Python Project 09: Contact Book

1. Project Overview

A **Contact Book** is a desktop application for storing and managing contact information.

Users can:

- Add contacts
- View contacts
- Search contacts
- Update contacts
- Delete contacts
- Store names, phone numbers, emails, and addresses
- Keep contacts saved between application sessions

The project uses **Tkinter** for the interface and **SQLite** for persistent data storage. Tkinter provides Python's standard GUI toolkit, while SQLite can store application data locally without requiring a separate database server.

---

2. Features

- Add contact
- View all contacts
- Search contacts
- Edit contact
- Delete contact
- Name
- Phone number
- Email

- Address
- SQLite database
- Input validation
- Contact table
- Clean dark UI
- No external packages

---

3. Technologies

- Python 3
- Tkinter
- `tkinter.ttk`
- SQLite
- `sqlite3`
- Object Oriented Programming

The `ttk.Treeview` widget is used to display contacts in columns.

---

4. Folder Structure

```
contact_book/
□□□ main.py
□□□ ui.py
□□□ database.py
□□□ contacts.db
```

File Responsibilities

File	Responsibility
main.py	Starts the application
ui.py	Handles the graphical interface
database.py	Handles SQLite operations
contacts.db	Stores contact information

The database file is created automatically when the application runs.

---

5. How the Project Works

User enters contact
↓
Tkinter Frontend
↓
Input Validation
↓
Contact Database
↓
SQLite
↓
contacts.db
↓
Contact displayed in table

For searching:

Search Text

Database Query

Matching Contacts

Treeview

---

6. Complete Backend Code

database.py

```
import sqlite3


class ContactDatabase:

    def __init__(self, database_name="contacts.db"):
        self.database_name = database_name
        self.create_table()

    def connect(self):
        return sqlite3.connect(self.database_name)

    def create_table(self):

        connection = self.connect()
        cursor = connection.cursor()

        cursor.execute("""
```

```
        CREATE TABLE IF NOT EXISTS contacts (
            id INTEGER PRIMARY KEY AUTOINCREMENT,
            name TEXT NOT NULL,
            phone TEXT NOT NULL,
            email TEXT,
            address TEXT
        )
    """)

    connection.commit()
    connection.close()

def add_contact(
    self,
    name,
    phone,
    email,
    address
):

    connection = self.connect()
    cursor = connection.cursor()

    cursor.execute("""
        INSERT INTO contacts
        (name, phone, email, address)
        VALUES (?, ?, ?, ?)
    """, (
        name,
        phone,
        email,
        address
    ))
```

```
        connection.commit()
        connection.close()

    def get_contacts(self):

        connection = self.connect()
        cursor = connection.cursor()

        cursor.execute("""
            SELECT id, name, phone, email, address
            FROM contacts
            ORDER BY name COLLATE NOCASE
        """)

        contacts = cursor.fetchall()

        connection.close()

        return contacts

    def search_contacts(self, search_text):

        connection = self.connect()
        cursor = connection.cursor()

        search_pattern = f"%{search_text}%"

        cursor.execute("""
            SELECT id, name, phone, email, address
            FROM contacts
            WHERE name LIKE ?
               OR phone LIKE ?
               OR email LIKE ?
               OR address LIKE ?
```

```
            ORDER BY name COLLATE NOCASE
        """, (
            search_pattern,
            search_pattern,
            search_pattern,
            search_pattern
        ))

        contacts = cursor.fetchall()

        connection.close()

        return contacts

    def get_contact(self, contact_id):

        connection = self.connect()
        cursor = connection.cursor()

        cursor.execute("""
            SELECT id, name, phone, email, address
            FROM contacts
            WHERE id = ?
        """, (contact_id,))

        contact = cursor.fetchone()

        connection.close()

        return contact

    def update_contact(
        self,
        contact_id,
```

```
        name,
        phone,
        email,
        address
    ):

        connection = self.connect()
        cursor = connection.cursor()

        cursor.execute("""
            UPDATE contacts
            SET name = ?,
                phone = ?,
                email = ?,
                address = ?
            WHERE id = ?
        """, (
            name,
            phone,
            email,
            address,
            contact_id
        ))

        connection.commit()
        connection.close()

    def delete_contact(self, contact_id):

        connection = self.connect()
        cursor = connection.cursor()

        cursor.execute("""
            DELETE FROM contacts
```

```
            WHERE id = ?
        """, (contact_id,))

        connection.commit()
        connection.close()
```

The SQL queries use `?` placeholders rather than inserting user input directly into SQL strings. Python's SQLite documentation recommends parameter substitution for values and warns against constructing SQL with string formatting because of SQL injection risks.

---

7. Complete Frontend Code

ui.py

```
import tkinter as tk
from tkinter import ttk, messagebox

from database import ContactDatabase


class ContactBookUI:

    BG = "#0F172A"
    CARD = "#1E293B"
    INPUT = "#334155"
    TEXT = "#F8FAFC"
    MUTED = "#94A3B8"
    ACCENT = "#38BDF8"
    DANGER = "#EF4444"

    def __init__(self, root):
```

```
        self.root = root

        self.root.title("Contact Book")
        self.root.geometry("950x650")
        self.root.minsize(850, 600)
        self.root.configure(bg=self.BG)

        self.database = ContactDatabase()

        self.selected_contact_id = None

        self.name_var = tk.StringVar()
        self.phone_var = tk.StringVar()
        self.email_var = tk.StringVar()
        self.address_var = tk.StringVar()
        self.search_var = tk.StringVar()

        self.build_ui()
        self.load_contacts()

    def build_ui(self):

        main = tk.Frame(
            self.root,
            bg=self.BG,
            padx=25,
            pady=20
        )

        main.pack(
            fill="both",
            expand=True
        )
```

```
        # Header

        tk.Label(
            main,
            text="Contact Book",
            font=("Segoe UI", 24, "bold"),
            bg=self.BG,
            fg=self.TEXT
        ).pack(anchor="w")

        tk.Label(
            main,
            text="Store and manage your contacts",
            font=("Segoe UI", 10),
            bg=self.BG,
            fg=self.MUTED
        ).pack(
            anchor="w",
            pady=(0, 20)
        )

        # Form card

        form_card = tk.Frame(
            main,
            bg=self.CARD,
            padx=15,
            pady=15
        )

        form_card.pack(
            fill="x",
            pady=(0, 15)
        )
```

```
        # Name

        tk.Label(
            form_card,
            text="Name",
            bg=self.CARD,
            fg=self.TEXT,
            font=("Segoe UI", 9, "bold")
        ).grid(
            row=0,
            column=0,
            sticky="w",
            padx=5
        )

        self.name_entry = tk.Entry(
            form_card,
            textvariable=self.name_var,
            bg=self.INPUT,
            fg=self.TEXT,
            insertbackground=self.TEXT,
            relief="flat"
        )

        self.name_entry.grid(
            row=1,
            column=0,
            sticky="ew",
            padx=5,
            pady=(5, 10),
            ipady=7
        )
```

```
        # Phone

        tk.Label(
            form_card,
            text="Phone",
            bg=self.CARD,
            fg=self.TEXT,
            font=("Segoe UI", 9, "bold")
        ).grid(
            row=0,
            column=1,
            sticky="w",
            padx=5
        )

        self.phone_entry = tk.Entry(
            form_card,
            textvariable=self.phone_var,
            bg=self.INPUT,
            fg=self.TEXT,
            insertbackground=self.TEXT,
            relief="flat"
        )

        self.phone_entry.grid(
            row=1,
            column=1,
            sticky="ew",
            padx=5,
            pady=(5, 10),
            ipady=7
        )

        # Email
```

```
        tk.Label(
            form_card,
            text="Email",
            bg=self.CARD,
            fg=self.TEXT,
            font=("Segoe UI", 9, "bold")
        ).grid(
            row=0,
            column=2,
            sticky="w",
            padx=5
        )

        self.email_entry = tk.Entry(
            form_card,
            textvariable=self.email_var,
            bg=self.INPUT,
            fg=self.TEXT,
            insertbackground=self.TEXT,
            relief="flat"
        )

        self.email_entry.grid(
            row=1,
            column=2,
            sticky="ew",
            padx=5,
            pady=(5, 10),
            ipady=7
        )

        # Address
```

```
        tk.Label(
            form_card,
            text="Address",
            bg=self.CARD,
            fg=self.TEXT,
            font=("Segoe UI", 9, "bold")
        ).grid(
            row=0,
            column=3,
            sticky="w",
            padx=5
        )

        self.address_entry = tk.Entry(
            form_card,
            textvariable=self.address_var,
            bg=self.INPUT,
            fg=self.TEXT,
            insertbackground=self.TEXT,
            relief="flat"
        )

        self.address_entry.grid(
            row=1,
            column=3,
            sticky="ew",
            padx=5,
            pady=(5, 10),
            ipady=7
        )

        # Buttons

        tk.Button(
```

```
            form_card,
            text="Add Contact",
            command=self.add_contact,
            bg=self.ACCENT,
            fg=self.BG,
            activebackground="#7DD3FC",
            activeforeground=self.BG,
            font=("Segoe UI", 9, "bold"),
            relief="flat",
            cursor="hand2"
        ).grid(
            row=2,
            column=0,
            padx=5,
            ipadx=10,
            ipady=6,
            sticky="ew"
        )

        tk.Button(
            form_card,
            text="Update Contact",
            command=self.update_contact,
            bg=self.INPUT,
            fg=self.TEXT,
            activebackground="#475569",
            activeforeground=self.TEXT,
            font=("Segoe UI", 9, "bold"),
            relief="flat",
            cursor="hand2"
        ).grid(
            row=2,
            column=1,
            padx=5,
```

```
            ipadx=10,
            ipady=6,
            sticky="ew"
        )

        tk.Button(
            form_card,
            text="Clear",
            command=self.clear_form,
            bg=self.INPUT,
            fg=self.TEXT,
            activebackground="#475569",
            activeforeground=self.TEXT,
            font=("Segoe UI", 9, "bold"),
            relief="flat",
            cursor="hand2"
        ).grid(
            row=2,
            column=2,
            padx=5,
            ipadx=10,
            ipady=6,
            sticky="ew"
        )

        tk.Button(
            form_card,
            text="Delete Contact",
            command=self.delete_contact,
            bg=self.DANGER,
            fg=self.TEXT,
            activebackground="#F87171",
            activeforeground=self.TEXT,
            font=("Segoe UI", 9, "bold"),
```

```
            relief="flat",
            cursor="hand2"
        ).grid(
            row=2,
            column=3,
            padx=5,
            ipadx=10,
            ipady=6,
            sticky="ew"
        )

        for column in range(4):
            form_card.columnconfigure(
                column,
                weight=1
            )

        # Search

        search_frame = tk.Frame(
            main,
            bg=self.BG
        )

        search_frame.pack(
            fill="x",
            pady=(0, 15)
        )

        tk.Label(
            search_frame,
            text="Search:",
            bg=self.BG,
            fg=self.TEXT,
```

```
            font=("Segoe UI", 10, "bold")
        ).pack(
            side="left",
            padx=(0, 10)
        )

        search_entry = tk.Entry(
            search_frame,
            textvariable=self.search_var,
            bg=self.INPUT,
            fg=self.TEXT,
            insertbackground=self.TEXT,
            relief="flat"
        )

        search_entry.pack(
            side="left",
            fill="x",
            expand=True,
            ipady=7
        )

        search_entry.bind(
            "<KeyRelease>",
            self.search_contacts
        )

        tk.Button(
            search_frame,
            text="Clear Search",
            command=self.clear_search,
            bg=self.INPUT,
            fg=self.TEXT,
            activebackground="#475569",
```

```
            activeforeground=self.TEXT,
            font=("Segoe UI", 9, "bold"),
            relief="flat",
            cursor="hand2"
        ).pack(
            side="right",
            padx=(10, 0),
            ipadx=10,
            ipady=5
        )

        # Table

        table_frame = tk.Frame(
            main,
            bg=self.CARD
        )

        table_frame.pack(
            fill="both",
            expand=True
        )

        columns = (
            "id",
            "name",
            "phone",
            "email",
            "address"
        )

        self.tree = ttk.Treeview(
            table_frame,
            columns=columns,
```

```
            show="headings"
        )

        self.tree.heading(
            "id",
            text="ID"
        )

        self.tree.heading(
            "name",
            text="Name"
        )

        self.tree.heading(
            "phone",
            text="Phone"
        )

        self.tree.heading(
            "email",
            text="Email"
        )

        self.tree.heading(
            "address",
            text="Address"
        )

        self.tree.column(
            "id",
            width=50,
            anchor="center"
        )
```

```
        self.tree.column(
            "name",
            width=160
        )

        self.tree.column(
            "phone",
            width=150
        )

        self.tree.column(
            "email",
            width=220
        )

        self.tree.column(
            "address",
            width=250
        )

        scrollbar = ttk.Scrollbar(
            table_frame,
            orient="vertical",
            command=self.tree.yview
        )

        self.tree.configure(
            yscrollcommand=scrollbar.set
        )

        self.tree.pack(
            side="left",
            fill="both",
            expand=True
```

```
        )

        scrollbar.pack(
            side="right",
            fill="y"
        )

        self.tree.bind(
            "<<TreeviewSelect>>",
            self.select_contact
        )

    def add_contact(self):

        name = self.name_var.get().strip()
        phone = self.phone_var.get().strip()
        email = self.email_var.get().strip()
        address = self.address_var.get().strip()

        if not name:
            messagebox.showerror(
                "Invalid Input",
                "Please enter a name."
            )
            return

        if not phone:
            messagebox.showerror(
                "Invalid Input",
                "Please enter a phone number."
            )
            return

        self.database.add_contact(
```

```
            name,
            phone,
            email,
            address
        )

        self.clear_form()
        self.load_contacts()

    def load_contacts(self, contacts=None):

        for item in self.tree.get_children():
            self.tree.delete(item)

        if contacts is None:
            contacts = self.database.get_contacts()

        for contact in contacts:

            self.tree.insert(
                "",
                "end",
                values=contact
            )

    def select_contact(self, event=None):

        selected = self.tree.selection()

        if not selected:
            return

        item = self.tree.item(
            selected[0]
```

```
        )

        values = item["values"]

        if not values:
            return

        self.selected_contact_id = values[0]

        self.name_var.set(values[1])
        self.phone_var.set(values[2])
        self.email_var.set(values[3])
        self.address_var.set(values[4])

    def update_contact(self):

        if self.selected_contact_id is None:
            messagebox.showwarning(
                "No Selection",
                "Please select a contact first."
            )
            return

        name = self.name_var.get().strip()
        phone = self.phone_var.get().strip()
        email = self.email_var.get().strip()
        address = self.address_var.get().strip()

        if not name:
            messagebox.showerror(
                "Invalid Input",
                "Please enter a name."
            )
            return
```

```
        if not phone:
            messagebox.showerror(
                "Invalid Input",
                "Please enter a phone number."
            )
            return

        self.database.update_contact(
            self.selected_contact_id,
            name,
            phone,
            email,
            address
        )

        self.clear_form()
        self.load_contacts()

    def delete_contact(self):

        if self.selected_contact_id is None:
            messagebox.showwarning(
                "No Selection",
                "Please select a contact first."
            )
            return

        confirm = messagebox.askyesno(
            "Delete Contact",
            "Are you sure you want to delete this contact?"
        )

        if not confirm:
```

```
            return

        self.database.delete_contact(
            self.selected_contact_id
        )

        self.clear_form()
        self.load_contacts()

    def search_contacts(self, event=None):

        search_text = self.search_var.get().strip()

        if not search_text:
            self.load_contacts()
            return

        contacts = self.database.search_contacts(
            search_text
        )

        self.load_contacts(contacts)

    def clear_search(self):

        self.search_var.set("")
        self.load_contacts()

    def clear_form(self):

        self.selected_contact_id = None

        self.name_var.set("")
        self.phone_var.set("")
```

```
        self.email_var.set("")
        self.address_var.set("")

        for item in self.tree.selection():
            self.tree.selection_remove(item)

        self.name_entry.focus()
```

---

8. Main Application Code

main.py

```
import tkinter as tk

from ui import ContactBookUI


def main():

    root = tk.Tk()

    ContactBookUI(root)

    root.mainloop()


if __name__ == "__main__":
    main()
```

---

9. How Frontend and Backend Connect

When the user clicks **Add Contact**, the frontend calls:

```
self.database.add_contact(
    name,
    phone,
    email,
    address
)
```

The database layer then stores the information in SQLite.

The complete flow is:

```
Tkinter Form
    
User Input
    
Validation
    
ContactDatabase
    
SQLite
    
contacts.db
```

When a contact is selected:

```
Treeview
  
Selected Contact ID
  
Form Fields
  
```

User Edits Information

Update Contact

SQLite UPDATE

For searching:

Search Box

search_contacts()

SQLite LIKE Query

Matching Contacts

Treeview

---

10. How to Run

Open the terminal inside the project folder:

```
cd contact_book
```

Run:

```
python main.py
```

The application will automatically create:

```
contacts.db
```

when it runs for the first time.

No external packages are required.

---

11. Using the Application

Add Contact

Enter:

```
Name: Hafsa
Phone: 03001234567
Email: hafsa@example.com
Address: Lahore
```

Click:

```
Add Contact
```

The contact will appear in the table.

Search Contact

Type something into the search field:

```
Hafsa
```

The table will automatically show matching contacts.

Search works across:

- Name
- Phone
- Email
- Address

Update Contact

1. Select a contact from the table.
2. Change the information.
3. Click **Update Contact**.

Delete Contact

1. Select a contact.
2. Click **Delete Contact**.
3. Confirm deletion.

Clear Form

Click:

`Clear`

to remove the current form values and start entering a new contact.

---

12. Basic Testing

Test 1: Add Contact

Enter:

```
Name: Ali
Phone: 03001234567
```

Email: ali@example.com
Address: Gujranwala

Expected result:

The contact appears in the table.

Test 2: Search

Search:

Ali

Expected result:

Only matching contacts are displayed.

Test 3: Update

Select a contact and change the phone number.

Click:

Update Contact

Expected result:

The updated information appears in the table.

Test 4: Delete

Select a contact and click:

`Delete Contact`

Expected result:

The contact is removed from the table and database.

Test 5: Persistence

1. Add a contact.
2. Close the application.
3. Run it again.

Expected result:

The contact is still available because it is stored in SQLite.

---

13. Small Improvements

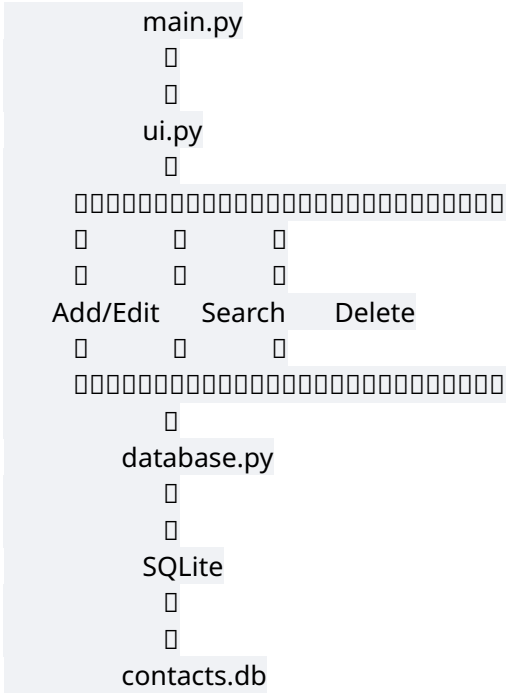
The project can later be extended with:

- Contact groups
- Favorites
- Profile pictures
- Date of birth
- Multiple phone numbers
- Notes
- Sorting
- Advanced filters
- Import contacts from CSV
- Export contacts to CSV

- Birthday reminders
- Backup and restore
- Password-protected contact book
- Cloud synchronization

---

14. Project Architecture



The main application pattern is:

User Input → Validation → Business Logic → Database → UI

The separation between the Tkinter interface and database layer makes the project easier to extend later. `sqlite3` supports parameterized queries, and `ttk.Treeview` is designed for displaying data in multiple columns, making both suitable for this project.

Visit haas.dev for more resources and guides.