AI활용프로그래밍

Week 2. 파이썬 환경 구축과 Hello World

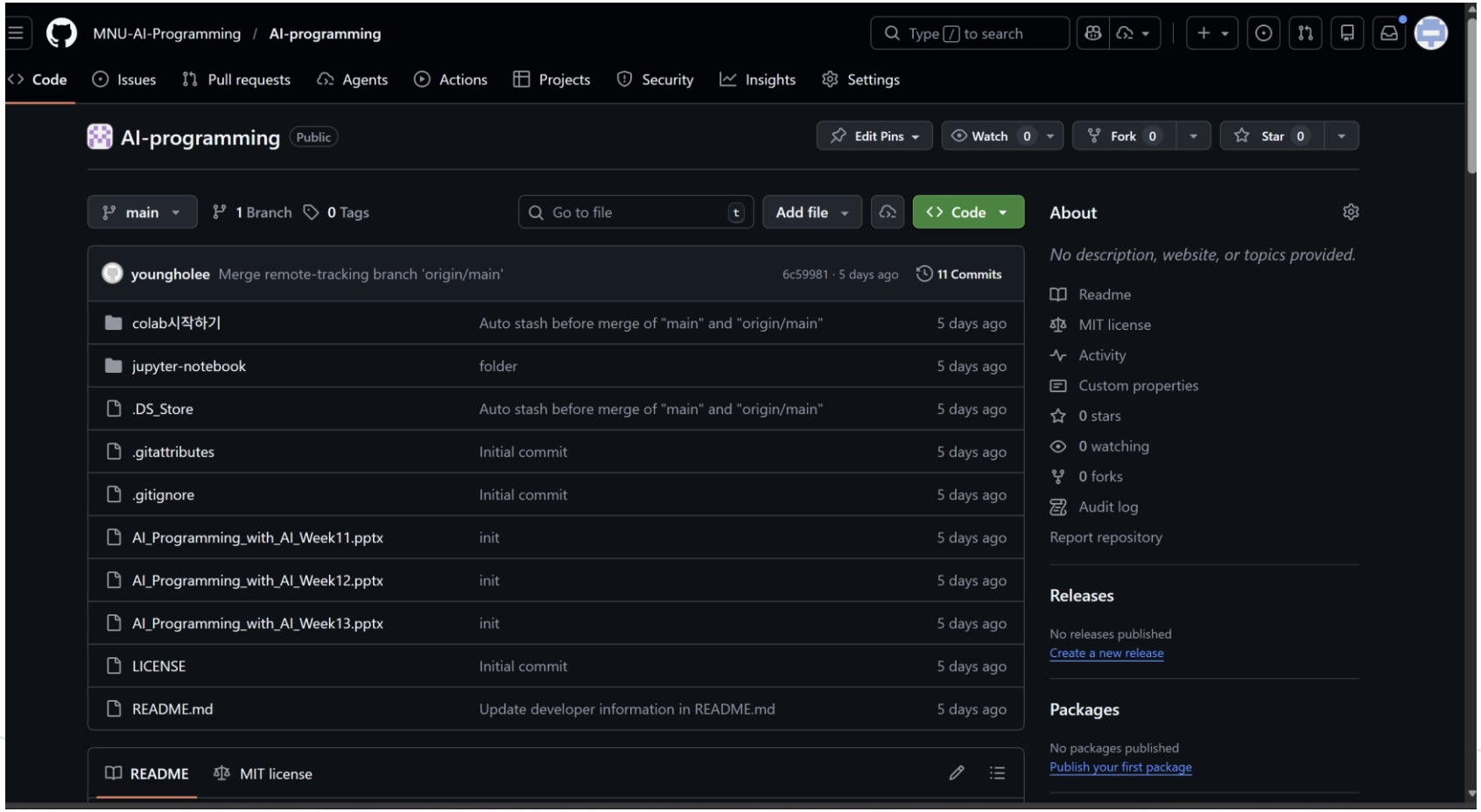
설치·설정·실행까지 한 번에 (With AI 활용)

학습 목표

- 파이썬 개발환경(인터프리터·에디터·터미널)을 구성할 수 있다.
- 가상환경(venv)과 패키지 관리자(pip)의 목적을 설명할 수 있다.
- Hello World 프로그램을 작성·실행하고, LLM으로 설명/개선을 요청할 수 있다.

Github 저장소

- 저장소: https://github.com/MNU-AI-Programming/AI-programming



오늘의 구성

- 개발환경 구성 요소 이해
- Python 설치 & 설치 확인
- VS Code 기본 설정
- 가상환경(venv) & pip 기본
- Hello World 작성/실행 + LLM 활용 실습

개발환경 구성 요소

오늘 목표: "설치→실행→확인→정리"까지 스스로 할 수 있게 만들기

Python 설치(개요)

- Windows: Python 설치 시 “Add Python to PATH” 옵션 체크 권장
- macOS: 공식 설치 또는 패키지 매니저(예: Homebrew) 사용 가능
- Linux: 배포판 패키지 매니저(apt/yum 등) 또는 공식 설치
- 수업에서는 Python 3.x 환경을 사용(버전은 학기 중 안내 기준 따름)

설치가 막히면: AI에게 에러 메시지/스크린샷을 포함해 구체적으로 자세하게 질문하세요.

잠깐! 터미널(Terminal)이란?

- 터미널은 컴퓨터에게 명령을 "글자"로 입력하는 창.
- 우리가 하는 일:어디에서 실행할지(폴더) 정하고 `python 파일이름.py` 처럼 실행 명령을 준다.
- 왜 쓰나?
  - 실행/설치/확인(버전 확인 등)을 가장 확실하게 할 수 있음
  - 에러 메시지가 그대로 남아서 디버깅/질문이 쉬움

- 용어 2개만 기억
  - 명령어(command): 터미널에 치는 문장
  - 현재 폴더(working directory): "지금 내가 작업하는 위치"

윈도우에서는
파워쉘이 많이
이용되는 터미널!

설치 확인(버전·실행)

```
# 터미널(명령 프롬프트)에서 확인
python --version
pip --version

# 파이썬 대화형 실행(REPL)
python
>>> print("Hello")
>>> exit()
```

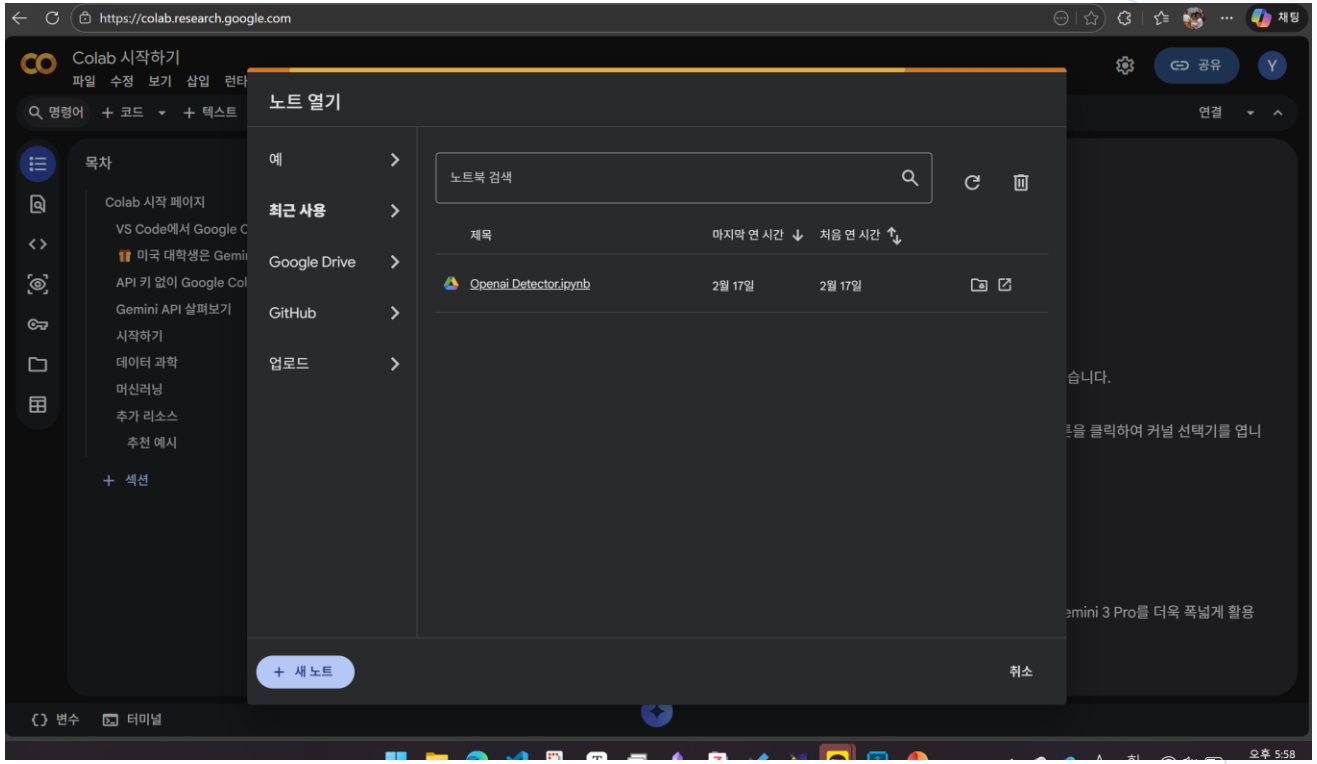
핵심 포인트

- python/pip 명령이 인식되지 않으면 PATH 설정 문제일 수 있음
- REPL은 "짧게 실험"하기에 매우 유용
- 환경 확인 결과(버전)는 과제 제출 시 증빙으로 활용

코드 에디터: 구글 Colab 사용하기

① 접속하기
- 웹 브라우저 열기
- 주소창에 입력
- https://colab.research.google.com
- 구글 계정으로 로그인

② 새 노트북 만들기
- 왼쪽 상단 **파일(File)** 클릭
- **새 노트북(New notebook)** 선택
- 자동으로 .ipynb 파일 생성됨



코드 에디터: VS Code 세팅

- VS Code 설치 후 Python 확장(Extension) 설치
- 인터프리터 선택: 가상환경(venv) 또는 설치된 Python 선택
- 터미널을 VS Code 내부에서 열어 실행 연습
- 저장(Save) → 실행(Run) 흐름을 습관화

VS Code(Visual Studio Code)는 마이크로소프트가 만든 무료 코드 편집기(에디터)
현업에서 웹/앱/서버/데이터 등 거의 모든 개발 작업에 많이 활용됨

프로젝트 폴더 구조(권장)

- 주차별 폴더를 만든다: week02/
- 파일 이름은 의미 있게: hello.py, greet.py, calc.py
- 실험용/제출용 분리도 좋다: /practice, /submit
- README.md에 실행 방법 3줄만 적어도 큰 도움이 된다

터미널 기본 명령어(최소)

```
# 폴더 이동
cd week02

# 폴더 목록 보기
ls   # macOS/Linux
dir  # Windows

# 새 폴더 만들기
mkdir hello_world
```

핵심 포인트

- 폴더 위치(working directory)를 의식하면 오류가 줄어듦
- "어디에서 실행했는지"가 결과에 영향을 주는 경우가 많음

가상환경(venv) 왜 쓰나?

- 프로젝트마다 필요한 패키지/버전이 다를 수 있음
- 가상환경은 “프로젝트 전용” 파이썬 환경(의존성 격리)
- 다른 과제/프로젝트와 충돌을 줄여줌
- 협업 시: requirements.txt로 환경을 재현하기 쉬움

venv 생성/활성화(Windows 예시)

```
# 프로젝트 폴더에서
python -m venv .venv

# 활성화(명령 프롬프트)
.venv\Scripts\activate

# 비활성화
deactivate
```

핵심 포인트

- 활성화되면 터미널 앞에 (.venv) 같은 표시가 보일 수 있음
- VS Code에서 인터프리터가 .venv를 가리키는지 확인

venv 생성/활성화(macOS/Linux 예시)

```
# 프로젝트 폴더에서
python3 -m venv .venv

# 활성화
source .venv/bin/activate

# 비활성화
deactivate
```

핵심 포인트

- 시스템에 따라 python 대신 python3를 쓰는 경우가 있음
- 권한 문제/명령어 미인식 시: 에러 메시지를 그대로 기록해두기

pip 기본 사용

```
# 패키지 설치
pip install numpy

# 설치 목록
pip list

# 현재 환경을 파일로 저장(재현용)
pip freeze > requirements.txt
```

핵심 포인트

- pip는 "파이썬 패키지 설치 도구"
- 가상환경을 켠 상태에서 설치하면 프로젝트에만 적용됨
- requirements.txt는 협업/재설치에 매우 중요

Hello World: 첫 스크립트

```python
# hello.py
print("Hello, World!")
```

핵심 포인트

- 파일 저장 후 실행: python hello.py
- print()는 화면(표준 출력)에 글자를 보여준다
- 가장 작은 성공 경험을 만든 뒤 확장한다

실행 방법 3가지

- 터미널에서 실행: python hello.py
- VS Code 실행 버튼(Run) 사용
- REPL에서 한 줄씩 실험: python → print("...")
- 추천: “REPL로 짧게 실험 → 파일로 정리 → 터미널로 실행”

print() 더 알아보기

```python
name = "Kim"
print("Hello", name)
print(f"Hello, {name}!")  # f-string
print("A", "B", "C", sep="-")  # sep
print("끝!", end="")  # end
```

핵심 포인트

- 출력 형식은 디버깅과 사용자 경험에 큰 영향
- f-string은 문자열에 값을 넣을 때 자주 사용

input()으로 사용자 입력 받기

```
1 name = input("이름을 입력하세요: ")
2 print(f"안녕하세요, {name}님!")
```

핵심 포인트

- input()은 "문자열"로 입력을 받음
- 숫자가 필요하면 다음 주에 int()/float() 변환을 배움
- 오늘은 흐름(입력→처리→출력)에 익숙해지는 것이 목표

LLM 활용 1: 설치/환경 오류 해결 프롬프트

```
역할: 너는 개발환경 트러블슈팅 전문가야.
목표: 아래 오류를 해결하는 “가능성 높은 원인 3개”와 “해결 절차”를 알려줘.
제약: (1) 단계별로, (2) 내가 확인해야 할 명령어도 같이 제시, (3) 위험한 설정 변경은 경고해줘.

내 환경:
- OS: (Windows/macOS/Linux)
- 실행한 명령어:
- 나온 오류 메시지:

오류 메시지:
```
(여기에 그대로 붙여넣기)
```
```

핵심 포인트

- 에러 메시지는 “요약하지 말고 그대로” 전달
- OS/명령어/출력 3가지를 함께 주면 정확도가 올라감

LLM 활용 2: 코드 설명/주석 생성 프롬프트

```
역할: 너는 파이썬 튜터야.
목표: 아래 코드를 “줄별로” 설명하고, 초보가 이해하기 쉬운 주석을 추가해줘.
제약: 주석은 과하지 않게(핵심만), 출력 결과는 바꾸지 말 것.

```python
# 여기에 내 코드를 붙여넣기
```
```

핵심 포인트

- 설명 + 주석 + 개선안을 한 번에 받으면 학습 효율이 좋아짐
- “출력 바꾸지 말 것” 같은 제약을 명확히 주기

LLM 활용 3: 작은 기능 추가 프롬프트

````
목표: 아래 Hello 프로그램에 기능을 추가해줘.
추가 기능:
1) 이름이 비어 있으면 다시 입력받기
2) 인사 문구를 3번 반복 출력하기

조건: 초보 수준에서 이해 가능하게, 코드에 주석 포함, 실행 예시(입력/출력)도 보여줘.

```python
name = input("이름: ")
print(f"안녕하세요, {name}님!")
```
````

핵심 포인트

- "기능 목록"을 번호로 적으면 요구사항이 명확해짐
- 실행 예시를 요구하면 결과를 빨리 검증할 수 있음

실습 체크리스트(오늘 완료 기준)

- python --version / pip --version 확인
- VS Code에서 파이썬 파일 생성·저장
- hello.py 실행 성공(터미널 또는 Run 버튼)
- 가상환경(venv) 생성 및 활성화(가능하면)
- LLM에게 코드 설명 요청 1회 + 내가 이해한 요약 3줄

실습: Hello World 변형(3개 중 2개)

- A) 이름을 입력받아 인사하기 (input + f-string)
- B) 좋아하는 음식 1개를 입력받아 출력하기
- C) 두 개의 문장을 출력 형식 바꿔보기 (sep/end 활용)
- 도전: LLM에게 “내 코드의 개선점 3가지”를 받아 반영해보기

미니 퀴즈(개념 확인)

문항

1) 가상환경(venv)을 쓰는 가장 큰 이유는?
A. 속도 향상 B. 의존성 격리 C. 인터넷 연결

2) input()의 반환 타입은 기본적으로?
A. int B. float C. str

3) requirements.txt는 언제 유용한가?
A. 환경 재현/협업 B. 그림 그리기 C. 실행 속도 측정

4) LLM에게 설치 오류를 물을 때 가장 도움이 되는 정보는?
A. 느낌 B. 에러 메시지 원문 C. 별명

정답: 1-B, 2-C, 3-A, 4-B

과제 제출 안내(예시)

- 제출물: week02 폴더(hello.py 포함) + 실행 화면(스크린샷) 1장
- 스크린샷에는: 터미널에서 python --version, hello.py 실행 결과가 보이게
- LLM 사용 내역: 사용한 프롬프트 1개 + AI 답변 요약 + 내가 반영한 수정 1개
- 제출 형식은 수업 공지 기준을 따른다

중요: "그냥 되게 하기"가 아니라 "왜 되는지 설명하기"까지!

요약 & 다음 주 예고

- 개발환경은 Python + 터미널 + 에디터 + pip + (가능하면) venv로 구성된다.
- Hello World를 실행하는 것이 “환경 구축 성공”의 최소 기준이다.
- LLM은 설치/오류/설명에 도움되지만, 에러 메시지와 맥락을 정확히 주자.
- 다음 주: 변수·자료형·연산자 + LLM 연습문제