Kelly Choi
Date: February 15, 2024
Title: Assignment 04
GitHub: https://github.com/kchoi78/IntroToProg-Python-Mod05

Collections of Data Pt 2

Introduction

In this lesson we took an even deeper dive into working with data files (JSON specifically), exception handling with the program, and using GitHub. JavaScript Object Notation, more commonly known as JSON, is a type of data format that contains key-value pairs. JSON files are widely used in data communication and storage. In this assignment, I took a version of the file from the previous assignment then edited steps to process existing JSON files and add in exception handling.

Processing the JSON data file

The menu option, the constants, and the variables remain the same in this assignment.
The json package must be installed in the beginning of the program, as well as the JSON Decode Error, in order for this program to run without a hitch.

```
#import the json package in python

import json
from json import JSONDecodeError
```

(Figure 1: Importing json package in PyCharm)
Instead of looping through a CSV file, the JSON file is simply loaded into the program by using the load() function in the json package. Previously when reading a CSV file, this process took 5 lines of code. Now it takes just 3 lines of code.

```
#NEW: when the program starts, the json file is automatically read into a list
    file = open(FILE_NAME, "r")
    students = json.load(file)
    file.close()
```

(Figure 2: Reading an existing JSON file)

The bulk of the program also remains the same as the previous assignment. Within a While loop, the program presents the user with the menu of options as previously defined, then per

each choice, the program will execute a different set of commands. This time, the program reads and processes a JSON file, instead of a csv file.

```
#NEW: write into dict, then display the list of dict rows
    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}.")
    continue
```

(Figure 3: Write into dictionary)

Option 1 still takes user input for first name, last name, and course name. What's new in this program is that the input variables are now stored into a dictionary called `student_data`, with their corresponding keys which are enclosed in double quotation marks. The new row of data is then appended to a list of dictionary rows called `students`. These input variables are then displayed as a string to the user.

When the user selects Option 2, the program will display all the values in the list, `students`. Like the previous program, this program loops and prints the three key-value pairs in the list of dictionary rows.

Option 3 saves the input to the JSON file. This is done with the json.dump() function, which is much simpler than looping through each row in the data collection then writing the lines.

Once the file has been updated with the new input, the user may select Option 4 and end the program.

Exception Handling

Exception handling, or structured error handling, is a fabulous way to cushion your program so that it can deal with new errors and bugs introduced by the program users. Sometimes you cannot control everything in life, but error handling is a way to contain and manage some of those mistakes. There are three places in this program that utilizes structured error handling.

```
#NEW: error handling - the program provides error handling when the file is read
try:
#NEW: when the program starts, the json file is automatically read into a list
    file = open(FILE_NAME, "r")
    students = json.load(file)
    file.close()
except FileNotFoundError as e:
    print("File must exist in the directory before running the script.")
    print("***Technical Error Message***")
    print(e, e.__doc__, type(e), sep='\n')
except Exception as e:
    print("There was a non-specific error!\n")
    print("***Technical Error Message***")
    print(e, e.__doc__, type(e), sep='\n')
except JSONDecodeError as e:
    print("Data in file is not valid. Resetting..")
    print("***Technical Error Message***")
    print(e, e.__doc__, type(e), sep='\n')

    file = open(FILE_NAME, 'w')
    json.dump(students, file)
finally: # this code is executed whether try finishes, or the except finishes
    if not file.closed:
        file.close()  # this would make sure the file is always closed
```

(Figure 4: Error handling #1)

The first error handling starts when the program is trying to open and read the JSON file. The statement `try:` opens the error handling block -- everything indented underneath is within the block. When the program tries to open the file, there may be a few errors. The first is `FileNotFoundError`, which is a predefined Exception class that says the file was not found. The second is `Exception`, which is a generic Exception class that is a catch-all. The third is a special Exception class called `JSONDecodeError` which occurs when the JSON data is invalid for whatever reason such as missing commas, missing brackets, or other general syntax issues. Once the program clears all the Exception classes, then it will open the JSON file and read into the program. The `finally:` statement is executed whether or not the errors are present. In this case, if the file isn't closed, the program will close the file.

```python
if menu_choice == "1":  # This will not work if it is an integer!
    try: #NEW: program provides error handling when numbers are entered
        student_first_name = input("Enter the student's first name: ")
        if not student_first_name.isalpha():
            raise ValueError("Please only enter letters.")

        student_last_name = input("Enter the student's last name: ")
        if not student_last_name.isalpha():
            raise ValueError("Please only enter letters.")

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

    #NEW: write into dict, then display the list of dict rows
        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}.")
        continue
    except ValueError as e:
        print(e)
        print("***Technical Error Message***")
        print(e.__doc__)
```

(Figure 5: Error handling #2)

The second error handling occurs at menu Option 1. I found this part to be particularly useful and challenging. The error handling here will throw a ValueError Exception class whenever the input is not all alphabet letters. This makes sense as most names do not contain special or numerical characters (sorry to Elon Musk's children!)

At first, I tried to run the program with just the `try-if not-raise` statements. This did not work because I had not closed the error handling block with either `except` or `finally`.

```python
# Save the data to a file
elif menu_choice == "3":
    try: #NEW: program handles errors
        file = open(FILE_NAME, "w")
        #NEW: write to JSON
        json.dump(students, file)
        file.close()
        print("The following data was saved to file!")
        for student in students:
            print(f"Student {student['FirstName']} {student['LastName']} is enrolled in {student['CourseName']}")
        continue
    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')
    except Exception as e:
        print("***Technical Error Message***")
        print(e, e.__doc__, type(e), sep='\n')
    finally:
        if file.closed == False:
            file.close()
```

(Figure 6: Error handling #3)

The third error handling occurs at menu Option 3, where the program tries to save the data to the JSON file. Similar to Figure 4 and Figure 5, the code starts with try: and tries to open the JSON file and save the new data into it. If it encounters a `TypeError` class, it means that the data type is invalid. The three print statements will print the enclosed strings, then also print the `__doc__` code, which is a built-in Python documentation string that describes the exception type, and the `__str__()` code, which returns the error message. It is important to provide an easy to understand custom error message for the users, as well as the technical error message.

Conclusion

In this assignment we have learned to read and process JSON files, and have added elevated features to the program such as structured error handling. Working with JSON files was much simpler than reading CSV files line by line; the keys in JSON files are also useful in gaining insight from the raw data. Adding ways to prevent user error was also very useful; this would be valuable in ensuring that your programs can run smoothly and that the data you collect is clean and readable.