

CHAPTER 04. 프롬프트(Prompt)

1. 프롬프트

언어 모델을 호출할 때 입력되는 글자(문장, 텍스트)를 **프롬프트**라고 합니다.
또한 언어 모델이 사용할 질문이나 명령을 만드는 과정이라고 할 수 있습니다.

프롬프트는 언어 모델이 특정 문맥에서 작동하도록 설정하는 역할을 합니다.
이를 통해 모델은 제공된 정보를 바탕으로 보다 정확하고 관련성 높은 답변을 생성할 수 있습니다.

여러 문서에서 검색된 정보는 서로 다른 관점이나 내용을 포함할 수 있습니다.
프롬프트 단계에서 이러한 정보를 통합하고, 모델이 이를 효율적으로 활용할 수 있는 형식으로 조정합니다.

질문에 대한 모델의 응답 품질은 프롬프트의 구성에 크게 의존합니다.
잘 구성된 프롬프트는 모델이 보다 정확하고 유용한 정보를 제공하게 돕습니다.

프롬프트 단계는 RAG 시스템에서 중요한 역할을 합니다.

이 단계를 통해 언어 모델은 사용자의 질문에 대해 최적화된 방식으로 응답을 생성할 수 있으며, 시스템 전체의 성능과 사용자 만족도에 직접적인 영향을 미칩니다. 프롬프트가 잘 구성되어 있지 않으면, 모델이 비효율적으로 작동할 수 있으며, 결과적으로 사용자의 요구에 부응하지 못하는 응답을 생성할 가능성이 높아집니다.

RAG 프롬프트 구조

- 지시사항(Instruction)
- 질문(사용자 입력 질문)
- 문맥(검색된 정보)

예시 - RAG 프롬프트 구조

당신은 질문-답변(Question-Answer) Task를 수행하는 AI 어시스턴트입니다.
검색된 문맥(context)를 사용하여 질문(question)에 답하세요.
만약, 문맥(context)으로부터 답을 찾을 수 없다면 '모른다'라고 말하세요.
한국어로 대답하세요.

#Question:
{이곳에 사용자가 입력한 질문이 삽입됩니다}

#Context:
{이곳에 검색된 정보가 삽입됩니다}

역할 전달 : 당신은 역사 전문가입니다. 프랑스 혁명에 대해서 알려주세요.

맥락 전달 : 그녀는 20년 경력의 프로그래머로, 이제 새로운 프로젝트 리더로 임명되었지만, 새로운 직책에 대해 불안해하고 있다.

목적 전달 : 파리에서 꼭 가봐야 할 미술관 3곳을 알려주세요.

출력 형식 전달 : 세계에서 가장 높은 산에 대해 글머리 기호 목록으로 알려주세요.

2. PromptTemplate

대부분 LLM applications은 사용자 입력을 직접 LLM에게 전달하지 않습니다.
일반적으로 prompt template 라는 더 큰 텍스트에 사용자 입력을 추가합니다.
언어 모듈을 호출할 때 미리 준비된 프롬프트와 파이썬 입력을 결합해서 사용합니다.

가. from_template() 메소드 사용

from_template() 메소드를 사용해서 PromptTemplate 객체를 생성해서 사용합니다.

치환될 변수를 { 변수 } 로 묶어서 템플릿을 정의합니다.

예제 - from_template() 메소드를 사용해서 객체 생성

```
from langchain_core.prompts import PromptTemplate
```

```
template = '{product}는 대한민국 어느 회사에서 만든 제품인가요?'
```

```
prompt = PromptTemplate.from_template(template)
print(prompt)
```

```
result = prompt.format(product='젤럭시')
```

```
print(result)
```


실행결과

```
input_variables=['product'] template='{product}는 대한민국 어느 회사에서 만든 제품인가요?'
젤럭시는 대한민국 어느 회사에서 만든 제품인가요?
```

예제 - from_template() 메소드를 사용해서 객체 생성

```
from langchain_core.prompts import PromptTemplate
```

```
prompt_template = PromptTemplate.from_template(
    "Tell me a {adjective} joke about {content}."
```



```
)
result = prompt_template.format(adjective="funny", content="chickens")
print(result)
```

실행결과

```
Tell me a funny joke about chickens.
```

예제 - from_template() 메소드를 사용해서 객체 생성

```
from langchain_core.prompts import PromptTemplate
```

```
prompt_template = PromptTemplate.from_template("Tell me a joke")
```

```
result = prompt_template.format()
```

```
print(result)
```

실행결과

Tell me a joke

예제 - from_template() 메소드를 사용해서 객체 생성 후 실행

```
from dotenv import load_dotenv
```

```
import os
```

```
load_dotenv(verbose=True)
```

```
key = os.getenv('OPENAI_API_KEY')
```

```
from langchain_openai import ChatOpenAI
```

```
from langchain_core.prompts import PromptTemplate
```

```
llm = ChatOpenAI(
    api_key=key,
    model_name='gpt-4o-mini',
    temperature=0.1,
    max_tokens=2048,
)
```

```
# template 정의
```

```
template = '{product}는 대한민국 어느 회사에서 만든 제품인가요?'
```

```
# from_template 메소드를 이용하여 PromptTemplate 객체 생성
```

```
prompt = PromptTemplate.from_template(template)
```

```
# chain 생성
```

```
chain = prompt | llm
```

```
# product 변수에 입력된 값(삼성 갤럭시폰)이 자동으로 치환되어 수행됨
```

```
answer = chain.invoke("삼성 갤럭시폰")
```

```
print(answer)
```

```
print(answer.content)
```

실행결과

```
content='삼성 갤럭시폰은 대한민국의 삼성전자에서 만든 제품입니다. 삼성전자는 스마트폰, 가전제품, 반도체 등 다양한 전자제품을 제조하는 글로벌 기업입니다. 갤럭시 시리즈는 삼성전자의 대표적인 스마트폰 브랜드 중 하나입니다.' response_metadata={'token_usage': {'completion_tokens': 61, 'prompt_tokens': 24, 'total_tokens': 85}, 'model_name': 'gpt-4o-mini-2024-07-18', 'system_fingerprint': 'fp_ba606877f9', 'finish_reason': 'stop', 'logprobs': None} id='run-b95422a1-dba2-4291-a968-e21041fd7e8e-0' usage_metadata={'input_tokens': 24, 'output_tokens': 61, 'total_tokens': 85}
```

삼성 갤럭시폰은 대한민국의 삼성전자에서 만든 제품입니다. 삼성전자는 스마트폰, 가전제품, 반도체 등 다양한 전자제품을 제조하는 글로벌 기업입니다. 갤럭시 시리즈는 삼성전자의 대표적인 스마트폰 브랜드 중 하나입니다.

나. PromptTemplate 객체 생성과 동시에 prompt 생성

PromptTemplate 객체 생성과 동시에 prompt를 생성합니다.

input_variables에 리스트 형태로 사용자 입력을 받는 변수를 지정합니다

예제 - PromptTemplate 객체 생성과 동시에 prompt를 생성

```
from langchain_core.prompts import PromptTemplate
```

template 정의

```
template = "{country}의 수도는 어디인가요?"
```

PromptTemplate 객체를 활용하여 prompt_template 생성

```
prompt = PromptTemplate(  
    template=template,  
    input_variables=["country"],  
)
```

```
print(prompt)
```

```
# PromptTemplate(input_variables=['country'], template='{country}의 수도는 어디인가요?')
```

```
prompt_text = prompt.format(country="대한민국")
```

```
print(prompt_text)
```

실행결과

```
input_variables=['country'] template='{country}의 수도는 어디인가요?'
```

```
대한민국의 수도는 어디인가요?
```

예제 - PromptTemplate 객체 생성과 동시에 prompt를 생성

```
from langchain_core.prompts import PromptTemplate
```

```
template = '{product}는 대한민국 어느 회사에서 만든 제품인가요?'
```

```
prompt_template = PromptTemplate(
    template=template,
    input_variables=['product'],
)
```

```
prompt = prompt_template.format(product='갤럭시')
print(prompt)
```

실행결과

갤럭시는 대한민국 어느 회사에서 만든 제품인가요?

예제 - PromptTemplate 객체 생성과 동시에 prompt를 생성

```
from dotenv import load_dotenv
import os
```

```
load_dotenv(verbose=True)
key = os.getenv('OPENAI_API_KEY')
```

```
from langchain_openai import ChatOpenAI
from langchain_core.prompts import PromptTemplate
```

```
llm = ChatOpenAI(
    api_key=key,
    model_name='gpt-4o-mini',
    temperature=0.1,
    max_tokens=2048,
)
```

```
template = "{country1}과 {country2}의 수도는 각각 어디인가요?"
```

```
# PromptTemplate 객체를 활용하여 prompt_template 생성
```

```
prompt = PromptTemplate(
    template=template,
    input_variables=["country1", "country2"]
)
```

```
chain = prompt | llm
answer = chain.invoke({"country1": "대한민국", "country2": "호주"})
print(answer.content)
```

실행결과

대한민국의 수도는 서울이고, 호주의 수도는 캔버라입니다.

예제 - PromptTemplate 객체 생성과 동시에 prompt를 생성

```
from dotenv import load_dotenv
import os

load_dotenv(verbose=True)
key = os.getenv('OPENAI_API_KEY')

from langchain_openai import ChatOpenAI
from langchain_core.prompts import PromptTemplate

llm = ChatOpenAI(
    api_key=key,
    model_name='gpt-4o-mini',
    temperature=0.1,
    max_tokens=2048,
)

template = "{country1}과 {country2}의 수도는 각각 어디인가요?"

# PromptTemplate 객체를 활용하여 prompt_template 생성
prompt = PromptTemplate(
    template=template,
    input_variables=["country1"],
    partial_variables={
        "country2": "미국"  # dictionary 형태로 partial_variables를 전달
    },
)

chain = prompt | llm
answer = chain.invoke({"country1": "대한민국"})
print(answer.content)
```



실행 결과

대한민국의 수도는 서울이고, 호주의 수도는 입니다.

예제 - PromptTemplate 객체 생성과 동시에 prompt를 생성

```
from dotenv import load_dotenv
import os

load_dotenv(verbose=True)
key = os.getenv('OPENAI_API_KEY')

from langchain_openai import ChatOpenAI
from langchain_core.prompts import PromptTemplate

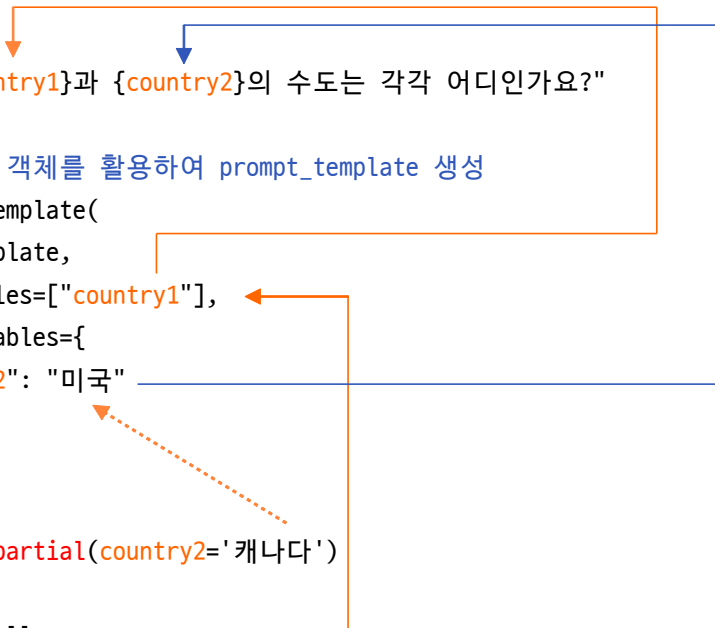
llm = ChatOpenAI(
    api_key=key,
    model_name='gpt-4o-mini',
    temperature=0.1,
    max_tokens=2048,
)

template = "{country1}과 {country2}의 수도는 각각 어디인가요?"

# PromptTemplate 객체를 활용하여 prompt_template 생성
prompt = PromptTemplate(
    template=template,
    input_variables=["country1"],
    partial_variables={
        "country2": "미국"
    },
)

prompt = prompt.partial(country2='캐나다')

chain = prompt | llm
answer = chain.invoke({"country1": "대한민국"})
print(answer.content)
```



실행결과

대한민국의 수도는 서울이고, 캐나다의 수도는 오타와입니다.

다. PromptTemplate 객체 생성과 동시에 prompt 생성

partial_variables: 부분 변수 채움

partial을 사용하는 일반적인 용도는 함수를 부분적으로 사용하는 것입니다.
날짜와 같이 항상 공통된 방식으로 가져오고 싶은 변수가 있는 경우입니다.

항상 현재 날짜가 표시되기를 원하는 프롬프트가 있다고 가정해 보겠습니다.

프롬프트에 하드 코딩할 수도 없고, 다른 입력 변수와 함께 전달하는 것도 번거롭습니다.

이 경우 항상 현재 날짜를 반환하는 함수를 사용하여 프롬프트를 부분적으로 변경할 수 있으면 매우 편리합니다.

예제 - partial_variables

```
from dotenv import load_dotenv
import os
```

```
load_dotenv(verbose=True)
key = os.getenv('OPENAI_API_KEY')
```

```
from langchain_openai import ChatOpenAI
from langchain_core.prompts import PromptTemplate
from datetime import datetime
```

날짜를 반환하는 함수 정의

```
def get_today():
    return datetime.now().strftime("%B %d")
```

```
llm = ChatOpenAI(
    api_key=key,
    model_name='gpt-4o-mini',
    temperature=0.1,
    max_tokens=2048,
)
```

```
prompt = PromptTemplate(
    template="오늘의 날짜는 {today}입니다. 오늘이 생일인 유명인 {n}명을 나열해 주세요. 생년월일을 표기해 주  
세요.",
    input_variables=["n"],
    partial_variables={
        "today": get_today
    },
)
```

```
# print(prompt.format(n=3))
chain = prompt | llm
```

```
answer = chain.invoke({"n": 3})    # {"today": "Jan 02", "n": 3}
print(answer.content)
```

실행결과

오늘, 1월 2일에 생일인 유명인 3명은 다음과 같습니다:

1. ****아이작 아시모프 (Isaac Asimov)**** - 1920년 1월 2일
2. ****J.R.R. 톨킨 (J.R.R. Tolkien)**** - 1892년 1월 2일
3. ****드류 배리모어 (Drew Barrymore)**** - 1975년 1월 2일

이 외에도 1월 2일에 태어난 많은 유명인들이 있습니다!

3. ChatPromptTemplate

ChatPromptTemplate 은 대화목록을 프롬프트로 주입하고자 할 때 활용할 수 있습니다.

메시지는 튜플(tuple) 형식으로 구성하며, (role, message) 로 구성하여 리스트로 생성할 수 있습니다.

role

- "system": 시스템 설정 메시지입니다. 주로 전역설정과 관련된 프롬프트입니다.
- "human": 사용자 입력 메시지입니다.
- "ai": AI의 답변 메시지입니다.

예제 - chatPromptTemplate

```
from langchain_core.prompts import ChatPromptTemplate

chat_prompt = ChatPromptTemplate.from_template("{country}의 수도는 어디인가요?")
# print(chat_prompt)

# print(chat_prompt.format(country="대한민국"))
```

실행결과

```
ChatPromptTemplate(input_variables=['country'],
  messages=[HumanMessagePromptTemplate(prompt=PromptTemplate(input_variables=['country'],
    template='{country}의 수도는 어디인가요?'))])

'Human: 대한민국의 수도는 어디인가요?'
```

예제 - chatPromptTemplate

```
from dotenv import load_dotenv
import os

from langchain_openai import ChatOpenAI
from langchain_core.prompts import ChatPromptTemplate

load_dotenv(verbose=True)
key = os.getenv('OPENAI_API_KEY')

llm = ChatOpenAI(
    api_key=key,
    model_name='gpt-4o-mini',
    temperature=0.1,
    max_tokens=2048,
)

chat_template = ChatPromptTemplate.from_messages(
    [
        ("system", "당신은 훌륭한 AI bot입니다. 당신의 이름은 {name}입니다."),
        ("human", "{product}는 대한민국의 어느 회사에서 만든 제품입니까?")
    ]
)

messages = chat_template.format_messages(name="Bob", product="갤럭시")
print(messages)

#생성한 메시지를 바로 주입하여 결과를 받을 수 있습니다.
result = llm.invoke(messages)
print(result)
```



실행결과

```
[SystemMessage(content='당신은 훌륭한 AI bot입니다. 당신의 이름은 Bob 입니다. '),
HumanMessage(content='갤럭시는 대한민국의 어느 회사에서 만든 제품입니까?')]
content='갤럭시는 대한민국의 삼성전자에서 만든 제품입니다. 삼성전자는 스마트폰, 태블릿, 가전제품 등 다양한 전자기기를 제조하는 글로벌 기업입니다.' response_metadata={'token_usage': {'completion_tokens': 38, 'prompt_tokens': 44, 'total_tokens': 82}, 'model_name': 'gpt-4o-mini-2024-07-18', 'system_fingerprint': 'fp_ba606877f9', 'finish_reason': 'stop', 'logprobs': None} id='run-1f557cd8-41c9-40ec-9039-e59dc1c31143-0' usage_metadata={'input_tokens': 44, 'output_tokens': 38, 'total_tokens': 82}
```

예제 - chatPromptTemplate

```
from dotenv import load_dotenv
import os

from langchain_openai import ChatOpenAI
from langchain_core.prompts import ChatPromptTemplate

load_dotenv(verbose=True)
key = os.getenv('OPENAI_API_KEY')

llm = ChatOpenAI(
    api_key=key,
    model_name='gpt-4o-mini',
    temperature=0.1,
    max_tokens=2048,
)

chat_template = ChatPromptTemplate.from_messages(
    [
        # role, message
        ("system", "당신은 친절한 AI 어시스턴트입니다. 당신의 이름은 {name} 입니다."),
        ("human", "반가워요!"),
        ("ai", "안녕하세요! 무엇을 도와드릴까요?"),
        ("human", "{user_input}"),
    ]
)

# 챗 message를 생성합니다.
messages = chat_template.format_messages(name="은영", user_input="당신의 이름은 무엇입니까?")

llm.invoke(messages)
```

실행결과

```
AIMessage(content='제 이름은 은영입니다! 당신과 대화하게 되어 기쁩니다. 어떤 이야기를 나눠볼까요?', response_metadata={'token_usage': {'completion_tokens': 26, 'prompt_tokens': 66, 'total_tokens': 92}, 'model_name': 'gpt-4o-mini-2024-07-18', 'system_fingerprint': 'fp_0f03d4f0ee', 'finish_reason': 'stop', 'logprobs': None}, id='run-fe5bba2f-a548-45c4-a9de-8b1e13124c2e-0', usage_metadata={'input_tokens': 66, 'output_tokens': 26, 'total_tokens': 92})
```

예제 - chatPromptTemplate

```
from dotenv import load_dotenv
import os

from langchain_openai import ChatOpenAI
from langchain_core.prompts import ChatPromptTemplate

load_dotenv(verbose=True)
key = os.getenv('OPENAI_API_KEY')

llm = ChatOpenAI(
    api_key=key,
    model_name='gpt-4o-mini',
    temperature=0.1,
    max_tokens=2048,
)

chat_template = ChatPromptTemplate.from_messages(
    [
        # role, message
        ("system", "당신은 친절한 AI 어시스턴트입니다. 당신의 이름은 {name} 입니다."),
        ("human", "반가워요!"),
        ("ai", "안녕하세요! 무엇을 도와드릴까요?"),
        ("human", "{user_input}"),
    ]
)

# 챗 message를 생성합니다.
messages = chat_template.format_messages(
    name="은영", user_input="당신의 이름은 무엇입니까?"
)

chain = chat_template | llm
answer = chain.invoke({"name": "은영", "user_input": "당신의 이름은 무엇입니까?"})
print(answer.content)
```

실행결과

제 이름은 은영입니다! 당신과 대화하게 되어 기쁩니다. 다른 질문이나 궁금한 점이 있으면 말씀해 주세요!

예제 - chatPromptTemplate

```
from dotenv import load_dotenv
import os

from langchain_openai import ChatOpenAI
from langchain_core.prompts import ChatPromptTemplate

load_dotenv(verbose=True)
key = os.getenv('OPENAI_API_KEY')

llm = ChatOpenAI(
    api_key=key,
    model_name='gpt-4o-mini',
    temperature=0.1,
    max_tokens=2048,
)

chat_template = ChatPromptTemplate.from_messages(
    [
        ("system", "당신은 훌륭한 AI bot입니다. 당신의 이름은 {name}입니다."),
        ("human", "{product}는 대한민국의 어느 회사에서 만든 제품입니까?")
    ]
)

chain = chat_template | llm

result = chain.invoke({'name': 'Bob', 'product': '갤럭시'})
print(result)
```

실행결과

```
content='갤럭시는 대한민국의 삼성전자에서 만든 제품입니다. 삼성전자는 스마트폰, 태블릿, 가전제품 등 다양한 전자제품을 제조하는 글로벌 기업입니다.' response_metadata={'token_usage': {'completion_tokens': 38, 'prompt_tokens': 44, 'total_tokens': 82}, 'model_name': 'gpt-4o-mini-2024-07-18', 'system_fingerprint': 'fp_ba606877f9', 'finish_reason': 'stop', 'logprobs': None} id='run-4c493d69-c219-4e20-b51f-e0bf23925947-0' usage_metadata={'input_tokens': 44, 'output_tokens': 38, 'total_tokens': 82}
```

4. MessagePlaceholder

LangChain은 포맷하는 동안 렌더링할 메시지를 완전히 제어할 수 있는 MessagePlaceholder를 제공합니다.

아직 확정된 메시지는 아니지만 나중에 채워질 메시지를 MessagePlaceholder로 잡아둡니다.

이렇게 MessagePlaceholder로 잡아두는 경우는 챗봇을 만들 때 대화를 주고받을 때 이 대화 내용을 기록하고자 할 때 이 MessagePlaceholder를 사용합니다.

대화 기록이라는 것은 우리가 대화를 하는 과정에서 계속 쌓여나가는데요 이 쌓여나가는 대화도 결국 나중에 프롬프트 입력으로 넣어줘야 됩니다. 그런데 중간 부분에 넣어주고 싶을 때.. 대화 기록이란 것이 아직 확정이 안된 상태입니다. (대화하는 도중이라서..) 그래서 MessagePlaceholder에 history라는 변수로 잡아둡니다. 잡아둔 다음에 나중에 또 이 대화 기록을 보고, 또 질문이 들어왔을 때 대처를 해주는 방식으로 구성해 나갈 수 있습니다.

MessagePlaceholder는 아직 확정되지 않은 메시지나 대화를 나중에 넣어주기 위해서 위치만 잡아두는 것입니다. 위치를 잡아둘 때는 키값이 필요합니다. `variable_name="conversation"` 과 같이 잡아둘 수 있습니다.

예제 -

```
from langchain_core.output_parsers import StrOutputParser
from langchain_core.prompts import ChatPromptTemplate, MessagesPlaceholder

chat_prompt = ChatPromptTemplate.from_messages(
    [
        (
            "system",
            "당신은 요약 전문 AI 어시스턴트입니다. 당신의 임무는 주요 키워드로 대화를 요약하는 것입니다."
        ),
        MessagesPlaceholder(variable_name="conversation"),
        ("human", "지금까지의 대화를 {word_count} 단어로 요약합니다."),
    ]
)

print(chat_prompt)
```

실행결과

```
input_variables=['conversation', 'word_count']
input_types={'conversation': typing.List[typing.Union[langchain_core.messages.ai.AIMessage,
langchain_core.messages.human.HumanMessage,
langchain_core.messages.chat.ChatMessage,
langchain_core.messages.system.SystemMessage,
langchain_core.messages.function.FunctionMessage,
langchain_core.messages.tool.ToolMessage]]}
messages=[SystemMessagePromptTemplate(prompt=PromptTemplate(input_variables=[], template='당신은 요약 전문 AI 어시스턴트입니다. 당신의 임무는 주요 키워드로 대화를 요약하는 것입니다.')),
MessagesPlaceholder(variable_name='conversation'),
HumanMessagePromptTemplate(prompt=PromptTemplate(input_variables=['word_count'], template='지금까지의 대화를 {word_count} 단어로 요약합니다.'))]
```

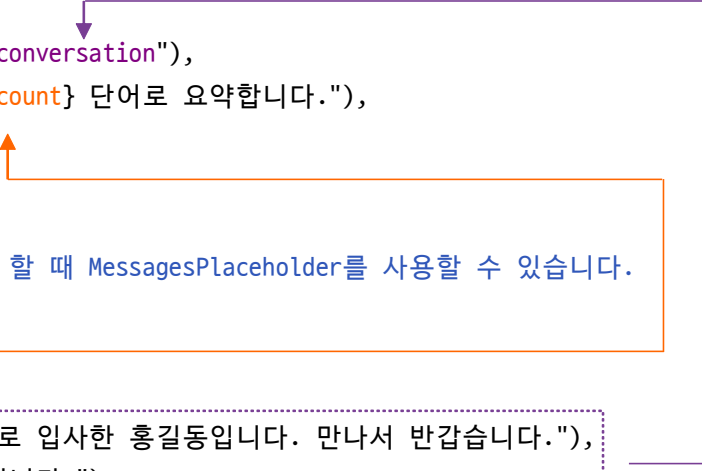
예제 -

```
from langchain_core.output_parsers import StrOutputParser
from langchain_core.prompts import ChatPromptTemplate, MessagesPlaceholder

chat_prompt = ChatPromptTemplate.from_messages(
    [
        (
            "system",
            "당신은 요약 전문 AI 어시스턴트입니다. 당신의 임무는 주요 키워드로 대화를 요약하는 것입니다."
        ),
        MessagesPlaceholder(variable_name="conversation"),
        ("human", "지금까지의 대화를 {word_count} 단어로 요약합니다."),
    ]
)

# conversation 대화목록을 나중에 추가하고자 할 때 MessagesPlaceholder를 사용할 수 있습니다.
formatted_chat_prompt = chat_prompt.format(
    word_count=5,
    conversation=[
        ("human", "안녕하세요! 저는 오늘 새로 입사한 홍길동입니다. 만나서 반갑습니다."),
        ("ai", "반가워요! 앞으로 잘 부탁드립니다."),
    ],
)

print(formatted_chat_prompt)
```



실행결과

System: 당신은 요약 전문 AI 어시스턴트입니다. 당신의 임무는 주요 키워드로 대화를 요약하는 것입니다.
Human: 안녕하세요! 저는 오늘 새로 입사한 홍길동입니다. 만나서 반갑습니다.
AI: 반가워요! 앞으로 잘 부탁드립니다.
Human: 지금까지의 대화를 5 단어로 요약합니다.

예제 -

```
from dotenv import load_dotenv
import os

from langchain_openai import ChatOpenAI
from langchain_core.prompts import ChatPromptTemplate
from langchain_core.output_parsers import StrOutputParser
from langchain_core.prompts import ChatPromptTemplate, MessagesPlaceholder

load_dotenv(verbose=True)
key = os.getenv('OPENAI_API_KEY')

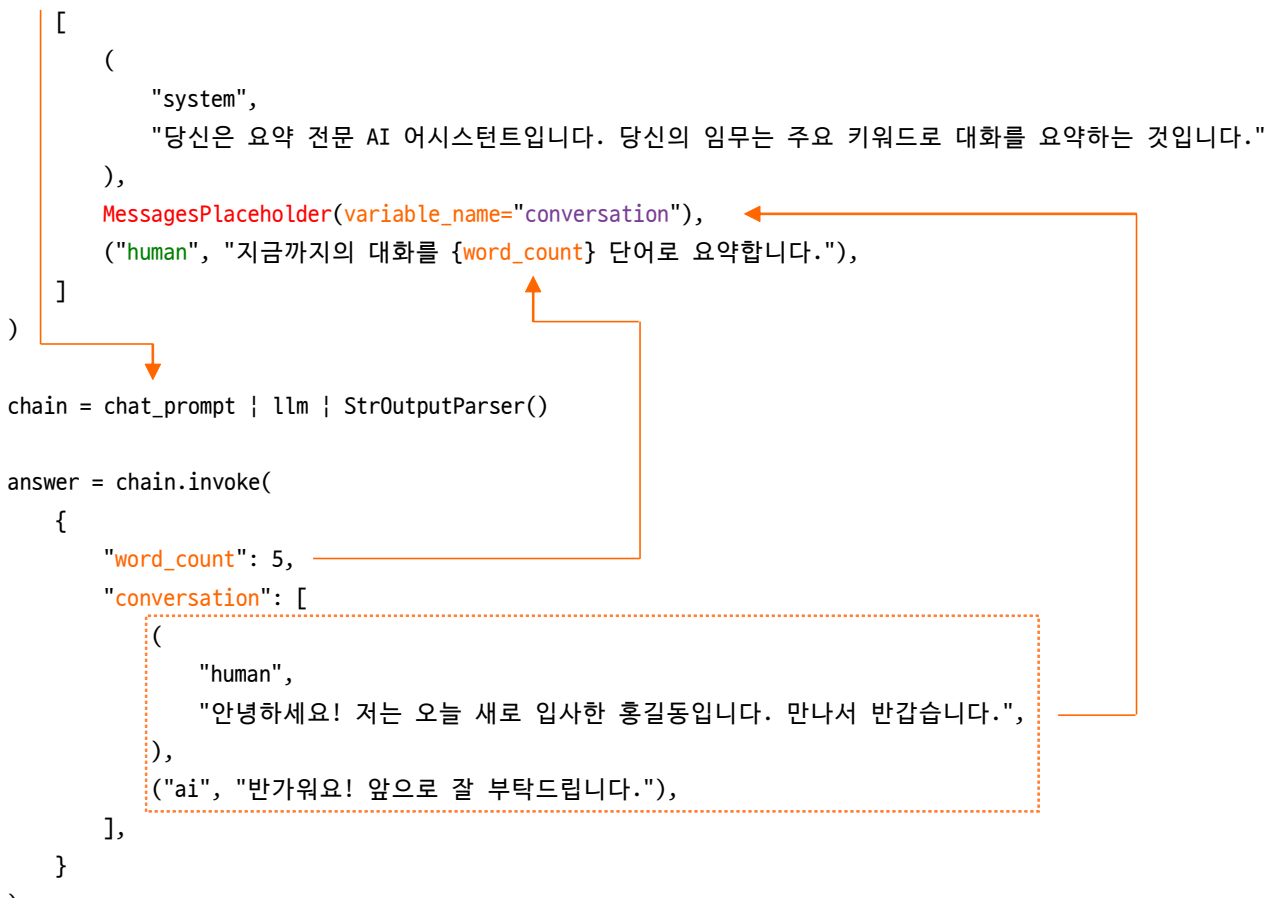
llm = ChatOpenAI(
    model_name='gpt-4o-mini',
    temperature=0.1,
    max_tokens=2048,
)

chat_prompt = ChatPromptTemplate.from_messages(
    [
        (
            "system",
            "당신은 요약 전문 AI 어시스턴트입니다. 당신의 임무는 주요 키워드로 대화를 요약하는 것입니다."
        ),
        MessagesPlaceholder(variable_name="conversation"),
        ("human", "지금까지의 대화를 {word_count} 단어로 요약합니다."),
    ]
)

chain = chat_prompt | llm | StrOutputParser()

answer = chain.invoke(
    {
        "word_count": 5,
        "conversation": [
            (
                "human",
                "안녕하세요! 저는 오늘 새로 입사한 홍길동입니다. 만나서 반갑습니다.",
            ),
            ("ai", "반가워요! 앞으로 잘 부탁드립니다."),
        ],
    }
)

print(answer)
```



실행결과

'홍길동, 새로 입사, 반가움.'

5. 프롬프트

예제 - prompt/fruit_color.yaml

```
_type: "prompt"
template: "{fruit}의 색깔이 뭐야?"
input_variables: ["fruit"]
```

예제 - 프롬프트

```
from dotenv import load_dotenv
import os

load_dotenv(verbose=True)
key = os.getenv('OPENAI_API_KEY')

from langchain_teddynote import logging
logging.langsmith('app_05')

from langchain_teddynote.prompts import load_prompt

prompt1 = load_prompt('prompts/fruit_color.yaml', encoding='utf-8')

print(prompt1)
print(prompt1.format(fruit='멜론'))
```

실행결과

```
PromptTemplate(input_variables=['fruit'], template='{fruit}의 색깔이 뭐야?')
```

실행결과

```
'멜론의 색깔이 뭐야?'
```

예제 - prompt/capital.yaml

```
_type: "prompt"
template: |
    {country}의 수도에 대해서 알려주세요.
    수도의 특징을 다음의 양식에 맞게 정리해 주세요.
    300자 내외로 작성해 주세요.
    한글로 작성해 주세요.
    ----
    [양식]
    1. 면적
    2. 인구
    3. 역사적 장소
    4. 특산물

    #Answer:
input_variables: ["country"]
```

예제 - 프롬프트

```
from dotenv import load_dotenv
import os

load_dotenv(verbose=True)
key = os.getenv('OPENAI_API_KEY')

from langchain_teddynote import logging
logging.langsmith('app_05')

from langchain_teddynote.prompts import load_prompt

prompt = load_prompt('prompts/capital.yaml')

print(prompt.format(country='대한민국'))
```

실행결과

```
'대한민국의 수도에 대해서 알려주세요.\n수도의 특징을 다음의 양식에 맞게 정리해 주세요.\n300자 내외로 작성해 주세요.\n한글로 작성해 주세요.\n----\n[양식]\n1. 면적\n2. 인구\n3. 역사적 장소\n4. 특산물\n\n#Answer:\n'
```
