
- `Colab에서 실행할 것!`

1.2. `Colab` 버전으로 시도

구분	교재 코드	코랩 안내 코드
실습 방식	외부 API/서비스 위주	오픈소스/무료 내장 함수 위주
요구사항	API 키, 서드파티 설치	키/서드파티 필요 없음
목적	API 실전 연결·LLM 활용 중점	도구 원리·로컬/무료 환경 학습 중점
확장성	다양한 외부 서비스와 연계 가능	Colab/Python 환경에서 모든 도구 자유롭게 확장

1) 코랩 환경 준비

- 코랩 환경 업데이트 및 필수 설치

```
!pip install langchain diffusers transformers accelerate --upgrade
```

- 뉴스 RSS 처리 / 이미지 출력을 위해 추가

- `RSS`

	설명
기본 개념	“Really Simple Syndication” 또는 “Rich Site Summary”의 줄임말 웹사이트(뉴스, 블로그 등)에 새 글이나 소식이 올라올 때, 그 소식을 자동으로 알려주는 전용 파일(피드) 파일 ≒ 일종의 알림 목록 → 사이트에서 새 소식이 생기면 그 내용이 바로 RSS 피드에 추가됨
RSS 피드 특징	`자동으로 최신 글을 알려줌` → 직접 웹사이트에 방문하지 않아도, 새로운 글을 놓치지 않고 받아볼 수 있음 **특별한 프로그램(=RSS 리더)** 사용 → 여러 사이트의 RSS 피드를 한 곳에서 한 번에 볼 수 있음 주로 `XML`**이라는 특별한 파일 형식을 사용** - 코드처럼 생겼지만, 사람이 읽을 수도 있고 컴퓨터도 쉽게 읽음
장점	`시간 절약`: 여러 뉴스 사이트, 블로그를 일일이 방문하지 않고도 모든 새 글을 한꺼번에 볼 수 있음 `광고나 쓸데없는 디자인 없이 보기 가능`: 필요한 정보만 골라서 볼 수 있음 `무료/개방형`: 누구나 사용할 수 있고, 사이트마다 거의 제한 없이 지원됨
단점	`사진·동영상` **등 화려한 콘텐츠는 보기 힘듦**: 거의 글 위주로 전달 → 사이트의 멋진 디자인은 잘 안 나옴 `인터넷 사이트가 RSS를 지원해야 함`: 모든 사이트가 RSS 피드를 제공하는 건 아님 `댓글, 포럼 등 상호작용 기능`**은** `제한적`: RSS로 단순 정보만 받을 뿐, 바로 댓글을 달거나 토론은 어려움
사용되는 경우	뉴스, 블로그, 유튜브 채널 등을 자주 확인할 때 공부 정보/사이트/웹툰 등 최신 업데이트를 빠르게 받고 싶을 때 여러 사이트의 새 글을 한 번에, 빠르고 간편하게 보고 싶을 때
사용 예시	`RSS 리더(앱)`: Feedly, Inoreader, Old Reader, Google 뉴스 등 `구글 뉴스 RSS`: 원하는 뉴스 키워드를 입력하면 “RSS” 파일로 최신 뉴스를 자동 수집해줌

	설명
	유튜브 채널 RSS: 좋아하는 채널별로 새 영상 알림을 RSS로 받을 수 있음
	블로그 피드: 공부 블로그, IT 소식, 만화, 스포츠 기사 등 새 글이 올라올 때마다 RSS로 알림
비유 및 정리	RSS = **자동 알림장을 모아 보는 서비스**
	≒ 우리 반, 옆반, 3학년 게시판 새 소식이 하나의 알림장에 자동으로 모여서 볼 수 있는 것
	∴ **RSS = 웹사이트 새 소식을 자동으로 모아서 한 번에 쉽게 알려주는 알림 시스템**

```
!pip install feedparser pillow --upgrade
```

2) *위키독스* vs `Colab` 실습 코드

	위키독스 코드	
① API 사용 방식	**외부 API/유료 서비스(DALL-E, Tavily 등)의 키 설정 및 호출** 중심	**오픈소스/무료**
		→ 모든 기능이
- 예시	DALL-E 이미지 생성 → DallEAPIWrapper로 OpenAI API 사용(유료 키 필요)	Stable Diffus
	TavilySearch → Tavily 서비스 API 키 등록 및 사용	구글 뉴스 RSS
② 도구 (Tool) 구현 방식	LangChain의 다양한 built-in tool, 실제 커스텀 tool 등 실습 예시 다수	정식 LangCh
	@tool 데코레이터와 Tool 객체 혼용	커스텀 함수/P
	Tavily/GoogleNews 등 일부는 래핑된 API 클래스를 바로 사용	Stable Diffus
③ 이미지 생성 부분	DALL-E API 기반, 프롬프트 생성 및 API 호출 흐름	**Stable Di**
	a. 프롬프트를 LLM으로 영어, 상세하게 생성	a. diffusers로
	b. DallEAPIWrapper로 이미지를 생성	b. 이미지 저장
	c. Image(url=image_url)로 결과 출력	
	반드시 OpenAI API 키가 있어야 사용 가능	**완전 무료** ·
④ 뉴스 검색 부분	langchain_teddynote의 GoogleNews 또는 TavilySearch tool 사용	**feedparse**
	a. teddynote 패키지(비공식) 설치 필요	@tool 데코레
	b. Tavily는 서드파티 API 키 필요	비용, 설치, 등
	함수 및 LangChain의 툴 형태로 래핑하여 agent workflow 연결	
⑤ LangChain 에이전트 연결	여러 built-in/사용자 도구를 에이전트에 연결, 대부분 LLM(OpenAI 기반) 연동 전제	도구 목록을 직
	agent/run으로 명령 작성 → workflow 자동화	무료 환경 실습
		실제 도구 기능
⑥ 환경과 의존성	API 키, 서드파티, 비공식 패키지(langchain_teddynote) 등이 곳곳에 필요	PyPI에서 공식
	일부 section은 설치/키 등록을 못하면 실습 불가	Colab에서 바
⑦ 학습 흐름 및 활용성	실습 방식: 외부 API/서비스 위주	실습 방식: 오픈
	요구 사항: API 키, 서드파티 설치	요구 사항: 키/
	목적: API 실전 연결·LLM 활용 중점	목적: 도구 원리
	확장성: 다양한 외부 서비스와 연계 가능	확장성: Colab

- 위키독스: *`실제 상업/실전 API와 LangChain의 공식 빌트인·커스텀 도구 연동, agent 활용까지 포괄적 학습용`*
- `Colab` 시도: **`비용·설치 제약 없는 환경에서 LangChain의 핵심 원리, 파이프라인 구현, agent+도구 구조를 실습하는 데 최적화`**

3) 이미지 생성

- **이미지 생성:** `Stable Diffusion`
  - `Stable Diffusion` 파이프라인 함수 만들기

```
from diffusers import StableDiffusionPipeline
import torch
from PIL import Image

# pipeline 셋업 (최초 실행 시 2~3분 소요)
pipe = StableDiffusionPipeline.from_pretrained(
    "runwayml/stable-diffusion-v1-5"
).to("cpu")                              # Colab: 'cuda', Mac: 'mps', CPU:

def generate_image(prompt: str, path="output.png") -> str:
    # 이미지를 프롬프트로 생성
    image = pipe(prompt).images[0]
    image.save(path)
    return path
```

```
Loading pipeline components...: 100% 7/7 [00:00<00:00, 15.76it/s]
```

```
# pipeline 셋업 (최초 실행 시 2~3분 소요)
pipe = StableDiffusionPipeline.from_pretrained(
    "runwayml/stable-diffusion-v1-5"
).to("cuda")  # Colab: 'cuda', Mac: 'mps', CPU: 'cpu'
```

- → `Error` 발생
- → `cuda`에서 `cpu`로 수정해서 실행

- `LangChain` 도구로 등록하기

```
from langchain.tools import tool

@tool
def generate_image_tool(prompt: str) -> str:
    """텍스트 프롬프트를 받아 이미지를 생성하고 파일 경로를 반환합니다."""
```

```
    print(f"이미지 프롬프트: {prompt}")
    return generate_image(prompt)
```

- *실습 예시*

```
# 직접 함수 테스트
image_path = generate_image_tool.invoke({"prompt": "A cat reading a book
Image.open(image_path).show()
```

```
이미지 프롬프트: A cat reading a book in a library, digital painting

# 28% 14/50 [06:38<15:08, 25.24s/it]
# 52% 26/50 [11:38<09:59, 24.98s/it]
# 82% 41/50 [17:28<03:30, 23.38s/it]

100% 50/50 [21:10<00:00, 24.15s/it]
```

```
import os
print(os.path.exists(image_path))
```

- `True`면 파일 생성 완료!

```
True
```

```
from PIL import Image
from IPython.display import display

img = Image.open(image_path)
display(img)
```

- **`display(img)`**

---

4) 구글 뉴스 검색

- `구글 뉴스 검색` - `공식 RSS` + `LangChain` 도구화
  - 뉴스 검색 함수 정의 *(feedparser 사용)*

```
import feedparser
import urllib.parse

def google_news_search(keyword: str, max_results=5):
    # 구글 뉴스 RSS URL 구성
    encoded_keyword = urllib.parse.quote(keyword)
    url = f"https://news.google.com/rss/search?q={encoded_keyword}&hl=ko&
    feed = feedparser.parse(url)
    results = []
    for entry in feed.entries[:max_results]:
        results.append({
            "title": entry.title,
            "url": entry.link,
            "published": entry.published,
            "summary": entry.summary
```

```
            })
        return results
```

- `LangChain` 도구로 등록하기

```
from langchain.tools import tool

@tool
def google_news_tool(keyword: str) -> list:
    """키워드로 구글 뉴스 검색 결과(최대 5개)를 리스트로 반환"""
    print(f"뉴스 키워드: {keyword}")
    return google_news_search(keyword)
```

- *실습 예시*

```
results = google_news_tool.invoke({"keyword": "일론 머스크"})
for idx, item in enumerate(results):
    print(f"{idx + 1}. {item['title']} ({item['published']})\n{item['url'
```

- **`뉴스 키워드`: `일론 머스크`**

```
뉴스 키워드: 일론 머스크
1. [일론 머스크 “TSMC·삼성전자와 AI5칩 함께 개발” - v.daum.net (Thu, 23 Oct 2025 01
https://news.google.com/rss/articles/CBMiT0FVX3lxTFBYU2RZNy05WnZqU011SkdON0

2. [일론 머스크 “‘로봇 군대’ 만들려면 내게 더 많은 지분 달라”...1조달러 보상안 논란 - 로봇신문
https://news.google.com/rss/articles/CBMibEFVX3lxTE9Uc3JLRElFalpYc1ozTkFSUm

3. [트럼프 정책 변화로 테슬라 AI 신사업 재조명, 일론 머스크 '비전' 힘 실린다 - 비즈니스포스트
https://news.google.com/rss/articles/CBMic0FVX3lxTE05UUZMT1ZhcGRlcTlMQmR5a1

4. [시키지도 않은 19금 그렸다…머스크가 푹 빠진 AI, 그록 -AI마스터클래스⑤ , 중앙일보 (Thu, 2
https://news.google.com/rss/articles/CBMiVkFVX3lxTE9BVHFzNW5WeTdMQmsxcUVzTC

5. [머스크, 로보택시 목표 대폭 축소…옵티머스 양산도 내년으로 연기 - 글로벌이코노믹 (Thu, 23 O
https://news.google.com/rss/articles/CBMiiAFBVV95cUxPYWl4TW9kNk1odXpGN0ZUc3
```

5) 최신 `LCEL` 문법 트렌드

- 핵심
  - `체인`(Chain), `도구`(Tool), `파서`(Parser), `프롬프트`(Prompt) 등 모든 객체가 **`.invoke()`** 또는 **`|`(파이프)** 오퍼레이터로 `연결`되는 **`함수형 파이프라인`** 스타일을 지

향

- **체인 생성/조합 문법이 일관되게 함수형 API로 개편**
- **`async 지원 확대`: `.ainvoke()`, `.abatch()` 등 비동기 처리 강화**
- **`중간 결과`(프롬프트, 파서 등) `추적`·`재사용`이 `간결`해짐**

- 최신 `LCEL` 코드의 예시

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

llm = ChatOpenAI(model="gpt-3.5-turbo")

prompt = ChatPromptTemplate.from_messages([
    ("system", "You are an expert."),
    ("human", "{input}")
])

chain = prompt | llm | StrOutputParser()

# 동기 실행
res = chain.invoke({"input": "Hello, what can you do?"})

# 비동기 실행
res = await chain.ainvoke({"input": "Hello"})
```

- *기존의 `LLMChain(prompt=..., llm=...)` 보다 훨씬 `직관적`*
- **`chain | tool | parser = 파이프라인 식 연결`**
- 특징
  - **`코드가 더 짧고 가독성, 확장성이 매우 좋음`**
  - **`비동기·스트림 처리 시나리오도 일관된 방식으로 가능`**
  - **`체인 내에 분기, 조건부 실행(| if, filter, map 등)도 LCEL에서 지원`**

6) `LangGraph` 최신 트렌드

- 핵심
  - **`LLM 워크플로우의 Directed Graph 구조`** 지향

  - → 노드(node: 도구/체인/함수 등)와 간선(edge: 조건 분기, 룰 등)으로 에이전트, 멀티스텝 플로우를 유연하게 구성
  - 거대한 파이프라인과 수많은 도구/프롬프트들이 실제 서비스 수준으로 배포 가능하도록 설계

- `LangGraph 에이전트 & 워크플로우` 정의 예시

```
from langgraph.prebuilt import create_react_agent, ToolNode
from langchain_openai import ChatOpenAI
from langchain_core.tools import tool

@tool
def calculator(expr: str) -> str:
    return str(eval(expr))

# 노드 정의
tools = [ToolNode(calculator)]
llm = ChatOpenAI(model="gpt-3.5-turbo")

# REACT 기반 LangGraph Agent 생성
agent = create_react_agent(llm, tools)

# 플로우 실행
result = agent.invoke({"messages": [("user", "2+2를 계산해줘")]})

print(result)
```

- - `노드` (`도구`, `프롬프트`, `체인`) → `간선` (`조건`, `라우팅`) `조합`으로 큰 `플로우 구현` 가능
- - `agent.batch()`, `ainvoke()` 등 대용량/비동기 지원

- 특징
  - 분기/조건/스테이트풀 워크플로우가 `그래프` 로 표현 → `복잡한 멀티에이전트` 또는 `연속 프로세스 구현`에 최적
  - 기존 체인의 `단방향` 파이프라인 `한계`를 `극복`
  - `대규모 서비스`, `자동화 에이전트`, `데이터 파이프라인`까지 손쉽게 `확장`

- next: `Agent` *도전*