
- 출처: LangChain 공식 문서 또는 해당 교재명
- 원본 URL: https://smith.langchain.com/hub/teddynote/summary-stuff-documents

---

10. `Runtime Arguments 바인딩`

1) `Runtime Arguments 바인딩`

- **`Runnable.bind()`**
  - `Runnable` 시퀀스 내에서 `Runnable` 호출 시, 이전 `Runnable` 출력이나 사용자 입력에 포함되지 않은 상수 인자를 전달해야 할 경우 사용
- **`RunnablePassthrough`**
  - `{equation_statement}` 변수를 프롬프트에 전달 → `StrOutputParser` 를 사용 → 모델의 출력을 문자열로 파싱하는 `runnable` 객체를 생성**
  - **`runnable.invoke()` 메서드 호출 → `"x raised to the third plus seven equals 12"` 라는 방정식 문장 전달 → 결과 출력**

- **환경설정**
- `LLM` 설정
  - ① `gemini-2.5.-flash-lite`
  - ② `gpt-4o-mini`

```
from langchain_core.output_parsers import StrOutputParser
from langchain_core.prompts import ChatPromptTemplate
from langchain_core.runnables import RunnablePassthrough

from langchain_google_genai import ChatGoogleGenerativeAI

from dotenv import load_dotenv
import os

# LLM 초기화
# API 키 확인
if not os.getenv("GOOGLE_API_KEY"):
    os.environ["GOOGLE_API_KEY"] = input("Enter your Google API key: ")

# LLM 생성하기
gemini_lc = ChatGoogleGenerativeAI(
    model="gemini-2.5-flash-lite",
    temperature=0,
)
```

- 기본 `LLM` 생성하기 (`gemini_lc`) - `gemini-2.5.flash-lite`

```
E0000 00:00:1760013374.064014 2375374 alts_credentials.cc:93] ALTS creds ignored. Not running on GCP and untruste
```

```
from langchain_core.output_parsers import StrOutputParser
from langchain_core.prompts import ChatPromptTemplate
from langchain_core.runnables import RunnablePassthrough
```

```
import getpass
import os

# 2nd OpenAI API key사용해 OpenAI 모델 초기화
OPEN_API_KEY = os.getenv("OPENAI_API_KEY2")

if not os.environ.get("OPENAI_API_KEY"):
    os.environ["OPENAI_API_KEY"] = getpass.getpass("Enter ✦ key for OpenAI: ")
```

```
from langchain_openai import ChatOpenAI

model = ChatOpenAI(model="gpt-4o-mini", temperature=0)              # 4.1s
```

- **프롬프트**

```
# 프롬프트 생성
prompt = ChatPromptTemplate.from_messages(
    [
        (
            "system",
            # 대수 기호를 사용하여 다음 방정식을 작성한 다음 풀이하세요.
            "Write out the following equation using algebraic symbols then solve it. "
            "Use the format\n\nEQUATION:...\nSOLUTION:...\n\n",
        ),
        (
            "human",
            "{equation_statement}",                          # 사용자가 입력한 방정식 문장을 변수로 받기
        ),
    ]
)
```

- **`runnable` 객체 생성하기**
  - ① `gemini-2.5-flash-lite`
  - ② `gpt-4o-mini`

```
# runnable 객체 생성하기 ①
# 방정식 문장을 입력 → 프롬프트에 전달 → 모델에서 생성된 결과 → 문자열로 파싱
runnable = (
    {"equation_statement": RunnablePassthrough()} | prompt | gemini_lc | StrOutputParser()
)
```

```
# runnable 객체 생성하기 ②
# 방정식 문장을 입력 → 프롬프트에 전달 → 모델에서 생성된 결과 → 문자열로 파싱
runnable = (
    {"equation_statement": RunnablePassthrough()} | prompt | model | StrOutputParser()
)
```

- **모델 호출** → **출력 결과 비교하기**
  - ① `gemini-2.5-flash-lite`
  - ② `gpt-4o-mini`

```
# query_1 - ①
# 예시 방정식 문장을 입력 → 결과 출력하기
print(runnable.invoke("x raised to the third plus seven equals 12"))
```

- `query_1` / `gemini-2.5-flash-lite` - (`1.5s`)

```
EQUATION: $x^3 + 7 = 12$

SOLUTION:
To solve for x, we first isolate the $x^3$ term by subtracting 7 from both sides of the equation:
$x^3 + 7 - 7 = 12 - 7$
$x^3 = 5$

Next, we take the cube root of both sides to solve for x:
$\sqrt[3]{x^3} = \sqrt[3]{5}$
$x = \sqrt[3]{5}$

The solution is $x = \sqrt[3]{5}$.
```

```
# query_1 - ②
# 예시 방정식 문장을 입력 → 결과 출력하기
print(runnable.invoke("x raised to the third plus seven equals 12"))
```

- `query_1` / `gpt-4o-mini` - (`3.4s`)

```
EQUATION: \( x^3 + 7 = 12 \)

SOLUTION:
1. Subtract 7 from both sides:
\[
x^3 = 12 - 7
\]
\[
x^3 = 5
\]

2. Take the cube root of both sides:
\[
x = \sqrt[3]{5}
\]

Thus, the solution is:
\[
x \approx 1.71
\] (approximately, to two decimal places)
```

- **특정 `stop` 단어 사용 → 모델 호출하기**
  - **`model.bind()`** → 언어 모델 호출 → 생성된 텍스트에서 **`SOLUTION`** **토큰까지만 출력하기**
- 모델 2개 사용
  - ① `gemini-2.5-flash-lite`
  - ② `gpt-4o-mini`

```
# ① gemini-2.5-flash-lite
runnable = (
    # 실행 가능한 패스스루 객체를 생성하여 "equation_statement" 키에 할당하기
    {"equation_statement": RunnablePassthrough()}
    | prompt                                                # 프롬프트를 파이프라인에 추가하기
    | gemini_lc.bind(                                       # 모델을 바인딩
        stop_sequences="SOLUTION"                           # gemini-config에서는 stop 대신 stop_sequences 사용
    )                                                       # "SOLUTION" 토큰에서 생성을 중지하도록 설정
    | StrOutputParser()                                     # 문자열 출력 파서를 파이프라인에 추가합니다.
)
```

```
# ② gpt-4o-mini
runnable = (
    # 실행 가능한 패스스루 객체를 생성하여 "equation_statement" 키에 할당하기
    {"equation_statement": RunnablePassthrough()}
    | prompt                                                # 프롬프트를 파이프라인에 추가하기
    | model.bind(                                           # 모델을 바인딩
        stop="SOLUTION"                                     # "SOLUTION" 토큰에서 생성을 중지하도록 설정
    )
    | StrOutputParser()                                     # 문자열 출력 파서를 파이프라인에 추가합니다.
)
```

- **`model.bind()`** 결과 확인하기
  - ① `gemini-2.5-flash-lite`
  - ② `gpt-4o-mini`

```
# query_2 ①
# "x raised to the third plus seven equals 12"라는 입력으로 파이프라인을 실행 → 결과 출력하기
print(runnable.invoke("x raised to the third plus seven equals 12"))
```

- `query_2` / `gemini-2.5-flash-lite` - (`1.1s`)

```
EQUATION: $x^3 + 7 = 12$

SOLUTION:
To solve for x, we first isolate the $x^3$ term by subtracting 7 from both sides of the equation:
$x^3 + 7 - 7 = 12 - 7$
$x^3 = 5$
```

```
Next, we take the cube root of both sides to solve for x:
$\sqrt[3]{x^3} = \sqrt[3]{5}$
$x = \sqrt[3]{5}$

The solution is $x = \sqrt[3]{5}$.
```

```
# query_2 ②
# "x raised to the third plus seven equals 12"라는 입력으로 파이프라인을 실행 → 결과 출력하기
print(runnable.invoke("x raised to the third plus seven equals 12"))
```

- `query_2` / `gpt-4o-mini` - (`1.2s`)

```
EQUATION: \( x^3 + 7 = 12 \)
```

- `gemini`, `gpt`의 호출 구조, 데이터 구조, config 방법 등이 달라 출력이 다르게 생성
- **`아래는 교재에 따라 OpenAI의 방법을 따라 실습 계속 진행 예정`**

---

2) `OpenAI Functions 기능 연결`

- `binding`의 유용한 활용 방법 중 하나 = **`OpenAI`** 모델에 **`OpenAI Functions`**를 **연결하는 것**

- **`OpenAI Functions`**를 **`스키마`에 맞게 정의한 코드**

```
openai_function = {
    "name": "solver",                                                    # 함수의 이름
    # 함수의 설명: "방정식을 수립하고 해결합니다."
    "description": "Formulates and solves an equation",
    # 함수의 매개변수
    "parameters": {
        "type": "object",                                                # 매개변수의 타입: 객체
        "properties": {                                                  # 매개변수의 속성
            "equation": {                                                # 방정식 속성
                "type": "string",                                        # 방정식의 타입: 문자열
                "description": "The algebraic expression of the equation",  # 방정식의 대수식 표현
            },
            "solution": {                                                # 해답 속성
                "type": "string",                                        # 해답의 타입: 문자열
                "description": "The solution to the equation",           # 방정식의 해답
            },
        },
        "required": ["equation", "solution"],                            # 필수 매개변수: 방정식과 해답
    },
}
```

- **`bind()`** 메서드 사용 → **`solver`** 함수 호출 → 모델에 바인딩하기

```
# 프롬프트 생성
prompt = ChatPromptTemplate.from_messages(
    [
        (
            "system",
            "Write out the following equation using algebraic symbols then solve it.",
            # 다음 방정식을 대수 기호를 사용하여 작성한 다음 해결하세요.
        ),
        ("human", "{equation_statement}"),
    ]
)

# LLM 초기화
model = ChatOpenAI(model="gpt-4o-mini", temperature=0).bind(     # openai_function schema 바인딩
    function_call={"name": "solver"},                            # 사전 정의한 openai_function의 name, solver
    functions=[openai_function],
)

# runnable 객체 생성하기
runnable = {"equation_statement": RunnablePassthrough()} | prompt | model
```

```
# 출력하기
runnable.invoke("x raised to the third plus seven equals 12")   # "x의 세제곱에 7을 더하면 12와 같다"
```

- **`model.bind()`** → **`openai_function`** - (`1.7s`)

```
AIMessage(content='', additional_kwargs={'function_call': {'arguments': '{"equation":"x^3 + 7 = 12","solution":"x
```

---

3) `OpenAI tools 연결하기`

- **`OpenAI tools`** **연결 → 활용하는 방법**
  - **`tools`** 객체 = **`OpenAI`** 의 다양한 기능을 간편하게 사용할 수 있도록 도움
  - *예시:* `tool.run` *메서드 호출 → 자연어 질문 입력 → 해당 질문에 대한 답변 생성*

```
# OpenAI tools

tools = [
    {
        "type": "function",
        "function": {
            "name": "get_current_weather",                                  # 현재 날씨를 가져오는 함수의 이름
            "description": "주어진 위치의 현재 날씨를 가져옵니다",                  # 함수에 대한 설명
            "parameters": {
                "type": "object",
                "properties": {
                    "location": {
                        "type": "string",
                        "description": "도시와 주, 예: San Francisco, CA",  # 위치 매개변수에 대한 설명
                    },
                    # 온도 단위 매개변수 (섭씨 또는 화씨)
                    "unit": {"type": "string", "enum": ["celsius", "fahrenheit"]},
                },
                "required": ["location"],                                   # 필수 매개변수 지정
            },
        },
    }
]
```

- **`bind()`** 메서드 → **`tools`** **를 모델에 바인딩**
- **`invoke()`** 메서드 호출 → *"샌프란시스코, 뉴욕, 로스앤젤레스의 현재 날씨에 대해 알려줘?"* *질문 → 모델에 전달*

```
# ChatOpenAI 모델 초기화 → 도구 바인딩하기
model = ChatOpenAI(model="gpt-4o-mini").bind(tools=tools)
```

```
# 모델 호출 → 샌프란시스코, 뉴욕, 로스앤젤레스의 날씨에 대해 질문하기
model.invoke("샌프란시스코, 뉴욕, 로스앤젤레스의 현재 날씨에 대해 알려줘?")
```

- **`model.bind()`** → **`openai tools`** **연결** - (`2.6s`)

```
AIMessage(content='', additional_kwargs={'tool_calls': [{'id': 'call_fqPlWHHjJKwI3LAUXSaFFoU6', 'function': {'arg
```

- next: ***`11. 폴백 (fallback) 모델 지정`***
