**Address-Book application**

import json

# Function to load address book data from file

def load\_data(filename):

try:

with open(filename, 'r') as file:

return json.load(file)

except FileNotFoundError:

return []

# Function to save address book data to file

def save\_data(filename, data):

with open(filename, 'w') as file:

json.dump(data, file)

# Function to add a contact

def add\_contact(address\_book):

name = input("Enter name: ")

phone = input("Enter phone number: ")

email = input("Enter email: ")

address\_book.append({"name": name, "phone": phone, "email": email})

print("Contact added successfully!")

# Function to view all contacts

def view\_contacts(address\_book):

if not address\_book:

print("Address book is empty.")

else:

for index, contact in enumerate(address\_book, start=1):

print(f"{index}. Name: {contact['name']}, Phone: {contact['phone']}, Email: {contact['email']}")

# Function to view saved data

def view\_saved\_data(address\_book):

if not address\_book:

print("No data saved yet.")

else:

print("Saved data:")

view\_contacts(address\_book)

# Main function

def main():

address\_book = load\_data("address\_book.json")

while True:

print("\nAddress Book Menu:")

print("1. Add Contact")

print("2. View Contacts")

print("3. View Saved Data")

print("4. Exit")

choice = input("Enter your choice: ")

if choice == '1':

add\_contact(address\_book)

elif choice == '2':

view\_contacts(address\_book)

elif choice == '3':

view\_saved\_data(address\_book)

elif choice == '4':

save\_data("address\_book.json", address\_book)

print("Address book saved. Goodbye!")

break

else:

print("Invalid choice. Please try again.")

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

main()

**Explanation of the above code**

**Step 1: Importing Necessary Modules**

import json

* This line imports the **json** module, which allows us to work with JSON data.

**Step 2: Defining Functions for Data Handling**

# Function to load address book data from file

def load\_data(filename):

try:

with open(filename, 'r') as file:

return json.load(file)

except FileNotFoundError:

return []

# Function to save address book data to file

def save\_data(filename, data):

with open(filename, 'w') as file:

json.dump(data, file)

* These functions (**load\_data** and **save\_data**) handle loading and saving address book data from/to a file in JSON format.
* **load\_data** tries to open the specified file and load the data. If the file doesn't exist or is empty, it returns an empty list.
* **save\_data** writes the provided data to the specified file in JSON format.

**Step 3: Defining Functions for Address Book Operations**

# Function to add a contact

def add\_contact(address\_book):

name = input("Enter name: ")

phone = input("Enter phone number: ")

email = input("Enter email: ")

address\_book.append({"name": name, "phone": phone, "email": email})

print("Contact added successfully!")

# Function to view all contacts

def view\_contacts(address\_book):

if not address\_book:

print("Address book is empty.")

else:

for index, contact in enumerate(address\_book, start=1):

print(f"{index}. Name: {contact['name']}, Phone: {contact['phone']}, Email: {contact['email']}")

* These functions (**add\_contact** and **view\_contacts**) handle adding new contacts and viewing existing contacts in the address book.
* **add\_contact** prompts the user to enter contact details (name, phone number, and email) and adds the contact to the address book.
* **view\_contacts** displays all contacts stored in the address book. If the address book is empty, it prints a message indicating that.

**Step 4: Main Function**

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

main()

This conditional statement checks if the script is being run directly (not imported as a module) and then calls the **main** function to start the program.

**Step 5: main Function**

# Main function

def main():

address\_book = load\_data("address\_book.json")

while True:

# Display menu options

print("\nAddress Book Menu:")

print("1. Add Contact")

print("2. View Contacts")

print("3. Exit")

choice = input("Enter your choice: ")

if choice == '1':

add\_contact(address\_book)

elif choice == '2':

view\_contacts(address\_book)

elif choice == '3':

save\_data("address\_book.json", address\_book)

print("Address book saved. Goodbye!")

break

else:

print("Invalid choice. Please try again.")

* The **main** function is the entry point of the program.
* It loads the address book data from the file **"address\_book.json"**.
* It then displays a menu with options to add a contact, view contacts, or exit the program.
* Based on the user's choice, it calls the corresponding function (**add\_contact**, **view\_contacts**, or **save\_data**) or displays an error message for an invalid choice.

**Introduction to json module in Python**

The **json** module in Python provides functions for encoding and decoding JSON (JavaScript Object Notation) data. JSON is a lightweight data interchange format that is easy for humans to read and write, and easy for machines to parse and generate. It is commonly used for exchanging data between a web server and a client or for storing configuration settings.

Here's what the **json** module can do:

1. **Encode JSON**: The **json.dumps()** function is used to serialize Python objects (such as dictionaries or lists) into a JSON formatted string. This is useful when you want to store Python data in a file or send it over the network.

import json

data = {"name": "John", "age": 30, "city": "New York"}

json\_string = json.dumps(data)

print(json\_string) # Output: {"name": "John", "age": 30, "city": "New York"}

1. **Decode JSON**: The **json.loads()** function is used to deserialize a JSON string into a Python object. This is useful when you receive JSON data from an external source and need to convert it into a Python object for further processing.

import json

json\_string = '{"name": "John", "age": 30, "city": "New York"}'

data = json.loads(json\_string)

print(data) # Output: {'name': 'John', 'age': 30, 'city': 'New York'}

1. **File I/O**: The **json.dump()** and **json.load()** functions are used to write JSON data to a file and read JSON data from a file, respectively. This is useful for storing and retrieving structured data in JSON format.

import json

# Writing JSON data to a file

data = {"name": "John", "age": 30, "city": "New York"}

with open("data.json", "w") as file:

json.dump(data, file)

# Reading JSON data from a file

with open("data.json", "r") as file:

data = json.load(file)

print(data) # Output: {'name': 'John', 'age': 30, 'city': 'New York'}

Overall, the **json** module provides a convenient way to work with JSON data in Python, allowing you to easily encode and decode data, as well as read from and write to JSON files.

**Assignment: Address Book Application**

**Objective:**

The objective of this assignment is to develop an address book application in Python that allows users to manage their contacts. In addition to the basic functionalities, students will implement some extra features to enhance the application.

**Basic Functionalities:**

1. **Add Contact**: Allow users to add a new contact with fields such as name, phone number, and email.
2. **View Contacts**: Display all contacts stored in the address book.
3. **Search Contacts**: Allow users to search for contacts by name or any other relevant field.
4. **Update Contact**: Enable users to update existing contact details.
5. **Delete Contact**: Allow users to remove contacts from the address book.
6. **Save and Load Data**: Implement functionality to save the address book data to a file and load it when the program starts.

**Extra Functionalities (to be implemented by students):**

1. **Export Contacts**: Allow users to export all contacts to a CSV file for easy sharing and backup.
2. **Sort Contacts**: Implement the ability to sort contacts alphabetically by name.
3. **Merge Contacts**: Enable users to merge duplicate contacts based on name or email.
4. **Birthday Reminders**: Add an option to store and display birthdays of contacts, and remind users of upcoming birthdays.

**Instructions:**

1. Students are required to implement the basic functionalities listed above using Python.
2. Additionally, students must choose at least two of the extra functionalities to implement in their application.
3. Students should thoroughly test their application to ensure all functionalities work correctly.
4. Documentation should be provided explaining how to use the application and describing the implemented functionalities.
5. The assignment submission should include the Python source code files and any necessary documentation.