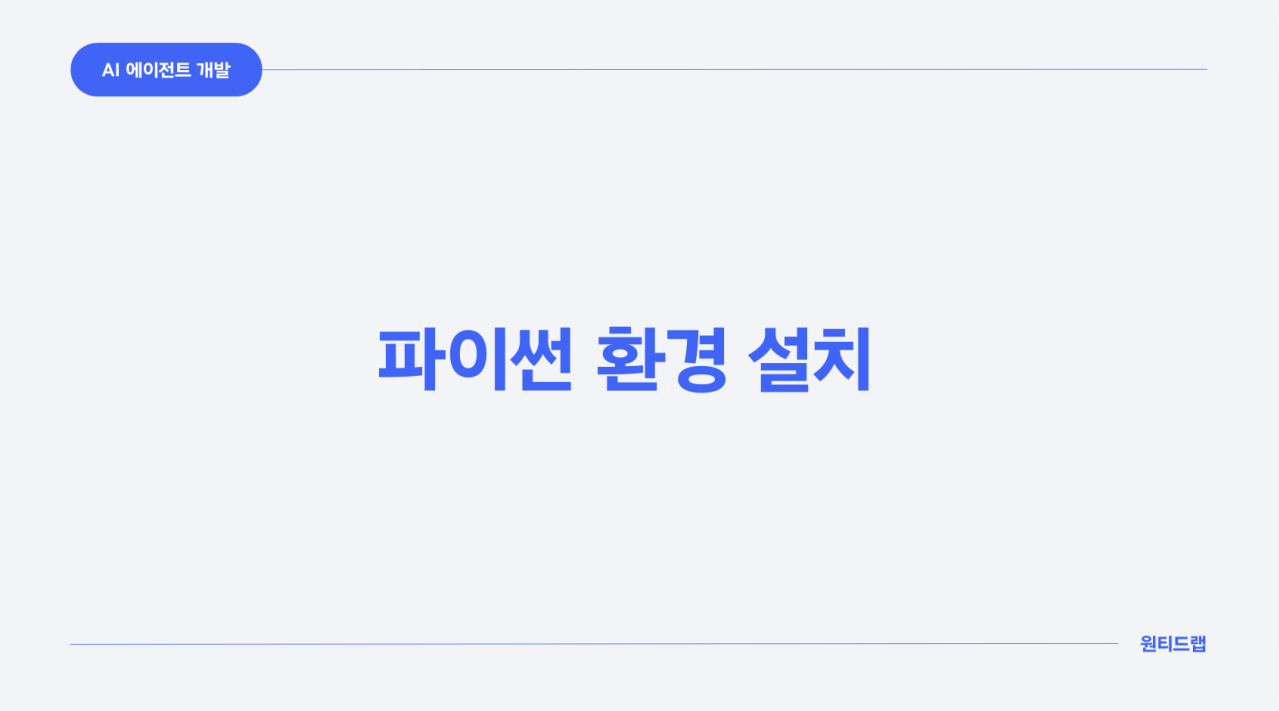
00 파이썬 환경 설치(ver2.0)



파이썬

- 오픈소스로 무료이며, 어디서든 다운받아 사용이 가능하다.
- 직관적으로 이해가 가능하다
- 다른 언어에 비해 간결하다
- 개발 속도가 빠르고 생산성이 높다
- 높은 확장성으로 자동화, 인공지능 등 다양한 분야에 활용이 가능하다

```
tomato = 3000
banana = 2500

price = tomato + banana

print(price)
```

```
input_id = "hello1234"

if input_id == "hi1234":
    print("로그인 성공")
else:
    print("ID가 일치하지 않습니다.")
```

Visual Studio Code 설치

🔗 참고자료: Visual Studio Code 공식 사이트

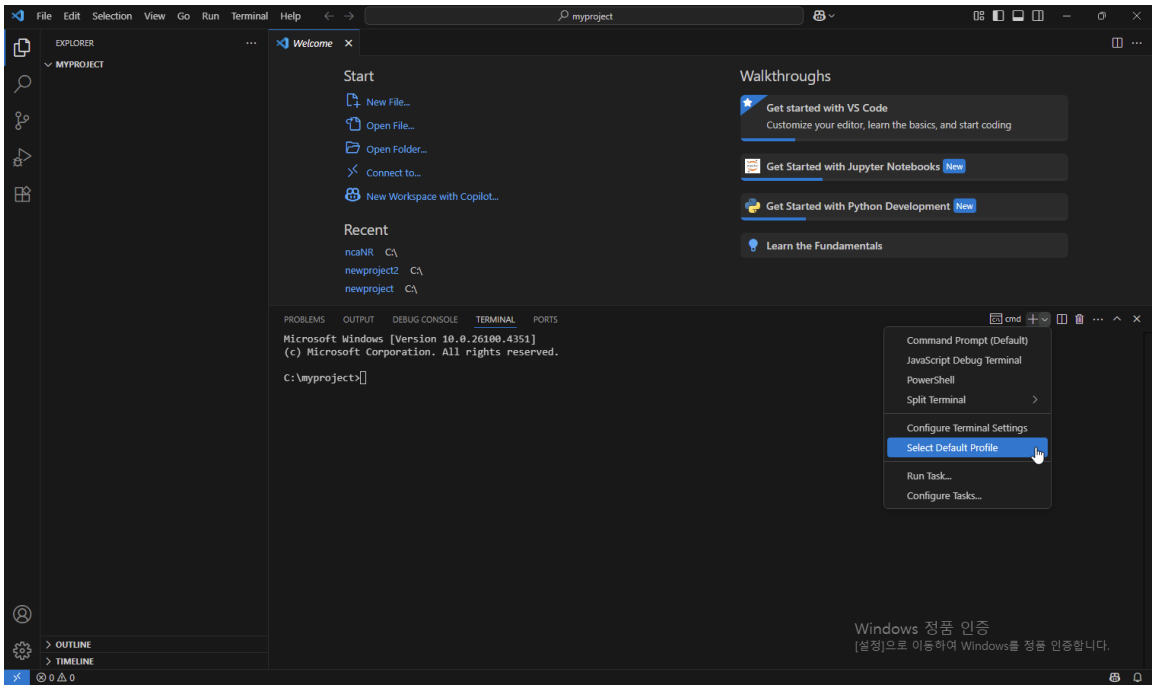
파이썬 Extension 추가

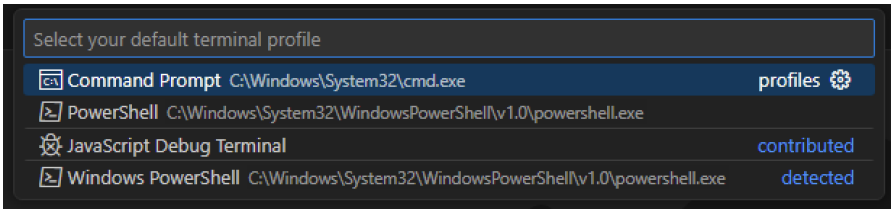
Visual Studio Code에서 Python을 사용하기 위해서는 Extension에서 Python을 설치해야 한다.
왼쪽 아이콘 중 4개의 박스로 표현된 아이콘이 Extension을 설치할 수 있는 버튼이다.

터미널 설정

오른쪽 위 4개의 레이아웃 아이콘 중 3번째 아이콘을 누르면 아래에 터미널 창이 나타난다.
파이썬 프로젝트를 관리할 때에는 되도록 Command Prompt 창을 이용하는 것을 추천한다.

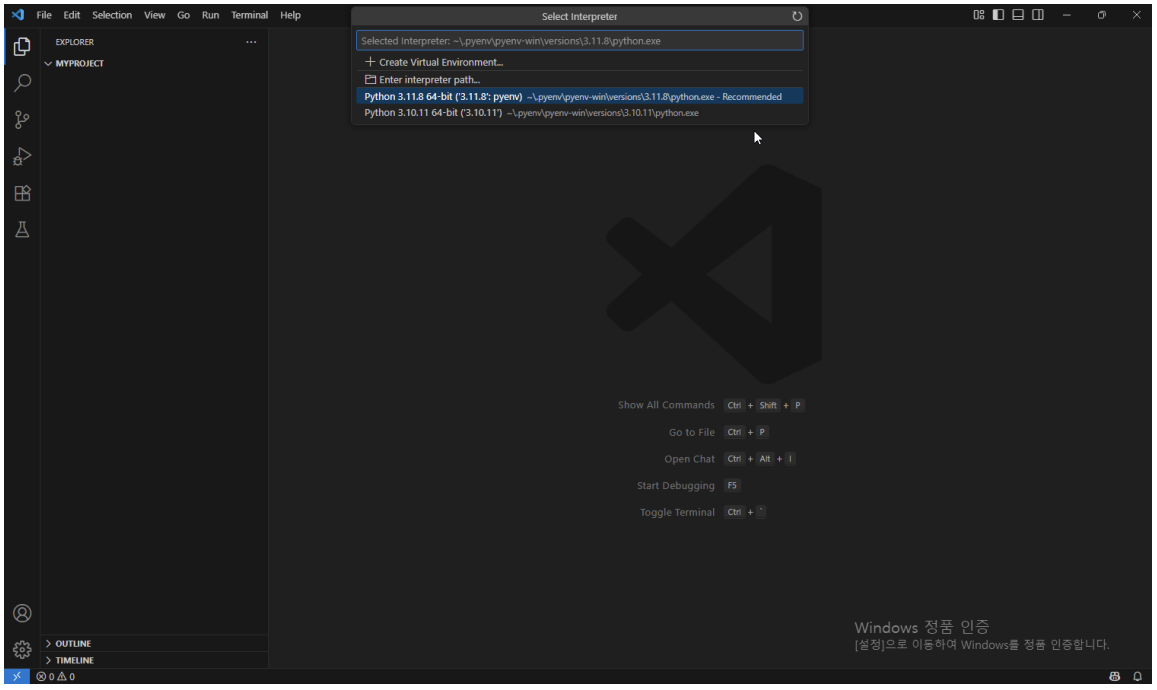
Visual Studio Code가 실행될 때마다 Command Prompt가 연결되게 하기 위해서는 터미널창 오른쪽에 "+"옆에 있는 아래쪽 화살표를 클릭하고, [Select Default Profile]을 클릭한다.





파이썬 연결

`Ctrl + Shift + P` 에서 [Python: Select Interpreter]-[Enter interpreter path...] 를 클릭한 후 가상환경을 설정해두면 Visual Studio Code가 실행될 때 자동으로 그 가상환경에 연결될 수 있게 할 수 있다.



UV 설치

Python으로 프로젝트를 하게 되면
가상환경을 만들고, 라이브러리를 설치하고, 환경을 공유하는 과정이 필수적이다.

그동안 프로젝트 관리는 다음과 같이 변화했다.

```
pip/venv → conda → poetry → uv
- pip/venv: 너무 수동적이고 설치 속도도 느림
- conda: 환경이 무거움
- poetry: 종속성 관리의 편리함이 있지만 속도 느림
```

uv는 기존 도구들의 단점을 거의 다 보완하면서도,

- 엄청 빠른 설치 속도
- 심플한 명령어
- 가상환경 생성부터 의존성 관리, 실행까지 한 번에 처리 가능한

현대적인 Python 도구이다.

1. 설치

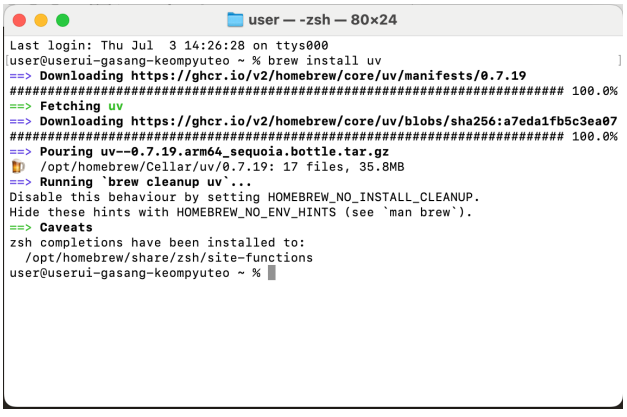
🔗 참고자료: uv Documentation

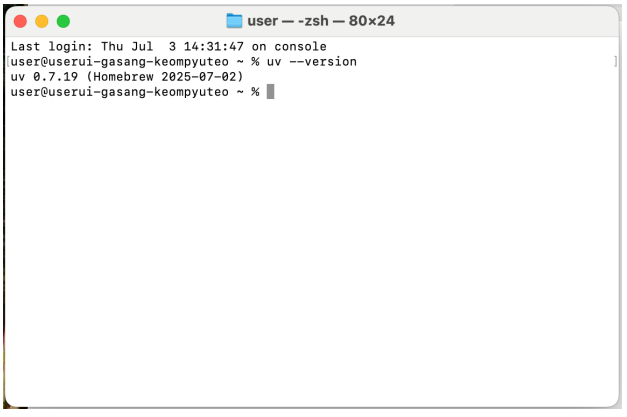
uv 설치 확인

```
uv --version
```

MacOS

```
brew install uv
```

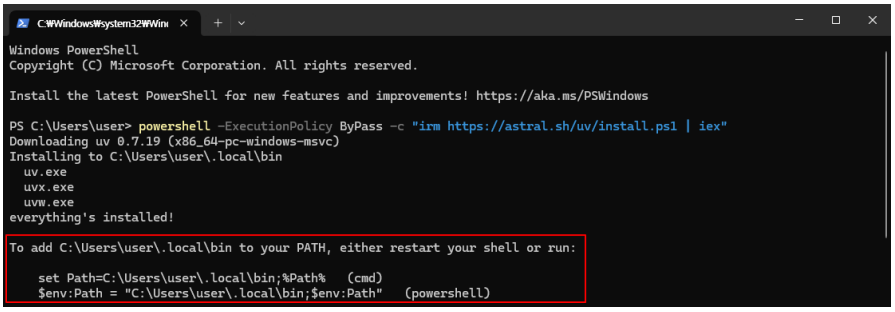




Windows

Powershell에서 아래의 명령어를 실행한다.

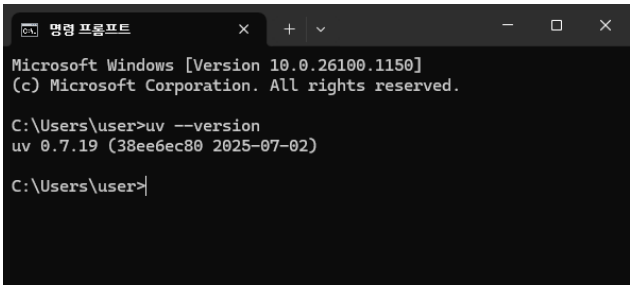
```
powershell -ExecutionPolicy ByPass -c "irm https://astral.sh/uv/install.ps1 | iex"
```



설치가 완료되면 환경변수에 경로를 추가해주어야 한다.
위의 그림 아래쪽에 보면 PowerShell 버전의 환경 변수 추가 명령어를 안내해준다. 그대로 복사해서 붙여넣기 한 후 실행하면 된다.

```
$env:Path="C:\Users\user\.local\bin;$env:Path"
```

새로운 명령 프롬프트 창을 열어 `uv`의 버전을 확인했을 때 버전이 명시되면 성공이다.



2. 명령어

> ⓘ 예시 이미지는 Windows 환경에서 Visual Studio Code 도구 내 터미널을 캡처한 것이다.

`myproject` 폴더를 생성한 후, Visual Studio Code에서 `myproject` 폴더를 열어보자.

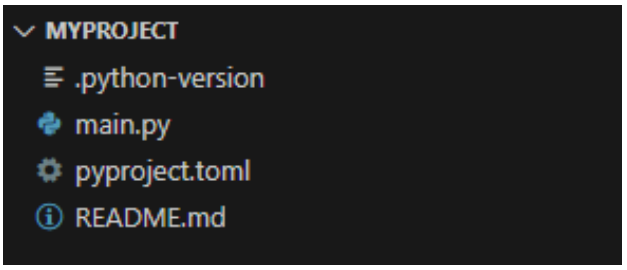
새 프로젝트 생성

```
uv init
```

프로젝트를 생성하면 `myproject` 폴더에 여러 개의 파일이 생성된다.

- `.python-version` : 현재 폴더에서 사용할 Python 버전이 명시되어 있는 파일이다.
- `main.py` : 기본 실행 파일이다
- `pyproject.toml` : 이 프로젝트에서 어떤 라이브러리를 사용할지, 어떤 Python 버전을 필요로 하는지 등을 정의하는 설정 파일이다. 프로젝트의 이름, 버전, 종속성 정보 등이 담겨 있으며 uv는 이 파일을 중심으로 작동한다.
- `README.md` : 프로젝트 소개 파일이다. 이 파일에는 “이 프로젝트는 무엇을 하는지”에 대한 설명을 Markdown 형식으로 작성할 수 있다. GitHub에 올리면 프로젝트 설명으로 자동 표시된다.
- `.gitignore` : 로컬 컴퓨터에 git이 설치되어 있는 경우 나타난다. GitHub와 같이 원격저장소에 올리지 않을 파일들을 관리하는 파일이다.

```
C:\myproject>uv init
Initialized project `myproject`
```



특정 Python version을 설정하고 싶은 경우

방법 1. `.python-version` 파일과 `pyproject.toml` 의 `requires-python` 키에 특정 버전을 직접 수정한다.
방법 2. 가상환경을 파이썬 특정 버전에 맞춰 생성한 후, `uv init` 을 한다.

```
uv venv -p 3.10
uv init
```

가상환경 생성

```
uv venv [가상환경이름]
```

가상환경 이름을 `myenv` 로 설정하면, 작업 디렉토리에 `myenv` 폴더가 생성된 것을 볼 수 있다.

```
C:\myproject>uv venv myenv
Using CPython 3.11.8 interpreter at: C:\Users\user\.pyenv\pyenv-win\versions\3.11.8\python.exe
Creating virtual environment at: myenv
Activate with: myenv\Scripts\activate
```

가상환경은 `.venv` 로 관리하는 것이 좋다.

- 프로젝트 관리도구는 기본적으로 `.venv` 를 가상환경으로 자동 감지하기 때문에 편리하다.
- `.venv` 는 숨김 폴더이기 때문에 실수로 열어보거나 건드는 일이 적다.
- 가상환경이름을 다르게 설정하면 배포 시 `.gitignore` 에 추가하는 데 실수할 가능성이 높다.

☆ 가상환경 폴더는 uv가 자동으로 관리하기 때문에 폴더 안에 파일을 추가하거나 수정하면 안된다.

가상환경 활성화

```
[가상환경폴더명]\Scripts\activate      # Windows 버전
source [가상환경폴더명]/bin/activate.  # MacOS 버전
```

가상환경을 활성화하면 터미널 창 안에 `(가상환경이름)` 이 항상 맨 앞에 출력된다.

```
C:\myproject>myenv\Scripts\activate

(myenv) C:\myproject>
```

☆ 가상환경 폴더를 설정하지 않으면 가상환경 이름은 폴더 이름이지만, 작업 디렉토리의 폴더 이름은 `.venv` 이다.

```
C:\myproject>uv venv
Using CPython 3.11.8 interpreter at: C:\Users\user\.pyenv\pyenv-win\versions\3.11.8\python.exe
Creating virtual environment at: .venv
Activate with: .venv\Scripts\activate

C:\myproject>.venv\Scripts\activate

(myproject) C:\myproject>
```

라이브러리 설치 및 삭제

```
uv add 라이브러리1 라이브러리2 ...      # 라이브러리 설치
uv remove 라이브러리1 라이브러리2 ...   # 라이브러리 삭제
```

라이브러리를 설치하면 `uv.lock` 파일이 함께 생성된다.
이 파일은 **설치된 라이브러리들의 정확한 버전과 의존성 정보를 기록해 두는 잠금 파일**이다.

라이브러리를 추가하거나 삭제할 때마다 `uv.lock` 과 `pyproject.toml` 파일이 함께 업데이트되며,
가상환경 폴더를 복사해오지 않아도 **누구나 동일한 환경을 재현할 수 있게 된다.**

```
(myproject) C:\myproject>uv add pandas
Resolved 7 packages in 728ms
Prepared 6 packages in 10.11s
Installed 6 packages in 692ms
 + numpy==2.3.1
 + pandas==2.3.0
 + python-dateutil==2.9.0.post0
 + pytz==2025.2
 + six==1.17.0
 + tzdata==2025.2

(myproject) C:\myproject>uv remove pandas
Resolved 1 package in 13ms
Uninstalled 6 packages in 580ms
 - numpy==2.3.1
 - pandas==2.3.0
 - python-dateutil==2.9.0.post0
 - pytz==2025.2
 - six==1.17.0
 - tzdata==2025.2
```

`uv.lock` 과 `pyproject.toml` 의 차이

pyproject.toml은 **어떤 라이브러리를 쓸지** 정리한 설정 파일이고, uv.lock은 **그 라이브러리들이 실제로 어떤 버전으로 설치되었는지** 기록한 파일이다.
특히 pyproject.toml에는 "pandas>=1.5"처럼 **버전 범위**만 적혀 있기 때문에, 다른 환경에서는 **다른 버전이 설치될 수 있다**
그래서 팀 프로젝트나 배포할 때는 uv.lock까지 함께 공유해야 **100% 동일한 환경을 재현**할 수 있다.

3. 시나리오별 환경 관리

내 프로젝트 관리

```
📁 myproject/
├── 📁 project1/
│   ├── .venv/
│   ├── 📁 subproject/
│   │   ├── .venv/
│   │   ├── pyproject.toml
│   │   └── main.py
│   ├── pyproject.toml
│   └── main.py
├── 📁 project2/
│   ├── .venv/
│   ├── pyproject.toml
```

```
│   └── main.py
└── 📁project3/
    ├── .venv/
    ├── pyproject.toml
    └── main.py
```

`.venv` 경로 기준으로 그 하위에 또 다른 `.venv` 가 존재하면 같은 Workspace로 인식하여 상위에 있는 `.venv` 에 통합 관리된다.

`subproject` 가상환경을 활성화한 후 라이브러리를 설치했다해도 `project1`의 가상환경에 설치된다.
이를 해결하기 위해서는

1. 설치할 때마다 `--active` 옵션을 추가하거나
2. `pyproject.toml`의 하단에 `[tool.uv.workspace]` 부분을 삭제한다.

Visual Studio Code에서 `myproject`폴더를 열게 되면 내가 project1, project2, project3을 왔다갔다 할 때마다
터미널에서 경로를 바꿔주어야 한다.

```
C:\myproject>project1\.venv\Scripts\activate   # project1 가상환경 활성화
C:\myproject>project2\.venv\Scripts\activate   # project2 가상환경 활성화
C:\myproject>project3\.venv\Scripts\activate   # project3 가상환경 활성화
```

이렇게 관리하게 되면 내가 설치하려고 한 라이브러리를 다른 가상환경에 설치하는 실수를 할 가능성이 높다.
따라서 되도록 Visual Studio Code에는 하나의 가상환경 `.venv` 이 담긴 폴더를 여는 것이 좋다.

개발을 다른 컴퓨터에서 이어서 진행할 때

PC1에서 PC2로 프로젝트를 옮겨야 한다고 하자.
먼저 PC1의 `myproject` 폴더 안의 파일들 중 **가상환경 폴더를 제외하고** PC2의 `newproject` 폴더로 옮긴다.
그 이후 `newproject` 폴더에서 **1) 새로운 가상환경을 만들고 활성화**한 뒤 **2) 라이브러리를 설치**하면 된다.

```
uv venv                    # 가상환경 생성
.venv\Scripts\activate     # 가상환경 활성화
uv sync
```

원래 컴퓨터와의 환경을 정확하게 구현하기 위해서는 실제 설치된 정확한 버전으로 라이브러리를 설치해야 한다.
`uv sync` 명령어를 사용하면 `uv.lock`에 있는 정확한 버전으로 설치할 수 있다.

다른 사람이 내가 만든 프로젝트를 사용할 때

내가 만든 프로젝트 GitHub에 오픈소스로 공개되어 있다고 하자.
다른 사람이 내가 만든 프로젝트를 다운받아 그 사람의 프로젝트에 이용하려고 한다.
다른 사람의 가상환경에는 이미 많은 라이브러리들이 설치되어 있는 상태이다.

```
uv venv                    # 가상환경 생성
.venv\Scripts\activate     # 가상환경 활성화
uv pip install .
```

설치가 끝나면 `build/` 폴더와 `myproject.egg-info/` 폴더가 생성된다.

- `build/` : 설치 과정 중에 내 프로젝트를 패키지(.whl)로 만들기 위한 임시 작업 파일들이 생성되는 공간이다. 설치가 끝난 후에는 삭제해도 된다.

- `myproject.egg-info/` : 내 프로젝트에 대한 메타데이터(프로젝트 이름, 버전, 의존성 정보 등)가 담긴 폴더다. 이 폴더는 패키지를 설치할 때 자동으로 생성되며, 설치된 라이브러리와의 연결을 위해 사용된다. 삭제해도 다시 생성되지만, 일반적으로는 그대로 두는 것이 좋다.

```
C:\newproject>uv venv
Using CPython 3.11.8 interpreter at: C:\Users\user\.pyenv\pyenv-win\versions\3.11.8\python.exe
Creating virtual environment at: .venv
Activate with: .venv\Scripts\activate

C:\newproject>.venv\Scripts\activate

(newproject) C:\newproject>uv pip install .
Resolved 7 packages in 113ms
      Built myproject @ file:///C:/newproject
Prepared 1 package in 1.92s
Installed 7 packages in 1.47s
 + myproject==0.1.0 (from file:///C:/newproject)
 + numpy==2.3.1
 + pandas==2.3.0
 + python-dateutil==2.9.0.post0
 + pytz==2025.2
 + six==1.17.0
 + tzdata==2025.2
```

4. 이슈 해결

× uv 프로젝트 도중 폴더 이름을 수정했는데 라이브러리를 설치하려고 하면 가상환경이 새로 생겨요. >

```
C:\newproject2>myenv\Scripts\activate

(myenv) C:\newproject2>uv add python-dotenv
warning: `VIRTUAL_ENV=C:\newproject\myenv` does not match the project environment path `.venv` and will be ignored; use `--active` to target the a
ctive environment instead
Using CPython 3.11.8 interpreter at: C:\Users\user\.pyenv\pyenv-win\versions\3.11.8\python.exe
Creating virtual environment at: .venv
Resolved 8 packages in 139ms
Prepared 1 package in 78ms
Installed 7 packages in 723ms
 + numpy==2.3.1
 + pandas==2.3.0
 + python-dateutil==2.9.0.post0
 + python-dotenv==1.1.1
 + pytz==2025.2
 + six==1.17.0
 + tzdata==2025.2
```

상황 설명

`newproject` 의 폴더에서 프로젝트를 진행 중이었는데 중간에 폴더 이름을 `newproject2` 로 변경했다. 그런데 가상환경에 새로운 라이브러리를 설치하려고 하면 `.venv` 가상환경이 새로 생성되고, 그 안에 라이브러리가 설치되는 문제가 발생했다.

발생 원인

가상환경 내부에는 생성 당시의 프로젝트 경로가 하드코딩되어 있다. 폴더 이름이 바뀌면 해당 경로가 달라지기 때문에, uv는 기존 가상환경폴더를 무시하고 새로운 가상환경을 다시 생성하게 된다.

해결 방법

기존의 가상환경폴더를 삭제하고 다시 가상환경을 생성한다.

```
uv venv myenv
myenv\Scripts\activate
uv sync
```

```
C:\newproject2>uv venv myenv
Using CPython 3.11.8 interpreter at: C:\Users\user\.pyenv\pyenv-win\versions\3.11.8\python.exe
Creating virtual environment at: myenv
Activate with: myenv\Scripts\activate

C:\newproject2>myenv\Scripts\activate

(myenv) C:\newproject2>uv pip install .
Using Python 3.11.8 environment at: myenv
Resolved 37 packages in 187ms
      Built myproject @ file:///C:/newproject2
Prepared 1 package in 1.87s
Installed 37 packages in 1.77s
 + annotated-types==0.7.0
 + anyio==4.9.0
 + certifi==2025.6.15
 + charset-normalizer==3.4.2
 + greenlet==3.2.3
 + h11==0.16.0
```

⭐ 이때 원래의 가상환경폴더가 `.venv` 라면 warning이 나올 뿐 자동으로 인식하기 때문에 정상적으로 설치가 된다. 하지만, 이 warning을 없애기 위해서는 `.venv` 폴더 내부를 수정해야 하기 때문에 가상환경을 삭제하고 다시 생성하는 것을 추천한다.

× uv 프로젝트 안에 uv 프로젝트를 만들고 새로운 가상환경에 라이브러리를 설치하는데 가상환경이 새로 생겨요. >

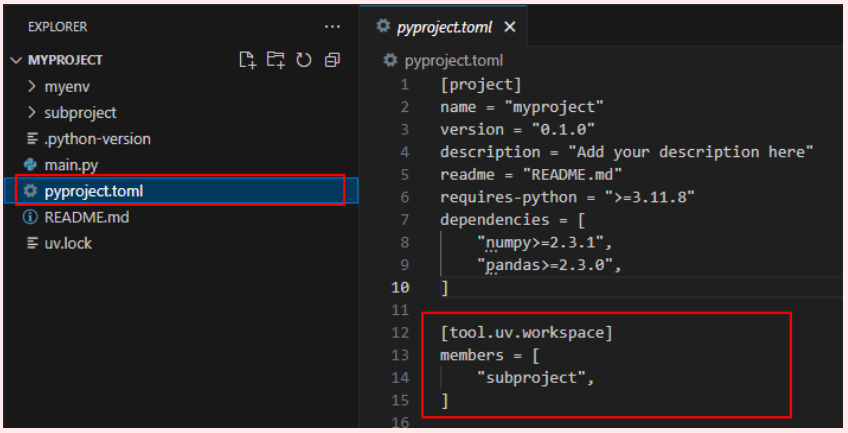
```
C:\myproject\subproject>uv init
Project `subproject` is already a member of workspace `C:\myproject`
Initialized project `subproject`

C:\myproject\subproject>uv venv subenv
Using CPython 3.11.8 interpreter at: C:\Users\user\.pyenv\pyenv-win\versions\3.11.8\python3.11.exe
Creating virtual environment at: subenv
Activate with: subenv\Scripts\activate

C:\myproject\subproject>subenv\Scripts\activate

(subenv) C:\myproject\subproject>uv add python-dotenv
warning: `VIRTUAL_ENV=subenv` does not match the project environment path `C:\myproject\.venv` and will be ignored; use `--active` to target the a
ctive environment instead
Using CPython 3.11.8 interpreter at: C:\Users\user\.pyenv\pyenv-win\versions\3.11.8\python3.11.exe
Creating virtual environment at: C:\myproject\.venv
Resolved 9 packages in 42ms
Installed 1 package in 89ms
 + python-dotenv==1.1.1
```

상황 설명

`newproject` 폴더에서 프로젝트를 진행 중이었는데 `newproject/subproject`를 생성한 후, 그 폴더에서 새로운 프로젝트를 생성하고, 새로운 가상환경인 `subenv`를 생성했다. 그리고 활성화한 뒤 새로운 라이브러리를 설치했다. 그런데 `newproject` 폴더에 `.venv`인 새로운 가상환경이 생기면서 그 환경에 라이브러리가 설치되는 문제가 발생했다.

발생 원인

uv는 상위 폴더에 이미 pyproject.toml이 존재하면, 하위 폴더에서 `uv init`을 해도 이를 workspace 멤버로 자동 인식한다.



다시 말하면 `subproject` 폴더는 독립된 프로젝트가 아닌 `myproject`의 일부로 간주되어, `subenv`를 활성화 했더라도 `myproject`에서 관리된다. 이 과정에서 `.venv`를 가상환경으로 자동인식하게 되는데 `myproject`의 가상환경이 `.venv`가 아닐 경우 가상환경이 없다고 판단하기 때문에 새로 생성하는 것이다.

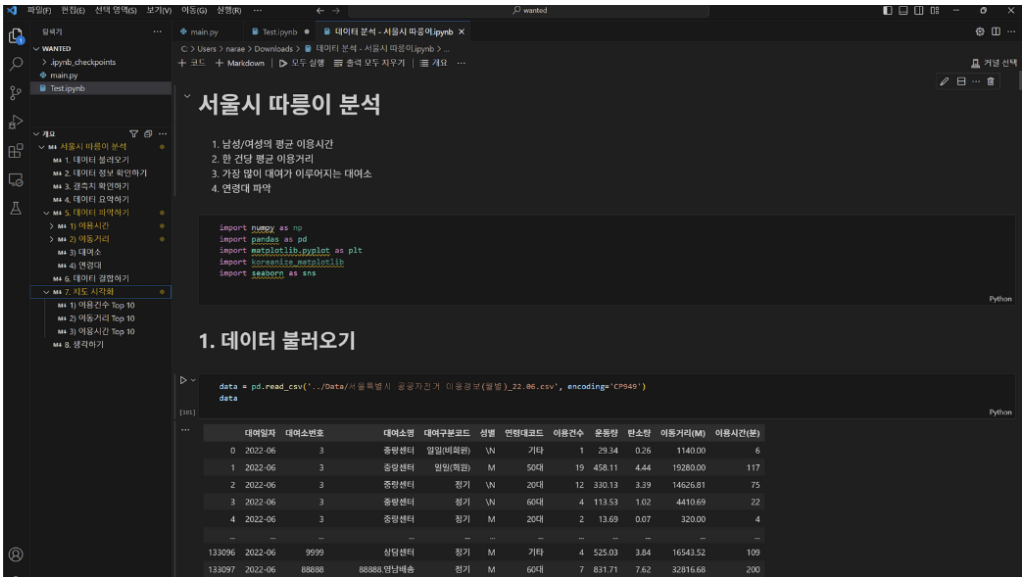
해결 방법

`myproject/pyproject.toml` 파일에서 workspace 내용을 삭제하거나, `subproject`를 `myproject` 폴더 바깥으로 이동시켜 관리해야 하는데 되도록이면 후자를 추천한다.

Jupyter Notebook 설치

- `.ipynb` 확장자를 가진 파일
- 코드, 문서, 시각화 등을 하나의 문서에 묶어 작성할 수 있다.
- 데이터 분석이나 머신 러닝 등의 작업을 할 때 편리하다.
- Visual Studio Code에서 Jupyter Notebook 사용을 위해서는 Extension 설치가 필요하다.
- Jupyter Notebook 사용을 위해서는 `jupyter` 라이브러리 설치가 필요하다.

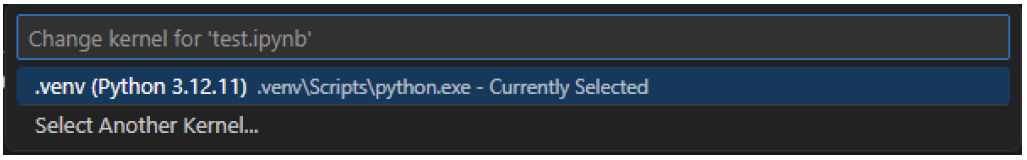
```
uv add jupyter
```



파이썬 연결

오른쪽 [커널 선택]을 눌러 원하는 파이썬 환경을 선택한다.

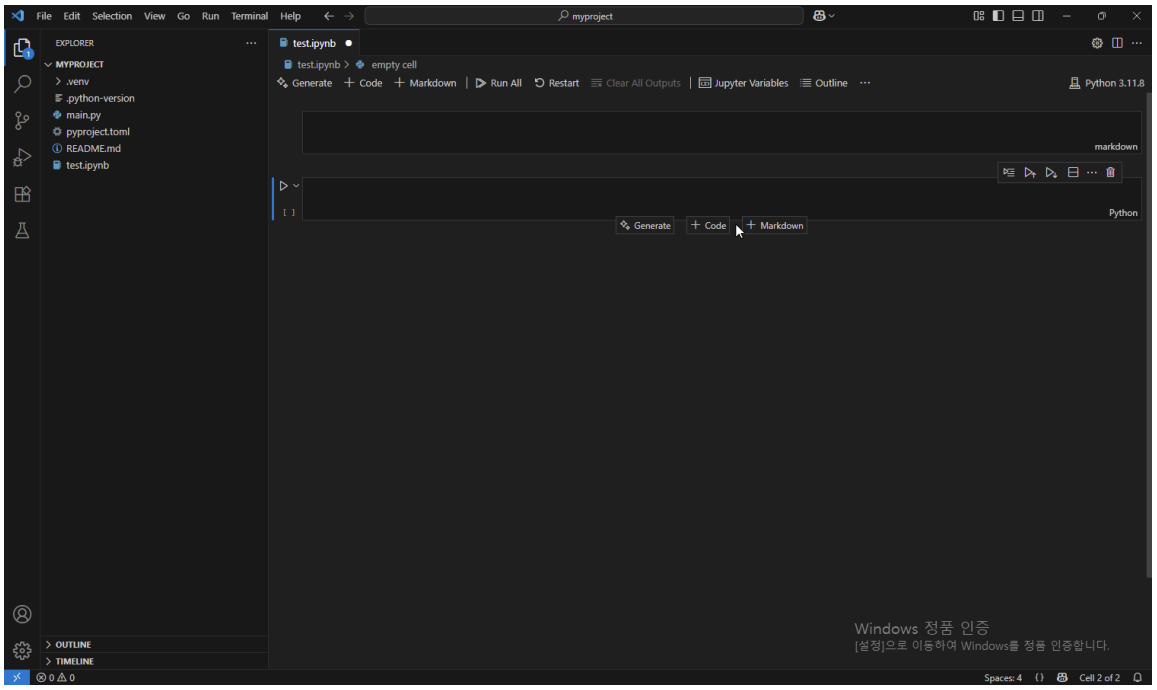
만약에 원하는 파이썬 환경이 없다면 [Select Another Kernel]-[Python Environments...]에서 찾을 수 있다.



그래도 없다면 `Ctrl + Shift + P` 에서 [Python: Select Interpreter]-[Enter interpreter path...] 클릭 후 내 가상환경 경로를 수동으로 입력하면 된다.

사용법

- 문서를 작성할 수 있는 Markdown 박스와 코드를 작성하고 실행할 수 있는 Code 박스로 이루어짐
- 박스 주변에 마우스를 가까이 하면 편리하게 박스를 삽입, 추가가 가능
- 코드 실행 : Ctrl + Enter


File Edit Selection View Go Run Terminal Help
myproject
EXPLORER
MYPROJECT
.venv
.python-version
main.py
pyproject.toml
README.md
test.ipynb
test.ipynb
test.ipynb > empty cell
Generate Code Markdown Run All Restart Clear All Outputs Jupyter Variables Outline
Python 3.11.8
markdown
Python
Generate Code Markdown
OUTLINE
TIMELINE
Windows 정품 인증
[설정]으로 이동하여 Windows를 정품 인증합니다.
Spaces: 4
Cell 2 of 2