🟥 Sequential Workflows in LangGraph

---

1. 📌 Today's Learning Goals

1. Learn how to write **basic LangGraph code** (first hands-on experience).
2. Use that knowledge to build **any sequential workflow** in LangGraph.
   - **Sequential workflow** = tasks connected **linearly**.
   - No branching or parallelism.

---

2. ⚙️ Setup and Installation

- Created a project folder → langgraph_tutorials (in VS Code).
- Steps:
  1. Create virtual environment → myenv.
  2. Activate environment.
  3. Install required libraries:
     - langgraph → workflow framework.
     - langchain → for LLM-related utilities (chat models, prompt templates, loaders, splitters).
     - langchain-openai → to use OpenAI models.
     - python-dotenv → to read API keys.
  4. Test imports with a notebook 0_test_installation.ipynb.
- Coding is done in **Jupyter notebooks** for easy visualization of graphs.

---

3. 🧪 Example 1 – BMI Calculator Workflow

📄 File: 01_bmi_workflow.ipynb

🔹 **Goal:**

- A simple **non-LLM workflow** → input height & weight → calculate BMI → output result.

🔹 **Steps:**

1. **Define State**
   - Create BMIState using TypedDict.
   - Attributes:
     - weight: float
     - height: float
     - bmi: float
2. **Define Graph**
   - Create graph object: graph = StateGraph(BMIState).
   - Add **node** → calculate_bmi.
     - Behind the scenes: each node = Python function.
   - Function logic:
     - Extract weight & height from state.
     - Compute BMI = weight / (height^2).
     - Round to 2 decimals.
     - Update state with BMI.
3. **Add Edges**
   - Start → calculate_bmi → End.
4. **Compile & Execute**
   - Compile graph → workflow = graph.compile().
   - Invoke with initial state: {weight: 80, height: 1.73}.
   - Output = updated state with BMI.
5. **Visualize Graph**
   - Code snippet used to render graph visually in Jupyter.
   - Shows linear flow: **Start → Calculate BMI → End**.

🔹 **Extension:**

- Added **new node → label_bmi**.

        - Classify BMI as: Underweight, Normal, Overweight, Obese.
        - Update state["category"].
    - New graph: **Start → Calculate BMI → Label BMI → End**.

✅ Learned: How to define state, nodes, edges, and run a sequential workflow.

---

5. 🧪 Example 2 – Simple LLM Workflow

📄 File: 02_simple_llm_workflow.ipynb

🔹 **Goal:**

- Build the simplest **LLM-based workflow** → ask a question → get answer.

🔹 **Steps:**

1. **Setup**
    - Import ChatOpenAI from langchain_openai.
    - Load .env file with OpenAI API key.
    - Initialize model: model = ChatOpenAI().
2. **Define State**
    - LLMState with 2 attributes:
        - question: str
        - answer: str
3. **Define Node → llm_qa**
    - Extract question from state.
    - Create prompt: "Answer the following question: {question}".
    - Call model → model.invoke(prompt).
    - Store result in state["answer"].
4. **Add Edges**
    - Start → llm_qa → End.
5. **Compile & Run**
    - Input: {question: "How far is the moon from Earth?"}.
    - Output: Answer stored in state.

✅ Learned: How **LangGraph + LangChain** work together for LLM workflows.

---

6. 🧪 Example 3 – Prompt Chaining Workflow

📄 File: 03_prompt_chaining.ipynb

🔹 **Goal:**

- Demonstrate **prompt chaining** (multiple LLM calls in sequence).
- Task: Generate a blog with outline → blog.

🔹 **Steps:**

1. **Define State** → BlogState
    - Attributes:
        - title: str
        - outline: str
        - content: str
2. **Define Nodes**
    - create_outline(state)
        - Input: title.
        - Prompt: "Generate a detailed outline for a blog on the topic {title}".
        - Output: outline → update state.
    - create_blog(state)
        - Input: title + outline.
        - Prompt: "Write a detailed blog on the title {title} using the following outline: {outline}".
        - Output: blog content → update state.
3. **Add Edges**
    - Start → create_outline → create_blog → End.
4. **Compile & Run**

    - Input: {title: "Rise of AI in India"}.
    - Output:
        - Title: "Rise of AI in India"
        - Outline: Intro, History, Current state, Challenges, Future outlook, Conclusion.
        - Content: Full blog text.

🔹 **Key Insight:**

- With **LangGraph state**, you keep **all intermediate results** (title, outline, content).
- In LangChain **chains**, you usually only get the final result (blog).

✅ Learned: How to chain multiple LLM calls in a linear sequence.

---

7. 📌 Homework / Practice Task

- Extend the **prompt chaining workflow** by adding a **third node**:
    - evaluate_blog(state) → Prompt: "Based on this outline, rate my blog and give an integer score."
    - Update state with score: int.
- This will give practice in:
    - Modifying state.
    - Adding new node.
    - Updating graph edges.

---

8. 🌟 Key Takeaways

- **LangGraph coding basics**:
    - Define State → Define Graph → Add Nodes → Add Edges → Compile → Execute.
- **Sequential workflows** = linear execution path (no branches/parallelism).
- Three workflows demonstrated:

1. **BMI calculator** (non-LLM).
2. **Simple QA** (LLM-based).
3. **Prompt chaining** (multiple LLM calls).

- **Power of LangGraph** = maintains **state across workflow**, enabling easy debugging, evaluation, and access to all intermediate outputs.