🧩 SQLite with Python: Full Guide 💡

SQLite is a lightweight **relational database** built into Python — no server setup required! 💡

🧱 1. 🔗 Import & Connect to a Database

```python
import sqlite3

# Connect to a database file (creates file if it doesn't exist)
conn = sqlite3.connect('my_database.db')

# Create a cursor object to execute SQL commands
cursor = conn.cursor()
```

> 🧠 Tip: Use `:memory:` to create a temporary in-memory DB:

```python
conn = sqlite3.connect(':memory:')
```

🛠️ 2. 🏗️ Create a Table

```python
cursor.execute('''
CREATE TABLE IF NOT EXISTS users (
    id INTEGER PRIMARY KEY AUTOINCREMENT,
    name TEXT NOT NULL,
    age INTEGER,
    email TEXT UNIQUE
)
''')
conn.commit()
```

> 🔐 Use `AUTOINCREMENT` for unique primary keys. ☑️ `conn.commit()` saves the changes.

✍️ 3. Insert Data into the Table

```python
cursor.execute("INSERT INTO users (name, age, email) VALUES (?, ?, ?)",
               ("Darshan", 23, "darshan@example.com"))
conn.commit()
```

> ☑️ Always use `?` placeholders to prevent SQL injection.

You can also insert multiple records:

```
users = [
    ("Asha", 28, "asha@mail.com"),
    ("Ravi", 32, "ravi@mail.com")
]
cursor.executemany("INSERT INTO users (name, age, email) VALUES (?, ?, ?)", users)
conn.commit()
```

🔍 4. Read / Query Data

```
cursor.execute("SELECT * FROM users")
rows = cursor.fetchall()

for row in rows:
    print(row)
```

> 🔁 Use `.fetchall()` to get all rows, or `.fetchone()` for a single record.

Query with a condition:

```
cursor.execute("SELECT name FROM users WHERE age > ?", (25,))
print(cursor.fetchall())
```

✏️ 5. Update Records

```
cursor.execute("UPDATE users SET age = ? WHERE name = ?", (24, "Darshan"))
conn.commit()
```

❌ 6. Delete Records

```
cursor.execute("DELETE FROM users WHERE name = ?", ("Ravi",))
conn.commit()
```

🧹 7. Close Connection

```
conn.close()
```

🔐 8. Use `with` Statement (Auto Close & Commit)

```
with sqlite3.connect('my_database.db') as conn:
    cursor = conn.cursor()
    cursor.execute("SELECT * FROM users")
    print(cursor.fetchall())
```

> 🚀 Best practice: Ensures connection closes automatically.

📚 BONUS: Dict-Like Access to Rows

To access rows as dictionaries:

```
conn.row_factory = sqlite3.Row
cursor = conn.cursor()
cursor.execute("SELECT * FROM users")
row = cursor.fetchone()

print(row["name"])  # dict-style access
```

🧪 Sample Project Use Case: Simple Login System

```
import sqlite3

def login_user(email):
    with sqlite3.connect('my_database.db') as conn:
        cursor = conn.cursor()
        cursor.execute("SELECT * FROM users WHERE email = ?", (email,))
        user = cursor.fetchone()
        if user:
            print(f"Welcome, {user[1]}!")
        else:
            print("User not found.")

login_user("darshan@example.com")
```

🧠 Tips and Gotchas

Tip 💡	Description

Tip 💡	Description
Use `executemany()`	For bulk insertions
Use `?` placeholders	Prevent SQL injection
Always commit	After insert/update/delete
Use `.fetchone()` for single record	Efficient for exact match queries
`.row_factory = sqlite3.Row`	Enables dict-like row access
Keep `conn` and `cursor` inside `with` block	Cleaner and safer

📦 Libraries Built on SQLite

- ◇ `SQLAlchemy` – ORM with SQLite backend
- ◇ `Peewee` – Lightweight ORM
- ◇ `Pandas` – You can read/write to SQLite using `.to_sql()` and `pd.read_sql()`

📜 SQLite Cheat Sheet in Python 🐍

📦 Import & Connect

```
import sqlite3

conn = sqlite3.connect('mydatabase.db')  # Creates or opens DB
cursor = conn.cursor()
```

🏗 Create Table

```
cursor.execute("""
CREATE TABLE IF NOT EXISTS users (
    id INTEGER PRIMARY KEY,
    name TEXT NOT NULL,
    email TEXT UNIQUE
)
""")
conn.commit()
```

➕ Insert Data

```
cursor.execute("INSERT INTO users (name, email) VALUES (?, ?)", ("Alice",
"alice@example.com"))
conn.commit()
```

🔍 Read Data

```
cursor.execute("SELECT * FROM users")
rows = cursor.fetchall()

for row in rows:
    print(row)
```

📝 Update Data

```
cursor.execute("UPDATE users SET email=? WHERE name=?", ("new@example.com",
"Alice"))
conn.commit()
```

❌ Delete Data

```
cursor.execute("DELETE FROM users WHERE name=?", ("Alice",))
conn.commit()
```

🔚 Close Connection

```
conn.close()
```

💼 Build a Mini CRUD App with SQLite + Python CLI

```
import sqlite3

def connect():
    return sqlite3.connect("crud_app.db")

def create_table():
```

```
    with connect() as conn:
        conn.execute('''CREATE TABLE IF NOT EXISTS notes (
            id INTEGER PRIMARY KEY,
            title TEXT NOT NULL,
            content TEXT
        )''')

def add_note(title, content):
    with connect() as conn:
        conn.execute("INSERT INTO notes (title, content) VALUES (?, ?)", (title,
content))

def view_notes():
    with connect() as conn:
        notes = conn.execute("SELECT * FROM notes").fetchall()
        for note in notes:
            print(note)

def update_note(note_id, title, content):
    with connect() as conn:
        conn.execute("UPDATE notes SET title=?, content=? WHERE id=?", (title,
content, note_id))

def delete_note(note_id):
    with connect() as conn:
        conn.execute("DELETE FROM notes WHERE id=?", (note_id,))

# Usage
create_table()
add_note("First Note", "This is a test.")
view_notes()
update_note(1, "Updated Note", "Changed the content.")
delete_note(1)
```

⚙️ Flask + SQLite Integration 💧

```
from flask import Flask, request, jsonify
import sqlite3

app = Flask(__name__)

def connect():
    return sqlite3.connect('flask_users.db')

@app.route('/users', methods=['GET'])
def get_users():
    conn = connect()
    cursor = conn.cursor()
    users = cursor.execute("SELECT * FROM users").fetchall()
```

```
    conn.close()
    return jsonify(users)

@app.route('/users', methods=['POST'])
def add_user():
    data = request.get_json()
    with connect() as conn:
        conn.execute("INSERT INTO users (name, email) VALUES (?, ?)",
(data['name'], data['email']))
    return jsonify({"message": "User added"}), 201

if __name__ == '__main__':
    app.run(debug=True)
```

⚡ FastAPI + SQLite Integration 🪄

```
from fastapi import FastAPI, HTTPException
from pydantic import BaseModel
import sqlite3

app = FastAPI()

class User(BaseModel):
    name: str
    email: str

def connect():
    return sqlite3.connect("fastapi_users.db")

@app.post("/users/")
def add_user(user: User):
    with connect() as conn:
        conn.execute("INSERT INTO users (name, email) VALUES (?, ?)", (user.name,
user.email))
    return {"message": "User created"}

@app.get("/users/")
def get_users():
    conn = connect()
    cursor = conn.cursor()
    users = cursor.execute("SELECT * FROM users").fetchall()
    conn.close()
    return {"users": users}
```

☑ Pro Tips

- Use `with sqlite3.connect()` for auto-closing connections.
- Use `row_factory = sqlite3.Row` for dict-like results.
- `sqlite3` is built-in. No need to install separately.
- For production, consider using SQLAlchemy for ORM.

🎬 YouTube Video Manager App with SQLite

🧠 Objective:

A CLI (Command Line Interface) app to manage a list of YouTube videos using **SQLite** for storage. Perform full CRUD operations:

- ☑ Create
- 📖 Read
- 🛠 Update
- ❌ Delete

📦 Technologies Used:

- `sqlite3` – Built-in Python module for SQLite DB
- `Python` – For scripting
- `Terminal/CLI` – To run and interact

🧱 Database Setup

```
import sqlite3

conn = sqlite3.connect('youtube_videos.db')  # 🎯 Creates or connects to SQLite
database file

cursor = conn.cursor()

cursor.execute('''
    CREATE TABLE IF NOT EXISTS videos (
        id INTEGER PRIMARY KEY,
        name TEXT NOT NULL,
        time TEXT NOT NULL
    )
''')
```

📌 Explanation:

- `sqlite3.connect()` opens the DB file (creates if not exists).
- `cursor.execute()` runs SQL commands.
- `CREATE TABLE IF NOT EXISTS` ensures the table is created only once.

🔍 Features & Functions

1️⃣ List All Videos

```
def list_videos():
    cursor.execute("SELECT * FROM videos")
    for row in cursor.fetchall():
        print(row)
```

📝 Fetches and prints all rows from `videos` table.

2️⃣ Add New Video

```
def add_video(name, time):
    cursor.execute("INSERT INTO videos (name, time) VALUES (?, ?)", (name, time))
    conn.commit()
```

📝 Adds a new record with safe parameterized SQL to prevent SQL injection.

3️⃣ Update Video Details

```
def update_video(video_id, new_name, new_time):
    cursor.execute("UPDATE videos SET name = ?, time = ? WHERE id = ?", (new_name,
new_time, video_id))
    conn.commit()
```

🔄 Modifies video info by `id`.

4️⃣ Delete a Video

```
def delete_video(video_id):
    cursor.execute("DELETE FROM videos WHERE id = ?", (video_id,))
    conn.commit()
```

🗑️ Deletes video from DB by its ID.

🎛️ Main App Loop

```python
def main():
    while True:
        print("\nYoutube manager app with DB")
        print("1. List Videos")
        print("2. Add Videos")
        print("3. Update Videos")
        print("4. Delete Videos")
        print("5. Exit App")

        choice = input("Enter your choice: ")

        if choice == '1':
            list_videos()
        elif choice == '2':
            name = input("Enter the video name: ")
            time = input("Enter the video time: ")
            add_video(name, time)
        elif choice == '3':
            video_id = input("Enter video ID to update: ")
            name = input("Enter the new video name: ")
            time = input("Enter the new video time: ")
            update_video(video_id, name, time)
        elif choice == '4':
            video_id = input("Enter video ID to delete: ")
            delete_video(video_id)
        elif choice == '5':
            break
        else:
            print("Invalid Choice")

    conn.close()

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

📌 Handles all UI and user inputs. Uses a loop to offer options repeatedly.

☑ Sample Output:

```
Youtube manager app with DB
1. List Videos
2. Add Videos
3. Update Videos
4. Delete Videos
5. Exit App
Enter your choice: 2
Enter the video name: Python Decorators
Enter the video time: 12:30
```

🛡️ Best Practices & Enhancements:

☑️ Error Handling:

Add try-except blocks to avoid crashes.

```
try:
    cursor.execute("...")
except sqlite3.Error as e:
    print("Database error:", e)
```

🚀 Feature Upgrades:

- 🔍 Search by name or ID
- ⏳ Sort by video duration or name
- 🧪 Unit Tests using `unittest`
- 🌐 Convert to web app using **Flask** or **FastAPI**
- 📄 Export data to CSV

🧪 Project Structure:

```
youtube_manager/
│
├── youtube_videos.db         # SQLite database file (auto-created)
└── app.py                    # Main Python script
```

💻 Convert to Flask/FastAPI (Bonus):

Flask Example:

```
from flask import Flask, request, jsonify
import sqlite3

app = Flask(__name__)

@app.route('/videos', methods=['GET'])
def get_videos():
    conn = sqlite3.connect('youtube_videos.db')
    cursor = conn.cursor()
    cursor.execute("SELECT * FROM videos")
    videos = cursor.fetchall()
    conn.close()
    return jsonify(videos)
```

```
# Similarly, add POST, PUT, DELETE routes
```
