To run your Flask to do app, follow these quick steps in your **Terminal** or **Command Prompt**:-

1. Organize Your Files

Ensure your project structure looks like this:

text

```
my_project/
├── app.py
└── templates/
    └── index.htm
```

3. Install Dependencies

Install Flask and the SQLAlchemy toolkit via PyPI:

```
pip install flask flask-sqlalchemy
```

4:- creat app.py file code:-

```
from flask import Flask, render_template, request, redirect, url_for

from flask_sqlalchemy import SQLAlchemy


app = Flask(__name__)


# Configures SQLAlchemy to use a local SQLite file named db.sqlite

app.config['SQLALCHEMY_DATABASE_URI'] = 'sqlite:///db.sqlite'

app.config['SQLALCHEMY_TRACK_MODIFICATIONS'] = False

db = SQLAlchemy(app)


# Database Model

class Todo(db.Model):
```

```
    id = db.Column(db.Integer, primary_key=True)
    title = db.Column(db.String(100), nullable=False)
    complete = db.Column(db.Boolean, default=False)

@app.route('/')
def index():
    todo_list = Todo.query.all()
    return render_template('index.html', todo_list=todo_list)

@app.route('/add', methods=['POST'])
def add():
    title = request.form.get('title')
    if title:
        new_todo = Todo(title=title, complete=False)
        db.session.add(new_todo)
        db.session.commit()
    return redirect(url_for('index'))

@app.route('/update/<int:todo_id>')
def update(todo_id):
    todo = Todo.query.filter_by(id=todo_id).first()
    todo.complete = not todo.complete
    db.session.commit()
    return redirect(url_for('index'))
```

```
@app.route('/delete/<int:todo_id>')
def delete(todo_id):
    todo = Todo.query.filter_by(id=todo_id).first()
    db.session.delete(todo)
    db.session.commit()
    return redirect(url_for('index'))

if __name__ == "__main__":
    # The [SQLAlchemy documentation](https://flask-sqlalchemy.palletsprojects.com)
    # recommends creating tables within the app context
    with app.app_context():
        db.create_all()
    app.run(debug=True)
```

templates/index.html file code:-

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>Simple Flask Todo</title>
    <style>
        body { font-family: sans-serif; max-width: 500px; margin: 50px auto; }
        .todo-item { display: flex; justify-content: space-between; margin-bottom: 10px; }
        .completed { text-decoration: line-through; color: gray; }
```

```
    </style>
</head>
<body>
    <h2>My To-Do List</h2>
    <form action="/add" method="POST">
        <input type="text" name="title" placeholder="Add a task..." required>
        <button type="submit">Add</button>
    </form>
    <hr>
    {% for todo in todo_list %}
    <div class="todo-item">
        <span class="{{ 'completed' if todo.complete else '' }}">{{ todo.title }}</span>
        <div>
            <a href="/update/{{ todo.id }}">✓</a>
            <a href="/delete/{{ todo.id }}" style="color:red; margin-left:10px;">X</a>
        </div>
    </div>
    {% endfor %}
</body>
</html>
```

4. Execute the Application

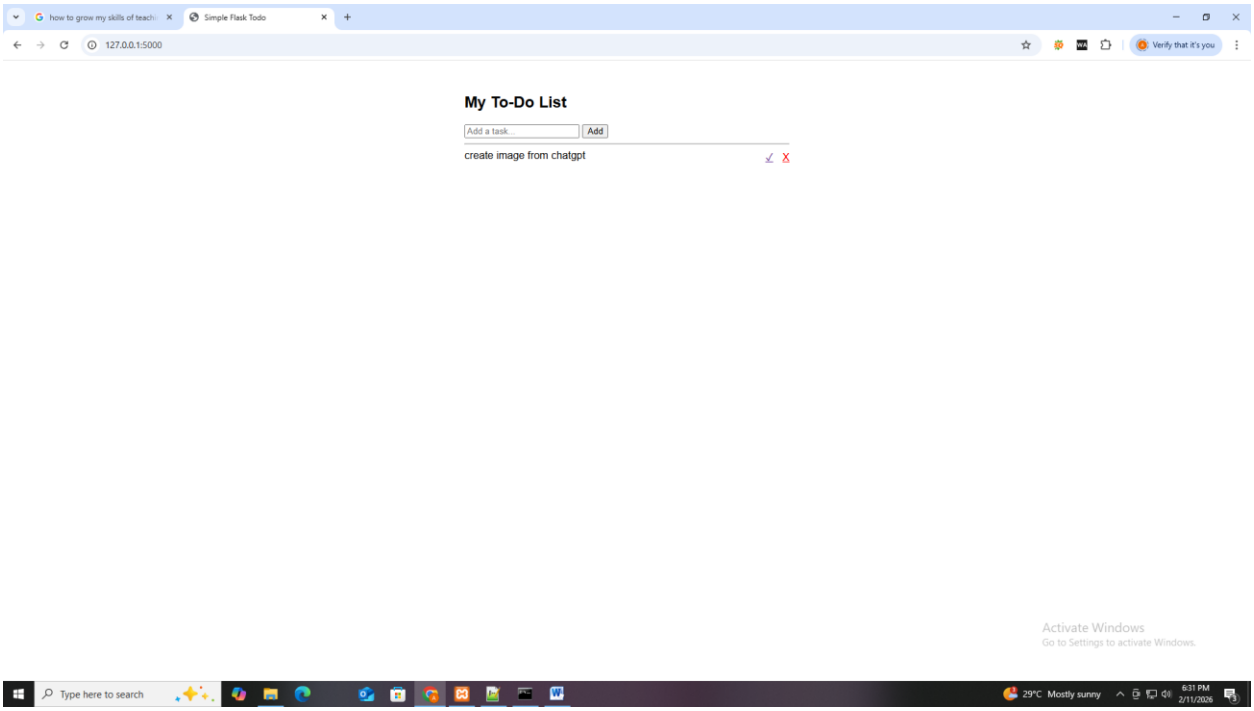
Run the script using the Python interpreter:

```
python app.py
```

5. Access the App

Once you see `* Running on http://127.0.0.1:5000`, open your web browser and go to:

http://127.0.0.1:5000



Here is a line-by-line breakdown Explanation:-

1. Imports & Core Setup

- `from flask import Flask, ...`: Imports the Flask framework and tools to handle HTML templates, user input, and page jumping (redirects).
- `from flask_sqlalchemy import SQLAlchemy`: Imports the SQLAlchemy ORM, which lets you talk to your database using Python classes instead of raw SQL.
- `app = Flask(__name__)`: Creates the actual application object. `__name__` tells Flask where to look for files.

2. Database Configuration

- `app.config['SQLALCHEMY_DATABASE_URI'] = 'sqlite:///db.sqlite'`: Tells Flask to create a local file named `db.sqlite` to store all data.
- `app.config['SQLALCHEMY_TRACK_MODIFICATIONS'] = False`: Disables a high-overhead feature we don't need, saving memory.
- `db = SQLAlchemy(app)`: Links the database tool to your Flask app.

3. The "Model" (Defining Data)

- `class Todo(db.Model)`: Defines a table in the database. Each "instance" of this class will be a row in that table.
- `id = db.Column(...)`: The "Primary Key"—a unique ID for every task (1, 2, 3...).
- `title = db.Column(...)`: A text field (max 100 characters) that cannot be empty (`nullable=False`).
- `complete = db.Column(...)`: A True/False switch to track if a task is done.

4. Routes (The Logic)

- `@app.route('/')`: The "Home" page. When a user visits, `Todo.query.all()` grabs every task from the database and sends them to `index.html`.
- `@app.route('/add', methods=['POST'])`: Only accepts "POST" requests (data sent from a form). It creates a `new_todo` object and uses `db.session.add` + `db.session.commit()` to save it.

- `@app.route('/update/<int:todo_id>')`: Uses a **Dynamic Route**. `<int:todo_id>` captures the ID of the specific task you clicked. It flips the `complete` status from True to False (or vice versa).
- `@app.route('/delete/<int:todo_id>')`: Finds the task by ID and uses `db.session.delete()` to remove it from the database forever.

5. The "Execution" Block

- `if __name__ == "__main__":`: Ensures the app only runs if you execute this specific file.
- `db.create_all()`: The "Magic Line." It looks at your `Todo` class and automatically creates the database file and tables if they don't exist yet.
- `app.run(debug=True)`: Starts the server. `debug=True` means the server restarts automatically whenever you change your code.

  Teacher's Tip: When explaining `db.session.commit()`, compare it to hitting **"Save"** on a Word document. Without it, your changes only exist in temporary memory!

Explanation index.html file :-This HTML file uses **Jinja2**, which is the templating engine for Flask. It allows you to mix standard HTML with Python-like logic.

1. Document Metadata (The "Brain")

- `<!DOCTYPE html>`: Tells the browser this is an **HTML5** document.
- `<html lang="en">`: Sets the primary language to English for search engines and screen readers.
- `<meta charset="UTF-8">`: Ensures all characters (like emojis or special symbols) display correctly.
- `<title>`: The text that appears on the browser tab.

2. CSS Styling (The "Look")

- `max-width: 500px; margin: 50px auto;`: Centers the app on the screen and prevents it from getting too wide.

- `.todo-item { display: flex; ... }`: Uses Flexbox to push the task text to the left and the buttons to the right.
- `.completed { text-decoration: line-through; ... }`: A specific class that crosses out text—we only apply this if the task is finished.

3. The Input Form (The "Action")

- `<form action="/add" method="POST">`: This is the most important part. When the button is clicked, it sends data to the `@app.route('/add')` in your Python file.
- `<input name="title" ... required>`: The `name="title"` is the key Flask uses to find the text (`request.form.get('title')`). `required` prevents empty tasks.

4. Jinja2 Logic (The "Magic")

- `{% for todo in todo_list %}`: This is a Jinja2 loop. It repeats the code inside for every single task found in your database.
- `{{ 'completed' if todo.complete else '' }}`: This is **Conditional Logic**. It checks the database: if `complete` is True, it adds the CSS class that crosses out the text.
- `{{ todo.title }}`: The double curly braces tell Flask to "print" the actual text of the task here.

5. Dynamic Links (The "Navigation")

- `<a href="/update/{{ todo.id }}">`: This creates a unique link for every task. If the task ID is 5, the link becomes `/update/5`.
- `{% endfor %}`: Tells Flask where the loop stops repeating.

Teacher's Tip: Explain to students that HTML is usually **static** (never changes), but by using `{{ }}` and `{% %}`, we make it **dynamic** (it changes based on what is in our database).