Codeboosters Tech
team_codeboosters
www.codeboosters.in

🧠 SECTION 1: IMPORTS

```python
import sqlite3
```

- `import` is a Python **keyword** that brings in external modules.
- `sqlite3` is a built-in Python module for working with SQLite databases (a lightweight database engine).
- Purpose: Allows interaction with an SQLite `.db` file (no server needed).

```python
from tkinter import *
```

- `from module import *` imports all functions, classes, and constants from `tkinter`.
- `tkinter` is Python's standard GUI (Graphical User Interface) toolkit.
- `*` brings everything (like `Button`, `Label`, `Entry`, etc.) into the global namespace for easier usage.

🗃️ SECTION 2: DATABASE SETUP

```python
conn = sqlite3.connect('contacts.db')
```

- Connects to (or creates) a file called `contacts.db`.

- `conn` is the database connection object.

```python
c = conn.cursor()
```

- `cursor()` gives us a cursor object to execute SQL commands like SELECT, INSERT, DELETE, etc.

```python
c.execute("CREATE TABLE IF NOT EXISTS contacts (name TEXT, phone TEXT)")
```

- `CREATE TABLE IF NOT EXISTS` is an SQL command to create a table only if it doesn't exist.
- `contacts` is the table name.
- `name TEXT, phone TEXT` defines two text-type fields (columns).

```python
conn.commit()
```

- Saves changes to the database.

```python
print("Database setup complete!")
```

- Prints a message in the console for confirmation.

🧩 SECTION 3: FUNCTION DEFINITIONS

🔸 `add()` – Adds a new contact

```python
def add():
```

- `def` is a Python keyword to define a function.

```python
    if name.get() and phone.get():
```

- Checks that both fields are **not empty**.
- `name.get()` / `phone.get()` gets text from the Tkinter Entry widgets.

```python
        c.execute("INSERT INTO contacts VALUES (?, ?)", (name.get(), phone.get()))
```

- **SQL INSERT**: Adds values to the table.
- `?` placeholders help prevent SQL injection.

```python
        conn.commit()
```

- Saves the new contact.

```python
        show()
        clear()
```

- Refreshes the contact list and clears form fields.

```python
    print("Contact added successfully!")
```

- Logs a message to the console.

🔸 `show()` – Displays all contacts

```python
def show():
    listbox.delete(0, END)
```

- Clears the `Listbox` before reloading.
- `0` **to** `END` means delete all entries.

```python
    for row in c.execute("SELECT rowid, name, phone FROM contacts"):
        listbox.insert(END, f"{row[0]}. {row[1]} - {row[2]}")
```

- `rowid`: a special SQLite auto-generated ID (used here as the unique identifier).
- Shows each contact in a readable format: `1. John - 12345`.

```python
    print("Contacts displayed successfully!")
```

🔸 `delete()` – Deletes selected contact

```python
def delete():
    selected = listbox.get(ACTIVE)
```

- `ACTIVE` gets the currently selected item from the listbox.

```python
    if selected:
```

```
        rowid = selected.split('.')[0]
```

- Extracts the contact's `rowid` (before the dot) from the displayed string.

```python
        c.execute("DELETE FROM contacts WHERE rowid=?", (rowid,))
        conn.commit()
        show()
        clear()
```

- Deletes from the database and refreshes the UI.

```python
        print("Contact deleted successfully!")
```

🔸 `select()` – Fills form when contact is clicked

```python
def select():
    selected = listbox.get(ACTIVE)
```

- Gets the selected line.

```python
    if selected:
        rowid, rest = selected.split('.', 1)
        contact = rest.strip().split(' - ')
```

- Splits the line to extract:
  - `rowid`
  - `name`

- `phone`

```python
        name.delete(0, END)
        phone.delete(0, END)
        name.insert(0, contact[0])
        phone.insert(0, contact[1])
        id_var.set(rowid)
```

- Clears and fills the Entry widgets with selected contact data.
- Stores the `rowid` into a **hidden variable** `id_var` to use during update.

```python
        print("Contact selected successfully!")
```

🔸 `update()` – Updates existing contact

```python
def update():
    if id_var.get():
        c.execute("UPDATE contacts SET name=?, phone=? WHERE rowid=?",
                  (name.get(), phone.get(), id_var.get()))
```

- Uses SQL UPDATE to modify the selected contact using stored `rowid`.

```python
        conn.commit()
        show()
        clear()
        print("Contact updated successfully!")
```

🔸 `clear()` – Resets the form fields

```python
def clear():
    name.delete(0, END)
    phone.delete(0, END)
    id_var.set("")
    print("Form cleared successfully!")
```

- Clears input boxes and resets hidden `rowid`.

🧱 SECTION 4: UI Setup with Tkinter

```python
root = Tk()
root.title("Simple Contact App with Edit")
```

- Creates the main window.
- `Tk()` is the main root window object.
- `.title()` sets the window title.

```python
id_var = StringVar()
```

- Tkinter variable to store the contact's hidden `rowid`.

```python
Label(root, text="Name").grid(row=0, column=0)
name = Entry(root)
name.grid(row=0, column=1)
```

- Creates a label and an entry field for Name.

- `.grid()` is a layout manager that positions widgets in a table/grid format.

Same applies for Phone field.

```python
Button(root, text="Add", command=add).grid(row=0, column=2)
Button(root, text="Update", command=update).grid(row=1, column=2)
Button(root, text="Delete", command=delete).grid(row=2, column=2)
```

- Buttons that call respective functions when clicked.

```python
listbox = Listbox(root, width=40)
listbox.grid(row=3, column=0, columnspan=3, pady=10)
listbox.bind('<<ListboxSelect>>', lambda e: select())
```

- Listbox shows contact list.
- `.bind()` listens for when a contact is clicked and calls `select()`.

```python
show()
root.mainloop()
```

- `show()` loads the data initially.
- `mainloop()` keeps the window open and waits for user interaction.

✅ Keywords Summary

Keyword	Meaning
`import`	Loads external Python modules

Keyword	Meaning
`from`	Imports specific parts from a module
`def`	Declares a function
`if`	Conditional statement
`for`	Loop through items
`in`	Checks membership or iterates
`lambda`	Creates a small anonymous function
`END`	Tkinter constant representing the end of a listbox or entry

✅ Complete Code with Comments

python

```python
import sqlite3                  # For using SQLite database
from tkinter import *           # For creating GUI using Tkinter

# -------------------------
# DATABASE SETUP
# -------------------------
conn = sqlite3.connect('contacts.db')      # Create/connect to SQLite database file
c = conn.cursor()                          # Create a cursor to execute SQL commands
c.execute("CREATE TABLE IF NOT EXISTS contacts (name TEXT, phone TEXT)")  # Create table if not exists
conn.commit()                              # Save changes to DB
print("Database setup complete!")
```

```
# -------------------------
# FUNCTION TO ADD A CONTACT
# -------------------------
def add():
    if name.get() and phone.get():   # Make sure both fields are not empty
        c.execute("INSERT INTO contacts VALUES (?, ?)", (name.get(), phone.get()))
# Insert values
        conn.commit()                # Save to DB
        show()                       # Refresh the contact list
        clear()                      # Clear form fields
        print("Contact added successfully!")

# -------------------------
# FUNCTION TO SHOW CONTACTS
# -------------------------
def show():
    listbox.delete(0, END)           # Clear listbox
    for row in c.execute("SELECT rowid, name, phone FROM contacts"):  # Fetch all
contacts
        listbox.insert(END, f"{row[0]}. {row[1]} - {row[2]}")  # Display nicely
    print("Contacts displayed successfully!")

# -------------------------
# FUNCTION TO DELETE CONTACT
# -------------------------
def delete():
    selected = listbox.get(ACTIVE)   # Get the selected contact
    if selected:
        rowid = selected.split('.')[0]  # Extract rowid (contact ID)
        c.execute("DELETE FROM contacts WHERE rowid=?", (rowid,))  # Delete by rowid
        conn.commit()
        show()
        clear()
        print("Contact deleted successfully!")

# -------------------------
# FUNCTION TO SELECT CONTACT (FOR EDIT)
# -------------------------
def select():
    selected = listbox.get(ACTIVE)
    if selected:
```

```
        rowid, rest = selected.split('.', 1)           # Split rowid from the rest
        contact = rest.strip().split(' - ')            # Split into name and phone
        name.delete(0, END)                            # Clear name field
        phone.delete(0, END)                           # Clear phone field
        name.insert(0, contact[0])                     # Fill with selected name
        phone.insert(0, contact[1])                    # Fill with selected phone
        id_var.set(rowid)                              # Store rowid for update
        print("Contact selected successfully!")

# -------------------------
# FUNCTION TO UPDATE CONTACT
# -------------------------
def update():
    if id_var.get():   # Only update if a contact is selected
        c.execute("UPDATE contacts SET name=?, phone=? WHERE rowid=?",
                  (name.get(), phone.get(), id_var.get()))  # Update data
        conn.commit()
        show()
        clear()
        print("Contact updated successfully!")

# -------------------------
# FUNCTION TO CLEAR FORM FIELDS
# -------------------------
def clear():
    name.delete(0, END)    # Clear name field
    phone.delete(0, END)   # Clear phone field
    id_var.set("")         # Clear stored ID
    print("Form cleared successfully!")

# -------------------------
# GUI SETUP
# -------------------------
root = Tk()                            # Create main window
root.title("Simple Contact App with Edit")  # Set window title

id_var = StringVar()                   # To store contact rowid (hidden)

# Labels and input fields
Label(root, text="Name").grid(row=0, column=0)
name = Entry(root)
name.grid(row=0, column=1)
```

```
Label(root, text="Phone").grid(row=1, column=0)
phone = Entry(root)
phone.grid(row=1, column=1)

# Buttons
Button(root, text="Add", command=add).grid(row=0, column=2)
Button(root, text="Update", command=update).grid(row=1, column=2)
Button(root, text="Delete", command=delete).grid(row=2, column=2)

# Listbox to show contacts
listbox = Listbox(root, width=40)
listbox.grid(row=3, column=0, columnspan=3, pady=10)
listbox.bind('<<ListboxSelect>>', lambda e: select())  # Bind click to select()

# Load contacts initially
show()

# Start GUI loop
root.mainloop()
```

💡 Student-Friendly Notes:

- Simple UI using only labels, entries, buttons, and a listbox.
- Only uses basic Python concepts and SQL commands (`INSERT` , `SELECT` , `DELETE` , `UPDATE`).
- Great for learning how **logic connects UI and databases**.

✅ Code Block:

```
python
```

```
if selected:
    rowid, rest = selected.split('.', 1)
    contact = rest.strip().split(' - ')
```

This block is used when a user selects a contact from the listbox.

🔹 Context:

The `Listbox` displays items like:

```
markdown

1. John - 9876543210
2. Alice - 1234567890
```

So a `selected` item looks like:

```
python

"1. John - 9876543210"
```

🔍 Line-by-line Explanation:

1. `if selected:`

- Ensures that something was actually selected before proceeding.

---

2. `rowid, rest = selected.split('.', 1)`

- `selected.split('.', 1)` splits the string into **two parts** at the **first dot**:
  - `"1. John - 9876543210"` → `['1', ' John - 9876543210']`
- `rowid` gets `"1"` → This is the unique **rowid** (used for update/delete).
- `rest` gets `" John - 9876543210"` → The actual contact info.

3. `contact = rest.strip().split(' - ')`

- `rest.strip()` removes extra spaces at the beginning (`' John - 9876543210'` → `'John - 9876543210'`)
- Then, `.split(' - ')` splits it into:
  - `['John', '9876543210']`

So now:

- `contact[0] = 'John'`
- `contact[1] = '9876543210'`

🧠 Why do we do this?

This helps you:

- Get the **ID** of the selected contact (`rowid`)
- Extract the **name** and **phone** values separately for editing

---