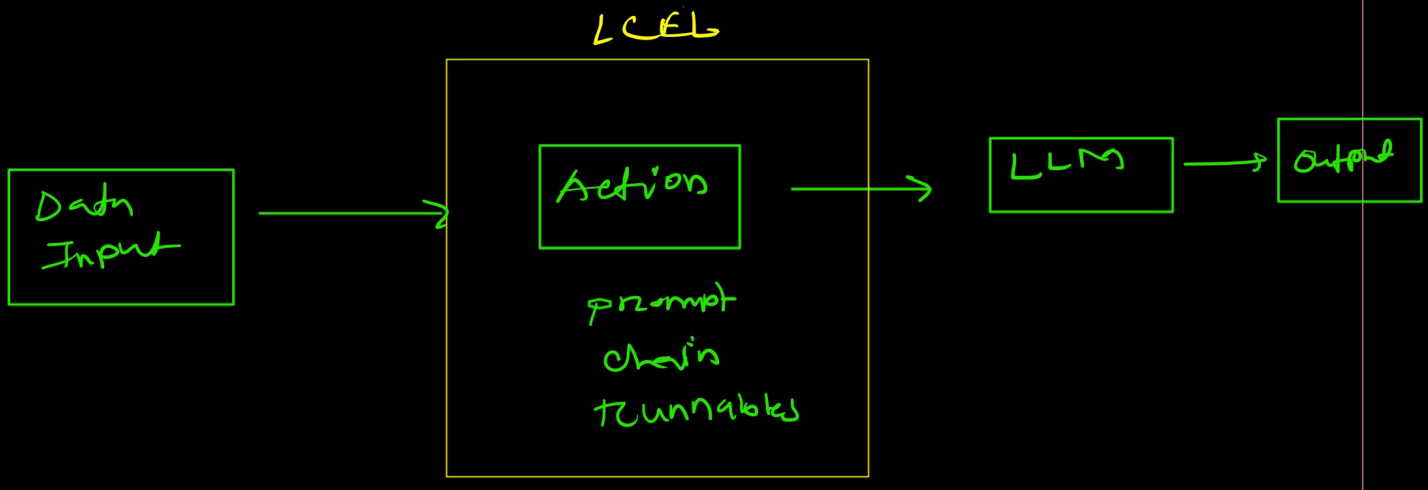
Langchain Expression Language (LCEL)

2nd Generation of Langchain

Agenda:

1. Runnables
2. LCEL Chain: sequence of Runnables
3. Runnables execution order
4. Runnables Execution Alternatives
5. Built-in Runnables and Function
6. Basic LCEL chain operations
7. RAG with LCEL

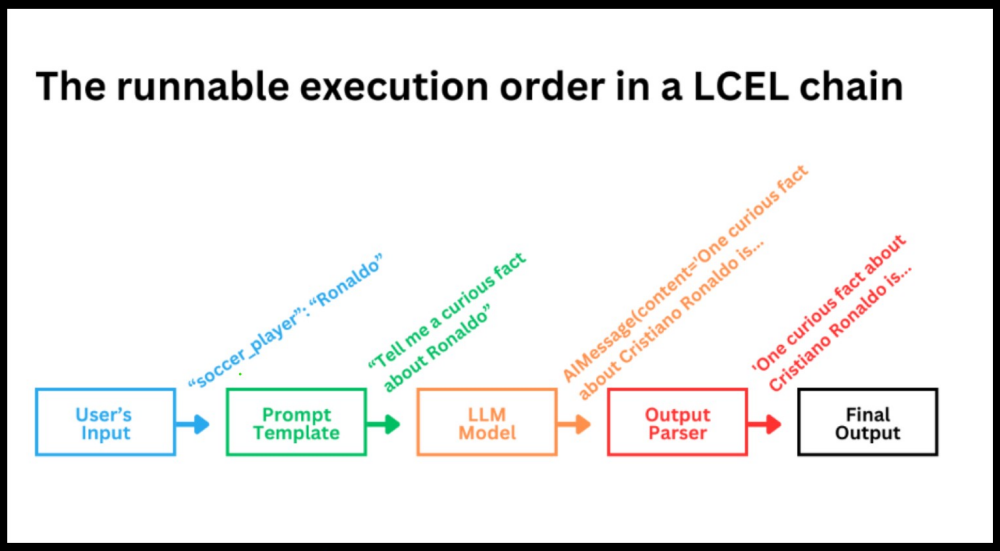


Simple Chain

- Perform several actions in a particular order.

Intro

- Chains are sequences of actions with input or output data.
- Since the emmergence of LCEL, LangChain is favouring LCEL chains over Traditional (Legacy) Built-in Chains, but these are still maintained and frequently used.



Intro to LCEL

Intro

- LCEL has become the backbone of the newest versions of LangChain.
- Traditional chains are still supported, but treated as "Legacy" and have less functionality than the new LCEL chains.
- Many students struggle with LCEL.

Main goals of LCEL

- Make it easy to build chains in a compact way.
- Support advanced LangChain functionality.

Ecosystem:

① Langsmith
② Langserve
③ Langgraph

LCEL

LCEL chain / runnables can be also used asynchronously

prompt | chatmodel | outputparser

big apps

$t_1 + t_2 + t_3 + t_4 + t_5 + t_6 = t_{out}$

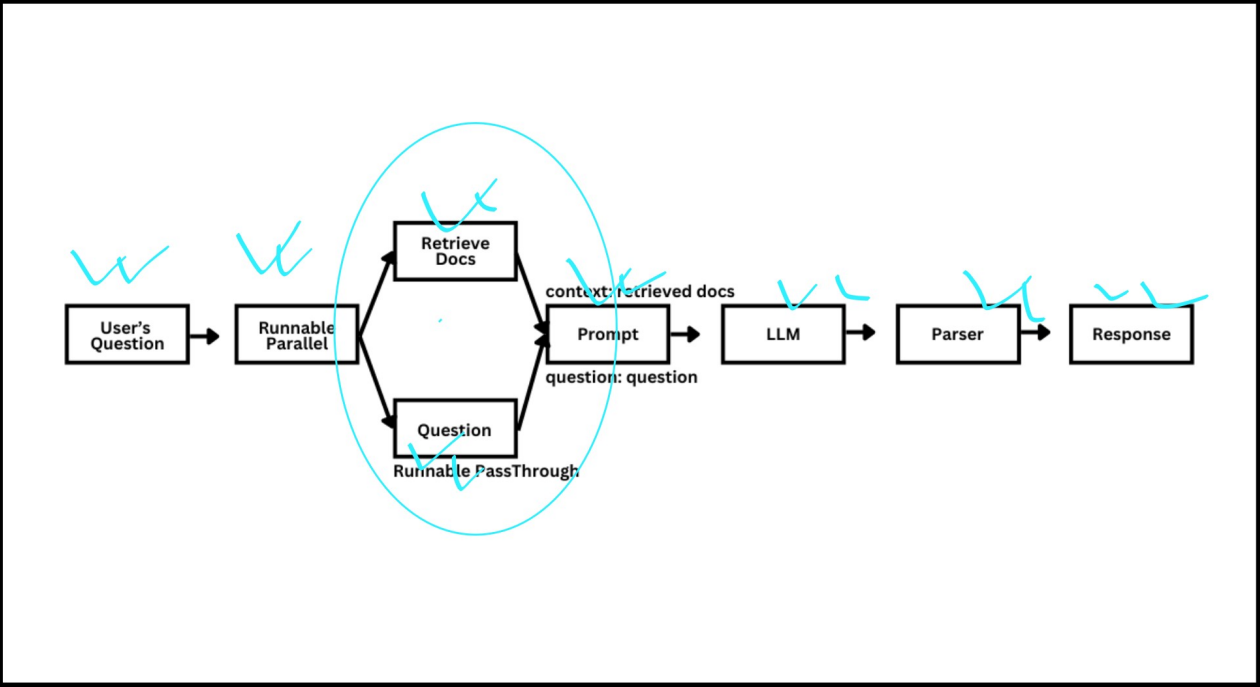
$R_1 \mid R_2 \mid R_3 \mid R_4 \mid R_5 \mid R_6 \cdots$

sequence

2 min

Builtin Runnables

1. RunnablePassthrough
2. RunnableLambda
3. RunnableParallel
   - → itemgetter



Important: the syntax of RunnableParallel can have several variations.

- When composing a RunnableParallel with another Runnable you do not need to wrap it up in the RunnableParallel class. Inside a chain, the next three syntaxs are equivalent:

```
RunnableParallel({"context": retriever, "question": RunnablePassthrough()})
RunnableParallel(context=retriever, question=RunnablePassthrough())
{"context": retriever, "question": RunnablePassthrough()}
```