LangChain이란 무엇인가?

- LangChain은 대규모 언어 모델(LLM)을 활용하여 응용 프로그램을 개발하기 위한 프레임워크

LangChain과 ChatGPT API

- LangChain: 대화 기록을 유지하고 관리하는 메모리 시스템을 제공
- ChatGPT API: 개발자가 직접 대화 맥락을 관리

langchain (코어 패키지)

1. 핵심 기능과 추상화 제공
   - 기본적인 체인(Chain)
   - 프롬프트 템플릿
   - 출력 파서
   - 메모리 관리
   - 기타 핵심 유틸리티
2. 안정적이고 검증된 기능들 포함
3. LangChain 팀이 직접 관리하고 지원

langchain_community

1. 커뮤니티에서 개발한 통합(integrations) 제공
   - 다양한 LLM 제공자들과의 통합
   - 벡터 저장소 통합
   - 문서 로더
   - 임베딩
   - 검색 도구
2. 서드파티 통합이 주를 이룸
3. 커뮤니티 기여자들이 주로 관리

라이브러리 설치

```
pip install openai
pip install langchain
pip install langchain-openai
pip install langchain_community
pip install langchain-core
```

ChatGPT API

In [3]:
```
from openai import OpenAI
import getpass
```

```
# API 키를 안전하게 입력받음
API_KEY = getpass.getpass("OpenAI API 키를 입력하세요: ")
```

리스트에 대화 내용을 저장하여 앞의 대화 이어가기

In [4]:

```
from openai import OpenAI

# OpenAI API 클라이언트 초기화
client = OpenAI(api_key=API_KEY)

# 대화 기록을 수동으로 관리
conversation_history = []

def chat(message):
    # 사용자 메시지를 대화 기록에 추가
    conversation_history.append({"role": "user", "content": message})

    # OpenAI API를 사용하여 응답 생성
    response = client.chat.completions.create(
        model="gpt-4o-mini",
        messages=conversation_history,
        max_tokens=100
    )

    # AI의 응답 추출
    ai_message = response.choices[0].message.content.strip()

    # AI 응답을 대화 기록에 추가
    conversation_history.append({"role": "assistant", "content": ai_message})

    return ai_message

# 첫 번째 대화
msg1 = "안녕하세요!"
print("사용자 : ", msg1)
print("AI : ", chat(msg1))

# 두 번째 대화
msg2 = "제 이름은 김철수입니다."
print("사용자 : ", msg2)
print("AI : ", chat(msg2))

# 세 번째 대화
msg3 = "제 이름이 뭐였죠?"
print("사용자 : ", msg3)
print("AI : ", chat(msg3))

# 세 번째 대화
msg4 = "제 이름이 뭐였죠? 어떻게 내 이름을 알고 있니?"
print("사용자 : ", msg4)
print("AI : ", chat(msg4))
```

```
사용자 :  안녕하세요!
AI :  안녕하세요! 무엇을 도와드릴까요?
사용자 :  제 이름은 김철수입니다.
AI :  반갑습니다, 김철수님! 어떻게 도와드릴까요?
사용자 :  제 이름이 뭐였죠?
AI :  김철수님이세요! 도움이 필요하신 부분이 있으면 말씀해 주세요.
사용자 :  제 이름이 뭐였죠? 어떻게 내 이름을 알고 있니?
AI :  김철수님이시라고 말씀하셨기 때문에 알고 있습니다! 대화 중에 언급된 정보는 기억하고 있습니다. 다른 궁금한 점이나 도움이 필요하신 점이 있다면 말씀해 주세요!
```

LangChain 메모리 시스템 사용

- LangChain은 자동으로 대화 맥락을 유지하므로, 세번째 질문에서 이름을 기억할 수 있다.

Python에서 .env 파일 읽기

```
pip install python-dotenv
```

초기 버전의 LangChain

In [10]:
```python
from dotenv import load_dotenv
import os

# .env 파일 로드
load_dotenv("chatgpt.env")

# .env 파일에서 API 키 가져오기
CHATGPT_APIKEY = os.getenv("API_KEY")

if not CHATGPT_APIKEY:
    raise ValueError("OPENAI_API_KEY가 .env 파일에 설정되지 않았습니다.")
```

In [34]:
```python
from langchain.memory import ConversationBufferMemory
from langchain.chains import ConversationChain
from langchain.llms import OpenAI


# 메모리 시스템 초기화
# 대화 내용을 순차적으로 저장하는 간단한 메모리 시스템
# 맥락 유지: ConversationBufferMemory를 통해 대화 맥락을 자동으로 유지
memory = ConversationBufferMemory()

# 대화 체인 생성
# 언어 모델, 메모리, 프롬프트 템플릿을 결합한 대화 시스템
conversation = ConversationChain(
    llm=OpenAI(),
    memory=memory,   # 위에서 생성한 메모리 시스템을 사용
    verbose=False     # 상세한 출력을 활성화
)

# 첫 번째 대화
# conversation.predict()를 사용하여 대화를 진행
msg1 = "안녕하세요!"
print("사용자 : ", msg1)
```

```
response1 = conversation.predict(input=msg1)
print(response1)

# 두 번째 대화 (이전 대화 맥락 유지)
msg2 = "제 이름은 김철수입니다."
print("사용자 : ", msg2)
response2 = conversation.predict(input=msg2)
print(response2)

# 세 번째 대화 (이름을 기억함)
response3 = conversation.predict(input="제 이름이 뭐였죠?")
print(response3)
```

```
사용자 :  안녕하세요!
 안녕하세요! 제 이름은 인공지능이고, 저는 영어, 한국어 그리고 일본어를 할 수 있어요. 지금은 미국 시간으로 오후 3시 30분이에요. 지금은 일요일이고, 오늘 날짜는 11월 15일이에요. 제가 도와드릴 수 있는 것이 있나요?
사용자 :  제 이름은 김철수입니다.
 반가워요, 김철수님! 저는 당신의 이름을 기억할 수 있어요. 제가 또 도와드릴 수 있는 게 있나요?
 죄송해요 김철수님, 제가 기억을 잘못해서 당신의 이름을 기억하지 못해요. 제 이름은 인공지능이에요.
```

여러가지 기능 변경 및 추가 버전

In [11]:
```
from langchain_openai import ChatOpenAI
from langchain.memory import ChatMessageHistory
from langchain_core.runnables import RunnableWithMessageHistory

# .env 파일 로드
load_dotenv("chatgpt.env")

# .env 파일에서 API 키 가져오기
CHATGPT_APIKEY = os.getenv("API_KEY")

if not CHATGPT_APIKEY:
    raise ValueError("OPENAI_API_KEY가 .env 파일에 설정되지 않았습니다.")
```

간단 모드

In [12]:
```
from langchain.memory import ChatMessageHistory
from langchain.chat_models import ChatOpenAI
from langchain_core.runnables import RunnableWithMessageHistory

# 간단한 히스토리 관리와 모델 초기화
memory = ChatMessageHistory()
llm = ChatOpenAI(model_name="gpt-4o-mini", openai_api_key=CHATGPT_APIKEY)

conversation = RunnableWithMessageHistory(runnable=llm, get_session_history=lamb


# 대화 예시
session_id = "session_123"  # 세션 ID 정의

# 단일 대화 흐름
msg1 = "안녕하세요!"
response1 = conversation.invoke(
    {"content": msg1},
```

```
    config={"configurable": {"session_id": session_id}}
)

print("AI:", response1.content)

msg2 = "제 이름은 김철수입니다."
response2 = conversation.invoke(
    {"content": msg2},
    config={"configurable": {"session_id": session_id}}
)

print("AI:", response2.content)

msg3 = "제 이름이 뭐였죠?"
response3 = conversation.invoke(
    {"content": msg3},
    config={"configurable": {"session_id": session_id}}
)


print("AI:", response3.content)
```

```
C:\Users\daniel_wj\AppData\Local\Temp\ipykernel_30416\4273041000.py:7: LangChainD
eprecationWarning: The class `ChatOpenAI` was deprecated in LangChain 0.0.10 and
will be removed in 1.0. An updated version of the class exists in the :class:`~la
ngchain-openai package and should be used instead. To use it run `pip install -U
:class:`~langchain-openai` and import as `from :class:`~langchain_openai import C
hatOpenAI``.
  llm = ChatOpenAI(model_name="gpt-4o-mini", openai_api_key=CHATGPT_APIKEY)
AI: 안녕하세요! 어떻게 도와드릴까요?
AI: 만나서 반갑습니다, 김철수님! 어떤 이야기를 나누고 싶으신가요?
AI: 당신의 이름은 김철수입니다! 더 궁금한 점이 있으신가요?
```

조금 복잡한 버전

In [13]:

```
# 새로운 메모리 관리 시스템
memory = ChatMessageHistory()

# LLM 설정
# OpenAI -> ChatOpenAI로 변경
# ChatOpenAI 초기화
llm = ChatOpenAI(
    model_name="gpt-4o-mini",  # 사용 모델 (예: "gpt-4")
    temperature=0.7,     # 응답의 창의성 조정
    max_tokens=500,      # 최대 출력 토큰 수
    openai_api_key=CHATGPT_APIKEY
)

# 세션 히스토리 저장소
session_histories = {}

# 세션 히스토리를 생성 또는 불러오는 함수
# def 함수명(매개변수명: 타입) -> 반환값_타입:
def get_session_history(session_id: str)-> ChatMessageHistory:
    if session_id not in session_histories:
        session_histories[session_id] = ChatMessageHistory()  # 새 히스토리 생성
    return session_histories[session_id]

# ConversationChain에서 RunnableWithMessageHistory로 대체
```

```python
# LangChain에서 대화 이력(chat history)을 관리하기 위한 래퍼(wrapper) 클래스
# 주로 언어 모델과의 상호작용에서 **메시지 히스토리(대화 기록)**를 처리하는 데 사
#  01. 대화 이력을 자동으로 관리
#  02. 각 세션별로 독립적인 대화 컨텍스트 유지
#  03. 메시지 저장 및 검색 기능 제공
# RunnableWithMessageHistory 초기화
conversation = RunnableWithMessageHistory(
    runnable=llm,  # 실행할 LLM이나 체인
    get_session_history=get_session_history,  # 세션 히스토리 생성 함수
    message_history_key="chat_history"  # 선택적: 히스토리 키 이름
)

# 대화 예시
session_id = "session_123"  # 세션 ID 정의

# 대화 예시
msg1 = "안녕하세요!"
print("사용자:", msg1)
response1 = conversation.invoke(
    {"content": msg1},
    config={"configurable": {"session_id": session_id}}
)
print("AI:", response1.content)

msg2 = "제 이름은 김철수입니다."
print("사용자:", msg2)
response2 = conversation.invoke(
    {"content": msg2},
    config={"configurable": {"session_id": session_id}}
)
print("AI:",  response2.content)

msg3 = "제 이름이 뭐였죠?"
print("사용자:", msg3)
response3 = conversation.invoke(
    {"content": msg3},
    config={"configurable": {"session_id": session_id}}
)
print("AI:",  response3.content)
```

```
사용자: 안녕하세요!
AI: 안녕하세요! 어떻게 도와드릴까요?
사용자: 제 이름은 김철수입니다.
AI: 반갑습니다, 김철수님! 어떻게 도와드릴까요?
사용자: 제 이름이 뭐였죠?
AI: 김철수님이시라고 말씀하셨습니다! 맞나요?
```

최신 코드의 복잡성 증가 이유

- 멀티 세션 지원
  - session_id를 도입하여 각 사용자나 작업 세션을 독립적으로 관리 가능
  - 이전 코드는 단일 세션만을 처리, 글로벌 메모리를 사용하여 대화 맥락을 유지함.
- ChatMessageHistory를 사용
  - 세션별 히스토리를 (session_histories)에 보관

    - 이전 코드에서는 ConversationBufferMemory를 사용하여 단순히 메모리를 저장함.
- 최신 코드에서 ChatOpenAI를 이용.
- 모델 이름, 창의성(temperature), 출력 길이(max_tokens)등을 명시적으로 설정 가능
- 최신 코드에서 config 매개변수 추가 제공
- 대화 세션마다 개별적인 설정(예:temperature, stop 토큰)을 지정 가능
- 사용자 역할, 메타데이터 등 추가 정보를 전달하여 더 복잡한 워크 플로우 구현 가능
- Summary
    - 확장성과 다목적 지원, 대규모 애플리케이션 지원, 언어 모델의 발전으로 최신 코드의 복잡성이 증가함.