Name: Luis Candela

Date: 11/13/2024

Course: Foundations of Programming (Python)

Module 05 - Advanced Collections and Error Handling

Introduction

In Python data processing needs a program that can compare, sort, analyze and store information. For each task the programmer uses functions, operators, conditional statements and loops. During Module 4 the student learned how to create a script that used input information from the user to display, manipulate and store information in a list format. Lists are a useful type of data collection that stores information as an array or table with rows and columns and specific indexes to locate an element or list inside of the original collection. Lists are mostly used when working with same types of data for example time series and scientific data analysis, they are fast to locate and iterable for repeated processing loops.
Another powerful data collection are dictionaries which contrary to lists work in a key-value reference system and are widely used in the industry to store information in a directory format where a set of information is referenced to a key which holds the location for that specific data. Dictionaries can store combine data of different types and sizes, making them helpful for data that has multiple attributes, classes, categories etc.
Using the knowledge on lists the user can collect information and create lists of dictionary rows which are the converted into dictionary format as a JSON file.
JSON files are a popular type of file that stores information as a dictionary, it is easily comprehended by users and programs and can store large amounts of data regardless of type and size.
In Module 4 the user created a script that worked with CVS files to collect and process data into lists. This time the user will build on the same script to store similar information into a list of dictionaries which can be read and edited into a JSON file.
To finalize this module covers the topic of Error Handling, a comprehensive way to notify a program user of an error without complex coding language.

Assignment04- Create a program

Create a Python program that demonstrates using constants, variables, and print statements to display a message about a student's registration for a Python course. This program is very similar to Assignment04, but **It adds the use of data processing using dictionaries and exception handling.**

As mentioned earlier the script builds of the program created in Module 4. It starts by describing the Variables and Constants that will be used for this program. The difference is that now the script uses lists and dictionaries. The file is a JSON type and the rows of the list although in strings are in a dictionary format (student_data: = {}) to be read and written into the Enrollments.json output file.
An important finding was the importance of staging variables as empty strings, lists or dictionaries because it helps the coder from mixing data types further in the program. Python will mark errors if the variables are misplaced, misnamed or mistaken for another data type.

```
# ------------------------------------------------------------------------------------------ #
# Title: Assignment05
# Desc: This assignment demonstrates using dictionaries, files, and exception handling
# Change Log: (Who, When, What)
# RRoot,1/1/2030,Created Script
# Luis, 11/13/2024, Edited script
# ------------------------------------------------------------------------------------------ #

# 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.
student_data: dict = {}  # one row of student data
students: list = []  # a table of student data
csv_data: str = ''  # Holds combined string data separated by a comma.
file = None  # Holds a reference to an opened file.
menu_choice: str  # Hold the choice made by the user.
json_data: str = ''
```

The Python JSON Module has the necessary functions to work with this type of files and must be imported at the start of the program. This module has built-in functions that can manipulate JSON files with fewer code steps.

```
import json # This imports the Python JSON Module to work with a file of this format
```

This program will incorporate error handling which uses custom Exceptions made acceptable to Python with CLASS and PASS functions.

```
# Custom Exception for invalid menu choice
class InvalidMenuChoiceError(Exception):
    pass
```

Before interacting with the user there is a back-end step that opens the existing JSON file and reads the dictionary rows. By using th except-try error handling the coder ensures the user is aware of a missing file and prompts a message to locate it. If file does exist the program executes the following rows.

```
# When the program starts, read the file data into a list of dictionaries
# Extract the data from the file
try:
    file = open(FILE_NAME, 'r')
    students = json.load(file)
    file.close()
    # This exception is a caveat of a missing file. It notifies the file
    # was not found and the following steps to correct the error.

except FileNotFoundError as e:
    print('The file',FILE_NAME,'you are trying to open does not exist or may be misplaced in a different directory')
```

Shifting from a “list” mindset a dictionary format was challenging, but after understanding the differences in notation and properties. The program worked well.

```
Traceback (most recent call last):
  File "/Users/candela/PycharmProjects/pythonProject/venv/Mod05/Assignment05.py", line 85, in <module>
    print(f"Student {student[0]} {student[1]} is enrolled in {student[2]}")
                    ~~~~~~~^^^
KeyError: 0
```

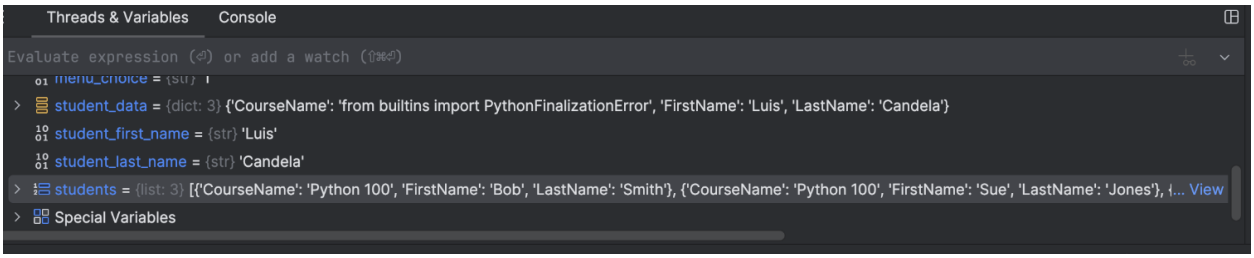
Once the file task is finished a for loop writes the file rows into a list called Students.

```
# Write the contents to a file for each row in the table
```

```
for item in students:
    print(f"FirstName: {item['FirstName']}, LastName: {item['LastName']}, CourseName:
{item['CourseName']}")
```

The following process is similar to the Module 4 script by displaying the Menu to the user and asking for a choice input. The difference here lies in another error handling exception for non-string input cases. As in the previous exception the program alerts the user of the mistake and asks for a valid input.

While writing the program the debugger function was useful to prevent error before they occurred and helped understand the back-end results of the process before the were displayed to the user.



An important lesson learned in this step was understanding the importance of indentation inside loops. Since this script uses several levels of conditions and exceptions inside the while loop, correct indentation is crucial for the conditionals to work or not depending of the users answers and repeat the input question if needed. As standard practice loops inside of loops must have 4-space indentations between them.

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

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

        if menu_choice not in ['1', '2', '3', '4']:
          raise InvalidMenuChoiceError("Invalid choice. Please select a valid option (1, 2, 3,
or 4).")
        #break  # Exit the loop if a valid choice is entered
    except InvalidMenuChoiceError as e:
        print(e)  # Print the exception message and prompt again
        continue
```

Once the menu choice is correctly entered the program proceeds to execute the menu task. Although the questions are similar there are two differences from Assignment 4. Both first name and last name have “ValueError” exceptions and the new information is written into a dictionary row.

```
if menu_choice == "1":  # This will not work if it is an integer!
    while (True):
     try:
         student_first_name = input("Enter the student's first name: ")
         if not student_first_name.isalpha():
           raise ValueError("Invalid entry. Input must be a string.\nPlease try again")
         break

     except ValueError as ve:
       print(ve)  # Print the exception message and prompt again
       continue

    while(True):
     try:
         student_last_name = input("Enter the student's last name: ")
         if not student_last_name.isalpha():
           raise ValueError("Invalid entry. Input must be a string.\nPlease try again")
         break

     except ValueError as ve: #"ve" is a variable reference for the exception "ValueError
       print(ve)  # Print the exception message and prompt again
       continue

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

    student_data = {
      "FirstName": student_first_name,
      "LastName": student_last_name,  # add input information into this dictionary row
      "CourseName": course_name
     }
```

The new dictionary row with the new input is appended into a list and a message is promped to let the user review the input.

```
    students.append(student_data) #this line appends the new student data row to the students list
    print(f"You have registered {student_first_name} {student_last_name} for {course_name}.") #prompts message
    continue
```

If user selects Choice #2 all data is displayed by iterating through the list of lists and printing out each students information in a customized sentence.

```
  # Present the current data
  elif menu_choice == "2": # Identifies the second menu choice

    # Process the data to create and display a custom message
    print("-"*50)
    for student in students: #using a for loop displays every element of the list of rows in a customized sentence.
        message = print(f"In the course '{student['CourseName']}', {student['FirstName']} {student['LastName']} is enrolled.")
    print("-"*50)
    continue
```

Menu choice #3 will rewrite old and new data into the Enrollments list and saves the action by closing the file.

```
  # Save the data to a file
  elif menu_choice == "3":
    file = open(FILE_NAME, "w") #This overwrites the previous file with new appended row.
    json.dump(students, file)# JSON Module function "json.dump" saves the new row into the JSON File
    file.close()
    continue
```

Finally menu choice for breaks out of the first level loop and ends the program. The previous exceptions make the “else” statements of the previous script redundant.

```
    # Stop the loop
    elif menu_choice == "4":
        break  # breaks out of the loop and ends the program.
    #else:          #this step was replaced by the exception
        #print("Please only choose option 1, 2, or 3")

print("Program Ended")
```

Summary

Dictionaries are a powerful collection to store information of different types and sizes into a list of categories identified by a key-value reference. By using lists and dictionaries together the coder can create comprehensive lists of data which can be not only stored but also displayed to the front-end users.
During the creation of this script previous concepts where practiced and understood more thoroughly.
Conditional statements are useful sorting tools and can be more user friendly by incorporating exception error handling with comprehensive error prompts.
Loops are more efficient when written in a more “pythonic” format and must be hierarchically placed in the correct indentation to follow the programs flow.
The debugger function from Pycharm was very useful to understand what a program is actually doing with the data and prevents future errors in the script
Combining the knowledge of this course we are able to create more complex code that better caters to the user’s needs while collecting and processing data in a more efficient way.