

2025-1 한국어 정보 처리

Python 가상 환경

전태희

taeheejeon22@gmail.com

목차

1. 가상 환경 소개
2. 가상 환경 구축 도구
3. virtualenv 사용법
4. 의존성 관리
5. GitHub 저장소를 이용한 연습
6. 마무리

1. 가상 환경 소개

가상 환경이란?

▪ 가상 환경이란?

- 파이썬 라이브러리와 종속성을 프로젝트별로 독립적으로 관리하는 환경
- 시스템 전체에 영향을 주지 않고 격리된 개발 공간 제공

▪ 구성 요소

- Python 인터프리터
 - Python 3.8, Python 3.12, ...
- 프로젝트에 필요한 라이브러리 및 각종 파일

가상 환경이 왜 필요한가?

▪ 라이브러리 충돌 방지

- 하나의 라이브러리가 다른 라이브러리에 의존하는 경우는 상황이 더 복잡함

- pandas의 예

- https://pandas.pydata.org/docs/getting_started/install.html

▪ 동일 라이브러리를 프로젝트마다 다른 버전으로 사용 가능

- 가상 환경마다 동일 라이브러리를 독립적으로 설치 가능함

▪ 협업, 배포 등에 유리

- 개발 환경을 쉽게 재현할 수 있기 때문에 사용자별 PC, OS 등 차이에 따른 문제 최소화

몇 가지 특징들

- 일회성

- 언제든 쉽게 삭제하고 다시 만들 수 있음

- 이동/복사는 권장되지 않음

- 새로운 위치(컴퓨터 혹은 경로)에서 다시 생성하는 것이 원칙

- 프로젝트 코드와는 독립적으로 존재함

- 필요 시 하나의 프로젝트에 대해 여러 개의 가상 환경을 구축할 수도 있음

- Git 등 버전 관리 시스템에서 관리하지 않음

가상 환경 수동 관리 능력의 중요성

- 전역(Global) 환경만을 이용해 라이브러리 설치를 계속하다 보면 버전 충돌 일어날 수 있음
- Colab 등 편하게 쓸 수 있는 기성 환경들의 문제
 - 굳이 라이브러리를 설치하지 않아도 이미 웬만한 것들이 설치돼 있고 바로 사용 가능
 - 그러나 어떤 코드를 Colab 등에서 실행했을 때
import가 되더라도 동작하지 않는 경우 있을 수 있음
 - 기성 환경만을 이용하는 경우는
내 프로젝트에 적합한 라이브러리 버전을 맞추는 데에 제약이 있음

2. 가상 환경 구축 도구

venv

- Python의 표준 내장 도구
- Python 3.3 이상이면 기본적으로 설치되어 있음
- 간단한 프로젝트에 적합
- 공식 문서
 - <https://docs.python.org/3/library/venv.html>

venv 기본 사용 방법

1. 터미널을 이용해 가상 환경 파일들을 설치할 경로로 이동

2. 가상 환경 생성

\$ python -m venv *가상_환경_이름*

○ macOS/Linux는 python 대신 python3

○ 원하는 python 버전이 있으면 특정 가능

○ 예: python3.9 -m venv myenv

3. 가상 환경 활성화

[Windows] \$ *가상_환경_경로*\Script\activate

[macOS/Linux] \$ source *가상_환경_경로*/bin/activate

4. 가상 환경 비활성화

\$ deactivate

virtualenv

- 가상 환경 구축용 서드 파티 라이브러리
- venv와 달리, Python 2, 3 모두 지원
- venv보다 더 많은 기능 제공
- 기본 사용 방법은 venv와 유사함

어느 것을 쓸 것인가

- 이 밖에도 conda, pyenv, poetry 등의 도구들 존재함
- 정답은 없으며 취향껏 사용하면 됨
- 본 수업에서는 상대적으로 가벼우면서도 기능이 풍부하며
별다른 버그 없이 쓸 수 있는 `virtualenv`를 사용하도록 함

3. virtualenv 사용법

virtualenv 설치

- virtualenv는 서드 파티 라이브러리이므로
Python 설치 시에 설치되지 않으며 별도의 설치 필요
\$ pip install virtualenv
- 가상 환경이 아닌, 전역 환경에 설치해야 함
 - 가상 환경은 전역 환경을 거쳐 만드는 것

가상 환경을 설치할 경로로 이동

- 기본적으로 아무 경로에 설치해도 사용하는 데에 문제는 없음
- Git으로 관리하는 프로젝트 폴더에 설치하는 경우, 버전 관리 대상에서 배제됨
 - 가상 환경은 프로젝트와 별개로, 언제든지 손쉽게 삭제하고 다시 설치할 수 있는 일회성 환경
 - 즉 버전 관리의 대상이 아님
 - 가상 환경 내의 파일 개수가 매우 많기 때문에 이로 인해 Git의 처리 속도가 느려질 수 있음
- 가상 환경들만 저장할 폴더를 하나 만들어 두고
한 폴더에서 가상 환경들을 관리하는 것도 하나의 방법
 - 예: C:\virtualenv 내에 모든 가상 환경 저장

가상 환경 생성

▪ 사용법

○ Windows

- \$ python -m virtualenv 가상_환경_이름

○ macOS/Linux

- \$ virtualenv 가상_환경_이름

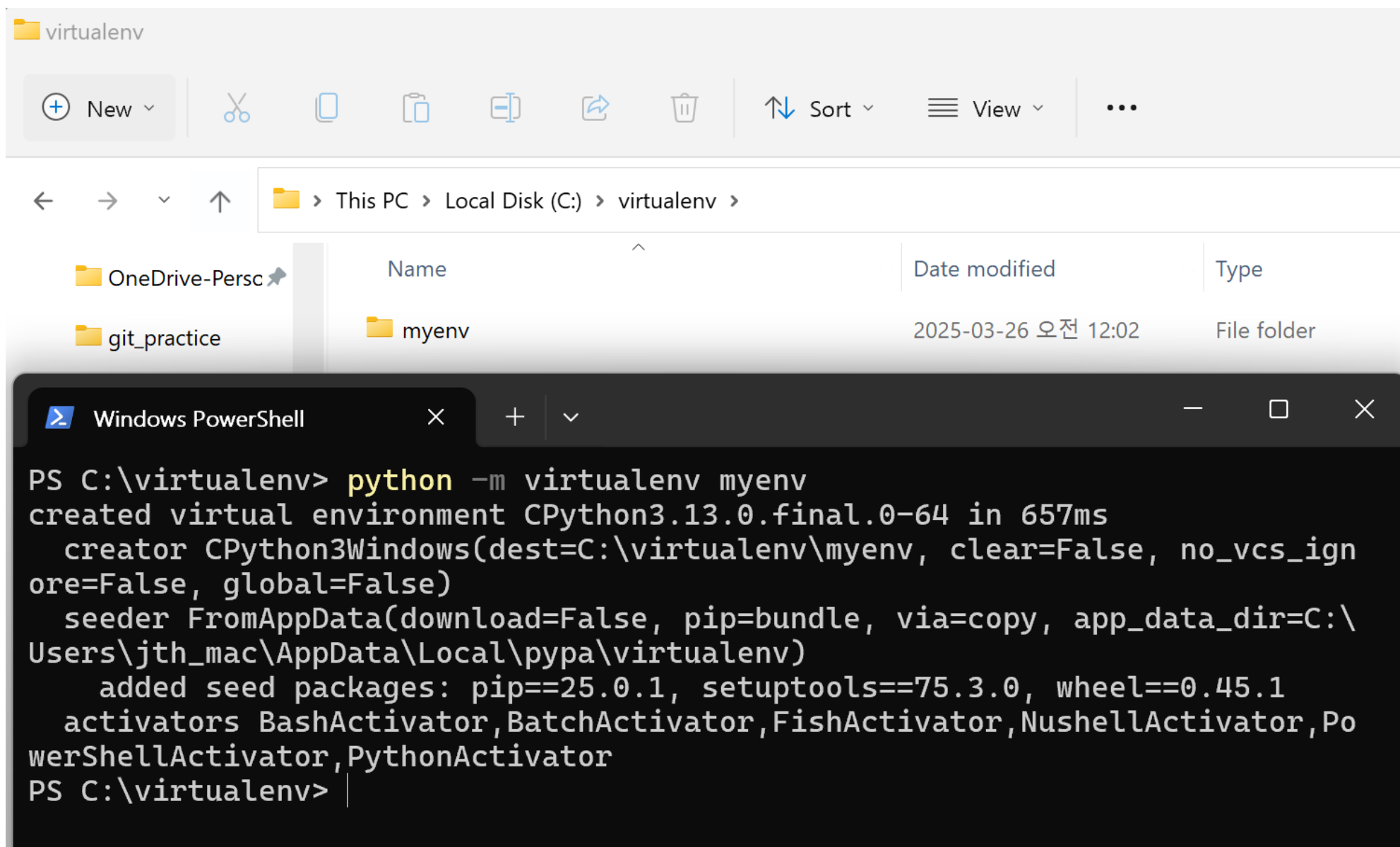
▪ Python 버전 설정 방법

○ \$ virtualenv -p python버전 가상_환경_이름

- 예: virtualenv -p python3.10 myenv

○ 설치하려는 버전의 Python을 설치하지 않았다면 별도의 설치가 필요

- 여러 버전의 MS Office가 하나의 운영체제 내에 존재할 수 있듯
Python 3.5, 3.9, 3.12 등은 독립적으로 존재 가능함



가상 환경 활성화 [1/2]

- 가상 환경을 생성하는 것만으로는 사용할 수 없으며
사용하고자 할 때, 즉 코드를 실행시키고자 할 때마다 활성화해야 함
 - 가상 환경을 활성화하지 않으면 전역 환경에서 실행됨
- 사용법
 - Windows
 - \$ 가상_환경_경로\Script\activate
 - macOS/Linux
 - \$ source 가상_환경_경로/bin/activate

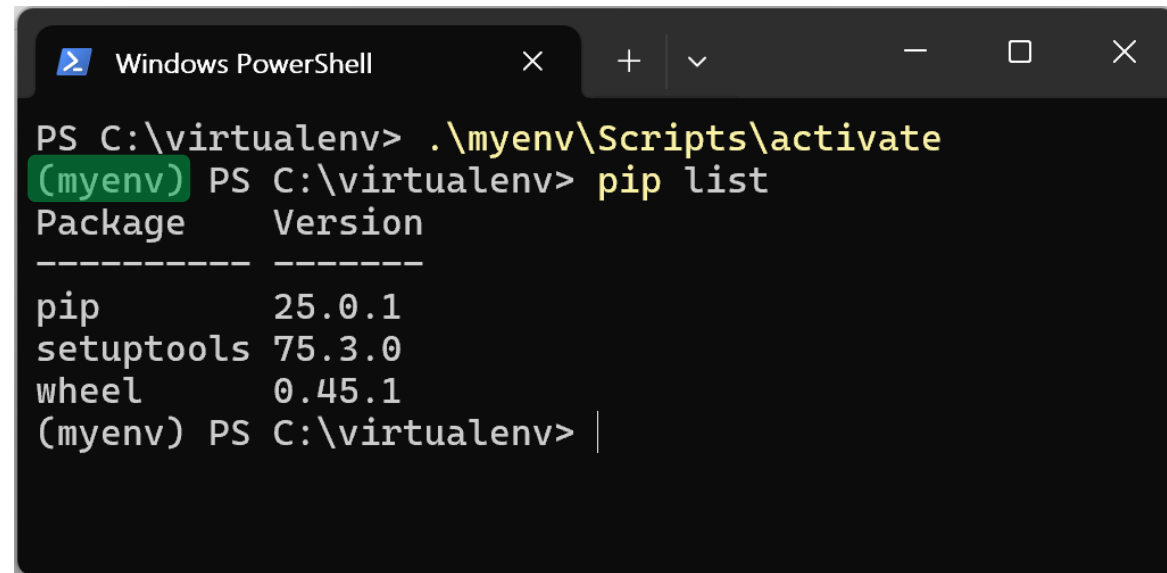
가상 환경 활성화 [2/2]

- 실행하면 터미널 좌측에

(가상_환경_이름) 표시됨

- pip list를 입력해 보면 최소의 기본 라이브러리만 설치돼 있는 것 확인 가능

- 즉 Python 최초 설치 시의 순정 상태와 동일함



```
Windows PowerShell
PS C:\virtualenv> .\myenv\Scripts\activate
(myenv) PS C:\virtualenv> pip list
Package      Version
-----
pip          25.0.1
setuptools   75.3.0
wheel        0.45.1
(myenv) PS C:\virtualenv> |
```

가상 환경 비활성화

▪ 사용법

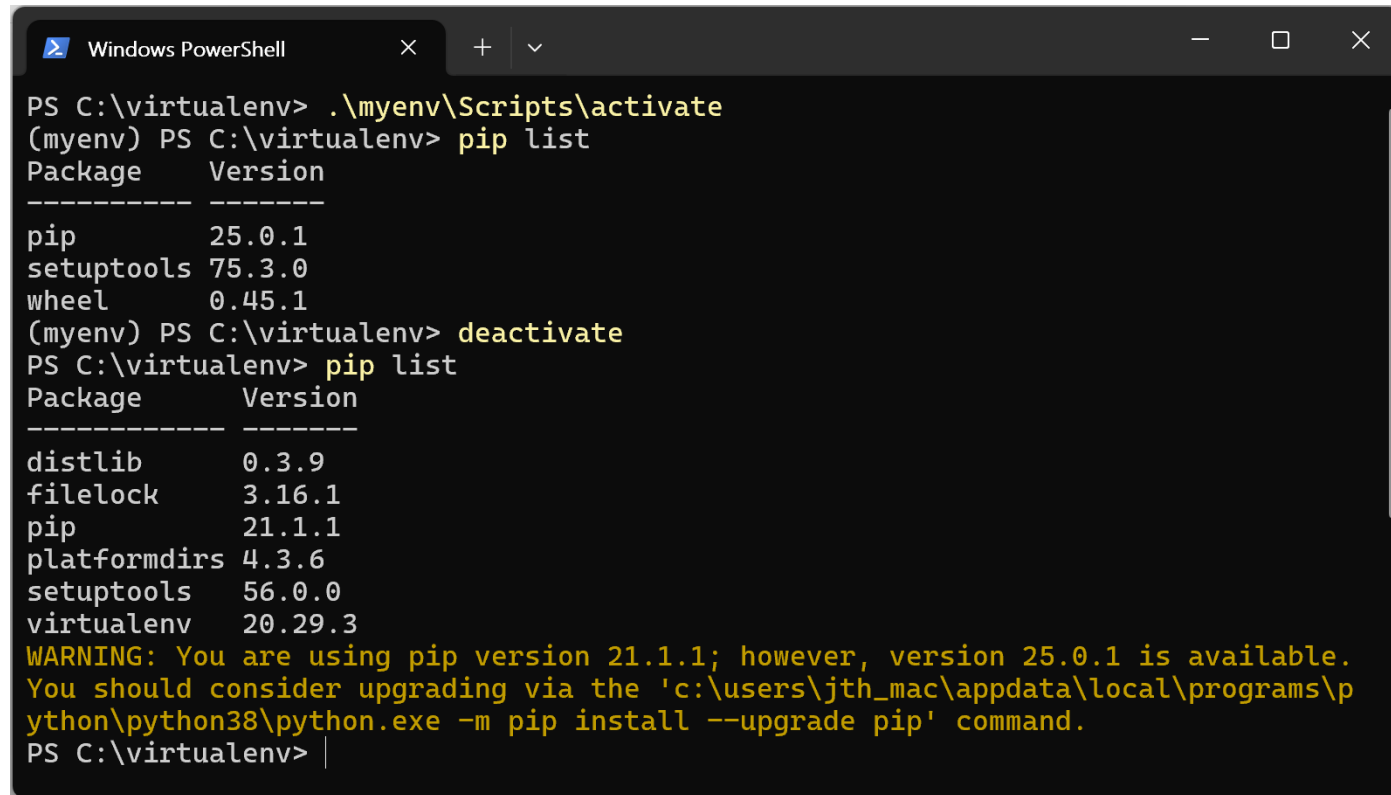
○ 가상 환경 활성화 상태에서
다음 커맨드 입력

- \$ deactivate

▪ 터미널 좌측의

(가상_환경_이름) 사라짐

▪ pip list로 라이브러리 설치 상태가 다름 확인 가능

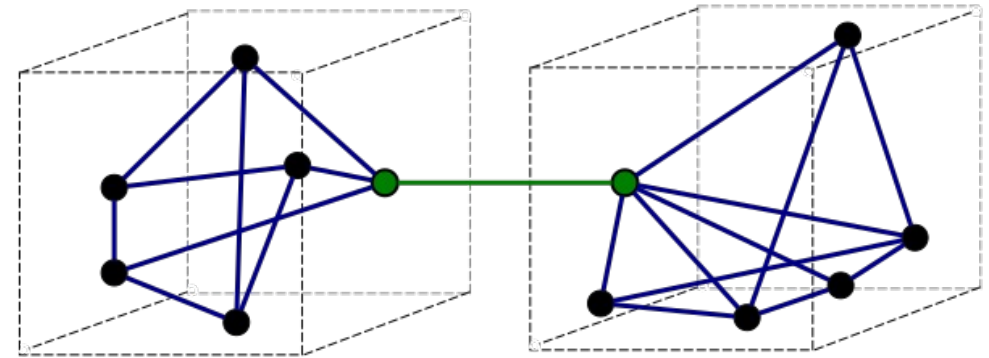


```
Windows PowerShell
PS C:\virtualenv> .\myenv\Scripts\activate
(myenv) PS C:\virtualenv> pip list
Package      Version
-----
pip          25.0.1
setuptools   75.3.0
wheel        0.45.1
(myenv) PS C:\virtualenv> deactivate
PS C:\virtualenv> pip list
Package      Version
-----
distlib      0.3.9
filelock     3.16.1
pip          21.1.1
platformdirs 4.3.6
setuptools   56.0.0
virtualenv   20.29.3
WARNING: You are using pip version 21.1.1; however, version 25.0.1 is available.
You should consider upgrading via the 'c:\users\jth_mac\appdata\local\programs\python\python38\python.exe -m pip install --upgrade pip' command.
PS C:\virtualenv> |
```

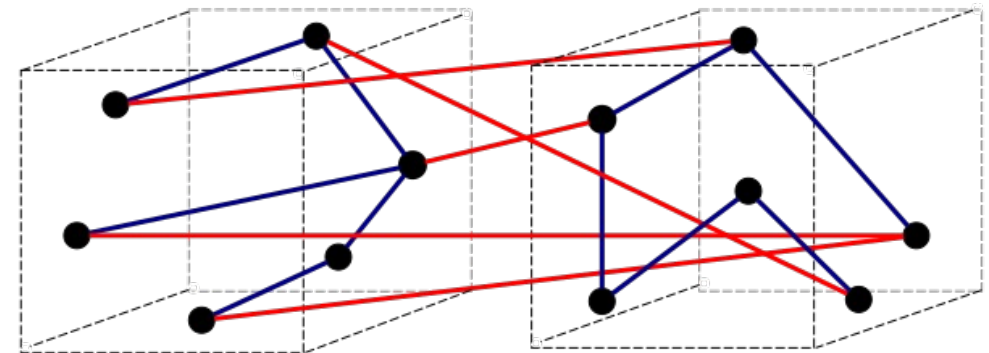
4. 의존성 관리

의존성 (Dependency) [1 / 2]

- 소프트웨어들이 상호 의존하는 정도
- 본 강의의 맥락에서는
하나의 프로젝트와 관련되는 코드,
라이브러리들이 상호 참조를 하는
정도와 관련됨
 - 참조한다는 것은
변수, 함수, 클래스를 타 라이브러리에서
import해서 쓰는 것



a) Good (loose coupling, high cohesion)



b) Bad (high coupling, low cohesion)

[Dependency \(Coupling\)](#)

의존성 (Dependency) [2/2]

▪왜 필요한가?

- 의존성 없이 개발을 하면 모든 기능을 직접 구현해야 함
 - Don't reinvent the wheel.
- 의존성을 이용하면 개발 속도를 향상시키고 코드의 공유, 유지 보수가 간편해짐

▪문제

- 버전 충돌
 - 라이브러리 A가 의존하는 것은 numpy 1.02, 내 환경에 설치된 것은 numpy 1.21일 때
내 환경에서 라이브러리 A를 이용하는 코드가 실행되지 않거나 의도치 않은 결과를 줄 수 있음
- 의존하고 있는 라이브러리가 삭제되거나 업데이트될 시 문제 발생할 수 있음

pip를 이용한 의존성 관리 [1 / 3]

- pip에서 라이브러리의 버전을 지정하여 설치할 수 있음

- 사용법

- \$ pip install 라이브러리명==버전

- 예

- \$ pip install pandas==1.3.2

- 연습

- 1) pip로 버전 지정 없이 pandas를 설치한 후 pip show로 버전 확인

- 2) 위 방법으로 버전을 지정하여 pandas를 설치한 후 pip show로 버전 확인

pip를 이용한 의존성 관리 [2/3]

- pip로 현재 환경에 설치된 모든 라이브러리 정보를 한꺼번에 관리 가능

- 사용법

- \$ pip freeze > requirements.txt

- 설치된 라이브러리 및 버전 정보를 현재 디렉토리에 requirements.txt라는 이름의 파일로 저장하라는 의미의 커맨드

- requirements.txt는 이 맥락에서 쓰이는 관습적인 파일명이므로 따르는 것이 좋음

- 연습

- 별개의 환경에서 requirments.txt 생성한 후 결과 확인

pip를 이용한 의존성 관리 [3/3]

▪ requirements.txt를 이용한 개발 환경 (라이브러리 및 버전) 재현

- 가상 환경 & requirements.txt만 있으면, 내 컴퓨터에서도 타인의 개발 환경을 손쉽게 재현 가능
 - Python 코드 공유의 핵심

▪ 사용법

- `$ pip install -r requirements.txt`
 - 현재 디렉토리에 있는 requirements.txt에 기술된 라이브러리 정보대로 설치 진행하라는 의미의 커맨드

▪ 연습

- 1) 하나의 환경에서 requirements.txt 생성
- 2) 새로운 가상 환경 생성 후 requirements.txt로부터 1)의 환경 복원

5. GitHub 저장소를 이용한 연습

대략적인 안내

- 연습용 GitHub repo의 run.py 성공적으로 실행시키기

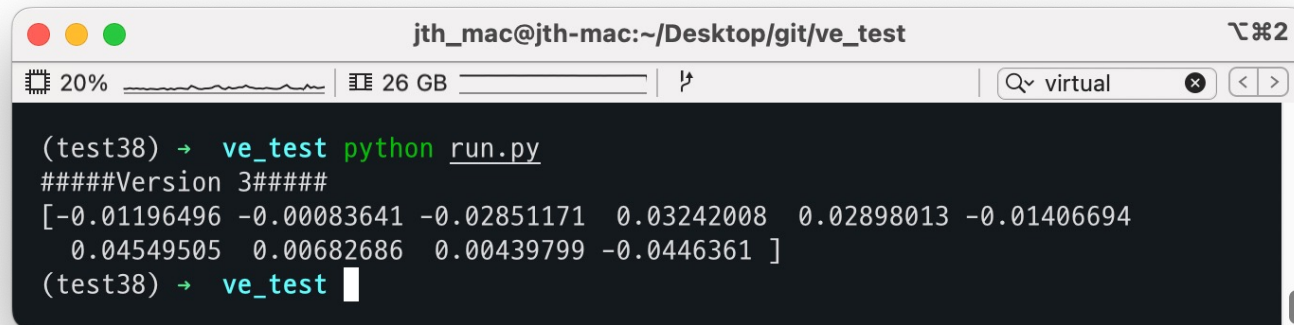
- https://github.com/taeheejeon22/ve_test

- 가상 환경 사용 필수

- 가상 환경의 Python 버전에 따라 특정 버전의 라이브러리 설치가 안 될 수 있음

- 낮은 버전의 Python (예: Python 3.8) 설치하여 새롭게 가상 환경 생성해 볼 것

- run.py가 문제 없이 실행되면 해결 완료



```
jth_mac@jth-mac:~/Desktop/git/ve_test
20% 26 GB virtual
(test38) → ve_test python run.py
#####Version 3#####
[-0.01196496 -0.00083641 -0.02851171  0.03242008  0.02898013 -0.01406694
 0.04549505  0.00682686  0.00439799 -0.0446361 ]
(test38) → ve_test
```

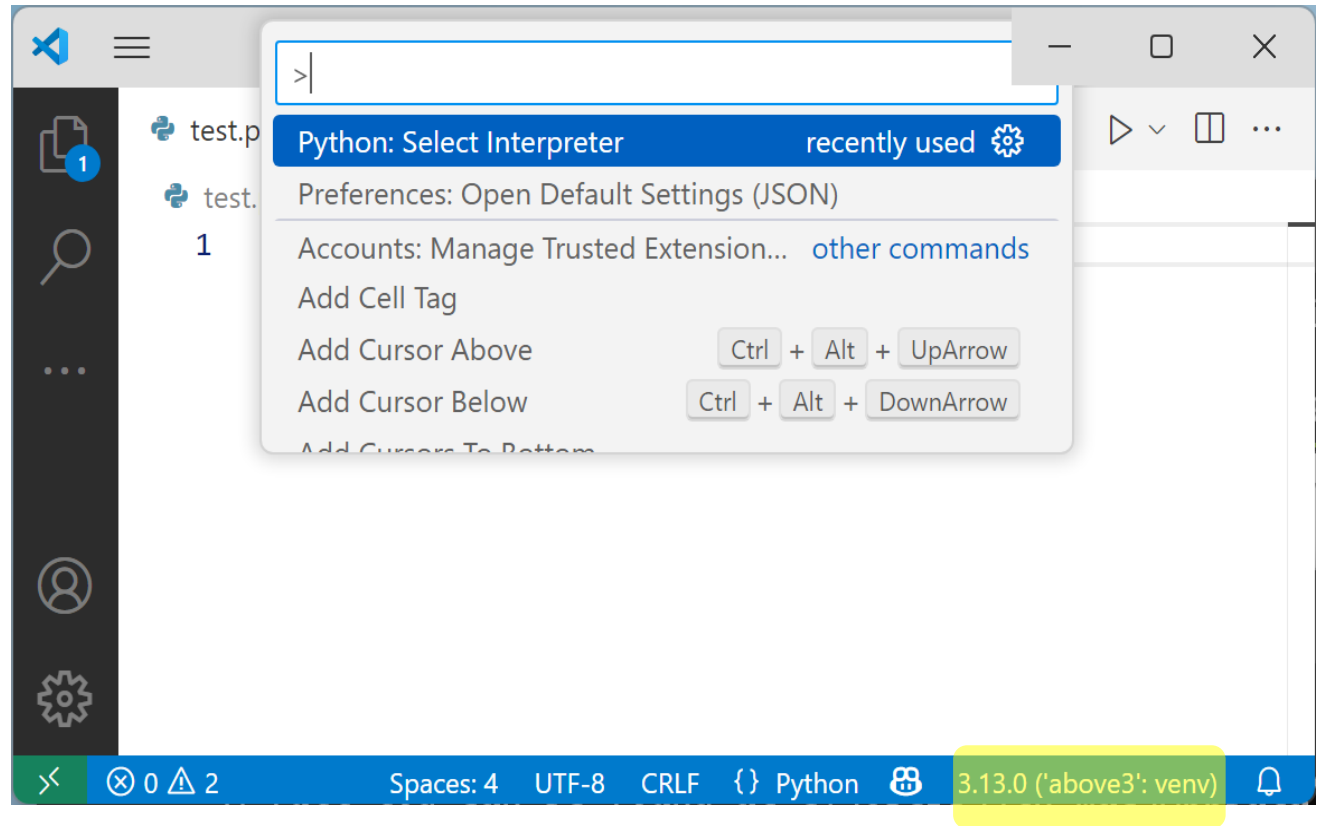
6. 마무리

VS Code에서 가상 환경 사용하기 [1/3]

■ 가상 환경 선택

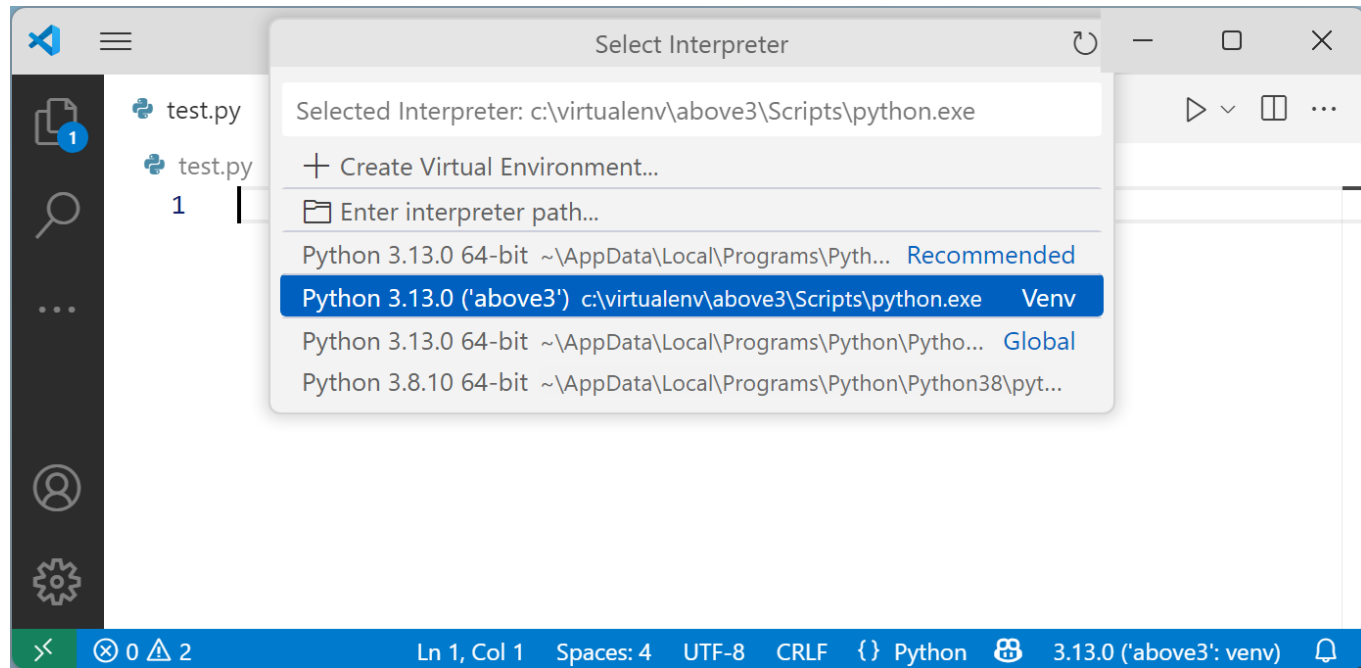
- Ctrl (Cmd) + Shift + P
입력 후 Python: Select
Interpreter 검색

- 혹은 우측 하단의 Python
인터프리터 부분 클릭



VS Code에서 가상 환경 사용하기 [2/3]

- 목록에서 내가 생성한 가상 환경이 보인다면 클릭
- 없다면
Enter interpreter path > Find
클릭 후 직접 인터프리터 지정해야 함



VS Code에서 가상 환경 사용하기 [3/3]

▪인터프리터 지정

- 가상 환경 생성한 폴더 경로로 이동 후 하위 폴더에서 인터프리터 찾기

- Windows

 - 가상_환경_폴더/Script/python.exe

- macOS/Linux

 - 가상_환경_폴더/bin/python

마치면서

- 가상 환경의 존재, 사용법을 아는지 여부는 초보 판별의 중요한 기준
- 실전에서는 사실상 사용이 필수
- 다만 가상 환경이 만능은 아님
 - Python 외부의 도구의 버전 문제를 해결할 수 없음
 - 운영체제의 유형, 버전 차이에 따른 문제를 해결할 수 없음
 - 관심 있는 사람은 Docker라는 것을 공부해 보기를