Jocelyne Booth
07/31/2024
ID FDN 110 A
Assignment 05

Advanced Collections and Error Handling

Introduction

This week, we're changing the type of file collection from lists to dictionaries, and storing them in json files instead of csv. Additionally, we are adding exceptions to

Modifying the Sample Code for JSON

Constants and Variables

Since this week we're working with json files instead of csv, I had to change the file name to be .json and set a variable called *json_data.* Also, we will be working with dictionaries rather than lists for the student data, so I defined *student_data* this way.

Importing the Data

To import the data, we first use the simple json.load() command and set this to our list of students. However, the print loop assumes that each student is entered with the map keys 'FirstName', 'LastName', 'CourseName' (calling this FLC for short). The Enrollments.csv data in the Mod05 folder has the second student's LastName column listed as 'Email.' When I tried running the loop in Fig. 1, I got an error that the LastName key could not be mapped due to the 2nd student'key being called Email. To fix this problem, I wanted to reset this student to the FLC format.

```
try:
    # Extract the data from the file
    file = open(FILE_NAME, "r")
    students = json.load(file)
    file.close()
    # Print the data. If the columns aren't matching the FirstName, LastName, CourseName format, a KeyError exception will be thrown.
    print('Data:')
    for student in students:
        print(f'{student['FirstName']} {student['LastName']} is in {student['CourseName']}.')
```

Figure 1. Importing the json file and trying to print it using the FLC keys. This is within the 'try' statement, so I can prepare for an error to be thrown if the keys of the json file are unexpected.

To do this, I set up a KeyError exception, which occurred when it tried reading the Email column as LastName. My plan was to then iterate through each student, take the

contents in columns 1 2 and 3, regardless of their map keys, and re-save it under columns with the keys matching our FLC format. I found the command dict.keys() which will create a dictionary list of the key names in order. To be able to iterate through these keys, I had to cast this to *list* type.

Then, I could save *student* as a new dictionary list in my desired FLC format and call the old student data using my dictionary keys list (Fig. 2). I tried appending this to the *students* list but somehow ended up in an infinite loop of re-defining *student* and re-appending. So, I just created a new temporary student list, *tempList*, outside the loop.

```
tempList = []  # make a new list, functions the same as students list
# Iterate through each student in the current json data
for student in students:
    keysList = list(student.keys())  # Gets the map keys for the current student, e.g. FirstName, LastName, CourseName
    print(keysList)
    # Redefine the current student to have the keys FirstName, LastName, CourseName. Uses the old keys to find the necessary entries
    student= {'FirstName':student[keysList[0]], 'LastName':student[keysList[1]], 'CourseName':student[keysList[2]]}
    tempList.append(student)  # Append the reassigned student data to tempList
    print(f'{student['FirstName']}, {student['LastName']}, {student['CourseName']}.')
```

Figure 2. Recreating the student list, but in FLC format. I iterate through the original students list and use each student's corresponding keys to re-map to the FLC keys.

Finally, I reopened the json file in write mode to rewrite the data using my FLC-corrected student data. I then also reopened it in read mode to reset the *students* list to now reflect these changes.

Option 1

I had to modify this option to now save the new student input data as a dictionary entry in FLC format. I did this by using the user inputs to save to the *student_data* dictionary and having the keys match FLC. Then, I just appended this to the *students* list.

Option 2

I modified option 2 in a similar way as option 1. Previously, it tries to iterate through *students* as if it's a list of lists, treating each *student* object as something that can be iterated through. However, dictionaries can't be indexed, so I just changed it from being student[index number] to student[key].

Option 3

Since saving data to .json files is different from .csv files, I just had to change the method used. The *students* list already had all the data in the correct format, so I just had to use the json.dump() into the file.

Option 4

Option 4 was the same this week as last week.

Error Handling

The biggest error to handle was when the .json file being read wasn't in FLC format, but I fixed this earlier. Next, I wanted to add errors for when someone types a number into the first name field.

To do this, I set up a *try* statement, and if any numbers are detected in the user's input for First Name, it would raise a ValueError exception. Then, I had an *except* statement that would describe the error to the user, then break out of the loop and end the program.

Summary

This assignment showed us how to use dictionaries and json files instead of lists, strings, and csv files. I think that these dictionaries are a lot more convenient than lists because of how explicit each column name is, and I like that you can define the column names at the same time as entering the data.