# [1] 챗지피티와 랭체인

## 챗지피티란?

- OpenAI가 2022년 11월에 공개한 챗지피티(ChatGPT)는 AI와 대화할 수 있는 웹 서비스.
- 챗지피티에는 OpenAI가 개발한 언어 모델인 GPT라는 기술이 사용됨.
  → 언어모델 : 인간의 언어(자연어)를 컴퓨터가 이해할 수 있게 하고, 이를 바탕으로 텍스트를 생성하기 위한 알고리즘이나 프로그램
- OpenAI는 이 GPT를 API로 공개하고 있으며, 직접 만든 애플리케이션에서 사용할 수 있게 지원.
- 고성능 언어 모델의 등장으로 기존 절차적 프로그래밍에서 어려웠던 기능 개발이 쉬워짐.
  RAG(Retrieval-Augmented Generation, 검색 증강 생성), ReAct(Reasoning and Acting, 추론 및 행동)등 추가기능도 만들어지고있음.

## OpenAI의 API에서 사용할 수 있는 대표적인 두 가지 언어 모델

### Chat 모델 :

- 채팅을 통해 대화를 저장하여, GPT로 하여금 이전 대화와의 경험으로 다음 질문에 대한 대답을 도출해냄.
- GPT-1, GPT-2, GPT-3, GPT-3.5, GPT-4 등 여러 모델이 있음.
  권장 버전은 3.5, 4이지만, 4는 3.5에 비해 고도화되어있어서 이용료가 비싸기 때문에 3.5로도 충분한 성능 사용 가능.
- 모델 선택 시 컨텍스트 길이 를 고려해야함.
  - 컨텍스트 길이 : 모델이 한 번에 처리할 수 있는 텍스트의 길이(토큰 수)
  - 일반 모델은 4k(4000)까지이지만 gpt-3.5-turbo-16K 모델은 16K까지 처리 가능.

```python
response = openai.ChatCompletion.create(
    model="gpt-3.5-turbo",      #호출할 언어 모델의 이름
    messages=[
        {
            "role": "user",     #user라는 역할로 openAI에서 제공하는 패키지 사용
            "content": "냉면의 원재료를 알려줘"     #입력할 문장(프롬프트)
        },
    ],
    max_tokens=100,
    temperature=1,
    n=2,
)

print(json.dumps(response, indent=2, ensure_ascii=False))   #결과 표시
```

## Completions 모델 :

- 특정 문장을 이어서 완성 시키거나 새로운 텍스트를 형성하는 특징
  ex) 완성되지 않은 문장을 제출했을 때.
- 현재 Complete에서 사용할 수 있는 모델은 GPT-3.5 계열만 존재.

```python
response = openai.Completion.create(    #Chat은 ChatCompletion, Complete는 Completion.
    engine="gpt-3.5-turbo-instruct",    #Chat은 model, Complete는 engine.
    prompt="오늘 날씨가 매우좋고 기분이",   #Chat은 message, Complete는 prompt 이 작성 방식의 차이가 chat과 complete의 가장 큰 차이!
    stop=".",
    max_tokens=100,
    n=2,
    temperature=0.5
)

print(json.dumps(response, indent=2, ensure_ascii=False))
```

# 모델 업데이트 (증분 업데이트)

GPT-3.5, GPT-4 업데이트는 증분 업데이트 형식으로 진행됨.

- 증분 업데이트 : 조금씩 기능을 추가하거나 수정하는 방식. 전체 기능 수정에 시간을 들이는 대신 단기간에 반복적으로 수정하는 개발 방식.
- gpt-3.5-turbo, gpt-3.5-turbo-16K 등 뒤에 4자리 숫자가 없는 모델은 최신 모델을 의미하고, 자동 업데이트됨.
  뒤에 4자리 숫자가 붙은 경우(gpt-3.5-turbo-0613 등)는 특정 버전이 고정된 것.
  → 고정 버전 모델은 특정 결과가 필요하거나 업데이트로 인한 결과의 변동을 피하고 싶을 때 사용.

# 요금 (종량제)

사용량에 따라 요금이 부과되는 종량제를 채택.

사용량은 API 호출 횟수가 아닌, 사용한 토큰 수를 기준으로 함.
- 토큰 : 언어 모델이 정보를 처리할 때 사용하는 최소 단위. 언어에 따라 차이가 있음. ( ex 영어는 1단어 1토큰, 한국어는 1문자당 1~2토큰)

## 랭체인이란?

언어 모델을 이용한 애플리케이션 개발을 지원하는 오픈소스 라이브러리.

→ 언어 모델이 아니라 개발을 돕는 라이브러리이므로 외부 언어 모델과 함께 사용해야함. (OpenAI 등)

## 랭체인에 준비된 6개의 모듈

- **Model I/O - 언어 모델을 쉽게 다루기**

  : 언어 모델을 호출하기 위해 필요한 '프롬프트 준비', '언어 모델 호출', '결과 수신'의 세 단계를 쉽게 구현할 수 있는 기능을 제공.

- **Retrieval - 알 수 없는 데이터를 다루기**

  : 미지의 정보를 RAG를 통해 해결

- **Memory - 과거의 대화를 장/단기적으로 기억하기**

  : 이전 문맥을 고려한 대화 형식으로 응답하게 하려면 이전까지의 대화를 API로 전송해야하는데, 이 기능을 제공

- **Chains - 여러 프로세스를 통합하기**

  : 여러 모듈과 다른 기능을 조합해 하나의 애플리케이션을 만들 수 있는데, 이 기능을 제공

- **Agents - 자율적으로 외부와 상호작용해 언어 모델의 한계를 뛰어넘기**

  : ReAct나 OpenAI Function Calling 기법을 사용해 언어 모델 호출로는 대응할 수 없는 작업을 실행(외부와 상호작용)

- **Callbacks - 다양한 이벤트 발생 시 처리하기**

  : 랭체인으로 만든 애플리케이션에서 이벤트 발생 시 처리를 수행하는 기능을 제공. 단독 사용 불가, 다른 모듈과 결합이 전제.
  주로 로그 출력이나 외부 라이브러리와의 연동에서 사용됨.

# ChatGPT API의 매개변수

ChatGPT Completions API의 매개변수 정리 (tistory.com)

# json.dumps에 관해

Python: JSON 개념과 json 모듈 사용법 (tistory.com)

- **indent=4**

  **indent**에 양의 숫자를 입력하면 입력한 숫자만큼의 공백(줄넘김 포함)이 생겨 보기 쉽게 정렬됨.
  문자가 입력된다면 그 문자가 indent를 표시하기 위해 사용. (ex) \t
  0, 음수 또는 “”의 들여쓰기 수준은 줄 넘김만 삽입.

- **ensure_ascii=False**

  dump는 파일 형태로 반환,  dumps는 string 형태로 반환.
  그래서 dumps 결과값에 한글이 들어가있으면 깨지는 현상 발생.
  그럴 때 깨지는 현상 막기 위해서 사용.

  - 깨지는 이유 : Python JSON에서 ensure_ascii=True값이 기본 값이며, 이는 JSON encoder에서 저장하는 문자열은 ascii 코드 값을 유지하도록하지만, utf8로 인코딩된 한글은 ascii 이외의 값을 가지고 있기 때문에 한글이 깨짐. 따라서 ensure_ascii=False값으로 설정하여 문자열의 값을 그대로 유지하여 한글이 깨지지 않도록 하는 것.

OpenAI의 Chat과 Complete 두 모델은 각기 다른 특성과 용도를 가지고 있으며, 따라서 기본적인 API 호출 방법과 결과 읽기 방법이 다름. → 적합한 작업에 따라 적절한 모델을 선택하는 것이 중요

# Chat / Complete 모델 정리

## Chat 모델

**특징:**

- **대화 지향:** 주로 사용자와의 자연스러운 대화를 목적으로 설계됨.
- **컨텍스트 유지:** 이전 메시지를 기억하고 대화의 맥락을 유지하는 능력이 뛰어남.
- **대화 흐름 관리:** 질문 응답, 팔로우업 질문, 대화 주제 변경 등 다양한 대화 시나리오를 자연스럽게 처리.

**적합한 작업:**

- **고객 지원:** 고객 문의에 실시간으로 응답하고 문제 해결 지원.
- **챗봇:** 웹사이트, 앱, SNS에서 사용자와의 상호작용.
- **튜토리얼 및 교육:** 교육 자료 제공, 학습자 질문에 응답.
- **상담:** 심리 상담, 생활 상담 등 다양한 분야에서의 대화형 지원.
- **게임 내 대화:** 게임 캐릭터와의 상호작용을 통해 몰입도 향상.

## Complete 모델

**특징:**

- **범용 텍스트 생성:** 다양한 종류의 텍스트를 생성하는 데 최적화.
- **단발성 작업:** 한 번의 요청에 대한 응답 생성에 중점을 둠.
- **창의적 작성:** 창의적이고 복잡한 텍스트 생성 능력이 뛰어남.

**적합한 작업:**

- **문서 작성:** 보고서, 기사, 블로그 포스트 등 다양한 문서 작성.
- **코드 생성 및 완성:** 프로그래밍 코드 작성 및 보완.
- **콘텐츠 생성:** 소설, 시, 마케팅 카피 등 창의적인 텍스트 작성.
- **번역 및 요약:** 텍스트 번역, 요약 제공.
- **데이터 분석:** 텍스트 기반 데이터 분석 및 요약.

## 요약

- **Chat 모델:** 실시간 대화와 지속적인 상호작용이 필요한 작업에 적합.
- **Complete 모델:** 단발성 요청에 대한 높은 품질의 텍스트 생성이 필요한 작업에 적합.

# [2] Model I/O - 언어 모델을 다루기 쉽게 만들기

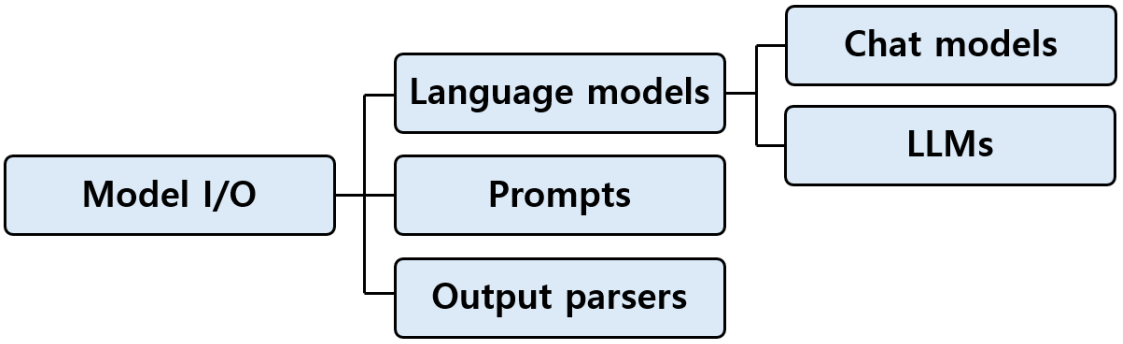
## {1} 언어 모델을 이용한 응용 프로그램 작동 방식

### 언어 모델 호출이란?

- 텍스트 상자에 메시지를 입력하고 보내기 버튼을 클릭하면 결과가 출력
  → 텍스트 상자에 입력한 메시지를 통해 언어 모델을 호출하는 것
- **프롬프트** : 언어 모델을 호출할 때 입력되는 텍스트

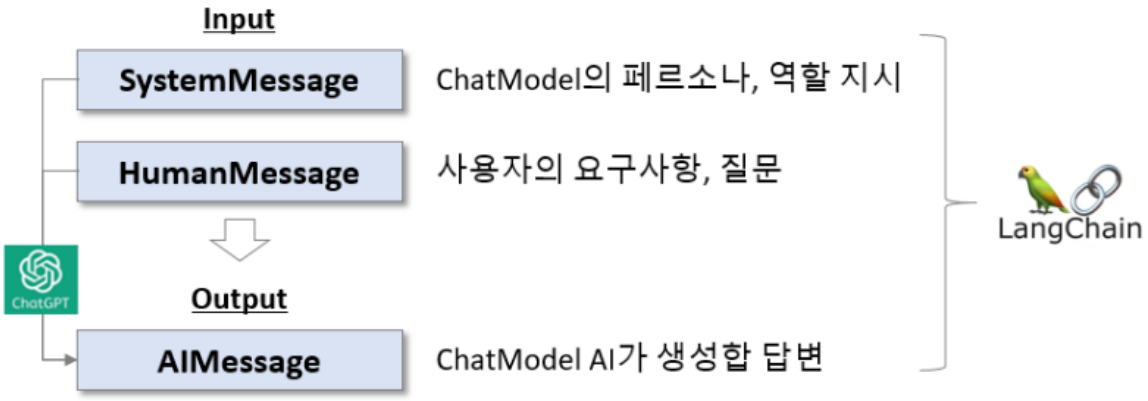
```python
import openai

response = openai.ChatCompletion.create(
    model="gpt-3.5-turbo",      #호출할 언어 모델의 이름
    messages=[
        {
            "role": "user",     #user라는 역할로 openAI에서 제공하는 패키지 사용
            "content": "냉면의 원재료를 알려줘"     #입력할 문장(프롬프트)
        },
    ],
    max_tokens=100,
    temperature=1,
    n=2,
)

print(json.dumps(response, indent=2, ensure_ascii=False))   #결과 표시
```

## Model I/O를 구성하는 3개의 서브모듈




## {1} Language models - 사용하기 쉬운 모델

- **다양한 언어 모델을 동일한 인터페이스에서 호출할 수 있는 기능을 제공**
- **Chat models** : OpenAI의 Chat 모델과 같은 대화형식으로 사용하는 언어모델을 다룸
  → HumanMessage 다음 응답을 예측. 대화형 텍스트 생성(챗봇) 개발에 적합.
  - SystemMessage, HumanMessage, AIMessage의 3가지 종류의 메시지 클래스를 제공.




```python
from langchain.chat_models import ChatOpenAI    #모듈 가져오기
from langchain.schema import HumanMessage    #사용자의 메시지인 HumanMessage 가져오기


chat = ChatOpenAI(      #클라이언트를 만들고 chat에 저장
    model = "gpt-3.5-turbo",    #호출할 모델 지정
)

result = chat(      #실행하기
    [
        HumanMessage(content="안녕하세요!")
    ]
)

print(result.content)
```

**결과** : 안녕하세요! 무엇을 도와드릴까요?

- 위 코드에서 실행문에 SystemMessage(content="당신은 친한친구입니다. 존댓말을 쓰지 않고 솔직하게 답해주세요") 와 같은 SystemMessage를 설정하여 성격이나 설정등을 입력하면 답변의 문체를 바꿀 수 있음. (ex 안녕! 잘 지내니? 와 같은 형태로 대답)

- **LLMs** : OpenAI의 Complete 모델과 같은 문장의 연속을 준비하는 언어모델을 다룸
→ 하나의 프롬프트만 고려. 대화가 아닌 문장의 연속을 예측. (내용의 생성이 목적)

```
llm = OpenAI(model="gpt-3.5-turbo-instruct")

result = llm(
    "맛있는 라면을",
    stop="."
)

print(result)
```

**결과** : 먹고싶어요

- **Language models의 편리한 기능**
  - **캐싱 - InMemoryCache**

    사용한 토큰 수에 따라 요금이 부과되므로, 같은 프롬프트를 두번 전송하면 두 번 분량의 요금이 부과됨. 또한 당연히 API를 두번 호출하게 되어 효율성도 떨어지므로 이러한 문제를 해결하기 위해 캐싱 기능을 제공.

    ```
    #llm_cache에 InMemoryCache 설정
    langchain.llm_cache = InMemoryCache()
    ```

    InMemoryCache는 메모리 내에 데이터를 일시적으로 보관하는 캐시 방법을 제공하는 클래스.

    위 코드를 삽입하면 특정 요청에 대한 응답이 한번 생성되면 캐시에 저장되며, 동일한 요청이 발생했을 때 이미 저장된 응답을 즉시 제공.

    메모리 내 캐시는 프로그램이 실행되는 동안은 유지되지만, 종료되면 삭제됨.

- **결과 순차표시(Streaming 모듈)**

  순차적 표시란 처리가 완료되기 전에 일부 결과를 순차적으로 수신해 표시하는 것.

  긴 응답을 생성하거나 사용자에게 실시간 응답을 제공하고자 할 때 유용함.

  랭체인에서는 이 Streaming 모듈을 활용하기 위해 Callbacks 모듈을 제공.

```
chat = ChatOpenAI(
        streaming=True,    #스트리밍 모드 실행
        callbacks=[
            StreamingStdOutCallbackHandler()    #결과를 터
        ]
)
```

## {2} Prompt Templates - 프롬프트 구축의 효율성 향상

- 언어모델의 input은 'text'이기 때문에, 원하는 답변을 이끌어내기 위해서는, 입력되는 text를 최적화하는 것이 필수적.
- 프롬프트 최적화를 통해 단순한 명령어로는 어려웠던 작업을 수행할 수 있도록 만드는 것이 '프롬프트 엔지니어링'
- **프롬프트 엔지니어링** : 프롬프트를 최적화하는 과정, 그 결과로 얻어지는 개선된 결과물
  ex) 과학 논문 요약 생성, 전문 지식이 필요한 문장 작성 등
- Templates 모듈은 프롬프트 엔지니어링을 돕고, 프롬프트를 쉽게 구축할 수 있는 기능을 제공

- **변수와 문자열 조합하기**

```
from langchain import PromptTemplate    #PromptTemplate 가져오기

prompt = PromptTemplate(    #PromptTemplate 초기화하기
    template = "{product}는 어느 회사에서 개발한 제품인가요?",  #{product}라는 변수를 포함하는 프롬프트 작성하기

    input_variables = [
        "product"       #product에 입력할 변수 지정
    ]
)

print(prompt.format(product="아이폰"))
print(prompt.format(product="갤럭시"))
```

**결과** : 아이폰은 어느 회사에서 개발한 제품인가요? / 갤럭시는 어느 회사에서 개발한 제품인가요?

만약 위 print문에서 product를 지정하지 않은 상태로 실행하면 KeyError: 'product' 라는 오류가 표시됨. (유효성검사 기능)

- input_variables를 직접 지정하지 않고 템플릿에서 직접 초기화할 수 도 있음

  **prompt = PromptTemplate.from_template("{product}는 어느 회사에서 개발한 제품인가요?")**

- JSON 파일을 읽어들여 PromptTemplate을 만드는 방법도 있음

  **loaded_prompt = load_prompt("prompt.json")**
  **print(loaded_prompt.format(product="아이폰"))**

- **Language models + PromptTemplate**

```python
#Language Model
chat = ChatOpenAI(
    model = "gpt-3.5-turbo",
)

#PromptTemplate
prompt = PromptTemplate(
    template = "{product}는 어느 회사에서 개발한 제품인가요?",

    input_variables = [
        "product"
    ]
)

#Language Model
result = chat(
    [
        HumanMessage(content=prompt.format(product="아이폰")),
    ]
)

print(result.content)
```

결과 : 아이폰은 애플 회사에서 개발한 제품입니다.

- **출력 예제가 포함된 퓨샷 프롬프트**

```
#입력, 출력 예시
examples = [
    {
        "input" : "충청도의 계룡산 전라도의 내장산은 모두 국립공원이다",
        "output" : "충청도의 계룡산, 전라도의 내장산은 모두 국립공원이다."
    }
]

#PromptTemplate 준비
prompt = PromptTemplate(
    input_variables=["input", "output"],
    template="입력 : {input}\n출력 : {output}",
)

#FewShotPromptTemplate 준비
few_shot_prompt = FewShotPromptTemplate(
    examples = examples,         #입력 예시와 출력 예시를 정의
    example_prompt = prompt,     #FewShotPromptTemplate에 PromptTemplate를 전달

    #지시어 추가
    prefix = "아래 문장부호가 빠진 입력에 문장부호를 추가하세요. 추가할 수 있는 문장부호는 ',', '.' 입니다.",

    suffix="입력 : {input_string}\n출력 : ",     #출력 예의 입력 변수를 정의
    input_variables = ["input_string"],         #FewShotPromptTemplate의 입력 변수를 설정
)

#FewShotPromptTemplate를 사용해 프롬프트 작성
formatted_prompt = few_shot_prompt.format(
    input_string = "집을 보러 가면 그 집이 내가 원하는 조건에 맞는지 살기에 편한지 확인해야한다"
)

result = llm.predict(formatted_prompt)
```

**결과** : 입력 : 집을 보러 가면 그 집이 내가 원하는 조건에 맞는지 살기에 편한지 확인해야한다

출력 : 집을 보러 가면 그 집이 내가 원하는 조건에 맞는지, 살기에 편한지 확인해야한다.

## {3} Output parsers - 출력 구조화

- **Output parsers**

  : 언어 모델에서 얻은 출력을 분석해 애플리케이션에서 사용하기 쉬운 형태로 변환하는 기능을 제공

  - **결과를 날짜와 시간 형식으로 받아보기(DatetimeOutputParser 사용)**

```
#언어 모델의 출력을 날짜 및 시간 형식으로 변환하는 DatetimeOutput
output_parser = DatetimeOutputParser()

(중략)
```

```
#출력 결과를 분석해 날짜 및 시간 형식으로 변환
output = output_parser.parse(result.content)
```

**결과** : 2020-09-01 00:00:00

- **출력 형식을 직접 정의하기(PydanticOutputParser 사용)**
  - PydanticOutputParser는 Pydantic 모델을 기반으로 언어 모델의 출력을 파싱.
  - **Pydantic 모델** : 파이썬에서 데이터 검증을 위한 라이브러리. type 힌트를 이용해 데이터 모델을 정의하고, 데이터 분석과 검증을 수행하는 도구.

```
#Pydantic 모델 정의
class Smartphone(BaseModel):
    #Field를 사용해 설명을 추가
    release_date: str = Field(description="스마트폰 출시
    model_name: str = Field(description="스마트폰 모델명

#PydanticOutputParser를 SmartPhone 모델로 초기화
parser = PydanticOutputParser(pydantic_object=Smartp

(중략)

#PydanticOutputParser를 사용해 문장을 파싱
parsed_result = parser.parse(result.content)
print("모델명: {parsed_result.model_name}")
print("스마트폰 출시일: {parsed_result.release_date}")
```

**결과** : 모델명 : Samsung Galaxy S21
스마트폰 출시일 : 2021-01-01

- **잘못된 결과가 반환될 시 수정을 지시(OutputFixingParser 사용)**

```
#OutputFixingParser를 사용하도록 작성
parser = OutputFixingParser.from_llm(
    #parser 설정
```

```
    parser = PydanticOutputParser(pydantic_object=Smart
    #수정에 사용할 언어 모델 설정
    llm = chat
)
```

# [3] Retrieval - 알지 못하는 데이터를 다루기

## {1} 언어 모델이 미지의 데이터를 처리할 수 있게 하려면

- GPT와 같은 언어 모델은 학습한 정보를 바탕으로 답변을 생성하지만, 학습하지 않은 내용에 대해서는 답변할 수 없음
  →
  **RAG(Retrieval-Augmented Generation)**를 사용하여 해결 가능
  : 사용자가 입력한 내용과 관련된 정보를 외부 데이터베이스 등에서 검색하고, 그 정보를 이용해 프롬프트를 만들어 언어 모델을 호출.
  이를 통해 학습하지 않은 지식이나 정보도 답변.

### RAG(Retrieval-Augmented Generation)

- **사용 절차**
  1. 사용자에게 질문받기
  2. 준비된 문장에서 답변에 **필요한 부분** 찾기
  3. 문장의 관련 부분과 사용자 질문을 조합해 프롬프트 생성하기
  4. 생성한 프롬프트로 언어 모델을 호출해 사용자에게 결과 반환하기
- **답변에 필요한 문장을 찾는 방법**
  - 답변에 필요한 문장을 어떻게 검색하고 가져오는지가 중요.
  - 텍스트를 컴퓨터가 이해할 수 있도록 텍스트를 숫자의 조합으로 표현하는 과정 : **텍스트 벡터화**

    **(의미를 고려하면서 수치로 표현하는 것)**

    → 수치로 표현하기 때문에 계산을 통해 **유사성 여부**를 확인 가능
  - OpenAI는 **'text-embedding-ada-002'**라는 언어 모델을 API로 제공. 이를 통해 텍스트 벡터화 진행

    ```
    embeddings = OpenAIEmbeddings(
        model = "text-embedding-ada-002"
    ```

```
)
#질문을 벡터화
query_vector = embeddings.embed_query("비행 자동차의 최고 ≒
```

## 벡터 유사도 검색

- 'text-embedding-ada-002'는 1536차원의 벡터를 출력. (1536개의 숫자를 가진 배열이 출력)
- 유사도의 경우, **코사인 유사도**를 사용해 유사도를 계산하는 것을 권장.
  → 두 벡터간의 코사인 각도를 이용해 유사도를 계산하는 방법으로, 0부터 1사이의 값을 가지며 1에 가까울수록 유사도가 높은 것으로 간주됨.
  (0.6666666667, 0.0, 0.475123645 등등)


## 벡터 유사도 검색에서 RAG를 통합하기 위한 사전준비

1. **텍스트 추출(Document loaders)**
   - RAG 기법을 적용하기 위해서 **모든 정보는 텍스트 형태**여야 하기 때문에, PDF, URL과 같은 형태의 정보를 텍스트화 해야 함.
   - 이러한 문제에 대응하기 위해 정보 획득의 원천이 되는 **텍스트 준비를 보조**하는 것이 **Document loaders** 모듈.
     (PDF, 엑셀 등의 파일이나 URL을 입력하면 웹 페이지 내 텍스트만 추출하는 기능)
     → 원본이 이미 텍스트 형식인 경우에는 Document loaders가 필요하지 않음.

2. **텍스트 분할(Text splitters)**
   - 너무 긴 텍스트는 언어 모델이 처리할 수 있는 글자 수 한계가 있음.

- 긴 텍스트를 적절한 길이로 나눌 때, 추출된 텍스트에서 의미가 있고 파편화되지 않은 위치에서 분할 필요.
  →
  **적절한 위치에서 적절한 길이**로 **문장을 분할**할 수 있는 기능을 제공하는 것이 **Text splitters**.

3. **텍스트 벡터화(Text embedding models)**
   - 벡터화의 목적 : 이후 단계에서 의미가 가까운 문장을 검색할 수 있게 하기 위함. (유사도 비교)
   - Retrieval 모듈에 Text embedding models, Llama를 사용한 LlamaCppEmbedding 등이 있음.

4. **텍스트와 벡터를 데이터베이스에 저장(Vector stores)**
   - 벡터화된 텍스트(숫자 배열)을 저장하는데 특화된 DB가 바로 벡터DB.
   - 벡터 데이터베이스에 쉽게 데이터를 투입할 수 있는 기능이 Vector stores이며, 파인콘(Pinecone), 크로마DB(ChromaDB) 등 다양한 종류의 벡터 데이터베이스가 존재

### 벡터 유사도 검색에서 RAG를 통합하기 위한 검색 및 프롬프트 구축

1. 사용자의 입력을 벡터화(Text embedding models)
2. 사용자 입력의 벡터를 미리 준비된 DB에서 검색해 문장 가져오기(Vector stores)
3. 획득한 유사 문장과 질문을 조합해 프롬프트 작성
   - Model I/O 모듈의 PromptTemplate을 사용해 유사 문장과 질문을 조합.
4. 생성한 프롬프트를 사용해 언어 모델 호출(Language models)
   - 3번에서 생성한 프롬프트를 사용해 Model I/O 모듈의 Chat models(언어 모델)를 호출

## {2} RetrievalQA로 QA 시스템 구축이 쉬워진다

### RetrievalQA

- RAG  기법을 이용한 QA 시스템을 보다 쉽게 개발할 수 있고, 다기능으로 만들기 위한 모듈.
- **검색, 프롬프트 구축, 언어 모델 호출 처리의 구현을 단순화**함.
- **Retrievers**를 사용해 쉽게 구현.

  1. 사용자가 질문을 입력
  2. Retrievers가 입력된 질문을 기반으로 관련 문서를 검색
  3. 검색된 문서를 LLM에 입력해서 질문에 대한 답변을 생성
  4. 생성된 답변 사용자에게 반환

  → 앞서 설명한 정보원 텍스트 구축, PromptTemplate을 이용한 프롬프트 구축 처리가 **코드에서 삭제되고, RetrievalQA 내에서 수행**됨.(단순화)

## Retrievers - 문서를 검색하는 기능 세트

- **Retrievers** : 특정 단어로 검색을 하면 관련된 여러 문서(문장)을 얻을 수 있는 일련의 기능을 총칭.
  - **질의(Query) 처리** : 사용자의 질문을 이해하고, 이를 기반으로 적절한 문서나 정보를 검색할 수 있는 질의로 변환.
  - **검색(Searching)** : 사전 정의된 인덱스나 데이터베이스, 검색 엔진 등을 통해 사용자 질의에 맞는 문서를 검색.
  - **결과 반환(Returning Results)** : 검색된 문서나 정보를 반환.
- 앞서 설명한 RetrievalQA는 받은 Retrievers를 이용해 문장을 검색하고, 검색된 문장을 기반으로 답변을 생성하는 기능을 가지고 있음
  → 즉, Retrievers를 교체함으로써 정보원 변경 가능

## RePhraseQueryRetriever

- RePhraseQueryRetriever를 사용하면 사용자의 자연스러운 질문을 적절한 키워드로 변환.
  ex) 나는 라면을 좋아하는데, 소주란 무엇인가요? → 소주란 무엇인가요? 부분을 추출해 검색

```
re_phrase_query_retriever = RePhraseQueryRetriever(
    retriever=retriever,
```

```
)
#RePhraseQueryRetriever 사용해 키워드 추출
documents = re_phrase_query_retriever.get_relevant_documen
```

# [4] Memory - 과거의 대화를 장/단기 기억하기

## {1} HumanMessage와 AIMessage를 번갈아 가며 대화

```
HumanMessage(content="계란찜의 재료를 말해줘"),
AIMessage(content="계란찜의 재료는 계란, 물, 양파, 당근, 대파, 소금,
HumanMessage(content="위의 답변을 영어로 번역해줘")
```

위처럼 기존 언어 모델이 대화 이력을 바탕으로 답변하기 위해서는 지금까지의 모든 대화이력을 HumanMessage와 AIMessage를 번갈아가며 수동으로 전송해야함.
→ 대화 기록을 저장하고 불러올 수 있는
**Memory 모듈을 사용해 기억을 가진 시스템**을 사용해 해결

1. 메모리에서 과거 메시지 검색
2. 새로운 메시지(input) 추가
3. 이 메시지를 언어 모델에 전달해 새로운 응답(output) 얻기
4. 새로운 응답을 메모리에 저장

랭체인에서는 위와 같은 과정을 쉽게 구현할 수 있는 **ConversationChain**을 제공함.

```
#chat 모델 설정
chat = ChatOpenAI(model="gpt-3.5-turbo")

# memory와 llm을 chaining
memory = ConversationBufferMemory()
chain = ConversationChain(memory=memory, llm=chat)

# 사용자 메시지를 chain에 넣음
result = chain(message)
```

## {2} 대화 기록을 저장하기 위한 DB

위 예제와 같이 BufferMemory를 사용하면 프로세스가 죽으면 메모리에 있는 대화들은 사라지기 때문에 대화 내역을 저장하기 위해서는 데이터베이스가 필요한데, 이를 위해 **레디스(Redis)**를 사용.

### Redis

- 캐시, 메시징 큐, 단기 메모리 등으로 사용되는 고속 오픈소스 인메모리 데이터 저장 시스템
- 데이터는 키-값 쌍 형태로 저장되며 다양한 데이터 유형(문자열, 목록, 집합 등)을 지원
- 메인 메모리에 데이터를 저장하기 때문에 디스크 기반 데이터베이스보다 훨씬 빠름.
  → 메모리 내 데이터는 휘발성이 있지만, 레디스는 주기적으로 디스크에 데이터를 기록함으로써 데이터 영속성을 제공.

### RedisChatMessageHistory

- Langchain에서 제공하는 RedisChatMessageHistory 모듈을 사용하면 Redis에 대화를 손쉽게 저장 및 관리할 수 있음.

```
history = RedisChatMessageHistory(
    #세션 이름 설정
    session_id = "chat_history",

    #미리 설정해둔 Redis 접속정보 환경변수 가져오기
    url = os.environ.get("REDIS_URL"),
)
```

  - session_id : 임의의 문자열을 지정. 여러 개의 대화 세션을 동시에 처리할 때 각각의 대화 내역을 구분하기 위한 ID
  - url : 미리 설정해둔 레디스의 URL을 지정. (레디스 접속정보가 담긴 REDIS_URL 환경변수 가져오기)

### ConversationBufferMemory

- RedisChatMessageHistory와 ConversationBufferMemory를 결합해 대화 내역을 레디스에 저장하고, 애플리케이션 종료 후에도 내역을 유지

```
memory = ConversationBufferMemory(
    return_messages = True,
    chat_memory = history,
)
```

## {3} 매우 긴 대화 기록 처리

언어 모델은 컨텍스트 길이의 한계(토큰 수 제한)를 넘어서는 처리를 허용하지 않음.

### 오래된 대화 삭제 - ConversationBufferWindowMemory

```
memory = ConversationBufferWindowMemory(
    #메시지 왕복횟수 지정
    k = 3,
    return_messages = True,
)
```

위 코드에서 **k=3**은 **왕복 3번까지 메시지를 유지**한다는 설정.
→ 즉, 4번째 대화에는 첫 번째 메시지를 삭제.

### 대화 요약 - ConversationSummaryMemory

```
memory = ConversationSummaryMemory(
    #Chat models 지정
    llm = chat,
    return_messages=True,
)
```

**요약 후 저장된 메시지**

'사람이 AI에게 볶음밥 만드는 법을 가르쳐 달라고 요청합니다. AI는 필요한 재료와 조리 과정 등 볶음밥을 만드는 과정을 단계별로 안내합니다 .또한 재료와 양념의 양은 개인 취향에 따라 조절할 수 있다고 알려줍니다. AI는 사람이 더 궁금한 점이 있는지 묻습니다.'

# [5] Chains - 여러 프로세스를 통합

**Chains : 일련의 처리를 하나의 묶음으로 처리할 수 있는 모듈**

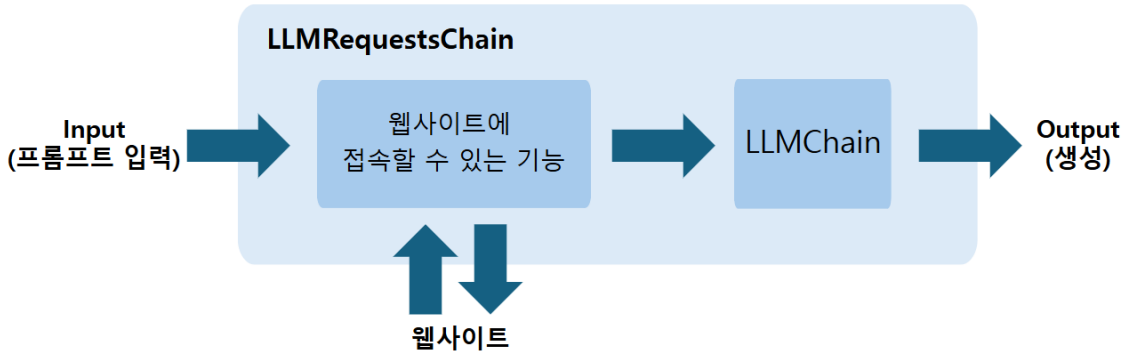
## 여러 모듈의 조합을 쉽게 함 - LLMChain, ConversationChain

- 예를 들어, **LLMChain**을 사용하면 PromptTemplate을 이용한 프롬프트 구축 + Chatmodels를 이용한 언어 모델 호출을 한번에 처리할 수 있음.


```
#LLMChain을 생성
chain = LLMChain(
    llm = chat,
    prompt = prompt,
)

#LLMChain을 실행
result = chain.predict(product=" ")
```

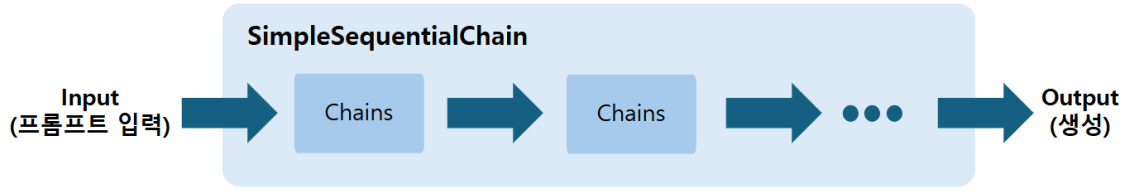
## 특정 용도에 특화된 체인 - LLMRequestsChain

- 언어 모델의 호출만으로는 대응하기 어려운 기능이나 복잡한 처리를 랭체인 측에서 미리 내장해 특정 용도에 특화된 Chains
- 예를 들어, **LLMRequestsChain**은 주어진 URL에 접속해 얻은 결과와 질문을 조합해 만든 프롬프트로 언어 모델을 호출.




## 체인 자체를 정리함 - SimpleSequentialChain

- 위에서 설명한 것처럼 하나의 Chains는 **'기능 덩어리'**
- 여러개의 Chains를 순서대로 실행하거나 필요에 따라 호출할 수 있도록 **Chains 자체를 묶을 수 있음.**




```
#기사를 쓰는 LLMChain
write_article_chain = LLMChain(
    (중략)
)

#번역하는 LLMChain
translate_chain = LLMChain(
    (중략)
)

#SimpleSequentialChain 생성
sequential_chain = SimpleSequentialChain(
    chains=[
        write_article_chain,
        translate_chain,
```

```
        ]
)
```

# [6] Agents - 자율적으로 외부와 상호작용 해 언어 모델의 한계를 뛰어넘기

## {1} 외부와 상호작용 하면서 자율적으로 행동하는 Agents

언어 모델 만으로는 텍스트를 전송하고 텍스트를 수신하는 것 이상을 할 수 없지만, Agents 모듈을 사용하면 다양한 작업을 수행할 수 있음.

Agents 모듈에는 **Tool, Agent** 두 가지 하위 모듈이 존재.

### Tool

- 계산, 검색과 같이 "언어 모델 만으로는 할 수 없는 일" 을 할 수 있게 하는 모듈
- 랭체인이 제공하는 Tool
  - **LLMMath(계산)** : 계산을 위한 Tool
  - **Requests(URL 호출)** : 지정된 URL로 요청을 보내고, 웹사이트의 정보를 가져오거나 공개된 API에서 정보를 가져오는데 사용.
  - **File System(로컬 파일 접근)** : PC 내 파일에 접근해 파일을 읽고 쓰는데 사용.
  - **SerpApi(검색)** : 구글이나 야후 검색을 API로 하는 SerpApi라는 웹 서비스와 연동.

  → 이 외에 이름, 기능 설명, 실행 함수만 준비하면 Tool을 직접 만들어서도 사용 가능. (Retrievers를 Tool로도 사용 가능)

```
#Tool을 추가
tools.append(
    #Tool의 기본구성
    Tool(
        name = "이름"
        description = "기능 설명",
        func = 실행 함수
```

```
    )
)
```

## Agent

- Tool을 선택하고 **다음 단계의 처리를 수행**하는 주체
- 단순히 Tool을 조작하는 것이 아닌, 스스로 어떤 Tool을 어떻게 사용하면 좋을지 고민하고 실행, 결과 검증까지 진행.
- Agent에서 가장 많이 사용하는 **'ReAct 기법'**
  1. 사용자로부터 작업을 받음
  2. 준비된 Tool 중에서 어떤것을 사용할지, 어떤 정보를 입력할지 결정
  3. Tool을 사용해 결과를 얻음
  4. 얻은 결과를 통해 과업이 달성되고 있는지 확인
  5. 에이전트가 작업을 완료했다고 판단할 수 있을 때까지 2~4번 반복

```
agent = initialize_agent(
    tools,
    chat,
    agent = AgentType.에이전트 유형,     #작동 방식 설정
    verbose=True                          #실행중 로그 표시
)
```

- **에이전트 유형의 종류**
  - **ZERO_SHOT_REACT_DESCRIPTION** : 주어진 질문이나 작업에 대해 즉각적으로 반응하여 답변을 생성.(챗봇, 정보제공 도우미 등)

```
llm = OpenAI(model="text-davinci-003")
agent = ZeroShotAgent(llm=llm)

query = "세계에서 가장 높은 산은 무엇인가요?"
```

  - **CHAT_ZERO_SHOT_REACT_DESCRIPTION** : 대화형으로, 사용자의 질문과 상호작용하여 대화를 이어나감. 문맥을 이해하고, 이전 내용을 기억하여 더 자연스러운 상호작용을 제공.(고객 서비스 챗봇, 개인 비서, 대화형 정보 제공 시스템 등)

```
llm = OpenAI(model="text-davinci-003")
agent = ConversationalAgent(llm=llm)

conversation = [
    {"role": "user", "content": "안녕하세요?"},
    {"role": "assistant", "content": "안녕하세요! 무엇을 도와드릴까요?"},
    {"role": "user", "content": "오늘 날씨가 어떤가요?"}
]
```

- **STRUCTURED_CHAT_ZERO_SHOT_REACT_DESCRIPTION** : 특정 도구나 API를 호출하여 복잡한 작업을 수행. 계산기, 캘린더, DB조회, 이메일발송 등 다양한 도구 활용 가능.(데이터 분석 및 보고서 생성, 일정 관리 및 예약 등)

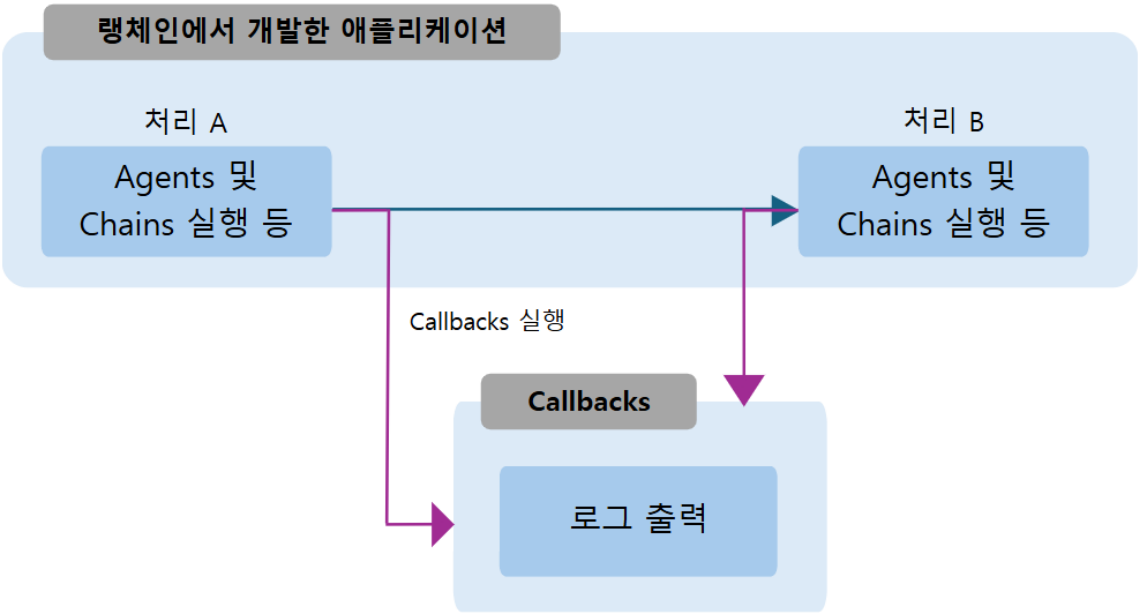
```
calculator = Calculator()
agent = ToolUsingAgent(tools=[calculator])

query = "123 곱하기 456은 얼마인가요?"
```

# [7] Callbacks - 다양한 이벤트 발생 시 처리하기

## Callbacks 모듈

```
chat = ChatOpenAI(
    model = "gpt-3.5-turbo",
    callbacks = [ 지정할 콜백 ]
)
```

- 언어 모델을 사용하는 애플리케이션에서 이벤트 발생 시 특정 처리를 실행하는 기능.
- 상세한 실행 로그를 파일이나 터미널로 출력 가능.
- 사용자가 직접 Callbacks를 구현할 수 도 있음.(보통 클래스 형태)




- **Callbacks로 연동할 수 있는 라이브러리 및 서비스**
  - **Streamlit** : 웹 애플리케이션을 간단하게 만들고 배포할 수 있는 오픈소스 프레임워크

    → Callbacks와 연동을 통해 실시간 데이터 시각화 대시보드 생성

- **LLMonitor** : AI를 사용한 애플리케이션의 성능을 모니터링하고, 추적하며, 분석할 수 있는 도구.

  → Callbacks와 연동을 통해 모델 성능 모니터링 및 분석

- **Context** : 특정 작업이나 질문에 대한 추가적인 정보나 배경 지식을 제공하는 도구.

  → Callbacks와 연동을 통해 문맥 정보 추가 및 통합