Sonali R. Mishra

May 14 2025

Foundations of Programming: Python

Assignment for Module 5

https://github.com/codebeaker/IntroToProg-Python-Mod05

Assignment 5: Lists of Dictionaries, Error Handling, and GitHub

Introduction

Module 5 teaches us how to use lists of dictionaries rather than lists of lists for data storage, how to read and write to JSON files instead of CSV, how to use try/except blocks for error handling, and how to set up a repo on GitHub. In this assignment, we continue our gradual build-up of a program for registering students by 1) changing the list of lists to lists of dictionaries; 2) reading and writing data to/from a JSON file rather than CSV; 3) setting up try/except blocks for error handling for a few key parts of the code; and 4) posting our code on GitHub.

Lists of Dictionaries

In this module as in the past one, we store tabular data in a structure embedded in a list. This week, however, the interior structure is a dictionary rather than a list. A **dictionary** is a collection of paired data, in the form of a key and a value. The use of keys makes dictionaries a little more intuitive to work with: you can, for example, call up student FirstNames instead of list index [0]. However, the dictionary is still embedded in a list, and each row in the "table" is its own dictionary.

In this assignment we change over our lists of lists to lists of dictionaries. The changes needed to do this are relatively minor and involve switching out types of brackets and using keys rather than list indices to call up desired values.

Reading and writing to/from JSON

In this module we learned about reading and writing files to JSON documents rather than CSV files. **JSON** files are relatively human-readable documents that structure data in key-value pairs parallel to Python's dictionaries. Python allows relatively straightforward reading and writing of dictionaries to JSON files, including a well-named dump() function for writing to JSON. JSON is less tabular than CSV and less well-suited for data analysis, but it is useful for passing data back and forth through web interfaces.

In this assignment we read data from a JSON file at the beginning, and write data back to the JSON file in Menu Choice 3. Note that the code blocks reading and writing the file are embedded in try-except blocks, discussed below.

Error Handling with Try-Except Blocks

In this module we learned about error handling with the **try-except** framework. This framework resembles a conditional if-else block, in that the code you want to run is nested under a 'try' clause, and the 'except' block describes how you want the computer to respond if certain error conditions are met. There are some differences in syntax from an if-else structure. For instance, in the try-except syntax, you raise errors even within the try block, whereas in the if-else structure code is more contained between conditions. Additionally, the try-except syntax is unlike other Python syntax we have learned so far, which according to the lecture notes is because of the influence of other programming languages.

In this assignment we use this framework to make sure the first and last name of the student are only alphabetic characters, to make sure the file exists before we read in data, and to make sure the data is in a valid JSON format when it is written to file.

The use of 'e' as a variable name within try-except blocks is conventional, but I find it confusing and a little hard to parse. For this reason I used more specific variable names, such as "write_error," in each block.

The code

Below is the code for this assignment.

```
# ----------------------------------------------------------------------------
--------------- #
# Title: Assignment05
# Desc: This assignment demonstrates using dictionaries, files, and exception
handling
# Change Log: (Who, When, What)
#   RRoot,1/1/2030,Created Script
#   Sonu Mishra,5/12/25, edited script
# ----------------------------------------------------------------------------
--------------- #
#import json
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.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 = 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
student_data_unsaved: dict = {} #one row of unsaved student data
students_unsaved: list = [] # A table of unsaved student data


#Embed opening the json file and reading data in a try/except block
try:
    file = open(FILE_NAME, "r")
    students = json.load(file)
    file.close()
#Error handling for if the file does not exist
except FileNotFoundError as nofile: #using nofile instead of e to be less
confusing
    if file.closed == False:
        file.close()
    print("JSON file must exist before running this script!\n")
    print("Built-in Python error info: ")
    print(nofile, nofile.__doc__, type(nofile), sep='\n')
#using 'generic_error' instead of e to be less confusing
#this is the generic exception
except Exception as generic_error:
    if file.closed == False:
        file.close()
    print("There was a non-specific error!")
    print("Built-in Python error info: ")
    print(generic_error, generic_error.__doc__, type(generic_error),
          sep='\n')



# Present and Process the data
while (True):

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

    # Option 1: Enable user to input data
    if menu_choice == "1":
        #Embed data input in try/except to maintain quality
        try:
            #Input the data
            student_first_name = input("Enter the student's first name: ")
            if not student_first_name.isalpha():
                raise ValueError("The first name should just be letters."
                                 "No Sus5an Smiths here.")

            student_last_name = input("Enter the student's last name: ")
            if not student_last_name.isalpha():
                raise ValueError("The last name should just be letters.")
            # Course name - can be alphanumeric but it's within the try block
because
            # it's weird if you get an error and then get asked for course
name
            course_name = input("Please enter the name of the course: ")
            # Add user input data to a list of dictionaries of unsaved data
            #This is within the try block, otherwise it will save bad data
            student_data_unsaved = {"FirstName": student_first_name,
                                    "LastName": student_last_name,
"CourseName": course_name}
            students_unsaved.append(student_data_unsaved)
            print(f"You are registering {student_first_name}
{student_last_name} "
                  f"for {course_name}.")
            print("Remember to save your data using Menu Option 3.")

        except ValueError as justalpha:
            print(justalpha)
            print("-- Technical Error Message --")
            print(justalpha.__doc__)
            print(justalpha.__str__())
            print("Try again")
        except Exception as generic_error:
            print("There was a non-specific error!")
            print("-- Technical Error Message --")
            print(generic_error, generic_error.__doc__, type(generic_error),
sep='\n')
            print("Try again")



        #back to menu
        continue

    # Present the current data
    elif menu_choice == "2":

        # Display the data that is already saved
        print("The data that is already saved to file is:\n")
        for student in students:
            print(f"Student {student["FirstName"]} {student["LastName"]} is
enrolled"
                  f" in {student["CourseName"]}.")
        print("-"*50)

        #Display the unsaved data if there is any
        if students_unsaved != []:
            print("The students you want to register in this session but "
                  "haven't saved to file yet are:")
            for student in students_unsaved:
                print(f"Student {student["FirstName"]} {student["LastName"]}
for"
                      f" {student["CourseName"]}.")
                print("Remember to save this data using Menu Option 3.")
        else:
            print("There is no unsaved data at this time.")

        #Back to menu
        continue

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

        # Tell the user if there is no unsaved data
        if students_unsaved == []:
            print("There is currently no unsaved data to save.")
        # Write unsaved data to file and inform user of what happened
        else:
            #Display the unsaved data that is being saved now
            print("The following new data is being saved to file:")
            for student in students_unsaved:
                print(f"Student {student["FirstName"]} {student["LastName"]}
is "
                      f"enrolled in {student["CourseName"]}.")

            print("\nThe following data was saved to file previously:")
            for student in students:
                print(f"Student {student["FirstName"]} {student["LastName"]}
is "
                      f"enrolled in {student["CourseName"]}.")

            # Add just-saved data to the main students list
            for student in students_unsaved:
                students.append(student)

            # Reset unsaved data structures to prevent dupes
            students_unsaved = []
            student_data_unsaved = {}

        # Write all the stored data to the file, overwriting the file so no
dupes.
        #Embed it in a try-except block.
        ### Note this is out of the if/else above so it happens regardless of
whether there is
        ### any unsaved data when the user hits 3
        try:
            file = open(FILE_NAME, "w")
            json.dump(students, file) #The elegant 'dump' function
            file.close()
            continue
        except TypeError as write_error:
            print("Please check that the data is a valid JSON format.\n")
            print("-- Technical Error Message --")
            print(write_error, write_error.__doc__, type(write_error),
sep='\n')
        except Exception as generic_error:
            print("-- Technical Error Message --")
            print("Built-in Python error info: ")
            print(generic_error, generic_error.__doc__,
type(generic_error),sep="\n")
        finally:
            if file.closed == False:
                file.close()

        # Return to the menu
        ### This code shouldn't run if the 'try' block above succeeds
        ### because that has 'continue' too
        continue
    # Stop the loop
    elif menu_choice == "4":
        break  # out of the loop
    else:
        print("Please only choose option 1, 2, or 3")

print("Program Ended")
```

Figure 1: Code for Assignment 5.

Future Work

In this assignment, if the user triggers an error message by say entering a first name with a number in it, they are returned to the menu after the error messages. I am interested in figuring out how to get the user back to the point at which they made the error – entering the student’s first name – while still allowing them the option to exit and return to menu. But that is a task for another day!

Summary

This program offers users a menu of options and responds conditionally based on their input. Users are able to input information about multiple students and their enrollment in a single session and save it to a JSON file. Users can distinguish between saved and unsaved data. Users are also able to view all data within the JSON file including data saved in past sessions. Several actions are embedded in try-except blocks, including user entry of first and last names and reading and writing to and from JSON.