

# Ch01. 생성형 AI 개발 Architecture

- 생성형 AI 애플리케이션과 LLM
- Flask, Spring 및 LangChain과의 Integration

## 생성형 AI 애플리케이션과 LLM

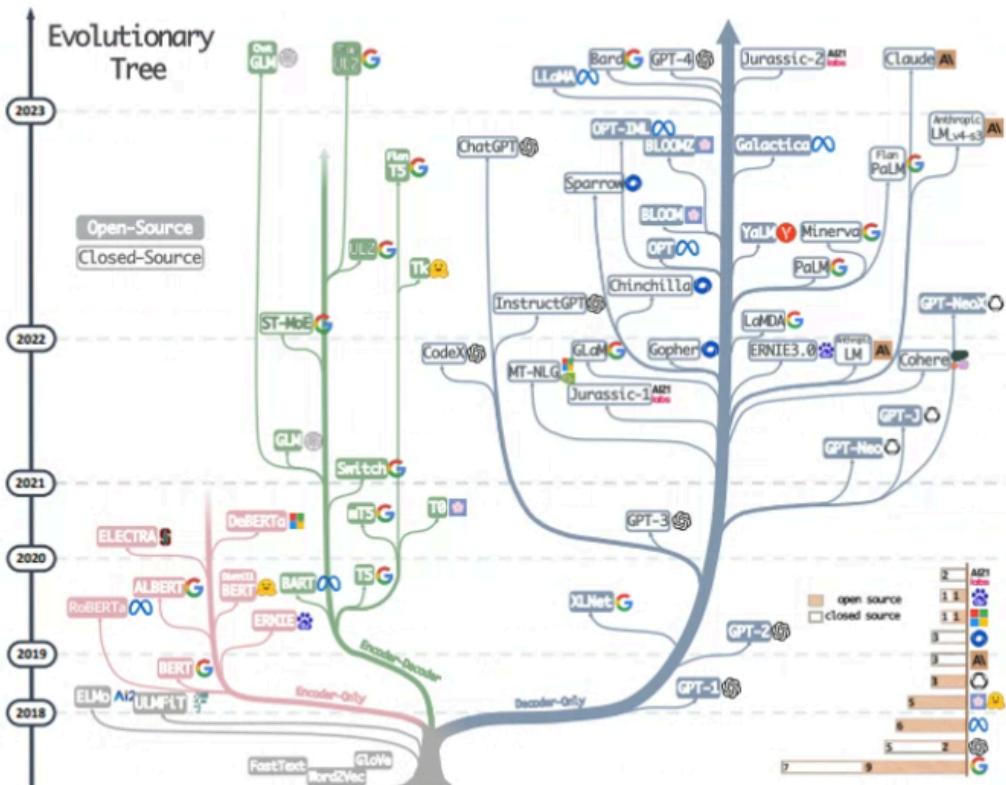
---

- 급부상 하는 LLM
  - 초기 접근 방식의 한계
    - 1950~60년대 : **Symbolic(기호)** 추론과 **Rule 기반** 시스템에 중점 - 매우 취약하고 능력이 제한됨
    - 1980년대 : Rule 기반 프로그래밍으로는 인간 지능의 다양성과 견고함을 재현할 수 없음을 깨달음.  
*(Expert System - Prolog, LISP)*
    - 1980년대 : **기계학습 기법** 출현하여 컴퓨터 비전과 음성 인식과 같은 분야에서는 어느 정도 진전이 있음.  
그러나 인공 일반 지능(AGI)을 달성하겠다는 목표는 요원함.
    - 2000년대 초반 : **딥러닝 신경망(deep learning neural network)** 등장 - Geoffrey Hinton등이 제안함(2006년)
    - 2010년대 : 딥러닝은 마침내 돌파구를 찾음. 충분한 데이터와 컴퓨팅 파워를 갖춘 심층 신경망은 이미지 분류와 음성 인식과 같은 분야에서 놀라운 정확도를 달성함.
      - 또 다른 도전과제 : 지도 학습을 위해 대규모로 레이블이 지정된 데이터셋이 필요하다는 점.
      - 자가 지도 학습(self-supervised learning) : 레이블이 없는 데이터 자체로 부터 표현을 생성하도록 거대한 신경망 모델을 훈련해, 시스템은 강력한 특징을 학습하게 됨
      - Auto-Regressive : 신경망이 텍스트 시퀀스에서 다음 단어를 예측하도록 훈련됨.

## 생성형 AI 애플리케이션과 LLM

### ▪ 신경망과 트랜스포머, 그리고 그 너머

- 신경 언어 모델링(neural language modeling) 분야가 발전하게 됨. 이는 자연어 처리 작업에 활용할 수 있었음.
- 2013년 word2vec : Google, **비지도 학습 접근 방식으로 레이블이 없는 텍스트 데이터**에서 단어 임베딩을 생성하도록 얇은 신경망(shallow neural network)을 효율적으로 훈련하는 방식임.  
(문맥 의존성 부족 등 문제 존재(word2vec은 하나의 단어당 하나의 벡터만 학습함))
- 2018년 ELMo(Embeddings from Language Models) : Univ of Washington, LSTM 사용
- 2018년 후반 BERT(Bidirectional Encoder Representation form Transformer) : Google
  - 2017년 Transformer 이론을 기반으로 구현된 모델임("Attention is All You Need")
  - BERT는 레이블이 없는 텍스트에 대한 masking 언어 모델이라는 새로운 사전 학습 방식을 도입함.
  - 사전 학습된 BERT 모델은 작업 데이터셋에서 단순히 파인 티닝만으로도 다양한 NLP 작업에서 큰 성능 향상을 보임.
  - BERT의 엄청난 성공은 NLP에서 "사전 학습 후 파인튜닝" 이란 패러다임을 확립함.
- 2020년 OpenAI는 모델 파라미터가 1750억 개인 GPT-3를 제안함.
  - 제로샷 및 퓨샷 학습 능력을 보여줌, 이 모델에서부터 부터 크기의 중요성이 강조되기 시작함.
  - 수천억 개 그 이상의 파라미터 가진 LLM이 등장하기 시작함  
(PaLM : 2022년 Google 5400억 파라미터)



## 생성형 AI 애플리케이션과 LLM

- 신경망과 트랜스포머, 그리고 그 너머
  - PLM(Pretrained Language Model, BERT, GPT-2 등)에서 LLM으로 넘어가며 엄청난 질적 변화 생김
  - LLM은 기존에 소형 모델에서 하지 못했던 퓨처 학습, 사고 연결, 지시 사항 따르기와 같은 놀라운 능력을 보임
  - 이러한 능력은 모델 크기가 일정 임곗값을 넘으면 갑자기 나타나는 현상으로, 점진적인 확장에 따르는 개선 결과가 아님.
  - LLM은 작은 문제를 해결하는 특화된 시스템에서 **다재 다능한 범용 모델로 AI 패러다임을 전환함.**
  - LLM은 아무런 파인튜닝 없이도 여러 도메인에서 인간 수준의 질의 응답 및 추론을 수행하는 능력을 보여줌.

## 생성형 AI 애플리케이션과 LLM

- 생성형 AI와 LLM의 차이
  - **생성형 AI**는 텍스트, 이미지, 비디오와 같은 다양한 유형의 콘텐츠를 생성하는 AI 시스템을 포괄하는 더 넓은 개념임.
  - 반면에 LLM은 자연어 데이터를 처리하고 이해하도록 설계된 특정 범위의 딥러닝 모델임.
  - 그러나 **현재에는 거의 차이가 없음.**
  - 생성형 AI 애플리케이션은 LLM을 핵심으로 하는 종합 애플리케이션임.
  - 다음은 생성형 AI 애플리케이션 종류
    - 대화형 에이전트 및 챗봇
    - 코드 완성과 프로그래밍 어시스턴트
    - 언어 번역
    - 텍스트 요약 및 생성

## 생성형 AI 애플리케이션과 LLM

- 생성형 AI 애플리케이션 3계층
  - 생성형 AI 애플리케이션을 개발하고 배포하려면 다양한 구성 요소가 통합된 기술 스택을 결합해야 함.
  - **스택의 세 가지 주요 계층인 인프라 계층, 모델 계층, 애플리케이션 계층** (AWS의 CEO Andy Jassy, 2024년 4월)
- **인프라 계층**
  - 이 계층은 LLM을 개발, 훈련, 제공하는데 필요한 기본 데이터, 컴퓨팅 및 도구 자원을 포함함.
  - **데이터 저장 및 관리**
    - 훈련 데이터(petabytes 단위)는 저장, 관리, 접근이 효율적이어야 함.  
(분산 파일 시스템이 적합함 - 아마존의 S3, 구글의 클라우드 스토리지 등)
    - 클라우드 저장소에 구축된 **데이터 레이크(data lake)**는 다양한 소스에서 수집된 이질적인 데이터를 중앙 집중적으로 저장, 정리, 및 처리하는 저장소 역할을 함.
    - 전반적으로 **LLM이 고품질의 잘 조직된 훈련 데이터로부터 나오는 혜택을 받기 위해서는 견고한 데이터 관리 플랫폼이 필수적임.**

### ▪ 인프라 계층

- 데이터 저장 및 관리
- 벡터 데이터베이스
  - LLM은 단어와 문서를 의미있는 숫자들로 구성된 벡터로 표현함.
  - Pinecone, Weaviate, Milvus, Chroma : 효율적인 유사성 검색을 지원해 수십억 개의 벡터를 저장하고 색인화하는데 특화 되어 있음. Cosine similarity와 같은 지표를 통해 의미론적 관계를 포착함.
  - 벡터 DB를 활용하면, 임베딩(지식 표현)을 모델 학습이나 추론과 분리해 독립적으로 관리할 수 있으며, 이를 다양한 애플리케이션에서 재사용하여 의미적 일관성을 유지할 수 있음 (RAG-외부 벡터 DB)
- 컴퓨팅 인프라
  - LLM을 훈련하고 실행하려면 광범위한 GPU/TPU 자원에 접근해야 함.
  - GPT-3를 훈련하는데 10,000개 이상의 GPU에서 3640 petaflop/s days.

## 생성형 AI 애플리케이션과 LLM

- **생성형 AI 애플리케이션 3계층**
  - 스택의 세 가지 주요 계층인 인프라 계층, **모델 계층**, 애플리케이션 계층
- **모델 계층**
  - 생성형 AI 애플리케이션 스택의 핵심으로, LLM의 선택과 조정이 중요함.
  - 생성형 AI 애플리케이션의 기능, 효율성 및 효과를 결정하는데 중요한 역할을 함.
  - **올바른 LLM 선택**
    - **모델 기능** : 모델의 고유 기능을 고려해야 함(Open/Close 등)
    - **계산 효율성** : 선택한 LLM을 훈련하고 배포하는데 필요한 계산 자원이 얼마나 되는지 판단해야 함.
    - **문제의 적합성** : 애플리케이션이 풀고자 하는 문제 및 도메인이 적합한지 판단해야 함. 텍스트 요약에 적합한 모델(T5 등)이 있는가 하면, 창의적 텍스 생성 모델(GPT)

## 생성형 AI 애플리케이션과 LLM

### ▪ 모델 계층

- **올바른 LLM 선택**
- **LLM 파인튜닝**
  - LLM을 선택한 후에는 애플리케이션의 요구 사항에 맞게 조정하는 것이 필수적임.
  - **전이 학습**을 기반으로 한 파인튜닝 기술은 이를 달성하는데 사용됨. 즉, 특정 도메인에 타겟 데이터셋을 사용함.
    - 1) **타겟 데이터 셋** : 파인튜닝은 일반적으로 LLM을 더 작은 특화된 작업을 위한 데이터 셋으로 훈련함. 이를 통해 모델은 일반적인 지능을 유지하면서도 특화된 작업에서 더 능숙해짐.
    - 2) **성능 향상** : 파인튜닝은 대화 시스템, 추론 또는 지식 검색과 같은 특정 영역에서 LLM의 성능을 높임.
    - 3) **치명적 망각** : 파인튜닝 중에 모델이 이전에 학습한 지식을 잊는 치명적 망각(catastrophic forgetting)에 주의가 필요함. 이를 완화하는데 **그래디언트 클리핑(gradient clipping)** 및 **특정 레이어의 선택적 파인튜닝**과 같은 기술이 사용됨.

- 전이 학습(transfer learning)이란 : 한 작업에서 학습한 지식 (모델, 파라미터 등)을 다른 관련 작업에 재사용하는 기법을 의미함.

## 생성형 AI 애플리케이션과 LLM

---

### ▪ 모델 계층

- 올바른 LLM 선택
  - LLM 파인튜닝
  - LLM 통합
    - 선택하고 파인튜닝된 LLM을 애플리케이션에 통합하는 단계임.
- 1) 데이터 변환 : 애플리케이션 코드는 텍스트나 다른 유형의 입력 데이터를 LLM에 적합한 토큰 임베딩으로 변환해야 함. 이 변환은 데이터를 효과적으로 처리하는데 필수적임.
- 2) 모델 아키텍쳐 : LLM의 아키텍쳐, 즉 LLM이 어떤 구조로 구성되어 있는지는 입력 데이터를 처리하는 방식에 중요한 역할을 함. 예를 들어, 입력 토큰에 대한 self-attention이 어떻게 적용되는지와 같은 요소는 모델이 데이터에서 넓은 범위의 의존성을 포착하는 능력을 결정함.
- 3) 확장 혁신 : 최근의 발전으로 LLM은 효율적으로 확장했음. 희소게이트 전문가 모델(SGMoE) 같은 기술은 모델 계층을 여러 전문가 그룹으로 분할해 확장성을 키움. (Sparsley Gated Mixture of Experts)

## 생성형 AI 애플리케이션과 LLM

---

### ▪ 생성형 AI 애플리케이션 3계층

- 스택의 세 가지 주요 계층인 인프라 계층, 모델 계층, 애플리케이션 계층

### ▪ 애플리케이션 계층

- 이 계층은 통합된 LLM을 활용해 생성형 AI 애플리케이션의 개발 및 배포를 간소화하는데 중점을 둠.
- 애플리케이션 개발 프레임워크
  - 생성형 AI 개발 프레임워크는 LangChain(Claude, GPT-3 등)과 같은 LLM API를 중심으로 구축되어 애플리케이션 구축을 간소화함.
  - 이러한 프레임워크는 세부적인 모델 배포에 신경 쓰지 않고, 추론에만 집중하도록 개발자 친화적인 API와 SDK를 제공함.
- LangChain
  - LLM을 기반으로 애플리케이션을 개발하는 라이브러리이자 플랫폼임. ([www.langchain.com](http://www.langchain.com))
  - 이는 여러 유용한 구성 요소를 연결해 최종 사용자에게 더 유용한 출력을 얻게 함.
  - 체인은 1)입력을 프롬프트 템플릿에 맞춰 생성하고, 2)이 생성된 입력을 LLM에 전달, 3)출력을 분석하고 처리해 화면에 표시하는 방식으로 세 단계로 구성함.

## 생성형 AI 애플리케이션과 LLM

### ▪ 생성형 AI 애플리케이션 3계층

- 스택의 세 가지 주요 계층인 인프라 계층, 모델 계층, 애플리케이션 계층

### ▪ 애플리케이션 계층

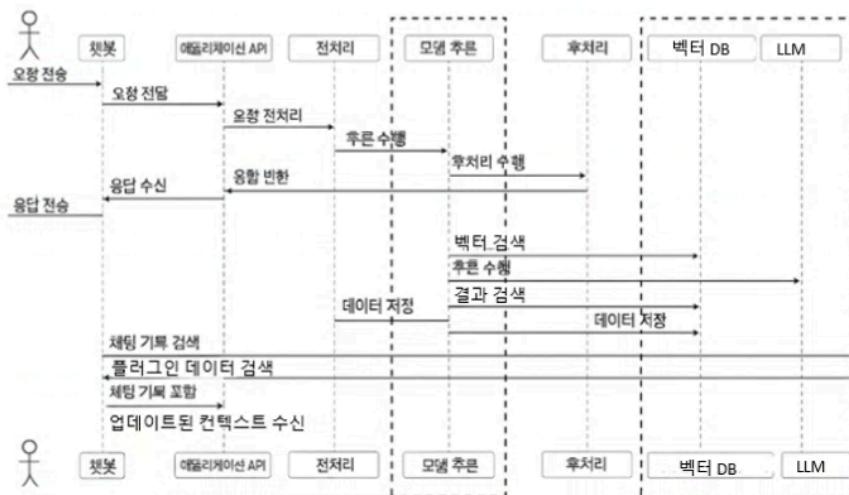
#### • LLM 기반 자율 에이전트

- 에이전트(Agent) : 외부 세계의 정보와 상호작용하는데 다양한 도구(tool)를 사용함
- 인기있는 도구로는 계산기, 파일 시스템, 워크백과 API등이 있음
- 자율 에이전트 (Autonomous Agent) : 인간의 개입없이 목표를 이해하고, 스스로 계획을 세우며, 외부 도구나 환경과 상호 작용하여 작업을 수행하는 시스템임. (AutoGPT, AgentGPT등)

#### • 애플리케이션 구축

- 개발자는 프레임워크의 인터페이스를 사용해 애플리케이션에 LLM 기능을 통합함.
- 비즈니스 로직은 사용자 입력을 받은 프레임워크의 API로 전송하고, 추론 결과를 처리함.
- 애플리케이션은 LLM 적용 외에도 신원 관리, 보안, 프라이버시, 개인화 및 모니터링과 같은 기능을 관리함.

## 생성형 AI 챗봇 애플리케이션의 요청 흐름 Sequence Diagram



## Flask, Spring 및 LangChain과의 Integration

### ▪ Flask의 Backend Server 역할

- AI 모델과 사용자 간의 interface를 담당함.

#### 1) RESTful API 서버

- Flask는 lightweight framework로서, AI 모델을 API 형태로 외부에 누출할 때 적합함.

#### 2) 모델 Serving Interface

- AI 모델(OpenAI, HuggingFace 모델등)을 Python에서 로딩하여 Interface를 수행하고, 이를 HTTP 요청으로 감싸 제공함.
- 요청 수신 및 Parsing/모델 호출/결과 포맷팅 및 응답 반환

#### 3) Frontend와의 연동

- React or HTML/JS 기반 프론트 엔드에서 사용자 입력을 보내면 Flask는 이를 받아 모델에 전달하고 결과를 반환함.

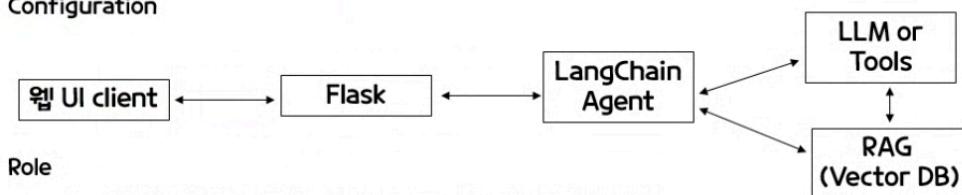
#### 4) 사용자 세션 및 간단한 상태 관리

- 대화형 AI (챗봇 등)에서는 간단한 세션 기반 상태 유지 기능을 Flask가 담당할 수 있음.

## Flask, Spring 및 LangChain과의 Integration

### ▪ Flask와 LangChain의 관계

- Flask는 웹 서버이고, LangChain은 생성형 AI 애플리케이션의 논리와 흐름(파이프라인)을 구성하는 라이브러리임.
- Configuration



#### • Role

- Flask : HTTP 요청 처리, 입력 파싱, LangChain 호출/ 응답 반환
- LangChain : LLM 기반 Workflow 관리 (Prompt Chain, Agent, Tools, Memory 등)
- LLM : 실제 응답을 생성 (HuggingFace, Anthropic, OpenAI 등)

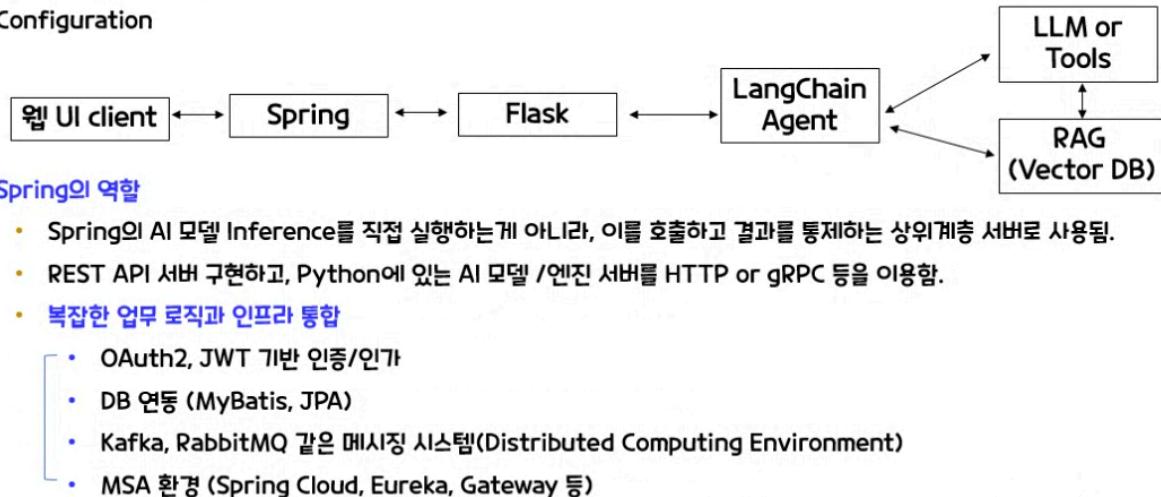
#### • Flask 한계

- 대규모 API 관리 어려움 (라우팅과 모듈분리, 타입 기반 개발 부족 등)
- 대형 프로젝트 구조화 어려움  
(MVC 구조에 대한 가이드 없음, 개발자마다 구조가 달라지면 일관성 부족, 팀 단위 개발에서 유지보수 어려움, 의존성 주입(DI)같은 고급 설계 어려움)
- 확장성과 미들웨어 통합의 어려움

슬라이드 노트의 내용을 입력하십시오

### ▪ Flask와 Spring과의 관계

- Spring은 웹 서버 및 API 서버, 서비스 계층, 보안 및 인증, DB연동, 마이크로 서비스 통신 등의 역할을 수행할 수 있음
- Configuration



#### • Spring의 역할

- Spring의 AI 모델 Inference를 직접 실행하는게 아니라, 이를 호출하고 결과를 통제하는 상위계층 서버로 사용됨.
- REST API 서버 구현하고, Python에 있는 AI 모델 /엔진 서버를 HTTP or gRPC 등을 이용함.
- 복잡한 업무 로직과 인프라 통합
  - OAuth2, JWT 기반 인증/인가
  - DB 연동 (MyBatis, JPA)
  - Kafka, RabbitMQ 같은 메시징 시스템(Distributed Computing Environment)
  - MSA 환경 (Spring Cloud, Eureka, Gateway 등)

# Ch01. 플라스크 개발 준비!

## 01-1 플라스크 개요

- 2010년 오스트리아의 오픈소스 개발자가 만든 **Python** 기반의 마이크로 웹 프레임워크임.
- 장고(Django)와 더불어 파이썬 웹 프레임워크의 양대 산맥으로 자리매김하고 있음.
- 장고는 많은 기능을 기본으로 내장하고 있지만,
- Flask는 필요한 기능만을 개발자가 직접 선택하고 추가할 수 있도록 설계됨
- Flask는 다음과 같은 장점을 갖고 있음.
  - 간결함과 유연성 : 최소한의 기능만 제공하며, 나머지 기능은 필요에 따라 확장할 수 있음.
  - 모듈화 : 필요한 기능을 모듈 형태로 추가할 수 있어, 프로젝트의 크기와 복잡성에 맞춰 쉽게 확장할 수 있음.
  - 자유로운 구조 : 개발자가 자신의 프로젝트에 맞는 구조를 자유롭게 설계할 수 있음.

## 01-1 플라스크 개요

### ▪ 확장성 있는 설계란?

- 플라스크에는 폼form, 데이터베이스database를 처리하는 기능이 없음.
- 플라스크는 확장 모듈이라는 것을 사용하여 이를 보완함.
- 그래서 플라스크 프로젝트는 가벼운 편임.

### ▪ 플라스크는 자유로운 프레임워크임

- 플라스크에는 최소한의 규칙만 있으므로 개발의 자유도는 다른 프레임워크보다 높다.

## 01-2 파이썬 설치

### 윈도우에서 파이썬 설치하기

#### ▪ 01단계: 최신 파이썬 설치 파일 내려받기

- 파이썬 공식 홈페이지에서 윈도우용 파이썬 설치 파일을 내려받자.  
(<https://www.python.org/downloads/>)

## 01-2 파이썬 설치

#### ▪ 03단계: 파이썬 설치 확인하기

- 파이썬이 제대로 설치됐는지 명령 프롬프트에서 아래와 같이 확인함.
- 파이썬 버전이 제대로 출력되지 않으면 삭제 후 다시 설치함.

```
명령 프롬프트
Microsoft Windows [Version 10.0.19045.6093]
(c) Microsoft Corporation. All rights reserved.

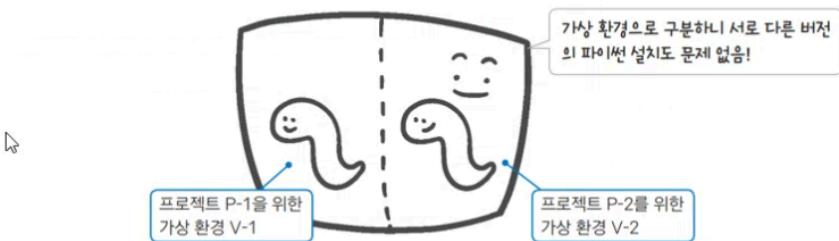
C:\Users\502-00>python -V
Python 3.11.9

C:\Users\502-00>
```

## 01-3 플라스크 개발 환경

### ▪ 파이썬 가상 환경 알아보기

- 파이썬 가상 환경은 파이썬 프로젝트를 진행할 때 독립된 환경을 만들어 주는 도구임.
- 하나의 디스크톱에 서로 다른 버전의 파이썬과 라이브러리를 쉽게 사용할 수 있음.



## 01-3 플라스크 개발 환경

### ▪ 파이썬 가상 환경 사용

- 파이썬 가상 환경에 플라스크를 설치하기 위해 먼저 내 컴퓨터에 파이썬 가상 환경부터 만든다.

### ▪ 01단계: 가상 환경 디렉터리 생성

- 명령 프롬프트에 다음 명령어를 입력해 C:\venvs라는 디렉터리를 만든다.

```
명령 프롬프트
C:\>mkdir Python_Project
C:\>cd Python_Project
C:\Python_Project>mkdir venvs
C:\Python_Project>cd venvs
C:\Python_Project\venvs>
```

## 01-3 플라스크 개발 환경

### ▪ 02단계: 가상 환경 만들기

- 다음의 명령어를 실행하면 파이썬 가상 환경이 만들어짐.
- C:\Python\_Project\venvs\myproject라는 디렉터리가 생성되었을 것임.

```
명령 프롬프트
C:\Python_Project\venvs>
C:\Python_Project\venvs>python -m venv myproject
C:\Python_Project\venvs>dir
C 드라이브의 볼륨에는 이름이 없습니다.
볼륨 일련 번호: C8C4-AFCF

C:\Python_Project\venvs 디렉터리

2025-07-18 오후 04:22 <DIR> .
2025-07-18 오후 04:22 <DIR> ..
2025-07-18 오후 04:22 <DIR> myproject
    0개 파일      0 바이트
    3개 디렉터리  154,486,919,168 바이트 남음

C:\Python_Project\venvs>
```

-m : python이 제공하는 특정 모듈을 스크립트처럼 실행하라는 옵션임. 여기서는 venv라는 모듈을 실행하라는 의미임.  
venv : python에서 제공하는 기본 가상환경을 만들기 위한 모듈  
myproject : 가상환경을 생성할 디렉터리 이름임.

## 01-3 플라스크 개발 환경

### ▪ 03단계: 가상 환경에 진입

- 가상 환경에 진입하려면,  
우리가 생성한 myproject 가상 환경에 있는 Scripts 디렉터리의 'activate'라는 명령 수행.
- 아래와 같은 명령으로 myproject 가상 환경에 진입함. (activate)

```
명령 프롬프트
C:\Python_Project\venvs\myproject\Scripts>
C:\Python_Project\venvs\myproject\Scripts>activate
```



```
명령 프롬프트
(myproject) C:\Python_Project\venvs\myproject\Scripts>
```

## 01-3 플라스크 개발 환경

### ▪ 04단계: 가상 환경에서 벗어나기

- 'deactivate'라는 명령 실행.
- 가상 환경에서 잘 벗어났다면 C:\W 왼쪽에 있던 (myproject)라는 프롬프트가 사라짐.

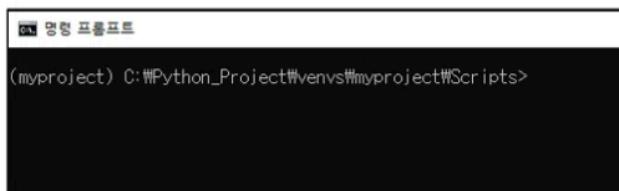
```
명령 프롬프트
(myproject) C:\Python_Project\venvs\myproject\Scripts>deactivate
C:\Python_Project\venvs\myproject\Scripts>
```

## 01-3 플라스크 개발 환경

### 플라스크 설치하기

#### ■ 01단계: 가상 환경인지 확인하기

- 명령 프롬프트 왼쪽에 (myproject) 프롬프트가 보이는지 확인.

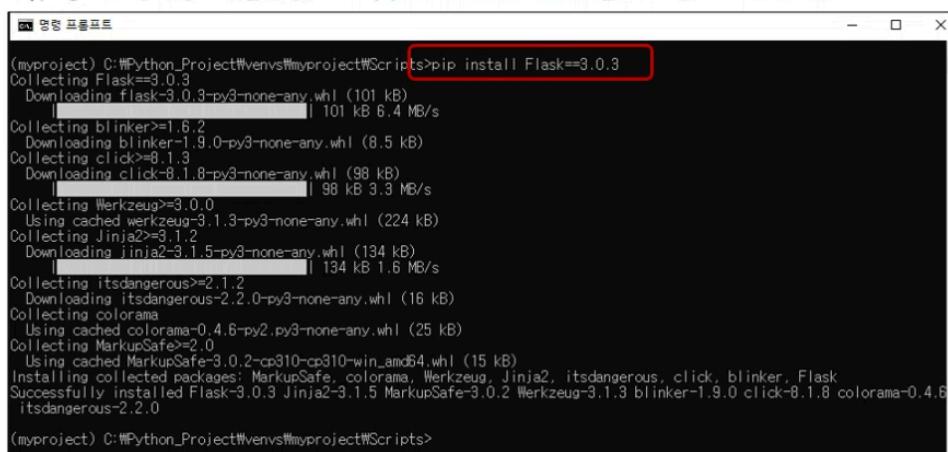


```
(myproject) C:\Python_Project\venvs\myproject\Scripts>
```

## 01-3 플라스크 개발 환경

### ■ 02단계: 가상 환경에서 플라스크 설치하기

- myproject 가상 환경에 진입한 상태에서 'pip install Flask==3.0.3' 입력하여 플라스크를 설치함.



```
(myproject) C:\Python_Project\venvs\myproject\Scripts>pip install Flask==3.0.3
Collecting Flask==3.0.3
  Downloading flask-3.0.3-py3-none-any.whl (101 kB) | 101 kB 6.4 MB/s
Collecting blinker>=1.6.2
  Downloading blinker-1.9.0-py3-none-any.whl (8.5 kB)
Collecting click>=8.1.3
  Downloading click-8.1.8-py3-none-any.whl (98 kB) | 98 kB 3.3 MB/s
Collecting Werkzeug>=3.0.0
  Using cached werkzeug-3.1.3-py3-none-any.whl (224 kB)
Collecting Jinja2>=3.1.2
  Downloading jinja2-3.1.5-py3-none-any.whl (134 kB) | 134 kB 1.6 MB/s
Collecting itsdangerous>=2.1.2
  Downloading itsdangerous-2.2.0-py3-none-any.whl (16 kB)
Collecting colorama
  Using cached colorama-0.4.6-py2.py3-none-any.whl (25 kB)
Collecting MarkupSafe>=2.0
  Using cached MarkupSafe-3.0.2-cp310-cp310-win_amd64.whl (15 kB)
Installing collected packages: MarkupSafe, colorama, Werkzeug, Jinja2, itsdangerous, click, blinker, Flask
Successfully installed Flask-3.0.3 Jinja2-3.1.5 MarkupSafe-3.0.2 Werkzeug-3.1.3 blinker-1.9.0 click-8.1.8 colorama-0.4.6 itsdangerous-2.2.0
(myproject) C:\Python_Project\venvs\myproject\Scripts>
```

## 01-3 플라스크 개발 환경

### ▪ 03단계: pip 최신 버전으로 설치하기

- 다음 명령을 입력해 pip를 최신 버전으로 설치.

```
[notice] A new release of pip is available: 24.0 -> 25.1.1
[notice] To update, run: python.exe -m pip install --upgrade pip
(myproject) C:\Python_Project\venvs\myproject\Scripts>python -m pip install --upgrade pip
Requirement already satisfied: pip in c:\python_project\venvs\myproject\lib\site-packages (24.0)
Collecting pip
  Using cached pip-25.1.1-py3-none-any.whl.metadata (3.6 kB)
Using cached pip-25.1.1-py3-none-any.whl (1.8 MB)
Installing collected packages: pip
  Attempting uninstall: pip
    Found existing installation: pip 24.0
    Uninstalling pip-24.0:
      Successfully uninstalled pip-24.0
      Successfully installed pip-25.1.1
(myproject) C:\Python_Project\venvs\myproject\Scripts>
```

## 01-4 플라스크 프로젝트 생성하기

### ▪ 플라스크 프로젝트를 생성하면 웹 사이트를 한 개 생성하는 것과 같음.

- 플라스크 프로젝트 안에는 보통 한 개의 플라스크 애플리케이션이 존재함.

### 프로젝트 디렉터리 생성하기

#### ▪ 01단계: 프로젝트 루트 디렉터리 생성하기

- 플라스크 프로젝트를 모아 둘 projects 디렉터리를 생성.

```
C:\> 명령 프롬프트
C:\>
C:\>
C:\>mkdir Flask_projects
C:\>cd Flask_projects
C:\Flask_projects>
```

## 01-4 플라스크 프로젝트 생성하기

### ■ 02단계: 프로젝트 루트 디렉토리 안에서 가상 환경에 진입하기

- 프로젝트 루트 디렉토리 안에서 다음 명령어를 입력해 앞에서 만든 myproject 가상 환경에 진입.
- 반드시 c:\Flask\_projects에서 명령어를 입력해야 함.

```
명령 프롬프트
C:\Flask_projects>
C:\Flask_projects>
C:\Flask_projects>c:\Python_Project\venvs\myproject\Scripts\activate
```

```
명령 프롬프트
(myproject) C:\Flask_projects>
```

## 01-4 플라스크 프로젝트 생성하기

### ■ 03단계: 플라스크 프로젝트를 담을 디렉터리 생성하고 이동하기

- 플라스크 프로젝트를 담을 flask\_basic 디렉터리를 생성하고 이동.

```
명령 프롬프트
(myproject) C:\Flask_projects>
(myproject) C:\Flask_projects>mkdir flask_basic
(myproject) C:\Flask_projects>cd flask_basic
(myproject) C:\Flask_projects\flask_basic>
```

## 01-4 플라스크 프로젝트 생성하기

### 배치 파일로 myproject 가상 환경에 간단히 진입하기

#### 01단계: 배치 파일 생성하기

- venvs 디렉터리에 myproject.cmd 파일을 만들고 코드를 작성한 후 저장함.

```
myproject.cmd
1 @echo off
2 cd C:\Flask_projects\flask_basic
3 C:\Python_Project\venvs\myproject\Scripts\activate
```

#### 02단계: 배치 파일 위치를 PATH 환경 변수에 추가하기

- 이 배치 파일이 명령 프롬프트 어느 곳에서나 수행될 수 있도록
- 'Window + R key' 입력하여, 입력란에 'sysdm.cpl'를 명령을 입력하면 '시스템 속성'창이 뜬다.
- C:\Python\_Project\venvs 디렉터리를 시스템의 환경 변수 PATH에 추가해야 함.

## 01-4 플라스크 프로젝트 생성하기

### 03단계: PATH 환경 변수 확인하기

- 명령 프롬프트를 다시 시작.
- set path 명령을 실행하여 변경된 환경 변수 PATH의 내용을 확인함.

```
명령 프롬프트
(c) Microsoft Corporation. All rights reserved.

C:\Users\502-00>set path
Path=C:\Program Files\Common Files\Oracle\Java\javapath;C:\oraclexe\app\oracle\product\11.2.0\server\bin;;C:\Program Files\Python38\Scripts\;C:\Program Files\Python38\;C:\Python313\Scripts\;C:\Python313\;C:\Windows\system32;C:\Windows;C:\Windows\System32\Wbem;C:\Windows\System32\WindowsPowerShell\v1.0\;C:\Windows\System32\OpenSSH\;C:\ProgramData\chocolatey\bin;C:\Users\502-00\AppData\Local\Programs\Python\Python311\Scripts\;C:\Users\502-00\AppData\Local\Programs\Python\Python311\;C:\Users\502-00\scoop\shims;C:\Users\502-00\AppData\Local\Microsoft\WindowsApps;%PyCharm Community Edition%;C:\Users\502-00\AppData\Local\Microsoft\WinGet\Packages\Schniz.fnn_Microsoft_Winget_Source_8wekyb3d8bbwe;C:\Users\502-00\AppData\Local\Programs\Microsoft VS Code\bin;C:\Python_Project\venvs;
PATHEXT=.COM;.EXE;.BAT;.CMD;.VBS;.VBE;.JS;.JSE;.WSF;.WSH;.MSC

C:\Users\502-00>
```

## 01-4 플라스크 프로젝트 생성하기

- 04단계: 배치 파일 실행하여 가상 환경에 진입하기
  - myproject 명령(배치 파일명)을 실행하여 가상 환경에 잘 진입하는지 확인.

```
▶ 명령 프롬프트
Microsoft Windows [Version 10.0.19045.5371]
(c) Microsoft Corporation. All rights reserved.

C:\Users\user>myproject
```

```
▶ 명령 프롬프트
(myproject) C:\Flask_projects\flask_basic>
```