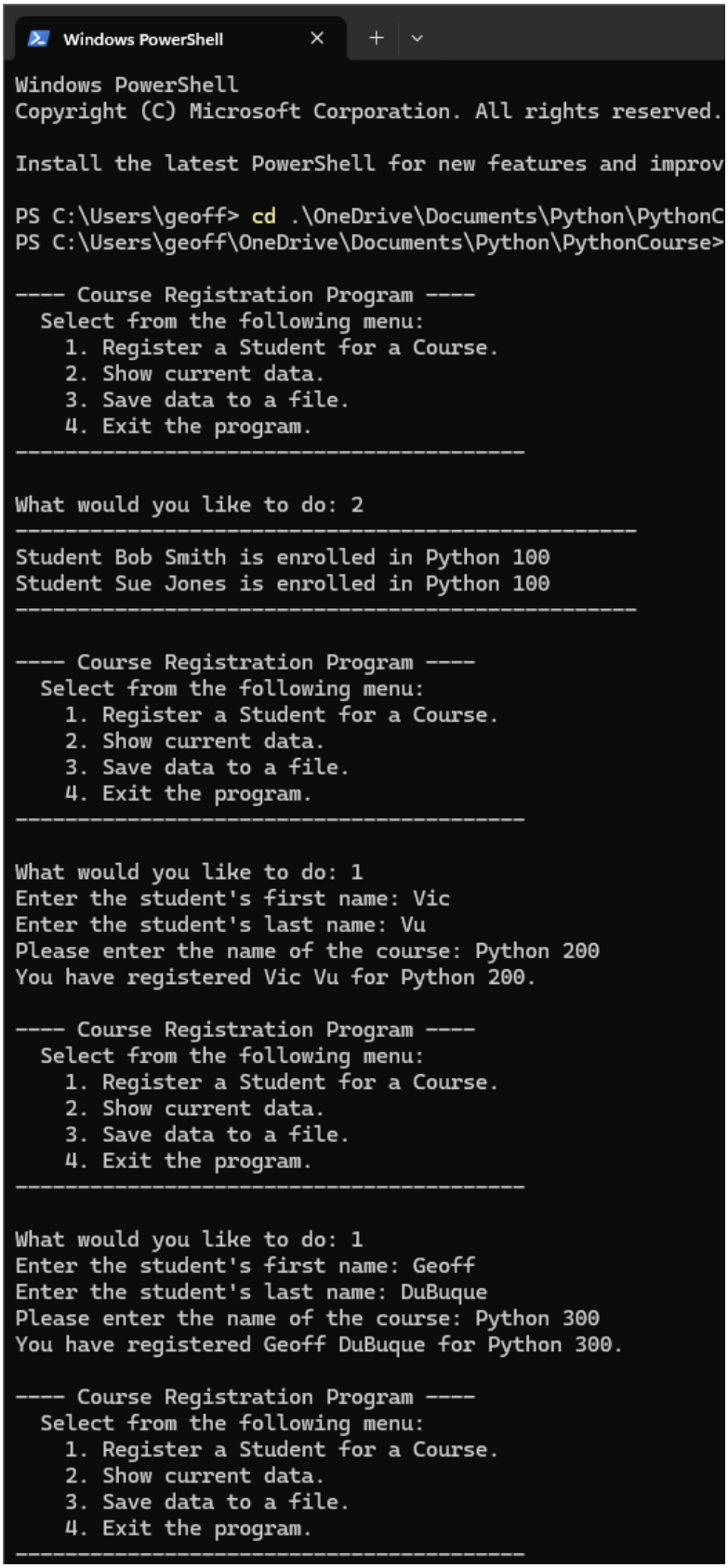


Figure 8: Program Ran in PyCharm



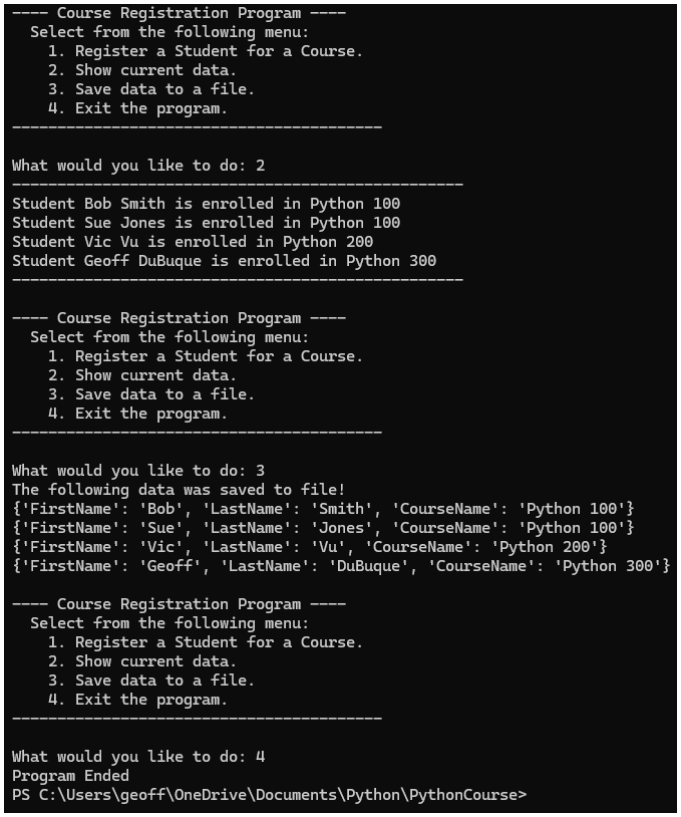
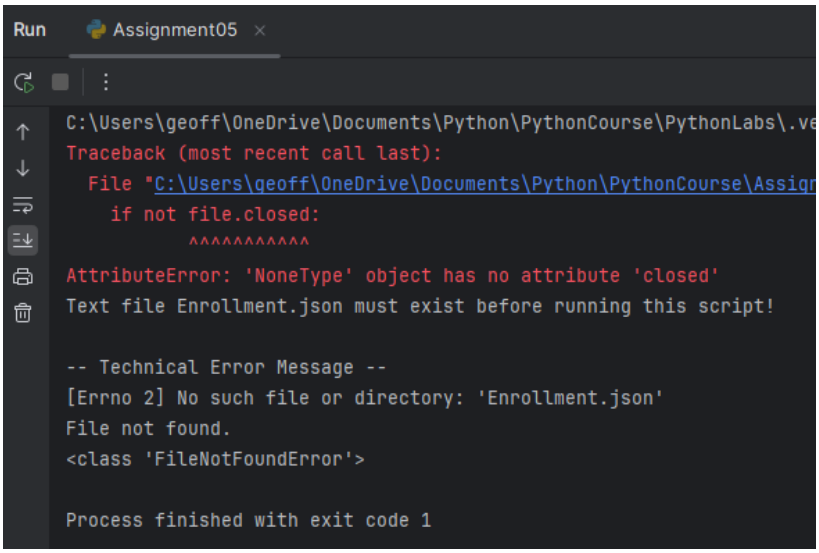


Figure 9: Program Ran in PowerShell



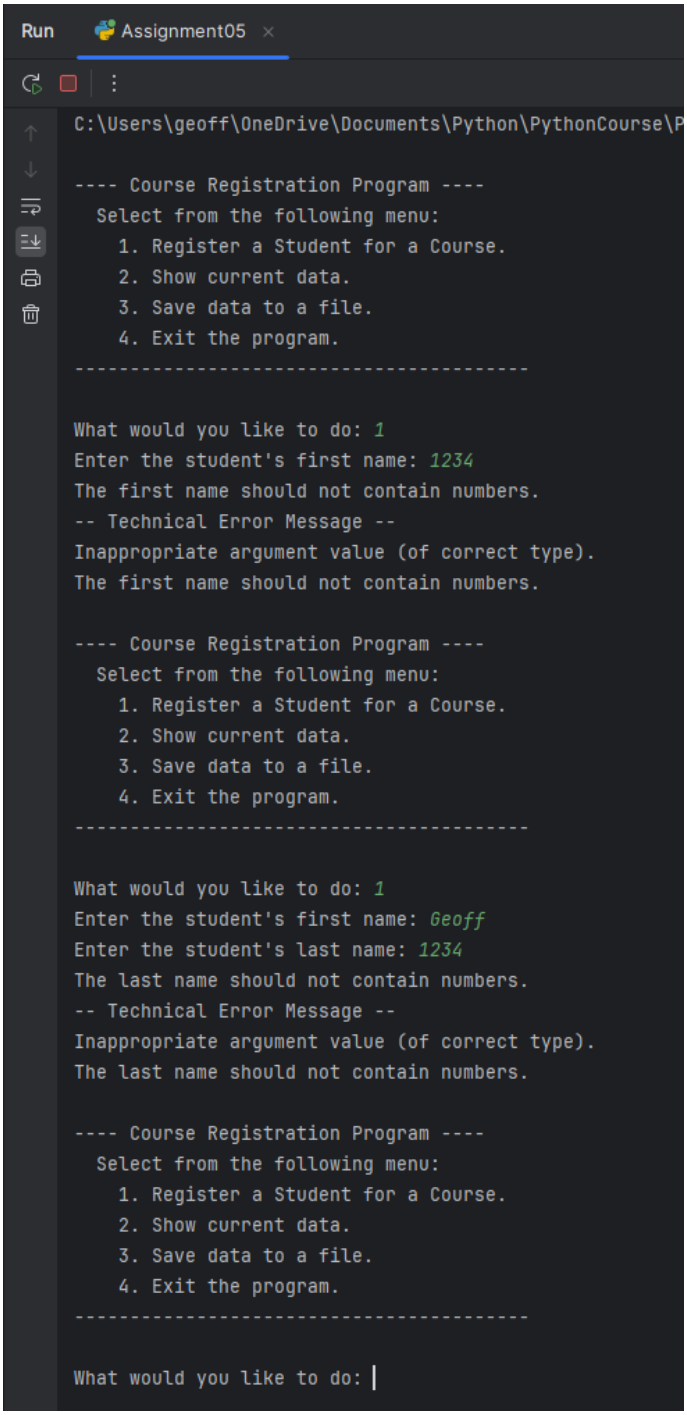


Figure 10: Error Handling in PyCharm

Summary

This programming assignment continued to use the many programming skills we used in the previous assignments and introduced 2 new ways to work with data, Python dictionaries and .json files and the JSON format, and a new programming concept to add structured error handling to our script. Dictionaries are an easy-to-read way to collect data that uses *keys*

instead of indexes to retrieve values, with a syntax that is compatible with the JSON format and files along with other Python collection types. The Python *json* module makes it extremely easy to read and write data from .json files using Python dictionaries and lists. Adding structured error handling to our program using the *try-except-finally* code blocks allows us to check for program and user errors, display custom and built-in error messages, and could allow our program to continue to run.