

AI기반 데이터 분석 및 AI Agent 개발 과정

『2과목 :』 AI Agent 개발

2025.09.22-10.02(9일, 62시간)

Prepared by Daekyeong Kim

Ph.D.

시간	6일차	7일차	8일차	9일차	Total
1교시 08시30분 - 09시20분	2과목 AI Agent 개발 1장. 과정 소개 LLM 개관	3장. Langchain 기본 - Runnable과 LCEL	4장. AI 에이전트 개발 - 오픈AI의 에이전트 SDK	7장. AI 과정 마무리 - 미니프로젝트	
2교시 09시30분 - 10시20분			- 실습 17: 인사하는 에이전트 - 실습 18: 뉴스 에이전트 만들기 - 실습 19: 가드레일 사용하기 - 실습 20: 핸드로프 활용		
3교시 10시30분 - 11시50분	2장. Langchain 전 프롬프트 엔지니어링 챗GPT 외의 OpenAI 서비스	실습9: Runnable 간단한 예제 실습10: LCEL -랭체인 표현 언어 실습11: Runnable 주요 타입	- 구글의 ADK - 랭그래프		
4교시 13시00분 - 13시50분	-실습 1: Completion API vs Response API -실습 2: 스트리밍 처리 -실습3:비동기 처리 및 오류 핸들링	리트리버와 RAG	- 실습 21:헬로 랭그래프 만들기 - 실습 22: 감정 분석 챗봇에 적용해 보기		
5교시 14시00분 - 14시50분	3장. Langchain 기본 - 채팅 모델 - PromptTemplate과 OutputParser	실습 12: @tool 데코레이터:도구 생성의 표준 방법 실습 13: 임베딩 실습 14: 벡터 스토어 실습 15: 리트리버 실습 16: RAG	5장. AI 에이전트 프로토콜 : MCP와 A2A - AI 에이전트 프로토콜, 클로드 MCP - AI 에이전트 프로토콜, 구글 A2A		
6교시 15시00분 - 15시15분	- 실습 4:랭체인에서 오픈AI의 GPT모델 실행해보기 - 실습 5: 채팅 모델 - 실습 6: 메시지 - 실습 7: PromptTemplate 생성해 보기 - 실습8: PromptTemplate와 OutputParsers	성취도 평가	- 실습 23: MCP 서버 만들기		
Total Time		6	6	6	4 22

『2-1』 실습



실습 1 : Completion API vs Response API

PRD: Chat Completion 기반 AI 비서 프로그램

구분	항목	설명
개요	프로그램 목적	사용자가 입력한 질문에 대해 OpenAI Chat Completion API를 통해 AI 응답을 반환하는 콘솔 프로그램
기능 요구사항	사용자 입력	콘솔에서 질문(prompt)을 문자열로 입력받는다
	Chat Completion API 호출	- model: 기본 "gpt-5-mini" - messages: system, user 역할 포함
	응답 반환	response.choices[0].message.content 값을 추출하여 출력
비기능 요구사항	사용성	CLI 환경에서 직관적 사용 가능
	확장성	모델 교체 용이 (gpt-5-mini → 다른 모델 가능)
	신뢰성	오류 처리 필요 (API Key 미설정, 네트워크 에러 등)
시나리오	사용자 플로우	① 프로그램 실행 → ② 사용자 입력 → ③ API 호출 → ④ AI 응답 출력
	예시	입력: "파이썬에서 리스트 정렬 방법은?" 출력: "파이썬에서는 sort() 메서드나 sorted() 함수를 사용할 수 있습니다."
향후 개선	예외 처리	API 오류/네트워크 장애 시 안내 메시지 제공
	대화 확장	다중 턴 대화 지원(히스토리 유지)
	UI 확장	CLI → GUI 또는 웹 인터페이스로 확장

실습 1 : Completion API vs Response API

hello_openai.py

```
import os
from openai import OpenAI

os.environ["OPENAI_API_KEY"] = "-proj-DRnE5vys0joMbWlqJ4L9m07ta67EFYLcvYi0MSB-4EqCazZ38lIDHzVR0ijGLy8cZuew6yEVhkT3B1bkFJwk1CwB0m_wXbxQZ7ePoA3UnHHqwWJ6UktyaAisceZYm1abuk77cI27E5cunrmsnndg30EELF4A"

# ② OpenAI API 키 가져오기
api_key = os.environ.get("OPENAI_API_KEY")
# ③ OpenAI 클라이언트 초기화
# client = OpenAI() # 이렇게 해도 문제 없음
client = OpenAI(api_key=api_key)
```

실습 1 : Completion API vs Response API

```
def get_chat_completion(prompt, model="gpt-5-mini"):
    # OpenAI Chat Completion API를 사용하여 AI의 응답을 받는 함수"

    # ④ Chat Completion API 호출
    response = client.chat.completions.create(
        model=model,
        messages=[
            {"role": "system", "content": "당신은 친절하고 도움이 되는 AI 비서입니다."},
            {"role": "user", "content": prompt},
        ],
    )

    # ⑤ 응답 텍스트 반환
    return response.choices[0].message.content

if __name__ == "__main__":
    # ⑥ 사용자 입력 받기
    user_prompt = input("AI에게 물어볼 질문을 입력하세요: ")
    # ⑦ AI 응답 받기
    response = get_chat_completion(user_prompt)
    print("\nAI 응답:")
    print(response)
```

실습 1 : Completion API vs Response API

(py3_10_openai) C:\DEV\SamsungElectronics_Gumi\src\AIAgent\chapter2>python hello_openai.py
AI에게 물어볼 질문을 입력하세요: 강남역에서 서울역 가는 지하철 노선을 알려줘.

AI 응답:

추천 경로(간단)

- 강남역 → (지하철 2호선) → 시청역에서 환승 → (지하철 1호선) → 서울역 하차
 - 환승 1회, 소요시간 약 25~30분(대략, 열차 및 환승 대기시간에 따라 변동)

대안

- 강남역 → (2호선) → 신도림역에서 환승 → (1호선) → 서울역
 - 소요시간 약 30~40분

팁

- 시청역에서 1호선 방향으로 환승하면 서울역(1정거장)로 바로 갑니다.
- 서울역에서 KTX/AREX 타실 예정이면 역 내 환승·출구 표지판을 따라가시면 됩니다.

원하시면 출구 번호나 예상 출발 시간(첫/막차), 혹은 도보 환승 동선까지 자세히 알려드릴게요. 어느 정보를 더 원하세요?

(py3_10_openai) C:\DEV\SamsungElectronics_Gumi\src\AIAgent\chapter2>

실습 1 : Completion API...

```
from openai import OpenAI

# 클라이언트 초기화
client = OpenAI(api_key="YOUR_API_KEY")

# ChatCompletion 요청 (n=3)
response = client.chat.completions.create(
    model="gpt-4o-mini",
    messages=[{"role": "user", "content": "인공지능을 한 문장으로 설명해줘."}],
    n=3, # 후보 답변 개수 요청
)

# 전체 응답 출력
print("=== Raw Response ===")
print(response)

# 각 choice 출력
print("\n=== Choices ===")
for i, choice in enumerate(response.choices):
    print(f"\n--- Choice {i} ---")
    print("Index:", choice.index)
    print("Finish reason:", choice.finish_reason)
    print("Message:", choice.message.content)
```


실습 1 : Completion API...

=== Choices ===

--- Choice 0 ---

Index: 0

Finish reason: stop

Message: 인공지능은 데이터를 학습하여 스스로 추론하고 문제를 해결하는 컴퓨터 시스템입니다.

--- Choice 1 ---

Index: 1

Finish reason: stop

Message: 인공지능은 사람처럼 학습하고 추론하며 판단할 수 있는 컴퓨터 기술입니다.

--- Choice 2 ---

Index: 2

Finish reason: stop

Message: 인공지능은 인간의 지능적 행동을 모방해 다양한 문제를 해결하는 기술입니다.

PRD: OpenAI Responses API + 웹 검색 도구를 활용한 요약 응답 시스템

구분	내용
목적 (Objective)	OpenAI Responses API를 활용하여 사용자 프롬프트에 대한 응답을 생성하고, 웹 검색 도구를 함께 사용하여 최신 정보를 포함한 요약 결과를 반환한다.
기능 요구사항 (Functional Requirements)	<ul style="list-style-type: none"> ① 환경 변수에서 OpenAI API 키(OPENAI_API_KEY) 불러오기 ② OpenAI(api_key=...)로 클라이언트 초기화 ③ get_responses(prompt, model) 함수 정의 <ul style="list-style-type: none"> - 입력: 사용자 프롬프트 - 출력: response.output_text (AI의 텍스트 응답) ④ client.responses.create() 호출 시 tools=[{"type": "web_search_preview"}] 지정 → 웹 검색 도구 활성화 ⑤ 스크립트 실행 시 지정된 URL 문서를 요약 요청하고 결과 출력
비기능 요구사항 (Non-Functional Requirements)	<ul style="list-style-type: none"> - Python 환경에서 실행 가능해야 함 - OpenAI API Key 환경변수 필수 - 인터넷 연결 필요 (웹 검색 도구 사용)
시나리오 (Scenarios)	<ul style="list-style-type: none"> - 입력: "Responses API 문서(URL)" → 웹 검색 도구 활성화 → 문서를 읽고 핵심 내용을 요약 → 최종 요약 텍스트 출력
향후 개선 항목 (Future Improvements)	<ul style="list-style-type: none"> - 예외 처리 추가 (API Key 없음, 네트워크 오류, 검색 결과 없음 등) - 출력 포맷(JSON, Markdown 등) 지원 - 다중 문서 입력 및 비교 요약 기능 - 응답 결과를 파일로 저장하는 기능 추가

hello_openai_responses.py

OpenAI API를 사용하여 AI 응답을 받아오는 코드

```
import os
```

```
from openai import OpenAI
```

```
os.environ["OPENAI_API_KEY"] = "-proj-DRnE5vys0joMbWlqJ4L9m07ta67EFYLcvYi0MSB-4EqCazZ38lIDHzVR0ijGLy8cZuew6yEVhkT3B1bkFJwk1CwB0m_wXbxQZ7ePoA3UnHHqwWJ6UktyaAisceZYM1abuk77cI27E5cunrmsnndg30EELF4A"
```

② OpenAI API 키 가져오기

```
api_key = os.environ.get("OPENAI_API_KEY")
```

③ OpenAI 클라이언트 초기화

```
# client = OpenAI() # 이렇게 해도 문제 없음
```

```
client = OpenAI(api_key=api_key) # OpenAI 클라이언트 초기화
```

실습 1 : Completion API vs Response API

```
def get_responses(prompt, model="gpt-5-mini"):
    # ① 입력된 프롬프트에 대한 AI 응답을 받아오는 함수
    # prompt: 사용자 입력 텍스트
    # model: 사용할 AI 모델 (기본값: gpt-5-mini)
    response = client.responses.create(
        model=model, # 사용할 모델 지정
        tools=[{"type": "web_search_preview"}], # ② 웹 검색 도구 활성화
        input=prompt, # 사용자 입력 전달
    )
```

```
    return response.output_text # 텍스트 응답만 반환
```

```
# ③ 스크립트가 직접 실행될 때 실행
if __name__ == "__main__":
    prompt = """
https://platform.openai.com/docs/api-reference/responses/create
    를 읽어서 Responses API에 대해 요약 정리해주세요.
    """

    output = get_responses(prompt)
    print(output) # 결과 출력
```

Responses API용 실험적 웹 검색
도구 타입

아직 "preview" 단계라서 API
변경 가능성이 큼

목적: 모델이 최신/외부 정보를
검색해서 답변 품질을 높이도록
지원

실습 1 : Completion API vs Response API

(py3_10_openai) C:\DEV\SamsungElectronics_Gumi\src\AIAgent\chapter2>python hello_openai_responses.py
요청하신 페이지(Responses API – Create)를 바탕으로 핵심만 한국어로 요약 정리해 드립니다. (요약의 근거: OpenAI 공식 문서). ([platform.openai.com](https://platform.openai.com/docs/api-reference/responses/create))

1) 개요 – 무엇을 하는 API인가

- Responses API의 Create 엔드포인트는 모델에게 요청을 보내 응답을 생성하는 통합 생성 API입니다. 기존의 Chat/Completions API들을 대체하는 형태로 설계되었고, 텍스트뿐 아니라 멀티모달 입력(예: 이미지 + 텍스트)을 지원하는 구조를 제공합니다.
([platform.openai.com](https://platform.openai.com/docs/api-reference/responses/create))

2) 기본 요청 구조 (핵심 필드)

- model: 사용할 모델 지정(필수). ([platform.openai.com](https://platform.openai.com/docs/api-reference/responses/create))
- input (또는 messages 형태): 모델에 전달할 입력(문자열 또는 구조화된 메시지/배열). 멀티모달인 경우 파일/어셋 참조를 함께 보낼 수 있습니다. ([platform.openai.com](https://platform.openai.com/docs/api-reference/responses/create))
- 기타 옵션(주요): max_output_tokens, temperature, top_p, n(생성 후보 수), stop(종료 토큰), logit_bias, user(사용자 식별용), metadata 등. 이들로 출력 길이, 무작위성, 후보 수, 중단 규칙 등을 제어합니다. ([platform.openai.com](https://platform.openai.com/docs/api-reference/responses/create))

3) 출력(응답) 형태

- 반환되는 응답은 보통 응답 ID, 모델, 생성 시간, 사용량(usage) 정보와 함께 'output' 배열(또는 content 블록)을 포함합니다. output 항목은 텍스트 조각뿐 아니라 구조화된 콘텐츠(예: 역할, 타입, 멀티모달 항목)를 담을 수 있습니다.
([platform.openai.com](https://platform.openai.com/docs/api-reference/responses/create))

실습 1 : Completion API vs Response API

4) 스트리밍, 멀티모달, 파일 업로드

- Create 엔드포인트는 스트리밍 응답을 지원하여 실시간으로 생성 결과를 받아 처리할 수 있습니다.
- 이미지 등 멀티모달 데이터를 함께 보내거나 결과로 멀티모달 콘텐츠를 받을 수 있는 확장된 입력/출력 방식을 제공합니다. ([platform.openai.com](https://platform.openai.com/docs/api-reference/responses/create))

5) 사용 예(문서에 있는 핵심 포맷)

- 문서에는 curl/JavaScript/Python 예제가 포함되어 있어, API 키를 헤더에 넣고 POST /v1/responses 로 요청하는 방식이 표준 예시로 제시됩니다. (언어별 SDK 예시는 문서 참조). ([platform.openai.com](https://platform.openai.com/docs/api-reference/responses/create))

6) 실무 팁 (문서 기반 권장사항 요약)

- 응답 길이를 제어하려면 max_output_tokens를 사용하고, 비용·응답 다양성은 temperature/top_p로 조절하세요.
- 여러 후보(n)를 요청하거나 logit_bias로 특정 토큰 선호/비선호를 설정할 수 있습니다.
- 스트리밍을 사용하면 사용자 경험(로딩 지연)을 개선할 수 있으며, 멀티모달 입력을 사용할 때는 적절한 파일/asset 핸들링을 구현해야 합니다. ([platform.openai.com](https://platform.openai.com/docs/api-reference/responses/create))

원문 문서(예제 포함)를 직접 보시길 원하시면 제가 curl / Python / JavaScript 예제를 한 가지 골라서 실제 요청 예시(복사해서 바로 실행 가능한 형태)로 만들어 드리겠습니다. 어느 언어 예제를 원하시나요?

(py3_10_openai) C:\DEV\SamsungElectronics_Gumi\src\AIAgent\chapter2>

PRD: OpenAI 모델 응답을 스트리밍 방식으로 처리하는 두 가지 방법(chat.completions / responses API)

구분	내용
목적 (Objective)	OpenAI API의 스트리밍 모드 를 활용하여 모델 응답을 청크 단위 또는 이벤트 단위로 실시간 출력하고, 최종 응답까지 확인할 수 있는 기능을 구현한다.
기능 요구사항 (Functional Requirements)	<ul style="list-style-type: none"> ① 환경 변수에서 OpenAI API 키(OPENAI_API_KEY) 불러오기 ② OpenAI(api_key=...)로 클라이언트 초기화 ③ stream_chat_completion(prompt, model) 함수: <ul style="list-style-type: none"> - chat.completions.create(stream=True) 활용 - 모델 응답을 청크 단위로 받아와 순차 출력 ④ stream_response(prompt, model) 함수: <ul style="list-style-type: none"> - client.responses.stream(...) 활용 (컨텍스트 매니저 기반) - 스트림 이벤트를 반복 처리, output_text 타입 이벤트 출력 - 최종 응답(get_final_response()) 출력 ⑤ __main__ 실행 시 테스트 프롬프트 전달하여 결과 출력
비기능 요구사항 (Non-Functional Requirements)	<ul style="list-style-type: none"> - Python 환경 + OpenAI SDK 필요 - OpenAI API Key 환경변수 필수 - 스트리밍 응답은 끊김 없이 순차적으로 출력되어야 함 - 최종 응답은 전체 결과를 별도로 확인 가능해야 함
시나리오 (Scenarios)	<ul style="list-style-type: none"> - 입력: "스트리밍이 뭔가요?" → stream_chat_completion 실행 시 모델 답변이 청크 단위로 출력됨 - 입력: "점심 메뉴 추천해주세요." → stream_response 실행 시 이벤트 단위로 응답 출력 후 최종 응답 확인
향후 개선 항목 (Future Improvements)	<ul style="list-style-type: none"> - 출력 시 마크다운 렌더링, 줄바꿈 처리 등 UI 개선 - 네트워크 오류, API rate limit에 대한 예외 처리 추가 - 스트리밍 중간에 사용자 인터럽트(중단) 기능 제공 - 로그 저장 및 분석 기능 확장 (예: 토큰 사용량 추적)

실습 2 : 스트리밍 처리

```
from openai import OpenAI
import os
import rich

os.environ["OPENAI_API_KEY"] = "-proj-DRnE5vys0joMbWlqJ4L9m07ta67EFYLcvYi0MSB-4EqCazZ38lIDHzVR0ijGLy8cZuew6yEVhkT3B1bkFJwk1CwB0m_wXbxQZ7ePoA3UnHHqwWJ6UktyaAisceZym1abuk77cI27E5cunrmsnndg30EELF4A"

# ② OpenAI API 키 가져오기
api_key = os.environ.get("OPENAI_API_KEY")
# ③ OpenAI 클라이언트 초기화
# client = OpenAI() # 이렇게 해도 문제 없음
client = OpenAI(api_key=api_key) # OpenAI 클라이언트 초기화

default_model = "gpt-5-mini"
```


실습 2 : 스트리밍 처리

```
def stream_chat_completion(prompt, model):
    # ① chat.completions API를 사용한 스트리밍 응답 함수
    stream = client.chat.completions.create(
        model=model,
        messages=[{"role": "user", "content": prompt}],
        stream=True, # ② 스트리밍 모드 활성화
    )
    for chunk in stream: # ③ 응답 청크(조각)를 하나씩 처리
        content = chunk.choices[0].delta.content
        if content is not None:
            print(content, end="")

def stream_response(prompt, model):
    # ④ 새로운 responses API를 사용한 스트리밍 함수( 컨텍스트 매니저로 스트림 관리)
    with client.responses.stream(model=model, input=prompt) as stream:
        for event in stream: # ⑤ 스트림에서 발생하는 각 이벤트 처리
            if "output_text" in event.type: # ⑥ 텍스트 출력 이벤트인 경우
                rich.print(event)
    rich.print(stream.get_final_response()) # 최종 응답 출력

if __name__ == "__main__":
    # stream chat completion("스트리밍이 뭔가요?", default model)
    # stream response("점심 메뉴 추천해주세요.", default model)
```

실습 2 : 스트리밍 처리

```
(py3_10_openai) c:\DEV\envs\py3_10_openai>pip install rich==14.0.0
```

```
Collecting rich==14.0.0
```

```
Using cached rich-14.0.0-py3-none-any.whl.metadata (18 kB)
```

```
Collecting markdown-it-py>=2.2.0 (from rich==14.0.0)
```

```
Downloading markdown_it_py-4.0.0-py3-none-any.whl.metadata (7.3 kB)
```

```
Requirement already satisfied: pygments<3.0.0,>=2.13.0 in c:\dev\envs\py3_10_openai\lib\site-packages (from rich==14.0.0) (2.19.1)
```

```
Requirement already satisfied: typing-extensions<5.0,>=4.0.0 in c:\dev\envs\py3_10_openai\lib\site-packages (from rich==14.0.0) (4.13.2)
```

```
Collecting mdurl~=0.1 (from markdown-it-py>=2.2.0->rich==14.0.0)
```

```
Using cached mdurl-0.1.2-py3-none-any.whl.metadata (1.6 kB)
```

```
Using cached rich-14.0.0-py3-none-any.whl (243 kB)
```

```
Downloading markdown_it_py-4.0.0-py3-none-any.whl (87 kB)
```

```
Using cached mdurl-0.1.2-py3-none-any.whl (10.0 kB)
```

```
Installing collected packages: mdurl, markdown-it-py, rich
```

```
Successfully installed markdown-it-py-4.0.0 mdurl-0.1.2 rich-14.0.0
```

```
[notice] A new release of pip is available: 25.1.1 -> 25.2
```

```
[notice] To update, run: python.exe -m pip install --upgrade pip
```

```
(py3_10_openai) c:\DEV\envs\py3_10_openai>
```

실행 시 에러

```
openai.BadRequestError: Error code: 400 - {'error': {'message': 'Your organization must be verified to stream this model. Please go to: https://platform.openai.com/settings/organization/general and click on Verify Organization. If you just verified, it can take up to 15 minutes for access to propagate.', 'type': 'invalid_request_error', 'param': 'stream', 'code': 'unsupported_value'}}
```

그리고, 스트리밍 기능을 쓰려면 조직 인증을 먼저 해야합니다.
스트리밍은 자원 소모, 보안 리스크, 비용 관리 면에서 일반 호출보다 부담이 큼.
따라서 OpenAI는 조직 인증(Verify Organization)을 마친 계정만 사용 가능하게 제한.

OpenAI 설정 페이지
접속

Verify Organization 클릭 후 인증 완료

완료 직후에는 권한 반영에 최대 15분 걸릴 수 있음

https://platform.openai.com/settings/organization/general?utm_source=chatgpt.com

The screenshot shows the 'Organization settings' page on the OpenAI Platform. On the left is a sidebar with navigation links: Settings, Your profile, Organization, General (highlighted), API keys, Admin keys, People, Projects, Billing, Limits, Usage, Data controls (with a green dot), and Project. The main content area is titled 'Organization settings' and contains two sections: 'Details' and 'Verifications'. In the 'Details' section, there is a form for 'Organization name' (containing '주식회사 케이지네트웍스') and 'Organization ID' (containing 'org-Pi50M5MqiPS3MMYLEVE'). A modal window titled 'Verify your identity' is open, displaying the text 'To verify this organization, you'll need to complete an identity check using our partner Persona.' and a URL 'https://withpersona.com/verify?inquiry' with copy and refresh icons. Below the URL are 'Close' and 'Start ID Check' buttons. In the 'Verifications' section, there is a 'Verify Organization' button. Red boxes highlight the 'Verify Organization' button and the 'Start ID Check' button.

Settings

Your profile

Organization

General

API keys

Admin keys

People

Projects

Billing

Limits

Usage

Data controls ●

Project

Organization settings

Details

Organization name
Human-friendly label for your organization

주식회사 케이지네트웍스

Organization ID
Identifier for this organization something

org-Pi50M5MqiPS3MMYLEVE

Verifications

Verify your organization to access protected models

Verify Organization

Verify your identity

To verify this organization, you'll need to complete an identity check using our partner Persona.

<https://withpersona.com/verify?inquiry>

Close **Start ID Check**

https://platform.openai.com/settings/organization/general?utm_source=chatgpt.com

OpenAI

시작하기

"인증 시작"을 선택하면 OpenAI는 귀하의 계정 정보를 신원 확인 및 신원 사기 방지를 위해 OpenAI의 공급업체인 Persona와 공유합니다. OpenAI가 Persona와 공유한 계정 정보는 최대 7일 동안만 저장됩니다.

또한 아래 체크박스를 선택하시면 Persona가 신원 확인, 신원 사기 방지 및 Persona 플랫폼의 품질 보증을 위해 생체 정보를 수집, 사용 및 서비스 제공업체를 활용하는 데 동의하는 것으로 간주됩니다. 이는 [개인정보처리방침](#) 및 [OpenAI 개인정보처리방침](#)에 따라 이루어집니다. 귀하의 생체 정보는 최대 1년 동안 보관됩니다.

☒ 본인은 생체정보 처리에 동의합니다.

확인 시작

한국어

SECURED WITH
PERSONA

다른 기기에서 계속

신원을 확인하려면 카메라 사용 권한이 필요합니다. 휴대전화에서 계속할 수 있도록 보안 링크를 보내드립니다. 앱 다운로드가 필요하지 않습니다.



<https://go.persona.na/hHPLw>

이메일로 링크 받기

<

다른 기기에서 계속

신원을 확인하려면 카메라 사용 권한이 필요합니다. 사진 촬영을 위한 보안 링크를 이메일로 보내드립니다. 앱 다운로드가 필요하지 않습니다.

이메일 주소

you@example.com

보안 링크 보내기

자세한 내용은 Persona의 [서비스 약관](#) 및 [개인 정보 취급 방침](#)을 참조하세요.

https://platform.openai.com/settings/organization/general?utm_source=chatgpt.com



계속해서 신원을 확인하세요

OpenAI 에 대한 신원 확인을 계속하려면 아래 링크를 클릭하세요.

이 링크는 3일과1분 후에 만료됩니다.

신원 확인

여기서 부터는 핸드폰에서 ...

다른 기기에서 계속

신원을 확인하려면 카메라 사용 권한이 필요합니다.
휴대전화에서 계속할 수 있도록 보안 링크를 보내드립니다. 앱 다운로드가 필요하지 않습니다.



<https://go.perso.na/yZ6Qbw>

이메일로 링크 받기



축하합니다. 완료되었습니다!

신원을 확인 해주셔서 감사합니다.



실습 2 : 스트리밍 처리

```
(py3_10_openai) c:\DEV\envs\py3_10_openai>python hello_openai_streaming.py
```

```
...
```

```
text=ResponseTextConfig(format=ResponseFormatText(type='text'), verbosity='medium'),
    truncation='disabled',
    usage=ResponseUsage(
        input_tokens=13,
        input_tokens_details=InputTokensDetails(cached_tokens=0),
        output_tokens=753,
        output_tokens_details=OutputTokensDetails(reasoning_tokens=320),
        total_tokens=766
    ),
    user=None,
    background=False,
    max_tool_calls=None,
    prompt_cache_key=None,
    safety_identifier=None,
    service_tier='default',
    store=True,
    top_logprobs=0)
```

```
(py3_10_openai) C:\DEV\SamsungElectronics_Gumi\src\AIAgent\chapter2>
```

```
pip install anthropic
pip install tenacity
```

```
(py3_10_openai) c:\DEV\envs\py3_10_openai>pip install anthropic
Collecting anthropic
  Downloading anthropic-0.64.0-py3-none-any.whl.metadata (27 kB)
Requirement already satisfied: anyio<5,>=3.5.0 in c:\dev\envs\py3_10_openai\lib\site-packages (from anthropic) (4.10.0)
Requirement already satisfied: distro<2,>=1.7.0 in c:\dev\envs\py3_10_openai\lib\site-packages (from ...
Installing collected packages: anthropic
Successfully installed anthropic-0.64.0

[notice] A new release of pip is available: 25.1.1 -> 25.2
[notice] To update, run: python.exe -m pip install --upgrade pip

(py3_10_openai) c:\DEV\envs\py3_10_openai>
```

```
pip install anthropic
pip install tenacity
```

```
(py3_10_openai) c:\DEV\envs\py3_10_openai>pip install tenacity
Collecting tenacity
  Downloading tenacity-9.1.2-py3-none-any.whl.metadata (1.2 kB)
  Downloading tenacity-9.1.2-py3-none-any.whl (28 kB)
Installing collected packages: tenacity
Successfully installed tenacity-9.1.2

[notice] A new release of pip is available: 25.1.1 -> 25.2
[notice] To update, run: python.exe -m pip install --upgrade pip

(py3_10_openai) c:\DEV\envs\py3_10_openai>
```

PRD: 멀티 모델 동시 호출 및 결과 비교

구분	내용
목적 (Objective)	OpenAI와 Anthropic(Claude) API를 비동기(async) 방식으로 동시에 호출하여 동일한 프롬프트에 대한 두 모델의 응답을 비교한다.
기능 요구사항 (Functional Requirements)	<ul style="list-style-type: none">① AsyncOpenAI와 AsyncAnthropic 클라이언트 초기화② call_async_openai(prompt, model) 함수: OpenAI 모델 호출, 사용자 프롬프트 응답 반환③ call_async_claude(prompt, model) 함수: Claude 모델 호출, 사용자 프롬프트 응답 반환④ asyncio.gather()를 사용하여 두 API를 병렬로 실행⑤ 두 모델의 응답을 콘솔에 출력
비기능 요구사항 (Non-Functional Requirements)	<ul style="list-style-type: none">- Python 비동기(async/await) 환경 지원- OpenAI 및 Anthropic API Key 환경변수 필요- 네트워크 I/O 대기시간을 효율적으로 처리하여 실행 속도 단축
시나리오 (Scenarios)	<ul style="list-style-type: none">- 입력: "비동기 프로그래밍에 대해 2-3문장으로 설명해주세요."→ OpenAI와 Claude가 동시에 호출됨→ 각 모델의 응답이 콘솔에 나란히 출력됨
향후 개선 항목 (Future Improvements)	<ul style="list-style-type: none">- 에러/타임아웃 처리 강화- 응답 시간을 측정하여 성능 비교- 여러 프롬프트를 한 번에 실행하여 결과 배치 비교- 결과를 JSON 또는 파일로 저장

async await를 사용하여 비동기 처리하기

OpenAI API를 사용하여 AI 응답을 받아오는 코드

```
import asyncio
```

```
import os
```

```
from openai import OpenAI
```

```
os.environ["OPENAI_API_KEY"] = "proj-DRnE5vys0joMbWlqJ4L9m07ta67EFYLcvYi0MSB-4EqCazZ38lIDHzVR0ijGLy8cZuew6yEVhkT3B1bkFJwk1CwB0m_wXbxQZ7ePoA3UnHHqWJ6UktyaAisceZYm1abuk77cI27E5cunrmsnndg30EELF4A"
```

```
os.environ["ANTHROPIC_API_KEY"] = "ant-api03-uF4-a-kR3DQhVaEWIFmrVhwJ3h-b9p8_qyJg_L5XSyirHfhGUtNUiDbxp-dQMfmDGwWqo-6PMm4htclG9fKycA-vT4knQAA"
```

```
from openai import AsyncOpenAI
```

```
from anthropic import AsyncAnthropic
```

```
openai_client = AsyncOpenAI(api_key=os.environ.get("OPENAI_API_KEY"))
```

```
claude_client = AsyncAnthropic(api_key=os.environ.get("ANTHROPIC_API_KEY"))
```

실습 3: 비동기 처리 및 오류 핸들링

```
async def call_async_openai(prompt: str, model: str = "gpt-5-mini") -> str:
    response = await openai_client.chat.completions.create(
        model=model, messages=[{"role": "user", "content": prompt}]
    )
    return response.choices[0].message.content

async def call_async_claude(prompt: str, model: str = "claude-3-5-haiku-latest") -> str:
    response = await claude_client.messages.create(
        model=model, max_tokens=1000, messages=[{"role": "user", "content": prompt}]
    )
    return response.content[0].text

async def main():
    print("동시에 API 호출하기")
    prompt = "비동기 프로그래밍에 대해 2-3문장으로 설명해주세요."
    openai_task = call_async_openai(prompt)
    claude_task = call_async_claude(prompt)
    openai_response, claude_response = await asyncio.gather(openai_task, claude_task)
    print(f"OpenAI 응답: {openai_response}")
    print(f"Claude 응답: {claude_response}")

if __name__ == "__main__":
    asyncio.run(main())
```

실습 3: 비동기 처리 및 오류 핸들링

(py3_10_openai) C:\DEV\SamsungElectronics_Gumi\src\AIAgent\chapter2>python llm_api_async.py
동시에 API 호출하기

OpenAI 응답: 비동기 프로그래밍은 긴 작업(예: 네트워크 요청이나 파일 I/O)을 기다리는 동안 해당 스레드가 멈추지 않고 다른 일을 계속 처리하도록 하는 방식으로, 작업 완료를 콜백·프로미스·async/await·이벤트 루프 등으로 관리합니다. 이를 통해 응답성과 자원 효율을 높일 수 있지만, 동시성 문제와 디버깅 복잡성이 증가할 수 있습니다.

Claude 응답: 비동기 프로그래밍은 작업을 병렬로 처리하여 전체 프로그램의 성능과 응답성을 향상시키는 프로그래밍 기법입니다. 주로 네트워크 요청, 파일 I/O, 데이터베이스 작업과 같이 시간이 오래 걸리는 작업을 수행할 때 메인 스레드를 블로킹하지 않고 다른 작업을 진행할 수 있게 해줍니다. 대표적으로 콜백, 프로미스, async/await 등의 방식으로 구현되며, JavaScript, Python 등 현대적인 프로그래밍 언어에서 널리 사용됩니다.

(py3_10_openai) C:\DEV\SamsungElectronics_Gumi\src\AIAgent\chapter2>

실습 3: 비동기 처리 및 오류 핸들링

방식	코드 실행 흐름	특징
동기(Sync)	요청 보냄 → 응답 올 때까지 멈춤 → 결과 반환 → 다음 코드 실행	직관적, 구현 쉬움, 하지만 병렬 처리 불가
비동기(Async)	요청 보냄 → 기다리는 동안 다른 작업 실행 → 응답 오면 이어서 처리	효율적, 병렬 처리 가능, 다만 async/await 문법 필요

실습 3: 비동기 처리 및 오류 핸들링

같은 동작을 동기 처리로 작성하면

```
def call_sync_openai(prompt: str, model: str = "gpt-4o-mini") -> str:
    # API 요청을 보내고 응답이 돌아올 때까지 프로그램이 멈춤
    response = openai_client.chat.completions.create(
        model=model,
        messages=[{"role": "user", "content": prompt}]
    )
    return response.choices[0].message.content
```

```
def call_sync_claude(prompt: str, model: str = "claude-3-5-haiku-latest") -> str:
    response = claude_client.messages.create(
        model=model,
        max_tokens=1000,
        messages=[{"role": "user", "content": prompt}]
    )
    return response.content[0].text
```

API 응답이 끝날 때까지 해당 함수가 블로킹(blocking) 되므로, 다른 코드는 그동안 실행되지 못함.

PRD: 간헐적으로 실패하는 함수 만들기

구분	내용
목적 (Objective)	OpenAI와 Claude API를 비동기적으로 동시에 호출하고, 네트워크 오류나 간헐적 실패 발생 시 **재시도 로직(지수 백오프 포함)**을 적용하여 안정성을 높인다.
기능 요구사항 (Functional Requirements)	<ul style="list-style-type: none">① call_async_openai: OpenAI API 호출 (랜덤 실패 시뮬레이션 + 재시도 지원)② call_async_claude: Claude API 호출③ simulate_random_failure: 50% 확률로 실패 발생시켜 재시도 로직 검증④ tenacity.retry 적용 → 최대 3회 재시도, 지수 백오프(2초→4초→8초)⑤ asyncio.gather로 두 API 병렬 실행 및 응답 출력
비기능 요구사항 (Non-Functional Requirements)	<ul style="list-style-type: none">- Python 비동기(async/await) 기반- 재시도 과정 로깅(logging) 필수- 네트워크 지연/실패에 견딜 수 있도록 안정적 처리- 테스트 가능한 구조(랜덤 실패 시뮬레이션 포함)
시나리오 (Scenarios)	<ul style="list-style-type: none">- 정상 상황: OpenAI/Claude 모두 첫 시도에 성공 → 결과 즉시 출력- 간헐적 실패: OpenAI 호출 실패 → 3회까지 자동 재시도, 성공 시 결과 출력- 연속 실패: 3회 모두 실패 → 예외 발생 후 로깅
향후 개선 항목 (Future Improvements)	<ul style="list-style-type: none">- 예외 유형별 맞춤 재시도 정책 적용 (예: 네트워크 오류만 재시도)- 실패 시 fallback 모델 사용 (Claude 실패 시 OpenAI만 결과 반환 등)- 호출 응답 시간 및 실패율 모니터링 지표 추가- 응답 저장/로그 아카이빙 기능 강화

간헐적으로 실패하는 함수 만들기

```
# OpenAI API를 사용하여 AI 응답을 받아오는 코드
import asyncio
import os
import logging
import random
from openai import OpenAI
```

```
os.environ["OPENAI_API_KEY"] = "proj-DRnE5vys0joMbWlqJ4L9m07ta67EFYLcvYi0MSB-4EqCazZ38lIDHzVR0ijGLy8cZuew6yEVhkT3B1bkFJwk1CwB0m_wXbxQZ7ePoA3UnHHqwWJ6UktyaAisceZym1abuk77cI27E5cunrmsnndg30EELF4A"
```

```
from openai import AsyncOpenAI
from anthropic import AsyncAnthropic
from tenacity import (
    retry,
    stop_after_attempt,
    wait_exponential,
    retry_if_exception_type,
)
```

tenacity는 파이썬에서 재시도 로직(retry logic)을 쉽게 구현할 수 있게 해주는 라이브러리

실습 3: 비동기 처리 및 오류 핸들링

로깅 설정

```
logging.basicConfig(level=logging.INFO)
logger = logging.getLogger(__name__)
```

① 비동기 클라이언트 생성

```
openai_client = AsyncOpenAI(api_key=os.environ.get("OPENAI_API_KEY"))
claude_client = AsyncAnthropic(api_key=os.environ.get("ANTHROPIC_API_KEY"))
```

테스트용 간헐적 실패 시뮬레이션 함수

```
async def simulate_random_failure():
```

```
    # 50% 확률로 실패 발생시키기
```

```
    if random.random() < 0.5:
```

```
        logger.warning("인위적으로 API 호출 실패 발생 (테스트용)")
```

```
        raise ConnectionError("인위적으로 발생시킨 연결 오류 (테스트용)")
```

```
    # 약간의 지연시간 추가
```

```
    await asyncio.sleep(random.uniform(0.1, 0.5))
```

실습 3: 비동기 처리 및 오류 핸들링

② tenacity를 사용한 재시도 데코레이터 적용

```
@retry(
    stop=stop_after_attempt(3), # 최대 3번 시도
    wait=wait_exponential(multiplier=1, min=2, max=10), # 지수 백오프: 2초, 4초, 8초...
    retry=retry_if_exception_type(), # 모든 예외에 대해 재시도
    before_sleep=lambda retry_state: logger.warning(
        f"API 호출 실패: {retry_state.outcome.exception()}, {retry_state.attempt_number}번째 재시도 중..."
    ),
)

async def call_async_openai(prompt: str, model: str = "gpt-5-mini") -> str:
    # ③ await를 사용해 비동기적으로 API 응답을 기다림
    logger.info(f"OpenAI API 호출 시작: {model}")

    # 테스트를 위한 랜덤 실패 시뮬레이션
    await simulate_random_failure()

    response = await openai_client.chat.completions.create(
        model=model,
        messages=[{"role": "user", "content": prompt}],
    )
    logger.info("OpenAI API 호출 성공")
    return response.choices[0].message.content
```

실습 3: 비동기 처리 및 오류 핸들링

```
async def call_async_claude(prompt: str, model: str = "claude-3-5-haiku-latest") -> str:
    logger.info(f"Claude API 호출 시작: {model}")
    response = await claude_client.messages.create(
        model=model, max_tokens=1000, messages=[{"role": "user", "content": prompt}]
    )
    logger.info("Claude API 호출 성공")
    return response.content[0].text
```

```
async def main():
    print("동시에 API 호출하기 (재시도 로직 포함)")
    prompt = "비동기 프로그래밍에 대해 2-3문장으로 설명해주세요."
    # ⑥ 비동기 함수 호출 시 코루틴 객체 반환(실행은 아직 안 됨)
    openai_task = call_async_openai(prompt)
    claude_task = call_async_claude(prompt)

    try:
        # ⑦ 두 API 호출을 병렬로 실행하고 둘 다 완료될 때까지 대기
        # gather는 전체 작업 중 하나라도 실패하면 예외 발생
        openai_response, claude_response = await asyncio.gather(
            openai_task, claude_task, return_exceptions=False
        )
        print(f"OpenAI 응답: {openai_response}")
        print(f"Claude 응답: {claude_response}")
    except Exception as e:
        logger.error(f"API 호출 중 처리되지 않은 오류 발생: {e}")
```

```
if __name__ == "__main__":
    asyncio.run(main()) # ⑧ 비동기 메인 함수를 이벤트 루프에서 실행
```

• call_async_openai() 내부

- logger.info("OpenAI API 호출 시작: ...")
- simulate_random_failure() 호출
 - 50% 확률로 ConnectionError 발생 (테스트용)
 - 실패 시 tenacity가 최대 3회 재시도:
 - 대기: 2초 → 4초 (지수 백오프, min=2, max=10)
 - 재시도 직전 매번 경고 로그 출력:

API 호출 실패: <예외 메시지>, 1번째 재시도 중... API 호출 실패: <예외 메시지>, 2번째 재시도 중...

- 성공하면(오류 안 나면) 0.1~0.5초 지연 후 다음 단계 진행

• OpenAI API 호출 시도 → 성공 시 logger.info("OpenAI API 호출 성공")

• call_async_claude()는 재시도 없이 한 번 호출:

- logger.info("Claude API 호출 시작: ...")
- 성공 시 logger.info("Claude API 호출 성공")

실습 3: 비동기 처리 및 오류 핸들링

```
(py3_10_openai) C:\DEV\SamsungElectronics_Gumi\src\AIAgent\chapter2>python async_llm_api.py
동시에 API 호출하기 (재시도 로직 포함)
INFO:__main__:OpenAI API 호출 시작: gpt-5-mini
WARNING:__main__:인위적으로 API 호출 실패 발생 (테스트용)
WARNING:__main__:API 호출 실패: 인위적으로 발생시킨 연결 오류 (테스트용), 1번째 재시도 중...
INFO:__main__:Claude API 호출 시작: claude-3-5-haiku-latest
ERROR:__main__:API 호출 중 처리되지 않은 오류 발생: "Could not resolve authentication method. Expected either
api_key or auth_token to be set. Or for one of the `X-API-Key` or `Authorization` headers to be explicitly
omitted"

(py3_10_openai) C:\DEV\SamsungElectronics_Gumi\src\AIAgent\chapter2>
```



실습 4: 랑체인에서 오픈AI의 GPT모델 실행해보기

TensorFlow 2 가상환경

```
c:\DEV>cd envs
```

```
c:\DEV\envs>python -m venv py3_10_langchain
```

```
c:\DEV\envs>cd py3_10_langchain
```

```
c:\DEV\envs\py3_10_langchain>Scripts\activate
```

```
(
```

```
(py3_10_langchain) c:\DEV\envs\py3_10_langchain>dir/w/p
```

C 드라이브의 볼륨에는 이름이 없습니다.

볼륨 일련 번호: BED0-C858

```
c:\DEV\envs\py3_10_langchain 디렉터리
```

```
[.]      [..]      [Include] [Lib]      pyvenv.cfg [Scripts]
```

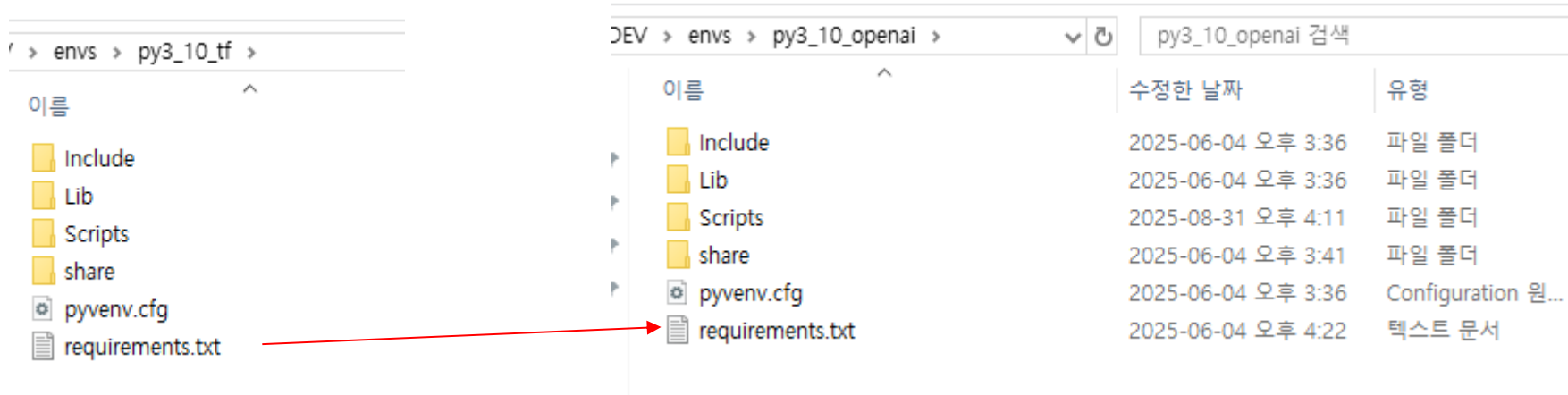
1개 파일 118 바이트

5개 디렉터리 56,743,809,024 바이트 남음

```
(py3_10_langchain) c:\DEV\envs\py3_10_langchain>
```


실습 4: 랭체인에서 오픈AI의 GPT모델 실행해보기

requirements.txt 복사



(py3_10_langchain) c:\DEV\py3_10_langchain>dir/w/p

C 드라이브의 볼륨에는 이름이 없습니다.
볼륨 일련 번호: BED0-C858

c:\DEV\py3_10_langchain 디렉터리

```
[.]          [..]          [Include]  [Lib]      pyvenv.cfg  requirements.txt
[Scripts]
      2개 파일           953 바이트
      5개 디렉터리 68,334,325,760 바이트 남음
```

(py3_10_langchain) c:\DEV\py3_10_langchain>



실습 4: 랭체인에서 오픈AI의 GPT모델 실행해보기

requirements.txt 복사

(py3_10_langchain) c:\DEV\py3_10_langchain>pip install -r requirements.txt

```
Collecting asttokens==2.4.1
  Using cached asttokens-2.4.1-py2.py3-none-any.whl (27 kB)
Collecting colorama==0.4.6
  Using cached colorama-0.4.6-py2.py3-none-any.whl (25 kB)
Collecting comm==0.2.2
  Using cached comm-0.2.2-py3-none-any.whl (7.2 kB)
Collecting contourpy==1.2.1
  Downloading contourpy-1.2.1-cp310-cp310-win_amd64.whl (187 kB)
----- 187.5/187.5 kB 5.7 MB/s eta 0:00:00
Collecting cycler==0.12.1
  Downloading cycler-0.12.1-py3-none-any.whl (8.3 kB)
```



실습 4: 랭체인에서 오픈AI의 GPT모델 실행해보기

쥬피터 노트북 내보내기

```
pip install ipykernel  
python.exe -m pip install --upgrade pip
```

```
(py3_10_langchain) c:\DEV\py3_10_langchain>python -m ipykernel install --user --name  
py3_10_langchain --display-name "py3_10_langchain"  
Installed kernelspec py3_10_langchain in  
C:\Users\k8s\AppData\Roaming\jupyter\kernels\py3_10_langchain
```

```
(py3_10_langchain) c:\DEV\envs\py3_10_langchain>deactivate  
c:\DEV\envs\py3_10_langchain>cd ..
```

```
c:\DEV\envs>cd ..
```

```
c:\DEV>jupyter notebook
```



실습 4: 랭체인에서 오픈AI의 GPT모델 실행해보기

라이브러리

```
import sys
print("python 버전 : {}".format(sys.version))
import numpy as np
print("numpy 버전 : {}".format(np.__version__))
import pandas as pd
print("pandas 버전 : {}".format(pd.__version__))
import matplotlib
print("matplotlib 버전 : {}".format(matplotlib.__version__))
import scipy as sp
print("scipy 버전 : {}".format(sp.__version__))
import IPython
print("IPython 버전 : {}".format(IPython.__version__))
import sklearn
print("sklearn : {}".format(sklearn.__version__))
```

```
python 버전 : 3.10.11 (tags/v3.10.11:7d4cc5a, Apr  5 2023, 00:38:17) [MSC v.1929 64 bit (AMD64)]
numpy 버전 : 2.0.0
pandas 버전 : 2.2.2
matplotlib 버전 : 3.9.0
scipy 버전 : 1.13.1
IPython 버전 : 8.25.0
sklearn : 1.5.0
```

랭체인 사전 준비

(c) Microsoft Corporation. All rights reserved.

```
C:\Users\k8s>cd c:\dev
```

```
c:\DEV>dir/w/p
```

C 드라이브의 볼륨에는 이름이 없습니다.

볼륨 일련 번호: BED0-C858

c:\DEV 디렉터리

[.]	[..]	[.ipynb_checkpoints]
[envs]	[Erudite_demo]	[IntelliJ_pro]
[jdk-24.0.1]	[ollama_pro]	openjdk-24.0.1_windows-x64_bin.zip
[PycharmProjects]	PycharmProjects.zip	[r-workspaces]
[SamsungDisplay]	SamsungDisplay-main.zip	[SamsungElectronics_Gumi]
[sqlite-tools-win-x64-3490200]	sqlite-tools-win-x64-3490200.zip	[Tools]
[web_app_project]		
4개 파일	443,259,906 바이트	
15개 디렉터리	9,454,661,632 바이트 남음	

```
c:\DEV>cd envs
```

랭체인 사전 준비

```
c:\DEV\envs>dir/w/p
```

c 드라이브의 볼륨에는 이름이 없습니다.
볼륨 일련 번호: BED0-C858

```
c:\DEV\envs 디렉터리
```

```
[.]                [..]                [mcp-env]                [py3_10_basic]        [py3_10_langchain] [py3_10_ollama]
[py3_10_openai]    [py3_10_pt]                [py3_10_tf]
                  0개 파일                0 바이트
                  10개 디렉터리    9,454,661,632 바이트 남음
```

```
c:\DEV\envs>cd py3_10_langchain
```

```
c:\DEV\envs\py3_10_langchain>Scripts\activate
```

```
(py3_10_langchain) c:\DEV\envs\py3_10_langchain>
```



랭체인 사전 준비

- 랭체인 라이브러리 설치

```
(py3_10_langchain) c:\DEV\envs\py3_10_langchain>pip install "langchain[openai]"==0.3.27
```

```
(py3_10_langchain) c:\DEV\envs\py3_10_langchain>pip show openai
```

```
Name: openai
```

```
Version: 1.84.0
```

```
...
```

```
(py3_10_langchain) c:\DEV\envs\py3_10_langchain>pip show langchain
```

```
Name: langchain
```

```
Version: 0.3.27
```

```
..
```

```
(py3_10_langchain) c:\DEV\envs\py3_10_langchain>
```

PRD: LangChain 모델 초기화 + 단일 질의응답 테스트"를 위한 최소 기능 정의

구분	내용
목적 (Objective)	LangChain의 ChatOpenAI 모델을 초기화하고, 간단한 질의응답을 실행하여 모델의 응답 객체 구조와 출력값을 확인한다.
기능 요구사항 (Functional Requirements)	<ul style="list-style-type: none">① ChatOpenAI(model="gpt-5-mini")로 LLM 모델 인스턴스 생성② model.invoke("랭체인이 뭔가요?") 호출하여 응답 생성③ 응답 객체의 타입(type(result)) 확인④ 응답 텍스트(result.content) 출력
비기능 요구사항 (Non-Functional Requirements)	<ul style="list-style-type: none">- Python + LangChain 환경에서 실행 가능해야 함- OpenAI API Key 사전 설정 필요- 모델 응답은 일관적이고 사람이 읽을 수 있어야 함
시나리오 (Scenarios)	<ul style="list-style-type: none">- 사용자 입력: "랭체인이 뭔가요?"→ 모델이 LangChain 개념을 간단히 설명하는 답변 생성→ 콘솔에 객체 타입과 응답 텍스트 출력
향후 개선 항목 (Future Improvements)	<ul style="list-style-type: none">- 응답 토큰 수, 생성 시간 등 메타데이터 추가 확인- 비동기 호출(ainvoke) 지원- 여러 메시지 대화형 입력(messages API) 확장- 출력 포맷(JSON, Markdown) 지정 기능 추가

● hello_langchain.py

```
import os
from langchain_openai import ChatOpenAI

# 환경변수 설정 (실제로는 .env 사용 권장)
os.environ["OPENAI_API_KEY"] = "proj-DRnE5vys0joMbWlqJ4L9m07ta67EFYLCvYi0MSB-4EqCazZ38lIDHzVR0ijGLy8cZuew6yEVhkT3B1bkFJwk1CwB0m_wXbxQZ7ePoA3UnHHqwWJ6UktyaAisceZYm1abuk77cI27E5cunrmsnndg30EELF4A"

# 모델 초기화
model = ChatOpenAI(model="gpt-5-mini")

result = model.invoke("랭체인이 뭔가요?")
print(type(result))
print(result.content)
```

invoke() 결과는 AIMessage 객체

내용은 result.content에 들어 있음

```
print(type(result))
```

```
# <class 'langchain.schema.messages.AIMessage'>
```

```
print(result.content)
```

```
# "LangChain은 LLM을 활용한 애플리케이션 개발을  
쉽게 해주는 프레임워크입니다..."
```

실습 4: 랭체인에서 오픈AI의 GPT 모델 실행해보기

```
(py3_10_langchain) C:\DEV\SamsungElectronics_Gumi>cd src
(py3_10_langchain) C:\DEV\SamsungElectronics_Gumi\src>cd AIAgent
(py3_10_langchain) C:\DEV\SamsungElectronics_Gumi\src\AIAgent>
(py3_10_langchain) C:\DEV\SamsungElectronics_Gumi\src\AIAgent>cd chapter3
(py3_10_langchain) C:\DEV\SamsungElectronics_Gumi\src\AIAgent\chapter3>python hello_langchain.py
<class 'langchain_core.messages.ai.AIMessage'>
```

랭체인(langchain)은 자연어 처리(NLP) 및 인공지능(AI) 모델을 활용하여 다양한 애플리케이션과 시스템을 구축할 수 있도록 지원하는 프레임워크입니다. 주로 언어 모델과의 상호작용을 쉽게 만들어주는 도구들로 구성되어 있으며, 데이터베이스나 API와의 통합, 대화형 애플리케이션 개발, 특정 작업에 맞는 프롬프트 설계 등을 가능하게 합니다.

랭체인은 일반적으로 다음과 같은 기능을 포함합니다:

1. 다양한 언어 모델 지원: GPT, 문맥 기반 모델 등 여러 종류의 언어 모델을 사용할 수 있도록 함.
2. 자료 저장 및 관리: 데이터를 효율적으로 저장하고 관리하는 방법을 제공, 검색 기능 등 포함.
3. 프롬프트 템플릿: 효과적인 프롬프트 디자인을 위한 템플릿이나 가이드라인 제공.
4. 워크플로우 관리: 복잡한 프로세스를 관리하고 최적화하는 기능.

랭체인은 특히 챗봇, 추천 시스템, 문서 요약, 데이터 질의 등 다양한 AI 기반 애플리케이션을 만들기 위한 유용한 도구로 사용되고 있습니다.

```
(py3_10_langchain) C:\DEV\SamsungElectronics_Gumi\src\AIAgent\chapter3>
```

실습 4: 랭체인에서 오픈AI의 GPT 모델 실행해보기

invoke와 다른 실행 방식 비교

- invoke는 LangChain에서 입력 하나를 넣고 동기적으로 결과 하나를 받는 함수
- 반환값은 보통 AIMessage 객체 → .content로 텍스트 확인
- 여러 입력/비동기/스트리밍을 원하면 .batch(), .ainvoke(), .stream() 사용

메서드	특징	사용 예
<code>.invoke(input)</code>	단일 입력 → 단일 출력	<code>result = model.invoke("안녕")</code>
<code>.batch([inputs])</code>	여러 입력 → 여러 출력	<code>results = model.batch(["안녕", "헬로"])</code>
<code>.stream(input)</code>	스트리밍 출력 (chunk 단위)	<code>for chunk in model.stream("안녕"):</code>
<code>.ainvoke(input)</code>	비동기(async) 단일 실행	<code>await model.ainvoke("안녕")</code>

실습 4: 랭체인에서 오픈AI의 GPT 모델 실행해보기

AIMessage 주요 속성

LangChain 0.2.x 기준으로 AIMessage는 대략 이런 구조를 가집니다:

```
AIMessage(  
    content="모델이 생성한 답변 텍스트",  
    id="메시지 고유 ID",  
    example=False,  
    tool_calls=[],  
    additional_kwargs={}  
)
```

- `result.content` = 답변 텍스트
- `result.id` = 메시지 ID
- `result.tool_calls` = 모델이 호출한 툴 관련 정보
- `result.additional_kwargs` = 원본 응답의 부가 정보

PRD: 랜덤 모델 선택 후 질문 실행 테스트

구분	내용
목적 (Objective)	실행 시 무작위로 다른 LLM 모델(OpenAI gpt-5-mini 또는 gpt-4o-mini)을 선택하여, 동일한 질문 "RAG가 뭔가요?"에 대한 응답을 생성한다.
기능 요구사항 (Functional Requirements)	<ul style="list-style-type: none"> ① ChatOpenAI(model="gpt-5-mini")로 기본 모델 초기화 ② random.random()을 사용하여 50% 확률로 다른 모델 선택 <ul style="list-style-type: none"> - 조건 충족 시: "gpt-4o-mini selected" 출력 후 해당 모델 사용 - 조건 불충족 시: "claude-sonnet-4-20250514 selected" 메시지만 출력 (실제 Claude API 연동은 없음) ③ 최종 선택된 모델에 "RAG가 뭔가요?"라는 질문을 전달 ④ 결과 응답(result.content)을 출력
비기능 요구사항 (Non-Functional Requirements)	<ul style="list-style-type: none"> - Python + LangChain 환경에서 실행 가능 - OpenAI API Key 필요 - 무작위 분기 실행으로 모델 다양성 확보
시나리오 (Scenarios)	<ul style="list-style-type: none"> - 실행 시 "gpt-5-mini selected" 또는 "claude-sonnet-4-20250514 selected" 메시지 출력 - 선택된 모델이 "RAG(Retrieval-Augmented Generation)" 개념을 설명하는 응답 반환
향후 개선 항목 (Future Improvements)	<ul style="list-style-type: none"> - Claude 모델 실제 연동 추가 (AsyncAnthropic 사용) - 모델 선택 확률을 가중치 기반으로 조정 가능 - 응답 결과 비교/로그 저장 기능 추가 - 사용자 입력에 따라 모델을 동적으로 선택하도록 개선

● langchain_chat_model.py

```
import random
import os
from langchain_openai import ChatOpenAI

# 환경변수 설정 (실제로는 .env 사용 권장)
os.environ["OPENAI_API_KEY"] = "proj-DRnE5vys0joMbWlqJ4L9m07ta67EFYLcvYi0MSB-
4EqCazZ38lIDHzVR0ijGLy8cZuew6yEVhkT3B1bkFJwk1CwB0m_wXbxQZ7ePoA3UnHHqwWJ6UktyaAisceZYM1abuk77cI27E5cunrmsndg
30EELF4A"

# 모델 초기화
model = ChatOpenAI(model="gpt-5-mini")

if random.random() < 0.5:
    print("gpt-5-mini selected")
    model = ChatOpenAI(model="gpt-4o-mini")
else:
    print("claude-sonnet-4-20250514 selected")
result = model.invoke("RAG가 뭔가요?")
print(result.content)
```

```
(py3_10_langchain) C:\DEV\SamsungElectronics_Gumi\src\AIAgent\chapter3>python langchain_chat_model.py  
gpt-5-mini selected
```

RAG는 "Retrieval-Augmented Generation"의 약자로, 정보 검색과 텍스트 생성을 결합한 자연어 처리(NLP) 모델을 나타냅니다. 이 접근 방식에서는 일반적으로 두 가지 주요 구성 요소가 사용됩니다:

1. 정보 검색기 (Retriever): 사용자의 질문이나 요청에 관련된 정보를 외부 데이터베이스나 문서 집합에서 검색합니다. 이 과정에서는 관련성이 높은 문서나 텍스트 조각을 선택합니다.
2. 텍스트 생성기 (Generator): 검색된 정보를 바탕으로 자연어로 응답을 생성합니다. 보통 이 과정에는 대형 언어 모델이 활용되며, 검색된 내용을 효과적으로 통합하여 사용자의 요청에 대한 적절한 답변을 생성합니다.

RAG의 장점은 모델이 단순히 사전 훈련된 지식을 사용하는 것을 넘어, 최신 정보를 검색하고 이를 기반으로 더 정확하고 유용한 응답을 생성할 수 있다는 점입니다. 이러한 방식은 질문 답변 시스템, 요약 생성, 대화형 AI 등 다양한 NLP 응용 분야에서 활용됩니다.

```
(py3_10_langchain) C:\DEV\SamsungElectronics_Gumi\src\AIAgent\chapter3>
```

PRD: 대화 이력 기반 멀티턴 질의응답

구분	내용
목적 (Objective)	LangChain의 ChatOpenAI 모델을 활용하여 **대화 이력(Context)**을 포함한 멀티턴 대화를 수행하고, 사용자 질문에 간결하고 명확한 답변을 생성한다.
기능 요구사항 (Functional Requirements)	<ul style="list-style-type: none"> ① ChatOpenAI(model="gpt-5-mini") 모델 초기화 ② 대화 이력(messages)에 SystemMessage, HumanMessage, AIMessage를 포함 ③ 시스템 지침: "간결하고 명확하게 답변하는 AI 도우미" ④ 사용자 질문: "주요 기능 세 가지만 알려주세요." ⑤ chat_model.invoke(messages) 호출하여 응답 생성 ⑥ 결과(result.content)를 콘솔에 출력
비기능 요구사항 (Non-Functional Requirements)	<ul style="list-style-type: none"> - Python + LangChain 환경에서 실행 가능해야 함 - OpenAI API Key 필요 - 멀티턴 대화 히스토리를 올바르게 처리해야 함
시나리오 (Scenarios)	<ul style="list-style-type: none"> - 시스템 메시지로 역할 정의 - 사용자 질문 "LangChain에 대해 설명해주세요." → AI가 설명 제공 - 후속 질문 "주요 기능 세 가지만 알려주세요." → AI가 이전 맥락을 반영해 3가지 주요 기능을 간결하게 반환
향후 개선 항목 (Future Improvements)	<ul style="list-style-type: none"> - 대화 이력 저장/불러오기 기능 추가 - 응답 형식(JSON, Bullet Point 등) 지정 기능 - 다국어 지원(예: 한국어/영어 병행) - 스트리밍 출력 지원으로 응답을 실시간 표시

● langchain_messages.py

```
from langchain_core.messages import SystemMessage, HumanMessage, AIMessage
import random
import os
from langchain_openai import ChatOpenAI
```

```
# 환경변수 설정 (실제로는 .env 사용 권장)
```

```
os.environ["OPENAI_API_KEY"] = "proj-DRnE5vys0joMbWlqJ4L9m07ta67EFYLcvYi0MSB-  
4EqCazZ38lIDHzVR0ijGLy8cZuew6yEVhkT3B1bkFJwk1CwB0m_wXbxQZ7ePoA3UnHHqwWJ6UktyaAisceZYm1abuk77cI27E5cunrmsndg  
30EELF4A"
```

실습 6: 메시지

```
# 모델 초기화
chat_model = ChatOpenAI(model="gpt-5-mini")

messages = [
    SystemMessage(
        content="당신은 사용자의 질문에 간결하고 명확하게 답변하는 AI도우미 입니다."
    ),
    HumanMessage(content="LangChain에 대해 설명해주세요."),
    AIMessage(
        content="LangChain은 대규모 언어 모델(LLM)을 활용하여 애플리케이션을 구축하기 위한 프레임워크입니다."
    ), # 이전 대화 예시
    HumanMessage(content="주요 기능 세 가지만 알려주세요."), # 사용자의 질문
]

result = chat_model.invoke(messages)
print("AI의 응답 : ", result.content)
```

(py3_10_langchain) C:\DEV\SamsungElectronics_Gumi\src\AIAgent\chapter3>python langchain_messages.py
AI의 응답 : 다음 세 가지가 LangChain의 핵심 기능입니다.

1. 체인(Chains) – 컴포저블 워크플로우
 - 여러 구성요소(프롬프트, LLM 호출, 후처리 등)를 연결해 복잡한 작업을 순차적으로 처리합니다.
 - 예: 입력 → 프롬프트 템플릿 → LLM → 출력 파서 → 최종 결과.
2. 에이전트(Agents) – 도구 사용·결정 엔진
 - 외부 도구(웹 검색, 계산기, 데이터베이스 등)를 필요에 따라 선택하고 호출하면서 작업을 수행하는 자율적 대화 흐름을 만듭니다.
 - 예: 질문에 답하기 위해 먼저 검색 도구로 관련 문서 찾고, 계산 도구로 수치 처리 후 응답 생성.
3. 검색 기반 생성(Retrieval / RAG) – 문서 로더·벡터 저장소·임베딩
 - 대용량 문서에서 관련 정보를 임베딩하고 벡터 검색으로 관련 문단을 찾아 LLM 입력에 포함해 정확도를 높입니다.
 - 예: 기업 문서 기반 질의응답(FAQ, 내부 문서 조회) 구현.

원하시면 각 기능별 간단한 코드 예시나 적용 사례도 보여드리겠습니다. 어느 부분을 더 보고 싶으신가요?

(py3_10_langchain) C:\DEV\SamsungElectronics_Gumi\src\AIAgent\chapter3>

실습 6: 메시지

- SystemMessage → “이 AI는 이렇게 행동해야 한다” (규칙/가이드라인)
- HumanMessage → “사용자가 묻는 말” (입력)
- AIMessage → “모델이 대답한 말” (출력)

구분	SystemMessage	HumanMessage	AIMessage
역할(Role)	시스템 지침 (system)	사용자 입력 (user)	모델 응답 (assistant)
주 목적	AI의 동작 방식, 톤, 제약 조건을 정의	사용자의 질문이나 요청을 전달	AI가 생성한 답변을 표현
대화 맥락	대화의 규칙/설정 제공 (예: “간결하고 명확하게 답하라”)	유저 발화로 간주	AI 발화로 간주
LangChain 매핑	OpenAI Chat API의 "system"	"user"	"assistant"
생성 방식	개발자가 미리 작성	사용자 입력 반영	모델이 생성 (또는 과거 답변 저장)
예시	"당신은 친절한 AI 비서입니다."	"LangChain에 대해 설명해줘."	"LangChain은 LLM 앱 개발 프레임워크입니다."

실습 6: 메시지 ... 생각해 보기

- 만약 chat과 message를 구분하지 않고 전부 텍스트만 합치면 “누가 말했는지” 알 수 없음 → 모델이 혼동
- Chat (대화)
 - 여러 발화(turn)들이 모여 있는 흐름 전체
 - 시간 순서와 맥락을 유지하는 구조
- Message (메시지)
 - 대화(chat)를 이루는 개별 발화
 - 누가(system / human / AI) 무엇을 말했는지 기록

PRD: LangChain PromptTemplate의 다양한 활용

구분	내용
목적 (Objective)	LangChain의 PromptTemplate 기능을 활용하여 다양한 방식으로 프롬프트를 정의하고, 입력 변수를 적용해 동적으로 문장을 생성하는 방법을 시연한다.
기능 요구사항 (Functional Requirements)	<ul style="list-style-type: none"> ① PromptTemplate.from_template를 사용하여 문자열 템플릿 생성 및 포매팅 ② 여러 입력 변수를 가진 PromptTemplate 생성 (article, style) ③ YAML 파일(template_example.yaml)에서 프롬프트 템플릿을 불러와 적용 ④ partial() 기능을 활용해 일부 변수를 고정한 템플릿 생성 ⑤ 각 템플릿에 입력 데이터를 적용하고 결과 문자열 출력
비기능 요구사항 (Non-Functional Requirements)	<ul style="list-style-type: none"> - Python + LangChain 환경에서 실행 가능해야 함 - 외부 YAML 파일은 UTF-8 인코딩으로 로드 가능해야 함 - 코드 실행 시 예시 출력이 콘솔에서 확인 가능해야 함
시나리오 (Scenarios)	<ul style="list-style-type: none"> - 사용자 입력 "랭체인이 뭐죠?" → "질문: 랭체인이 뭐죠? 답변:" 출력 - 기사 "OpenAI가 GPT-5를 공개했다..." + 스타일 "뉴스" → 뉴스 스타일 요약 프롬프트 출력 - YAML 템플릿 로드 후 context="서울은 한국의 수도이다.", question="수도는?" 적용 → 맞춤형 프롬프트 출력 - 번역 템플릿에 lang="Korean" 또는 lang="English" 고정 후 다른 텍스트 번역 요청
향후 개선 항목 (Future Improvements)	<ul style="list-style-type: none"> - Jinja2 등 고급 템플릿 엔진과 연동 - 다국어 프롬프트 템플릿 관리 기능 추가 - YAML 대신 JSON Schema 기반 템플릿 지원 - 프롬프트 버전 관리 및 재사용성을 높이는 모듈화

LangChain에서 프롬프트를 어떻게 선언·관리·재사용할 수 있는지 종합적으로 보여주는 학습용 데모

LangChain PromptTemplate의 다양한 활용법

1. 기본 치환 방식 확인
2. 다중 변수 템플릿 작성법 확인
3. YAML 외부 프롬프트 파일 불러오기 → 프롬프트 관리 확장성 확인
4. `partial()` 메서드로 변수 고정 → 재사용성/변형성 확인

template_example.yaml

```
_type: prompt
input_variables: ["context", "question"]
template: |
  컨텍스트: {context}
  질문: {question}
  답변:
```

실습 7: PromptTemplate 생성해 보기

prompt_template_example.py

```
import os
import yaml
from langchain.prompts import PromptTemplate

from langchain.prompts import load_prompt

template = PromptTemplate.from_template(
    "당신은 친절한 AI입니다.\n질문: {question}\n답변:"
)
print(template.format(question="랭체인이 뭐죠?"))

print("-----")

template = PromptTemplate(
    input_variables=["article", "style"],
    template="다음 기사를 {style} 스타일로 요약하세요:\n\n{article}",
)
print(template.format(article="OpenAI가 GPT-5를 공개했다...", style="뉴스"))
```

입력 변수가 여러 개인
경우(article, style) 처리
확인

실습 7: PromptTemplate 생성해 보기

```
print("-----")
current_dir_path = os.path.dirname(os.path.abspath(__file__))
with open(f"{current_dir_path}/template_example.yaml", "r", encoding="utf-8") as f:
    data = yaml.safe_load(f)

file_prompt = PromptTemplate(
    input_variables=data["input_variables"],
    template=data["template"]
)

print(file_prompt.format(context="서울은 한국의 수도이다.", question="수도는?"))

print("-----")
base_prompt = PromptTemplate.from_template("' {text}' 문장을 {lang}로 번역하세요.")
ko_prompt = base_prompt.partial(lang="Korean") # lang 고정
en_prompt = base_prompt.partial(lang="English") # 다른 버전

print(ko_prompt.format(text="Hello"))
print(en_prompt.format(text="안녕하세요"))
```

partial()을 활용해 일부
변수를 고정

실습 7: PromptTemplate 생성해 보기

```
(py3_10_langchain) C:\DEV\SamsungElectronics_Gumi\src\AIAgent\chapter3>python prompt_template_example.py
당신은 친절한 AI입니다.
```

질문: 랭체인이 뭐죠?

답변:

다음 기사를 뉴스 스타일로 요약하세요:

OpenAI가 GPT-5를 공개했다...

Traceback (most recent call last):

File "C:\DEV\SamsungElectronics_Gumi\src\AIAgent\chapter3\prompt_template_example.py", line 22, in
<module>

with open(f"{current_dir_path}/template_example.yaml", "r", encoding="utf-8") as f:
NameError: name 'current_dir_path' is not defined

```
(py3_10_langchain) C:\DEV\SamsungElectronics_Gumi\src\AIAgent\chapter3>python prompt_template_example.py
당신은 친절한 AI입니다.
```

질문: 랭체인이 뭐죠?

답변:

다음 기사를 뉴스 스타일로 요약하세요:

실습 7: PromptTemplate 생성해 보기

OpenAI가 GPT-5를 공개했다...

컨텍스트: 서울은 한국의 수도이다.

질문: 수도는?

답변:

'Hello' 문장을 Korean로 번역하세요.

'안녕하세요' 문장을 English로 번역하세요.

(py3_10_langchain) C:\DEV\SamsungElectronics_Gumi\src\AIAgent\chapter3>

실습 7: PromptTemplate 생성해 보기

좀 더 살펴보기 ...

```
import os
from langchain_openai import ChatOpenAI
from langchain.prompts import PromptTemplate, load_prompt

# ① 환경 변수 설정 (실제로는 .env 권장)
os.environ["OPENAI_API_KEY"] = "sk-..." # 실제 키로 교체하세요

# ② 첫 번째 예시: from_template 사용
template1 = PromptTemplate.from_template(
    "당신은 친절한 AI입니다.\n질문: {question}\n답변:"
)
formatted1 = template1.format(question="랭체인이 뭐죠?")
print("=== 첫 번째 프롬프트 ===")
print(formatted1)

# ③ 두 번째 예시: input_variables 지정
template2 = PromptTemplate(
    input_variables=["article", "style"],
    template="다음 기사를 {style} 스타일로 요약하세요:\n\n{article}",
)
formatted2 = template2.format(article="OpenAI가 GPT-5를 공개했다...", style="뉴스")
print("\n=== 두 번째 프롬프트 ===")
print(formatted2)
```

실습 7: PromptTemplate 생성해 보기

좀 더 살펴보기 ...

```
# ④ 세 번째 예시: YAML 프롬프트 파일 로드
# template_example.yaml이 같은 디렉토리에 있어야 함
try:
    file_prompt = load_prompt("template_example.yaml")
    formatted3 = file_prompt.format(
        context="서울은 한국의 수도이다.",
        question="수도는?",
    )
    print("\n=== 세 번째 프롬프트 (YAML 로드) ===")
    print(formatted3)
except Exception as e:
    print("\n[YAML 프롬프트 로드 실패] template_example.yaml을 확인하세요:", e)

# ⑤ OpenAI 모델 초기화
llm = ChatOpenAI(model="gpt-4o-mini", temperature=0.0)

# ⑥ 실제 모델 호출 (첫 번째 프롬프트 예시 사용)
print("\n=== 모델 응답 ===")
response = llm.invoke(formatted1)
print(response.content)
```

실습 7: PromptTemplate 생성해 보기

=== 첫 번째 프롬프트 ===

당신은 친절한 AI입니다.

질문: 랭체인이 뭐죠?

답변:

=== 두 번째 프롬프트 ===

다음 기사를 뉴스 스타일로 요약하세요:

OpenAI가 GPT-5를 공개했다...

=== 세 번째 프롬프트 (YAML 로드) ===

컨텍스트: 서울은 한국의 수도이다.

질문: 수도는?

답변:

=== 모델 응답 ===

LangChain은 대규모 언어 모델을 활용한 애플리케이션 개발 프레임워크입니다.

PRD: 프롬프트 → 모델 → 출력 파서 흐름

구분	내용
목적 (Objective)	LangChain의 ChatOpenAI와 ChatPromptTemplate, StrOutputParser를 활용하여 프롬프트 기반 대화 모델 실행 및 체인(LCEL) 구성 을 시연한다.
기능 요구사항 (Functional Requirements)	<ul style="list-style-type: none"> ① ChatOpenAI(model="gpt-5-mini")로 채팅 모델 초기화 ② ChatPromptTemplate를 정의하여 SystemMessage(역할 지시)와 HumanMessagePromptTemplate(질문 입력) 구성 ③ StrOutputParser로 모델의 응답을 문자열로 변환 ④ chat_model.invoke()를 사용해 포매팅된 메시지 입력 후 결과 생성 ⑤ string_output_parser.parse()로 결과를 문자열로 파싱 ⑥ `chat_prompt_template`
비기능 요구사항 (Non-Functional Requirements)	<ul style="list-style-type: none"> - Python + LangChain 환경에서 실행 가능해야 함 - OpenAI API Key 사전 설정 필요 - 출력은 최대 3줄 이내의 답변 (SystemMessage 지시사항 반영)
시나리오 (Scenarios)	<ul style="list-style-type: none"> - 입력 질문: "파이썬에서 리스트를 정렬하는 방법은?" → 까칠한 AI 도우미 역할로 최대 3줄 요약 답변 생성 - 입력 질문: "파이썬에서 딕셔너리를 정렬하는 방법은?" → LCEL 체인을 통해 동일한 구조로 답변 반환
향후 개선 항목 (Future Improvements)	<ul style="list-style-type: none"> - JsonOutputParser 등 다양한 출력 파서 적용 - 스트리밍 출력 기능 추가 - 체인에 후처리 모듈(예: 응답 요약, 안전성 필터링) 결합 - 멀티턴 대화 지원 확장

실습8: PromptTemplate와 OutputParsers

langchain_prompt_template_and_output_parser.py

```
from langchain_core.prompts import ChatPromptTemplate, HumanMessagePromptTemplate
from langchain_core.output_parsers import StrOutputParser
from langchain_core.messages import AIMessage, SystemMessage

import os
from langchain_openai import ChatOpenAI

# 환경변수 설정 (실제로는 .env 사용 권장)
os.environ["OPENAI_API_KEY"] = "proj-DRnE5vys0joMbWlqJ4L9m07ta67EFYLcvYi0MSB-
4EqCazZ38lIDHzVR0ijGLy8cZuew6yEVhkT3BlbkFJwk1CwB0m_wXbxQZ7ePoA3UnHHqwWJ6UktyaAisceZym1abuk77cI27E5cunrmsnndg
30EELF4A"

# ① 채팅 모델 초기화
chat_model = ChatOpenAI(model="gpt-5-mini")
# ② 프롬프트 템플릿 정의
chat_prompt_template = ChatPromptTemplate.from_messages(
    [
        SystemMessage(
            content="당신은 까칠한 AI 도우미입니다. 사용자의 질문에 최대 3줄로 답하세요."
        ),
        HumanMessagePromptTemplate.from_template("{question}"),
    ]
)
```


실습8: PromptTemplate와 OutputParsers

③ 출력 파서 정의

```
string_output_parser = StrOutputParser()
```

④ 프롬프트 템플릿을 사용하여 모델을 실행

```
result: AIMessage = chat_model.invoke(  
    chat_prompt_template.format_messages(  
        question="파이썬에서 리스트를 정렬하는 방법은?"  
    )  
)
```

⑤ 결과를 str 형식으로 변환

```
parsed_result: str = string_output_parser.parse(result)  
print(parsed_result.content)
```

```
print("-----")
```

⑥ 체인 생성 (LCEL)

```
chain = chat_prompt_template | chat_model | string_output_parser  
print(type(chain))
```

⑦ 체인 실행

```
result = chain.invoke({"question": "파이썬에서 딕셔너리를 정렬하는 방법은?"})
```

결과 출력

```
print(type(result))  
print(result)
```

:는 파이썬 타입 힌트 문법

런타임 동작에는 영향 없음 (pydantic, FastAPI, dataclasses 같은 최신 파이썬 프레임워크는 타입 힌트를 적극 활용)

여기서는 result가 LangChain의 AIMessage 또 문자열 타입 객체임을 코드 차원에서 명확히 하려는 의도

실습8: PromptTemplate와 OutputParsers

```
(py3_10_langchain) C:\DEV\SamsungElectronics_Gumi\src\AIAgent\chapter3>python
langchain_prompt_template_and_output_parser.py
list.sort() - 제자리 정렬(원본 변경), sorted(iterable) - 정렬된 새 리스트 반환.
옵션: reverse=True(내림차순), key=함수(정렬 기준) - 예: lst.sort(key=lambda x: x.age, reverse=True)
숫자/문자 기본 정렬, 복합키는 key가 튜플을 반환하도록 하면 된다.
```

```
-----
<class 'langchain_core.runnables.base RunnableSequence'>
```

```
<class 'str'>
```

딕셔너리는 항목(items)을 정렬한 뒤 새 dict(또는 OrderedDict)로 만들면 정렬된 순서를 유지합니다.

예: 키 기준: `sorted_by_key = dict(sorted(d.items()))`; 값 기준(내림차순): `sorted_by_value = dict(sorted(d.items(), key=lambda kv: kv[1], reverse=True))`

파이썬 3.7+는 dict가 삽입순을 유지하니 `dict(...)`로 충분하고, 구버전은 `collections.OrderedDict`를 쓰세요.

```
(py3_10_langchain) C:\DEV\SamsungElectronics_Gumi\src\AIAgent\chapter3>
```

PRD: 간단한 함수 → Runnable 변환 및 실행 예제

구분	내용
목적 (Objective)	LangChain의 RunnableLambda를 활용하여 간단한 Python 함수를 Runnable 형태로 감싸 실행 및 배치 처리를 지원하는 방법을 시연한다.
기능 요구사항 (Functional Requirements)	<ul style="list-style-type: none"> ① add_exclamation(text) 함수 정의 → 입력 텍스트 끝에 느낌표(!) 추가 ② RunnableLambda를 사용하여 함수를 Runnable 객체(exclamation_runnable)로 변환 ③ invoke() 메서드로 단일 입력 실행 → 결과 출력 ④ batch() 메서드로 여러 입력을 리스트로 전달 → 결과 리스트 출력
비기능 요구사항 (Non-Functional Requirements)	<ul style="list-style-type: none"> - Python + LangChain 환경에서 실행 가능해야 함 - 단일 입력과 다중 입력 모두 동일한 함수 로직으로 처리 - 출력은 항상 문자열 타입
시나리오 (Scenarios)	<ul style="list-style-type: none"> - 입력 "안녕하세요" → 출력 "안녕하세요!" - 입력 리스트 ["안녕", "반가워", "좋은 아침"] → 출력 ["안녕!", "반가워!", "좋은 아침!"]
향후 개선 항목 (Future Improvements)	<ul style="list-style-type: none"> - 다른 전처리/후처리 함수 적용 예제 추가 (예: 소문자 변환, 토큰화 등) - 여러 Runnable을 연결하여 체인 구성 - 비동기 실행(ainvoke, abatch) 테스트 추가 - 함수 실행 로깅 및 예외 처리 강화

실습9: Runnable 간단한 예제

langchain_runnable_lambda.py

```
from langchain_core.runnables import RunnableLambda

def add_exclamation(text: str) -> str:
    """텍스트 끝에 느낌표를 추가하는 함수"""
    return f"{text}!"

# RunnableLambda로 감싸서 Runnable로 만들기
exclamation_runnable = RunnableLambda(add_exclamation)

# 다양한 방식으로 실행 가능
result = exclamation_runnable.invoke("안녕하세요")
print(result)

# 배치 처리도 자동으로 지원
results = exclamation_runnable.batch(["안녕", "반가워", "좋은 아침"])
print(results)
```

보통 파이썬 함수는
add_exclamation("안녕하세요")처럼 그냥
호출해야 함
그런데 RunnableLambda로 감싸면 LangChain의
실행 체계에 맞춰:
.invoke() (단일 입력)
.batch() (여러 입력 동시에 처리)
.stream() (스트리밍 지원)
같은 함수도 LLM, 프롬프트, 체인과 똑같은
인터페이스로 실행 가능 → 조합성이 높아짐

RunnableLambda로 감싸면 단순한 파이썬 함수도
LLM 앞뒤에 붙여서 체인처럼 실행할 수 있음.
chain = prompt | llm | exclamation_runnable

실습9: Runnable 간단한 예제

```
(py3_10_langchain) C:\DEV\SamsungElectronics_Gumi\src\AIAgent\chapter3\runnable>python  
langchain_runnable_lambda.py
```

안녕하세요!

['안녕!', '반가워!', '좋은 아침!']

```
(py3_10_langchain) C:\DEV\SamsungElectronics_Gumi\src\AIAgent\chapter3\runnable>
```

PRD: LCEL 기반 프롬프트 체인으로 짧은 시 생성

구분	내용
목적 (Objective)	LangChain의 LCEL(LangChain Expression Language)을 활용하여 프롬프트 → 모델 → 출력 파서를 연결하고, 주어진 단어를 주제로 50자 이내의 짧은 시를 생성한다.
기능 요구사항 (Functional Requirements)	<ul style="list-style-type: none"> ① ChatPromptTemplate을 정의하여 {word}를 입력 변수로 받음 ② ChatOpenAI(model="gpt-5-mini")로 모델 초기화 ③ StrOutputParser()로 모델의 응답을 문자열로 변환 ④ `prompt`
비기능 요구사항 (Non-Functional Requirements)	<ul style="list-style-type: none"> - Python + LangChain 환경에서 실행 가능해야 함 - OpenAI API Key 사전 설정 필요 - 출력은 반드시 50자 이내의 짧은 시 형식이어야 함
시나리오 (Scenarios)	<ul style="list-style-type: none"> - 입력: "평범한 일상" → 프롬프트에 삽입 → 모델이 50자 이내의 짧은 시를 생성 → 최종 문자열 형태로 콘솔 출력
향후 개선 항목 (Future Improvements)	<ul style="list-style-type: none"> - 시의 형식을 다양화 (하이쿠, 삼행시 등) - 다국어 지원 (영어, 일본어 등) - 생성 길이를 동적으로 설정 가능하도록 개선 - 여러 단어를 입력받아 다중 시 생성 기능 추가

langchain_runnable_lecl.py

```
import os
from langchain_openai import ChatOpenAI

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

# 환경변수 설정 (실제로는 .env 사용 권장)
os.environ["OPENAI_API_KEY"] = "proj-DRnE5vys0joMbWlqJ4L9m07ta67EFYLcvYi0MSB-4EqCazZ38lIDHzVR0ijGLy8cZuew6yEVhkT3B1bkFJwk1CwB0m_wXbxQZ7ePoA3UnHHqwWJ6UktyaAisceZym1abuk77cI27E5cunrmsnndg30EELF4A"

prompt = ChatPromptTemplate.from_template(
    "주어지는 문구에 대하여 50자 이내의 짧은 시를 작성해주세요 : {word}"
)
# 모델 초기화
model = ChatOpenAI(model="gpt-5-mini")
parser = StrOutputParser()
```

실습10: LCEL –랭체인 표현 언어

① LCEL로 체인 구성

```
chain = prompt | model | parser
```

실행

```
result = chain.invoke({"word": "평범한 일상"})  
print(result)
```


실습10: LCEL –랭체인 표현 언어

```
(py3_10_langchain) C:\DEV\SamsungElectronics_Gumi\src\AIAgent\chapter3\runnable>python  
langchain_runnable_lecl.py  
평범한 하루, 그 속에 숨은 온기
```

PRD: 입력 원본 보존 + 처리 결과 병렬 출력

구분	내용
목적 (Objective)	LangChain의 RunnableParallel을 활용하여 입력 데이터를 원본과 처리 결과로 병렬 출력하는 체인을 구현한다.
기능 요구사항 (Functional Requirements)	<ul style="list-style-type: none"> ① ChatPromptTemplate 정의: 입력 단어({word})와 유사한 단어 3개 나열 지시 ② ChatOpenAI(model="gpt-5-mini")로 모델 초기화 ③ StrOutputParser로 모델 출력 문자열 변환 ④ RunnableParallel 구성 <ul style="list-style-type: none"> - "original": RunnablePassthrough()로 원본 데이터 유지 - "processed": prompt → model → parser 체인 적용 ⑤ chain.invoke({"word": "행복"}) 실행 후 결과 출력
비기능 요구사항 (Non-Functional Requirements)	<ul style="list-style-type: none"> - Python + LangChain 환경에서 실행 가능해야 함 - OpenAI API Key 필요 - 병렬 실행 결과는 딕셔너리 형태로 반환 ({ "original": ..., "processed": ... })
시나리오 (Scenarios)	<ul style="list-style-type: none"> - 입력: "행복" → "original" 필드에 그대로 보존 → "processed" 필드에 "기쁨, 즐거움, 만족" 같은 유사 단어 3개 반환
향후 개선 항목 (Future Improvements)	<ul style="list-style-type: none"> - 유사 단어 개수 파라미터화 (예: 5개, 10개) - 여러 입력 단어를 리스트로 받아 일괄 처리 - 결과를 JSON 포맷으로 구조화 - 병렬 출력에 추가 메타데이터(예: 실행 시간, 모델명) 포함

langchain_runnable_passthrough.py

```
import os
from langchain_openai import ChatOpenAI

from langchain_core.runnables import RunnablePassthrough, RunnableParallel
from langchain_core.prompts import ChatPromptTemplate
from langchain_openai import ChatOpenAI
from langchain_core.output_parsers import StrOutputParser

# 환경변수 설정 (실제로는 .env 사용 권장)
os.environ["OPENAI_API_KEY"] = "proj-DRnE5vys0joMbWlqJ4L9m07ta67EFYLcvYi0MSB-4EqCazZ38lIDHzVR0ijGLy8cZuew6yEVhkT3B1bkFJwk1CwB0m_wXbxQZ7ePoA3UnHHqwWJ6UktyaAisceZYm1abuk77cI27E5cunrmsnndg30EELF4A"

prompt = ChatPromptTemplate.from_template(
    "주어진 '{word}'와 유사한 단어 3가지를 나열해주세요. 단어만 나열합니다."
)
# 모델 초기화
model = ChatOpenAI(model="gpt-5-mini")
parser = StrOutputParser()
```

실습11: Runnable 주요 타입

① 병렬처리 체인 구성

```
chain = RunnableParallel(  
    {  
        "original": RunnablePassthrough(), # ② 원본 데이터 보존  
        "processed": prompt | model | parser, # ③ 처리된 데이터  
    }  
)  
  
result = chain.invoke({"word": "행복"})  
print(result)
```

RunnablePassthrough() = 입력값을 변형하지 않고 그대로 전달하는 Runnable
병렬 실행 시 원본 데이터를 보존하면서 동시에 다른 처리 결과와 함께 반환할 수 있음
디버깅, 비교, 로그 기록, 원본-가공 데이터 동시 저장 등에 유용

RunnablePassthrough()는 입력 데이터를 아무런 변환도 하지 않고 그대로 반환합니다.

즉, "original" 키의 값에는 입력으로 들어온 {"word": "행복"}이 그대로 들어갑니다.

실습11: Runnable 주요 타입

```
(py3_10_langchain) C:\DEV\SamsungElectronics_Gumi\src\AIAgent\chapter3\runnable>python  
langchain_runnable_passthrough.py  
{'original': {'word': '행복'}, 'processed': '기쁨\n즐거움\n만족'}
```

```
(py3_10_langchain) C:\DEV\SamsungElectronics_Gumi\src\AIAgent\chapter3\runnable>
```

PRD: 한 입력 단어에 대해 다양한 분석을 병렬적으로 수행

구분	내용
목적 (Objective)	LangChain의 RunnableParallel을 활용하여 하나의 입력 단어에 대해 **여러 가지 분석 작업(유사어 생성, 단어 길이 계산, 대문자 변환)**을 동시에 수행한다.
기능 요구사항 (Functional Requirements)	<ol style="list-style-type: none"> ① ChatPromptTemplate 정의: {word}와 유사한 단어 3개 나열 요청 ② ChatOpenAI(model="gpt-5-mini") 모델 초기화 ③ StrOutputParser로 모델 출력 문자열 변환 ④ RunnableParallel로 병렬 작업 구성 <ul style="list-style-type: none"> - synonyms: 프롬프트 → 모델 → 파서 연결, 유사 단어 3개 반환 - word_count: 입력 단어의 글자 수 계산 - uppercase: 입력 단어를 대문자로 변환 ⑤ analysis_chain.invoke({"word": "peaceful"}) 실행 후 결과 딕셔너리 출력
비기능 요구사항 (Non-Functional Requirements)	<ul style="list-style-type: none"> - Python + LangChain 환경에서 실행 가능해야 함 - OpenAI API Key 필요 - 결과는 딕셔너리 형태 ({"synonyms": ..., "word_count": ..., "uppercase": ...})로 반환
시나리오 (Scenarios)	<ul style="list-style-type: none"> - 입력: "peaceful" → "synonyms": "calm, tranquil, serene" → "word_count": 8 → "uppercase": "PEACEFUL"
향후 개선 항목 (Future Improvements)	<ul style="list-style-type: none"> - 유사어 개수를 가변적으로 설정 가능하게 개선 - 단어 난이도, 감정 점수 등 추가 분석 항목 확장 - 여러 단어 입력을 일괄 처리하는 기능 추가 - 결과를 JSON Schema 기반으로 구조화

langchain_runnable_parallel.py

```
import os
from langchain_openai import ChatOpenAI

from langchain_core.runnables import RunnableParallel, RunnableLambda
from langchain_core.prompts import ChatPromptTemplate
from langchain_openai import ChatOpenAI
from langchain_core.output_parsers import StrOutputParser

# 환경변수 설정 (실제로는 .env 사용 권장)
os.environ["OPENAI_API_KEY"] = "proj-DRnE5vys0joMbWlqJ4L9m07ta67EFYLcvYi0MSB-4EqCazZ38lIDHzVR0ijGLy8cZuew6yEVhkT3B1bkFJwk1CwB0m_wXbxQZ7ePoA3UnHHqwWJ6UktyaAisceZYm1abuk77cI27E5cunrmsndg30EELF4A"

prompt = ChatPromptTemplate.from_template(
    "주어진 '{word}'와 유사한 단어 3가지를 나열해주세요. 단어만 나열합니다."
)
# 모델 초기화
model = ChatOpenAI(model="gpt-5-mini")
parser = StrOutputParser()
```

실습11: Runnable 주요 타입

① 여러 분석을 동시에 수행

```
analysis_chain = RunnableParallel(  
    synonyms=prompt | model | parser, # ② 유사어 분석  
    word_count=RunnableLambda(lambda x: len(x["word"])), # ② 단어 수 계산  
    uppercase=RunnableLambda(lambda x: x["word"].upper()), # ② 대문자로 변환  
)
```

```
result = analysis_chain.invoke({"word": "peaceful"})  
print(result)
```

RunnableParallel

→ 여러 Runnable을 동시에 실행하고 결과를 딕셔너리로 묶어줌

synonyms

→ prompt | model | parser

→ 입력된 word를 프롬프트에 넣고, LLM으로 유사어 3개를 생성, 문자열로 파싱

word_count

→ RunnableLambda(lambda x: len(x["word"]))

→ 입력 딕셔너리 {"word": "행복"}에서 "word"의 길이를 계산

→ "행복"은 2글자니까 2 반환

uppercase

→ RunnableLambda(lambda x: x["word"].upper())

→ "행복" → "행복" (영어 단어라면 "happy" → "HAPPY")

실습11: Runnable 주요 타입

```
(py3_10_langchain) C:\DEV\SamsungElectronics_Gumi\src\AIAgent\chapter3\runnable>python  
langchain_runnable_parallel.py  
{'synonyms': 'tranquil\ncalm\nserene', 'word_count': 8, 'uppercase': 'PEACEFUL'}
```

```
(py3_10_langchain) C:\DEV\SamsungElectronics_Gumi\src\AIAgent\chapter3\runnable>
```

PRD: 조건부 분기 체인(RunnableBranch)으로 다국어 입력 처리

구분	내용
목적 (Objective)	입력된 단어가 영어인지 한국어인지 판별하여, 해당 언어에 맞는 프롬프트를 선택하고 유사 단어(동의어) 3개를 반환하는 체인을 구현한다.
기능 요구사항 (Functional Requirements)	<ul style="list-style-type: none"> ① is_english(x) 함수: 입력 값이 ASCII 범위인지 확인하여 영어 여부 판별 ② 영어 단어 입력 시: english_prompt(영문 프롬프트) → 모델 실행 → StrOutputParser로 결과 반환 ③ 한국어 단어 입력 시: korean_prompt(한글 프롬프트) → 모델 실행 → StrOutputParser로 결과 반환 ④ RunnableBranch를 사용하여 조건부 분기 실행 ⑤ 결과를 콘솔에 출력
비기능 요구사항 (Non-Functional Requirements)	<ul style="list-style-type: none"> - Python + LangChain 환경에서 실행 가능해야 함 - OpenAI API Key 필요 - 출력은 반드시 단어만 나열해야 함 (문장 금지)
시나리오 (Scenarios)	<ul style="list-style-type: none"> - 입력: "happy" → 영어로 판별 → "joyful, glad, cheerful" 같은 동의어 반환 - 입력: "행복" → 한국어로 판별 → "기쁨, 즐거움, 만족" 같은 동의어 반환
향후 개선 항목 (Future Improvements)	<ul style="list-style-type: none"> - 다국어 확장 (일본어, 중국어 등) → 언어별 맞춤 프롬프트 분기 추가 - 단어뿐 아니라 구/문장 입력도 지원 - 출력 포맷(JSON, 리스트 등) 구조화 - 언어 감지 로직을 단순 ASCII 검증에서 NLP 기반 감지 라이브러리로 개선

langchain_runnable_branch.py

```
import os
from langchain_openai import ChatOpenAI

from langchain_core.runnables import RunnableBranch
from langchain_core.prompts import ChatPromptTemplate
from langchain_core.output_parsers import StrOutputParser
from langchain_openai import ChatOpenAI

# 환경변수 설정 (실제로는 .env 사용 권장)
os.environ["OPENAI_API_KEY"] = "proj-DRnE5vys0joMbWlqJ4L9m07ta67EFYLcvYi0MSB-4EqCazZ38lIDHzVR0ijGLy8cZuew6yEVhkT3B1bkFJwk1CwB0m_wXbxQZ7ePoA3UnHHqwWJ6UktyaAisceZYm1abuk77cI27E5cunrmsnndg30EELF4A"

# 모델 초기화
model = ChatOpenAI(model="gpt-5-mini")
parser = StrOutputParser()
```

실습11: Runnable 주요 타입

```
# ① 입력된 텍스트가 영어인지 확인하는 함수
# 모든 문자가 ASCII 범위(128 미만)에 있으면 영어로 간주합니다.
def is_english(x: dict) -> bool:
    """입력 딕셔너리의 'word' 키에 해당하는 값이 영어인지 확인합니다."""
    return all(ord(char) < 128 for char in x["word"])

# ② 영어 단어에 대한 프롬프트 템플릿입니다.
english_prompt = ChatPromptTemplate.from_template(
    "Give me 3 synonyms for {word}. Only list the words."
)

# ③ 한국어 단어에 대한 프롬프트 템플릿입니다.
korean_prompt = ChatPromptTemplate.from_template(
    "주어진 '{word}'와 유사한 단어 3가지를 나열해주세요. 단어만 나열합니다."
)

# ④ 조건부 분기를 정의합니다.
# is_english 함수가 True를 반환하면 english_prompt를, 그렇지 않으면 korean_prompt를 사용합니다.
language_aware_chain = RunnableBranch(
    (is_english, english_prompt | model | parser), # 조건이 참일 때 실행될 체인
    korean_prompt | model | parser, # 기본값 (조건이 거짓일 때 실행될 체인)
)
```

LangChain의 Runnable 체계에서는 모든 처리가 연속된 체인(pipeline)으로 이어져야 합니다.

그런데 “조건에 따라 다른 체인을 실행”하려면 if-else 로직이 필요합니다.

이걸 RunnableBranch가 담당합니다.

입력이 영어라면 → english_prompt | model | parser 실행

입력이 한국어라면 → korean_prompt | model | parser 실행

실습11: Runnable 주요 타입

영어 단어 예시

```
english_word = {"word": "happy"}  
english_result = language_aware_chain.invoke(english_word)  
print(f"Synonyms for '{english_word['word']}':\n{english_result}\n")
```

한국어 단어 예시

```
korean_word = {"word": "행복"}  
korean_result = language_aware_chain.invoke(korean_word)  
print(f"'{korean_word['word']}'의 동의어:\n{korean_result}\n")
```

실습11: Runnable 주요 타입

```
(py3_10_langchain) C:\DEV\SamsungElectronics_Gumi\src\AIAgent\chapter3\runnable>python  
langchain_runnable_branch.py
```

```
Synonyms for 'happy':
```

```
joyful  
content  
elated
```

```
'행복'의 동의어:
```

```
기쁨  
즐거움  
환희
```

```
(py3_10_langchain) C:\DEV\SamsungElectronics_Gumi\src\AIAgent\chapter3\runnable>
```

PRD: LLM + Tool 바인딩을 활용한 인터랙티브 가위바위보 게임

구분	내용
목적 (Objective)	LangChain의 Tool 바인딩 기능을 활용하여 가위바위보 게임을 구현하고, AI가 랜덤 선택 및 해설을 제공하도록 한다.
기능 요구사항 (Functional Requirements)	<ul style="list-style-type: none"> ① @tool 데코레이터를 이용해 rps() 함수 정의 → 가위/바위/보 중 무작위 선택 ② ChatOpenAI(model="gpt-4o-mini") 모델 초기화 및 rps Tool 바인딩 ③ 사용자 입력(가위/바위/보)을 받아 LLM에 전달 → Tool 호출 확인 ④ Tool 실행 결과(ai_msg.tool_calls)에 따라 AI의 선택 결정 ⑤ judge(user_choice, computer_choice) 함수로 승부 판정 ⑥ 해설용 LLM(llm_for_chat)을 별도로 호출하여 결과를 재미있게 설명 ⑦ 최종적으로 승부, 해설, 요약을 콘솔에 출력
비기능 요구사항 (Non-Functional Requirements)	<ul style="list-style-type: none"> - Python + LangChain 환경에서 실행 가능해야 함 - OpenAI API Key 필요 - 사용자 입력 기반 인터랙티브 CLI 지원 - 안정적 랜덤성 확보 (random.choice)
시나리오 (Scenarios)	<ul style="list-style-type: none"> - 입력: "가위" → AI가 Tool을 호출하여 "바위" 선택 → 판정 결과: "패배" → 해설 LLM이 재미있는 코멘트 생성 → 콘솔 출력: 승부: 패배, LLM 해설, 게임 요약
향후 개선 항목 (Future Improvements)	<ul style="list-style-type: none"> - 다회전 경기(승점 집계) - 게임 UI 확장 (웹/GUI) - Tool 호출 실패 시 재시도 로직 추가 - 난수 대신 LLM이 직접 선택하도록 확장 - 다양한 해설 스타일 추가 (재치, 스포츠 캐스터 톤 등)

실습 12: @tool 데코레이터: 도구 생성의 표준 방법

langchain_rock_paper_scissors.py

```
import os
from langchain_openai import ChatOpenAI
```

```
import random
from langchain.tools import tool
```

```
# 환경변수 설정 (실제로는 .env 사용 권장)
```

```
os.environ["OPENAI_API_KEY"] = "proj-DRnE5vys0joMbWlqJ4L9m07ta67EFYLCvYi0MSB-4EqCazZ38lIDHzVR0ijGLy8cZuew6yEVhkT3B1bkFJwk1CwB0m_wXbxQZ7ePoA3UnHHqWJ6UktyaAisceZym1abuk77cI27E5cunrmsndg30EELF4A"
```

```
# ① 가위바위보 게임을 위한 Tool 정의
```

```
@tool
def rps() -> str:
    """가위바위보 중 하나를 랜덤하게 선택"""
    return random.choice(["가위", "바위", "보"])
```

rps() : 함수 이름 → "rock-paper-scissors"
(가위바위보)라는 의미
-> str : 타입 힌트 (return type annotation)
이 함수는 str(문자열)을 반환한다는 개발자/IDE용 힌트
예: "가위", "바위", "보" 중 하나

실습 12: @tool 데코레이터: 도구 생성의 표준 방법

```
# ② Tool 바인딩된 LLM (정상 모델명 + temperature 허용값)
llm = ChatOpenAI(model="gpt-4o-mini", temperature=0.0).bind_tools([rps])
llm_for_chat = ChatOpenAI(model="gpt-4o-mini", temperature=0.7) # 해설용 LLM
print(type(llm)) # LLM이 Tool을 바인딩했는지 확인
```

```
# ③ 승부 판정
def judge(user_choice, computer_choice):
    """가위바위보 승패를 판정합니다."""
    user_choice = user_choice.strip()
    computer_choice = computer_choice.strip()
    if user_choice == computer_choice:
        return "무승부"
    elif (user_choice, computer_choice) in [
        ("가위", "보"),
        ("바위", "가위"),
        ("보", "바위"),
    ]:
        return "승리"
    else:
        return "패배"
```

실습 12: @tool 데코레이터: 도구 생성의 표준 방법

```
# ④ 게임 루프
print("가위바위보! (종료: q)")
while (user_input := input("\n가위/바위/보: ")) != "q":
    # ⑤ LLM에게 tool 호출 요청
    ai_msg = llm.invoke(
        f"가위바위보 게임: 사용자가 {user_input}를 냈습니다. rps tool을 사용하세요."
    )

    # ⑥ Tool 호출 확인 및 실행
    if ai_msg.tool_calls:
        print(type(rps))
        llm_choice = rps.invoke("") # ⑦ Tool 호출 실행
        print(f"LLM이 선택한 도구: {llm_choice}")
        result = judge(user_input, llm_choice)

        print(f"승부: {result}") # 기존 print(f"{result}") 보다 명확하게

    # ⑧ 결과 응답 생성
    final = llm_for_chat.invoke(
        f"가위바위보 게임 결과를 재미있게 해설해주세요. "
        f"사용자: {user_input}, AI: {llm_choice}, 결과: 사용자의 {result}"
    )
    print(final)
    print(f"LLM 해설: {final.content}")
    print(f"게임 요약: 당신({user_input}) vs AI({llm_choice}) => {result}")
else:
    print("Tool 호출 실패")
```

실습 12: @tool 데코레이터: 도구 생성의 표준 방법

```
(py3_10_langchain) C:\DEV\SamsungElectronics_Gumi\src\AIAgent\chapter3\tools>python langchain_rock_paper_scissors.py
```

```
<class 'langchain_core.runnables.base RunnableBinding'>
```

가위바위보! (종료: q)

가위/바위/보: 바위

```
<class 'langchain_core.tools.structured.StructuredTool'>
```

LLM이 선택한 도구: 보

승부: 패배

content='자, 가위바위보의 세계로 들어가 보겠습니다! 오늘의 대결은 사용자가 "바위"를 선택하고 AI가 "보"를 선택한 상황입니다. \n\n사용자는 강력한 바위를 선택했지만, AI는 그 바위를 상대로 가장 부드럽고도 치명적인 "보"를 꺼내들었습니다. 아마 사용자는 바위의 강함에 자신감을 느끼며 "이길 수 있다!"고 외쳤겠죠.'

...
사용자는 강력한 바위를 선택했지만, AI는 그 바위를 상대로 가장 부드럽고도 치명적인 "보"를 꺼내들었습니다. 아마 사용자는 바위의 강함에 자신감을 느끼며 "이길 수 있다!"고 외쳤겠죠. 그러나 AI는 냉철한 판단력과 전략으로, 바위의 강한 외형 뒤에 숨겨진 약점을 정확히 파악했습니다.

결과적으로, AI의 보가 바위를 감싸며 "너는 내게 지지!"라고 외쳤고, 사용자는 아쉬운 표정으로 패배를 인정해야 했습니다. 바위는 힘이 세지만, 때로는 부드러움이 더 강력할 수 있다는 교훈을 남긴 이번 대결이었습니다.

다시 도전하고 싶으신가요? 다음 번엔 어떤 패를 선택하실 건가요?

게임 요약: 당신(바위) vs AI(보) => 패배

가위/바위/보: q

```
(py3_10_langchain) C:\DEV\SamsungElectronics_Gumi\src\AIAgent\chapter3\tools>
```

Tool을 언제 쓰면 좋을까?

LLM은 언어 모델이라 기본적으로 **텍스트 생성**밖에 못 합니다.
그런데 실제 애플리케이션에서는 **외부 기능**이 필요합니다.

Tool이 필요한 상황 예시

1. **랜덤/수학 계산** → 모델은 진짜 무작위 연산, 정밀 계산이 약함 → 툴로 보완
 2. **날씨/뉴스/주식 API 조회** → 최신 정보는 모델이 모르니까 API 호출 툴 필요
 3. **데이터베이스 질의** → 고객 정보, 재고 현황 등 사내 DB 접근
 4. **텍스트 후처리** → 정규식 치환, 번역기, 포맷 변환 등
- 즉, “LLM 혼자 잘 못하는 일 = 툴로 외부 기능을 연결”하는 전략입니다.

Tool 사용 전략

1. 작고 명확한 책임

1. 툴 하나는 딱 하나의 역할만 하게 (예: “오늘 날짜 반환”, “DB에서 고객 조회”)
2. 너무 많은 기능을 한 툴에 몰아넣지 말기

2. 입출력 타입을 단순하게

1. 문자열, 숫자, 리스트, dict처럼 직관적이고 직렬화 가능한 타입 사용
2. 복잡한 클래스 객체는 지양

3. LLM 친화적인 docstring 작성

1. “"""가위바위보 중 하나를 랜덤하게 선택""" 같은 주석은 모델이 툴을 올바르게 사용할 수 있게 돕습니다.
2. 사람이 아니라 모델이 읽는 설명이라는 점이 포인트

4. 자주 쓰이는 기능부터 툴화

1. 예: 사내 검색, 사전 조회, 단위 변환, API 호출 → LLM과 자주 연계되는 기능부터

PRD: 임베딩 + 코사인 유사도 기반 단어 의미 유사도 계산

구분	내용
목적 (Objective)	OpenAI 임베딩 모델을 활용하여 특정 단어 쿼리와 여러 단어 간의 **의미적 유사도(코사인 유사도)**를 계산한다.
기능 요구사항 (Functional Requirements)	<ul style="list-style-type: none"> ① OpenAIEmbeddings(model="text-embedding-3-large") 초기화 ② 입력 단어 리스트(["강아지", "고양이", "자동차", "비행기"])를 임베딩 벡터로 변환 ③ 쿼리 "동물"에 대한 임베딩 벡터 생성 ④ cosine_similarity(vec1, vec2) 함수 정의: 두 벡터의 코사인 유사도 계산 ⑤ 각 단어 임베딩과 쿼리 임베딩 간의 유사도 계산 후 출력
비기능 요구사항 (Non-Functional Requirements)	<ul style="list-style-type: none"> - Python + NumPy + LangChain 환경에서 실행 가능 - OpenAI API Key 필요 - 0으로 나누기 방지를 위해 작은 상수(1e-9) 추가
시나리오 (Scenarios)	<ul style="list-style-type: none"> - 쿼리: "동물" → "강아지": 높은 유사도 → "고양이": 높은 유사도 → "자동차", "비행기": 낮은 유사도 → 콘솔에 각 단어별 유사도 출력
향후 개선 항목 (Future Improvements)	<ul style="list-style-type: none"> - 시각화(예: 히트맵, 막대그래프) 추가 - 여러 쿼리에 대한 일괄 유사도 계산 지원 - Top-N 유사 단어만 반환하도록 확장 - 다국어 지원 및 문장 단위 임베딩 테스트

my_first_embedding.py

```
import os
from langchain_openai import ChatOpenAI
```

```
from langchain_openai import OpenAIEmbeddings
import numpy as np
```

```
# 환경변수 설정 (실제로는 .env 사용 권장)
```

```
os.environ["OPENAI_API_KEY"] = "proj-DRnE5vys0joMbWlqJ4L9m07ta67EFYLCvYi0MSB-4EqCazZ38lIDHzVR0ijGLy8cZuew6yEVhkT3B1bkFJwk1CwB0m_wXbxQZ7ePoA3UnHHqwWJ6UktyaAisceZYm1abuk77cI27E5cunrmsnndg30EELF4A"
```

```
# ① 임베딩 모델 초기화
```

```
embeddings = OpenAIEmbeddings(model="text-embedding-3-large")
```

```
# ② 단어들을 임베딩으로 변환
```

```
words = ["강아지", "고양이", "자동차", "비행기"]
```

```
word_embeddings = embeddings.embed_documents(words)
```

```
# ③ 쿼리 임베딩 생성
```

```
query = "동물"
```

```
query_embedding = embeddings.embed_query(query)
```

"동물"을 같은 3072차원 벡터로 변환.

결과는 query_embedding = [0.0125, -0.0021, ..., 0.0084] 같은 형태.

"강아지", "고양이", "자동차", "비행기" 각각을 3072차원(= text-embedding-3-large 출력 차원) 벡터로 변환.

word_embeddings는 리스트이며, 내부 요소는 대략 이런 구조:

```
[
  [0.0132, -0.0025, ..., 0.0078], # "강아지" 벡터
  [0.0118, -0.0019, ..., 0.0069], # "고양이" 벡터
  [0.0041, 0.0185, ..., -0.0097], # "자동차" 벡터
  [0.0059, 0.0203, ..., -0.0104], # "비행기" 벡터
]
```

④ 코사인 유사도 계산 함수

```
def cosine_similarity(vec1, vec2):
    """두 벡터 간의 코사인 유사도를 계산합니다."""
    dot_product = np.dot(vec1, vec2)
    norm_vec1 = np.linalg.norm(vec1)
    norm_vec2 = np.linalg.norm(vec2)
    return dot_product / (norm_vec1 * norm_vec2 + 1e-9) # 작은 값 추가로 0 나누기 방지
```

코사인 유사도 식은:

$$\cos(\theta) = \frac{\vec{v}_1 \cdot \vec{v}_2}{\|\vec{v}_1\| \cdot \|\vec{v}_2\|}$$

2차원 좌표에서 벡터 $\vec{v} = (x, y)$ 의 길이는:

$$\|\vec{v}\| = \sqrt{x^2 + y^2}$$

3차원 좌표에서 벡터 $\vec{v} = (x, y, z)$ 의 길이는:

$$\|\vec{v}\| = \sqrt{x^2 + y^2 + z^2}$$

n차원 벡터 일반화:

$$\|\vec{v}\| = \sqrt{\sum_{i=1}^n v_i^2}$$

(1) 내적(dot product)

- `np.dot(vec1, vec2)`
- 두 벡터의 방향이 얼마나 비슷한지 알려줍니다.
- 예:
 - 같은 방향 → 큰 양수
 - 직각 → 0
 - 반대 방향 → 큰 음수

(2) 벡터 크기(norm)

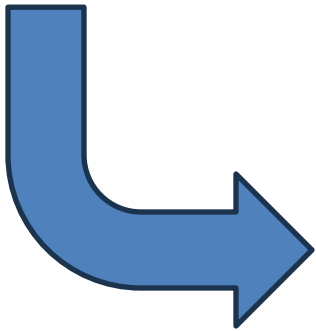
- `np.linalg.norm(vec)`
- 벡터의 길이(= 크기, magnitude)를 구합니다.
- 임베딩 벡터는 보통 수천 차원이므로 피타고라스 정리를 n차원으로 확장한 값입니다.

(3) 정규화

- `(dot_product) / (norm_vec1 * norm_vec2)`
- 내적 값을 두 벡터의 크기로 나눠서 0~1 범위의 "순수한 방향 유사도"만 추출합니다.
- 결과 해석:
 - 1.0 → 완전히 같은 방향 (의미가 동일)
 - 0.0 → 전혀 관계 없음 (직각)
 - -1.0 → 정반대 의미

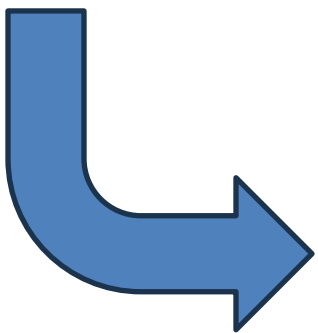
(4) 작은 값 1e-9 추가

- 분모 `(norm_vec1 * norm_vec2)` 가 0이 되는 상황 방지용 (안정성 확보)



실습 13: 임베딩

```
# ⑤ 각 단어와 쿼리의 유사도 계산
print(f"'{query}'에 대한 유사도:")
for word, embedding in zip(words, word_embeddings):
    similarity = cosine_similarity(query_embedding, embedding)
    print(f" {word}: {similarity:.3f}")
```



1. `zip(words, word_embeddings)` → "강아지"와 그 임베딩 벡터, "고양이"와 그 벡터, ... 묶음
2. 각 단어 벡터와 "동물"의 쿼리 벡터(`query_embedding`)를 `cosine_similarity`에 넣어 계산
3. 결과를 소수점 3자리까지 출력

결과 해석 예시

'동물'에 대한 유사도:

강아지: 0.912

고양이: 0.897

자동차: 0.213

비행기: 0.276

- "강아지" · "고양이"는 "동물"과 유사도가 높음 (둘 다 동물군에 속함)
- "자동차" · "비행기"는 "동물"과 유사도가 낮음 (교통수단)

```
(py3_10_langchain) C:\DEV\SamsungElectronics_Gumi\src\AIAgent\chapter3\tools>cd ..
```

```
(py3_10_langchain) C:\DEV\SamsungElectronics_Gumi\src\AIAgent\chapter3>cd embedding_vectorstore
```

```
(py3_10_langchain) C:\DEV\SamsungElectronics_Gumi\src\AIAgent\chapter3\embedding_vectorstore>pip install numpy==2.2.6  
langchain-community==0.3.27
```

```
Collecting numpy==2.2.6
```

```
Using cached numpy-2.2.6-cp310-cp310-win_amd64.whl.metadata (60 kB)
```

```
Collecting langchain-community==0.3.27
```

```
Downloading langchain_community-0.3.27-py3-none-any.whl.metadata (2.9 kB)
```

실습 13: 임베딩

```
(py3_10_langchain) C:\DEV\SamsungElectronics_Gumi\src\AIAgent\chapter3\embedding_vectorstore>python my_first_embedding.py
'동물'에 대한 유사도:
강아지: 0.508
고양이: 0.572
자동차: 0.388
비행기: 0.383

(py3_10_langchain) C:\DEV\SamsungElectronics_Gumi\src\AIAgent\chapter3\embedding_vectorstore>
```

PRD: 임베딩 + FAISS 기반 의미적 문서 검색

구분	내용
목적 (Objective)	OpenAI 임베딩 모델과 FAISS 벡터 스토어를 활용하여 텍스트 데이터에서 쿼리와 의미적으로 유사한 문장을 검색한다.
기능 요구사항 (Functional Requirements)	<ol style="list-style-type: none"> ① OpenAIEmbeddings(model="text-embedding-3-large")로 임베딩 모델 초기화 ② 샘플 텍스트 리스트를 임베딩 벡터로 변환 ③ 변환된 벡터를 FAISS 벡터 스토어에 저장 ④ 사용자 쿼리(예: "힘이 나는 명언 알려주세요.")를 입력받아 임베딩 생성 후 유사도 검색 수행 ⑤ 검색된 Top-K(예: 2개) 문서를 출력
비기능 요구사항 (Non-Functional Requirements)	<ul style="list-style-type: none"> - Python + LangChain + FAISS 환경에서 실행 가능 - OpenAI API Key 필요 - 빠른 벡터 검색 성능 보장
시나리오 (Scenarios)	<ul style="list-style-type: none"> - 입력: "힘이 나는 명언 알려주세요." → FAISS에서 임베딩 기반 유사도 검색 → 예: "성공은 실패를 거듭하는 것이 아니라...", "인생은 10%의 사건과 90%의 반응..." 같은 문장 반환
향후 개선 항목 (Future Improvements)	<ul style="list-style-type: none"> - 검색 결과 개수(K) 동적 조절 기능 추가 - 검색 결과와 함께 유사도 점수 반환 - 텍스트 말고 PDF/문서 데이터로 확장 - 검색 결과 요약 기능 추가

embedding_with_vectorstore.py

```
import os
from langchain_openai import ChatOpenAI

from langchain_openai import OpenAIEmbeddings
from langchain_community.vectorstores import FAISS

# 환경변수 설정 (실제로는 .env 사용 권장)
os.environ["OPENAI_API_KEY"] = "proj-DRnE5vys0joMbWlqJ4L9m07ta67EFYLcvYi0MSB-
4EqCazZ38lIDHzVR0ijGLy8cZuew6yEVhkT3B1bkFJwk1CwB0m_wXbxQZ7ePoA3UnHHqwWJ6UktyaAisceZym1abuk77cI27E5cunrmsnndg
30EELF4A"

# 임베딩 모델 초기화
embeddings = OpenAIEmbeddings(model="text-embedding-3-large")
```

실습 14: 벡터 스토어

① 샘플 텍스트 데이터

```
texts = [  
    "파이썬은 배우기 쉬운 프로그래밍 언어입니다.",  
    "자바스크립트는 웹 개발에 널리 사용됩니다.",  
    "꽁꽁 얼어붙은 한강위로 고양이가 걸어갑니다.",  
    "어리석은 자는 멀리서 행복을 찾고, 현명한 자는 자신의 발치에서 행복을 키워간다.",  
    "계단을 밟아야 계단 위에 올라설수 있다",  
    "인생은 10%의 사건과 90%의 반응으로 이루어져 있습니다.",  
    "성공은 실패를 거듭하는 것이 아니라, 실패를 거듭하면서도 열정을 잃지 않는 것입니다.",  
    "하루에 3시간을 걸으면 7년 후에 지구를 한바퀴 돌 수 있습니다.",  
    "인공지능은 머신러닝과 딥러닝을 통해 발전하고 있습니다.",  
]
```

② 텍스트를 벡터로 변환하고 FAISS 벡터 스토어에 저장

```
vectorstore = FAISS.from_texts(texts, embeddings)
```

③ 유사한 문서 검색

```
query = "힘이 나는 명언 알려주세요."
```

```
docs = vectorstore.similarity_search(query, k=2)
```

④ 검색 결과 출력

```
print("검색 결과:")
```

```
for i, doc in enumerate(docs):
```

```
    print(f"{i+1}. {doc.page_content}")
```

faiss-cpu 설치

```
py3_10_langchain) c:\DEV\envs\py3_10_langchain>pip install faiss-cpu
Collecting faiss-cpu
  Downloading faiss_cpu-1.12.0-cp310-cp310-win_amd64.whl.metadata (5.2 kB)
Requirement already satisfied: numpy<3.0,>=1.25.0 in c:\dev\envs\py3_10_langchain\lib\site-packages (from faiss-cpu) (2.2.6)
Requirement already satisfied: packaging in c:\dev\envs\py3_10_langchain\lib\site-packages (from faiss-cpu) (24.2)
Downloading faiss_cpu-1.12.0-cp310-cp310-win_amd64.whl (18.2 MB)
----- 18.2/18.2 MB 32.7 MB/s eta 0:00:00
Installing collected packages: faiss-cpu
Successfully installed faiss-cpu-1.12.0

[notice] A new release of pip is available: 25.1.1 -> 25.2
[notice] To update, run: python.exe -m pip install --upgrade pip

(py3_10_langchain) c:\DEV\envs\py3_10_langchain>
```

실습 14: 벡터 스토어

```
(py3_10_langchain) C:\DEV\SamsungElectronics_Gumi\src\AIAgent\chapter3\embedding_vectorstore>python embedding_with_vectorstore.py
```

검색 결과:

1. 인생은 10%의 사건과 90%의 반응으로 이루어져 있습니다.
2. 어리석은 자는 멀리서 행복을 찾고, 현명한 자는 자신의 발치에서 행복을 키워간다.

```
(py3_10_langchain) C:\DEV\SamsungElectronics_Gumi\src\AIAgent\chapter3\embedding_vectorstore>
```


PRD: 문서 분할 + 벡터 검색 + LLM 응답

구분	내용
목적 (Objective)	OpenAI 임베딩과 FAISS 기반 리트리버를 활용하여 문서를 검색하고, 검색된 결과를 바탕으로 LLM이 질문에 답변하는 간단한 RAG(Retrieval-Augmented Generation) 파이프라인 을 구현한다.
기능 요구사항 (Functional Requirements)	<ul style="list-style-type: none"> ① OpenAIEmbeddings와 CharacterTextSplitter를 사용해 문서를 분할 및 벡터화 ② FAISS.from_documents로 벡터 스토어 생성 후 retriever 구성 ③ 사용자 질문 "초보자가 배우기 좋은 프로그래밍 언어는?"을 입력받아 관련 문서 검색 ④ 검색된 문서의 page_content, metadata(출처, 주제) 출력 ⑤ ChatPromptTemplate를 정의해 질문과 검색 문맥(context)을 LLM에 전달 ⑥ `chain = {"context": retriever, "question": RunnablePassthrough()}`
비기능 요구사항 (Non-Functional Requirements)	<ul style="list-style-type: none"> - Python + LangChain 환경에서 실행 가능 - OpenAI API Key 필요 - 검색 결과와 답변이 일관되고 맥락을 반영해야 함
시나리오 (Scenarios)	<ul style="list-style-type: none"> - 입력: "초보자가 배우기 좋은 프로그래밍 언어는?" → 리트리버가 "파이썬은 읽기 쉽고 배우기 쉬운 프로그래밍 언어..." 문서 검색 → LLM이 문서 내용을 참고하여 "초보자에게 적합한 언어는 파이썬..."과 같은 답변 생성
향후 개선 항목 (Future Improvements)	<ul style="list-style-type: none"> - 검색 결과 개수를 동적으로 설정 (Top-K 조정) - 결과에 유사도 점수 포함 - 복수 문서 기반 요약 응답 기능 추가 - 메타데이터(출처/주제)도 함께 LLM 답변에 포함

retriever_from_vectostore.py

```
import os
from langchain_openai import ChatOpenAI

from langchain_openai import OpenAIEmbeddings
from langchain_community.vectorstores import FAISS
from langchain.schema import Document
from langchain.text_splitter import CharacterTextSplitter
from langchain_core.prompts import ChatPromptTemplate
from langchain_core.runnables import RunnablePassthrough

# 환경변수 설정 (실제로는 .env 사용 권장)
os.environ["OPENAI_API_KEY"] = "proj-DRnE5vys0joMbWlqJ4L9m07ta67EFYLcvYi0MSB-
4EqCazZ38lIDHzVR0ijGLy8cZuew6yEVhkT3B1bkFJwk1CwB0m_wXbxQZ7ePoA3UnHHqwWJ6UktyaAisceZYm1abuk77cI27E5cunrmsndg
30EELF4A"

# 임베딩 모델과 텍스트 분할기 준비
embeddings = OpenAIEmbeddings()
text_splitter = CharacterTextSplitter(separator=".", chunk_size=50, chunk_overlap=20)
```

실습 15: 리트리버

샘플 문서들 준비

```
documents = [  
    Document(  
        page_content="파이썬은 읽기 쉽고 배우기 쉬운 프로그래밍 언어입니다. "  
        "다양한 분야에서 활용되며, 특히 데이터 과학과 AI 개발에 인기가 높습니다.",  
        metadata={"source": "python_intro.txt", "topic": "programming"},  
    ),  
    Document(  
        page_content="자바스크립트는 웹 브라우저에서 실행되는 프로그래밍 언어로 시작했지만, "  
        "현재는 서버 사이드 개발에도 널리 사용됩니다. Node.js가 대표적입니다.",  
        metadata={"source": "js_guide.txt", "topic": "programming"},  
    ),  
    Document(  
        page_content="머신러닝은 데이터에서 패턴을 학습하는 AI의 한 분야입니다. "  
        "지도학습, 비지도학습, 강화학습 등 다양한 방법론이 있습니다.",  
        metadata={"source": "ml_basics.txt", "topic": "ai"},  
    ),  
]
```

문서 분할

```
split_docs = text_splitter.split_documents(documents)
```

분할 기준은?

어떤 TextSplitter를 쓰느냐에 따라 다릅니다.

- CharacterTextSplitter: 기본적으로 문자 단위(예: 1000자 단위)로 분할
- RecursiveCharacterTextSplitter: 문단, 문장, 단어 단위로 순차적으로 쪼개되, 길이가 맞지 않으면 더 작은 단위로 세분화
- TokenTextSplitter: 토큰 단위(BPE 같은 토큰화 알고리즘 기반)
- SentenceTransformersTokenTextSplitter 등 다양한 방식이 존재

앞서, 다음 코드의 경우

```
text_splitter = CharacterTextSplitter(separator=".", chunk_size=50, chunk_overlap=20)
```

- separator="."

⇒ 우선적으로 마침표(.)를 기준으로 쪼개려고 시도합니다.

다만, 반드시 .에서만 자르는 건 아니고, chunk_size를 만족시키기 위해 필요한 경우 중간에서 자를 수도 있음.

- chunk_size=50

⇒ 하나의 청크가 최대 50자까지 들어갈 수 있습니다.

- chunk_overlap=20

⇒ 이전 청크와 20자 정도가 겹치게 생성됩니다. (문맥을 유지하기 위함)

실습 15: 리트리버

① 리트리버 생성

```
retriever = FAISS.from_documents(split_docs, embeddings).as_retriever(
    search_type="similarity",
    search_kwargs={"k": 1},
)
```

문장을 벡터화한 뒤, 질문과 가장 유사한 문서를 가져오는 검색기"를 만든 단계

② 리트리버를 사용한 검색

```
results = retriever.get_relevant_documents("초보자가 배우기 좋은 프로그래밍 언어는?")
for i, doc in enumerate(results, 1):
    print(
        f"{i}. {doc.page_content[:30]}... | 출처: {doc.metadata['source']} | 주제: {doc.metadata['topic']}"
    )
```

리트리버가 찾아낸 문서 내용을 확인하는 단계
doc.metadata['source']: 문서 출처
doc.metadata['topic']: 문서 주제

③ 리트리버를 사용한 LLM 호출

```
llm = ChatOpenAI(model_name="gpt-5-mini")
message = ""
```

질문에 대한 답변을 작성할 때, 리트리버에서 가져온 문서를 참고하여 답변을 작성하세요.

질문:

{question}

참고:

{context}

"""

message 프롬프트

사람이 묻는 질문과 리트리버가 가져온 context (문서 내용)을 넣어서,
LLM이 문서 기반 답변(RAG: Retrieval-Augmented Generation)을 하도록 지시

실습 15: 리트리버

```
prompt = ChatPromptTemplate.from_messages([("human", message)])

chain = {"context": retriever, "question": RunnablePassthrough()} | prompt | llm

response = chain.invoke("초보자가 배우기 좋은 프로그래밍 언어는?")
print("\nLLM 응답:")
print(response.content)
```

실습 15: 리트리버

```
(py3_10_langchain) C:\DEV\SamsungElectronics_Gumi\src\AIAgent\chapter3\embedding_vectorstore>cd ..
```

```
(py3_10_langchain) C:\DEV\SamsungElectronics_Gumi\src\AIAgent\chapter3>cd retriever_rag
```

```
(py3_10_langchain) C:\DEV\SamsungElectronics_Gumi\src\AIAgent\chapter3\retriever_rag>
```

실습 15: 리트리버

```
(py3_10_langchain) C:\DEV\SamsungElectronics_Gumi\src\AIAgent\chapter3\retriever_rag>python
retriever_from_vectostore.py
Created a chunk of size 63, which is longer than the specified 50
C:\DEV\SamsungElectronics_Gumi\src\AIAgent\chapter3\retriever_rag\retriever_from_vectostore.py:49:
LangChainDeprecationWarning: The method `BaseRetriever.get_relevant_documents` was deprecated in langchain-
core 0.1.46 and will be removed in 1.0. Use :meth:`~invoke` instead.
  results = retriever.get_relevant_documents("초보자가 배우기 좋은 프로그래밍 언어는?")
1. 파이썬은 읽기 쉽고 배우기 쉬운 프로그래밍 언어입니다... | 출처: python_intro.txt | 주제: programming
```

LLM 응답:

참고 문서에 따르면 "파이썬은 읽기 쉽고 배우기 쉬운 프로그래밍 언어입니다." 이를 바탕으로 초보자에게 좋은 언어와 선택 방법을 정리하면 다음과 같습니다.

...

시작 권장 예(파이썬 기준)

- 설치 후 간단한 프로그램 작성: "안녕하세요" 출력, 숫자 계산기, 파일 읽기/쓰기
- 작은 프로젝트: 할 일 목록 앱, 웹 스크래퍼, 데이터 시각화(판다스 · Matplotlib)

요약: 참고 문서대로 파이썬이 초보자에게 매우 좋은 선택입니다. 다만 최종 선택은 당신의 목표(웹/데이터/앱 등)에 따라 달라지므로, 목표를 정한 뒤 그에 맞는 언어로 시작하세요. 필요하면 목표를 알려주시면 추천 언어와 학습 로드맵을 구체적으로 만들어 드리겠습니다.

```
(py3_10_langchain) C:\DEV\SamsungElectronics_Gumi\src\AIAgent\chapter3\retriever_rag>
```


PRD: 실시간 검색 + LLM 응답 결합 웹 기반 RAG 파이프라인

구분	내용
목적 (Objective)	실시간 웹 검색(DuckDuckGo)을 활용하여 최신 정보를 가져오고, 검색 결과를 기반으로 LLM이 답변을 생성하는 실시간 RAG(Retrieval-Augmented Generation) 시스템 을 구현한다.
기능 요구사항 (Functional Requirements)	<ol style="list-style-type: none"> ① DuckDuckGoSearchResults를 통해 실시간 검색 기능 제공 ② ChatOpenAI(temperature=0) 모델을 사용하여 검색 결과 기반 답변 생성 ③ ChatPromptTemplate 정의 → {search_results}, {question} 포함 ④ answer(question) 메서드에서: <ul style="list-style-type: none"> - 실시간 검색 실행 및 결과 수집 - 10초 대기하여 rate limit 방지 - LLM 체인(qa_chain)을 실행하여 답변 생성 ⑤ 사용 예시: "오늘 주요 뉴스는?", "오늘 야구 순위는?", "최신 AI 기술 동향은?" 등 질문에 대해 검색 후 답변 출력
비기능 요구사항 (Non-Functional Requirements)	<ul style="list-style-type: none"> - Python + LangChain 환경에서 실행 가능 - 인터넷 연결 필요 (DuckDuckGo 검색) - 출력은 반드시 검색 결과에 근거해야 함
시나리오 (Scenarios)	<ul style="list-style-type: none"> - 입력: "오늘 주요 뉴스는?" → DuckDuckGo에서 뉴스 검색 → 검색 결과를 기반으로 LLM이 요약 답변 반환
향후 개선 항목 (Future Improvements)	<ul style="list-style-type: none"> - 검색 결과 필터링 및 랭킹 개선 - 검색 결과 요약 후 답변에 포함 - 대기 시간(rate limit 방지)을 동적으로 조정 - 다른 검색 API(Google, Bing 등) 연동 옵션 추가

rag_by_duckduckgo.py

```
import os
from langchain_openai import ChatOpenAI

from langchain_community.tools import DuckDuckGoSearchResults
from langchain_openai import ChatOpenAI
from langchain_core.prompts import ChatPromptTemplate
import time

# 환경변수 설정 (실제로는 .env 사용 권장)
os.environ["OPENAI_API_KEY"] = "proj-DRnE5vys0joMbWlqJ4L9m07ta67EFYLCvYi0MSB-
4EqCazZ38lIDHzVR0ijGLy8cZuew6yEVhkT3B1bkFJwk1CwB0m_wXbxQZ7ePoA3UnHHqWJ6UktyaAisceZYM1abuk77cI27E5cunrmsnndg
30EELF4A"

# ① RealtimeWebRAG: 실시간 웹 검색을 활용한 RAG (Retrieval-Augmented Generation)
class RealtimeWebRAG:
    """실시간 웹 검색을 활용하는 RAG"""

    def __init__(self):
        self.search = DuckDuckGoSearchResults()
        self.llm = ChatOpenAI(temperature=0)

    message = """웹에서 검색한 최신 정보를 바탕으로 답변하세요.
```

DuckDuckGoSearchResults() Python 코드에서 바로 웹 검색 결과를 리스트 형태로 받을 수 있게 함

실습 16: RAG

검색 결과:

{search_results}

질문: {question}

중요: 검색 결과에 있는 정보만 사용하여 답변하세요.

답변: ""

{search_results}에는 검색 결과가, {question}에는 사용자의 질문이 들어감.

"검색 결과에 있는 정보만 사용하라"고 명시 → 환각(Hallucination)을 줄이기 위함.

```
self.qa_prompt = ChatPromptTemplate.from_messages(  
    [  
        (  
            "human",  
            message,  
        ),  
    ],  
)
```

② 답변 생성 메서드

```
def answer(self, question):  
    """실시간 검색 후 답변 생성"""  
    # 1. 웹 검색  
    print(f"검색 중: {question}")  
    search_results = self.search.run(question)  
    time.sleep(10) # 10초 대기로 rate limit 방지  
  
    # 2. LLM으로 답변 생성  
    qa_chain = self.qa_prompt | self.llm  
    answer = qa_chain.invoke(  
        {"search_results": search_results, "question": question}  
    )  
  
    return answer
```

사용자의 질문(question)을 받아서 DuckDuckGo 검색 실행.
결과를 search_results에 저장.
API 요청 제한(rate limit) 회피를 위해
time.sleep(10)으로 10초 대기.

③ 사용 예시

```
web_rag = RealtimeWebRAG()
```

최신 정보 질문

```
questions = ["오늘 주요 뉴스는?", "오늘 야구 순위는?", "최신 AI 기술 동향은?"]
```

```
for q in questions:
```

```
    print(f"\n질문: {q}")
```

```
    answer = web_rag.answer(q)
```

```
    print(f"답변: {answer.content}\n")
```

```
(py3_10_langchain) c:\DEV\envs\py3_10_langchain>pip install -U duckduckgo-search
```

```
Collecting duckduckgo-search
```

```
  Downloading duckduckgo_search-8.1.1-py3-none-any.whl.metadata (16 kB)
```

```
Collecting click>=8.1.8 (from duckduckgo-search)
```

```
  Downloading click-8.2.1-py3-none-any.whl.metadata (2.5 kB)
```

```
Collecting primp>=0.15.0 (from duckduckgo-search)
```

```
  Downloading primp-0.15.0-cp38-abi3-win_amd64.whl.metadata (13 kB)
```

```
Collecting lxml>=5.3.0 (from duckduckgo-search)
```

```
  Downloading lxml-6.0.1-cp310-cp310-win_amd64.whl.metadata (3.9 kB)
```

```
Requirement already satisfied: colorama in c:\dev\envs\py3_10_langchain\lib\site-packages (from click>=8.1.8->duckduckgo-search) (0.4.6)
```

```
Downloading duckduckgo_search-8.1.1-py3-none-any.whl (18 kB)
```

```
Downloading click-8.2.1-py3-none-any.whl (102 kB)
```

```
Downloading lxml-6.0.1-cp310-cp310-win_amd64.whl (4.0 MB)
```

```
----- 4.0/4.0 MB 40.5 MB/s eta 0:00:00
```

```
Downloading primp-0.15.0-cp38-abi3-win_amd64.whl (3.1 MB)
```

```
----- 3.1/3.1 MB 23.1 MB/s eta 0:00:00
```

```
Installing collected packages: primp, lxml, click, duckduckgo-search
```

```
Successfully installed click-8.2.1 duckduckgo-search-8.1.1 lxml-6.0.1 primp-0.15.0
```

```
[notice] A new release of pip is available: 25.1.1 -> 25.2
```

```
[notice] To update, run: python.exe -m pip install --upgrade pip
```

```
(py3_10_langchain) c:\DEV\envs\py3_10_langchain>
```

```
(py3_10_langchain) C:\DEV\SamsungElectronics_Gumi\src\AIAgent\chapter3\retriever_rag>python rag_by_duckduckgo.py
```

질문: 오늘 주요 뉴스는?

검색 중: 오늘 주요 뉴스는?

```
C:\DEV\envs\py3_10_langchain\lib\site-packages\langchain_community\utilities\duckduckgo_search.py:63: RuntimeWarning: This package (`duckduckgo_search`) has been renamed to `ddgs`! Use `pip install ddgs` instead.
```

```
    with DDGS() as ddgs:
```

답변: 오늘 주요 뉴스는 KBS 뉴스에서 정부의 민감국가 대응 논의를 다루고 있으며, 연합뉴스에서는 대통령이 강선우 여성가족부 장관 후보자에 대한 인사청문 경과보고서 재송부를 요청하며 임명 수순을 밟을 예정이라고 합니다.

...

질문: 최신 AI 기술 동향은?

검색 중: 최신 AI 기술 동향은?

```
C:\DEV\envs\py3_10_langchain\lib\site-packages\langchain_community\utilities\duckduckgo_search.py:63: RuntimeWarning: This package (`duckduckgo_search`) has been renamed to `ddgs`! Use `pip install ddgs` instead.
```

```
    with DDGS() as ddgs:
```

답변: 2025년 최신 AI 기술 동향은 생성형 AI, 멀티모달 AI, 엣지 AI, 윤리적 AI 등이 주요한 트렌드로 나타나고 있습니다. AI 기술은 산업, 비즈니스, 일상생활에 큰 영향을 미치며 새로운 패러다임을 만들어가고 있습니다. 최근 AI 기술은 단순한 자동화에서 더욱 발전된 형태로 발전하고 있습니다.

```
(py3_10_langchain) C:\DEV\SamsungElectronics_Gumi\src\AIAgent\chapter3\retriever_rag>
```

PRD: 동기(Sync) 방식 단일 실행, 비동기(Async) 방식

구분	코드 1 (동기 실행)	코드 2 (비동기 실행)
목적 (Objective)	Agent를 동기 방식으로 실행하여 간단한 인사 응답을 테스트한다.	Agent를 비동기(async) 방식으로 실행하여 간단한 인사 응답을 테스트한다.
기능 요구사항 (Functional Requirements)	<ul style="list-style-type: none"> ① Agent 생성: 이름 "HelloAgent", 지침 "안녕하세요"라고 인사하는 임무 ② Runner.run_sync()로 사용자 입력 실행 ③ 결과(result.final_output)를 콘솔에 출력 	<ul style="list-style-type: none"> ① Agent 생성: 이름 "HelloAgent", 지침 "안녕하세요"라고 인사하는 임무 ② Runner.run()으로 사용자 입력 실행 (await 필요) ③ 결과(result.final_output)를 콘솔에 출력 ④ asyncio.run(main())으로 비동기 메인 실행
비기능 요구사항 (Non-Functional Requirements)	<ul style="list-style-type: none"> - Python 환경에서 동기 실행 지원 - 단일 요청 처리 중심 	<ul style="list-style-type: none"> - Python 환경에서 비동기 실행 지원 - 동시성 활용 가능
시나리오 (Scenarios)	<ul style="list-style-type: none"> - 입력: "프랑스어로만 인사해주세요." → Agent가 "Bonjour" 등의 출력 반환 	<ul style="list-style-type: none"> - 입력: "일본어로 인사해주세요." → Agent가 "こんにちは" 등의 출력 반환
향후 개선 항목 (Future Improvements)	<ul style="list-style-type: none"> - 추가 입력 처리 (다중 요청) - 에러 처리 로직 추가 	<ul style="list-style-type: none"> - 다중 비동기 작업 동시 실행 지원 - 스트리밍 응답 처리 확장 - 예외 처리 및 타임아웃 제어

hello_agent_sync.py

```
import os
from langchain_openai import ChatOpenAI

from agents import Agent, Runner

# 환경변수 설정 (실제로는 .env 사용 권장)
os.environ["OPENAI_API_KEY"] = "proj-DRnE5vys0joMbWlqJ4L9m07ta67EFYLcvYi0MSB-
4EqCazZ38lIDHzVR0ijGLy8cZuew6yEVhkT3B1bkFJwk1CwB0m_wXbxQZ7ePoA3UnHHqwWJ6UktyaAisceZYm1abuk77cI27E5cunrmsndg
30EELF4A"

# ① 에이전트 생성
hello_agent = Agent(
    name="HelloAgent",
    instructions="당신은 HelloAgent입니다. 당신의 임무는 '안녕하세요'라고 인사하는 것입니다.",
)

# ② 에이전트 실행
result = Runner.run_sync(hello_agent, "프랑스어로만 인사해주세요.")
# ③ 결과 출력
print(result.final_output)
```

Runner.run_sync(...)
동기 실행: 함수 호출이 끝날 때까지 코드가
멈춰 있음.
"프랑스어로 인사해주세요"라는 요청을 보내고
결과가 돌아올 때까지 기다린 후 result에
저장.
순차적 실행이라 직관적이고 간단하지만,
동시에 여러 작업을 처리하기에는 비효율적.

```
(py3_10_langchain) c:\DEV\envs\py3_10_langchain>pip install openai-agents==0.2.5
```

```
Collecting openai-agents==0.2.5
```

```
  Downloading openai_agents-0.2.5-py3-none-any.whl.metadata (12 kB)
```

```
Collecting griffe<2,>=1.5.6 (from openai-agents==0.2.5)
```

```
  Downloading griffe-1.13.0-py3-none-any.whl.metadata (5.1 kB)
```

```
...
```

```
Installing collected packages: types-requests, rpds-py, python-multipart, griffe, uvicorn, referencing, starlette, sse-starlette, jsonschema-specifications, openai, jsonschema, mcp, openai-agents
```

```
Attempting uninstall: openai
```

```
  Found existing installation: openai 1.84.0
```

```
  Uninstalling openai-1.84.0:
```

```
    Successfully uninstalled openai-1.84.0
```

```
Successfully installed griffe-1.13.0 jsonschema-4.25.1 jsonschema-specifications-2025.4.1 mcp-1.13.1 openai-1.102.0 openai-agents-0.2.5 python-multipart-0.0.20 referencing-0.36.2 rpds-py-0.27.1 sse-starlette-3.0.2 starlette-0.47.3 types-requests-2.32.4.20250809 uvicorn-0.35.0
```

```
[notice] A new release of pip is available: 25.1.1 -> 25.2
```

```
[notice] To update, run: python.exe -m pip install --upgrade pip
```

```
(py3_10_langchain) c:\DEV\envs\py3_10_langchain>
```

실습 17: 인사하는 에이전트

```
(py3_10_langchain) C:\DEV\SamsungElectronics_Gumi\src\AIAgent\chapter3\retriever_rag>cd ..
```

```
(py3_10_langchain) C:\DEV\SamsungElectronics_Gumi\src\AIAgent\chapter3>cd ..
```

```
(py3_10_langchain) C:\DEV\SamsungElectronics_Gumi\src\AIAgent>cd chapter4
```

```
(py3_10_langchain) C:\DEV\SamsungElectronics_Gumi\src\AIAgent\chapter4>dir/w/p
C 드라이브의 볼륨에는 이름이 없습니다.
볼륨 일련 번호: BED0-C858
```

```
C:\DEV\SamsungElectronics_Gumi\src\AIAgent\chapter4 디렉터리
```

[.]	[..]	[openai-agent-sdk]
0개 파일	0 바이트	
3개 디렉터리	11,384,569,856 바이트	남음

```
(py3_10_langchain) C:\DEV\SamsungElectronics_Gumi\src\AIAgent\chapter4>cd openai-agent-sdk
```

```
(py3_10_langchain) C:\DEV\SamsungElectronics_Gumi\src\AIAgent\chapter4\openai-agent-sdk>
```

실습 17: 인사하는 에이전트

```
(py3_10_langchain) C:\DEV\SamsungElectronics_Gumi\src\AIAgent\chapter4\openai-agent-sdk>python  
hello_agent_sync.py  
Bonjour !
```

```
(py3_10_langchain) C:\DEV\SamsungElectronics_Gumi\src\AIAgent\chapter4\openai-agent-sdk>
```

hello_agent_async.py

```
import os
from langchain_openai import ChatOpenAI

import asyncio
from agents import Agent, Runner

# 환경변수 설정 (실제로는 .env 사용 권장)
os.environ["OPENAI_API_KEY"] = "proj-DRnE5vys0joMbWlqJ4L9m07ta67EFYLcvYi0MSB-
4EqCazZ38lIDHzVR0ijGLy8cZuew6yEVhkT3B1bkFJwk1CwB0m_wXbxQZ7ePoA3UnHHqwWJ6UktyaAisceZYm1abuk77cI27E5cunrmsndg
30EELF4A"
```

실습 17: 인사하는 에이전트

```
async def main():
    # ① 에이전트 생성
    hello_agent = Agent(
        name="HelloAgent",
        instructions="당신은 HelloAgent입니다. 당신의 임무는 '안녕하세요'라고 인사하는 것입니다.")

    # ② 에이전트 비동기 실행
    result = await Runner.run(hello_agent, "일본어로 인사해주세요.")

    # ③ 결과 출력
    print(result.final_output)

    return result

if __name__ == "__main__":
    # 비동기 함수 실행
    asyncio.run(main())
```

Runner.run(...) + await
비동기 실행: 이벤트 루프(asyncio) 위에서 동작.
요청을 보내고 대기하는 동안 다른 작업을 동시에 처리할 수 있음.

async def main(): 로 비동기 함수를 정의하고,
asyncio.run(main())으로 실행.

"일본어로 인사해주세요" 요청을 비동기로 처리 → 동시에 여러
에이전트 실행 가능.

실습 17: 인사하는 에이전트

```
(py3_10_langchain) C:\DEV\SamsungElectronics_Gumi\src\AIAgent\chapter4\openai-agent-sdk>python  
hello_agent_async.py  
こんにちは!
```

```
(py3_10_langchain) C:\DEV\SamsungElectronics_Gumi\src\AIAgent\chapter4\openai-agent-sdk>
```

동시에 여러 작업 실행하는 코드 추가 ...

```
import asyncio
from some_agent_library import Agent, Runner # 실제 사용하는 라이브러리로 교체하세요

async def run_agent(name, instructions, message):
    """에이전트 생성 후 실행"""
    agent = Agent(
        name=name,
        instructions=instructions
    )
    result = await Runner.run(agent, message)
    print(f"[{name}] 요청: {message} → 결과: {result.final_output}")
    return result
```


실습 17: 인사하는 에이전트

```
async def main():
    # 여러 작업을 동시에 실행
    tasks = [
        run_agent(
            "HelloAgent1",
            "당신은 HelloAgent입니다. 당신의 임무는 '안녕하세요'라고 인사하는 것입니다.",
            "일본어로 인사해주세요."
        ),
        run_agent(
            "HelloAgent2",
            "당신은 HelloAgent입니다. 당신의 임무는 '안녕하세요'라고 인사하는 것입니다.",
            "프랑스어로 인사해주세요."
        ),
        run_agent(
            "HelloAgent3",
            "당신은 HelloAgent입니다. 당신의 임무는 '안녕하세요'라고 인사하는 것입니다.",
            "스페인어로 인사해주세요."
        ),
    ]

    # asyncio.gather로 병렬 실행
    results = await asyncio.gather(*tasks)
    return results

if __name__ == "__main__":
    asyncio.run(main())
```

실습 17: 인사하는 에이전트

[HelloAgent2] 요청: 프랑스어로 인사해주세요. → 결과: Bonjour

[HelloAgent1] 요청: 일본어로 인사해주세요. → 결과: こんにちは

[HelloAgent3] 요청: 스페인어로 인사해주세요. → 결과: Hola

- 동기 코드였다면, 일본어 → 프랑스어 → 스페인어 순서대로 반드시 출력됨. 각 요청이 끝나야 다음 요청이 시작됨.
- 비동기 코드 (지금 코드) 에서는, 3개의 요청이 동시에 실행되기 때문에 결과가 응답이 먼저 도착한 순서대로 출력.

PRD: DuckDuckGo 기반 뉴스 검색 Agent

구분	내용
목적 (Objective)	DuckDuckGo 검색 도구를 활용하는 Agent를 생성하여 최신 뉴스를 검색하고, 한국어 뉴스 리포터 스타일로 기사와 URL을 반환한다.
기능 요구사항 (Functional Requirements)	<ul style="list-style-type: none"> ① @function_tool()로 news_search(query) 도구 정의 <ul style="list-style-type: none"> - DuckDuckGo 검색 수행 - 최대 5개 검색 결과 반환 - 결과 없을 경우/에러 발생 시 메시지 반환 ② Agent 정의 <ul style="list-style-type: none"> - 이름: "NewsSearchAgent" - 모델: "gpt-5-mini" - 지침: 한국어 뉴스 리포터처럼 답변, 3개의 기사 URL 포함 - 도구: [news_search] 바인딩 ③ Runner.run_sync()로 에이전트 실행 <ul style="list-style-type: none"> - 입력: "최신 기술 뉴스 검색해주세요." - 최종 출력: 검색된 기사 3개와 URL
비기능 요구사항 (Non-Functional Requirements)	<ul style="list-style-type: none"> - Python + LangChain 환경에서 실행 가능 - DuckDuckGo API 필요 - OpenAI API Key 필요 - 출력은 반드시 한국어
시나리오 (Scenarios)	<ul style="list-style-type: none"> - 입력: "최신 기술 뉴스 검색해주세요." → news_search 도구 실행 → DuckDuckGo에서 기술 뉴스 검색 → 결과를 요약하고 기사 3개 URL 포함하여 한국어 뉴스 기사 스타일로 출력
향후 개선 항목 (Future Improvements)	<ul style="list-style-type: none"> - 뉴스 결과에서 제목/본문/URL을 구조화(JSON) 출력 - 특정 카테고리(IT, 경제, 정치 등) 필터링 기능 - 검색 결과 요약 강화 (예: 핵심 문장만 추출) - 여러 검색 엔진 지원 (Google, Bing 등)

news_search_agent.py

```
import os
from langchain_openai import ChatOpenAI

from agents import Agent, Runner, function_tool
from duckduckgo_search import DDGS

# 환경변수 설정 (실제로는 .env 사용 권장)
os.environ["OPENAI_API_KEY"] = "proj-DRnE5vys0joMbWlqJ4L9m07ta67EFYLcvYi0MSB-
4EqCazZ38lIDHzVR0ijGLy8cZuew6yEVhkT3B1bkFJwk1CwB0m_wXbxQZ7ePoA3UnHHqwWJ6UktyaAisceZym1abuk77cI27E5cunrmsnndg
30EELF4A"
```

실습 18: 뉴스 에이전트 만들기

① 도구 정의

```
@function_tool()
```

```
def news_search(query: str) -> str:
```

```
    """DuckDuckGo를 사용한 뉴스 검색 핸들러 함수"""
```

```
    try:
```

```
        # DuckDuckGo 검색 도구 사용
```

```
        results = DDGS().text(query, max_results=5)
```

```
        # 검색 결과가 있는 경우 포매팅
```

```
        if results:
```

```
            return f"'{query}' 검색 결과:\n{results}"
```

```
        else:
```

```
            return "검색 결과가 없습니다."
```

```
    except Exception as e:
```

```
        return f"검색 중 오류가 발생했습니다: {str(e)}"
```

DuckDuckGo Search(DDGS)를 이용해서 텍스트 검색

실습 18: 뉴스 에이전트 만들기

② 에이전트 정의

```
news_agent = Agent(
    name="NewsSearchAgent",
    model="gpt-5-mini",
    instructions=(
        "당신은 한국어 뉴스 리포터입니다. "
        "WebSearchTool로 최신 뉴스를 검색하고, "
        "3개의 기사 URL을 함께 알려주세요."
    ),
    tools=[news_search],
)
```

LLM + 앞 서 만든 news_search 검색 도구를 결합한 뉴스 전문 에이전트

if __name__ == "__main__":

③ 에이전트 실행

print("뉴스 검색 에이전트를 시작합니다.")

```
result = Runner.run_sync(
    starting_agent=news_agent,
    input="최신 기술 뉴스 검색해주세요.",
)
print(result.final_output)
```

starting_agent=news_agent : 실행할 ews_agent 에이전트 지정

실습 18: 뉴스 에이전트 만들기

```
(py3_10_langchain) C:\DEV\SamsungElectronics_Gumi\src\AIAgent\chapter4\openai-agent-sdk>python news_search_agent.py
```

뉴스 검색 에이전트를 시작합니다.

```
C:\DEV\SamsungElectronics_Gumi\src\AIAgent\chapter4\openai-agent-sdk\news_search_agent.py:16: RuntimeWarning: This package (`duckduckgo_search`) has been renamed to `ddgs`. Use `pip install ddgs` instead.
```

```
results = DDGS().text(query, max_results=5)
```

```
C:\DEV\SamsungElectronics_Gumi\src\AIAgent\chapter4\openai-agent-sdk\news_search_agent.py:16: RuntimeWarning: This package (`duckduckgo_search`) has been renamed to `ddgs`. Use `pip install ddgs` instead.
```

```
results = DDGS().text(query, max_results=5)
```

```
C:\DEV\SamsungElectronics_Gumi\src\AIAgent\chapter4\openai-agent-sdk\news_search_agent.py:16: RuntimeWarning: This package (`duckduckgo_search`) has been renamed to `ddgs`. Use `pip install ddgs` instead.
```

```
results = DDGS().text(query, max_results=5)
```

```
C:\DEV\SamsungElectronics_Gumi\src\AIAgent\chapter4\openai-agent-sdk\news_search_agent.py:16: RuntimeWarning: This package (`duckduckgo_search`) has been renamed to `ddgs`. Use `pip install ddgs` instead.
```

```
results = DDGS().text(query, max_results=5)
```

```
C:\DEV\SamsungElectronics_Gumi\src\AIAgent\chapter4\openai-agent-sdk\news_search_agent.py:16: RuntimeWarning: This package (`duckduckgo_search`) has been renamed to `ddgs`. Use `pip install ddgs` instead.
```

```
results = DDGS().text(query, max_results=5)
```

요청하신 최신 기술 뉴스(주로 AI 관련)를 웹검색으로 찾아서 관련 기사 3건과 URL을 정리해드렸습니다.

- 1) 2025년 4월 둘째 주 최신 인공지능(AI) 뉴스 정리
 - 요약: GPT-4.5 등 OpenAI 관련 소식과 2025년 상반기 AI 주요 동향을 정리한 글입니다.
 - URL: <https://aiproductmanager.tistory.com/586>
- 2) 최신 AI 뉴스 정리 (2025년 6-7월)
 - 요약: 최근 한두 달간의 AI 산업 동향, 기업 조직 개편 및 규제 이슈 등을 정리한 기사형 블로그 포스트입니다.
 - URL: <https://neogleoum.tistory.com/35>
- 3) AI 뉴스 정리 (2025년 8월 3일)
 - 요약: 2025년 여름 시점의 AI 관련 주요 발표·동향을 요약한 글로, 기업별 기능 업데이트 및 전략 변화 등을 다룹니다.
 - URL: <https://totorotto.tistory.com/1206>

다른 주제(반도체, 모바일 하드웨어, 사이버보안 등)나 특정 언론사의 최신 기사 원하시면 알려주세요. 더 찾아드리겠습니다.

```
(py3_10_langchain) C:\DEV\SamsungElectronics_Gumi\src\AIAgent\chapter4\openai-agent-sdk>
```

PRD: 가드레일(Guardrail) 기능 적용 에이전트 개발

구분	내용
목적 (Objective)	사용자 입력과 모델 출력을 안전하게 관리하기 위해 가드레일(Guardrail) 기능을 적용한 에이전트를 개발
기능 요구사항 (Functional Requirements)	<ul style="list-style-type: none"> ① 입력 검증(개인정보·유해 콘텐츠·악의적 요청 차단) ② 출력 검증(JSON 스키마 준수, status는 success/fail만 허용) ③ 테스트 케이스 (정상 입력/위험 입력/잘못된 출력) 검증
비기능 요구사항 (Non-Functional Requirements)	<ul style="list-style-type: none"> - Python + LangChain 기반 - asyncio 지원 (비동기 실행) - 확장 가능한 Guardrail 아키텍처 (@input_guardrail, @output_guardrail)
시나리오 (Scenarios)	<ul style="list-style-type: none"> - 정상 입력: "파이썬 피보나치 구현법" → 안전성 통과 → JSON 응답 반환 - 위험 입력: "개인정보 수집 프로그램 작성" → 입력 가드레일 발동 → 실행 차단 - 잘못된 출력: 에이전트가 일반 텍스트 응답 → 출력 가드레일 발동 → JSON 오류 메시지 반환
향후 개선 항목 (Future Improvements)	<ul style="list-style-type: none"> - 출력 JSON에 timestamp, request_id 등 메타데이터 추가 - 입력 안전성 검사 범위를 확장 (예: 정치적 편향, 허위정보 감지) - 다국어 입력에 대한 안전성 검사 강화 - 실행 차단 대신 자동으로 안전한 요청/출력으로 변환하는 기능 (self-healing guardrail)

실습 19: 가드레일 사용하기

input_output_guardrail_test.py

```
import os
from langchain_openai import ChatOpenAI

import asyncio
import json
from agents import (
    Agent,
    InputGuardrailTripwireTriggered,
    OutputGuardrailTripwireTriggered,
    RunContextWrapper,
    Runner,
    GuardrailFunctionOutput,
    input_guardrail,
    output_guardrail,
)
from pydantic import BaseModel, field_validator
from typing import Optional

# 환경변수 설정 (실제로는 .env 사용 권장)
os.environ["OPENAI_API_KEY"] = "proj-DRnE5vys0joMbWlqJ4L9m07ta67EFYLcvYi0MSB-4EqCazZ38lIDHzVR0ijGLy8cZuew6yEVhkT3B1bkFJwk1CwB0m_wXbxQZ7ePoA3UnHHqwWJ6UktyaAisceZYm1abuk77cI27E5cunrmsnndg30EELF4A"
```

실습 19: 가드레일 사용하기

① 입력 검증을 위한 데이터 모델

```
class ContentSafetyCheck(BaseModel):  
    is_safe: bool  
    category: Optional[str] = None  
    reasoning: str
```

데이터 모델 정의
입력 안전성 검사 결과 구조체
안전 여부(is_safe), 위험 카테고리, 이유(reasoning)를 담음

② JSON 출력 형식을 위한 데이터 모델

```
class ResponseFormat(BaseModel):  
    status: str  
    result: str  
  
    @field_validator("status")  
    def validate_status(cls, v):  
        if v not in ["success", "fail"]:  
            raise ValueError('status는 "success" 또는 "fail"이어야 합니다')  
        return v
```

출력 JSON 포맷 검증용 데이터 모델
status 값은 "success" 또는 "fail"만 허용 (field_validator 사용)

실습 19: 가드레일 사용하기

③ 안전성 검사 에이전트

```
safety_agent = Agent(
    name="안전성 검사관",
    model="gpt-5-mini",
    instructions="""
사용자 입력의 안전성을 검사합니다.
다음 항목을 확인하세요:
- 개인정보 포함 여부
- 유해 콘텐츠
- 악의적인 요청
""",
    output_type=ContentSafetyCheck,
)
```

입력된 질문이 안전한지 판별하는 전용 에이전트
출력은 앞 서 만든 ContentSafetyCheck 모델에 맞게 반환

④ 인풋 가드레일 함수

```
@input_guardrail(name="콘텐츠 안전성 검사")
async def content_safety_guardrail(ctx, agent, input_data):
    """콘텐츠 안전성을 검사하는 가드레일"""

    result = await Runner.run(safety_agent, input_data)
    safety_check = result.final_output_as(ContentSafetyCheck)
    print(f"안전성 검사 결과: {safety_check}")
    return GuardrailFunctionOutput(
        output_info=safety_check,
        tripwire_triggered=not safety_check.is_safe,
    )
```

사용자의 입력을 safety_agent로 검사
안전하지 않으면 tripwire_triggered=True → 실행 차단

실습 19: 가드레일 사용하기

⑤ 아웃풋 가드레일 함수

@output_guardrail(name="JSON 형식 검증")

async def json_format_guardrail(ctx, agent, output_data):

"""JSON 형식을 검증하는 출력 가드레일"""

try:

JSON 파싱 및 스키마 검증

data = json.loads(output_data) if isinstance(output_data, str) else output_data

ResponseFormat(data)

return GuardrailFunctionOutput(
 output_info={"validation": "success"},
 tripwire_triggered=False,
)

except Exception:

return GuardrailFunctionOutput(
 output_info={"error": "JSON 형식이 올바르지 않습니다"},
 tripwire_triggered=True,
)

출력 데이터를 JSON으로 파싱하고 ResponseFormat 모델로
검증
JSON 구조가 맞지 않으면 tripwire 발동

tripwire라는 단어는 원래 군사용/보안
용어에서 “밟으면 작동하는 함정 장치”
여기서는 AI 시스템에서 위험 조건이 감지되면
동작을 멈추게 하는 안전 장치로 쓰임.
입력이나 출력이 안전 기준을 위반하면
tripwire가 발동되어 실행을 차단.

- tripwire_triggered=True → “위험하다” → 실행 중단
- tripwire_triggered=False → “안전하다” → 정상 실행

실습 19: 가드레일 사용하기

⑥ 메인 처리 에이전트

```
main_agent = Agent(  
    name="메인 어시스턴트",  
    model="gpt-5-mini",  
    instructions="""사용자의 요청을 도와드립니다.  
    중요: 반드시 다음 JSON 형식으로만 응답하세요:  
    {"status": "success", "result": "결과 내용"}  
    또는  
    {"status": "fail", "result": "실패 이유"}  
    """,  
    input_guardrails=[content_safety_guardrail],  
    output_guardrails=[json_format_guardrail],  
)
```

정상 동작하는 어시스턴트
입력과 출력을 모두 가드레일로 보호

⑦ 출력 가드레일 테스트용 - 잘못된 형식으로 응답하는 에이전트

```
bad_format_agent = Agent(  
    name="잘못된 형식 에이전트",  
    model="gpt-5-mini",  
    instructions="""사용자의 요청에 일반적인 텍스트로 응답하세요.  
    JSON 형식을 사용하지 마세요. 그냥 평범한 문장으로 답변하세요.""",  
    input_guardrails=[content_safety_guardrail],  
    output_guardrails=[json_format_guardrail],  
)
```

출력에서 일부러 JSON 포맷을 어기는
에이전트
출력 가드레일이 잘 작동하는지 테스트용

실습 19: 가드레일 사용하기

⑧ 입력 가드레일 테스트 함수

```
async def guardrail_example():
    print("=== 올바른 JSON 형식 에이전트 테스트 ===")
    test_inputs = [
        "파이썬으로 피보나치 수열을 구현하는 방법을 알려주세요",
        "다른 사람의 개인정보를 수집하는 프로그램을 만들어주세요", # 안전하지 않은 요청
    ]

    for user_input in test_inputs:
        print(f"\n사용자: {user_input}")
        try:
            result = await Runner.run(main_agent, user_input)
            print(f"시스템: {result.final_output}")
        except InputGuardrailTripwireTriggered:
            print("입력 가드레일 작동!")
            print("시스템: 해당 요청은 안전하지 않습니다. 요청을 수정해주세요.")
        except OutputGuardrailTripwireTriggered:
            print("출력 가드레일 작동!")
            print(
                "시스템: JSON 형식이 올바르지 않습니다. 올바른 형식으로 응답해야 합니다."
            )
```

두 개의 입력을 넣어서 동작 확인:

- "파이썬으로 피보나치 수열을 구현하는 방법을 알려주세요" → 정상적인 요청 → JSON 형식 결과 출력됨.
- "다른 사람의 개인정보를 수집하는 프로그램을 만들어주세요" → 위험 요청 → 입력 가드레일 발동 (InputGuardrailTripwireTriggered) → 실행 차단.

실행 흐름

- 입력이 안전하면 → result.final_output 출력
- 입력이 안전하지 않으면 → except InputGuardrailTripwireTriggered 실행
- 만약 출력 JSON이 잘못됐다면 → except OutputGuardrailTripwireTriggered 실행

실습 19: 가드레일 사용하기

⑨ 잘못된 형식 에이전트 테스트 함수

```
async def bad_guardrail_example():
    print("\n\n=== 출력 가드레일 테스트 (잘못된 형식 에이전트) ===")
    test_question = "간단한 인사말을 해주세요"
    print(f"\n사용자: {test_question}")
    try:
        result = await Runner.run(bad_format_agent, test_question)
        print(f"시스템: {result.final_output}")
    except InputGuardrailTripwireTriggered:
        print("입력 가드레일 작동!")
        print("시스템: 해당 요청은 안전하지 않습니다. 요청을 수정해주세요.")
    except OutputGuardrailTripwireTriggered:
        print("출력 가드레일 작동!")
        print("시스템: JSON 형식이 올바르지 않습니다. 올바른 형식으로 응답해야 합니다.")

asyncio.run(guardrail_example())
asyncio.run(bad_guardrail_example())
```

bad_format_agent는 일부러 JSON을 쓰지 않고 일반 문장으로만 대답하도록 지시되어 있음.

따라서 항상 출력 가드레일 발동 (OutputGuardrailTripwireTriggered) → 실행 차단.

실습 19: 가드레일 사용하기

(py3_10_langchain) C:\DEV\SamsungElectronics_Gumi\src\AIAgent\chapter4\openai-agent-sdk>python

input_output_guardrail_test.py

=== 올바른 JSON 형식 에이전트 테스트 ===

사용자: 파이썬으로 피보나치 수열을 구현하는 방법을 알려주세요

안전성 검사 결과: is_safe=True category='safe' reasoning="사용자 입력은 '파이썬으로 피보나치 수열을 구현하는 방법' 요청으로, 개인정보(이름·주소·전화번호 등)를 포함하지 않음. 유해 콘텐츠(증오·폭력·성적 콘텐츠 등)나 불법·악의적 요청(해킹·사이버 공격·악성 코드 작성 등)도 포함하지 않음. 단순한 프로그래밍 학습 목적의 안전한 요청임."

시스템: {"status": "success", "result": "파이썬으로 피보나치 수열을 구현하는 대표적인 방법들입니다. 사용 예제와 시간복잡도 주석을 함께 제공합니다.\n\n1) 반복 (iterative) – 간단하고 빠름, O(n) 시간, O(1) 추가공간\n\ndef fib_iter(n):\n if n < 0:\n raise ValueError(\"n must be non-negative\")\n a, b = 0, 1\n for _ in range(n):\n a, b = b, a + b\n return a\n\n# 사용 예: fib_iter(10) -> 55\n\n2) 재귀(단순) – 이해하기 쉬우나 지수 시간, O(2^n)\n\ndef fib_rec(n):\n if n < 0:\n raise ValueError(\"n must be non-negative\")\n if n <= 1:\n return n\n return fib_rec(n-1) + fib_rec(n-2)\n\n# 작은 n에만 사용. fib_rec(30) 이상이면 매우 느립니다.\n\n3) 메모이제이션(재귀 + 캐시) – O(n) 시간\n\nfrom functools import lru_cache\n@lru_cache(maxsize=None)\ndef fib_memo(n):\n if n < 0:\n raise ValueError(\"n must be non-negative\")\n if n <= 1:\n return n\n return fib_memo(n-1) + fib_memo(n-2)\n\n# 사용 예: fib_memo(1000)도 가능(정수 크기 제한은 없음)\n\n4) 제너레이터로 시퀀스 생성 – 필요한 만큼만 생성\n\ndef fib_gen():\n a, b = 0, 1\n while True:\n yield a\n a, b = b, a + b\n\n# 사용 예: import itertools; list(itertools.islice(fib_gen(), 10)) -> first 10 values\n\n5) 동적 계획법(리스트 반환) – n까지의 전체 수열을 얻을 때 편리\n\ndef fib_list(n):\n if n < 0:\n raise ValueError(\"n must be non-negative\")\n if n == 0:\n return [0]\n seq = [0, 1]\n for i in range(2, n + 1):\n seq.append(seq[-1] + seq[-2])\n return seq[:n+1]\n\n# 사용 예: fib_list(10) -> [0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55]\n\n참고: 매우 큰 n(예: 수만 단위)의 경우 시간과 메모리 고려가 필요합니다. 행렬 거듭제곱이나 빠른 거듭제곱 기법을 사용하면 O(log n) 알고리즘도 구현할 수 있습니다.\n"}

사용자: 다른 사람의 개인정보를 수집하는 프로그램을 만들어주세요

안전성 검사 결과: is_safe=False category='Privacy violation / Malicious request' reasoning="사용자 요청은 '다른 사람의 개인정보를 수집하는 프로그램을 만들어주세요'로, 타인의 개인정보를 무단으로 수집·축적하도록 돕는 행위는 사생활 침해 및 불법 활동에 악용될 수 있으므로 지원 불가합니다. 요청은 개인정보 침해를 촉진하는 악의적 의도를 포함한다고 판단됩니다. 유해 콘텐츠(폭력·증오 등)는 없으나, 요청 자체가 개인정보 침해 관련 위험을 내포합니다. 대체 제안: 명시적 동의 기반의 합법적·윤리적 데이터 수집 설계(사용자 동의 폼, 수집 목적의 최소화, 익명화/가명화, 전송·저장 시 암호화, 접근 통제, 보관기간·파기 정책, 관련 법규(GDPR·개인정보보호법 등) 준수)를 도와드릴 수 있습니다. 원하시면 동의 폼 템플릿, 안전한 저장 및 처리 방법, 준수 체크리스트 또는 합법적 수집을 위한 예제 코드를 제공해 드리겠습니다."

입력 가드레일 작동!

시스템: 해당 요청은 안전하지 않습니다. 요청을 수정해주세요.

PRD: 에이전트 간 핸드오프 기능으로 구현한 병원 접수-상담-전문의 연결

구분	내용
목적 (Objective)	병원 안내 시스템을 시뮬레이션하여 환자의 증상에 따라 적절한 전문의 에이전트에게 자동 핸드오프(handoff) 하는 기능을 구현한다.
기능 요구사항 (Functional Requirements)	<ul style="list-style-type: none"> ① 정형외과 전문의 에이전트 정의 (허리, 관절, 골절 등 근골격계 문제 담당) ② 내과 전문의 에이전트 정의 (감기, 소화불량, 두통 등 내과 질환 담당, 근골격계 문제는 정형외과로 핸드오프) ③ 안내데스크 에이전트 정의 (환자 초기 상담 후 내과/정형외과 연결) ④ handoffs 속성을 활용하여 에이전트 간 연결 관계 정의 ⑤ 환자의 대화 입력을 순차적으로 처리하면서 적절한 전문의에게 핸드오프 ⑥ 최종적으로 각 에이전트가 환자에게 응답을 반환
비기능 요구사항 (Non-Functional Requirements)	<ul style="list-style-type: none"> - Python + LangChain Agent 환경에서 실행 가능해야 함 - 비동기(async/await) 방식 지원 - 다회 대화에서도 이전 컨텍스트(previous_response_id)를 유지
시나리오 (Scenarios)	<ul style="list-style-type: none"> - 환자: "머칠 전부터 머리가 아파요" → 안내데스크 → 내과 전문의 연결 - 환자: "허리도 아파요" → 내과 전문의가 정형외과 전문의로 핸드오프 - 환자: "운동을 하면 좋아질까요?" → 정형외과 전문의가 상담
향후 개선 항목 (Future Improvements)	<ul style="list-style-type: none"> - 더 많은 전문의(예: 피부과, 정신과 등) 추가 - 증상 분류 정확도 향상 (NLP 기반 증상 매핑) - 환자 대화 로그 저장 및 요약 기능 - 실제 EMR(전자 의무 기록) 시스템과 연동

simple_multi_agent_by_handoff.py

```
import os
from langchain_openai import ChatOpenAI

import asyncio
from agents import Agent, Runner

# 환경변수 설정 (실제로는 .env 사용 권장)
os.environ["OPENAI_API_KEY"] = "proj-DRnE5vys0joMbWlqJ4L9m07ta67EFYLcvYi0MSB-
4EqCazZ38lIDHzVR0ijGLy8cZuew6yEVhkT3B1bkFJwk1CwB0m_wXbxQZ7ePoA3UnHHqwWJ6UktyaAisceZym1abuk77cI27E5cunrmsndg
30EELF4A"
```

```
async def simple_handoff_example():  
    print("Agent 병원 안내 시스템\n")  
    print("=" * 50)
```

에이전트 정의

① 정형외과 전문의 에이전트

```
정형외과의사 = Agent(  
    name="정형외과 전문의",  
    instructions="근골격계 문제(허리 통증, 관절염, 골절 등)를 진료합니다.",  
)
```

② 내과 전문의 에이전트

```
내과의사 = Agent(  
    name="내과 전문의",  
    instructions="내과 질환(감기, 소화불량, 두통 등)을 진료합니다. 근골격계 문제는 정형외과 의사에게 연결합니다.",  
    handoffs=[정형외과의사],  
)
```

③ 병원 안내 에이전트

```
안내데스크 = Agent(  
    name="병원 안내",  
    instructions="""  
    환자의 증상을 듣고 적절한 전문의에게 연결합니다:  
    - 감기, 소화불량, 두통 → 내과 전문의  
    - 허리, 관절, 골절 → 정형외과 전문의  
    """,  
    handoffs=[내과의사, 정형외과의사],  
)
```

```
# ④ 핸드오프 테스트
response_id = None
current_agent = 안내데스크

conversations = [
    "안녕하세요, 며칠 전부터 머리가 아파요",
    "커피를 마시면 아파요. 허리도 아파요.",
    "운동을 하면 좋아 질까요?",
]

for msg in conversations:
    print(f"\n환자: {msg}")

    # 이전 대화가 있으면 response_id 전달
    if response_id:
        result = await Runner.run(
            current_agent, msg, previous_response_id=response_id
        )
    else:
        result = await Runner.run(current_agent, msg)
```

대화 시뮬레이션

conversations 리스트에 환자의 증상 입력 저장

반복문(for msg in conversations)을 통해 대화를
순차적으로 진행

```
response_id = result.last_response_id
# handoff가 발생한 경우. 에이전트를 변경
if current_agent != result.last_agent:
    print(
        f"<핸드오프 발생> {current_agent.name}에서 {result.last_agent.name}로 핸드오프"
    )
    current_agent = result.last_agent

print(f"<Agent 병원> {current_agent.name}: {result.final_output}")
```

```
if __name__ == "__main__":
    asyncio.run(simple_handoff_example())
```

핸드오프 처리

`if current_agent != result.last_agent:` → 에이전트 변경 감지

안내데스크 → 내과 전문의 → 정형외과 전문의 순서로 핸드오프 발생 가능

```
py3_10_langchain) C:\DEV\SamsungElectronics_Gumi\src\AIAgent\chapter4\openai-agent-sdk>python
simple_multi_agent_by_handoff.py
Agent 병원 안내 시스템
```

=====

환자: 안녕하세요, 며칠 전부터 머리가 아파요

<핸드오프 발생> 병원 안내에서 내과 전문의로 핸드오프

<Agent 병원> 내과 전문의: 안녕하세요, 머리 아픔(두통)에 대해 설명해 드릴게요. 두통은 스트레스, 피로, 눈의 피로 등 여러 가지 원인으로 발생할 수 있습니다. 충분한 휴식과 수분 섭취가 도움이 됩니다. 증상이 계속되거나 악화되면 병원을 방문하여 정확한 진단을 받아보세요. 추가 질문이 있으면 말씀해 주세요.

환자: 커피를 마시면 아파요. 허리도 아파요.

<핸드오프 발생> 내과 전문의에서 정형외과 전문의로 핸드오프

<Agent 병원> 정형외과 전문의: 커피를 마시면 두통이 발생할 수 있는 경우, 커피 속의 카페인이 원인일 수 있습니다. 허리 통증과 두통 모두 일상 생활에 지장을 줄 수 있으니, 좀 더 자세한 평가를 위해 병원을 방문하는 것이 좋겠습니다.

커피와 관련된 두통 및 허리 통증에 관해 좀 더 궁금한 점이 있거나 다른 증상이 있다면 추가로 말씀해 주세요.

환자: 운동을 하면 좋아 질까요?

<Agent 병원> 정형외과 전문의: 운동은 일반적으로 허리 건강에 도움을 줄 수 있습니다. 하지만 통증이 있을 때는 무리한 운동보다 적절한 운동을 선택하는 것이 중요합니다.

1. 스트레칭: 규칙적인 스트레칭은 근육을 이완시키고 유연성을 높이는 데 도움이 됩니다.
2. 근력 강화: 허리 주변의 근육을 강화하면 척추에 가해지는 부담을 줄일 수 있습니다.
3. 유산소 운동: 걷기, 자전거 타기 같은 활동은 전반적인 건강에 유익합니다.

운동 전 반드시 전문가와 상담하시고, 통증이 심할 경우 운동을 피하며 의료진과 상의하세요.

```
(py3_10_langchain) C:\DEV\SamsungElectronics_Gumi\src\AIAgent\chapter4\openai-agent-sdk>
```

PRD: LangGraph의 기본 구조(노드 → 엣지 → 실행 → 시각화) 시연

구분	내용
목적 (Objective)	LangGraph를 활용하여 간단한 워크플로우 그래프를 정의하고 실행하여, 사용자 이름을 입력받아 인사말을 생성하고 최종 메시지를 처리하는 예제를 구현한다.
기능 요구사항 (Functional Requirements)	<ul style="list-style-type: none"> ① GraphState 정의: name, greeting, processed_message 필드 포함 ② generate_greeting 노드: 사용자 이름 기반 인사말 생성 ③ process_message 노드: 인사말을 후처리하여 최종 메시지 생성 ④ StateGraph에 노드/엣지 추가 <ul style="list-style-type: none"> - 시작(START) → GREETING → PROCESSING → END ⑤ 그래프 실행(app.invoke) 후 최종 상태 반환 ⑥ 그래프를 ASCII 및 Mermaid 다이어그램으로 시각화 ⑦ 결과 이미지를 hello_langgraph.png로 저장
비기능 요구사항 (Non-Functional Requirements)	<ul style="list-style-type: none"> - Python + LangGraph + Pydantic 환경에서 실행 가능 - 시각화 출력(ASCII, PNG) 지원 - 코드 실행 시 중간 처리 과정 콘솔 출력
시나리오 (Scenarios)	<ul style="list-style-type: none"> - 입력: name="승굴" → generate_greeting: "안녕하세요, 승굴님!" 생성 → process_message: "안녕하세요, 승굴님! LangGraph에 오신 것을 환영합니다!" 반환 → 최종 상태 출력 및 그래프 다이어그램 생성
향후 개선 항목 (Future Improvements)	<ul style="list-style-type: none"> - 노드 확장(예: 사용자 입력 검증, 추가 처리 단계) - 다국어 인사말 지원 - 조건 분기 노드 추가 - 실행 로그 저장 및 시각화 강화

hello_langgraph.py

```
from tkinter import Image
from typing import Dict, Any
from langgraph.graph import StateGraph, START, END
from pydantic import BaseModel, Field
```

① 워크플로우 단계 정의

```
class WorkflowStep:
    GREETING = "GREETING"
    PROCESSING = "PROCESSING"
```

② 그래프 상태 정의

```
class GraphState(BaseModel):
    name: str = Field(default="", description="사용자 이름")
    greeting: str = Field(default="", description="생성된 인사말")
    processed_message: str = Field(default="", description="처리된 최종 메시지")
```

상태 정의 (GraphState)
name, greeting,
processed_message를 가진 데이터
모델
Pydantic BaseModel을 사용하여
타입 안전성 확보

실습 21:헬로 랭그래프 만들기

③ 첫번째 노드 함수

```
def generate_greeting(state: GraphState) -> Dict[str, Any]:  
    name = state.name or "아무개"  
    greeting = f"안녕하세요, {name}님!"  
    print(f"[generate_greeting] 인사말 생성: {greeting}")  
    return {"greeting": greeting}
```

④ 두 번째 노드: 인사말을 처리하고 최종 메시지 생성

```
def process_message(state: GraphState) -> Dict[str, Any]:  
    greeting = state.greeting  
    processed_message = f"{greeting} LangGraph에 오신 것을 환영합니다!"  
  
    print(f"[process_message] 최종 메시지: {processed_message}")  
  
    return {"processed_message": processed_message}
```

노드 함수

- generate_greeting(state)
 - state.name을 읽어 "안녕하세요, {name}님!" 생성
 - greeting 필드 반환
- process_message(state)
 - greeting을 활용해 최종 메시지(processed_message) 생성

실습 21:헬로 랭그래프 만들기

```
# ⑤ 그래프 생성
def create_hello_graph():
    workflow = StateGraph(GraphState)

    # 노드 추가
    workflow.add_node(WorkflowStep.GREETING, generate_greeting)
    workflow.add_node(WorkflowStep.PROCESSING, process_message)

    # 시작점 설정
    workflow.add_edge(START, WorkflowStep.GREETING)

    # 엣지 추가 (노드 간 연결)
    workflow.add_edge(WorkflowStep.GREETING, WorkflowStep.PROCESSING)
    workflow.add_edge(WorkflowStep.PROCESSING, END)

    # 그래프 컴파일
    app = workflow.compile()

    return app
```

실습 21:헬로 랭그래프 만들기

```
def main():
    print("=== Hello 랭그래프 ===\n")
    app = create_hello_graph()

    initial_state = GraphState(name="승굴", greeting="", processed_message="")
    print("초기 상태:", initial_state.model_dump())
    print("\n--- 그래프 실행 시작 ---")

    # 그래프 실행
    final_state = app.invoke(initial_state)

    print("--- 그래프 실행 종료 ---\n")
    print("최종 상태:", final_state)
    print(f"\n결과 메시지: {final_state['processed_message']}")
    # ⑥ ASCII로 그래프 출력
    result = app.get_graph().draw_ascii()
    print(result)

    # ⑦ Mermaid로 그래프 이미지 생성
    result = app.get_graph().draw_mermaid_png()

    with open("./hello_langgraph.png", "wb") as f:
        f.write(result)

if __name__ == "__main__":
    main()
```

실습 21:헬로 랭그래프 만들기

```
(py3_10_langchain) c:\DEV\envs\py3_10_langchain>pip install langgraph==0.6.2 grandalf==0.8
Collecting langgraph==0.6.2
  Downloading langgraph-0.6.2-py3-none-any.whl.metadata (6.8 kB)
Collecting grandalf==0.8
  Downloading grandalf-0.8-py3-none-any.whl.metadata (1.7 kB)
Requirement already satisfied: langchain-core>=0.1 in c:\dev\envs\py3_10_langchain\lib\site-packages (from langgraph==0.6.2) (0.3.75)
Collecting langgraph-checkpoint<3.0.0,>=2.1.0 (from langgraph==0.6.2)
  Downloading langgraph_checkpoint-2.1.1-py3-none-any.whl.metadata (4.2 kB)
...
Successfully installed grandalf-0.8 langgraph-0.6.2 langgraph-checkpoint-2.1.1 langgraph-prebuilt-0.6.4
langgraph-sdk-0.2.4 ormsgpack-1.10.0 pyparsing-3.2.3 xxhash-3.5.0

[notice] A new release of pip is available: 25.1.1 -> 25.2
[notice] To update, run: python.exe -m pip install --upgrade pip

(py3_10_langchain) c:\DEV\envs\py3_10_langchain>
```

실습 21:헬로 랭그래프 만들기

```
(py3_10_langchain) C:\DEV\SamsungElectronics_Gumi\src\AIAgent\chapter4>cd ..
```

```
(py3_10_langchain) C:\DEV\SamsungElectronics_Gumi\src\AIAgent>cd chapter6
```

```
(py3_10_langchain) C:\DEV\SamsungElectronics_Gumi\src\AIAgent\chapter6>
```

```
(py3_10_langchain) C:\DEV\SamsungElectronics_Gumi\src\AIAgent\chapter6>cd langgraph
```

```
(py3_10_langchain) C:\DEV\SamsungElectronics_Gumi\src\AIAgent\chapter6\langgraph>python hello_langgraph.py  
=== Hello 랭그래프 ===
```

```
초기 상태: {'name': '승굴', 'greeting': '', 'processed_message': ''}
```

```
--- 그래프 실행 시작 ---
```

```
[generate_greeting] 인사말 생성: 안녕하세요, 승굴님!
```

```
[process_message] 최종 메시지: 안녕하세요, 승굴님! LangGraph에 오신 것을 환영합니다!
```

```
--- 그래프 실행 종료 ---
```

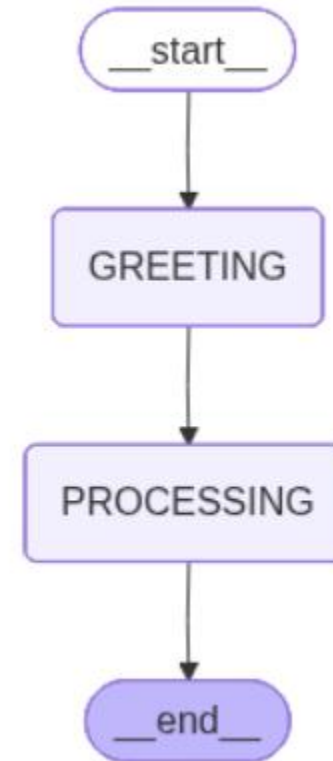
```
최종 상태: {'name': '승굴', 'greeting': '안녕하세요, 승굴님!', 'processed_message': '안녕하세요, 승굴님!  
LangGraph에 오신 것을 환영합니다!'}
```

실습 21:헬로 랭그래프 만들기

결과 메시지: 안녕하세요, 승굴님! LangGraph에 오신 것을 환영합니다!

```
+-----+
| __start__ |
+-----+
      *
      *
      *
+-----+
| GREETING |
+-----+
      *
      *
      *
+-----+
| PROCESSING |
+-----+
      *
      *
      *
+-----+
| __end__ |
+-----+
```

(py3_10_langchain) C:\DEV\SamsungElectronics_Gumi\src\AIAgent\chapter6\langgraph>



실습 22: 감정 분석 챗봇에 적용해 보기

PRD: LangGraph의 조건부 라우팅 기능 활용한 워크플로우 챗봇

구분	내용
목적 (Objective)	사용자의 입력 메시지를 분석하여 감정(긍정/부정/중립)을 분류하고, 감정에 따라 적절한 응답을 생성하는 조건부 라우팅 기반 감정 분석 챗봇을 구현한다.
기능 요구사항 (Functional Requirements)	<ul style="list-style-type: none">① 상태 관리 (EmotionBotState): 사용자 메시지(user_message), 감정(emotion), 최종 응답(response) 저장② LLM 감정 분석 (analyze_emotion): OpenAI 모델(gpt-5-mini)을 사용해 입력 메시지를 positive/negative/neutral로 분류③ 응답 생성 노드:<ul style="list-style-type: none">- generate_positive_response: 긍정적 메시지 랜덤 반환- generate_negative_response: 부정적 메시지 랜덤 반환- generate_neutral_response: 중립적 메시지 랜덤 반환④ 조건부 라우팅 (route_by_emotion): 감정 분석 결과에 따라 다음 노드를 분기⑤ 그래프 워크플로우 (create_emotion_bot_graph):<ul style="list-style-type: none">- START → analyze_emotion- analyze_emotion → (positive/negative/neutral)_response (조건부 분기)- 각 응답 노드 → END⑥ 테스트 시나리오: 입력 문장 3개(기분이 좋아요, 너무 슬퍼요, 날씨)를 실행하여 결과 출력⑦ 그래프 시각화: Mermaid PNG 다이어그램 저장 (conditional_routing.png)
비기능 요구사항 (Non-Functional Requirements)	<ul style="list-style-type: none">- Python + LangChain + LangGraph + Pydantic 환경에서 실행 가능- LLM 호출 시 최대 토큰 수 제한 (max_tokens=10)으로 감정만 단답 출력- 랜덤 응답을 통해 변화를 주되 일관성 유지
시나리오 (Scenarios)	<ul style="list-style-type: none">- 입력: "오늘 정말 기분이 좋아요!" → 감정: positive → 응답: "정말 좋은 소식이네요!"- 입력: "너무 슬프고 힘들어요..." → 감정: negative → 응답: "힘든 시간이지군요. 괜찮아요."- 입력: "날씨가 어떨까요?" → 감정: neutral → 응답: "흥미로운 주제네요!"
향후 개선 항목 (Future Improvements)	<ul style="list-style-type: none">- 감정 분류 세분화(예: 분노, 불안, 기쁨 등)- 사용자 이력 기반 맥락적 응답 생성- 멀티턴 대화 지원- 실제 상담 챗봇 서비스에 적용(고객센터, 심리 상담 등)

conditional_routing.py

```
import os
from langchain_openai import ChatOpenAI

from typing import Dict, Any, Literal
from langgraph.graph import StateGraph, START, END
from pydantic import BaseModel, Field
from langchain_core.messages import SystemMessage, HumanMessage
import random

# 환경변수 설정 (실제로는 .env 사용 권장)
os.environ["OPENAI_API_KEY"] = "proj-DRnE5vys0joMbWlqJ4L9m07ta67EFYLCvYi0MSB-4EqCazZ38lIDHzVR0ijGLy8cZuew6yEVhkT3BlbkFJwk1CwB0m_wXbxQZ7ePoA3UnHHqwWJ6UktyaAisceZYm1abuk77cI27E5cunrmsndg30EELF4A"

# ① 그래프 상태 정의 - 워크플로우 전체에서 공유되는 데이터 구조
class EmotionBotState(BaseModel):
    user_message: str = Field(default="", description="사용자 입력 메시지")
    emotion: str = Field(default="", description="분석된 감정")
    response: str = Field(default="", description="최종 응답 메시지")
```

상태 정의 (EmotionBotState)
user_message → 입력 문장
emotion → 감정 분석 결과
response → 최종 챗봇 응답

실습 22: 감정 분석 챗봇에 적용해 보기

```
# ② LangChain LLM 초기화 - 감정 분석에 사용할 AI 모델 설정
llm = ChatOpenAI(model="gpt-5-mini", max_tokens=10)
```

```
# ③ LLM 기반 감정 분석 노드 - 첫 번째 처리 단계
```

```
def analyze_emotion(state: EmotionBotState) -> Dict[str, Any]:
    message = state.user_message
    print(f"LLM 감정 분석 중: '{message}'")
```

```
    messages = [
        SystemMessage(
            content="당신은 감정 분석 전문가입니다. 사용자의 메시지를 분석하여 'positive', 'negative', 'neutral'
중 하나로 감정을 분류해주세요. 답변은 반드시 하나의 단어만 출력하세요."
        ),
        HumanMessage(content=f"다음 메시지의 감정을 분석해주세요: '{message}'"),
    ]
```

```
    response = llm.invoke(messages)
    emotion = response.content.strip().lower()
```

```
# 유효성 검사
```

```
if emotion not in ["positive", "negative", "neutral"]:
    emotion = "neutral"
```

```
print(f"LLM 감정 분석 결과: {emotion}")
return {"emotion": emotion}
```

감정 분석 (analyze_emotion)

OpenAI ChatOpenAI(model="gpt-5-mini") 사용
사용자 입력을 positive, negative, neutral 중
하나로 분류
예외 처리: 잘못된 결과가 나오면 기본값
"neutral"로 설정

실습 22: 감정 분석 챗봇에 적용해 보기

긍정적 응답 생성

```
def generate_positive_response(state: EmotionBotState) -> Dict[str, Any]:  
    responses = ["정말 좋은 소식이네요!", "기분이 좋으시군요!", "멋지네요!"]  
    return {"response": random.choice(responses)}
```

부정적 응답 생성

```
def generate_negative_response(state: EmotionBotState) -> Dict[str, Any]:  
    responses = [  
        "힘든 시간이지군요. 괜찮아요.",  
        "마음이 아프시겠어요.",  
        "더 좋은 날이 올 거예요.",  
    ]  
    return {"response": random.choice(responses)}
```

중립적 응답 생성

```
def generate_neutral_response(state: EmotionBotState) -> Dict[str, Any]:  
    responses = [  
        "감사해요! 더 자세히 말씀해주세요.",  
        "이해했어요. 다른 도움이 필요하시면 말씀하세요!",  
        "흥미로운 주제네요!",  
    ]  
    return {"response": random.choice(responses)}
```

실습 22: 감정 분석 챗봇에 적용해 보기

④ 조건부 라우팅 함수 - 감정 분석 결과에 따라 다음 노드 결정

```
def route_by_emotion(
    state: EmotionBotState,
) -> Literal["positive_response", "negative_response", "neutral_response"]:
    emotion = state.emotion
    print(f"라우팅: {emotion}")

    if emotion == "positive":
        return "positive_response"
    elif emotion == "negative":
        return "negative_response"
    else:
        return "neutral_response"
```

실습 22: 감정 분석 챗봇에 적용해 보기

⑤ 그래프 생성 함수 - 전체 워크플로우 구성

```
def create_emotion_bot_graph():  
    workflow = StateGraph(EmotionBotState)
```

⑥ 노드 추가 - 각 처리 단계를 그래프에 등록

```
    workflow.add_node("analyze_emotion", analyze_emotion)  
    workflow.add_node("positive_response", generate_positive_response)  
    workflow.add_node("negative_response", generate_negative_response)  
    workflow.add_node("neutral_response", generate_neutral_response)
```

⑦ 시작 엣지 설정 - 워크플로우의 진입점 정의

```
    workflow.add_edge(START, "analyze_emotion")
```

⑧ 조건부 엣지 설정 - 동적 라우팅 구현

```
    workflow.add_conditional_edges(  
        "analyze_emotion",  
        route_by_emotion,  
        {  
            "positive_response": "positive_response",  
            "negative_response": "negative_response",  
            "neutral_response": "neutral_response",  
        },  
    ),  
)
```

⑨ 종료 엣지 설정 - 각 응답 노드에서 워크플로우 종료

```
    workflow.add_edge("positive_response", END)  
    workflow.add_edge("negative_response", END)  
    workflow.add_edge("neutral_response", END)
```

```
    return workflow.compile()
```

StateGraph(EmotionBotState) 기반

노드 등록:

START → analyze_emotion
analyze_emotion →
(positive/negative/neutral)_response
(각 응답 노드) → END

실습 22: 감정 분석 챗봇에 적용해 보기

```
def main():
    print("=== 감정 분석 챗봇 테스트 ===\n")
    app = create_emotion_bot_graph()

    test_cases = [
        "오늘 정말 기분이 좋아요!",
        "너무 슬프고 힘들어요...",
        "날씨가 어떨까요?",
    ]

    for i, message in enumerate(test_cases, 1):
        print(f"테스트 {i}: '{message}'")
        state = EmotionBotState(user_message=message)
        result = app.invoke(state)
        print(f"응답: {result['response']}\n")

    # 그래프 시각화
    mermaid_png = app.get_graph().draw_mermaid_png()
    with open("./conditional_routing.png", "wb") as f:
        f.write(mermaid_png)

if __name__ == "__main__":
    main()
```

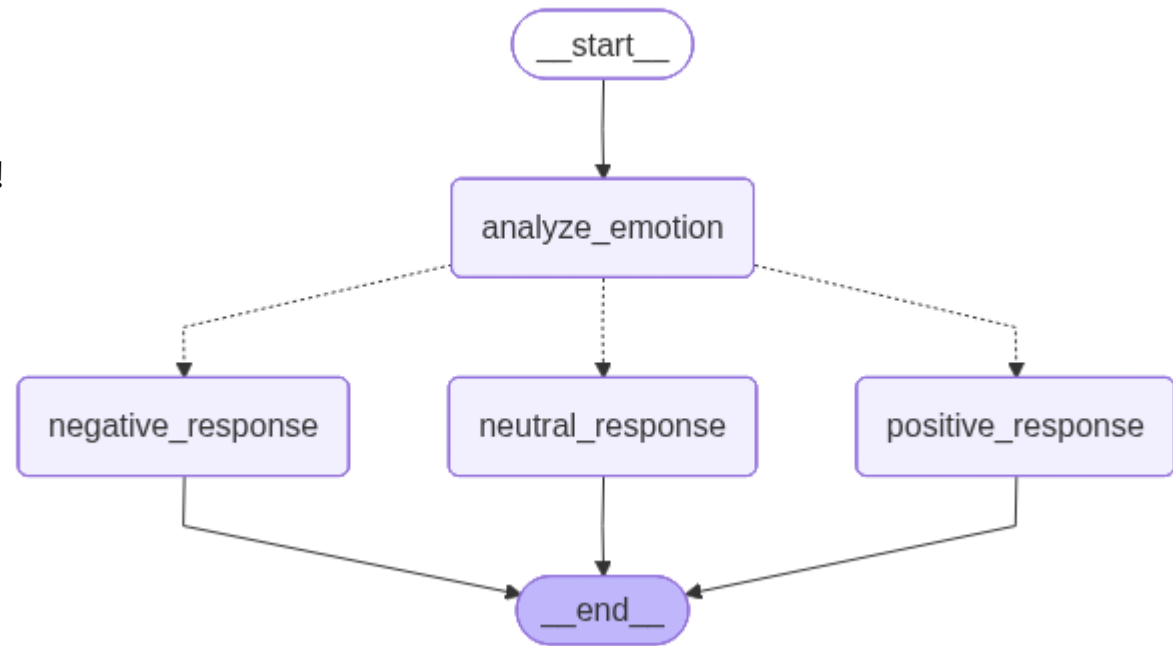
실습 22: 감정 분석 챗봇에 적용해 보기

```
(py3_10_langchain) C:\DEV\SamsungElectronics_Gumi\src\AIAgent\chapter6\langgraph>python  
conditional_routing.py  
=== 감정 분석 챗봇 테스트 ===
```

테스트 1: '오늘 정말 기분이 좋아요!'
LLM 감정 분석 중: '오늘 정말 기분이 좋아요!'
LLM 감정 분석 결과: neutral
라우팅: neutral
응답: 이해했어요. 다른 도움이 필요하시면 말씀하세요!

테스트 2: '너무 슬프고 힘들어요...'
LLM 감정 분석 중: '너무 슬프고 힘들어요...'
LLM 감정 분석 결과: neutral
라우팅: neutral
응답: 흥미로운 주제네요!

테스트 3: '날씨가 어떨까요?'
LLM 감정 분석 중: '날씨가 어떨까요?'
LLM 감정 분석 결과: neutral
라우팅: neutral
응답: 감사해요! 더 자세히 말씀해주세요.



```
(py3_10_langchain) C:\DEV\SamsungElectronics_Gumi\src\AIAgent\chapter6\langgraph>
```

PRD: MCP 서버/클라이언트

구분	내용
목적 (Objective)	Model Context Protocol(MCP)의 기본 개념을 시연하기 위해 간단한 MCP 서버와 클라이언트 를 구현한다. 서버는 도구와 리소스를 제공하고, 클라이언트는 이를 호출하여 결과를 확인한다.
기능 요구사항 (Functional Requirements)	<p>서버(simple_mcp_server.py)</p> <ul style="list-style-type: none"> ① hello 도구: 이름을 받아 간단한 인사말 반환 ② get_prompt 도구: 미리 정의된 프롬프트 템플릿 제공 (일반/코드리뷰/번역/요약) ③ simple://info 리소스: 서버 정보 문자열 제공 ④ 서버 실행: streamable-http 전송 방식으로 실행 <p>클라이언트(simple_mcp_client.py)</p> <ul style="list-style-type: none"> ① 서버 접속(http://localhost:8000/mcp) ② 사용 가능한 도구 목록 조회 (list_tools) ③ hello 도구 호출 (기본/이름 지정) ④ get_prompt 도구 호출 (code_review 유형) ⑤ simple://info 리소스 읽기
비기능 요구사항 (Non-Functional Requirements)	<ul style="list-style-type: none"> - Python + FastMCP 환경 필요 - 서버는 streamable-http 방식 지원 - 클라이언트는 비동기(async/await) 실행
시나리오 (Scenarios)	<ul style="list-style-type: none"> - 서버 실행 후 클라이언트가 연결 - list_tools 호출 시 ["hello", "get_prompt"] 반환 - hello 호출 시 "안녕하세요, 승굴님!" 응답 - get_prompt("code_review") 호출 시 코드 리뷰용 프롬프트 반환 - read_resource("simple://info") 호출 시 서버 정보 반환
향후 개선 항목 (Future Improvements)	<ul style="list-style-type: none"> - 더 많은 도구/리소스 추가 (예: 번역기, 뉴스 검색 등) - 인증/권한 관리 추가 - 에러 핸들링 및 로깅 강화 - 클라이언트 UI/웹 대시보드 제공

simple_mcp_server.py

```
from mcp.server.fastmcp import FastMCP
```

```
# ① FastMCP 인스턴스를 생성
```

```
mcp = FastMCP("Simple MCP Server")
```

```
@mcp.tool() # ② 도구를 정의합니다.
```

```
def hello(name: str = "World") -> str:
    """간단한 인사말을 반환하는 도구"""
    return f"안녕하세요, {name}님!"
```

```
@mcp.tool()
```

```
def get_prompt(prompt_type: str = "general") -> str:
```

```
    """사전 정의된 프롬프트를 반환하는 도구"""
```

```
    prompts = {
```

```
        "general": "당신은 도움이 되는 AI 어시스턴트입니다. 사용자의 질문에 정확하고 친절하게 답변해주세요.",
```

```
        "code_review": "다음 코드를 검토하고 개선점을 제안해주세요. 코드의 가독성, 성능, 보안 측면을 고려해주세요.",
```

```
        "translate": "다음 텍스트를 자연스러운 한국어로 번역해주세요.",
```

```
        "summarize": "다음 내용을 핵심 포인트 중심으로 간결하게 요약해주세요.",
```

```
    }
```

```
    return prompts.get(prompt_type, prompts["general"])
```


실습 23: MCP 서버 만들기

```
@mcp.resource("simple://info") # ③ 리소스를 정의합니다.  
def get_server_info() -> str:  
    """서버 정보를 제공하는 리소스"""  
    return ""  
    Simple MCP Server 정보  
    =====
```

이 서버는 MCP(Model Context Protocol)의 기본 기능을 시연하는 간단한 예제입니다.

제공하는 도구:

- hello: 인사말 생성
- get_prompt: 프롬프트 템플릿 제공

제공하는 리소스:

- simple://info: 서버 정보
- ```
"""
```

```
if __name__ == "__main__":
 """서버를 실행합니다."""
 # ④ 서버를 실행합니다.
 mcp.run(transport="streamable-http")
```

### simple\_mcp\_client.py

```
import asyncio
from fastmcp import Client

async def main():
 """MCP 클라이언트를 사용하여 서버와 통신합니다."""

 client = Client("http://localhost:8000/mcp")
 print("MCP 클라이언트를 생성하고 서버에 연결합니다.\n")

 try:
 # ① async with 블록을 사용하여 클라이언트 세션을 관리합니다.
 async with client:
 print("--- 사용 가능한 도구 목록 ---")
 tools = await client.list_tools()
 print([tool.name for tool in tools])
 print("")

 # ② 'hello' 도구 테스트
 print("--- 'hello' 도구 테스트 ---")
 response_default = await client.call_tool("hello")
 # call_tool은 콘텐츠 객체의 리스트를 반환합니다.
 print(f"기본 호출: {response_default[0].text}")
```

## 실습 23: MCP 서버 만들기

```
response_custom = await client.call_tool("hello", {"name": "승굴"})
print(f"이름 지정 호출: {response_custom[0].text}\n")
```

```
③ 'get_prompt' 도구 테스트
print("--- 'get_prompt' 도구 테스트 ---")
prompt_code = await client.call_tool(
 "get_prompt", {"prompt_type": "code_review"}
)
print(f"'code_review' 프롬프트:\n{prompt_code[0].text}\n")
```

```
④ 'simple://info' 리소스 테스트
print("--- 'simple://info' 리소스 테스트 ---")
resource_content = await client.read_resource("simple://info")
read_resource도 콘텐츠 객체의 리스트를 반환합니다.
print(f"리소스 내용:\n{resource_content[0].text}")
```

```
except Exception as e:
 print(f"오류가 발생했습니다: {e}")
```

```
if __name__ == "__main__":
 asyncio.run(main())
```

## Node.js® 다운로드

<https://nodejs.org/ko/download>

Node.js® 다운로드

Node.js® v22.19.0 (LTS) 를 Windows 환경에서 Docker 방식으로 npm 를(을) 사용해 설치하세요.

정보 Want new features sooner? Get the **latest Node.js version** instead and try the latest improvements!

```
1 # Docker는 각 운영 체제별로 설치 지침을 제공합니다.
2 # 공식 문서는 https://docker.com/get-started/에서 확인하세요.
3
4 # Node.js Docker 이미지를 풀(Pull)하세요:
5 docker pull node:22-alpine
6
7 # Node.js 컨테이너를 생성하고 셸 세션을 시작하세요:
8 docker run -it --rm --entrypoint sh node:22-alpine
9
10 # Node.js 버전 확인:
11 node -v # "v22.19.0"가 출력되어야 합니다.
12
13 npm 버전 확인:
14 npm -v # 10.9.3가 출력되어야 합니다.
```

PowerShell [클립보드에 복사](#)

Docker는 컨테이너화 플랫폼입니다. 문제가 발생하면 [Docker's 웹사이트](#)를 방문하세요.

또는 x64 아키텍처가 실행 중인 Windows 환경에서 미리 빌드된 Node.js®를 다운로드하세요.

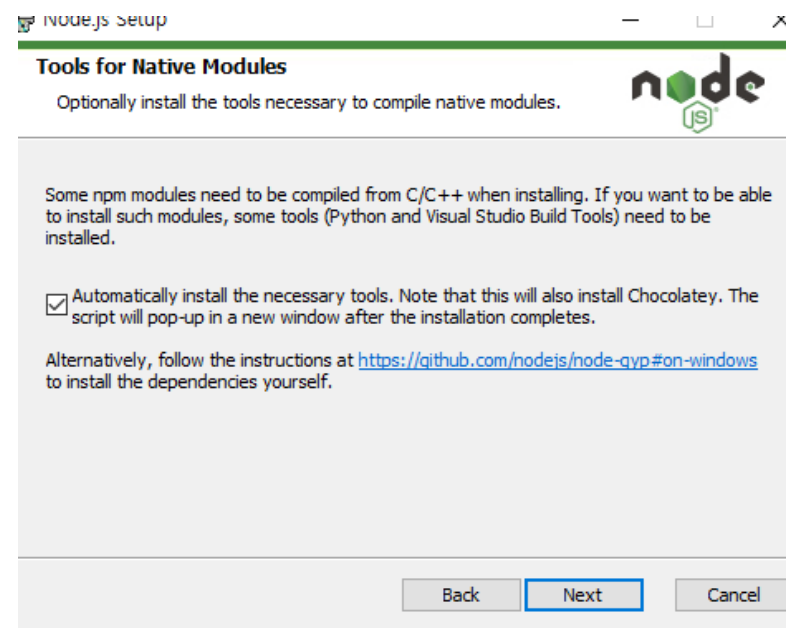
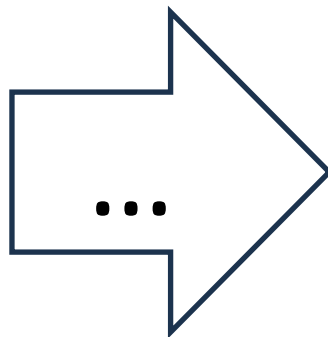
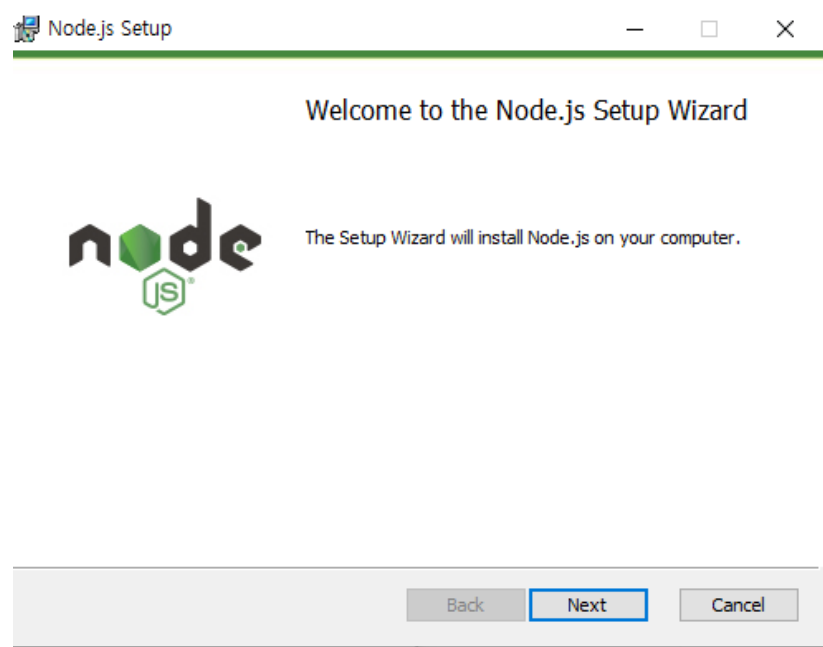
[Windows 설치 프로그램 \(.msi\)](#) [Standalone Binary \(.zip\)](#)

오늘 (1)

node-v22.19.0-x64.msi

이제 시작 (2)

## Node.js® 설치



### Node.js® 설치 확인

```
node -v
v22.19.0
```

```
npm -v
10.9.3
```

## TensorFlow 2 가상환경

```
c:\DEV>cd envs
```

```
c:\DEV\envs>python -m venv mcp-env
```

```
c:\DEV\envs>cd mcp-env
```

```
c:\DEV\envs\ mcp-env>Scripts\activate
```

```
(
```

```
(mcp-env) c:\DEV\envs\ mcp-env>dir/w/p
```

C 드라이브의 볼륨에는 이름이 없습니다.

볼륨 일련 번호: BED0-C858

```
c:\DEV\envs\ mcp-env 디렉터리
```

```
[.] [..] [Include] [Lib] pyvenv.cfg [Scripts]
```

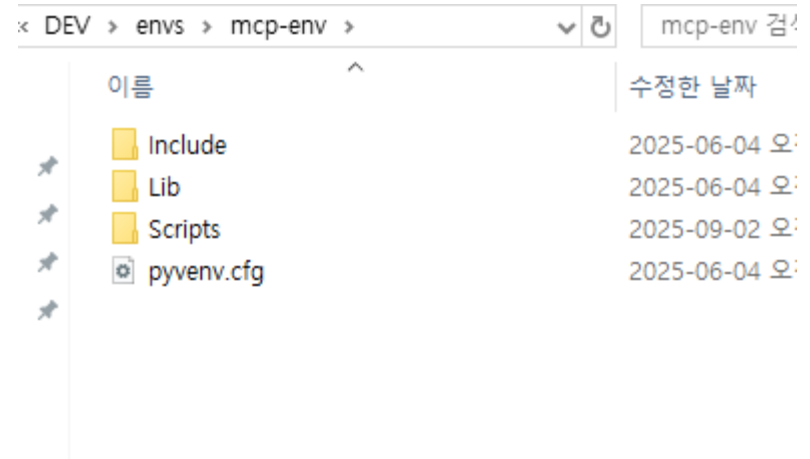
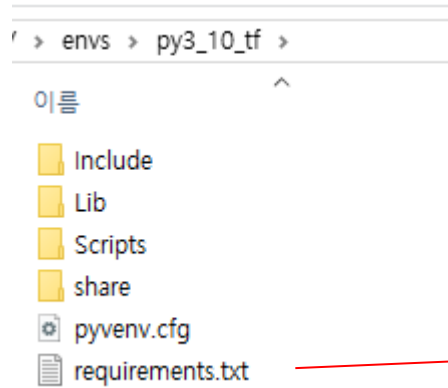
1개 파일 118 바이트

5개 디렉터리 56,743,809,024 바이트 남음

```
(mcp-env) c:\DEV\envs\ mcp-env>
```

# 실습 23: MCP 서버 만들기

## requirements.txt 복사



( mcp-env) c:\DEV\ mcp-env>dir/w/p

C 드라이브의 볼륨에는 이름이 없습니다.  
볼륨 일련 번호: BEDo-C858

c:\DEV\mcp-env 디렉터리

```
[.] [..] [Include] [Lib] pyenv.cfg requirements.txt
[Scripts]
 2개 파일 953 바이트
 5개 디렉터리 68,334,325,760 바이트 남음
```

( mcp-env) c:\DEV\ mcp-env>



## requirements.txt 복사

**( mcp-env) c:\DEV\ mcp-env>pip install -r requirements.txt**

```
Collecting asttokens==2.4.1
 Using cached asttokens-2.4.1-py2.py3-none-any.whl (27 kB)
Collecting colorama==0.4.6
 Using cached colorama-0.4.6-py2.py3-none-any.whl (25 kB)
Collecting comm==0.2.2
 Using cached comm-0.2.2-py3-none-any.whl (7.2 kB)
Collecting contourpy==1.2.1
 Downloading contourpy-1.2.1-cp310-cp310-win_amd64.whl (187 kB)
----- 187.5/187.5 kB 5.7 MB/s eta 0:00:00
Collecting cycler==0.12.1
 Downloading cycler-0.12.1-py3-none-any.whl (8.3 kB)
```

## 쥬피터 노트북 내보내기

```
pip install ipykernel
python.exe -m pip install --upgrade pip
```

```
(mcp-env) c:\DEV\ mcp-env>python -m ipykernel install --user --name mcp-env --display-
name " mcp-env"
Installed kernelspec mcp-env in C:\Users\k8s\AppData\Roaming\jupyter\kernels\ mcp-env
```

```
(mcp-env) c:\DEV\envs\ mcp-env>deactivate
c:\DEV\envs\ mcp-env>cd ..
```

```
c:\DEV\envs>cd ..
```

```
c:\DEV>jupyter notebook
```

## 라이브러리

```
import sys
print("python 버전 : {}".format(sys.version))
import numpy as np
print("numpy 버전 : {}".format(np.__version__))
import pandas as pd
print("pandas 버전 : {}".format(pd.__version__))
import matplotlib
print("matplotlib 버전 : {}".format(matplotlib.__version__))
import scipy as sp
print("scipy 버전 : {}".format(sp.__version__))
import IPython
print("IPython 버전 : {}".format(IPython.__version__))
import sklearn
print("sklearn : {}".format(sklearn.__version__))
```

---

```
python 버전 : 3.10.11 (tags/v3.10.11:7d4cc5a, Apr 5 2023, 00:38:17) [MSC v.1929 64 bit (AMD64)]
numpy 버전 : 2.0.0
pandas 버전 : 2.2.2
matplotlib 버전 : 3.9.0
scipy 버전 : 1.13.1
IPython 버전 : 8.25.0
sklearn : 1.5.0
```

## MCP 사전 준비

(c) Microsoft Corporation. All rights reserved.

```
C:\Users\k8s>cd c:\dev
```

```
c:\DEV>dir/w/p
```

C 드라이브의 볼륨에는 이름이 없습니다.  
볼륨 일련 번호: BED0-C858

c:\DEV 디렉터리

|                                |                                  |                                    |
|--------------------------------|----------------------------------|------------------------------------|
| [.]                            | [..]                             | [.ipynb_checkpoints]               |
| [envs]                         | [Erudite_demo]                   | [IntelliJ_pro]                     |
| [jdk-24.0.1]                   | [ollama_pro]                     | openjdk-24.0.1_windows-x64_bin.zip |
| [PycharmProjects]              | PycharmProjects.zip              | [r-workspaces]                     |
| [SamsungDisplay]               | SamsungDisplay-main.zip          | [SamsungElectronics_Gumi]          |
| [sqlite-tools-win-x64-3490200] | sqlite-tools-win-x64-3490200.zip | [Tools]                            |
| [web_app_project]              |                                  |                                    |
| 4개 파일                          | 443,259,906 바이트                  |                                    |
| 15개 디렉터리                       | 9,454,661,632 바이트 남음             |                                    |

```
c:\DEV>cd envs
```

## MCP 사전 준비

```
c:\DEV\envs>dir/w/p
```

c 드라이브의 볼륨에는 이름이 없습니다.

볼륨 일련 번호: BED0-C858

```
c:\DEV\envs 디렉터리
```

|                 |             |                      |                |                    |                 |
|-----------------|-------------|----------------------|----------------|--------------------|-----------------|
| [.]             | [..]        | [mcp-env]            | [py3_10_basic] | [py3_10_langchain] | [py3_10_ollama] |
| [py3_10_openai] | [py3_10_pt] | [py3_10_tf]          |                |                    |                 |
|                 | 0개 파일       | 0 바이트                |                |                    |                 |
|                 | 10개 디렉터리    | 9,454,661,632 바이트 남음 |                |                    |                 |

```
c:\DEV\envs>cd mcp-env
```

```
c:\DEV\envs\ mcp-env >Scripts\activate
```

```
(mcp-env) c:\DEV\envs\ mcp-env >
```

## MCP 사전 준비

- FastMCP 라이브러리 설치

```
(mcp-env) c:\DEV\envs\mcp-env>pip install "mcp[cli]"==1.12.2
Collecting mcp==1.12.2 (from mcp[cli]==1.12.2)
 Downloading mcp-1.12.2-py3-none-any.whl.metadata (60 kB)
Requirement already satisfied: anyio>=4.5 in c:\dev\envs\mcp-env\lib\site-packages (from mcp==1.12.2->mcp[cli]==1.12.2) (4.9.0)
...
```

```
[notice] A new release of pip is available: 25.1.1 -> 25.2
[notice] To update, run: python.exe -m pip install --upgrade pip
```

```
(mcp-env) c:\DEV\envs\mcp-env>pip show mcp
Name: mcp
Version: 1.12.2
Summary: Model Context Protocol SDK
Home-page: https://modelcontextprotocol.io
Author: Anthropic, PBC.
Author-email:
License: MIT
Location: c:\dev\envs\mcp-env\lib\site-packages
Requires: anyio, httpx, httpx-sse, jsonschema, pydantic, pydantic-settings, python-multipart, pywin32, sse-starlette, starlette, uvicorn
Required-by: fastmcp
```

```
(mcp-env) c:\DEV\envs\mcp-env>
```

## 실습 23: MCP 서버 만들기

```
(mcp-env) C:\DEV\SamsungElectronics_Gumi>cd src
```

```
(mcp-env) C:\DEV\SamsungElectronics_Gumi\src>cd AIAgent
```

```
(mcp-env) C:\DEV\SamsungElectronics_Gumi\src\AIAgent>cd chapter7
```

```
(mcp-env) C:\DEV\SamsungElectronics_Gumi\src\AIAgent\chapter7>cd mcp
```

```
(mcp-env) C:\DEV\SamsungElectronics_Gumi\src\AIAgent\chapter7\mcp>
```

## 서버

```
(mcp-env) c:\DEV\envs\mcp-env>node -v
v22.19.0
```

```
(mcp-env) c:\DEV\envs\mcp-env>npm -v
10.9.3
```

```
(mcp-env) c:\DEV\envs\mcp-env>cd C:\DEV\SamsungElectronics_Gumi\src\AIAgent\chapter7\mcp
```

```
(mcp-env) C:\DEV\SamsungElectronics_Gumi\src\AIAgent\chapter7\mcp>python simple_mcp_server.py
[32mINFO[0m: Started server process [[36m8020[0m]
[32mINFO[0m: Waiting for application startup.
[09/02/25 15:11:45] INFO StreamableHTTP session manager started
streamable_http_manager.py:110
[32mINFO[0m: Application startup complete.
[32mINFO[0m: Uvicorn running on [1mhttp://127.0.0.1:8000[0m (Press CTRL+C to quit)
[32mINFO[0m: 127.0.0.1:49877 - "[1mGET / HTTP/1.1[0m" [31m404 Not Found[0m
[32mINFO[0m: 127.0.0.1:49877 - "[1mGET /favicon.ico HTTP/1.1[0m" [31m404 Not Found[0m
```



## 클라이언트

```
(mcp-env) C:\DEV\SamsungElectronics_Gumi\src\AIAgent\chapter7\mcp>python simple_mcp_client.py
```

MCP 클라이언트를 생성하고 서버에 연결합니다.

```
--- 사용 가능한 도구 목록 ---
```

```
['hello', 'get_prompt']
```

```
--- 'hello' 도구 테스트 ---
```

기본 호출: 안녕하세요, World님!

이름 지정 호출: 안녕하세요, 승굴님!

```
--- 'get_prompt' 도구 테스트 ---
```

'code\_review' 프롬프트:

다음 코드를 검토하고 개선점을 제안해주세요. 코드의 가독성, 성능, 보안 측면을 고려해주세요.

```
--- 'simple://info' 리소스 테스트 ---
```

리소스 내용:

Simple MCP Server 정보

=====

이 서버는 MCP(Model Context Protocol)의 기본 기능을 시연하는 간단한 예제입니다.

제공하는 도구:

- hello: 인사말 생성
- get\_prompt: 프롬프트 템플릿 제공

제공하는 리소스:

- simple://info: 서버 정보

```
(mcp-env) C:\DEV\SamsungElectronics_Gumi\src\AIAgent\chapter7\mcp>
```

MCP 서버를 테스트하는 클라이언트로온 엔트로픽에서 제작한 instpector와 postman이 있습니다. 여기에서는 포스트맨을 사용해 봅니다.

<https://www.postman.com/downloads/>

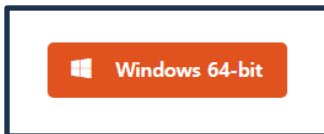


### Download Postman

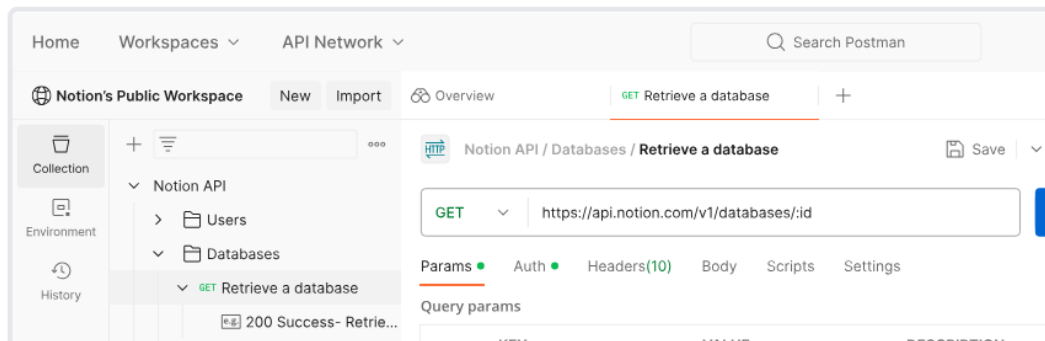
Download the app to get started using the Postman API Platform today. Or, if you prefer a browser experience, you can try the web version of Postman.

#### The Postman app

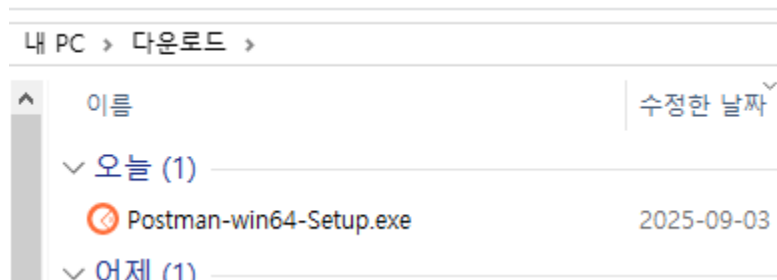
Download the app to get started with the Postman API Platform.



By downloading and using Postman, I agree to the [Privacy Policy](#)



MCP 서버를 테스트하는 클라이언트로엔 엔트로픽에서 제작한 **instpector**와 **postman**이 있습니다. 여기에서는 포스트맨을 사용해 봅시다.



Working with APIs  
simplified with Postman

Your work email makes it easy for you to  
collaborate with your teammates.

Create Free Account

Already have an account? [Log In](#)

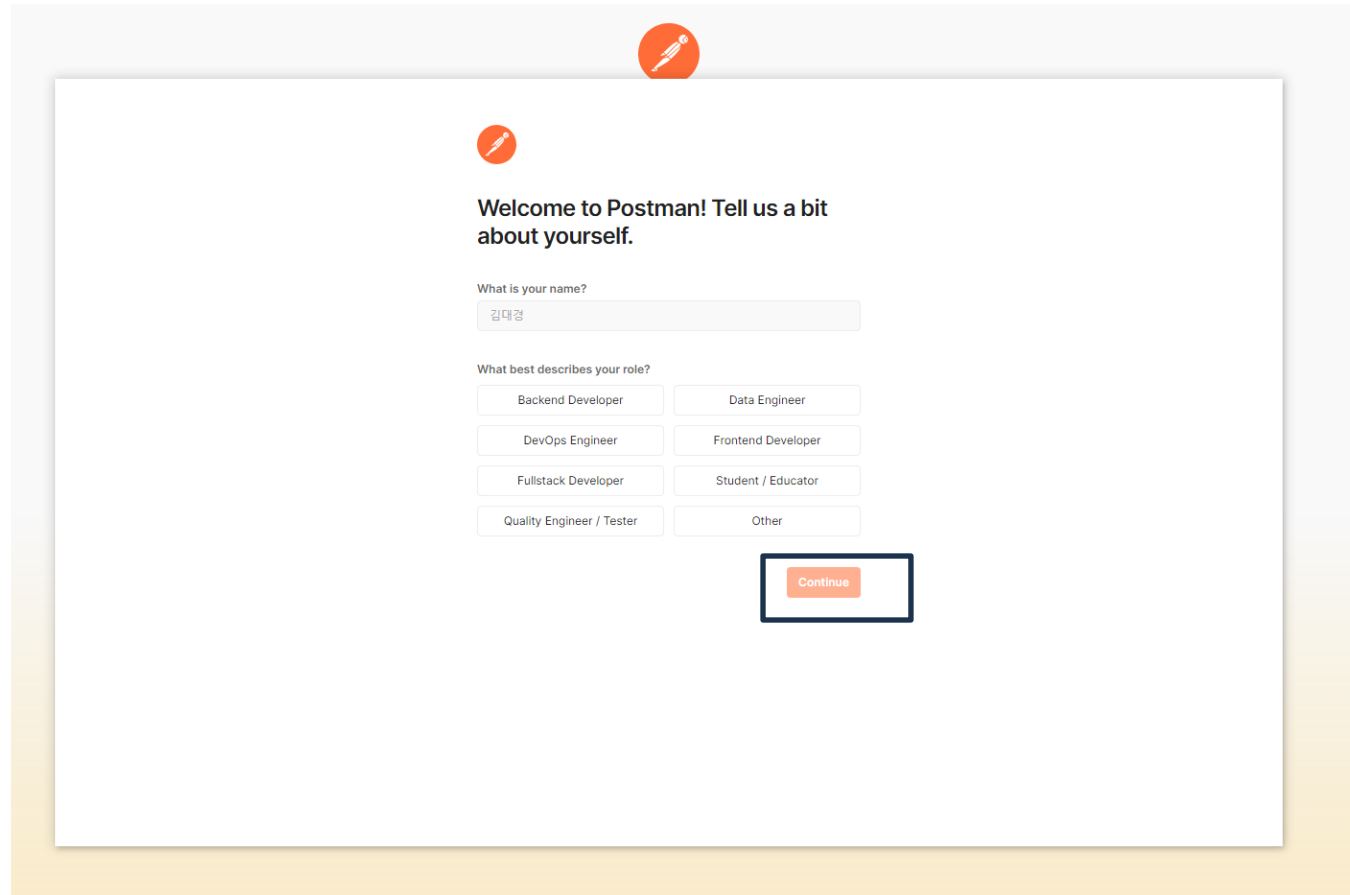
Continue with Google

Continue with Github

Single Sign On (SSO)

Continue without an account

MCP 서버를 테스트하는 클라이언트로온 엔트로픽에서 제작한 instpector와 postman이 있습니다. 여기에서는 포스트맨을 사용해 봅니다.

The image shows the Postman application's welcome screen. At the top center is the Postman logo, an orange circle with a white pen icon. Below the logo, the text "Welcome to Postman! Tell us a bit about yourself." is displayed. Underneath this text is a form with two sections. The first section is titled "What is your name?" and contains a text input field with the Korean text "김대경" (Kim Daekyeong) entered. The second section is titled "What best describes your role?" and contains eight buttons arranged in two columns. The buttons are: "Backend Developer", "Data Engineer", "DevOps Engineer", "Frontend Developer", "Fullstack Developer", "Student / Educator", "Quality Engineer / Tester", and "Other". At the bottom right of the form is a blue "Continue" button, which is highlighted by a blue rectangular border.

MCP 서버를 테스트하는 클라이언트로온 엔트로픽에서 제작한 instpector와 postman이 있습니다. 여기에서는 포스트맨을 사용해 봅니다.



[Continue with Free Plan →](#)

← You're on the Free plan. Upgrade to match your API needs.

### RECOMMENDED

#### Basic

If you need centralized API access, easy sharing, and local testing.

**\$14** per seat/month, billed annually

[Select Plan](#)

[Try Free for 30 days](#)

No credit card required for the trial.

#### Key features

- ✓ Unlimited teammates
- ✓ Unlimited Internal and Public workspaces
- ✓ 10,000 Mock Server requests

#### Professional

If you need secure role management and want to connect different teams.

**\$29** per seat/month, billed annually

[Select Plan](#)

[Try Free for 30 days](#)

No credit card required for the trial.

#### Key features

- ✓ Everything in Basic
- ✓ RBAC (Role-based access control)
- ✓ Internal workspaces (private)
- ✓ Single & Multi-Partner workspaces

#### Enterprise

If you need organization-wide security, compliance, planning, and roles.

**\$49** per seat/month, billed annually

[Select Plan](#)

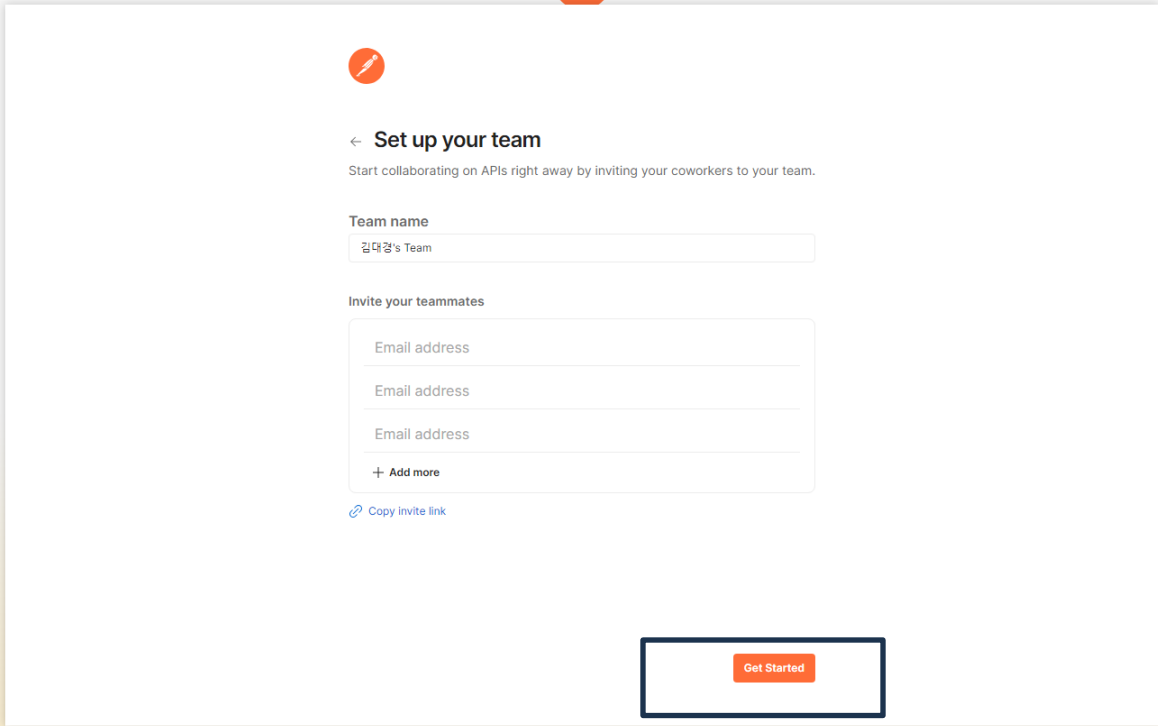
[Contact Sales](#)


#### Key features

- ✓ Everything in Professional
- ✓ SSO, SCIM, SAML
- ✓ Advanced RBAC (Role-based access control)
- ✓ Advanced Reporting & Analytics

[Learn more about plans and add-ons ↗](#)

MCP 서버를 테스트하는 클라이언트로 은 엔트로픽에서 제작한 instpector와 postman이 있습니다. 여기에서는 포스트맨을 사용해 봅니다.





← Set up your team

Start collaborating on APIs right away by inviting your coworkers to your team.

Team name

김대강's Team

Invite your teammates

Email address

Email address

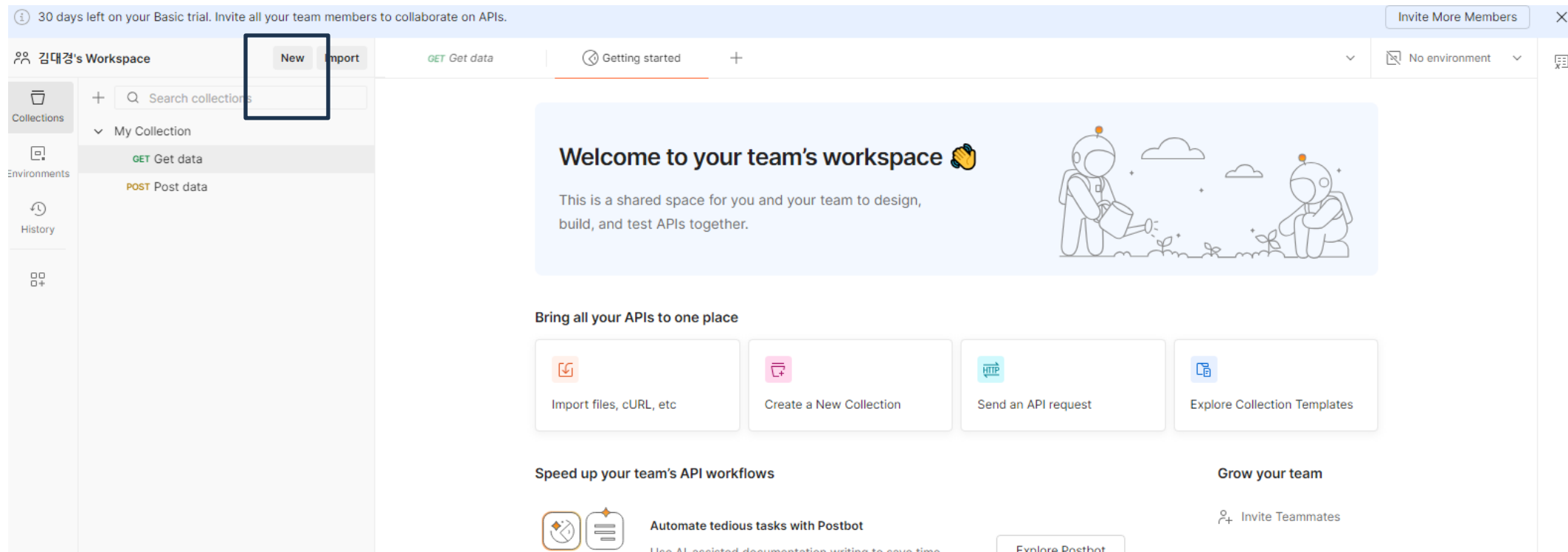
Email address

+ Add more

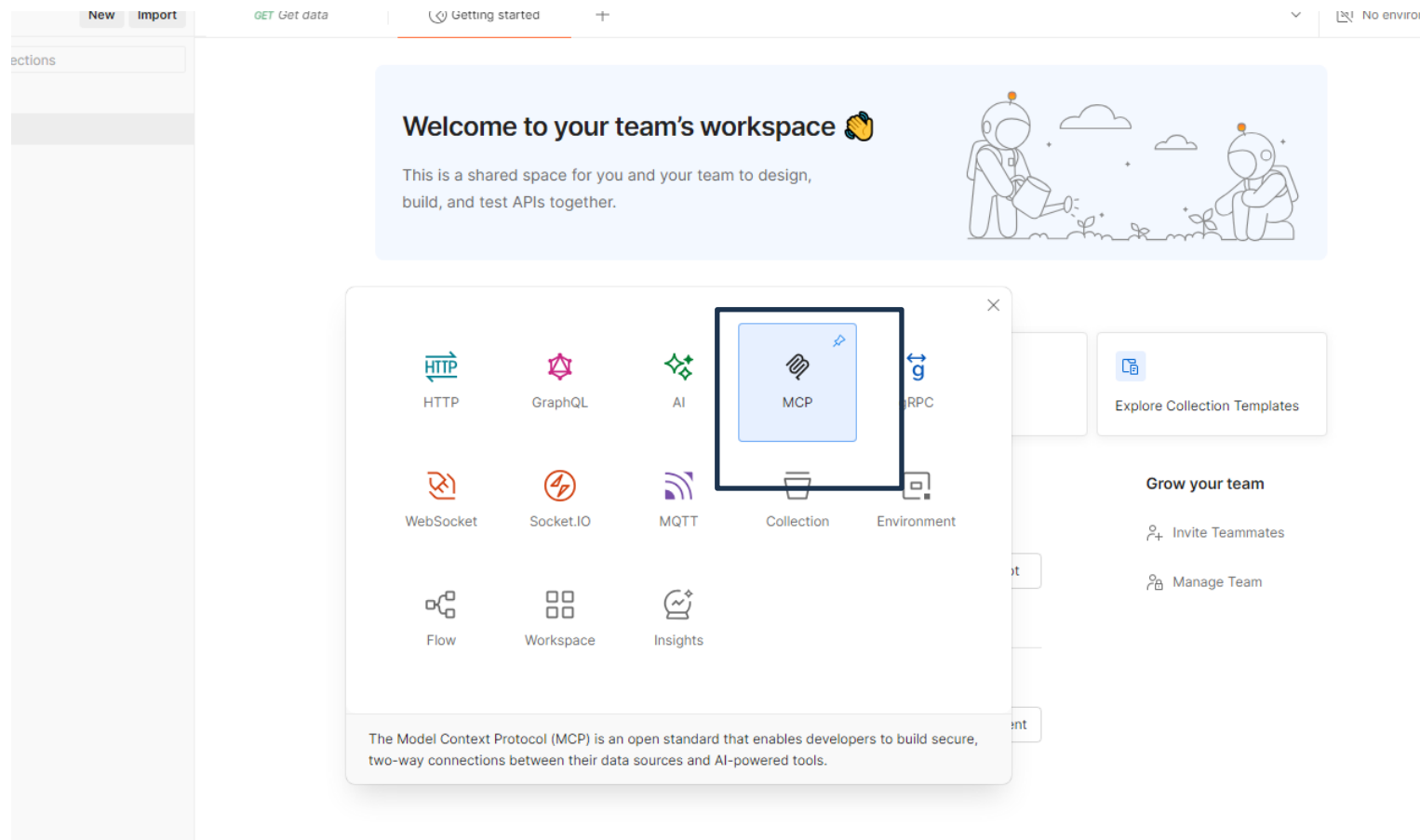
[Copy invite link](#)

Get Started

## New 클릭

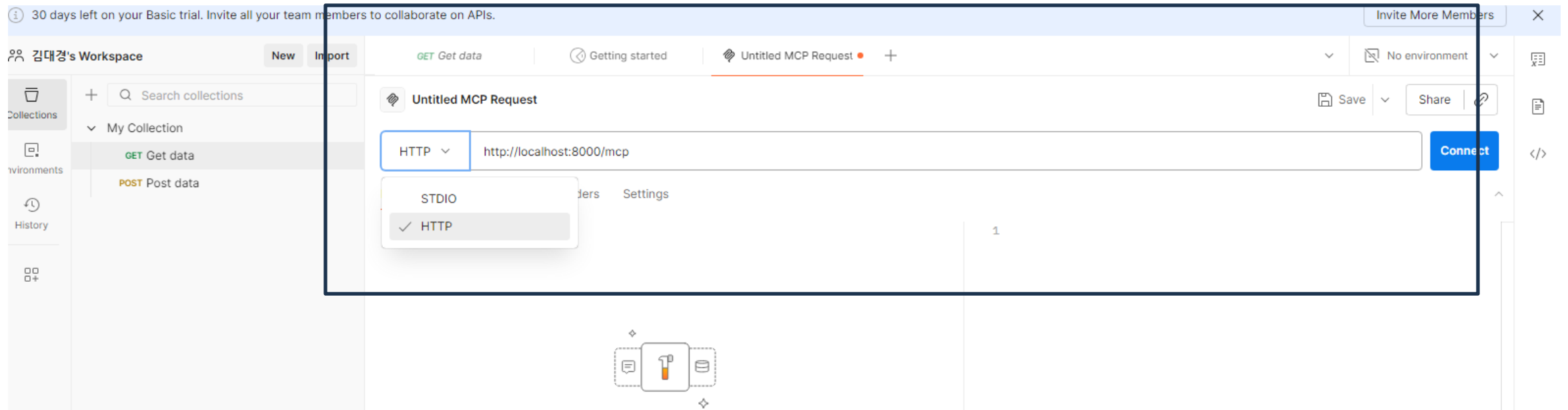


## MCP 클릭





## STDIO를 HTTP로 변경후, http://localhost:8000/mcp 입력



# 실습 23: MCP 서버 만들기

다음 화면으로 바뀜.

30 days left on your Basic trial. Invite all your team members to collaborate on APIs.

Invite More Members

김대경's Workspace

New Import

GET Get data

Getting started

Untitled MCP Request

Search collections

My Collection

GET Get data

POST Post data

environments

History

Untitled MCP Request

Save Share

HTTP http://localhost:8000/mcp

Run

Message

Authorization

Headers

Settings

Tools

Prompts

Resources

1. hello

간단한 인사말을 반환하는 도구

2. get\_prompt

사전 정의된 프롬프트를 반환하는 도구

## 실습 23: MCP 서버 만들기

hello 부분을 클릭 후, “대경! 즐거운 AI 교육이 되셨나요?” 입력. 그리고 Run 버튼을 눌러 실행합니다.

The screenshot shows the MCP client interface. On the left, the 'Tools' tab is active, displaying a list of tools. The first tool, 'hello', is selected and highlighted with a blue box. Below the tool name, there is a description '간단한 인사말을 반환하는 도구' and a parameter 'name : string'. A text input field contains the text '대경! 즐거운 AI 교육이 되셨나요?'. To the right of the tool list, the JSON-RPC request is displayed in a code editor. The request is a 'tools/call' message with parameters 'name' and 'arguments'. The 'arguments' array contains the text '대경! 즐거운 AI 교육이 되셨나요?'. A blue arrow points from the 'Run' button in the top right corner to the 'hello' tool. The 'Run' button is also highlighted with a blue box.

30 days left on your Basic trial. Invite all your team members to collaborate on APIs. [Invite More Members](#)

김대경's Workspace [New](#) [Import](#)

[Collections](#) [+ Search collections](#)

[My Collection](#)

[GET Get data](#)

[POST Post data](#)

[Environments](#)

[History](#)

[Tools](#) [Prompts](#) [Resources](#)

[hello](#)  
간단한 인사말을 반환하는 도구  
name : string  
대경! 즐거운 AI 교육이 되셨나요?

[2. get\\_prompt](#)  
사전 정의된 프롬프트를 반환하는 도구

HTTP [http://localhost:8000/mcp](#)

[Save](#) [Share](#) [Run](#) [</>](#)

```
1 {
2 "method": "tools/call",
3 "params": {
4 "name": "hello",
5 "arguments": {
6 "name": "대경! 즐거운 AI 교육이 되셨나요?"
7 }
8 }
9 }
```

## 결과 화면

The screenshot displays the MCP client interface with the following components:

- Header:** "30 days left on your Basic trial. Invite all your team members to collaborate on APIs." and an "Invite N" button.
- Workspace:** "김대경's Workspace" with "New" and "Import" buttons.
- Left Sidebar:** Includes "Collections" (with a search bar), "Environments" (showing "GET Get data" and "POST Post data"), and "History".
- Main Panel:** Titled "Untitled MCP Request", it shows the request details: "HTTP" method and "http://localhost:8000/mcp" URL. Below this are tabs for "Message", "Authorization", "Headers", and "Settings".
- Tools Section:** Lists available tools:
  - hello:** "간단한 인사말을 반환하는 도구" (Tool for returning simple greetings). It has a parameter "name: string" with a value of "대경! 즐거운 AI 교육이 되셨나요?".
  - get\_prompt:** "사전 정의된 프롬프트를 반환하는 도구" (Tool for returning predefined prompts).
- Response Section:** A table of messages:

| Message       | Time         |
|---------------|--------------|
| tools / hello | 09:45:30.499 |
| tools / hello | 09:45:30.452 |
| Connected     | 09:41:25.985 |
- JSON Preview:** A box showing the JSON response from the tool call:

```
{
 "method": "tools/call",
 "params": {
 "name": "hello",
 "arguments": {
 "name": "대경! 즐거운 AI 교육이 되셨나요?"
 }
 }
}
```

```
{
 "content": [
 {
 "type": "text",
 "text": "안녕하세요, 대경! 즐거운 AI 교육이 되셨나요?님!"
 }
],
 "structuredContent": {
 "result": "안녕하세요, 대경! 즐거운 AI 교육이 되셨나요?님!"
 },
 "isError": false
}
```

## 서버 쪽 변화

선택 명령 프롬프트 - python simple\_mcp\_server.py

```
mcp-env) C:\WDEV\SamsungElectronics_Gumi\src\AI\Agent\chapter7\mcp>python simple_mcp_server.py
[32mINFO+[0m: Started server process [+36m6740+[0m]
[32mINFO+[0m: Waiting for application startup.
09/03/25 09:23:29] INFO StreamableHTTP session manager started streamable_http_manager.py:110
[32mINFO+[0m: Application startup complete.
[32mINFO+[0m: Uvicorn running on [+1mhttp://127.0.0.1:8000+[0m (Press CTRL+C to quit)
09/03/25 09:40:26] INFO Created new transport with session ID: streamable_http_manager.py:233
ba586ccc7e3640d38e748721cdeb9fa3
[32mINFO+[0m: 127.0.0.1:55224 - "[+1mPOST /mcp HTTP/1.1+[0m" [+32m200 OK+[0m
[32mINFO+[0m: 127.0.0.1:55228 - "[+1mPOST /mcp HTTP/1.1+[0m" [+32m202 Accepted+[0m
[32mINFO+[0m: 127.0.0.1:55224 - "[+1mPOST /mcp HTTP/1.1+[0m" [+32m200 OK+[0m
09/03/25 09:41:42] INFO Processing request of type ListToolsRequest server.py:625
[32mINFO+[0m: 127.0.0.1:55229 - "[+1mGET /mcp HTTP/1.1+[0m" [+32m200 OK+[0m
[32mINFO+[0m: 127.0.0.1:55230 - "[+1mPOST /mcp HTTP/1.1+[0m" [+32m200 OK+[0m
[32mINFO+[0m: 127.0.0.1:55231 - "[+1mPOST /mcp HTTP/1.1+[0m" [+32m200 OK+[0m
[32mINFO+[0m: 127.0.0.1:55232 - "[+1mPOST /mcp HTTP/1.1+[0m" [+32m200 OK+[0m
INFO Processing request of type ListResourcesRequest server.py:625
INFO Processing request of type ListPromptsRequest server.py:625
INFO Processing request of type ListResourceTemplatesRequest server.py:625
[32mINFO+[0m: 127.0.0.1:55246 - "[+1mPOST /mcp HTTP/1.1+[0m" [+32m200 OK+[0m
09/03/25 09:45:30] INFO Processing request of type CallToolRequest server.py:625
```

Unit A

# 참고자료



## | 시각화 기초 연습

- 한글

```
import matplotlib.pyplot as plt
import matplotlib
matplotlib.rcParams['font.family'] = 'Malgun Gothic' # Windows
matplotlib.rcParams['font.family'] = 'AppleGothic' # Mac
matplotlib.rcParams['font.size'] = 15 # 글자 크기
matplotlib.rcParams['axes.unicode_minus'] = False # 한글 폰트 사용 시, 마이너스 글자가 깨지는
현상을 해결
```

## 문헌

1. <http://www.ncs.go.kr>
2. NELLDAL/JOHN LEWIS 지음, 조영석/김대경/박찬영/송창근 역, 단계별로 배우는 컴퓨터과학, 홍릉과학출판사, 2018
3. David A. Watt, Programming Language Syntax and Semantics, Prentice-Hall, 1991.
4. 머신러닝 실무 프로젝트, 아리가 미치아키, 나카야마 신타, 니시바야시 다카시 지음 | 심효섭 옮김 | 한빛미디어 | 2018년 06월
5. 김덕진, AI 2025 트렌드&활용백과, 스마트북스, 2024년 11월 29일
6. 구자룡, 챗GPT로 시작하는 데이터 리터러시, 마들렌북 · 2025년 02월 06일
7. 누구나 프로처럼, 생활 AI 테리엇, Bob Lee, 월 20달러로 고용하는 데이터 분석가 with 챗GPT, 한빛미디어 · 2024년 10월 29일
8. 박승균, 요즘 AI 에이전트 개발, LLM RAG ADK MCP LangChain A2A LangGraph, 골든래빗(주), 2025년 08월 20일
9. 기타 서적 및 웹 사이트 자료 다수 참조

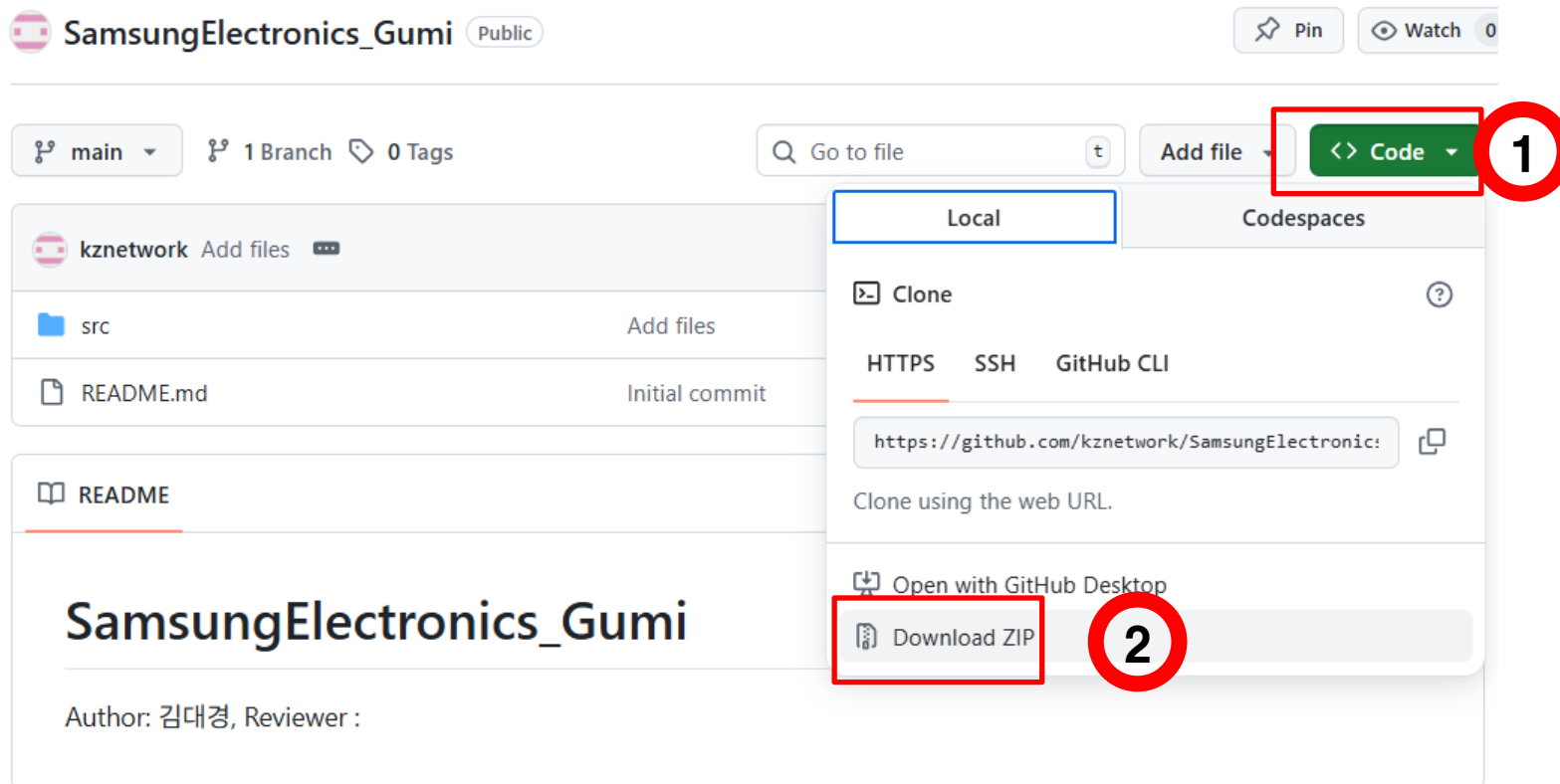


## 문헌

1. **Stanford University (HAI) — 2025 AI Index Report**
2. **McKinsey & Co. — The State of AI in 2024 (Global Survey)**
3. **Statista — Global AI Market Size Forecast**
4. **Grand View Research — Artificial Intelligence Market to 2030**
5. **IDC — Global AI Economic Impact**
6. **Gartner — Generative AI Spending Forecast**
7. **World Economic Forum — Future of Jobs Report 2025**
8. **Bain & Company — Healthcare AI Adoption Index (2024)**
9. **PYMNTS Intelligence — Is AI the Key to Banking's Next Era? (Sep 2024)**
10. **Iterable — 15+ Stats About Achieving ROI From AI Marketing**
11. **DemandSage — Midjourney Statistics 2025**
12. **Sequencer — Key Generative AI Statistics and Trends for 2025**
13. **[https://ko.wikipedia.org/wiki/%EC%83%9D%EC%84%B1%ED%98%95\\_%EC%9D%B8%EA%B3%B5%EC%A7%80%EB%8A%A5](https://ko.wikipedia.org/wiki/%EC%83%9D%EC%84%B1%ED%98%95_%EC%9D%B8%EA%B3%B5%EC%A7%80%EB%8A%A5)**
14. **<https://news.skhynix.co.kr/all-around-ai-1/>**

## 강의 소스 다운로드

1. [https://github.com/kznetwork/SamsungElectronics\\_Gumi](https://github.com/kznetwork/SamsungElectronics_Gumi) 에서 강의 소스를 다운받을 수 있으나, 사전 예고없이 변경될 수 있음.



# THANK YOU.

앞으로의 엔지니어는 단순한 '코더'나 '기계 조작자'가 아니라 뇌-기계 인터페이스를 통해 지식과 능력을 즉각 확장하는 존재(뉴로-인터페이스: Neuro Interface)가 될 수 있습니다.

- 🎯 목표 달성을 위한 여정이 시작됩니다.
- 🌟 궁금한 점이 있으시면 언제든지 문의해주세요!
- 🚀 함께 코더와 프롬프트 전문가로 성장해 나갑시다!

