December 4, 2024

steve stylin

Bellevue University

Advance ePython

Module 8.2: JSON Practice

# Code Explanation

The Student Management Program is a well-structured Python application that utilizes JSON for data storage and Tkinter for user interaction. Below, I will break down the program into its key components, explaining each part in detail.

*1. Importing Libraries*

The program begins with the programmer’s comment and the importation of the necessary libraries:

import json

import os

import tkinter as tk

from tkinter import messagebox

* **json**: This library is used for parsing JSON data, allowing the program to read from and write to JSON files.
* **os**: Although not explicitly used in the provided code, this library can be useful for file path manipulations.
* **tkinter**: This is the standard GUI toolkit for Python, enabling the creation of windows and dialogs.
* **messagebox**: A module within Tkinter that provides a simple way to display message boxes.

*2. Defining the JSON File Path*

The path to the JSON file where student data is stored is defined as follows:

json\_file\_path = 'C:\csd\csd-325\Module-8\student.json'

This path should be adjusted based on the actual location of the student.json file on your system.

*3. Loading Student Data*

The load\_students function is responsible for loading student data from the JSON file:

def load\_students():

try:

with open(json\_file\_path, 'r') as file:

students = json.load(file)

return students

except FileNotFoundError:

print("Error: The file student.json was not found.")

exit()

except json.JSONDecodeError:

print("Error: The file student.json is not a valid JSON.")

exit()

* **Error Handling**: The function includes error handling to manage scenarios where the file does not exist or contains invalid JSON. If an error occurs, a message is printed, and the program exits gracefully.

*4. Printing the Student List*

The print\_students function displays the list of students:

def print\_students(students):

print("Original Student List:")

for student in students:

print(f"{student['F\_Name']} , {student['L\_Name']} : ID = {student['Student\_ID']} , Email = {student['Email']}")

This function iterates through the list of students and prints their first name, last name, student ID, and email in a formatted manner.

*5. Adding a New Student*

The add\_student function allows the user to add a new student to the list:

def add\_student(students, first\_name, last\_name, student\_id, email):

for student in students:

if student['Student\_ID'] == student\_id:

print(f"\nError: Student ID {student\_id} already exists.")

exit()

new\_student = {

"F\_Name": first\_name,

"L\_Name": last\_name,

"Student\_ID": student\_id,

"Email": email

}

students.append(new\_student)

* **Duplicate Check**: Before adding a new student, the function checks if the student ID already exists in the list. If it does, an error message is displayed, and the program exits.

A screen shot of a computer

Description automatically generated

* **Appending New Student**: If the ID is unique, a new student dictionary is created and appended to the existing list.

*6. Saving the Updated Student List*

The save\_students function saves the updated list back to the JSON file:

def save\_students(students):

try:

with open(json\_file\_path, 'w') as file:

json.dump(students, file, indent=4)

except IOError:

print("Error: Unable to write to the file student.json.")

exit()

This function attempts to write the updated student list to the JSON file. If an IOError occurs, an error message is printed, and the program exits.

*7. GUI Notification for Saving*

The ask\_to\_save function prompts the user with a message box:

def ask\_to\_save():

root = tk.Tk()

root.withdraw() # Hide the main window

return messagebox.askyesno(f"\n{json\_file\_path} ","This file has been modified outside. Do you want to reload it?")

* **Tkinter Window**: The main window is hidden, and a message box is displayed asking the user if they want to reload the modified file. The user's response is returned as a boolean.

A screenshot of a computer error message

Description automatically generated

*8. Main Function Execution*

The main function orchestrates the program's flow:

def main():

students = load\_students()

print\_students(students)

# User input for new student details

first\_name = "Steve"

last\_name = "Stylin"

student\_id = 57852 # The student id is fictional

email = "sstylin@bellevue.edu" # the university email is fictional

add\_student(students, first\_name, last\_name, student\_id, email)

print("Updated Student List:")

print\_students(students)

if ask\_to\_save():

save\_students(students)

print("Changes saved to student.json.") # The new student will be added to the file if user clicks on YES on the notification GUI message

A computer screen shot of a program

Description automatically generated

else:

print("No changes were saved.") # If user clicks NO, no change will be made on the student.json file

* **Loading and Displaying Students**: The program starts by loading the student data and displaying the original list.
* **Adding a New Student**: It then simulates adding a new student with predefined details.
* **User Interaction**: Finally, it prompts the user to save changes, handling their response accordingly.

A computer screen shot of a program

Description automatically generated

*9. Program Entry Point*

The following code ensures that the main function is executed when the script is run directly:

if \_\_name\_\_ == "\_\_main\_\_":

main()

*Conclusion*

This Student Management Program is a robust example of how to manage data using Python and JSON. It effectively demonstrates file handling, error management, and user interaction through a GUI. By following the structure outlined above, you can easily modify or extend the program to suit additional requirements or functionalities.

**PS**: The student.json file is in its original state. When the program runs, student Steve Stylin and information will be added to it if the user clicks on the yes button.