Chains:

# Chains

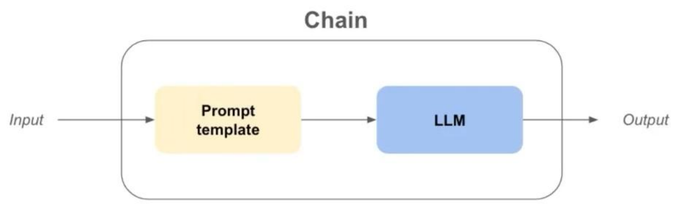
Chains in LangChain are sequences of modular components—like prompts, models, and tools—executed in a predefined flow. They enable complex reasoning by combining multiple steps, such as input transformation, decision logic, and LLM invocation.

## What are Chains in LangChain?

**Definition**

A **Chain** in LangChain is a sequence of components — such as prompts, LLMs, retrievers, tools, or memory — linked together to perform a **complex task** in a structured and reusable way.

Think of a chain as a **workflow pipeline** where each step builds on the previous one to reach a goal — like answering a question, summarizing a document, or generating code.



Prompt, LLM and Chain

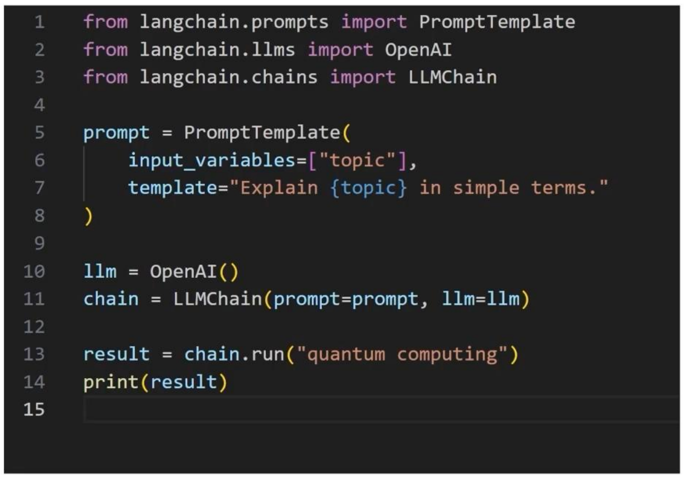
## Why Chains Are Useful

- ❑ Allow **modular composition** of LLM apps
- ❑ Make LLM workflows **reusable and maintainable**
- ❑ Enable complex logic: prompt → LLM → output → transformation → new input
- ❑ Essential for **production-ready** apps beyond just using .invoke() or .predict()

## Types of Chains in LangChain

| Chain Type | Description | Use Case |
|---|---|---|
| **LLMChain** | Most basic chain: prompt → LLM → output | Q&A, text generation, classification |
| **SequentialChain** | Runs multiple chains one after another (output of one is input to next) | Multi-step reasoning tasks |
| **SimpleSequentialChain** | Executes multiple LLMs/prompts sequentially (less config) | Text rewriting → summarization |
| **RouterChain** | Routes input to different chains based on context or conditions | Chatbots with multiple domains |
| **ConversationalChain** | Maintains chat history using memory and context | Chatbots, virtual assistants |
| **RetrievalQAChain** | Combines a retriever (vector DB) with LLM for RAG (Retrieval-Augmented Generation) | Q&A from documents |
| **AgentExecutor** | Advanced use case: Executes a chain of tools + memory + LLM + decision logic | AI Agents (tools + reasoning + memory) |

## Basic Example: LLMChain



```python
from langchain.prompts import PromptTemplate
from langchain.llms import OpenAI
from langchain.chains import LLMChain

prompt = PromptTemplate(
    input_variables=["topic"],
    template="Explain {topic} in simple terms."
)

llm = OpenAI()
chain = LLMChain(prompt=prompt, llm=llm)

result = chain.run("quantum computing")
print(result)

```