Miyles Clark
7/29/2024
Foundations of Programming: Python
Assignment 05
https://github.com/macxuw/IntroToProg-Python-Mod05/blob/main/Assignment05.py

Collections in JSON Menu Loop

Introduction

Assignment 05 utilizes a menu and while loop to save the student_data dictionary into a table of students. Once run, the program reads a file and loads the data into the student data table. Unlike previous assignments, Assignment 05 uses the JSON file format to load data, read it into a list of students, and write the old data and any user inputs to the same file.

A menu asks the user to choose:

- Enter new data.
- Display student data.
- Write student data to a JSON file.
- Exit the program.

Another objective in this assignment is to error-handle the initial JSON file read, ensure valid student first and last name inputs, and write to the JSON file.

Adding Code

The main objectives in Assignment 05 were to append the data in an existing JSON file, add that to a student list, and write new inputs to the JSON file. While the student list remained, the student_data list became a student_data dictionary. The file name constant switched from a CSV file extension to a JSON extension, and the student_data variable was edited to reflect the new dictionary type.

Lists and Collections

JSON files are more straightforward to load because they cannot be split into parts using .readlines(); JSON opens the file in read mode and loads it into the list of students (Figure 1).

```
file = open(FILE_NAME, "r")

students = json.load(file)
```

```
        file.close()
```

Figure 1: Code to load existing data to students list

The program prompts the user to enter a first, last, and course name in menu choice one. The user can enter multiple students and only need to write to the file once; after every input, the data is assigned, in order, to the dictionary holding row data – student_data – and appended to the students list (Figure 2).

```
        student_data = {"FirstName": student_first_name, "LastName":
                student_last_name, "CourseName": course_name}
students.append(student_data)
```

Figure 2: Specify variable order in a dictionary and append to list

Menu choice two creates a loop to display a formatted data output of the student_data in the list of students (Figure 3).

```
    for student in students:
      print(f"{student["FirstName"]} {student["LastName"]} is enrolled in
                {student["CourseName"]}")
```

Figure 3: All data in the students list

Menu choice three writes all student data to a JSON file (Figure 4).

```
        file = open(FILE_NAME, "w")
json.dump(students, file, indent=1)
```

Figure 4: Save the students list to JSON file

The code opens the file in write mode and dumps – saves – the list to the specified file. Setting an indent to 1 saves the file in a more readable and user-friendly format.

Error Handling

Assignment 05 includes several methods of handling the errors a user may encounter. The initial code to read an existing JSON file and menu choices one and three use the Try-Except code to catch specific errors the program confronts. At the program's start, a Try-Except block will flag a missing JSON file, give the user a custom error message, and initialize an empty file. The try argument is the code from Figure 1, and the except argument will run if the file is not found. (Figure 5).

```
except FileNotFoundError as e:
# If file not found, create
print("This script requires an existing file! Initializing file.\n")
file = open(FILE_NAME, "w")
```

Figure 5: Error handling if file not found

Menu choice three will display a message if the file is not saved. Unlike Figure 5, it has no automatic fix; however, like all the Try-Except blocks, the code identifies the type of error found. Error handling in menu choice one employs a Try-Except block, while also coded to accept inputs that meet certain conditions. The first and last name must only contain letters and must be two or more characters (Figure 6). A custom error message tells the user the input requirements if the input does not meet the criteria specified.

```
if not student_first_name.isalpha() or len(student_first_name) < 2:
```

Figure 6: Ensure names are letters only and 2 or more characters

Courses usually contain numbers, so the only restriction on this input is that it must be three or more characters (Figure 7). Again, the error message informs the user what occurred.

```
if not len(course_name) > 2:
```

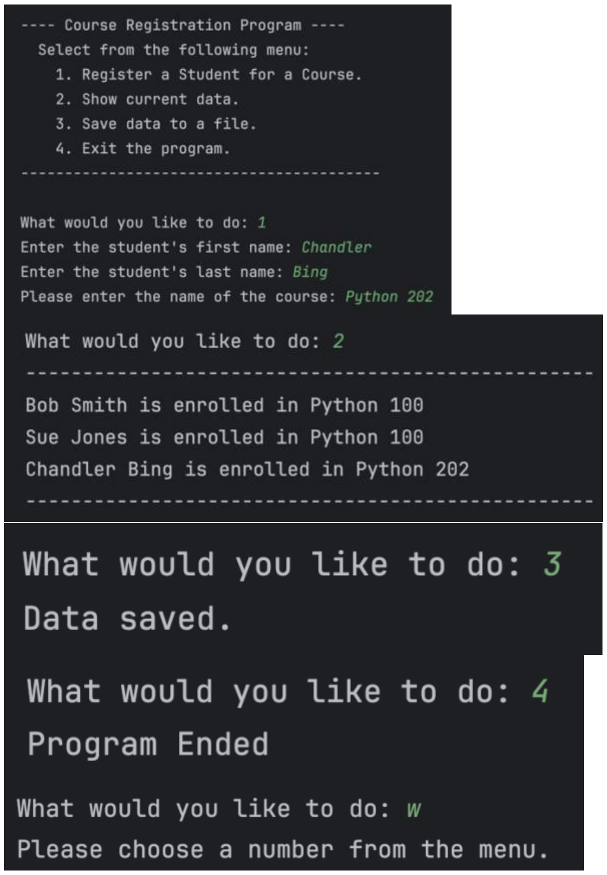
Figure 7: Course name must be at least 3 characters

Disability Accessibility

All user inputs have a .strip() string method attached to prevent errant spaces on either side of the inputs, making a more user-friendly program. Users with disabilities may accidentally add spaces when typing; for example, the angle of a user’s hands may hit the spacebar when typing the adjacent letters. Data uniformity increases when inputs automatically strip extra spaces. Error handling features further aid disabled users by specifying the types of data accepted.

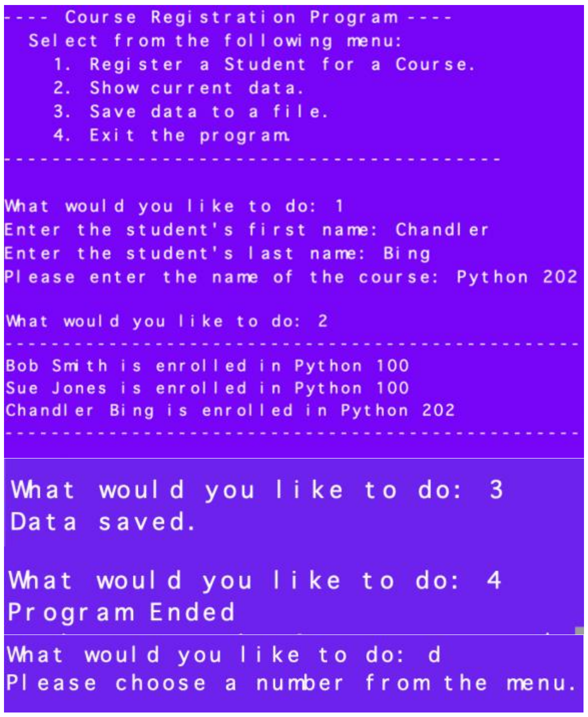
Results

Testing my program in PyCharm (Figure 8) and Terminal (Figure 9) returned the correct outcome, whether a prompt for data, a print statement, or an end to the program.



```
---- 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.
-----------------------------------------

What would you like to do: 1
Enter the student's first name: Chandler
Enter the student's last name: Bing
Please enter the name of the course: Python 202
What would you like to do: 2
--------------------------------------------------
Bob Smith is enrolled in Python 100
Sue Jones is enrolled in Python 100
Chandler Bing is enrolled in Python 202
--------------------------------------------------
What would you like to do: 3
Data saved.
What would you like to do: 4
Program Ended
What would you like to do: w
Please choose a number from the menu.
```

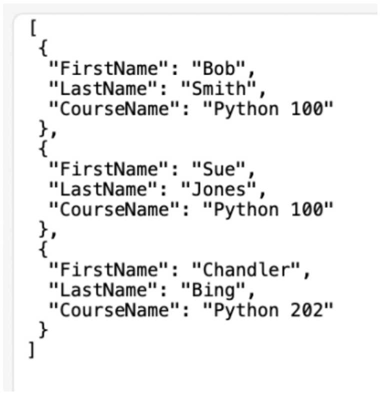
Figure 8: Program output in PyCharm



```
---- 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.
-----------------------------------------

What would you like to do: 1
Enter the student's first name: Chandler
Enter the student's last name: Bing
Please enter the name of the course: Python 202

What would you like to do: 2
--------------------------------------------------
Bob Smith is enrolled in Python 100
Sue Jones is enrolled in Python 100
Chandler Bing is enrolled in Python 202
--------------------------------------------------

What would you like to do: 3
Data saved.

What would you like to do: 4
Program Ended
What would you like to do: d
Please choose a number from the menu.
```

Figure 9: Program output in Terminal

The JSON file for this assignment returns a student's first and last name and a course name. The data is listed next to the corresponding dictionary key (Figure 10).



```
[
 {
  "FirstName": "Bob",
  "LastName": "Smith",
  "CourseName": "Python 100"
 },
 {
  "FirstName": "Sue",
  "LastName": "Jones",
  "CourseName": "Python 100"
 },
 {
  "FirstName": "Chandler",
  "LastName": "Bing",
  "CourseName": "Python 202"
 }
]
```

Figure 10: CSV file data

Summary

Saving to a JSON file is more straightforward than a CSV, and using a dictionary returns a more user-friendly output. The catch with JSON is that a file must exist to load it – it will not automatically generate an empty file. Error handling addresses this by generating a file if the program does not find one. Moreover, error handling using the Try-Except code allows the programmer to specify the conditions required for individual user input prompts.