

함께하는 AI 에이전트 개발

멀티 에이전트 협업 및 리포트 생성 시스템 완성

"여러 전문가 AI가 협업하여 완벽한 리포트를 만든다"

⌚ PROJECT GOAL

AI Research
Assistant 완성

⌚ INSTRUCTOR

Thomas Moon

</> CORE STACK

Python,
OpenAI API

🏆 STATUS

최종 주차

Today's Agenda

- 01 지난 주 복습 및 전체 시스템 점검
- 02 멀티 에이전트 아키텍처 이론
- 03 멀티 에이전트 시스템 구현
- 04 리포트 생성 시스템 구현
- 05 최종 통합 및 실행
- 06 커스터マイ징 가이드 & 향후 학습
- 07 마무리 - 5주간의 여정 회고

5-Week Master Roadmap



01

SECTION

REVIEW

지난 주 복습 및 현황 점검

4주차 성과 점검과 개선 포인트 확정

 현재 프로젝트 구조 & 핵심 클래스

 Project Explorer

```

ai-research-assistant/
  main.py # CLI 진입점
  config/
    prompts.py # 시스템 프롬프트
    settings.py # 설정값
  src/
    conversation_manager.py # Week 1
    search_agent.py # Week 2
    memory_manager.py # Week 3
    task_planner.py # Week 4
    react_engine.py # Week 4
    quality_manager.py # 품질 평가
    loop_prevention.py # 루프 방지
    orchestrator.py # 전체 조율
  tools/
    tool_definitions.py # 도구 정의
    web_search.py # 검색 함수
  utils/
    embeddings.py # 임베딩
  data/

```

 Key Classes Implemented (Total 7)
Conversation Manager

Week 1

대화 관리 & 인터페이스 (OpenAI)

Search Agent

Week 2

웹 검색 및 정보 수집 (Tavily)

Memory Manager

Week 3

벡터 DB 기반 지식 관리 (Chroma)

Quality Manager

Week 4

결과 품질 평가 및 점수화

Task Planner

Week 4

작업 분해 및 계획 수립

ReAct Engine

Week 4

자율 추론-실행 순환 엔진

Autonomous Orchestrator

Week 4

전체 시스템 조율 (Loop Prevention 포함)

⑤ 5주차 목표 - 최종 시스템 완성

Target Goal



Core Objective

"여러 전문 에이전트가 협업하여,
주제를 받으면 자동으로 전문 리포트를 생성하는 시스템"

★ 5주차 새로운 기능

- ✓ 역할별 전문 에이전트 도입
Researcher, Analyzer, Writer, Critic
- ✓ Research Coordinator 협업 관리
에이전트 간의 작업 흐름 및 데이터 전달 조율
- ✓ 구조화된 리포트 자동 생성
Markdown 및 HTML 형식으로 최종 결과물 변환
- ✓ 반복 개선 루프 (Feedback Loop)
Critic 피드백 기반 리포트 품질 향상 프로세스
- ✓ 최종 파일 저장 및 미리보기
결과물 로컬 저장 및 검토 기능

▣ 구현할 핵심 컴포넌트



ResearchCoordinator

전체 에이전트 협업 조율 및 관리

ResearchAgent

정보 수집 전문

AnalysisAgent

데이터 분석 전문

ReportWriter

리포트 작성 전문

QualityCritic

품질 검증 전문

BaseAgent

에이전트 공통 추상 클래스

최종 디렉토리 및 파일 구성

Updated for Week 05 Final

Project Explorer

```

ai-research-assistant/
  main.py # 시스템 진입점(최종수정)
  config/
    prompts.py # 프롬프트 추가(에이전트별)
    settings.py
  src/
    orchestrator.py # Week 4
    research_coordinator.py Week 5 ★
    report_formatter.py Week 5 ★
    agents/ Week 5 ★
      __init__.py
      base_agent.py # 추상 클래스
      research_agent.py # 정보 수집
      analysis_agent.py # 분석
      report_writer.py # 작성
      quality_critic.py # 평가
    tools/
    utils/
  data/
    reports/ Week 5 ★
  tests/
    test_agents.py Week 5 ★
    test_full_pipeline.py Week 5 ★

```

★ Week 5 Major Updates

Research Coordinator

New

멀티 에이전트 협업 조율자

- 전체 워크플로우 관리 (Research → Analysis → Write)
- 에이전트 간 데이터 전달 및 반복 루프 제어

src/agents

New

역할별 전문 에이전트 모음

- | | |
|-----------------|-----------------|
| ✓ ResearchAgent | ✓ AnalysisAgent |
| ✓ ReportWriter | ✓ QualityCritic |

Report Formatter

New

최종 결과물 출력 및 저장

- Markdown (.md) 및 HTML 파일 변환
- data/reports/ 폴더에 자동 저장

Integrated Tests

New

전체 파이프라인 검증 테스트 추가

02

SECTION

멀티에이전트 아키텍처 이론

◎ Key Learning Points

- ✓ Multi-Agent System
- ✓ Communication Protocol
- ✓ Role Division
- ✓ Collaboration Pattern

2025년 현재, AI 에이전트는 "혼자 일하는 도구"에서 "함께 협업하는 팀"으로 진화하고 있습니다.
이 섹션에서는 업계 최신 표준과 이론을 살펴본 뒤, 우리 프로젝트에 적용할 핵심 개념을 정리합니다.

THEORY

AI FUNDAMENTALS

Stage 1: 단일 챗봇

2023

"질문 → 답변"
단순 질의응답 처리



Week 1

Stage 2: 도구 에이전트

2024

"판단 → 도구 → 결과"
외부 API 및 데이터 연동



Week 2~4

Stage 3: 멀티 에이전트

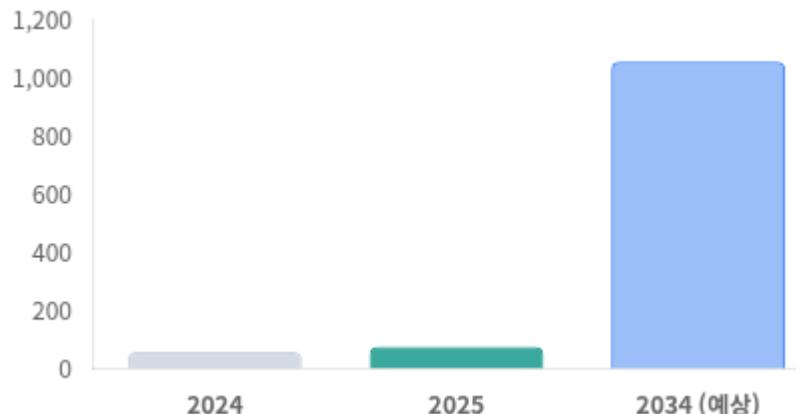
2025

"역할 분담 → 협업 → 검증"
자율적 팀워크 시스템



Week 5

글로벌 시장 규모



Source: Global Market Insights

프로덕션 도입

- 78% 기업이 AI 도구 사용 중
- 29% 프로덕션 운영 중
- 44% 1년 내 도입 예정
- 46% 미도입 시 경쟁 뒤처짐 우려

프로토콜 생태계

- 13,000+ MCP 서버 (GitHub)
- 9,700만+ SDK 다운로드/월
- 150+ A2A 지원 조직

"2026년까지 거의 모든 비즈니스 앱에 AI 어시스턴트가 탑재되며, 이후 1년 내 40%가 특화 에이전트를 통합할 것"

- Gartner 전망

“ A2A란?

서로 다른 플랫폼·프레임워크 에이전트들이 발견·소통·협업할 수 있게 하는 개방형 프로토콜

③ 타임라인**2025.04**

Google이 50+ 파트너사와 발표
Google Cloud Next

2025.06

Linux Foundation에 기증
벤더 중립 거버넌스 확보

2025.07

v0.3 릴리스
gRPC 지원, 보안 카드 서명

2025.12

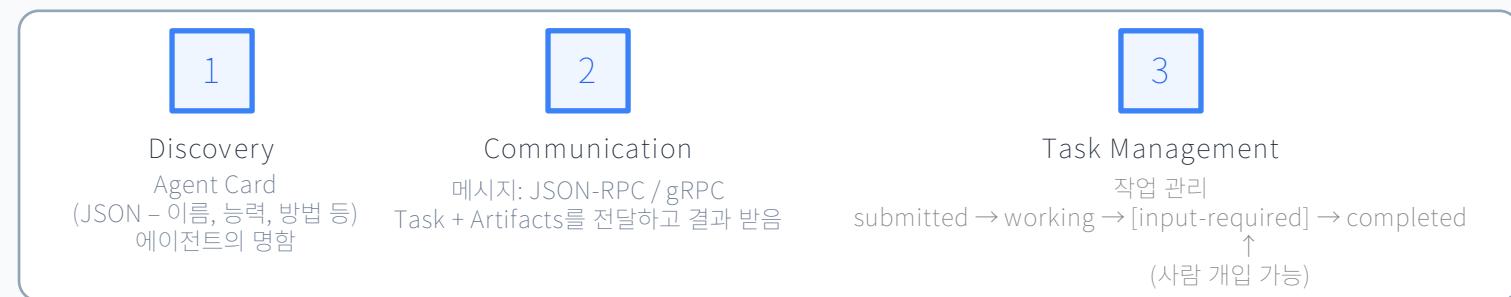
Agentic AI Foundation 출범
150+ 조직 참여 (Adobe, SAP 등)

150+

Organizations

Native

Support in GCP

**5대 설계 원칙**

원칙	설명
능력 존중	에이전트를 "도구"로 격하하지 않고, 독립된 "동료"로 소통
표준 활용	HTTP, SSE, JSON-RPC 등 이미 검증된 웹 기술 기반
기본 보안	엔터프라이즈급 인증/인가 지원 (OAuth 등)
장기 실행	몇 시간~며칠 걸리는 작업도 상태 추적 가능
불투명성	내부 메모리·도구·로직을 노출하지 않고 협업

MCP: 폭발적 커뮤니티 채택 (즉시 사용)

A2A: 엔터프라이즈 중심 확산 (인프라 필요)

"MCP로 시작하고, 규모가 커지면 A2A를 추가한다"

💡 실제 사용 사례:

- Tyson Foods + Gordon Food Service: A2A로 제품 데이터 공유, 공급망 최적화
- Twilio: A2A 확장으로 Latency Aware Agent Selection 구현
- ServiceNow: AI Agent Fabric에 A2A 통합 → 크로스 시스템 자동화

MCP + A2A 전체 에이전트 아키텍처 조감도

Architecture v2.0



■ 프로토콜 한눈에 비교

구분	MCP	A2A
목적	에이전트↔ 도구/데이터	에이전트↔ 에이전트
비유	USB-C 포트	인터넷 (HTTP)
방향	수직적 (도구 접근)	수평적 (동료 협업)
제공자	Anthropic (2024.11)	Google (2025.04)
강의 차수	Week 2~4 (도구 통합)	Week 5 (멀티 에이전트)

“ MAS란?

특정 역할을 가진 여러 에이전트가 상호 통신하며 공동의 목표를 달성하는 시스템

👤 인간 조직과의 비유 (Analogy)		
인간 조직	우리 프로젝트 (MAS)	A2A 대응 개념
팀장 (Lead)	Research Coordinator	Client Agent (조율자)
리서처	Research Agent	Remote Agent (전문가)
분석가	Analysis Agent	Remote Agent (전문가)
라이터	Report Writer	Remote Agent (전문가)
QA 담당자	Quality Critic	Remote Agent (검증자)
이력서	Agent의 System_prompt	Agent Card (능력 명세)

⚖️ 단일 vs 멀티 에이전트 비교		
항목	단일 에이전트 (week 4)	멀티 에이전트 (week 5)
역할	모든 작업을 하나가 처리	전문 역할별 분담
프롬프트	범용 프롬프트 1개	역할별 최적화 프롬프트
품질 관리	자기 평가 (Self-review)	독립적 Critic 검증
확장성	프롬프트 복잡도 ↑	에이전트 추가로 해결
비용	API 호출 적음	API 호출 많음
복잡도	단순	통신/조율 필요
업계 대응	단일 MCP/Agent	MCP + A2A 생태계

❓ 왜 여러 에이전트가 필요한가?

⭐ 전문성
역할 특화 프롬프트/도구

✅ 품질
독립적 Critic 검증

✖ 확장성
에이전트 추가로 기능 확장

✖ 병렬 처리
독립 작업 동시 실행

❗️ 불투명성 (Opacity)
내부 구현 노출 없이 협업 (A2A 원칙)

패턴 1: 순차 (Sequential)

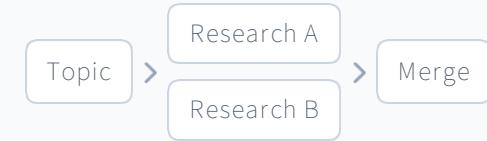
기본



- 가장 단순하고 안정적인 파이프라인
- 각 단계 출력이 다음 단계 입력
- ★ 우리 프로젝트 기본 구조

패턴 2: 병렬 (Parallel)

속도



- 독립적 작업의 동시 실행 (Fan-out)
- 전체 처리 시간 단축

패턴 3: 반복 (Iterative Refinement)

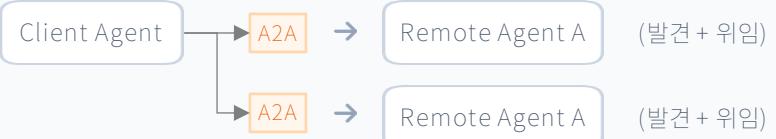
품질



- 품질 기준 충족 시까지 반복
- ★ 우리 프로젝트 검증 로직 (7점 기준, 최대 3회)

패턴 4: 위임 (Delegation)

확장



- 에이전트가 Agent Card로 적합한 동료를 “발견”
- 작업을 위임하고 결과를 수집
- 플랫폼·프레임워크가 달라도 협업 가능
- 차세대 패턴 (A2A 프로토콜 기반)

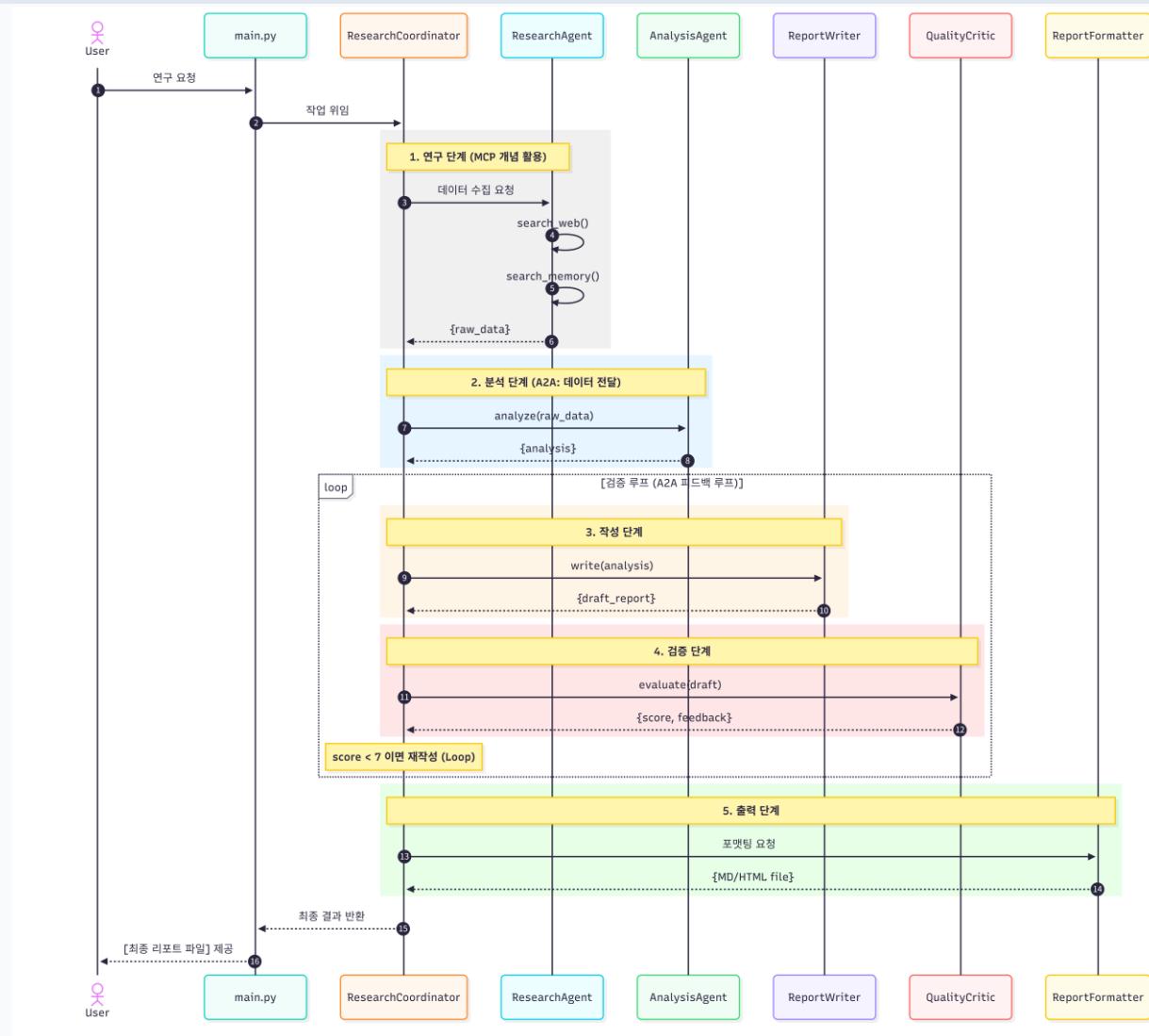
우리가 구현하는 협업 흐름



에이전트 역할 정의

에이전트	역할	입력	출력	사용 도구
Research Agent	정보 수집 전문	연구 주제	검색 결과 모음	Search Agent, Memory Manager
Analysis Agent	분석 전문	검색 결과	인사이트, 트랜드	LLM 분석
Report Writer	리포트 작성 전문	분석 결과	구조화된 리포트	LLM 작성
Quality Critic	품질 검증 전문	리포트 초안	평가+피드백	LLM 평가

☰ 시퀀스 다이어그램 (Sequence Diagram)



프레임워크 비교

항목	Crew AI	Auto Gen (MS)	Lang Graph	Meta GPT	Open AI – Agents SDK
개발사	Crew AI Inc.	Microsoft Research	Lang Chain	Deep Wisdom	Open AI
핵심 패러다임	역할 기반 (Crew)	대화 기반 (Group Chat)	상태 그래프 (DAG)	SOP 기반	함수 호출 기반
강점	직관적, 빠른 시작	유연한 비동기 대화	복잡한 분기/상태	SW 개발 자동화	Open AI 생태계 통합
난이도	★★	★★★	★★★★★	★★★	★★
MCP 지원도	✓	✓	✓	제한적	✓
A2A 지원	✓ (공식)	➡ 진행중	✓ (공식)	✗	✗

어떤 프레임워크를 쓸까?

역할 기반 팀 작업 (우리 프로젝트와 가장 유사)
→ CrewAI

복잡한 분기/상태 관리 (프로덕션 워크플로우)
→ LangGraph

대화형 멀티턴 협업 (브레인스토밍, 토론)
→ AutoGen

소프트웨어 개발 자동화 (PRD→설계→코드→테스트)
→ MetaGPT

OpenAI 스택 옮기 (빠른 프로토타이핑)
→ OpenAI Agents SDK

우리의 접근: 직접 구현 (No Framework)

- ▣ 원리 체득: 에이전트 통신, 상태 관리 등 내부 동작 완전 이해, MCP/A2A의 개념을 코드로 체험
- ▣ 의존성 제로: 프레임워크 의존성 없이 가벼운 시스템
- ▣ 확장 용이성: 학습 후 프로덕션 프레임워크로의 전환이 쉬움

03

SECTION

HANDS-ON

IMPLEMENTATION

HANDS-ON

PART 1

멀티 에이전트 시스템 구현

◎ Key Learning Points

- ✓ Base Agent 추상 클래스 정의
- ✓ 3개 전문 에이전트 구현
- ✓ 에이전트 협업 로직 구현
- ✓ 에이전트별 시스템 프롬프트 작성

Keywords: ● Base Agent ● Research Agent ● AnalysisAgent ● Research Coordinator

Part 1 작업 목표

구현 계획 및 파일 목록

구현할 파일 목록

#	파일 경로	클래스명	설명
1	src/agents/base_agent.py	BaseAgent	에이전트 공통 추상 클래스 (Interface)
2	src/agents/research_agent.py	ResearchAgent	정보 수집 에이전트 (Search + Memory)
3	src/agents/analysis_agent.py	AnalysisAgent	분석 에이전트 (Insight Extraction)
4	src/agents/quality_critic.py	QualityCritic	품질 검증 에이전트 (Evaluation)
5	src/research_coordinator.py	ResearchCoordinator	전체 협업 조율 (Orchestration)
6	config/prompts.py	-	에이전트별 시스템 프롬프트 추가

구현 작업 흐름



Step 01

BaseAgent 추상 클래스 정의



Step 02

3개 전문 에이전트 구현
(Research, Analysis, Critic)



Step 03

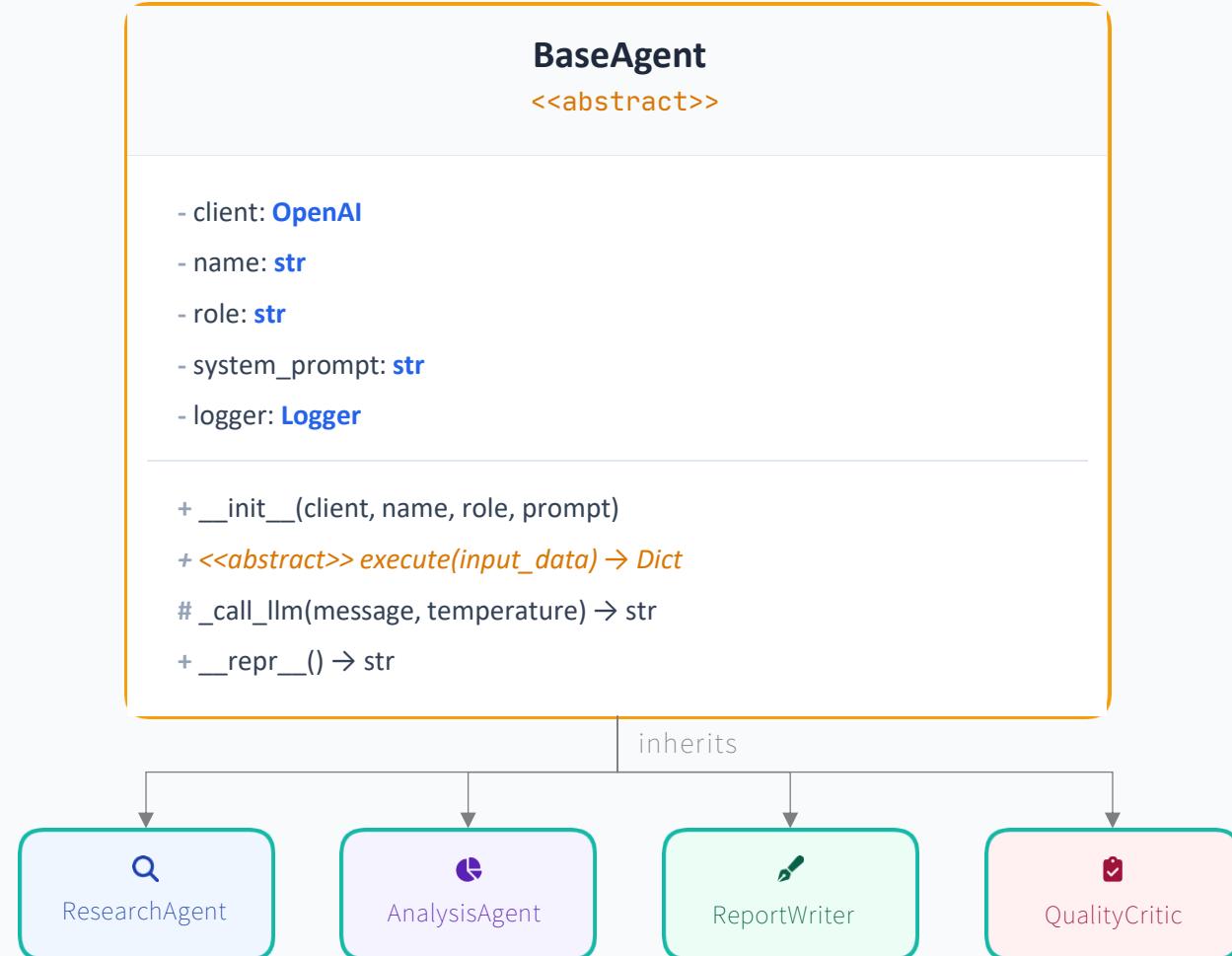
ResearchCoordinator로 협업 로직 구현



Step 04

단위 테스트

BaseAgent 추상 클래스 설계



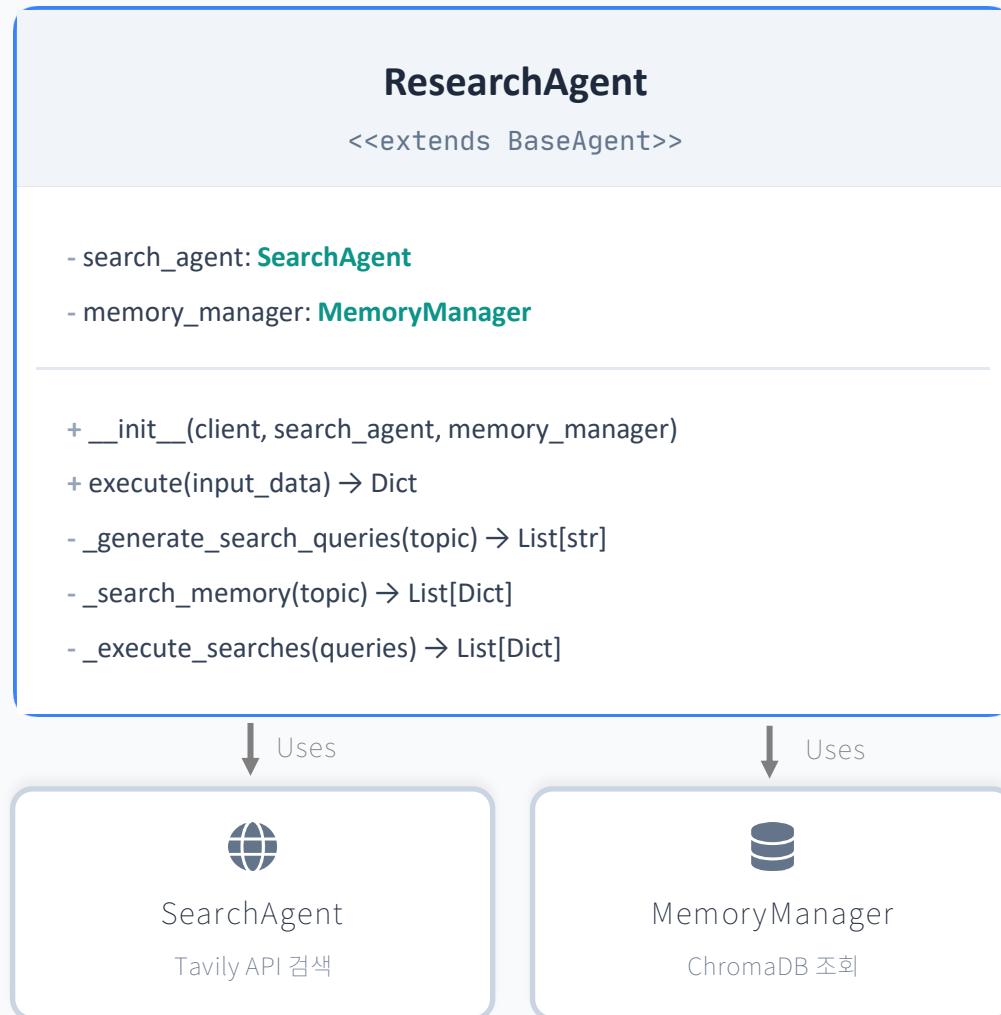
class **BaseAgent(ABC)**

핵심 설계 원칙

- 추상화 (Abstraction)
execute()를 반드시 구현하도록 강제 (ABC)
- 재사용 (Reusability)
_call_llm()으로 LLM 호출 로직 공유 (protected)
- 일관성 (Consistency)
모든 에이전트가 동일한 인터페이스 (다형성)
- 로깅 (Logging)
에이전트별 독립 로거 (Agent:{name})

🔍 ResearchAgent 설계

class ResearchAgent(BaseAgent)



↔ 입출력 흐름 (I/O Flow)

Input

```
{  
    "topic":  
        "AI 반도체"  
}
```



Execute()

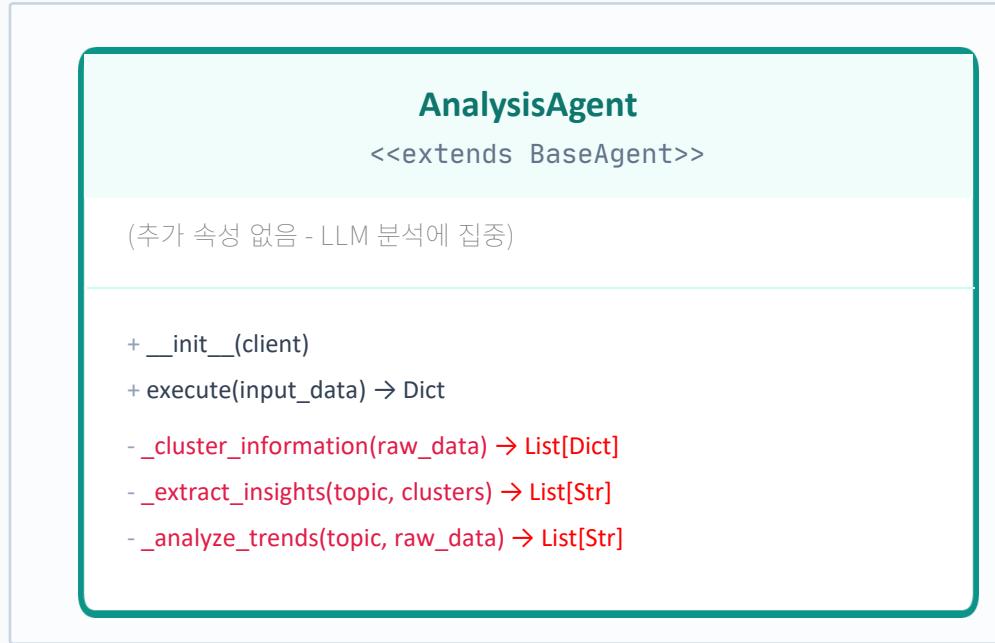
Output

```
{  
    "topic": str,  
    "memory_data": [],  
    "search_data": [],  
    "source_count": N,  
    "queries_used": []  
}
```

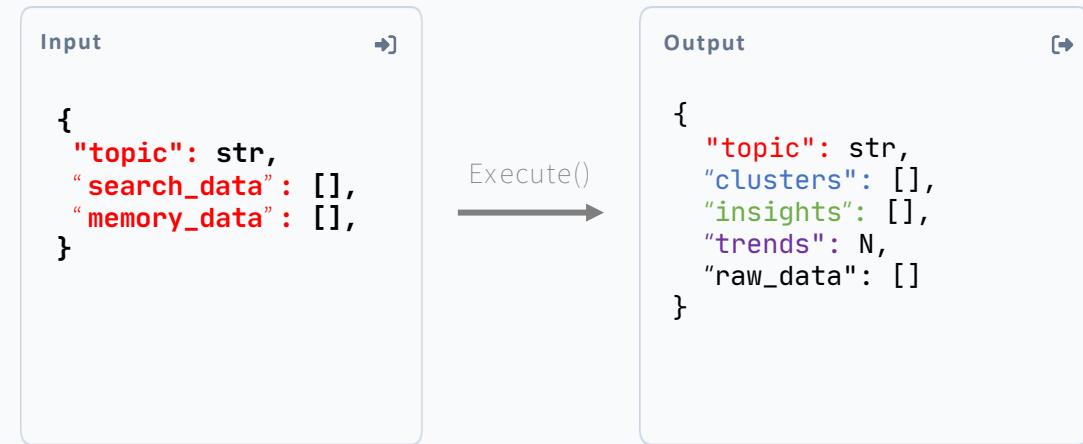


AnalysisAgent 설계

```
class AnalysisAgent(BaseAgent)
```

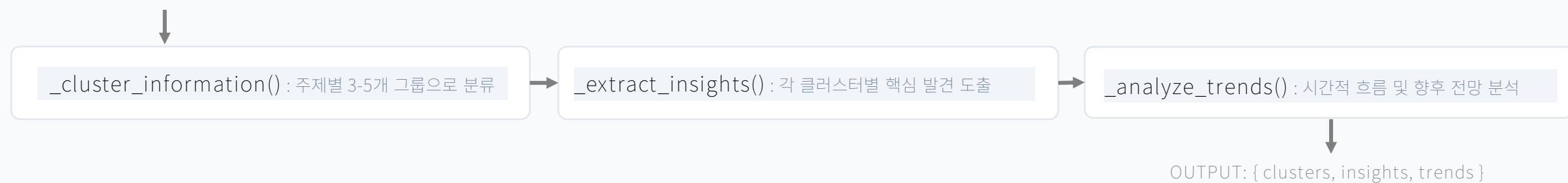


↔ 입출력 흐름 (I/O Flow)



분석 파이프라인 (Analysis Pipeline)

INPUT: Search_data (10-20건)

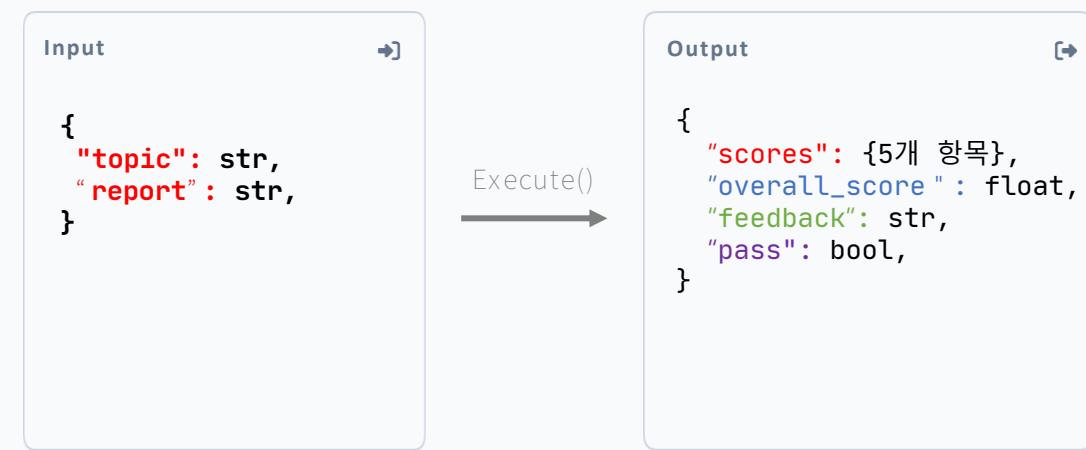


✓ QualityCritic 설계

CLASS QUALITYCRITIC(BASEAGENT)
QUALITYCRITIC(BASEAGENT)



↔ 입출력 흐름 (I/O Flow)



★ 5가지 평가 기준 (5-Dimension Scoring)

Completeness 주제를 충분히 다루었는가?

1-10점

Accuracy 정보가 정확하고 사실인가?

1-10점

Clarity 이해하기 쉽고 명확한가?

1-10점

Structure 논리적 구조를 갖추었는가?

1-10점

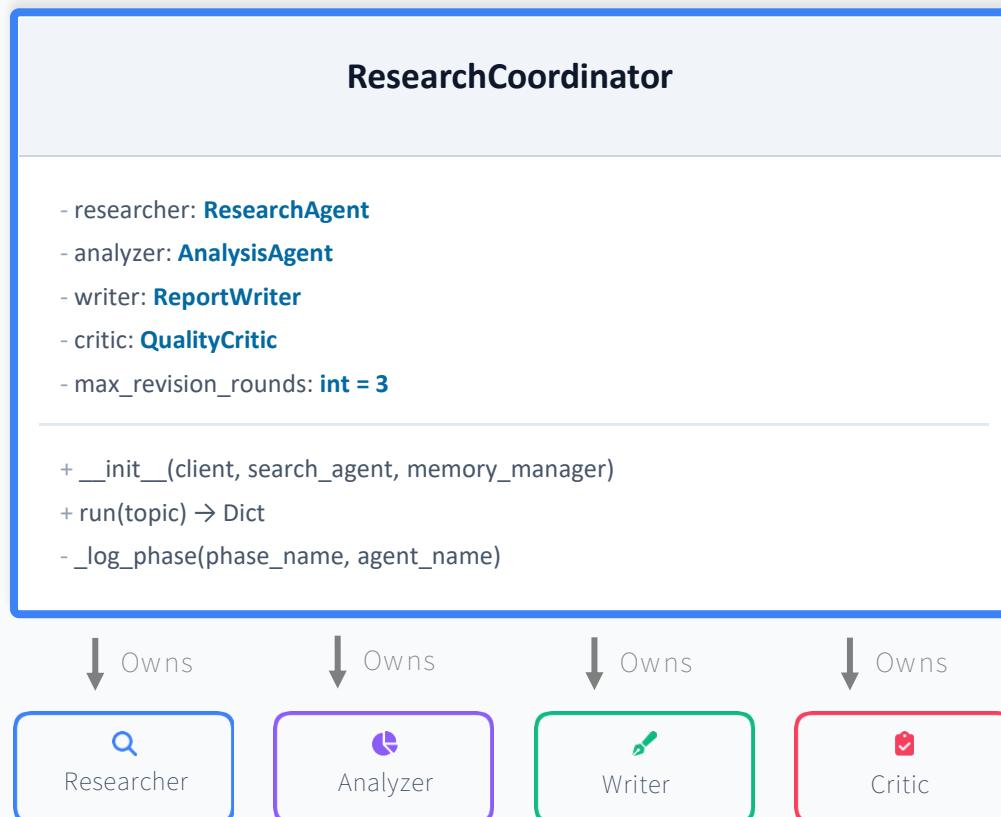
Source Quality 출처가 신뢰할 수 있는가?

1-10점

Overall 평균 7점 이상 → ✓ 통과 |
평균 7점 미만 → ✗ 재작성 요청

ResearchCoordinator 설계

class ResearchCoordinator



▶ 오케스트레이션 실행 흐름 (run)

Run(topic: str)

Phase 1: Research

researcher.execute({"topic": topic}) → research_data



Phase 2: Analysis

analyzer.execute(research_data) → analysis



Phase 3: Write + Review Loop

writer.execute({topic, analysis, feedback}) → draft

critic.execute({topic, draft}) → review (score, feedback)

review.pass == True? — Yes —→ break

|
No —→ feedback = review.feedback

반복: Max 3 회

Return: {"report": str, "score": float}

실습 Part1 구현 프롬프트 - 1

config/prompts.py 파일의 맨 아래에 Week 5용 에이전트 프롬프트를 추가해주세요.

기존 코드를 수정하지 말고, 파일 맨 아래에 다음 5개 프롬프트 상수를 추가합니다:

1. RESEARCH_AGENT_PROMPT

- 역할: 정보 수집 전문가
- 주어진 주제에 대해 3-5개의 검색 쿼리를 생성하는 능력
- 검색 결과를 구조화하여 정리하는 능력
- 출력 형식: JSON (queries 배열)

2. ANALYSIS_AGENT_PROMPT

- 역할: 데이터 분석 전문가
- 수집된 정보를 주제별로 클러스터링하는 능력
- 핵심 인사이트 3-5개를 도출하는 능력
- 트렌드와 패턴을 식별하는 능력
- 출력 형식: JSON (clusters, insights, trends)

3. REPORT_WRITER_PROMPT

- 역할: 리포트 작성 전문가
- 분석 결과를 바탕으로 구조화된 리포트 작성
- 리포트 구조: 제목, Executive Summary, 서론, 본론(주제별 섹션), 트렌드 및 전망, 결론, 참고자료
- Markdown 형식으로 작성
- 피드백이 있을 경우 해당 부분을 개선하여 재작성

4. CRITIC_AGENT_PROMPT

- 역할: 품질 검증 전문가
- 5가지 기준으로 평가: completeness, accuracy, clarity, structure, source_quality
- 각 항목 1-10점 채점
- overall 점수 = 5개 항목의 평균
- 구체적인 개선 피드백 제공
- 출력 형식: JSON

5. COORDINATOR_PROMPT

- 역할: 리서치 프로젝트 총괄 코디네이터
- 연구 주제 분석 및 검색 전략 수립
- 각 에이전트에게 적절한 지시 전달
- 최종 결과물 품질 관리

각 프롬프트는 기존 prompts.py의 스타일(docstring, 섹션 구분 주석)과 일관되게 작성해주세요.

실습 Part1 구현 프롬프트 - 2

src/agents/ 디렉토리를 생성하고, BaseAgent 추상 클래스를 구현해주세요.

1. src/agents/__init__.py 생성

- BaseAgent, ResearchAgent, AnalysisAgent, ReportWriter, QualityCritic을 임포트
- __all__ 리스트 정의
- (아직 존재하지 않는 클래스는 주석 처리해두고, 구현 후 주석 해제)

2. src/agents/base_agent.py 생성

BaseAgent 클래스 요구사항:

- ABC를 상속받는 추상 클래스
- __init__(self, client: OpenAI, name: str, role: str, system_prompt: str)
 - self.client = client (OpenAI 인스턴스)
 - self.name = name
 - self.role = role
 - self.system_prompt = system_prompt
 - self.logger = logging.getLogger(f"Agent:{name}")
- @abstractmethod execute(self, input_data: Dict[str, Any]) -> Dict[str, Any]
 - 각 에이전트가 반드시 구현해야 하는 핵심 로직
- _call_llm(self, user_message: str, temperature: float = 0.7) -> str
 - OpenAI API 호출 공통 메서드
 - model은 config.settings.DEFAULT_MODEL 사용
 - messages: system_prompt + user_message
 - 예러 발생 시 로깅 후 빈 문자열 반환
- _call_llm_json(self, user_message: str, temperature: float = 0.3) -> Dict
 - JSON 응답 전용 LLM 호출
 - response_format={"type": "json_object"} 사용
 - JSON 파싱 실패 시 빈 dict 반환
- __repr__(self) -> str
 - f"<{self.name} ({self.role})>" 형식

기존 프로젝트의 코딩 스타일을 따라주세요:

- 타입 힌트 사용
- docstring 포함
- logging 사용

실습 Part1 구현 프롬프트 - 3

src/agents/research_agent.py에 ResearchAgent 클래스를 구현해주세요.

ResearchAgent는 BaseAgent를 상속하며, 기존 SearchAgent(src/search_agent.py)와 MemoryManager(src/memory_manager.py)를 활용하여 정보를 수집합니다.

클래스 요구사항:

- BaseAgent를 상속
- __init__(self, client: OpenAI, search_agent: SearchAgent, memory_manager: MemoryManager)
 - super().__init__()으로 name="Researcher", role="정보 수집 전문가", system_prompt=RESEARCH_AGENT_PROMPT 전달
 - self.search_agent = search_agent
 - self.memory_manager = memory_manager
- execute(self, input_data: Dict) -> Dict
 - input_data에서 "topic" 키로 주제를 받음
 - 1단계: _generate_search_queries(topic) → 3-5개 검색 쿼리 생성 (LLM 사용)
 - 2단계: _search_memory(topic) → 메모리에서 기존 지식 검색
 - 3단계: _execute_searches(queries) → 각 쿼리로 웹 검색 실행
 - 반환: {"topic": str, "memory_data": list, "search_data": list, "source_count": int, "queries_used": list}
- _generate_search_queries(self, topic: str) -> List[str]
 - _call_llm_json()으로 RESEARCH_AGENT_PROMPT 활용하여 쿼리 생성
 - 반환: ["query1", "query2", ...] (3-5개)
 - 실패 시 [topic] 반환 (최소 1개 보장)
- _search_memory(self, topic: str) -> List[Dict]
 - self.memory_manager.search_memory(topic, top_k=3) 호출
 - 에러 발생 시 빈 리스트 반환

- _execute_searches(self, queries: List[str]) -> List[Dict]

- 각 query에 대해 self.search_agent.search(query) 호출
- 결과를 self.search_agent.format_for_llm(result)로 변환하여 수집
- 각 검색 결과를 {"query": query, "result": formatted_result} 형태로 저장
- 실패한 검색은 건너뛰기 (로그 남기기)

주의사항:

- 기존 SearchAgent의 search() 메서드와 format_for_llm() 메서드를 그대로 활용합니다
- 기존 MemoryManager의 search_memory() 메서드를 그대로 활용합니다
- 새로운 API를 만들지 말고, 기존 인터페이스를 재사용합니다

실습 Part1 구현 프롬프트 - 4

src/agents/analysis_agent.py에 AnalysisAgent 클래스를 구현해주세요.

AnalysisAgent는 BaseAgent를 상속하며, 수집된 데이터를 분석하여 인사이트를 도출합니다.

클래스 요구사항:

- BaseAgent를 상속
- __init__(self, client: OpenAI)
 - super().__init__()으로 name="Analyzer", role="데이터 분석 전문가", system_prompt=ANALYSIS_AGENT_PROMPT 전달
- execute(self, input_data: Dict) -> Dict
 - input_data에서 "topic", "search_data", "memory_data"(optional) 키를 받음
 - 1단계: 검색 데이터를 하나의 텍스트로 통합 (_prepare_data)
 - 2단계: _analyze(topic, combined_data) → LLM으로 분석 수행
 - 반환: {"topic": str, "clusters": list, "insights": list, "trends": list, "raw_data": 원본 search_data}
- _prepare_data(self, search_data: List[Dict], memory_data: List[Dict]) -> str
 - search_data와 memory_data를 하나의 분석용 텍스트로 통합
 - 각 데이터 항목을 번호 매기기하여 정리
 - "==== 웹 검색 결과 ===" 와 "==== 기존 지식 ===" 섹션으로 구분
- _analyze(self, topic: str, data: str) -> Dict
 - _call_llm.json()으로 분석 수행
 - 프롬프트에 topic과 data를 전달하여 clusters, insights, trends를 JSON으로 반환 요청
 - 실패 시 기본값 반환: {"clusters": [], "insights": ["분석 실패"], "trends": []}

주의사항:

- AnalysisAgent는 외부 도구(검색, DB)를 사용하지 않고, 순수 LLM 분석만 수행합니다
- 입력 데이터가 너무 길 경우를 대비해 _prepare_data에서 각 항목을 500자로 truncate합니다

실습 Part1 구현 프롬프트 - 5

src/agents/report_writer.py에 ReportWriter 클래스를 구현해주세요.

ReportWriter는 BaseAgent를 상속하며, 분석 결과를 바탕으로 구조화된 리포트를 작성합니다.

클래스 요구사항:

- BaseAgent를 상속
- __init__(self, client: OpenAI)
 - super().__init__()으로 name="Writer", role="리포트 작성 전문가", system_prompt=REPORT_WRITER_PROMPT 전달
- execute(self, input_data: Dict) -> Dict
 - input_data에서 "topic", "analysis" (AnalysisAgent 출력), "feedback" (optional, QualityCritic 피드백) 키를 받음
 - feedback이 있으면: _revise_report(topic, analysis, previous_report, feedback)
 - feedback이 없으면: _write_initial_report(topic, analysis)
 - 반환: {"topic": str, "report": str(Markdown), "word_count": int, "has_revision": bool}
- _write_initial_report(self, topic: str, analysis: Dict) -> str
 - _call_llm()으로 REPORT_WRITER_PROMPT 활용
 - 프롬프트에 topic, analysis["clusters"], analysis["insights"], analysis["trends"]를 전달
 - Markdown 형식의 리포트 반환
 - temperature=0.7 사용
- _revise_report(self, topic: str, analysis: Dict, previous_report: str, feedback: str) -> str
 - _call_llm()으로 수정 요청
 - 프롬프트에 이전 리포트 + 피드백을 포함하여 개선된 버전 요청
 - temperature=0.7 사용

주의사항:

- 리포트는 반드시 Markdown 형식으로 작성되어야 합니다
- execute() 결과의 "report" 키에 전체 Markdown 텍스트가 들어갑니다
- word_count는 report.split() 길이로 계산합니다

실습 Part1 구현 프롬프트 - 6

src/agents/quality_critic.py에 QualityCritic 클래스를 구현해주세요.

QualityCritic은 BaseAgent를 상속하며, 기존 QualityManager(src/quality_manager.py)와는 별도로 리포트 전용 5항목 평가를 수행합니다.

⚠️ 기존 QualityManager(3항목: completeness, accuracy, relevance)는 수정하지 않습니다.
QualityCritic은 리포트 품질에 특화된 새로운 5항목 평가 에이전트입니다.

클래스 요구사항:

- BaseAgent를 상속
- PASS_THRESHOLD = 7.0 (클래스 변수)
- __init__(self, client: OpenAI)
 - super().__init__()으로 name="Critic", role="품질 검증 전문가", system_prompt=CRITIC_AGENT_PROMPT 전달

```

- execute(self, input_data: Dict) -> Dict
  - input_data에서 "topic", "report" 키를 받음
  - _evaluate_report(topic, report) 호출
  - 반환:
  {
    "scores": {
      "completeness": float, # 완성도
      "accuracy": float, # 정확성
      "clarity": float, # 명확성
      "structure": float, # 구조
      "source_quality": float # 출처 품질
    },
    "overall_score": float, # 5개 평균
    "feedback": str, # 개선 피드백
    "pass": bool # overall_score >= PASS_THRESHOLD
  }

```

- _evaluate_report(self, topic: str, report: str) -> Dict
- _call_llm_json()으로 CRITIC_AGENT_PROMPT 활용
- 5개 항목 각각 1-10점으로 평가 요청
- overall이 없으면 5개 평균으로 계산
- 실패 시 기본값 반환 (all 5.0, pass=False)

주의사항:

- 기존 QualityManager(src/quality_manager.py)는 절대 수정하지 마세요
- QualityCritic은 src/agents/ 디렉토리에 새로 만드는 별도 클래스입니다

실습 Part1 구현 프롬프트 - 7

tests/test_week5_part1.py에 Part 1의 단위 테스트를 작성해주세요.

테스트는 실제 API 호출 없이 클래스 구조와 인터페이스를 검증합니다.

테스트 항목 (총 8개):

1. test_base_agent_is_abstract()

- BaseAgent를 직접 인스턴스화하면 TypeError가 발생하는지 확인

2. test_base_agent_subclass_interface()

- BaseAgent를 상속한 임시 클래스(DummyAgent)를 만들어서 execute()를 구현하면 정상 생성되는지 확인

- name, role, system_prompt 속성이 올바르게 설정되는지 확인

3. test_research_agent_creation()

- ResearchAgent가 올바르게 생성되는지 확인
- name이 "Researcher"인지 확인
- search_agent, memory_manager 속성이 설정되는지 확인
- Mock 객체로 OpenAI, SearchAgent, MemoryManager를 대체

4. test_analysis_agent_creation()

- AnalysisAgent가 올바르게 생성되는지 확인
- name이 "Analyzer"인지 확인

5. test_report_writer_creation()

- ReportWriter가 올바르게 생성되는지 확인
- name이 "Writer"인지 확인

6. test_quality_critic_creation()

- QualityCritic이 올바르게 생성되는지 확인
- name이 "Critic"인지 확인
- PASS_THRESHOLD가 7.0인지 확인

7. test_prompts_exist()

- config.prompts에 RESEARCH_AGENT_PROMPT, ANALYSIS_AGENT_PROMPT, REPORT_WRITER_PROMPT, CRITIC_AGENT_PROMPT, COORDINATOR_PROMPT가 존재하는지 확인
- 각 프롬프트가 빈 문자열이 아닌지 확인

8. test_agents_package_imports()

- from src.agents import BaseAgent, ResearchAgent, AnalysisAgent, ReportWriter, QualityCritic이 정상 동작하는지 확인

테스트 실행 방법:

```
```bash
pytest tests/test_week5_part1.py -v
````
```

주의사항:

- OpenAI 클라이언트는 unittest.mock.MagicMock으로 대체합니다
- SearchAgent, MemoryManager도 Mock으로 대체합니다
- 실제 API 호출은 하지 않습니다
- 기존 테스트 파일의 스타일(pytest, 한글 설명)을 따릅니다

Part 1 단위 테스트

💡 가상환경 활성화 후, 테스트 스크립트 실행

1 \$ python -m venv venv

2 Win > venv\Scripts\activate

Mac \$ source venv/bin/activate

아래와 같이 나오면, 가상환경에서 동작중인 것임!

(venv)user@pc:~\$

이 상태에서 터미널 창에서
의존성 패키지 업데이트!

pytest tests/test_week5_part1.py -v

⚙️ 단위 테스트 항목

BaseAgent 테스트 (2개)

test_base_agent_creation() - BaseAgent 추상 클래스 생성 및 속성 확인
test_base_agent_call_llm() - _call_llm() 메서드 정상 동작 확인

ResearchAgent 테스트 (3개)

test_research_agent_creation() - ResearchAgent 생성 및 속성 확인
test_research_agent_execute() - execute() 메서드 정상 동작 확인
test_research_agent_generate_queries() - _generate_search_queries()
메서드 확인

AnalysisAgent 테스트 (3개)

test_analysis_agent_creation() - AnalysisAgent 생성 및 속성 확인
test_analysis_agent_execute() - execute() 메서드 정상 동작 확인
test_analysis_agent_prepare_data() - _prepare_data() 메서드 확인

ReportWriter 테스트 (3개)

test_report_writer_creation() - ReportWriter 생성 및 속성 확인
test_report_writer_initial_report() - _write_initial_report() 메서드 확인
test_report_writer_with_feedback() - 피드백을 반영한 리포트 수정 확인

QualityCritic 테스트 (2 개)

test_critic_agent_creation() - Quality Critic 생성 및 속성 확인
test_critic_agent_evaluate() - evaluate() 메서드 정상 동작 및 5개 평가 항목 확인

Part 1 단위 테스트 완료 화면

IMPLEMENTATION CHECK

```
(venv) PS C:\Users\asakh\Documents\GitHub\AI_agent_lecture_final> pytest tests/test_week5_part1.py -v  
===== test session starts =====  
platform win32 -- Python 3.12.10, pytest-9.0.2, pluggy-1.6.0 -- C:\Users\asakh\Documents\GitHub\AI_agent_lecture_final\venv\Scripts\python.exe  
cachedir: .pytest_cache  
rootdir: C:\Users\asakh\Documents\GitHub\AI_agent_lecture_final  
configfile: pytest.ini  
plugins: anyio-4.12.1  
collected 8 items  
  
tests/test_week5_part1.py::test_base_agent_is_abstract PASSED [ 12%]  
tests/test_week5_part1.py::test_base_agent_subclass_interface PASSED [ 25%]  
tests/test_week5_part1.py::test_research_agent_creation PASSED [ 37%]  
tests/test_week5_part1.py::test_analysis_agent_creation PASSED [ 50%]  
tests/test_week5_part1.py::test_report_writer_creation PASSED [ 62%]  
tests/test_week5_part1.py::test_research_agent_creation PASSED [ 37%]  
tests/test_week5_part1.py::test_analysis_agent_creation PASSED [ 50%]  
tests/test_week5_part1.py::test_research_agent_creation PASSED [ 37%]  
tests/test_week5_part1.py::test_research_agent_creation PASSED [ 37%]  
tests/test_week5_part1.py::test_research_agent_creation PASSED [ 37%]  
tests/test_week5_part1.py::test_analysis_agent_creation PASSED [ 50%]  
tests/test_week5_part1.py::test_analysis_agent_creation PASSED [ 50%]  
tests/test_week5_part1.py::test_report_writer_creation PASSED [ 62%]  
tests/test_week5_part1.py::test_quality_critic_creation PASSED [ 75%]  
tests/test_week5_part1.py::test_prompts_exist PASSED [ 87%]  
tests/test_week5_part1.py::test_agents_package_imports PASSED [100%]  
  
===== 8 passed in 2.54s =====
```

04

SECTION

HANDS-ON

IMPLEMENTATION

HANDS-ON PART 2

리포트 생성 시스템 구현

◎ Key Implementation Goals

ResearchCoordinator로 4개 에이전트의 협업을 조율하고, ReportFormatter로 Markdown/HTML 출력 기능을 구현한다.

Keywords: ● Report Writer ● Report Formatter ● Markdown

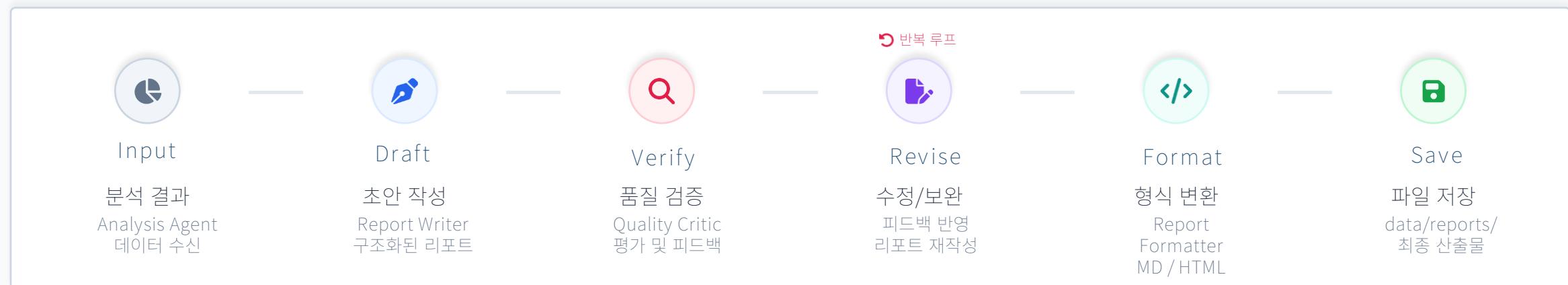
Part 2 작업 목표

리포트 생성 및 저장 시스템

구현할 파일 목록

| # | 파일 경로 | 클래스명 | 설명 |
|---|-----------------------------|-----------------|----------------------------------|
| 1 | src/agents/report_writer.py | ReportWriter | 리포트 작성 에이전트 (초안 작성 및 수정) |
| 2 | src/report_formatter.py | ReportFormatter | 출력 형식 변환 (Markdown / HTML) |
| 3 | config/prompts.py | - | REPORT_WRITER_PROMPT 시스템 프롬프트 추가 |

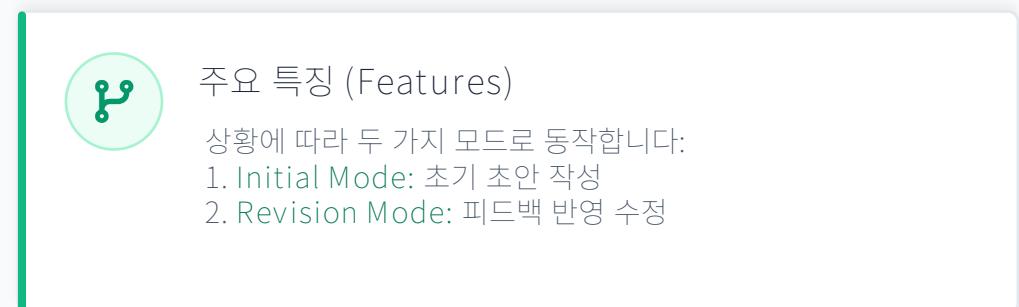
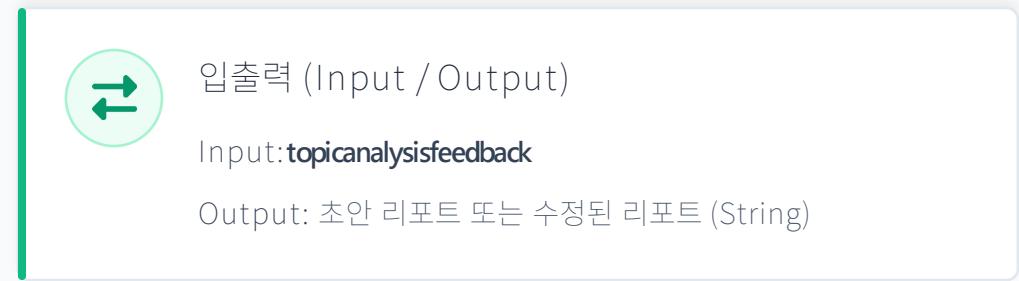
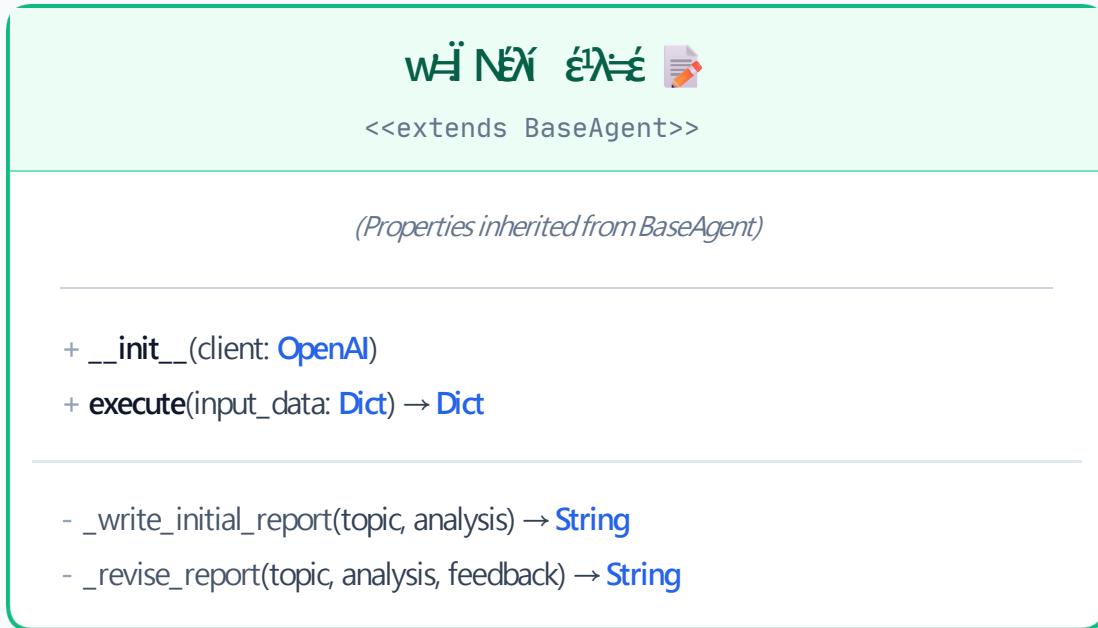
리포트 생성 파이프라인





ReportWriter 클래스 설계

• ReportWriter 는 AI 분석 결과를 Markdown 형식으로 리포트로 작성하는 클래스입니다.



리포트 구조 템플릿

```
template_structure.md
```

```
# [리포트 제목]
## Executive Summary
> 3-5문장으로 핵심 내용 요약
## 1. 서론
- 배경 및 목적
- 연구 범위
## 2. 본론
### 2.1 [주요 주제 1]
- 현황 분석
- 핵심 데이터
### 2.2 [주요 주제 2]
- 시장/기술 동향
- 주요 플레이어
### 2.3 [주요 주제 3]
- 비교 분석
- 강점/약점
## 3. 트렌드 및 전망
- 단기/중기 전망
- 기회와 위험 요소
## 4. 결론
- 핵심 발견 정리
- 제언
## 참고 자료
- [출처 1]: URL
- [출처 2]: URL
```

Standard Report Structure (Markdown)

Executive Summary

바쁜 의사결정권자를 위해 리포트의 핵심 결론과 제언을 3-5문장으로 요약합니다.

본론 (Main Body)

3개의 하위 주제로 나누어 현황, 동향, 비교 분석을 심층적으로 다룹니다. (데이터 중심)

트렌드 및 전망

현재의 분석을 넘어 미래의 기회와 위험 요소를 예측하여 인사이트를 제공합니다.

결론 (Conclusion)

연구 결과의 핵심 발견을 재확인하고 실행 가능한 제언을 제시합니다.

참고 자료 (References)

정보의 신뢰성을 확보하기 위해 사용된 모든 출처와 URL을 명시합니다.

Markdown 형식 변환

class ReportFormatter

```
src/report_formatter.py

class ReportFormatter:
    """리포트 출력 형식 변환"""

    @staticmethod
    def to_markdown(report: str, metadata: Dict) -> str:
        """리포트를 정리된 Markdown으로 변환"""
        header = f"""---
title: {metadata.get('topic', 'Research Report')}
date: {datetime.now().strftime('%Y-%m-%d')}
agent_score: {metadata.get('score', 'N/A')}/10
sources: {metadata.get('source_count', 0)}
---"""

        """
        return header + report

    @staticmethod
    def save_markdown(content: str, filename: str) -> str:
        """Markdown 파일 저장"""
        filepath = f"data/reports/{filename}.md"
        os.makedirs("data/reports", exist_ok=True)
        with open(filepath, "w", encoding="utf-8") as f:
            f.write(content)
        return filepath
```

주요 기능



Markdown 변환

LLM이 생성한 리포트 본문에 메타데이터 헤더(Frontmatter)를 추가하여 완성된 문서로 만듭니다.



메타데이터 관리

문서의 체계적인 관리를 위해 핵심 정보를 자동으로 삽입합니다.

title

date

agent_score

sources



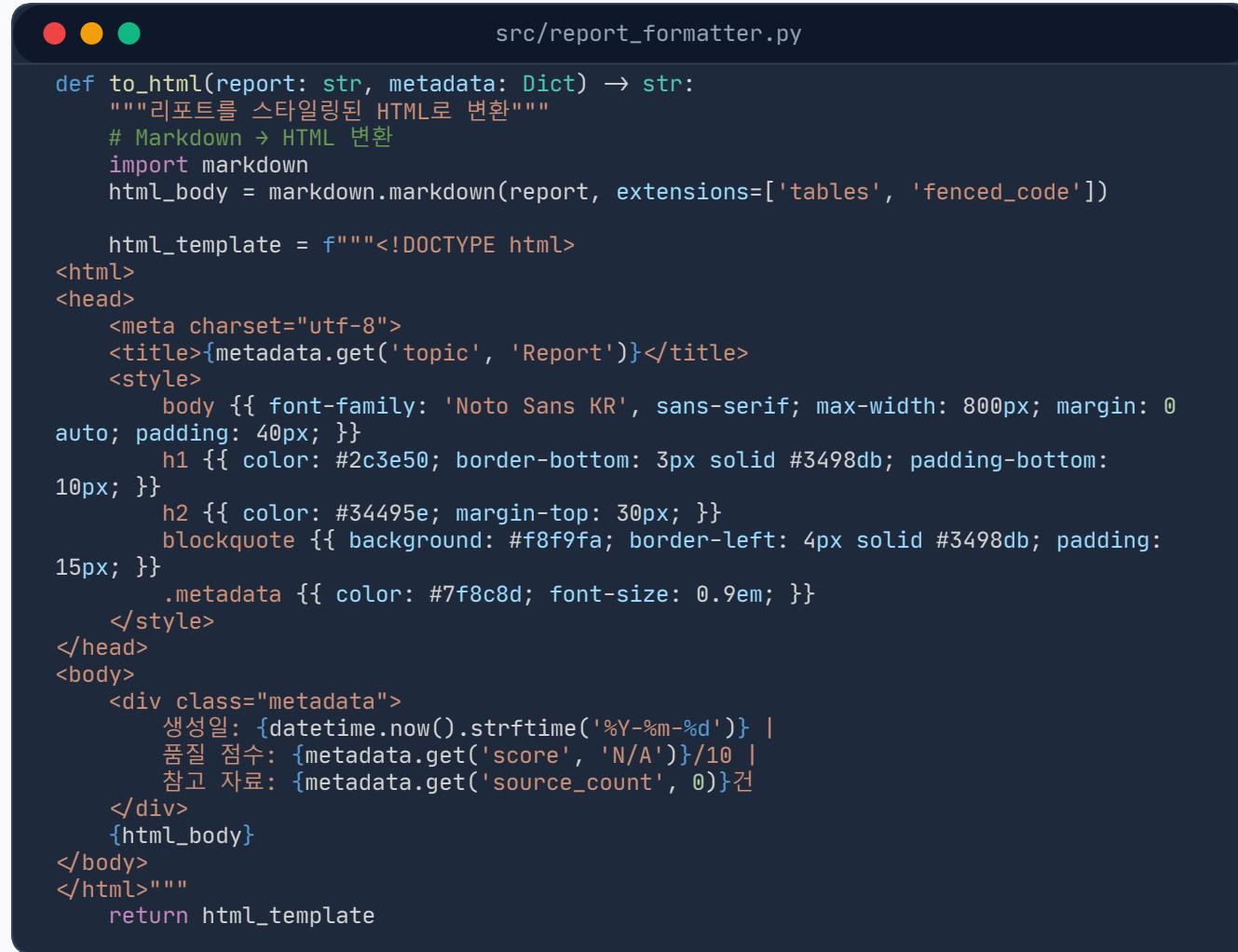
파일 저장 자동화

지정된 경로에 폴더를 자동 생성하고, 인코딩(UTF-8)을 처리하여 파일을 안전하게 저장합니다.

Path: data/reports/*.md

HTML 형식 변환 및 스타일링

class ReportFormatter



The screenshot shows a terminal window with a dark theme. The title bar says "src/report_formatter.py". The code inside defines a function `to_html` that takes a report string and metadata as input and returns an HTML string. It uses the `markdown` library to convert Markdown to HTML. The generated HTML includes a header with a title and styles for body, h1, h2, and blockquote elements. It also includes a `metadata` div with information like creation date, score, and source count.

```
def to_html(report: str, metadata: Dict) → str:
    """리포트를 스타일링된 HTML로 변환"""
    # Markdown → HTML 변환
    import markdown
    html_body = markdown.markdown(report, extensions=['tables', 'fenced_code'])

    html_template = f"""<!DOCTYPE html>
<html>
<head>
    <meta charset="utf-8">
    <title>{metadata.get('topic', 'Report')}</title>
    <style>
        body {{ font-family: 'Noto Sans KR', sans-serif; max-width: 800px; margin: 0
auto; padding: 40px; }}
        h1 {{ color: #2c3e50; border-bottom: 3px solid #3498db; padding-bottom:
10px; }}
        h2 {{ color: #34495e; margin-top: 30px; }}
        blockquote {{ background: #f8f9fa; border-left: 4px solid #3498db; padding:
15px; }}
        .metadata {{ color: #7f8c8d; font-size: 0.9em; }}
    </style>
</head>
<body>
    <div class="metadata">
        생성일: {datetime.now().strftime('%Y-%m-%d')} |
        품질 점수: {metadata.get('score', 'N/A')}/10 |
        참고 자료: {metadata.get('source_count', 0)}건
    </div>
    {html_body}
</body>
</html>"""
    return html_template
```

파일 저장 및 미리보기 기능

ReportFormatter.save_and_preview

```
src/report_formatter.py

@staticmethod
def save_and_preview(report: str, metadata: Dict, format: str = "markdown") -> Dict:
    """리포트 저장 + 미리보기 정보 반환"""
    timestamp = datetime.now().strftime('%Y%m%d_%H%M%S')
    safe_topic = metadata['topic'][:30].replace(' ', '_')
    base_filename = f"{safe_topic}_{timestamp}"

    result = {"files": []}

    # Markdown 저장
    md_content = ReportFormatter.to_markdown(report, metadata)
    md_path = ReportFormatter.save_markdown(md_content, base_filename)
    result["files"].append({"format": "markdown", "path": md_path})

    # HTML 저장
    html_content = ReportFormatter.to_html(report, metadata)
    html_path = f"data/reports/{base_filename}.html"
    with open(html_path, "w", encoding="utf-8") as f:
        f.write(html_content)
    result["files"].append({"format": "html", "path": html_path})

    result["preview"] = report[:500] + "..."
    result["word_count"] = len(report.split())

    return result
```

⚡ 핵심 기능 요약

- ✓ 자동화된 파일 관리: 타임스탬프와 주제를 조합하여 고유 파일명 생성
- ✓ 멀티 포맷 지원: 원본 Markdown과 배포용 HTML 동시 생성
- ✓ 즉각적인 피드백: 저장 직후 미리보기 및 메타데이터 반환

출력 예시

```
리포트가 생성되었습니다A

품질 점수: 8.2/10
분량: 1,200 단어
참고 자료: 8건

저장된 파일:
• data/reports/AI_반도체_시장_20250209_143022.md
• data/reports/AI_반도체_시장_20250209_143022.html
```

실습 Part2 구현 프롬프트 - 1

src/research_coordinator.py에 ResearchCoordinator 클래스를 구현해주세요.

ResearchCoordinator는 4개의 전문 에이전트(ResearchAgent, AnalysisAgent, ReportWriter, QualityCritic)를 순차적으로 조율하여 주제를 받아 최종 리포트를 생성합니다.

기존 AutonomousOrchestrator(src/orchestrator.py)를 참고하되, 수정하지 마세요.
ResearchCoordinator는 별도의 새 파일입니다.

클래스 요구사항:

- __init__(self, client: OpenAI, search_agent: SearchAgent, memory_manager: MemoryManager)
 - 4개 에이전트 인스턴스 생성:
 - self.researcher = ResearchAgent(client, search_agent, memory_manager)
 - self.analyzer = AnalysisAgent(client)
 - self.writer = ReportWriter(client)
 - self.critic = QualityCritic(client)
 - self.max_revision_rounds = 2 (최대 수정 반복 횟수)
 - self.logger 설정
- run(self, topic: str, verbose: bool = True) -> Dict
 - 전체 실행 흐름:

Phase 1: 연구 (Research)

- verbose면 "🔍 [Phase 1/4] Researcher: 정보 수집 중..." 출력
- research_data = self.researcher.execute({"topic": topic})
- verbose면 수집 결과 요약 출력 (쿼리 개수, 검색 결과 건수)

Phase 2: 분석 (Analysis)

- verbose면 "📊 [Phase 2/4] Analyzer: 데이터 분석 중..." 출력
- analysis = self.analyzer.execute(research_data)
- verbose면 분석 결과 요약 출력 (클러스터 수, 인사이트 수)

Phase 3: 작성 + 검증 루프 (Write + Review)

- 최대 max_revision_rounds + 1 회 반복
- verbose면 "📝 [Phase 3/4] Writer: 리포트 작성 중..." 출력
- draft = self.writer.execute({"topic": topic, "analysis": analysis, "feedback": feedback or None})
- verbose면 "🔎 [Phase 3/4] Critic: 품질 검증 중..." 출력
- review = self.critic.execute({"topic": topic, "report": draft["report"]})
- verbose면 각 점수와 종합 점수 출력
- review["pass"]가 True면 루프 종료
- False면 feedback = review["feedback"]로 다음 루프에 전달

반환:

```
{
  "topic": topic,
  "report": draft["report"],      # 최종 리포트 (Markdown)
  "score": review["overall_score"], # 최종 품질 점수
  "scores": review["scores"],     # 상세 점수
  "revision_count": round_num,   # 수정 횟수
  "research_summary": {          # 연구 요약
    "queries_used": research_data["queries_used"],
    "source_count": research_data["source_count"],
    "insights_count": len(analysis.get("insights", []))
  }
}
```

get_agents_info(self) -> List[Dict]

- 각 에이전트의 name, role 정보를 리스트로 반환
- UI 표시용

주의사항:

- 기존 orchestrator.py는 수정하지 마세요
- 에러 발생 시 각 Phase에서 try/except로 잡아서 로깅 후 부분 결과라도 반환합니다
- verbose 출력은 main.py에서 사용자에게 진행 상황을 보여주는 용도입니다

실습 Part2 구현 프롬프트 - 2

src/report_formatter.py에 ReportFormatter 클래스를 구현해주세요.

ReportFormatter는 리포트를 Markdown, HTML 형식으로 변환하고 파일로 저장하는 유틸리티 클래스입니다. 모든 메서드는 @staticmethod입니다.

클래스 요구사항:

- 모든 메서드가 @staticmethod

- to_markdown(report: str, metadata: Dict) -> str

- 리포트 앞에 YAML front matter 메타데이터 추가

- 메타데이터: title, date(현재 시각), agent_score, source_count, revision_count

- 예시:

title: AI 반도체 시장 분석

date: 2025-02-09

agent_score: 7.8/10

sources: 12

revisions: 1

(이후 report 원문)

- to_html(report: str, metadata: Dict) -> str

- markdown 라이브러리를 사용하여 Markdown → HTML 변환

- 완성된 HTML 문서 (<!DOCTYPE html>부터 </html>까지)

- 스타일: Noto Sans KR 폰트, max-width: 800px, 깔끔한 CSS

- 메타데이터를 페이지 상단에 표시 (생성일, 품질점수, 참고자료 수)

- markdown 라이브러리가 없으면 간단한 HTML 태그로 대체

- save_report(report: str, metadata: Dict, output_dir: str = "data/reports") -> Dict

- output_dir 디렉토리 생성 (exist_ok=True)

- 파일명: {safe_topic}_{timestamp}.md / .html

- safe_topic: 공백 → _ 특수문자 제거, 30자 제한

- timestamp: %Y%m%d_%H%M%S 형식

- Markdown 파일 저장

- HTML 파일 저장

- 반환:

{

 "files": [

 {"format": "markdown", "path": "data/reports/xxx.md"},

 {"format": "html", "path": "data/reports/xxx.html"}

],

 "preview": report[:500] + "...",

 "word_count": len(report.split())

}

- print_report_summary(result: Dict, score: float) -> None

- 터미널에 리포트 생성 결과를 예쁘게 출력

- 포함: 품질 점수, 분량, 저장된 파일 경로

requirements.txt에 markdown 패키지를 추가해주세요:

markdown>=3.5.0

주의사항:

- 파일명의 한글은 유지합니다 (URL 인코딩 불필요)

- encoding="utf-8"로 저장합니다

- output_dir 경로가 존재하지 않아도 자동 생성합니다

실습 Part2 구현 프롬프트 - 3

tests/test_week5_part2.py에 Part 2의 단위 테스트를 작성해주세요.

테스트 항목 (총 7개):

1. test_research_coordinator_creation()

- ResearchCoordinator가 올바르게 생성되는지 확인
- 4개 에이전트가 초기화되는지 확인 (researcher, analyzer, writer, critic)
- max_revision_rounds가 2인지 확인
- Mock 객체로 OpenAI, SearchAgent, MemoryManager 대체

2. test_research_coordinator_agents_info()

- get_agents_info()가 4개 에이전트 정보를 반환하는지 확인
- 각각 name, role 키가 있는지 확인

3. test_report_formatter_to_markdown()

- ReportFormatter.to_markdown() 호출
- 결과에 YAML front matter ("---")가 포함되는지 확인
- metadata의 title이 결과에 포함되는지 확인
- 원본 report 내용이 결과에 포함되는지 확인

4. test_report_formatter_to_html()

- ReportFormatter.to_html() 호출
- 결과에 "<!DOCTYPE html>"이 포함되는지 확인
- 결과에 "<html>"과 "</html>"이 포함되는지 확인
- metadata 정보가 HTML에 포함되는지 확인

5. test_report_formatter_save_report()

- 임시 디렉토리(tempfile.mkdtemp())에 리포트 저장
- save_report() 결과에 "files" 키가 있는지 확인
- files 리스트에 2개 파일(md, html)이 있는지 확인
- 실제 파일이 디스크에 존재하는지 확인 (os.path.exists)
- 테스트 후 임시 파일 정리

6. test_report_formatter_safe_filename()

- 특수문자가 포함된 주제("AI 반도체 시장 분석!! @2025")로 save_report() 호출
- 생성된 파일명에 특수문자가 없는지 확인
- 파일이 정상적으로 생성되는지 확인

7. test_report_formatter_print_summary()

- print_report_summary()가 에러 없이 실행되는지 확인
- capsys 또는 redirect로 출력 내용 캡처하여 점수가 출력되는지 확인

테스트 실행 방법:

```
```bash
pytest tests/test_week5_part2.py -v
````
```

주의사항:

- 파일 저장 테스트는 tempfile을 사용하여 실제 data/ 디렉토리에 영향을 주지 않습니다
- OpenAI 클라이언트는 Mock으로 대체합니다
- 기존 테스트 스타일을 따릅니다

Part 2 단위 테스트

💡 가상환경 활성화 후, 테스트 스크립트 실행

1 \$ python -m venv venv

2 Win > venv\Scripts\activate

Mac \$ source venv/bin/activate

아래와 같이 나오면, 가상환경에서 동작중인 것임!

(venv)user@pc:~\$

이 상태에서 터미널 창에서
의존성 패키지 업데이트!

pytest tests/test_week5_part2.py -v

⚙️ 단위 테스트 항목

ResearchCoordinator 테스트 (2개)

test_research_coordinator_creation()

- ✓ ResearchCoordinator가 올바르게 생성되는지 확인
- ✓ 4개 에이전트(researcher, analyzer, writer, critic) 초기화 확인
- ✓ max_revision_rounds가 2인지 확인
- ✓ Mock 객체로 OpenAI, SearchAgent, MemoryManager 대체

test_research_coordinator_agents_info()

- ✓ get_agents_info()가 4개 에이전트 정보를 반환하는지 확인
- ✓ 각 에이전트에 name, role 키가 있는지 확인

ReportFormatter 테스트 (5 개)

test_report_formatter_to_markdown()

- ✓ ReportFormatter.to_markdown() 호출
- ✓ 결과에 YAML front matter ("---")가 포함되는지 확인
- ✓ metadata의 title이 결과에 포함되는지 확인
- ✓ 원본 report 내용이 결과에 포함되는지 확인

test_report_formatter_to_html()

- ✓ ReportFormatter.to_html() 호출
- ✓ 결과에 "<!DOCTYPE html>"이 포함되는지 확인
- ✓ 결과에 "<html>"과 "</html>"이 포함되는지 확인
- ✓ metadata 정보가 HTML에 포함되는지 확인

test_report_formatter_save_report()

- ✓ 임시 디렉토리(tempfile.mkdtemp())에 리포트 저장
- ✓ save_report() 결과에 "files" 키가 있는지 확인
- ✓ files 리스트에 2개 파일(md, html)이 있는지 확인
- ✓ 실제 파일이 디스크에 존재하는지 확인 (os.path.exists)
- ✓ 테스트 후 임시 파일 정리

test_report_formatter_safe_filename()

- ✓ 특수문자가 포함된 주제("AI 반도체 시장 분석!! @2025")로 save_report() 호출
- ✓ 생성된 파일명에 특수문자가 없는지 확인
- ✓ 파일이 정상적으로 생성되는지 확인

test_report_formatter_print_summary()

- ✓ print_report_summary()가 에러 없이 실행되는지 확인
- ✓ capsys 또는 redirect로 출력 내용 캡처하여 점수가 출력되는지 확인

Part 2 단위 테스트 완료 화면

IMPLEMENTATION CHECK

```
● (venv) PS C:\Users\asakh\Documents\GitHub\AI_agent_lecture_final> pytest tests/test_week5_part2.py -v  
===== test session starts =====  
platform win32 -- Python 3.12.10, pytest-9.0.2, pluggy-1.6.0 -- C:\Users\asakh\Documents\GitHub\AI_agent_lecture_final\venv\Scripts\python.exe  
cachedir: .pytest_cache  
rootdir: C:\Users\asakh\Documents\GitHub\AI_agent_lecture_final  
configfile: pytest.ini  
plugins: anyio-4.12.1  
collected 7 items  
  
tests/test_week5_part2.py::test_research_coordinator_creation PASSED [ 14%]  
tests/test_week5_part2.py::test_research_coordinator_agents_info PASSED [ 28%]  
tests/test_week5_part2.py::test_report_formatter_to_markdown PASSED [ 42%]  
tests/test_week5_part2.py::test_report_formatter_to_html PASSED [ 57%]  
tests/test_week5_part2.py::test_report_formatter_save_report PASSED [ 71%]  
tests/test_week5_part2.py::test_report_formatter_safe_filename PASSED [ 85%]  
tests/test_week5_part2.py::test_report_formatter_print_summary PASSED [100%]  
  
===== 7 passed in 1.91s =====
```

05

SECTION

HANDS-ON

IMPLEMENTATION

HANDS-ON PART 3

최종 통합 구현

☰ 작업 목표

☰ 구현할 파일 목록

| # | 작업 | 설명 |
|---|---------------|---------------------------------|
| 1 | Main.py 최종 수정 | 'report' 명령어 추가 |
| 2 | 전체 파이프라인 연결 | Coordinator → Formatter → 파일 저장 |
| 3 | 통합 테스트 | 실제 주제로 전체 시스템 테스트 |
| 4 | 에러 핸들링 | API 에러, 타임아웃 처리 |

```
>Main.py

# main.py - 최종 버전
from src.research_coordinator import ResearchCoordinator
from src.report_formatter import ReportFormatter

def main():
    # 초기화
    client = OpenAI()
    memory_manager = MemoryManager("main_memory", "data/chroma_db")
    search_agent = SearchAgent(memory_manager=memory_manager)
    conversation_manager = ConversationManager(search_agent, memory_manager)
    orchestrator = AutonomousOrchestrator(client)
    coordinator = ResearchCoordinator(client, search_agent, memory_manager)
    print("🤖 AI Research Assistant v5.0 (FINAL)")
    print("-" * 50)
    print("명령어:")
    print(" 일반 대화 - 자유롭게 질문")
    print(" /search - 웹 검색")
    print(" /memory - 메모리 통계")
    print(" auto [주제] - 자율 실행 모드")
    print(" report [주제] - 멀티에이전트 리포트 생성")
    print(" /quit - 종료")
    while True:
        user_input = input("\n👤 You: ").strip()

        if user_input.startswith("report "):
            topic = user_input[7:]
            print(f"\n📝 멀티 에이전트 리포트 생성 시작: {topic}")
            print("-" * 50)
            result = coordinator.run(topic)

            # 리포트 저장
            files = ReportFormatter.save_and_preview(
                result["report"],
                {"topic": topic, "score": result["score"]}
            )
            print(f"\n✅ 리포트 생성 완료!")
            for f in files["files"]:
                print(f" 📁 {f['path']}")
```

실습 Part3 구현 프롬프트 - 1

main.py에 멀티 에이전트 리포트 생성 기능을 추가해주세요.

기존 코드의 구조를 최대한 유지하면서 다음 변경사항을 적용합니다:

1. import 추가 (파일 상단)

- from src.research_coordinator import ResearchCoordinator
- from src.report_formatter import ReportFormatter

2. print_welcome() 함수 수정

- 버전을 "v3.0"으로 변경
- "멀티 에이전트 리포트 생성 기능이 추가되었습니다!" 메시지 추가
- 명령어 목록에 추가:
 - "report <주제>" : NEW 멀티에이전트 리포트 생성
 - "report-agents" : NEW 에이전트 정보 보기

3. main() 함수의 초기화 부분에 추가 (AutonomousOrchestrator 초기화 후)

- ResearchCoordinator 초기화:

```
```python
print("Initializing Research Coordinator...")
coordinator = ResearchCoordinator(
 client=OpenAI(),
 search_agent=search_agent,
 memory_manager=memory_manager
)
print("✓ Research Coordinator Ready (4 agents)")
```

```

4. 대화 루프에 report 명령어 처리 추가 (auto 명령어 처리 블록 뒤)

```
"report <주제>" 명령어:
- user_input_lower.startswith("report ") 체크
- goal = user_input[7:].strip()
- 빈 입력이면 사용법 안내
- 실행:
  print(f"\n 멀티 에이전트 리포트 생성 시작: {topic}")
  print("-" * 50)
  result = coordinator.run(topic, verbose=True)

# 리포트 저장
save_result = ReportFormatter.save_report(
    result["report"],
    {"topic": topic, "score": result["score"],
     "source_count": result["research_summary"]["source_count"],
     "revision_count": result["revision_count"]})

# 결과 출력
ReportFormatter.print_report_summary(save_result, result["score"])

- 여러 발생 시: print(f" X 리포트 생성 오류: {e}")
"report-agents" 명령어:
- 에이전트 정보 출력 (coordinator.get_agents_info())
"report" 만 입력 시:
- 사용법 안내: "사용법: report <주제>"
```

5. handle_command() 함수에 추가

- 'report' 명령어 단독 입력 시 사용법 안내 (auto와 동일한 패턴)

주의사항:

- 기존 코드 (대화, 검색, 메모리, 자율실행)는 그대로 유지합니다
- coordinator 초기화 실패 시에도 기존 기능은 정상 동작해야 합니다
- report 명령어와 auto 명령어는 독립적으로 동작합니다

실습 Part3 구현 프롬프트 - 2

tests/test_week5_part3.py에 5주차 전체 시스템 통합 테스트를 작성해주세요.

이 테스트는 전체 파이프라인이 올바르게 연결되었는지 검증합니다.

테스트 항목 (총 6개):

1. test_full_pipeline_structure()

- ResearchCoordinator를 Mock 의존성으로 생성
- coordinator.researcher, coordinator.analyzer, coordinator.writer, coordinator.critic이 모두 존재하는지 확인
- 각각 올바른 클래스 인스턴스인지 확인 (isinstance)

2. test_report_formatter_integration()

- 실제 Markdown 텍스트로 ReportFormatter 메서드들을 순차적으로 호출
- to_markdown() → to_html() → save_report() 체이닝 테스트
- 최종 파일이 올바르게 생성되는지 확인

3. test_agent_execute_interface()

- 각 에이전트의 execute()가 올바른 키를 가진 Dict를 반환하는지 확인
- Mock된 _call_llm, _call_llm_json을 사용
- ResearchAgent.execute() → "topic", "search_data", "source_count" 키 확인
- AnalysisAgent.execute() → "clusters", "insights", "trends" 키 확인
- ReportWriter.execute() → "report", "word_count" 키 확인
- QualityCritic.execute() → "scores", "overall_score", "pass" 키 확인

4. test_coordinator_run_with_mocked_agents()

- ResearchCoordinator의 run() 메서드를 테스트
- 4개 에이전트의 execute()를 모두 Mock으로 대체하여 정해진 결과를 반환하게 설정
- run()이 올바른 결과 Dict를 반환하는지 확인
- "report", "score", "revision_count" 키 확인

5. test_main_imports()

- main.py에서 필요한 모든 import가 정상 동작하는지 확인
- ResearchCoordinator, ReportFormatter import 확인

6. test_existing_features_intact()

- 기존 4주차 클래스들이 여전히 정상 import되는지 확인
- AutonomousOrchestrator, TaskPlanner, ReActEngine, QualityManager, LoopPrevention
- 기존 기능이 5주차 변경으로 깨지지 않았는지 검증

테스트 실행 방법:

```
```bash
Part 3 테스트만
pytest tests/test_week5_part3.py -v
```
# 5주차 전체 테스트
pytest tests/test_week5_part1.py tests/test_week5_part2.py tests/test_week5_part3.py -v
```

주의사항:

- 실제 API 호출은 하지 않습니다 (모든 OpenAI 호출은 Mock)
- 기존 4주차 테스트도 여전히 통과해야 합니다
- Mock 설정 시 side_effect가 아닌 return_value를 사용하여 안정적으로 동작하게 합니다

Part 3 단위 테스트

① 가상환경 활성화 후, 테스트 스크립트 실행

1 \$ python -m venv venv

2 Win > venv\Scripts\activate

Mac \$ source venv/bin/activate

아래와 같이 나오면, 가상환경에서 동작중인 것임!

(venv)user@pc:~\$

이 상태에서 터미널 창에서
의존성 패키지 업데이트!

pytest tests/test_week5_part3.py -v

단위 테스트 항목

전체 파이프라인 구조 테스트 (2개)

test_full_pipeline_structure()

- ✓ ResearchCoordinator를 Mock 의존성으로 생성
- ✓ coordinator.researcher, coordinator.analyzer, coordinator.writer, coordinator.critic이 모두 존재하는지 확인
- ✓ 각각 올바른 클래스 인스턴스인지 확인 (isinstance)

test_report_formatter_integration()

- ✓ 실제 Markdown 텍스트로 ReportFormatter 메서드들을 순차적으로 호출
- ✓ to_markdown() → to_html() → save_report() 체이닝 테스트
- ✓ 최종 파일이 올바르게 생성되는지 확인

에이전트 인터페이스 테스트 (2개)

test_agent_execute_interface()

- ✓ 각 에이전트의 execute()가 올바른 키를 가진 Dict를 반환하는지 확인
- ✓ Mock된 _call_llm, _call_llm_json 사용
- ✓ ResearchAgent.execute() → "topic", "search_data", "source_count" 키 확인
- ✓ AnalysisAgent.execute() → "clusters", "insights", "trends" 키 확인
- ✓ ReportWriter.execute() → "report", "word_count" 키 확인
- ✓ QualityCritic.execute() → "scores", "overall_score", "pass" 키 확인

test_coordinator_run_with_mocked_agents()

- ✓ ResearchCoordinator의 run() 메서드를 테스트
- ✓ 4개 에이전트의 execute()를 모두 Mock으로 대체하여 정해진 결과를 반복하게 설정
- ✓ run()이 올바른 결과 Dict를 반환하는지 확인
- ✓ "report", "score", "revision_count" 키 확인

시스템 통합 및 호환성 테스트 (2개)

test_main_imports()

- ✓ main.py에서 필요한 모든 import가 정상 동작하는지 확인
- ✓ ResearchCoordinator, ReportFormatter import 확인

test_existing_features_intact()

- ✓ 기존 4주차 클래스들이 여전히 정상 import되는지 확인
- ✓ AutonomousOrchestrator, TaskPlanner, ReActEngine, QualityManager, LoopPrevention
- ✓ 기존 기능이 5주차 변경으로 깨지지 않았는지 검증

Part 3 단위 테스트 완료 화면

IMPLEMENTATION CHECK

```
(venv) PS C:\Users\asakh\Documents\GitHub\AI_agent_lecture_final> pytest tests/test_week5_part3.py -v
=====
platform win32 -- Python 3.12.10, pytest-9.0.2, pluggy-1.6.0 -- C:\Users\asakh\Documents\GitHub\AI_agent_lecture_final\venv\Scripts\python.exe
cachedir: .pytest_cache
rootdir: C:\Users\asakh\Documents\GitHub\AI_agent_lecture_final
configfile: pytest.ini
plugins: anyio-4.12.1
collected 6 items

tests/test_week5_part3.py::test_full_pipeline_structure PASSED [ 16%]
tests/test_week5_part3.py::test_report_formatter_integration PASSED [ 33%]
tests/test_week5_part3.py::test_agent_execute_interface PASSED [ 50%]
tests/test_week5_part3.py::test_coordinator_run_with_mocked_agents PASSED [ 66%]
tests/test_week5_part3.py::test_main_imports PASSED [ 83%]
tests/test_week5_part3.py::test_existing_features_intact PASSED [100%]

===== 6 passed in 3.24s =====
```

통합 테스트 실행

💡 가상환경 활성화 후, 테스트 스크립트 실행

1 \$ python -m venv venv

2 Win > venv\Scripts\activate

Mac \$ source venv/bin/activate

아래와 같이 나오면, 가상환경에서 동작중인 것임!

(venv)user@pc:~\$

이 상태에서 터미널 창에서
의존성 패키지 업데이트!

> python main.py

또는 스크립트 사용:

Windows: run.bat

macOS/Linux: ./run.sh

시나리오 1: 기본 대화 기능 테스트

◎ 목적

사용자 입력 처리 및 응답 생성의 정확성 검증
대화 이력 관리 기능 확인
기본 AI 어시스턴트로서의 동작 검증

> 테스트 입력

COMMAND

안녕하세요 # 예상: 자연스러운 인사 응답
파이썬이 뭐야? # 예상: 이전 대화 참조 응답
그럼, 자바랑 뭐가 달라? # 예상: 이전 대화 참조 응답

✓ 통과 기준 (Pass Criteria)

사용자 입력에 대해 적절한 응답 생성

이전 대화 내용을 기억하고 참조

conversation_history에 모든 대화 기록 저장

에러 없이 연속 대화 처리

Output Console - Expected Result

You: 안녕하세요.

AI: 안녕하세요! 어떻게 도와드릴까요? 궁금한 점이나 필요한 정보가 있다면 말씀해 주세요.

You: 파이썬이 뭐야?

AI: 파이썬에 대해 어떤 측면이 궁금하신가요? 예를 들어, 파이썬의 용도, 장점, 기본 문법 등에 대해 설명해 드릴 수 있습니다.

You: 파이썬의 용도에 대해 알려줘.

AI: 파이썬의 주요 용도는 다음과 같습니다:

1. **웹 개발**: Django, Flask와 같은 프레임워크를 사용하여 동적 웹사이트를 구축할 수 있습니다.
2. **데이터 분석 및 과학**: Pandas, NumPy, Matplotlib 등을 활용하여 데이터 분석 및 시각화를 수행합니다.
3. **인공지능 및 머신러닝**: TensorFlow, PyTorch 등과 같은 라이브러리를 통해 AI 모델을 개발합니다.
4. **자동화 및 스크립트 작성**: 반복적인 작업을 자동화하기 위해 스크립트를 작성하는 데 사용됩니다.
5. **게임 개발**: Pygame과 같은 라이브러리를 사용하여 게임을 개발할 수 있습니다.

더 궁금한 점이 있으시면 말씀해 주세요!

시나리오 2: 웹 검색 기능 테스트

◎ 목적

Tavily API를 통한 실시간 정보 검색 기능 검증
Function Calling 메커니즘 정확성 확인
검색 결과의 적절한 활용 여부 평가

>_ 테스트 입력

COMMAND

2026년 AI기술 트렌드를 알려줘.
 # 예상: Web_search 함수 호출하여 결과 기반 응답
1+1은?
 # 예상: 검색 없이 직접 응답

✓ 통과 기준 (Pass Criteria)

- 최신 정보가 필요한 질문에서 web_search 함수 자동 호출
- 검색 결과를 기반으로 정확한 답변 생성
- 불필요한 검색 없이 효율적 판단
- 검색 실패 시 적절한 오류 처리

Output Console - Expected Result

You: 2026년 AI기술 트렌드를 알려줘.

AI: [핵심 요약]

2026년의 AI 기술 트렌드는 에이전틱 AI, 안전한 배포, 기업 워크플로우 자동화에 중점을 두고 있으며, 지속 가능한 IT 관행의 중요성이 커지고 있습니다.

[상세 내용]

- 주요 포인트 1: 에이전틱 AI(Agentic AI)의 발전이 두드러질 것으로 예상되며, 이는 AI의 자율성과 의사결정 능력을 강조합니다.
- 주요 포인트 2: 기업들은 AI의 가치를 분석적, 생성적, 에이전틱 형태로 활용하는 데 집중할 것입니다.
- 주요 포인트 3: 지속 가능한 IT 관행이 중요해지며, 기업들은 안전한 AI 배포를 위해 노력할 것입니다.

[참고 출처]

1. [MIT Sloan Management Review](<https://sloanreview.mit.edu/article/five-trends-in-ai-and-data-science-for-2026/>)
2. [International Banker](<https://internationalbanker.com/technology/five-significant-tech-trends-that-will-feature-in-2026/>)
3. [IBM](<https://www.ibm.com/think/news/ai-tech-trends-predictions-2026>)

You: 1+1은?

AI: 1 + 1은 2입니다. 다른 질문이 있으시면 말씀해 주세요!

시나리오 3: 메모리(벡터 DB) 기능 테스트

◎ 목적

ChromaDB 기반 벡터 메모리 저장 및 검색 검증
이전 대화/검색 결과의 메모리 축적 확인
메모리 대시보드 및 직접 검색 기능 동작 확인

>_ 테스트 입력

COMMAND

블록체인 기술의 핵심 원리를 설명해줘
memory
memory-search 블록체인

✓ 통과 기준 (Pass Criteria)

- 블록체인 관련 대화 후 해당 내용이 메모리에 저장됨
- memory 명령어로 메모리 대시보드 출력
(총 문서 수, 컬렉션 정보)
- memory-search 블록체인 입력 시 유사도 점수와 함께 관련 결과 출력
- 검색 결과에 "유사도: X.XX" 형식의 점수가 표시됨

Output Console - Expected Result

You: 블록체인 기술의 핵심 원리를 설명해줘

AI: 블록체인 기술의 핵심 원리에 대해 어떤 측면이 궁금하신가요? 예를 들어, 블록체인의 구조, 작동 방식, 또는 보안 기능 등에 대해 설명해 드릴 수 있습니다.

You: 구조, 작동방식, 보안기능, 최근 추세까지 모두 알려줘.

AI: 블록체인 기술에 대한 정보를 구조, 작동 방식, 보안 기능, 그리고 최근 추세로 나누어 설명하겠습니다.

1. 구조

- **블록**: 블록체인은 여러 개의 블록으로 구성되며, 각 블록은 데이터(거래 정보 등)와 이전 블록의 해시값을 포함합니다.

- **체인**: 블록들은 시간 순서대로 연결되어 체인을 형성합니다. 각 블록은 이전 블록의 해시를 포함하여 변조를 방지합니다.

(중략)

You: memory

=====
 메모리 시스템 대시보드
=====

컬렉션: research_assistant_memory

총 문서 수: 35개

평균 텍스트 길이: 166자

임베딩 차원: 1536

📁 소스별 분포:

- conversation: 20개
- autonomous_agent: 15개

You: memory-search 블록체인

=====
 메모리 검색: 블록체인

[유사도: 0.25]

시나리오 4: 자율 실행(ReAct) 기능 테스트

◎ 목적

AutonomousOrchestrator의 작업 분해 → ReAct 실행 → 품질 평가
파이프라인 검증

TaskPlanner, ReActEngine, QualityManager 연동 확인
자율 실행 통계 기능 동작 확인

>_ 테스트 입력

COMMAND

```
auto AI 에이전트의 개념과 주요 유형 정리
auto-stats
```

✓ 통과 기준 (Pass Criteria)

- "자율 실행 모드 시작" 메시지와 함께 실행 개시
- 작업 분해 → Thought-Action-Observation 루프가 진행됨
- 최종 "📋 최종 리포트" 출력 (분석 결과 텍스트)
- auto-stats 명령어로 총 실행 횟수, 평균 품질 점수, 품질 통과율 표시

Output Console - Expected Result

You: auto AI 에이전트의 개념과 주요 유형 정리

🚀 자율 실행 모드 시작

🎯 목표: AI 에이전트의 개념과 주요 유형 정리

=====

📋 Step 1: 작업 분해 중...

2026-02-10 14:36:54,218 - httpx - INFO - HTTP Request: POST https://api.openai.com/v1/chat/completions

=====

📋 Task Plan: AI 에이전트의 개념과 주요 유형 정리...

=====

⌚ [task_1] AI 에이전트의 정의와 기본 개념을 조사하고 정리하기

Priority: 1 | Status: pending

⌚ [task_2] AI 에이전트의 주요 유형(예: 규칙 기반, 기계 학습, 강화 학습 등)을 조사하고 정리하기

Priority: 2 | Status: pending (depends: task_1)

(중략)

=====

📋 최종 리포트

=====

AI 에이전트의 개념과 주요 유형 리포트

1. 개요

AI 에이전트는 환경을 인식하고 목표를 달성하기 위해 행동하는 자율 시스템으로, 다양한 산업에서 혁신적인 변화를 이끌고 있습니다. 이 리포트는 AI 에이전트의 정의, 주요 유형, 특징, 장단점, 그리고 실제 응용 사례를 종합하여 정리합니다.

(중략)

You: auto-stats

📊 자율 실행 통계

총 실행 횟수: 1

평균 품질 점수: 8.9/10

품질 통과율: 100.0%

시나리오 5: 멀티 에이전트 리포트 생성 — 기본 주제

◎ 목적

ResearchCoordinator의 4-Phase 파이프라인 동작 검증
 4개 전문 에이전트(Researcher → Analyzer → Writer → Critic) 순차
 실행 확인
 리포트 파일 출력(Markdown/HTML) 정상 생성 검증

> 테스트 입력

COMMAND

report Python vs JavaScript 비교

✓ 통과 기준 (Pass Criteria)

- Phase 1~4가 순서대로 진행되며 각 단계별 진행 상황 출력
- Critic 평가 점수가 5개 항목(completeness, accuracy, clarity, structure, source_quality)으로 출력
- 종합 점수(overall_score)가 표시됨
- data/reports/ 디렉토리에 .md 파일과 .html 파일 2개 생성
- Markdown 파일에 YAML front matter (title, date, agent_score 등) 포함
- HTML 파일이 브라우저에서 정상 렌더링 (Noto Sans KR 폰트, 스타일 적용)

Output Console - Expected Result

You: report Python vs JavaScript 비교

문서 목록

[Phase 1/4] Researcher: 정보 수집 중...

쿼리 5개, 검색·메모리 결과 8건

[Phase 2/4] Analyzer: 데이터 분석 중...

2026-02-10 14:47:45,354 - httpx - INFO - HTTP Request: POST https://api.openai.com/v1/chat/completions
 "HTTP/1.1 200 OK"

클러스터 3개, 인사이트 3개

[Phase 3/4] Writer: 리포트 작성 중...

2026-02-10 14:47:59,154 - httpx - INFO - HTTP Request: POST https://api.openai.com/v1/chat/completions
 "HTTP/1.1 200 OK"

[Phase 3/4] Critic: 품질 검증 중...

2026-02-10 14:48:03,872 - httpx - INFO - HTTP Request: POST https://api.openai.com/v1/chat/completions
 "HTTP/1.1 200 OK"

- completeness: 8.0

- accuracy : 7.0

- clarity : 8.0

- structure: 7.0

- source_quality : 8.0

종합: 7.6 (합격 기준: 7.0)

✓ 품질 기준 충족, 완료

리포트 생성 완료

품질 점수: 7.6/10

문량: 403단어

저장된 파일:

- [markdown] data/reports\Python_vs_JavaScript_비교_20260210_144803.md
- [html] data/reports\Python_vs_JavaScript_비교_20260210_144803.html

시나리오 6: 멀티 에이전트 리포트 생성 — 심화 주제

◎ 목적

복잡한 주제에서의 멀티 에이전트 협업 품질 검증
 Critic 피드백 루프(재작성) 동작 확인
 대량 검색 데이터 처리 시 안정성 검증

> 테스트 입력

COMMAND

report 2026년 생성형 AI 시장 동향과 주요 기업 전략 분석

✓ 통과 기준 (Pass Criteria)

- 검색 쿼리가 3~5개 생성되어 웹 검색 실행
- 검색 결과 + 메모리 데이터가 분석 단계에 전달됨
- 분석 결과에 clusters, insights, trends가 포함됨
- Critic 평가에서 7점 미만일 경우 Writer가 피드백을 반영하여 재작성 실행
- 최대 3회(max_revision_rounds + 1) 이내에 루프 종료
- 최종 리포트 구조: 제목, Executive Summary, 서론, 본론(주제별 섹션), 트렌드 및 전망, 결론, 참고자료

Output Console - Expected Result

You: report 2026년 생성형 AI 시장 동향과 주요 기업 전략 분석

■ 멀티 에이전트 리포트 생성 시작: 2026년 생성형 AI 시장 동향과 주요 기업 전략 분석

[Phase 1/4] Researcher: 정보 수집 중...
 2026-02-10 15:01:24,713 - src.tools.web_search - INFO - Tavily 검색 완료 - 결과: 5개, 소요 시간: 3.75초
 쿼리 5개, 검색·메모리 결과 8건
 [Phase 2/4] Analyzer: 데이터 분석 중...
 2026-02-10 15:01:33,277 - httpx - INFO - HTTP Request: POST https://api.openai.com/v1/chat/completions
 "HTTP/1.1 200 OK"
 클러스터 3개, 인사이트 3개
 [Phase 3/4] Writer: 리포트 작성 중...
 2026-02-10 15:01:48,369 - httpx - INFO - HTTP Request: POST https://api.openai.com/v1/chat/completions
 "HTTP/1.1 200 OK"
 [Phase 3/4] Critic: 품질 검증 중...
 2026-02-10 15:01:55,782 - httpx - INFO - HTTP Request: POST https://api.openai.com/v1/chat/completions
 "HTTP/1.1 200 OK"
 - completeness: 8.0
 - accuracy : 7.0
 - clarity : 8.0
 - structure: 7.0
 - source_quality : 8.0
 종합: 7.6 (합격 기준: 7.0)
 ✓ 품질 기준 충족, 완료

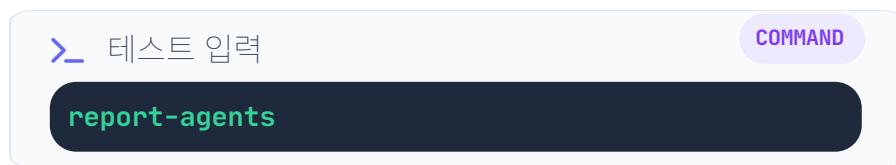
■ 리포트 생성 완료

품질 점수: 7.6/10
 분량: 384단어
 저장된 파일:
 - [markdown] data/reports\2026년_생성형_AI_시장_동향과_주요_기업_전략_분_20260210_150155.md

시나리오 7: 에이전트 정보 조회 테스트

◎ 목적

ResearchCoordinator의 에이전트 메타데이터 조회 기능 검증
4개 전문 에이전트의 역할 정보 정확성 확인



✓ 통과 기준 (Pass Criteria)

- 4개 에이전트 정보가 출력됨
 - Researcher (정보 수집 전문가)
 - Analyzer (데이터 분석 전문가)
 - Writer (리포트 작성 전문가)
 - Critic (품질 검증 전문가)

- 각 에이전트의 name, role이 정확히 표시됨

- 에러 없이 즉시 출력 (API 호출 없음)

Output Console - Expected Result

You: report-agents

리서치 에이전트 정보

- Researcher: 정보 수집 전문가
- Analyzer: 데이터 분석 전문가
- Writer: 리포트 작성 전문가
- Critic: 품질 검증 전문가

시나리오 8: 시스템 명령어 종합 테스트

◎ 목적

모든 시스템 명령어(status, save, summary, clear)의 정상 동작 검증
명령어 간 간섭 없이 독립적으로 동작하는지 확인

▶ 테스트 입력

COMMAND

```
오늘 날씨가 좋네요
status
summary
save
clear
status
```

✓ 통과 기준 (Pass Criteria)

- status: "  현재 상태:" 와 함께 검색 기능 상태, 대화 횟수, 검색 횟수 출력
- summary: 지금까지의 대화 요약 출력
- save: 대화 내용이 파일로 저장됨 (저장 경로 표시 또는 성공 메시지)
- clear: 대화 히스토리 초기화 확인 메시지 출력
- clear 후 status에서 대화 횟수가 0으로 초기화됨
- 각 명령어가 다른 명령어에 영향을 주지 않음

Output Console - Expected Result

You: 오늘 날씨가 좋네요.

AI: 오늘의 날씨에 대해 구체적으로 어떤 정보가 궁금하신가요? 예를 들어, 기온, 습도, 강수 확률 등을 알고 싶으신지 말씀해 주시면 확인해 드리겠습니다.

You: status

 현재 상태:

- 검색 기능: 활성화
- 대화 횟수: 16회
- 검색 횟수: 1회

You: summary

=====
대화 요약

사용자는 블록체인 기술의 구조, 작동 방식, 보안 기능, 그리고 최근 추세에 대해 질문했습니다. 이에 대해 블록체인의 기본 구성 요소와 작동 원리, 보안 메커니즘, 그리고 현재의 주요 트렌드를 설명했습니다.
마지막으로 사용자가 날씨에 대한 언급을 하며 추가 정보를 요청했습니다.

You: save

2026-02-10 15:13:06,450 - src.conversation_manager - INFO - 대화가 저장되었습니다:
data\conversation_20260210_151306.json (메시지 수: 19, 대화 횟수: 16)
✓ 대화가 저장되었습니다: data\conversation_20260210_151306.json

You: clear

2026-02-10 15:13:09,167 - src.search_agent - INFO - 검색 히스토리 초기화 - 삭제된 검색 기록: 1개
2026-02-10 15:13:09,168 - src.conversation_manager - INFO - 대화 히스토리 초기화됨
✓ 대화 히스토리가 초기화되었습니다.

You: status

 현재 상태:

- 검색 기능: 활성화
- 대화 횟수: 0회
- 검색 횟수: 0회

시나리오 9: 에러 핸들링 및 엣지 케이스 테스트

◎ 목적

ReportFormatter의 Markdown/HTML 변환 품질 검증
파일명 생성 규칙 (특수문자 처리, 타임스탬프) 준수 여부 확인
생성된 파일의 실제 내용 및 구조 검증

> 테스트 입력

COMMAND

(빈 입력 – Enter만 누르기)
`report`
`auto`
`memory-search`
`report`

✓ 통과 기준 (Pass Criteria)

- 파일명에 특수문자(!, @, -)가 제거되고 공백이 _로 치환됨
- 파일명 형식: {safe_topic}_{YYYYMMDD_HHMMSS}.md / .html
- 파일명 길이: 30자 이내로 제한됨
- Markdown 파일 상단에 YAML front matter 포함:

Output Console - Expected Result

You:
메시지를 입력해주세요.

You: report
사용법: report <주제>
예시: report AI 반도체 시장 동향

You: auto
사용법: auto <목표>
예시: auto AI 반도체 시장 동향 분석

You: memory-search
사용법: memory-search <검색어>

You: report
사용법: report <주제>
예시: report AI 반도체 시장 동향

시나리오 10: 리포트 출력 파일 품질 테스트

◎ 목적

잘못된 입력에 대한 방어 처리 검증
빈 입력, 명령어 오타, 잘못된 인자 등에 대한 안정성 확인
프로그램 크래시 없이 정상 복구되는지 검증

> 테스트 입력

COMMAND

(빈 입력 – Enter만 누르기)
report
auto
memory-search
report

✓ 통과 기준 (Pass Criteria)

- 빈 입력: "메시지를 입력해주세요." 출력 후 정상 대기
- report (주제 없이): "사용법: report <주제>" 형태의 안내 메시지 출력
- auto (목표 없이): "사용법: auto <목표>" 형태의 안내 메시지 출력
- memory-search (검색어 없이): "사용법: memory-search <검색어>" 안내 출력
- report (공백만): 주제가 비어있으므로 사용법 안내 출력
- 모든 케이스에서 프로그램이 종료되지 않고 다음 입력 대기 상태로 복귀

Output Console - Expected Result

You: report AI 반도체 시장 분석!! @2025 – 최신 트렌드

문서 멀티 에이전트 리포트 생성 시작: AI 반도체 시장 분석!! @2025 – 최신 트렌드

[Phase 1/4] Researcher: 정보 수집 중...

쿼리 5개, 검색·메모리 결과 8건

[Phase 2/4] Analyzer: 데이터 분석 중...

2026-02-10 15:25:25,165 - httpx - INFO - HTTP Request: POST https://api.openai.com/v1/chat/completions
 "HTTP/1.1 200 OK"

클러스터 3개, 인사이트 3개

[Phase 3/4] Writer: 리포트 작성 중...

2026-02-10 15:25:43,299 - httpx - INFO - HTTP Request: POST https://api.openai.com/v1/chat/completions
 "HTTP/1.1 200 OK"

[Phase 3/4] Critic: 품질 검증 중...

2026-02-10 15:25:51,097 - httpx - INFO - HTTP Request: POST https://api.openai.com/v1/chat/completions
 "HTTP/1.1 200 OK"

- completeness: 8.0

- accuracy : 7.0

- clarity : 8.0

- structure: 7.0

- source_quality : 8.0

종합: 7.6 (합격 기준: 7.0)

✓ 품질 기준 충족, 완료

=====

리포트 생성 완료

=====

품질 점수: 7.6/10

=====

분량: 404단어

=====

저장된 파일:

- [markdown] data/reports\AI_반도체_시장_분석_2025_최신_트렌드_20260210_152551.md
 - [html] data/reports\AI_반도체_시장_분석_2025_최신_트렌드_20260210_152551.html

시나리오 11: 종료 및 대화 저장 테스트

◎ 목적

프로그램 종료 프로세스의 정상 동작 검증
종료 시 대화 저장 옵션 기능 확인
다양한 종료 명령어의 동일한 동작 보장

▶ 테스트 입력

COMMAND

```
quit  
y
```

✓ 통과 기준 (Pass Criteria)

- quit 입력 시 "대화를 저장하시겠습니까? (y/n):" 프롬프트 출력
- y 입력 시 대화 내용이 파일로 저장되고 저장 완료 메시지 표시
- 종료 메시지 출력: "대화를 종료합니다. 안녕히 가세요!"
- 총 대화 횟수 표시
- exit, 종료 명령어도 동일하게 동작
- 프로그램이 정상적으로 종료됨 (exit code 0)

Output Console - Expected Result

You: quit

2026-02-10 15:56:48,980 - __main__ - INFO - 사용자가 종료 명령어를 입력했습니다.

대화를 저장하시겠습니까? (y/n): Y

2026-02-10 15:56:50,268 - src.conversation_manager - INFO - 대화가 저장되었습니다:

data\conversation_20260210_155650.json (메시지 수: 1, 대화 횟수: 0)

✓ 대화가 저장되었습니다: data\conversation_20260210_155650.json

=====

👉 대화를 종료합니다. 안녕히 가세요!

총 대화: 0회

총 검색: 0회

=====

2026-02-10 15:56:50,271 - __main__ - INFO - 프로그램 종료. 총 대화 횟수: 0회, 총 검색 횟수: 0회

06

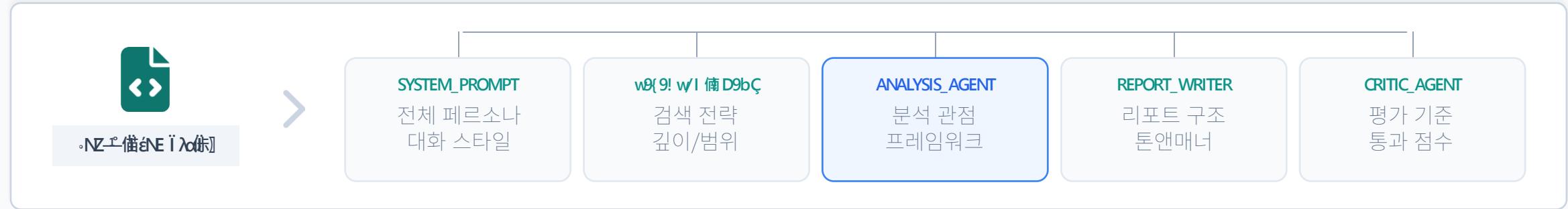
SECTION

SUMMARY

REVIEW

커스터마이징 가이드 & 향후 학습

프롬프트 커스터마이징 방법



Case 1: 마케팅 분석용

마케팅 전문 프레임워크 적용

```
ANALYSIS_AGENT_PROMPT = """
```

당신은 수석 마케팅 분석가입니다.
수집된 정보를 다음 프레임워크로 분석하세요:

- STP 분석 (Segmentation, Targeting, Positioning)
- 4P Mix (Product, Price, Place, Promotion)
- SWOT 분석 (강점, 약점, 기회, 위협)

타겟 오디언스의 페인 포인트와 경쟁사 차별점에
집중하여 인사이트를 도출하세요.

...
...
...

Case 2: 투자 분석용

재무 및 시장성 중심 분석

```
ANALYSIS_AGENT_PROMPT = """
```

당신은 VC 투자 심사역입니다.
다음 지표를 중심으로 기업/시장을 분석하세요:

1. 시장 규모 (TAM / SAM / SOM)
2. 경쟁 강도 (Porter's Five Forces)
3. 재무 건전성 및 밸류에이션 추정
4. 핵심 리스크 요인 (Regulatory, Tech)

성장 잠재력과 투자 회수(Exit) 가능성을
중점적으로 평가하세요.

...
...
...

🔧 새로운 도구 추가하기

新三 도구 추가 3단계

1

도구 함수 작성
src/tools/*.py

2

スキマ 등록
tool_definitions.py

3

에이전트 참조
Agent Initialization



News API

실시간 뉴스 수집

NewsAPI.org



데이터 분석

CSV/Excel 처리

pandas



웹 스크래핑

상세 페이지 크롤링

BeautifulSoup



이메일 발송

리포트 자동 발송

SendGrid



DB 연동

구조화 데이터 저장

SQLite/PostgreSQL



차트 생성

데이터 시각화

matplotlib

News API 도구 추가 예시

src/tools/news_search.py

```
# src/tools/news_search.py
import requests
def search_news(query: str, language: str = "ko") -> list:
    url = "https://newsapi.org/v2/everything"
    params = {
        "q": query,
        "language": language,
        "sortBy": "publishedAt",
        "apiKey": os.getenv("NEWS_API_KEY")
    }
    response = requests.get(url, params=params)
    return response.json().get("articles", [])[:5]
```

프로덕션 배포 고려사항

프로덕션으로 가기 전 고려사항

| 항목 | 현재 (학습용) | 프로덕션 (상용 서비스) |
|-------|---------------|----------------------------|
| 인터페이스 | CLI (터미널) | Web UI (Streamlit/FastAPI) |
| DB | ChromaDB (로컬) | Pinecone / Weaviate (클라우드) |
| 비용 관리 | 제한 없음 | 토큰 사용량 모니터링 |
| 에러 처리 | 기본 try/except | 재시도 로직 + 알림 |
| 보안 | . | 환경 변수 + Secret Manager |
| 모니터링 | print/logging | LangSmith / LangFuse |

Web UI 전환 예시 (Streamlit)

```
● ● ● app.py

import streamlit as st
from src.research_coordinator import ResearchCoordinator

st.title("🤖 AI Research Assistant")
topic = st.text_input("리포트 주제를 입력하세요")

if st.button("리포트 생성"):
    with st.spinner("멀티 에이전트가 작업 중..."):
        result = coordinator.run(topic)
    st.markdown(result["report"])
    st.metric("품질 점수", f"{result['score']}/10")
```

07

SECTION

SUMMARY

REVIEW

마무리
5주간의 여정 회고



🏁 5주간의 여정 회고

WEEK
01

대화형 AI 코어 구축

“ “안녕하세요” 첫 응답의 순간

✓ ConversationManager

</> OpenAI API

💬 Context 관리

WEEK
02

정보 수집 에이전트

“ 스스로 검색을 시작한 순간

✓ SearchAgent

🌐 Tavily API

⚙️ Function Calling

WEEK
03

지식 베이스 & 메모리

“ 대화를 기억하기 시작한 순간

✓ MemoryManager

💽 ChromaDB

🧠 RAG

WEEK
04

자율 실행 에이전트

“ 스스로 계획하고 실행한 순간

✓ TaskPlanner

🤖 ReActEngine

✖️ QualityManager

WEEK
05

멀티 에이전트 협업

★ 여러 AI가 협업하여 리포트 완성!

✓ ResearchCoordinator

人群 4 Agents

📄 Formatter



최종 완성 시스템 8대 기능 총정리

| # | 핵심 기능 | 구현 주차 | 핵심 기술 스택 |
|---|-----------------|--------|---|
| 1 | 자연스러운 대화 인터페이스 | WEEK 1 | OpenAI API, System Prompt |
| 2 | 자동 웹 검색 및 정보 수집 | WEEK 2 | Tavily API, Function Calling |
| 3 | 벡터 DB 기반 지식 관리 | WEEK 3 | ChromaDB, Embeddings, RAG |
| 4 | ReAct 패턴 자율 실행 | WEEK 4 | Thought-Action-Observation |
| 5 | 작업 분해 및 계획 | WEEK 4 | Task Decomposition |
| 6 | 멀티 에이전트 협업 | WEEK 5 | Role Assignment, Communication Protocol |
| 7 | 자동 품질 검증 | WK 4+5 | Self-Reflection, Critic Agent |
| 8 | 전문적인 리포트 생성 | WEEK 5 | ReportWriter, Format Converter |

TECHNOLOGY STACK SUMMARY

Language: Python 3.11+

LLM Core: GPT-4o-mini

Vector DB: ChromaDB

IDE: Cursor (AI-assisted)

Patterns: ReAct, RAG, Multi-Agent

Search: Tavily Search API



수료를 축하합니다!

“ 5주 동안 함께 만들어온 AI Research Assistant,
이제 여러분만의 AI 에이전트로 발전시켜 나가세요! ”



1,000+ 줄의 Python 코드



8개 핵심 기능 구현



7개 주요 클래스 설계



5개 전문 에이전트 개발



완전한 멀티 에이전트 시스템

✉ 강사 연락처 / 커뮤니티

thomas@itengineers.net