Project Report

Student Name: Suraj Kumar **UID:** 24MCA20013
Branch: MCA General **Section/Group:** 3_B
Semester: 1st **Date of Performance:** 05/11/2024
Subject Name: PYTHON PROGRAMMING LAB **Subject Code:** 24CAH-606

1. **Aim of the project:** Create a Personalized Contact Book app with tkiner.
2. **Software Requirements:**

Python Setup

1. **Python Interpreter**
   - **Version:** Install Python (e.g., 3.8, 3.9, 3.10).
   - **Installation:** Download from Python website or use package managers (apt, brew).
2. **Development Environment**
   - **IDE/Text Editor:**
     - **Jupyter Notebook:** For interactive coding, especially in data science.
3. **Python Installation:** Python Version: Python 3.6 or later. Download the latest version from the officialDownload Python | Python.org

4. **Install Anaconda and Jupyter Notebook:** Downlods and install Anaconda from https://repo.anaconda.com/archive/Anaconda3-2022.05-Windows-x86_64.exe.
Open "Anaconda Prompt" by finding it in the windows (start) Menu.

3. **Program Logic:**

This Tkinter-based application allows users to manage a contact book where they can add, edit, delete, and search contacts. The contacts are stored in a JSON file, allowing the data to persist even after the application is closed. Below is an explanation of the program logic, broken down into key components:

1. **Loading and Saving Contacts**

- **Loading Contacts (load_contacts)**:
    - The program first attempts to load contacts from a JSON file (contacts.json). If the file is missing or contains invalid data, it returns an empty dictionary.
    - The contacts are stored in the contacts variable as a dictionary, where each contact's name is the key, and the value is another dictionary containing phone and email.
- **Saving Contacts (save_contacts)**:
    - The contacts dictionary is saved back to the contacts.json file whenever a contact is added, deleted, or edited. This is done using the json.dump() method.

2. Adding a Contact (add_contact)

- Users can input a contact's name, phone, and optionally email.
- If both name and phone are provided, the contact is added to the contacts dictionary, and the list of contacts is updated in the Listbox widget.
- After the new contact is added, the contact list is saved to the file, and the input fields are cleared.
- If any required field is missing, a warning message box appears.

3. Updating the Contact List (update_contact_listbox)

- This function is called whenever there is a change in the contact list (e.g., adding, deleting, or editing contacts).
- It clears the current Listbox and reloads it with the names of all contacts from the contacts dictionary.

4. Deleting a Contact (delete_contact)

- The user can select a contact from the Listbox and delete it.
- The contact is removed from the contacts dictionary, and the contact list is updated.
- After deletion, the updated list of contacts is saved back to the JSON file.
- If no contact is selected, a warning message is displayed.

5. Searching for a Contact (search_contact)

- The search functionality allows the user to type in a search query (e.g., part of a contact's name).
- The program filters the contacts and only displays those whose names match the search query (case-insensitive).
- If no contacts match, an informational message is displayed.

6. Viewing and Editing Contact Details (view_contact_details and edit_contact)

- **View Contact Details**: When a user selects a contact from the Listbox, its details (name, phone, email) are loaded into the input fields so the user can view or edit them.
- **Edit Contact**: The user can modify the details of the selected contact. If the contact's name is changed, the old contact entry is removed from the dictionary, and the new entry is saved. The contact list and JSON file are updated accordingly.

7. UI Elements (Tkinter Widgets)

- **Entry Fields**: For entering contact details (name, phone, and email).
- **Buttons**: For adding, editing, deleting, and searching contacts.
- **Listbox**: To display the contact names and allow the user to select a contact for editing or deletion.
- **Search Bar**: An input field where the user can type a search query, and a button to trigger the search function.

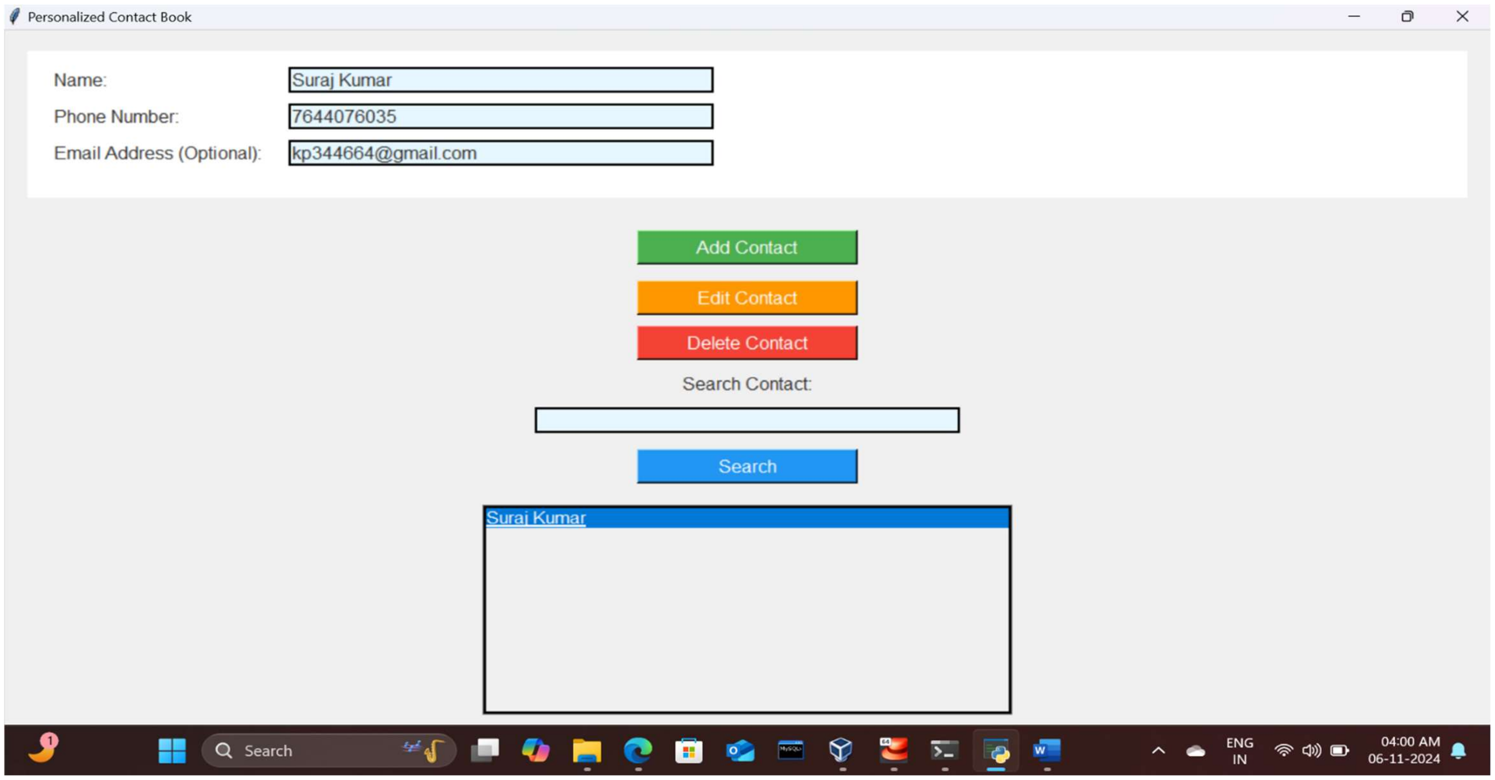
4. Code:

```
import tkinter as tk
from tkinter import messagebox
import json
def load_contacts():
    try:
        with open("contacts.json", "r") as file:
            return json.load(file)
    except (FileNotFoundError, json.JSONDecodeError):
        return {}
def save_contacts():
    with open("contacts.json", "w") as file:
        json.dump(contacts, file)
def add_contact():
    name = name_entry.get()
    phone = phone_entry.get()
    email = email_entry.get()

    if name and phone:
        contacts[name] = {"phone": phone, "email": email}
        update_contact_listbox()
        save_contacts()
        clear_input_fields()
    else:
        messagebox.showwarning("Input Error", "Please enter both Name and Phone.")
def update_contact_listbox():
    contact_listbox.delete(0, tk.END)
    for name in contacts:
        contact_listbox.insert(tk.END, name)
def clear_input_fields():
    name_entry.delete(0, tk.END)
    phone_entry.delete(0, tk.END)
    email_entry.delete(0, tk.END)
```

```
def delete_contact():
    selected_contact = contact_listbox.get(tk.ACTIVE)
    if selected_contact:
        del contacts[selected_contact]
        update_contact_listbox()
        save_contacts()
    else:
        messagebox.showwarning("Selection Error", "Please select a contact to delete.")
def search_contact():
    search_query = search_entry.get().lower()
    found_contacts = {name: details for name, details in contacts.items() if search_query in name.lower()}

    contact_listbox.delete(0, tk.END)
    if found_contacts:
        for name in found_contacts:
            contact_listbox.insert(tk.END, name)
    else:
        messagebox.showinfo("No Results", "No contacts found matching your search.")
def view_contact_details(event):
    selected_contact = contact_listbox.get(tk.ACTIVE)
    if selected_contact:
        name_entry.delete(0, tk.END)
        phone_entry.delete(0, tk.END)
        email_entry.delete(0, tk.END)

        name_entry.insert(0, selected_contact)
        phone_entry.insert(0, contacts[selected_contact]["phone"])
        email_entry.insert(0, contacts[selected_contact]["email"])
def edit_contact():
    selected_contact = contact_listbox.get(tk.ACTIVE)
    if selected_contact:
        new_name = name_entry.get()
        new_phone = phone_entry.get()
        new_email = email_entry.get()

        if new_name and new_phone:
            contacts[new_name] = {"phone": new_phone, "email": new_email}
            if new_name != selected_contact:
                del contacts[selected_contact]
            update_contact_listbox()
            save_contacts()
            clear_input_fields()
        else:
            messagebox.showwarning("Input Error", "Please enter both Name and Phone.")
    else:
        messagebox.showwarning("Selection Error", "Please select a contact to edit.")
root = tk.Tk()
root.title("Personalized Contact Book")
root.geometry("500x500")
root.config(bg="#f0f0f0")  # Set the background color of the window
contacts = load_contacts()
frame = tk.Frame(root, bg="#ffffff", bd=10)
frame.pack(pady=20, padx=20, fill=tk.BOTH, expand=True)
name_label = tk.Label(frame, text="Name:", bg="#ffffff", font=("Arial", 12), fg="#333333")
name_label.grid(row=0, column=0, padx=10, pady=5, sticky="w")
name_entry = tk.Entry(frame, width=40, font=("Arial", 12), bg="#e6f7ff", fg="#333333", bd=2,
relief="solid")
name_entry.grid(row=0, column=1, padx=10, pady=5)

phone_label = tk.Label(frame, text="Phone Number:", bg="#ffffff", font=("Arial", 12), fg="#333333")
phone_label.grid(row=1, column=0, padx=10, pady=5, sticky="w")
phone_entry = tk.Entry(frame, width=40, font=("Arial", 12), bg="#e6f7ff", fg="#333333", bd=2,
relief="solid")
phone_entry.grid(row=1, column=1, padx=10, pady=5)

email_label = tk.Label(frame, text="Email Address (Optional):", bg="#ffffff", font=("Arial", 12),
fg="#333333")
email_label.grid(row=2, column=0, padx=10, pady=5, sticky="w")
email_entry = tk.Entry(frame, width=40, font=("Arial", 12), bg="#e6f7ff", fg="#333333", bd=2,
relief="solid")
email_entry.grid(row=2, column=1, padx=10, pady=5)

add_button = tk.Button(root, text="Add Contact", width=20, bg="#4CAF50", fg="white", font=("Arial",
12), command=add_contact)
add_button.pack(pady=10)

edit_button = tk.Button(root, text="Edit Contact", width=20, bg="#ff9800", fg="white", font=("Arial",
12), command=edit_contact)
edit_button.pack(pady=5)

delete_button = tk.Button(root, text="Delete Contact", width=20, bg="#f44336", fg="white",
font=("Arial", 12), command=delete_contact)
delete_button.pack(pady=5)
```

```
search_label = tk.Label(root, text="Search Contact:", bg="#f0f0f0", font=("Arial", 12), fg="#333333")
search_label.pack(pady=5)
search_entry = tk.Entry(root, width=40, font=("Arial", 12), bg="#e6f7ff", fg="#333333", bd=2,
relief="solid")
search_entry.pack(pady=5)

search_button = tk.Button(root, text="Search", width=20, bg="#2196F3", fg="white", font=("Arial", 12),
command=search_contact)
search_button.pack(pady=10)

contact_listbox = tk.Listbox(root, height=10, width=50, font=("Arial", 12), bg="#f0f0f0", fg="#333333",
selectmode=tk.SINGLE, bd=2, relief="solid")
contact_listbox.pack(pady=10)

contact_listbox.bind("<ButtonRelease-1>", view_contact_details)

update_contact_listbox()

root.mainloop()
```

5. Output:


Personalized Contact Book

Name: Suraj Kumar
Phone Number: 7644076035
Email Address (Optional): kp344664@gmail.com

Suraj Kumar

5. Learning outcomes (What I have learnt):

- Learn how to create and manage basic Tkinter widgets such as buttons, labels, entry fields, listboxes, and frames.
- Understand the use of grid and pack layout managers to arrange widgets within a window.
- Learn how to bind events (e.g., selecting an item in a listbox) to functions that update the UI.
- Learn how to add new records to a data structure (in this case, a dictionary) and save them to a file.
- Learn how to work with dictionaries to store complex data structures (e.g., a contact's name as the key, with details like phone and email as the values).
- Learn how to design simple but effective user interfaces that are both functional and visually appealing. This includes organizing UI components logically and applying basic styling.