Different Types of Propts in LangChain

In LangChain, prompts are primarily categorized into two types:

1. Message propt
2. ChatPromptTemplate

1. Message Propt:

- A structured approach using specific message objects like SystemMessage, HumanMessage, or AIMessage to represent different roles in a conversation.

```
from langchain_core.messages import HumanMessage, SystemMessage

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

print(message)
```

1.1. Use Case:

- When creating multi-turn conversations or managing specific contexts and roles for the AI.

```
# Import necessary libraries
import os
from dotenv import load_dotenv
from langchain_groq import ChatGroq
from langchain_core.output_parsers import StrOutputParser
from langchain_core.messages import HumanMessage, SystemMessage

# Load environment variables from the .env file
load_dotenv()
# Retrieve the API key for the Groq model from environment variables
groq_api_key = os.getenv("GROQ_API_KEY")

# Define the input messages for the conversational AI
# SystemMessage provides context or instructions to the model
# HumanMessage represents the user's input
message = [
    SystemMessage(content="Translate the following from English to Telugu"),
    HumanMessage(content="Hello How are you?")
]

# Initialize the ChatGroq model with the specified configuration
model = ChatGroq(model="Gemma2-9b-It", groq_api_key=groq_api_key)
```

```
# Define an output parser to format the model's response as a string
parser = StrOutputParser()

# Combine the model and parser into a chain to process the input and generate the
output
lcel_chain = model | parser

# Invoke the chain with the input message and get the final translated output
final_result = lcel_chain.invoke(message)

# Print the translated result
print(final_result)
```

2. ChatPromptTemplate:

- A template-based approach for generating messages dynamically. It allows you to define reusable templates with placeholders that can be formatted at runtime.

```
from langchain_core.prompts import ChatPromptTemplate

system_msg = "Trnaslate the following into {language}:"

propt = ChatPromptTemplate([
    ("system", system_msg), ("user","{text}")
])
```

2.1. Use Case:

- When you need flexible, dynamic prompts that can handle multiple inputs and are reusable across different tasks.

```
# Import necessary libraries
import os
from dotenv import load_dotenv
from langchain_core.prompts import ChatPromptTemplate
from langchain_groq.chat_models import ChatGroq
from langchain_core.output_parsers import StrOutputParser

load_dotenv()
groq_api_key = os.getenv("GROQ_API_KEY")

# Define a system message template for the model
# {language} is a placeholder that will be dynamically replaced with the target
language
system_msg = "Translate the following into {language}:"

# Create a ChatPromptTemplate to structure the conversation
# The system message sets the context, and the user message contains the input
```

```
text
propt = ChatPromptTemplate([
    ("system", system_msg),  # Instruction to the model
    ("user", "{text}")       # User input with a dynamic placeholder for the text
to be translated
])

# Initialize the ChatGroq model with the specified configuration
model = ChatGroq(model="Gemma2-9b-It", api_key=groq_api_key)

# Initialize an output parser to process the model's response into a plain string
parser = StrOutputParser()

# Create a chain by combining the prompt, model, and parser
# The pipeline processes the input, runs it through the model, and parses the
output
lcel_chain = propt | model | parser

# Invoke the chain with the desired input
# Replace the placeholders {language} and {text} with actual values
result = lcel_chain.invoke({"language": "Telugu", "text": "How are You"})

# Print the final result, which is the translated text
print(result)
```

3. Key Differences

Feature	Message Prompt	ChatPromptTemplate
Structure	Static list of predefined messages	Template with dynamic placeholders
Flexibility	Fixed content for each message	Dynamic content generated at runtime
Use Case	Multi-turn or context-specific conversations	Reusable and parameterized task prompts
Ease of Use	Straightforward for simple tasks	More adaptable for complex or repetitive tasks