LCEL Chain

LCEL (LangChain Expression Language)

- It is a declarative way to compose chains in LangChain. It enables **pipelining** (using `|` syntax) and **efficient streaming** of LLM calls, retrievers, and tools.

💡 **Instead of explicitly detailing every step of execution, you declare *what* you want to happen, and LangChain's LCEL engine optimizes *how* it happens.**

🔥 Key Features of LCEL

1. **Pipelining with `|`** – Chain components sequentially.
2. **Automatic Async Support** – Built-in parallel execution.
3. **Streaming** – Real-time token output.
4. **Batch Processing** – Run multiple inputs at once.
5. **Retries & Fallbacks** – Handle API failures gracefully.

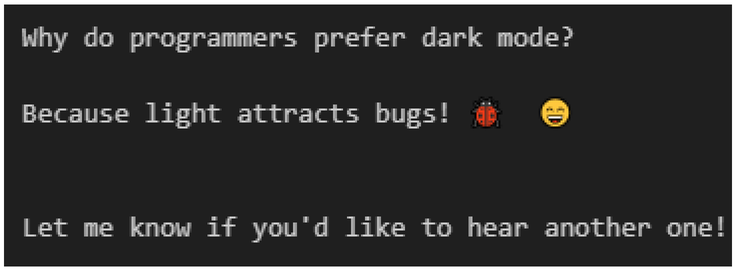
🚀 Basic LCEL Syntax

1. Simple Prompt → LLM Chain

```
pip install langchain_core
```

```
from langchain_core.prompts import ChatPromptTemplate
from langchain_groq import ChatGroq
```

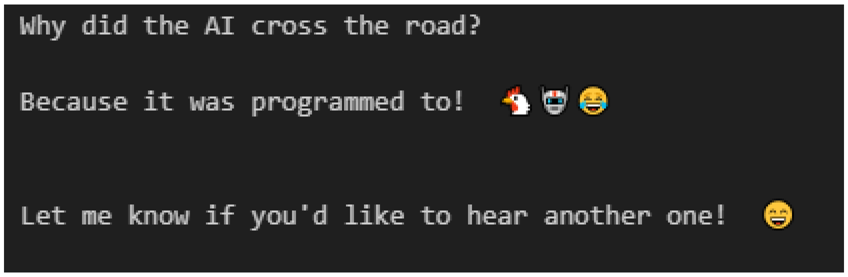
```
prompt = ChatPromptTemplate.from_template("Tell me a joke about {topic}")
model=ChatGroq(model="Gemma2-9b-It")

# Pipe operator (|) chains components
chain = prompt | model

response = chain.invoke({"topic": "programming"})
print(response.content)
```



```
Why do programmers prefer dark mode?

Because light attracts bugs! 🐞  😄


Let me know if you'd like to hear another one!
```

2. Adding an Output Parser

```
from langchain_core.output_parsers import StrOutputParser

chain = prompt | model | StrOutputParser()  # Converts to clean text

response = chain.invoke({"topic": "AI"})
print(response)
```



```
Why did the AI cross the road?

Because it was programmed to!  🐔🤖😂


Let me know if you'd like to hear another one!  😄
```

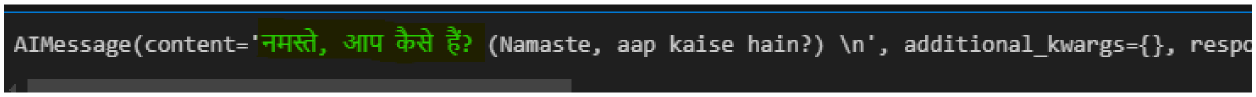
Translation Model

Human Message v System Message

- **Human Message →** Message provides by the user
- **System Message**→ Instruction to the model

```
from langchain_core.messages import HumanMessage,SystemMessage
```

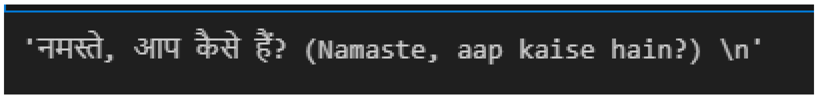
```
from langchain_core.messages import HumanMessage,SystemMessage

messages=[
    SystemMessage(content="Translate the following from English to Hindi"),
    HumanMessage(content="Hello How are you?")
]

result=model.invoke(messages)

result
```



```
AIMessage(content='नमस्ते, आप कैसे हैं? (Namaste, aap kaise hain?) \n', additional_kwargs={}, respo
```

Use Parser

```
from langchain_core.output_parsers import StrOutputParser
parser=StrOutputParser()
parser.invoke(result)
```



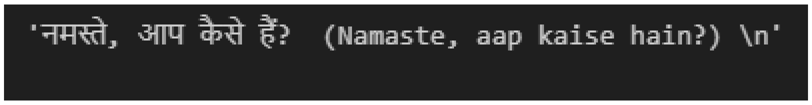
```
'नमस्ते, आप कैसे हैं? (Namaste, aap kaise hain?) \n'
```

💡 **Note: Here, we invoke the prev generated result.**

Use LCEL

- We can chain the components using LCEL

```
### Using LCEL- chain the components
chain= model|parser
chain.invoke(messages)
```



```
'नमस्ते, आप कैसे हैं?  (Namaste, aap kaise hain?) \n'
```

Prompt Templates

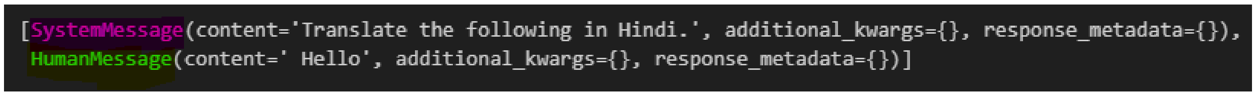
```
from langchain_core.prompts import ChatPromptTemplate

generic_template = "Translate the following in {language}."

prompt = ChatPromptTemplate.from_messages(
        [("system", generic_template),
         ("user"," {text}")
         ]
)
```

```
result=prompt.invoke({"language":"Hindi","text":"Hello"})
result
```

```
Output:

ChatPromptValue(messages=[SystemMessage(content='Translate the followi
ng in Hindi.', additional_kwargs={}, response_metadata={}), HumanMessage(c
ontent=' Hello', additional_kwargs={}, response_metadata={})])
```

💡 **Here, you don’t pass the system & human message separately.**

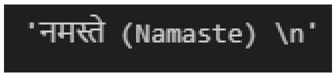
```
result.to_messages()
```



```
[SystemMessage(content='Translate the following in Hindi.', additional_kwargs={}, response_metadata={}),
 HumanMessage(content=' Hello', additional_kwargs={}, response_metadata={})]
```

💡 **It’s not translating because we haven’t yet passed the model.**

```
##Chaining together components with LCEL
chain=prompt|model|parser
chain.invoke({"language":"Hindi","text":"Hello"})
```



```
'नमस्ते (Namaste) \n'
```