

2부

API 기반 업무 자동화 실전

코드 모음

7장. API 기반 LLM 실행 환경 구축

LLM 호출과 데이터 처리를 위한 도우미 함수 구성

(174쪽)

LLM 호출 및 응답 정규화 함수

```
from langchain.chat_models import init_chat_model
from langchain.messages import SystemMessage, HumanMessage
from langchain_core.messages import BaseMessage
from typing import Any

DEFAULT_MODEL = 'google_genai:gemini-flash-latest'

def get_llm_response(
    instruction: str | None = None,
    user_message: str | None = None,
    model: str = DEFAULT_MODEL,
    *,
    messages: list[BaseMessage] | None = None,
    **kwargs: Any
) -> str:
    """
    LangChain을 사용해 LLM을 한 번 호출하고 텍스트 응답을 반환하는 도우미 함수

    사용 방식은 세 가지다.
    1) user_message만 전달 (가장 기본)
    2) instruction을 함께 전달 (시스템 지침 추가)
    3) messages를 직접 전달 (메시지 구조를 직접 제어)

    model 파라미터를 바꾸면 동일한 코드로 다른 LLM을 사용할 수 있다.
    """

    # 모델의 인스턴스를 생성한다.
    llm = init_chat_model(model, **kwargs)
```

```

# 메시지를 구성한다.
if messages is None:
    messages = []
    if instruction:
        messages.append(SystemMessage(content=instruction))
    messages.append(HumanMessage(content=user_message))

# 모델을 호출한다.
response = llm.invoke(messages)

# 모델의 응답을 정규화한다. (모델별 응답 형식을 일관된 구조로 변환)
content = response.content
if isinstance(content, list):
    return ".join(
        block.get('text', '')
        for block in content
        if isinstance(block, dict)
    )"

return content or "

```

(177쪽)

깃허브 저장소에서 텍스트 파일을 불러오는 함수

```

import urllib.request, json, fnmatch

def load_text_files_from_github(pattern: str) -> dict[str, str]:
    API_URL = 'https://api.github.com/repos/ds-snu/text-datasets/contents'
    headers = {'User-Agent': 'Mozilla/5.0'}

    req = urllib.request.Request(API_URL, headers=headers)
    with urllib.request.urlopen(req) as response:
        files = json.loads(response.read().decode('utf-8'))

    texts = {}
    for file in files:
        name = file['name']

```

```
if fnmatch.fnmatch(name, pattern):
    raw_url = file['download_url']
    file_req = urllib.request.Request(raw_url, headers=headers)
    with urllib.request.urlopen(file_req) as r:
        texts[name] = r.read().decode('utf-8')

return texts
```

8장. 메시지 구조와 대화의 작동 원리

메시지 구조의 구현과 처리 흐름

(180쪽)

페르소나 기법을 활용해 문체 설정하기

```
instruction = '윤동주 시인의 스타일로 대답해야 합니다.'
user_message = input('>>> ')
response = get_llm_response(instruction, user_message)
print(response)
```

(181쪽)

특정 시스템 역할 흉내내게 하기

```
# --- 역할 지정에서 역할이 반드시 사람이어야 할 필요는 없다.
instruction = '''당신은 Windows 명령 프롬프트 역할을 해야 합니다.
사용자가 명령을 입력하면 명령 프롬프트 창에 표시되어야 할 내용으로 답을 해야 합니다.
다른 정보는 출력하지 마세요.
'''

user_message = input('>>> ')
response = get_llm_response(instruction, user_message)
print(response)
```

이전 대화를 기억하지 않는 구조적 한계

(182쪽).

대화 기록이 없을 때의 한계 확인하기

```
instruction = '당신은 친근한 챗봇입니다.'
```

```

while True:
    user_message = input('>>> ').strip()
    if user_message == '종료':
        print('대화를 종료합니다.')
        break
    response = get_llm_response(instruction, user_message)
    print(response)

```

(184쪽)

대화 기록을 넣어 모델에 기억력 제공하기

```

from langchain.messages import SystemMessage, AIMessage, HumanMessage

instruction = '당신은 친근한 챗봇입니다.'

# 대화 내역을 시스템 메시지로 초기화한다.
conversation_history = [SystemMessage(content=instruction)]

while True:
    user_message = input('>>> ').strip()
    conversation_history.append(HumanMessage(content=user_message))
    if user_message == '종료':
        print('대화를 종료합니다.')
        break
    response = get_llm_response(messages=conversation_history)
    print(response)
    conversation_history.append(AIMessage(content=response))

```

(185쪽)

```

print(conversation_history)

```

9장. 텍스트 생성과 변형의 자동화

사용자 그룹별 맞춤형 텍스트 생성

(192쪽)

제품 정보표로 사용자 그룹별 맞춤형 상품 설명 생성하기

제품 정보표

fact_sheet = ''

제품명: [르네상스 스타일 사무용 가구 컬렉션]

개요

- 종류: 서류 캐비닛, 작업대, 서가, 대형 회의 테이블 등 다양한 사무용 가구 포함.
- 디자인 영감: 르네상스 시대 미술과 건축에서 영감을 받은 디자인.
- 컬러 & 마감: 여러 색상의 쉘과 다양한 베이스 마감 옵션 제공.
- 장식 옵션: 플라스틱 배면 및 전면 천 장식(REN-4) 또는 전체 천 장식(REN-6), 12가지 원단 및 5가지 가죽 선택 가능.
- 베이스 마감 옵션: 브러시드 니켈, 매트 그레이, 광택 블랙, 골드 마감.
- 의자 옵션: 팔걸이 추가 여부 선택 가능.
- 사용처: 가정 및 사무용, 상업적 계약에 적합.

구조

- 베이스: 4개의 바퀴가 달린 메탈릭 코팅 알루미늄 베이스.
- 조절 기능: 전자식 높낮이 조절 기능.

치수

- 너비: 55 CM | 21.65"
- 깊이: 55 CM | 21.65"
- 높이: 82 CM | 32.28"
- 시트 높이: 46 CM | 18.11"
- 시트 깊이: 43 CM | 16.93"

옵션

- 캐스터 타입: 부드러운 바닥용 또는 딱딱한 바닥용 캐스터.

- 좌석 폼 밀도: 표준 (1.5 lb/ft³) 또는 높음 (2.5 lb/ft³).
- 팔걸이: 없음 또는 5 위치 조절 가능한 메탈릭 팔걸이.

재료

- 셸 & 베이스 글라이더: 변형된 나일론 PA6 코팅 알루미늄, 셸 두께 12 mm.
- 시트: 고밀도 폴리우레탄 폼.

원산지

- 이탈리아

'''

instructions = [

(# 일반 소비자를 대상으로 하는 예시

'당신의 업무는 주어진 제품 사양에 기반하여 온라인 '

'판매에서 사용할 제품 설명을 만드는 일입니다.Wn'

'최대 100 단어를 사용하세요.'

),

(# 가구 소매업체를 대상으로 하는 예시

'당신의 업무는 주어진 제품 사양에 기반하여 온라인 '

'판매에서 사용할 제품 설명을 만드는 일입니다.Wn'

'제품 설명은 가구 소매업체를 대상으로 하므로 '

'기술적인 내용이어야 하며 제품을 구성하는 재료에 '

'중점을 두어야 합니다.Wn'

'최대 100 단어를 사용하세요.'

)

]

user_message = f'''

다음 제품 사양이 제공하는 정보를 바탕으로 제품 설명을 작성하세요.

제품 사양:

{fact_sheet}

'''

for instruction in instructions:

response = get_llm_response(instruction, user_message)

print(response, '-' * 75)

반복적인 홍보 문구 자동 생성

(195쪽)

홍보 문구 자동 생성하기

```
from IPython.display import display, Markdown

instruction = '''당신은 15년 이상의 경력을 가진 마케팅 전문가입니다.
수많은 브랜드의 성공적인 캠페인을 이끌었으며,
소비자의 마음을 사로잡는 문구를 창의적이고 설득력 있게 작성하는 데 능숙합니다.

아래에 주어진 제품, 서비스, 아이디어, 혹은 개념에 대해 강렬하고 매력적인
**홍보 자료(마케팅 카피)**를 작성하세요.
대상 독자에게 어필할 수 있도록, 간결하면서도 인상 깊은 문장으로 표현해주세요.
필요하다면 감성적 언어, 수사적 질문, CTA(Call to Action)도 활용하세요.

<입력>
[사용자가 입력한 내용]
</입력>

<출력 형식(마크다운)>
## 제목(슬로건)
2~3문단의 홍보 문구
`해시태그`
</출력 형식(마크다운)>
'''

coffee_types = [
    '에스프레소',
    '아메리카노',
    '카페 라떼',
    '카푸치노',
    '마키아토',
    '모카',
    '콜드브루'
]
```

```

for coffee in coffee_types:
    response = get_llm_response(instruction, coffee)
    display(Markdown(response))
else:
    print(f"--- 작업을 완료했습니다. 총 {len(coffee_types)}개의 홍보 자료를 생성했습니다. {'-' * 15}")

```

관점과 기준에 따른 정보 요약

■ 같은 리뷰를 다른 기준으로 요약

(199쪽)

특정 제품의 리뷰를 다른 기준으로 요약하기

제품명: 하이테크 슬림 노트북

user_message = '''최근에 하이테크 슬림 노트북을 구매했습니다.
이 노트북은 정말 가볍고 휴대성이 뛰어납니다.
디자인도 매우 세련되어 보이고, 화면 해상도가 높아서 영화 보거나 문서 작업하기에 아주 좋아요.
특히 배터리 수명이 길어서 하루 종일 충전 걱정 없이 사용할 수 있어서 대만족입니다.
하지만 가격이 조금 높은 편이라서 비용을 고려해야 할 것 같아요.
비슷한 성능을 가진 다른 브랜드의 노트북이 좀 더 저렴할 수도 있으니까요.
배송은 예상보다 빨리 와서 기분이 좋았습니다.
노트북을 받자마자 바로 세팅해서 사용해봤는데, 사용하기 쉽고 빠른 부팅 속도에 만족합니다.

```

instructions = [
    ( # 단어/문장/글자 제한으로 요약하기
      '당신의 업무는 전자상거래 사이트에 올라온 제품 리뷰를 짧게 '
      '요약하는 것입니다. 리뷰를 최대 50 단어로 요약하세요.'
    ),
    ( # 배송 및 배달을 중심으로 요약하기

```

```

'당신의 업무는 전자상거래 사이트에 올라온 제품 리뷰를 짧게 '
'요약하여 배송 부서에 피드백을 제공하는 것입니다. '
'리뷰를 제품의 배송 및 배달과 관련된 내용에 중점을 두고 '
'최대 50 단어로 요약하세요.'
),
( # '요약' 대신 '추출' (한국어 특성상 영어처럼 잘 작동하지 않을 수도 있다.)
'당신의 업무는 전자상거래 사이트에 올라온 제품 리뷰에서 '
'배송 부서에 필요한 정보만을 추출(extract)하여 피드백을 '
'제공하는 것입니다. 리뷰에서 제품의 배송 및 배달과 관련된 '
'정보를 수정하지 말고 있는 그대로 추출(extract)하십시오. '
'50 단어로 제한합니다.'

),
( # 가격과 가치를 중심으로 요약하기
'당신의 업무는 전자상거래 사이트에 올라온 제품 리뷰를 짧게 '
'요약하여 제품 가격을 결정하는 가격 책정 부서에 피드백을 '
'제공하는 것입니다. 리뷰를 가격 및 인지한 가치와 관련된 '
'모든 측면에 중점을 두고 최대 50 단어로 요약하세요.'
)
]

for instruction in instructions:
    response = get_llm_response(instruction, user_message)
    print(response, '-' * 75, sep='\n')

```

다른 리뷰를 동일 기준으로 요약

(202쪽)

여러 제품의 리뷰를 동일 기준으로 요약하기

```

instruction = '''당신의 업무는
전자상거래 사이트에 올라온 제품 리뷰를 짧게 요약하는 것입니다.
리뷰를 최대 30 단어로 요약하세요.
'''

# 하이테크 슬립 노트북

```

```
review_1 = '''최근에 하이테크 슬림 노트북을 구매했습니다.  
이 노트북은 정말 가볍고 휴대성이 뛰어납니다.  
디자인도 매우 세련되어 보이고, 화면 해상도가 높아서 영화 보거나 문서 작업하기에 아주 좋아요.  
특히 배터리 수명이 길어서 하루 종일 충전 걱정 없이 사용할 수 있어서 대만족입니다.  
하지만 가격이 조금 높은 편이라서 비용을 고려해야 할 것 같아요.  
비슷한 성능을 가진 다른 브랜드의 노트북이 좀 더 저렴할 수도 있으니까요.  
배송은 예상보다 빨리 와서 기분이 좋았습니다.  
노트북을 받자마자 바로 세팅해서 사용해봤는데, 사용하기 쉽고 빠른 부팅 속도에 만족합니다.  
'''
```

스마트워치 리뷰

```
review_2 = '''최근에 구입한 스마트워치가 기대에 미치지 못했어요.  
배터리 수명이 너무 짧아서 하루에 한 번 이상 충전해야 해요.  
터치스크린 반응도 느리고, 때때로 앱이 제대로 동작하지 않는 문제가 있었습니다.  
가격 대비 성능이 좋지 않아요.  
배송은 빨랐지만, 제품에 대한 전반적인 만족도는 낮습니다.  
'''
```

진공 청소기 리뷰

```
review_3 = '''이 진공 청소기는 정말 강력하고 사용하기 쉬워요.  
무선이라서 어디든지 편하게 가져갈 수 있고, 청소 효율이 좋아요.  
배터리 수명도 꽤 괜찮습니다.  
다만, 청소기가 조금 무거운 편이라 오래 사용하면 팔이 피곤해질 수 있어요.  
가격 대비 성능은 만족스럽습니다.  
배송도 빠르고 제품 상태도 완벽해서 좋았습니다.  
'''
```

블루투스 헤드폰 리뷰

```
review_4 = '''이 블루투스 헤드폰은 소리 품질이 좋지 않아요.  
저음이 거의 없고, 때때로 연결이 끊기는 현상이 발생했습니다.  
착용감도 불편하고, 귀가 아프게 만드는 경우가 종종 있었어요.  
가격이 저렴한 편이지만, 이 제품은 가격만큼의 가치도 없는 것 같습니다.  
배송 과정에서 상자가 약간 손상된 채로 도착했어요.  
'''
```

전동 칫솔 리뷰

```
review_5 = '''이 전동 칫솔은 정말 효과적이에요.  
칫솔질이 훨씬 쉬워졌고, 이가 깨끗해진 느낌이 들어요.  
여러 모드가 있어서 상황에 따라 조절할 수 있고, 배터리 수명도 길어요.  
다만, 헤드 교체 비용이 조금 부담스러울 수 있어요.
```

가격이 적당하고, 배송도 빨라서 만족합니다.

'''

```
reviews = [review_1, review_2, review_3, review_4, review_5]
```

```
# --- for 루프를 사용하여 위 리뷰 5개를 요약한 결과를 출력한다.
```

```
for i, review in enumerate(reviews, start=1):
```

```
    response = get_llm_response(instruction, review)
```

```
    print(f'리뷰 {i}: {response}', '-' * 75, sep='\n')
```

10장. 정보 추출과 분류 자동화

키워드 추출

(217쪽)

웹 문서에서 핵심 키워드 추출하기

```
import json, urllib.parse, urllib.request

def fetch_wikipedia_plaintext(url: str, max_chars: int = 8000) -> str:
    """
    Wikipedia 문서 URL을 받아 요약/본문(plain text)을 가져온다.
    """
    try:
        parsed = urllib.parse.urlparse(url)
        title = urllib.parse.unquote(parsed.path.rpartition('/')[1]) # 한글/영문 문서명
        is_ko = parsed.netloc.startswith('ko.')
        lang = 'ko' if is_ko else 'en'

        api = f'https://{lang}.wikipedia.org/w/api.php'
        params = {
            'action': 'query',
            'format': 'json',
            'prop': 'extracts',
            'explaintext': '1',
            'exintro': '1',    # 서론만: 과도한 분량 확장 방지
            'redirects': '1',
            'titles': title,
        }
        api_url = f"{api}?{urllib.parse.urlencode(params)}"

        req = urllib.request.Request(
            api_url, headers={'User-Agent': 'Mozilla/5.0'}
        )
```

```

with urllib.request.urlopen(req, timeout=20) as r:
    data = json.loads(r.read().decode('utf-8', errors='ignore'))

    pages = data['query']['pages']
    page = next(iter(pages.values()))
    text = page.get('extract', "") or ""
    return text[:max_chars]
except Exception:
    # 네트워크 오류 등이 발생해도 프로그램을 멈추지 않고 빈 문자열을 반환한다.
    return ""

instruction = """다음 텍스트에서 핵심 키워드 5개만 추출하세요.
출력은 정확히 5줄만 허용합니다.
각 줄은 '- '로 시작하고, 키워드만 씁니다.
그 외 어떤 글도 쓰지 마세요.
"""

URLs = (
    'https://ko.wikipedia.org/wiki/한글',
    'https://ko.wikipedia.org/wiki/세종',
    'https://en.wikipedia.org/wiki/SpaceX',
    'https://en.wikipedia.org/wiki/Mars',
    'https://en.wikipedia.org/wiki/Agentic_AI'
)

for url in URLs:
    article = url.rpartition('/')[0]
    page_text = fetch_wikipedia_plaintext(url)
    response = get_llm_response(instruction, page_text)
    print(f'## {article} ##\n{response}', '-' * 75, '\n', sep='\n')

```

리뷰 및 문서 분류

(221쪽)

아웃도어 제품 리뷰 불러오기

```
reviews = load_text_files_from_github('review-outdoor-*.txt')

for name in sorted(reviews): # 이름순으로 정렬하여 출력한다.
    print(f'--- {name} ---')
    print(reviews[name], 'WnWn')
```

(223쪽)

미리 정의한 태그 목록으로 리뷰 분류하기

instruction = '''제공되는 사용자 리뷰를 분석하여
아래 목록에서 태그를 선택해 제공하는 것이 당신의 업무입니다.
답변은 글머리 기호 형태로 제공하세요. 아래에 있는 태그 목록에서만 선택해야합니다.

긍정적 태그 vs. 부정적 태그 중 하나만 선택하고 둘 다 선택하지 마세요.

- 가격 대비 좋은 가치 제공 vs. 비용이 너무 많이 듦
- 기대치보다 더 잘 작동함 vs. 기대치만큼 잘 작동하지 않음
- 필수 기능 포함 vs. 필수 기능 부족
- 사용하기 쉬움 vs. 사용하기 어려움
- 품질과 내구성이 높음 vs. 품질과 내구성이 떨어짐
- 유지 및 수리가 쉽고 비용이 저렴함 vs. 유지 및 수리가 어렵거나 비용이 많이 듦
- 운반하기 쉬움 vs. 운반하기 어려움
- 보관하기 쉬움 vs. 보관하기 어려움
- 다른 장치나 시스템과 호환됨 vs. 다른 장치나 시스템과 호환되지 않음
- 안전하고 사용자 친화적임 vs. 사용하기에 안전하지 않거나 위험함
- 우수한 고객 지원 vs. 미흡한 고객 지원
- 넉넉하고 포괄적인 보증 vs. 제한적이거나 불충분한 보증

'''

```
for filename, review in reviews.items():
    response = get_llm_response(instruction, review)
```

```
print(f'--- {filename} ---\n{response}\n')
else:
    print('!!! 작업을 완료했습니다, ' * 57)
```

고객 요청 분류

(227쪽)

고객 문의를 JSON 형식으로 분류하기

instruction = f'''당신은 고객 서비스 부서에서 근무하는 친절한 직원입니다.
고객의 각 문의는 주 카테고리 와 부 카테고리 로 분류해야 합니다.
주 카테고리 와 부 카테고리 로 구분하여 **코드 블록 없이**, 순수한 JSON 객체 형식만 출력하세요.

주 카테고리: 청구, 기술 지원, 계정 관리, 일반 문의

청구의 부 카테고리:

구독 취소 또는 업그레이드

결제 수단 추가

과금에 대한 설명

청구에 대한 이의 제기

기술 지원의 부 카테고리:

일반적인 문제 해결

기기 호환성

소프트웨어 업데이트

계정 관리의 부 카테고리:

비밀번호 재설정

개인정보 업데이트

계정 해지

계정 보안

일반 문의의 부 카테고리:

제품 정보

가격

피드백

서비스 담당자와 연결

'''

```
user_requests = [  
    '노트북에 대해 자세히 알려주세요.',  
    '계정을 삭제했으면 합니다. 내 프로필과 나의 모든 사용자 데이터를 삭제해 주시길 원합니다.',  
    '다음 달 요금이 왜 이렇게 많이 나왔는지 설명 부탁드립니다.',  
    '결제 카드 정보를 새로 등록하고 싶어요. 어떻게 하나요?',  
    '앱이 자꾸 강제 종료돼요. 해결 방법이 있을까요?',  
    '제 계정에 로그인할 수 없습니다. 비밀번호를 잊어버렸어요.',  
    '프로 플랜으로 업그레이드하고 싶습니다. 안내 부탁드립니다.',  
    '앱이 실행되지 않아서 여러 번 재설치해 봤는데 계속 같은 문제가 발생합니다.',  
    '최근 청구 내역에 오류가 있는 것 같습니다. 이의 제기를 하고 싶어요.',  
    '내 정보를 최신 주소와 전화번호로 업데이트하려면 어떻게 해야 하나요?',  
    '계정 보안 설정을 강화하고 싶어요. 2단계 인증이 가능한가요?',  
    '서비스에 대한 피드백을 남기고 싶습니다. 만족도 조사 링크가 있을까요?',  
    '신규 출시된 제품에 대해 자세히 알려주세요.',  
    '고객센터 담당자와 직접 통화할 수 있을까요?',  
    '최근에 받은 소프트웨어 업데이트 이후 앱이 느려졌어요.',  
    '청구서에 표시된 '기타 수수료'가 뭔가요? ',  
    '결제 수단으로 페이팔을 추가할 수 있나요?',  
    '서비스를 취소하고 환불을 받고 싶어요.'  
]
```

```
for request in user_requests:  
    response = get_llm_response(instruction, request)  
    print(response, '-' * 75, sep='\\n')  
else:  
    print('!!! 작업을 완료했습니다, '!' * 57)
```

11장. 개체 식별과 활용

문서 안의 객체 식별

(241쪽)

제품 리뷰 데이터 불러오기

```
reviews = load_text_files_from_github('review-product-*.txt')

for name in sorted(reviews): # 이름순으로 정렬하여 출력한다.
    print(f'--- {name} ---')
    print(reviews[name], 'WnWn')
else:
    print('!!! 작업을 완료했습니다, '!' * 57)
```

(242쪽)

불러온 리뷰에서 브랜드와 제품명 추출하기

```
from IPython.display import Markdown

instruction = """당신의 업무는 주어진 개체(entity)를 추출하는 것입니다.
제품 리뷰 텍스트에서 다음 항목을 식별하세요.
- 그 제품을 만든 회사
- 평가자가 구매한 제품

답변은 'brand'와 'item'을 제목으로 하는 마크다운 표 형식으로 작성하세요.
정보가 없다면 값(value)으로 'unknown'을 사용하세요.
답변은 가능한 짧게 해주세요.
"""

contents = [content for content in reviews.values()]
response = get_llm_response(instruction, str(contents))
Markdown(response)
```

기준에 따른 문서 분류

(244쪽)

명소와 식당 소개 블로그만 골라내기

```
instruction = "'O' 또는 'X'으로만 답변하시고 그 외 어떤 말도 하지 마세요.  
이 블로그 글은 지역의 명소와 각 명소 근처의 식당이나 카페를 소개하고 있습니다.  
블로그 글:  
'''  
  
blog_files = []  
  
# 깃허브 저장소에서 blog-*.txt 파일들을 불러온다.  
blogs = load_text_files_from_github('blog-*.txt')  
  
for name, blog_post in blogs.items():  
    response = get_llm_response(instruction, blog_post)  
    # 모델의 응답이 'O'이면 해당 파일명을 목록에 추가한다.  
    if response == 'O':  
        blog_files.append(name)  
    print(f'{name} -> {response}')  
else:  
    print(blog_files)
```

(245쪽)

```
print(blogs['blog-sydney.txt'])
```

(246쪽)

```
print(blogs['blog-toronto.txt'])
```

문서 정보 및 계층적 식별

■ 명소, 식당, 카페 자동 표시

(248쪽)

블로그 글을 HTML 형태로 꾸며 출력하기

```
from IPython.display import display, HTML
```

```
instruction = '''아래 블로그 글은  
한 도시의 주요 명소와 각 명소 근처에서 추천할 만한 식당이나 카페를 소개하고 있습니다.
```

```
당신의 작업은 다음과 같습니다.
```

1. 도시 이름은 강조하지 마세요. (즉, 색상 표시 및 하이라이트를 하지 마세요.)
2. 명소 이름은 굵게 표시하고 고유한 색상으로 하이라이트하세요.
3. 각 명소 근처의 식당 또는 카페 이름은 명소와 동일한 색상으로 굵게 표시하세요. 하지만 하이라이트는 하지 마세요.
4. 명소나 식당/카페 이름에 괄호 속 원어 표기가 있더라도, 하이라이트하지 마세요.
5. 명소나 식당/카페가 아닌 다른 텍스트에는 어떠한 색상이나 포매팅도 적용하지 마세요.
6. 최종 결과는 Jupyter notebook에서 직접 HTML로 렌더링될 수 있도록 하되, 코드 블록 태그(예: ``html 등)는 포함하지 마세요.
7. HTML 문단 구조(`<p>` 태그 등)는 원본 텍스트의 문단 개행을 그대로 반영해서 구성하세요.
8. 글자 크기는 기본 크기를 유지하며, 폰트 크기 조정은 하지 마세요.

```
블로그 글:
```

```
'''
```

```
for name in blog_files:  
    print(f'--- {name} ---')  
    blog_post = blogs[name]  
    response = get_llm_response(instruction, blog_post)  
    display(HTML(response))  
    print()  
else:  
    print('!!! 작업을 완료했습니다, '!' * 57)
```

■ 명소, 식당, 카페 이름 분리 추출

(251쪽)

블로그에서 명소, 식당, 카페 목록 추출하기

instruction = '''다음 블로그 글에서

소개한 명소와 각 명소 근처에 추천한 식당이나 카페 이름을 추출하여 목록으로 만들어 주세요.

명소 근처에 식당이나 카페가 없는 경우에는 'N/A'로 표시하세요.

명소, 식당, 카페 모두 블로그 글에 나온 이름과 표기(예: 한글(원어), 영어, 현지어 등)를 **글에 나온 그대로** 사용하세요.

식당과 카페의 구분이 명확하지 않거나 애매한 경우에는 다음 기준을 따르세요:

- 커피, 차, 디저트, 간단한 샌드위치나 브런치 등이 중심이면 '카페'로,
- 식사(점심·저녁 등) 메뉴가 중심이면 '식당'으로 분류하세요.
- 여전히 애매하다면 '카페'로 분류하세요.

중요:

- 명소, 식당, 카페 이름에 쉼표(,)가 포함된 경우에만 해당 필드를 큰따옴표(")로 감싸세요.
- 쉼표(,)가 포함되지 않은 경우에는 따옴표를 절대 붙이지 마세요.
예: 푸에르타 델 솔(Puerta del Sol),N/A,산 히네스 초콜라테리아(Chocolatería San Ginés)
- 각 컬럼(식당, 카페)에는 반드시 **한 개의 이름만** 넣으세요.
- 식당/카페 후보가 둘 이상일 때는 **블로그 글에 먼저 등장한 이름**만 넣으세요.
- 반드시 쉼표(,)만을 구분자로 사용하세요. 세미콜론(;)이나 다른 기호를 절대 사용하지 마세요.

대답은 CSV 형식으로 제공되되, "````csv"와 같은 코드 블록 표시는 절대 하지 마세요.

쉼표 뒤에 공백을 추가하지 말고, 반드시 컬럼 헤더를 포함하세요.

형식:

명소,식당,카페

명소_1,식당_1, 카페_1

명소_2,식당_2, X

...

블로그 글:

'''

for name in blog_files:

print(f'--- {name} ---')

blog_post = blogs[name]

```

response = get_llm_response(instruction, blog_post)
print(response, '\n')

# 결과를 파일로 저장한다.
csv_filename = name.rsplit('.', 1)[0] + '.csv'
with open(csv_filename, mode='w', encoding='utf-8') as file:
    file.write(response)
else:
    print('!!! 작업을 완료했습니다, '!' * 57)

```

식별한 개체를 활용한 휴가 계획 생성

■ 일정 데이터 읽기

(256쪽)

깃허브에서 여행 일정 CSV 파일 읽어오기

```

itinerary = load_text_files_from_github('itinerary.csv')['itinerary.csv']

for line in itinerary.splitlines():
    print(line)

```

■ 특정 도시 샘플 생성

(257쪽)

특정 도시로 일일 일정 샘플 생성하기

```

# 깃허브에서 CSV 파일 내용을 읽어온다.
itinerary = load_text_files_from_github('itinerary.csv')['itinerary.csv']

```

```

# 헤더와 특정 행을 선택한다.
lines = itinerary.splitlines()
header = lines[0] # 아이슬란드
row = lines[4]

# 행을 컬럼 단위로 분해한다.
city, country, arrival, departure = [v.strip() for v in row.split(',')]

user_message = f'''
{arrival}부터 {departure}까지 {city}, {country}를 방문할 계획입니다.
자세한 일일 일정을 작성해주세요.
'''

response = get_llm_response("", user_message)
print(response)

```

■ 식당 리스트 생성

(261쪽)

도시별 식당과 카페 CSV 불러오기

```

# 깃허브에서 CSV 파일을 읽어온다.
reykjavik = load_text_files_from_github('blog-reykjavik.csv')['blog-reykjavik.csv']

lines = reykjavik.splitlines()

# 헤더에서 '식당' 컬럼 위치를 찾는다.
header = [col.strip() for col in lines[0].split(',')]
restaurant_idx = header.index('식당')

# 식당 컬럼만 추출한다. (N/A 제외)
restaurants = []
for line in lines[1:]:
    cols = [col.strip() for col in line.split(',')]
    value = cols[restaurant_idx]
    if value and value != 'N/A':

```

```

    restaurants.append(value)
else:
    print(restaurants)

```

(262쪽)

일정과 식당 정보를 연결해 여행 계획 확장하기

```

instruction = f'''{arrival}부터 {departure}까지 {country} {city}를 여행할 예정입니다.
매일 상세한 활동 계획이 들어간 일정을 만들어 주세요.

각 날짜별로 **오전, 오후, 저녁** 일정이 모두 포함되어야 하며, 각 시간대에는 활동 또는 방문 장소가
반드시 명시되어야 합니다.
아침, 점심, 저녁 식사 시간(예: 아침 8:00, 점심 12:30, 저녁 18:00)과 방문할 식당 이름도 꼭 함께
적어 주세요.
저녁 일정(저녁 식사 후의 활동이나 자유시간 등)도 반드시 포함해 주세요.

사용자로부터 제공받은 식당 리스트에 있는 식당은 한 번씩만 추천하고, 식사가 더 필요한 경우에는
{city}에서 새로 추천할 만한 식당을 자유롭게 추가해 주세요.
'''

response = get_llm_response(instruction, str(restaurants))
print(response)

```

■ 전체 휴가 계획 생성

(265쪽)

전체 도시 일정 일괄 생성하기

```

import csv

# 한글 도시명과 실제 파일명을 매핑한다.
city_name = {

```

```

'교토': 'blog-kyoto.csv',
'케이프타운': 'blog-cape-town.csv',
'토론토': 'blog-toronto.csv',
'레이카비크': 'blog-reykjavik.csv',
'파리': 'blog-paris.csv',
'마드리드': 'blog-madrid.csv'
}

```

각 목적지의 상세 일정을 저장할 딕셔너리다.

```
detailed_itinerary = {}
```

깃허브에서 itinerary.csv 내용을 읽어온다.

```

itinerary = load_text_files_from_github('itinerary.csv')['itinerary.csv']
itinerary_lines = itinerary.splitlines()

```

csv.reader는 쉼표가 포함된 필드(따옴표 처리)를 안전하게 처리한다.

```
itinerary_reader = csv.reader(itinerary_lines)
```

헤더 한 줄을 건너뛰는다.

```
next(itinerary_reader)
```

블로그 CSV는 미리 한 번에 가져와 둔다. (반복 호출 방지)

```
blog_csvs = load_text_files_from_github('blog-*.csv')
```

itinerary.csv에서 헤더를 제외한 각 여행 일정을 한 행씩 순회한다.

```
for row in itinerary_reader:
```

```
    arrival = row[0].strip()
```

```
    departure = row[1].strip()
```

```
    city = row[2].strip()
```

```
    country = row[3].strip()
```

```
    filename = city_name[city]
```

```
    csv_text = blog_csvs[filename]
```

```
    print(f'> {country} {city}의 상세 일정을 생성합니다.')
```

```
    instruction = f'''{arrival}부터 {departure}까지 {country} {city}를 여행할 예정입니다.
```

```
매일 상세한 활동 계획이 들어간 일정을 만들어 주세요.
```

각 날짜별로 **오전, 오후, 저녁** 일정이 모두 포함되어야 하며,

각 시간대에는 활동 또는 방문 장소가 반드시 명시되어야 합니다.

아침, 점심, 저녁 식사 시간(예: 아침 8:00, 점심 12:30, 저녁 18:00)과 방문할 식당 이름도 꼭 함께 적어 주세요.
저녁 일정(저녁 식사 후의 활동이나 자유시간 등)도 반드시 포함해 주세요.

사용자로부터 제공받은 식당 리스트에 있는 식당은 한 번씩만 추천하고, 식사가 더 필요한 경우에는 {city}에서 새로 추천할 만한 식당을 추가해 주세요.
'''

도시별 CSV에서 '식당' 컬럼만 뽑는다. (따옴표/쉼표 포함 필드 안전 처리)

```
csv_lines = csv_text.splitlines()
```

```
blog_reader = csv.reader(csv_lines)
```

```
blog_header = [h.strip() for h in next(blog_reader)]
```

```
restaurant_idx = blog_header.index('식당')
```

```
restaurants: list[str] = []
```

```
for blog_row in blog_reader:
```

```
    if len(blog_row) <= restaurant_idx:
```

```
        continue
```

```
    value = blog_row[restaurant_idx].strip()
```

```
    if value and value != 'N/A':
```

```
        restaurants.append(value)
```

각 목적지의 상세 일정을 생성해 저장한다.

```
detailed_itinerary[city] = get_llm_response(instruction, '\n'.join(restaurants))
```

```
else:
```

```
    print('\n모든 여행 일정이 완료되었습니다.')
```

(268쪽)

전체 도시 일정 출력하기

```
for city, response in detailed_itinerary.items():
```

```
    print(f'--- {city} ---\n{response}\n\n')
```

12장. 감성 분석과 자동 응대

감성 분석 및 사용자 의도 식별

(281쪽)

입력할 리뷰 텍스트 구성하기

```
instruction = '''고객이 작성한 제품 리뷰에서 다음 항목들을 식별하세요.
```

- 제품 리뷰를 작성한 고객의 감정 상태 (positive 또는 negative)
- 제품 리뷰를 작성한 고객이 구입한 제품에 만족하는가? (true 또는 false)
- 제품 리뷰를 작성한 고객이 구매한 제품 이름
- 그 제품을 만든 회사

```
답변은 'sentiment', 'satisfaction', 'item', 'brand'를 키(key)로 가지는 JSON 형식으로 작성하세요.
```

```
정보가 없다면 값(value)으로 'unknown'을 사용하세요.
```

```
'satisfaction' 값(value)은 불린(boolean) 형식으로 표시하세요.
```

```
'''
```

```
user_message = '''
```

```
최근에 키친아트 스마트케틀 전기 주전자를 구매했는데, 디자인은 괜찮았지만 가격 대비 기대에 못  
미쳤어요.
```

```
배송은 빨랐지만, 제품을 사용해보니 물이 살짝 새는 문제가 있었습니다.
```

```
이 문제를 키친아트에 알렸더니 새 스마트케틀을 보내주겠다고 했지만, 배송이 예상보다 오래  
걸렸어요.
```

```
새로 온 주전자는 문제가 없었지만, 처음에 문제가 생긴 것이 실망스러웠습니다.
```

```
또, 스마트케틀이 조금 시끄러워서 조용한 환경에서는 사용하기가 불편합니다.
```

```
'''
```

```
response = get_llm_response(instruction, user_message)
```

```
print(response)
```

분석 결과 기반의 맞춤형 응대 자동 생성

(283쪽)

리뷰 파일 읽어오기

```
import urllib.request, json, fnmatch

API_URL = 'https://api.github.com/repos/ds-snu/text-datasets/contents'

# 파일 목록을 받아온다.
with urllib.request.urlopen(API_URL) as response:
    files = json.loads(response.read().decode('utf-8'))

reviews = {}
for file in files:
    name = file['name']
    # customer-review-*.txt 패턴인지 확인한다.
    if fnmatch.fnmatch(name, 'customer-review-*.txt'):
        raw_url = file['download_url']
        with urllib.request.urlopen(raw_url) as r:
            content = r.read().decode('utf-8')
            reviews[name] = content

# 이름순으로 정렬하여 출력한다.
for name in sorted(reviews):
    print(f'--- {name} ---')
    print(reviews[name], 'WnWn')
```

(285쪽)

리뷰에 대한 답변 이메일 초안 작성하기

```

instruction = """당신은 고객 서비스 담당 부서에서 일하는 아주 친절한 직원입니다.
하의 임무는 소중한 고객에게 이메일 답장을 작성해서 보내는 것입니다.
고객의 이메일을 받아, 그들의 제품 평가에 감사하는 답장을 작성하세요.
제품 평가의 감정이 긍정적이거나 중립적이라면, 그들의 평가에 감사하다고 답장하세요.
만약 감정이 부정적이라면, 정중하게 사과하고 고객 서비스에 연락할 것을 제안하세요.
제품 평가에서 구체적인 내용을 사용하는 것을 잊지 마세요.
간결하고 전문적인 어조로 작성하세요.
이메일을 '고객 담당 AI 직원'으로 서명하세요.
"""

for filename, review in reviews.items():
    response = get_llm_response(instruction, review)
    print(f'--- {filename} ---\n{response}\n')
else:
    print('--- 리뷰에 대한 이메일 초안 작성이 완료되었습니다.', '-' * 45)

```

13장. 텍스트 정제와 변환

오타자 교정과 문법 정제

(297쪽)

잘못된 영어 문장을 표준 문장으로 교정하기

```
instruction = '제공되는 텍스트를 표준 영어로 번역합니다.'
user_message = 'She no went to the market.'
response = get_llm_response(instruction, user_message)
print(response)
```

(298쪽)

문장 교정 결과를 JSON 형식으로 구조화하기

```
instruction = """당신의 업무는 사용자가 제공한 문장을
교정하고 올바르게 수정한 문장으로 다시 작성하는 것입니다.
수정한 문장을 보여주고, 마지막에 '수정 전'과 '수정 후'를
키(key)로 가지는 JSON 형식으로 작성하고 값(value)으로
각각 수정 전 단어와 수정 후 단어를 넣은 후 출력하세요.
"""

user_message = """내일은 날씨가 맑을 거예요.
그래서 나는 내일 학교에 갈게요. 나는 연구실에서 조용이 컴퓨터로 작업하기를 선호합니다.
그 덕분에 나의 논문이 유명 학술지에 게재되었습니다. 그나저나 너는 어떻게 지내고 있어?
"""

response = get_llm_response(instruction, user_message)
print(response)
```

(299쪽)

오류 수정하고 문장 스타일을 개선하기

```
instruction = '''당신의 업무는 사용자가 제공한
문장을 교정하고 올바르게 수정한 문장으로 다시 작성하는 것입니다.
고급 독자들이 이 글의 대상이기 때문에 문장을 좀 더 세련되고 설득력 있게 만드세요.
'''

user_message = '''내일은 날씨가 맑을 꺼예요.
그래서 나는 내일 학교에 갈게요. 나는 연구실에서 조용이 컴퓨터로 작업하기를 선호합니다.
그 덕분에 나의 논문이 유명 학술지에 게재되었습니다. 그나저나 너는 어떻게 지내고 있어?'
'''

response = get_llm_response(instruction, user_message)
print(response)
```

페르소나에 따른 어조 변환

(301쪽)

비격식 문장을 정중하고 공식적인 문장 스타일로 변환하기

```
instruction = '당신은 제공받은 문장을 정중하고 공식적인 문장으로 바꾸는 전문가입니다.'

user_message = '어이, 나는 홍길동인디, 이 노트북 사양 좀 줘봐.'

response = get_llm_response(instruction, user_message)
print(response)
```

(302쪽)

여러 거친 문장을 일관된 정중한 표현으로 변환하기

```
instruction = '''당신은 제공받은 문장을 정중하고 공식적인 문장으로 바꾸는 전문가입니다.  
불친절한 표현이 있다면 간결하면서도 친절하고 예의바른 문장으로 바꿔주세요.  
'''
```

```
rude_sentences = [  
    '야, 그것 좀 빨리 가져와 봐.',  
    '너 뭐 하는 거야? 제대로 안 해?',  
    '그거 하나도 제대로 못 하냐?',  
    '말귀를 못 알아먹어?',  
    '왜 이렇게 눈치가 없어?',  
    '좀 빨리빨리 움직여 봐.',  
    '내가 하라는 대로 하면 되잖아.',  
    '이거 하나 처리 못하냐 진짜?',  
    '시키는 거나 잘해라.'  
]
```

```
for rude_sentence in rude_sentences:  
    response = get_llm_response(instruction, rude_sentence)  
    print(response, '-' * 75, sep='\n')
```

다국어 번역

(304쪽)

사용자 입력을 받아 6개 언어로 동시에 번역하기

```
instruction = '''당신은 사용자가 제공한 영어를 번역하는 다국어 번역 전문가입니다.  
사용자가 제공한 내용을 다음 형식에 맞추어 프랑스어, 아랍어, 스페인어, 중국어, 일본어, 한국어로  
번역합니다.  
French:
```

```

Arabic:
Spanish:
Chinese:
Japanese:
Korean:
'''

user_message = input('>>> ')

response = get_llm_response(instruction, user_message)
print(response)

```

범용 번역기 설계

(306쪽)

입력 언어를 영어와 한국어로 번역해 일관된 형식으로 출력하기

```

instruction = '''당신은 사용자가 제공한 문장을 번역하는 다국어 번역 전문가입니다.
다양한 국가의 사용자가 작성한 문장을 영어와 한국어로 번역합니다.
원어 옆의 괄호 안에는 사용자가 작성한 문장의 원어를 명시해야 합니다.
한국어로 번역할 때는 존칭어를 사용합니다.
다음 형식에 맞추어 번역합니다.
원어 (한국어로 원어 표기):
영어:
한국어:
'''

user_messages = (
    'La batterie de mon téléphone est trop courte et dure à peine une journée.',
    'La calidad de la cámara de mi teléfono es mala, especialmente en condiciones de poca luz.',
    'Spesso riscontro chiamate perse e scarsa potenza del segnale con il mio telefono.',
    'Сенсорный экран моего телефона не реагирует и иногда не регистрирует касания.',
    'Bản cập nhật phần mềm mới nhất đã làm cho điện thoại của tôi chạy chậm và hay gặp lỗi hơn.'
)

```

```
for user_message in user_messages:  
    response = get_llm_response(instruction, user_message)  
    print(response, '-' * 75, sep='\n')
```

14장. 실행 가능한 구조로의 변환

자연어 기반 SQL 쿼리 생성

(315쪽)

데이터베이스 스키마를 바탕으로 SQL 쿼리 생성하기

instruction = ""다음 SQL 테이블들이 주어졌을 때,
사용자의 요청에 따라 SQL 쿼리를 작성하는 것이 당신의 업무입니다.

```
CREATE TABLE Orders (  
    OrderID int,  
    CustomerID int,  
    OrderDate datetime,  
    OrderTime varchar(8),  
    PRIMARY KEY (OrderID)  
);
```

```
CREATE TABLE OrderDetails (  
    OrderDetailID int,  
    OrderID int,  
    ProductID int,  
    Quantity int,  
    PRIMARY KEY (OrderDetailID)  
);
```

```
CREATE TABLE Products (  
    ProductID int,  
    ProductName varchar(50),  
    Category varchar(50),  
    UnitPrice decimal(10, 2),  
    Stock int,  
    PRIMARY KEY (ProductID)  
);
```

```
CREATE TABLE Customers (
    CustomerID int,
    FirstName varchar(50),
    LastName varchar(50),
    Email varchar(100),
    Phone varchar(20),
    PRIMARY KEY (CustomerID)
);
'''
user_message = '2050년 7월 1일의 모든 주문에 대한 평균 주문 총액을 계산합니다.'

response = get_llm_response(instruction, user_message)
print(response)
```

대시보드 및 웹 인터페이스 제작

(318쪽)

자연어로 요구해 단일 페이지 웹사이트 생성하기

```
from IPython.display import HTML

instruction = '''당신의 업무는 사용자가 요청한 내용을 기반으로
단일 페이지 웹사이트를 작성하는 것입니다.
웹사이트는 자바스크립트와 CSS가 내장된 HTML 파일이어야 합니다.
HTML 문서로만 변환하고 추가 설명은 하지 마세요.
그리고 답변의 시작과 끝에 HTML 코드를 표시하는 삼중 백틱(```html)을 반드시 생략하고 HTML
코드만 출력하세요.
'''

user_message = '''
차트(원형, 막대, 선 그래프 등 다양한 형태), 테이블, 드롭다운, 모달 등 여러 UI 컴포넌트를 포함한
모던하고 반응형인 대시보드 웹사이트를 작성해주세요.
사용자 인증 상태 표시, 실시간 데이터 연동(예: 간단한 가짜 API 사용)도 구현해 주시고, 색상은 깔끔한
```

다크 모드 톤으로 해주세요.

각 컴포넌트에는 간략한 주석과 실제 사용 예시도 함께 써주세요.

'''

```
response = get_llm_response(instruction, user_message)
```

```
HTML(response)
```

15장. 추론 제어와 사고 과정 설계

사고 추론 과정 확인

(324쪽)

추론 과정을 점검하여 응답이 올바른지 확인하기
instruction = f'''당신은 전자제품 회사의 판매 사원입니다. 고객 문의에 답변하려면 다음 단계를 따르세요. 1단계: 먼저 고객이 특정 제품 또는 제품들에 대한 문의를 하고 있는지 판단하세요. 이때 제품 카테고리는 고려하지 않습니다. 2단계: 고객이 특정 제품에 대해 묻는 경우, 해당 제품이 다음 목록에 있는지 확인하세요. 구매 가능한 제품 목록: 1. 제품: 스마트홈허브 카테고리: 스마트홈 기기 브랜드: 스마트코리아 모델 번호: SH-HB100 보증기간: 1년 평점: 4.5 기능: 음성 인식, 다양한 연결 기기 지원, 무선 연결 설명: 다양한 스마트 기기를 제어하는 중앙 허브. 가격: 150,000원 2. 제품: 에어컨 휴대용 선풍기 카테고리: 계절가전 브랜드: 쿨한세상 모델 번호: AC-PF200 보증기간: 1년 평점: 4.2 기능: USB 충전, 3단계 바람 조절, 저소음 설명: 여름철 필수 휴대용 선풍기.

가격: 20,000원

3. 제품: 클린봇 로봇청소기

카테고리: 생활가전

브랜드: 클린월드

모델 번호: CB-RV300

보증기간: 2년

평점: 4.7

기능: 자동 장애물 감지, 스마트폰 연동, 무선 충전

설명: 편리하게 청소를 도와주는 스마트 청소기.

가격: 500,000원

4. 제품: 쿡마스터 전기레인지

카테고리: 주방가전

브랜드: 쿡코리아

모델 번호: CK-ER500

보증기간: 1년

평점: 4.4

기능: 터치스크린, 타이머 기능, 다양한 조리 메뉴

설명: 빠르고 편리한 요리를 도와주는 전기레인지.

가격: 120,000원

5. 제품: FreshFridge 냉장고

분류: 주방가전

브랜드: FreshFridge

모델 번호: FF-RF300

보증: 2년

평점: 4.8

특징: 300리터 용량, 냉동/냉장/김치 저장 가능, 스마트 연결 기능

설명: 다용도 저장공간과 최신 기술을 탑재한 가정용 냉장고입니다.

가격: 1,199,000원

6. 제품: CleanAir 공기청정기

분류: 계절가전

브랜드: CleanAir

모델 번호: CA-AP100

보증: 1년

평점: 4.6

특징: 4단계 필터, 자동 오염 감지, 저소음

설명: 고성능 공기청정기로, 실내의 미세먼지와 악취를 효과적으로 제거합니다.

가격: 299,000원

7. 제품: SoundMax 무선 스피커

분류: TV/음향

브랜드: SoundMax

모델 번호: SM-WS500

보증: 1년

평점: 4.7

특징: 블루투스 연결, 24시간 연속재생, 방수 기능

설명: 뛰어난 음질과 긴 배터리 수명을 자랑하는 프리미엄 무선 스피커입니다.

가격: 149,000원

3단계: 문의 메시지가 위의 목록에 있는 제품을 포함하고 있다면, 고객이 문의하는 메시지에서 가정하는 내용을 나열하세요.

예를 들어, 가전 제품 A의 평점이 B보다 높다든가, 가전 제품 C의 보증 기간이 2년이다와 같은 가정입니다.

4단계: 고객이 어떤 가정을 했다면, 그 가정이 제품 정보에 기반한 사실인지를 확인하세요.

5단계: 먼저 고객이 잘못된 가정을 했다면, 고객의 잘못된 가정을 정중하게 바로잡아주세요.

매장에서 판매하는 상품은 여기 있는 7개뿐이니 7개의 구매 가능한 제품 목록에 있는 가전 제품만 언급하거나 참조하세요.

고객에게 친절한 어조로 답변하세요.

반드시 다음 형식을 사용하세요. '4단계' 다음에는 '5단계'가 아닌 '고객에 대한 응답'으로 출력하세요:

1단계: 1단계의 추론을 설명하세요.

2단계: 2단계의 추론을 설명하세요.

3단계: 3단계의 추론을 설명하세요.

4단계: 4단계의 추론을 설명하세요.

고객에 대한 응답: 고객에게 설명하세요.

'''

```
user_message = input('>>> ')
```

```
response = get_llm_response(instruction, user_message)
```

```
print(response)
```

(328쪽)

추론 결과에서 최종 고객 응답만 추출하기

```
try:
    final_response = response.split('고객에 대한 응답:')[1].strip()
except Exception:
    final_response = '죄송합니다, 제가 지금 문제가 있어서요, 다른 질문을 해보세요.'

print(final_response)
```

사고 경로 분기와 탐색

■ 프롬프트 설계

(330쪽)

ToT용 단계별 프롬프트 설계하기

```
# --- prompts
# {input}, {factors}는 입력 변수다.
prompt_1 = '''1 단계:
아래 문제에 대한 해결책을 세 가지 제안해 주세요.
각 해결책은 서로 다른 접근 방식을 사용해야 하며,
아래 제약 요인을 반드시 고려해 주세요.

<문제>
{input}
</문제>

<제약 요인>
{factors}
</제약 요인>
'''
```

```
# {solutions}, {factors}는 입력 변수다.
```

```
prompt_2 = '''2 단계:
```

```
아래 해결책들을 비교 평가해 주세요.
```

```
각 해결책에 대해 장점과 단점, 초기 노력, 구현 난이도,
```

```
핵심 리스크, 제약 요인과의 적합성을 기준으로
```

```
10점 만점 점수와 한 줄 근거를 제시해 주세요.
```

```
<해결책>
```

```
{solutions}
```

```
</해결책>
```

```
<제약 요인>
```

```
{factors}
```

```
</제약 요인>
```

```
'''
```

```
# {solutions}, {factors}는 입력 변수다.
```

```
prompt_3 = '''3단계:
```

```
각 해결책에 대해 사고 과정을 심화해 주세요.
```

```
아래 항목을 모두 포함해 정리해 주세요.
```

```
1. 잠재 시나리오(성공, 실패)
```

```
2. 실행 전략(단계별)
```

```
3. 필요한 파트너십과 리소스
```

```
4. 예상 장애물과 극복 방안
```

```
5. 예상치 못한 결과와 대응 방법
```

```
<해결책>
```

```
{solutions}
```

```
</해결책>
```

```
<제약 요인>
```

```
{factors}
```

```
</제약 요인>
```

```
'''
```

```
# {deepen_thoughts}는 입력 변수다.
```

```
prompt_4 = '''4단계:
```

```
지금까지의 평가와 심화 내용을 바탕으로 해결책을 우선순위로 정리해 주세요.
```

```
1. 1위부터 3위까지 순위를 매기고
```

2. 각 순위의 근거를 3문장 이내로 설명하고
3. 1위 해결책의 즉시 실행 가능한 다음 행동을 5개 제시해 주세요.

```
<검토 내용>
{deepen_thoughts}
</검토 내용>
'''
```

■ 모델 선정 및 초기화

(332쪽)

단계별 역할에 맞는 모델로 초기화하기

```
# OpenAI 및 Anthropic 모델 사용을 위한 라이브러리 설치
!pip install -qU langchain-openai langchain-anthropic

from langchain.chat_models import init_chat_model

# 동일한 모델을 사용해도 되지만, 각 역할에 적합한 모델을 설정하면 결과 품질이 훨씬 좋다.
model_1 = init_chat_model('google_genai:gemini-3.5-flash')
model_2 = init_chat_model('openai:gpt-5.4-mini')
model_3 = init_chat_model('anthropic:claude-sonnet-5')
model_4 = init_chat_model('openai:gpt-5.5')
```

■ 1단계 후보 생성

(333쪽)

문제 해결 후보 생성하기

```
solutions = model_1.invoke(prompt_1.format(
    input='인간의 화성 이주',
```

```
factors='지구와 화성 사이의 거리가 매우 멀어 정기적인 재보급이 어렵다.'
))
# print(solutions.content)
```

■ 2단계 비교 평가

(334쪽)

후보 해결책 비교 평가하기

```
evaluations = model_2.invoke(prompt_2.format(
    solutions=solutions.content,
    factors='지구와 화성 사이의 거리가 매우 멀어 정기적인 재보급이 어렵다.'
))
# print(evaluations.content)
```

■ 3단계 사고 심화

(335쪽)

후보별 사고를 심화해 실행 시나리오 구체화하기

```
deepen_thoughts = model_3.invoke(prompt_3.format(
    solutions=solutions.content,
    factors='지구와 화성 사이의 거리가 매우 멀어 정기적인 재보급이 어렵다.'
))
# print(deepen_thoughts.content)
```

■ 4단계 종합 결론

(335쪽)

최종 우선순위와 실행 계획 도출하기

```
result = model_4.invoke(prompt_4.format(  
    deepen_thoughts=deepen_thoughts.content  
))  
# print(result.content)
```

(336쪽)

최종 결과를 마크다운으로 정리해 출력하기

```
from IPython.display import Markdown  
Markdown(result.content)
```