Python 가상환경의 거의 모든 것

virtualenv부터 uv까지, Python 가상환경의 역사와 미래

이승헌

새로운 Python 프로젝트를 시작한다

```
mkdir new-project // 1. 디렉토리 생성
cd new-project

python -m venv venv // 2. 가상환경 생성

source venv/bin/activate // 3. 가상환경 활성화

pip install django // 4. 의존성 설치


```

1. 디렉토리를 생성한다.
2. 가상환경을 생성한다.
3. 가상환경을 활성화한다.
4. 필요한 의존성을 설치한다.
5. 개발 시작!

우리 곁에 살아숨쉬는 가상환경(venv)

why?

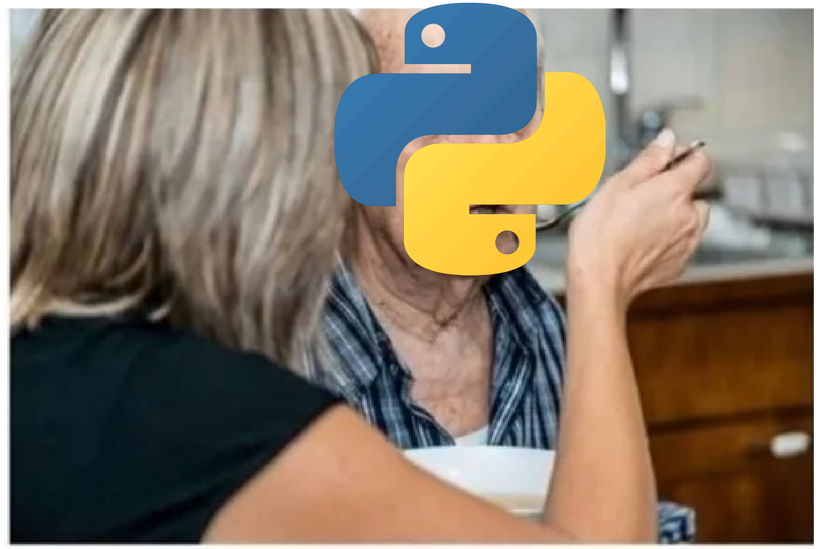
how?

Why: 어쩌다 가상환경을 사용하게 되었는가

가상환경을 따로 구축하는 언어는 흔치 않다.

Python은 생각보다 역사가 깊다.

상상



또 이러신다.. 밥이나 드세요

현실

Python은 생각보다 나이가 많다.

91년 2월생

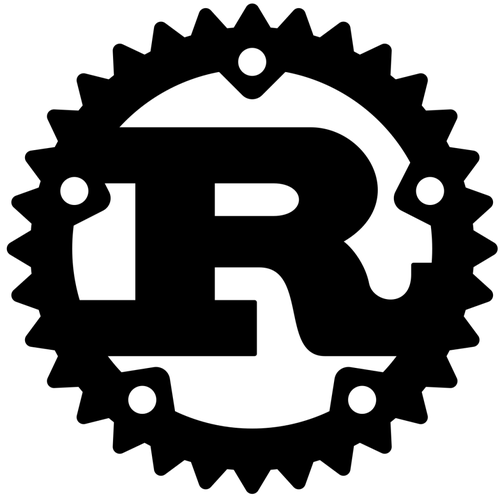
91년 8월생
(발표 기준)

95년생

Python은 생각보다 나이가 많다.

91년 2월생

09년생

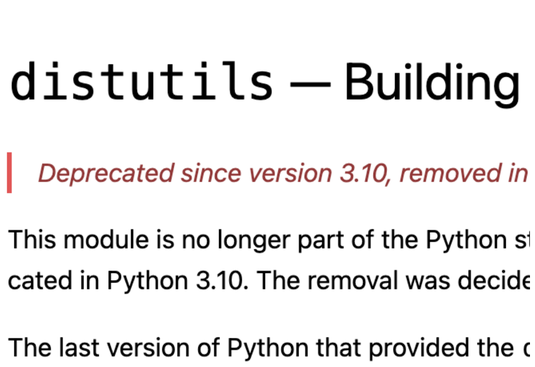


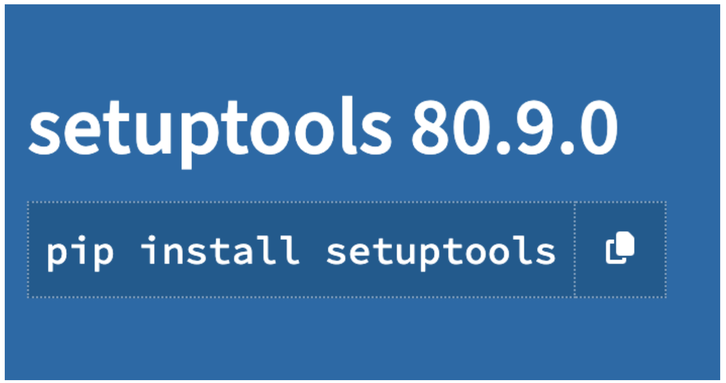
10년생

초창기의 Python: Before 2007

무법과 낭만의 시대: Everything in global site-packages

초창기의 Python: Before 2007



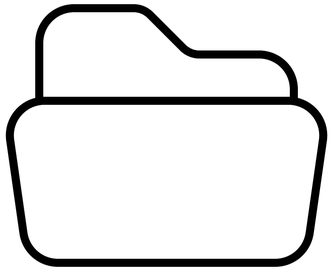


distutils(1998):
최초의 파이썬 패키지 빌드 툴

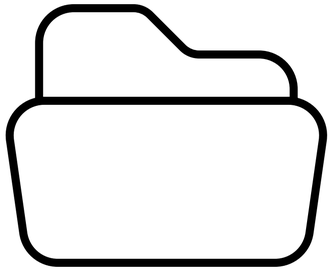
PyPI(2001):
파이썬 패키지 저장소

setuptools, easy_install (2004):
파이썬 패키지 설치기

초창기의 Python: Before 2007



Project 1:
djnago==2.2.0



Project 2:
djnago==3.2.0

virtualenv의 등장 (2007)

Ian Bicking

pip, virtualenv...

virtualenv의 등장 (2007)

```
virtualenv my_project_env

source my_project_env/bin/activate

pip install django==3.2.0
```

각 프로젝트를 위한 독립적인 파이썬 환경

virtualenv를 표준으로!: PyPA

The Python Packaging Authority (PyPA) is a working group that maintains a core set of software projects used in Python packaging.

virtualenv를 표준으로!: PEP 405 (2011)

PEP 405 – Python Virtual Environments

Author:	Carl Meyer <carl at oddbird.net>
BDFL-Delegate:	Alyssa Coghlan
Status:	Final
Type:	Standards Track
Topic:	Packaging
Created:	13-Jun-2011
Python-Version:	3.3
Post-History:	24-Oct-2011, 28-Oct-2011, 06-Mar-2012, 24-May-2012
Resolution:	Python-Dev message

▶ **Table of Contents**

Abstract

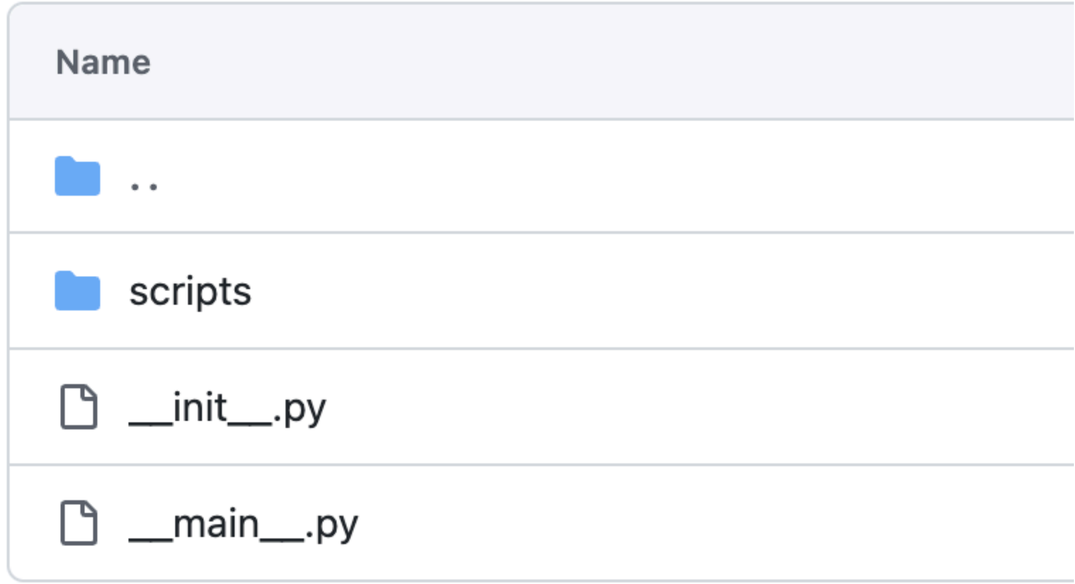
This PEP proposes to add to Python a mechanism for lightweight “virtual environments” with their own site directories, optionally isolated from system site directories. Each virtual environment has its own Python binary (allowing creation of environments with various Python versions) and can have its own independent set of installed Python packages in its site directories, but shares the standard library with the base installed Python.

To make lightweight virtualenvs in Python built-in!

venv의 탄생

```
mkdir new-project // 1. 디렉토리 생성
cd new-project

python -m venv venv // 2. 가상환경 생성

source venv/bin/activate // 3. 가상환경 활성화

pip install django // 4. 의존성 설치


```

venv(pkg)의 내부 동작



cpython/Lib/venv

venv(가상환경)의 내부 구조

```
venv/
├── bin/                    ---------- OS 별 스크립트
│   ├── Activate.ps1
│   ├── activate
│   ├── activate.csh
│   ├── activate.fish
│   ├── pip
│   ├── pip3
│   ├── pip3.12
│   ├── python
│   ├── python3
│   └── python3.12
├── include/                ---------- C header files
├── lib/site-packages       ---------- Packages
└── pyvenv.cfg              ---------- metadata for venv
```

pyvenv.cfg

```
home = /Library/Frameworks/Python.framework/Versions/3.12/bin

include-system-site-packages = false

version = 3.12.5

executable = /Library/Frameworks/Python.framework/Versions/3.12/bin/python3.12

command = /Library/Frameworks/Python.framework/Versions/3.12/bin/python -m venv /Users/name/path/to/project/venv
```

venv activation

1. It Copies Structure and Files / 구조와 파일을 복사한다
2. It Adapts the Prefix-Finding Process / 프리픽스(경로) 찾기 과정을 적응한다
3. It Links Back to Your Standard Library / 표준 라이브러리와 연결한다
4. It Modifies Your PYTHONPATH / PYTHONPATH를 수정한다
5. It Changes Your Shell PATH Variable on Activation / 활성화 시 셸 PATH 변수를 바꾼다
6. It Runs From Anywhere With Absolute Paths / 절대 경로로 어디서든 동작한다

It Copies Structure and Files/구조와 파일을 복사한다

```
venv/

├── bin/
│   ├── Activate.ps1
│   ├── activate
│   ├── activate.csh
│   ├── activate.fish
│   ├── pip
│   ├── pip3
│   ├── pip3.12
│   ├── python
│   ├── python3
│   └── python3.12
│
├── include/
│
├── lib/site-packages
│
└── pyvenv.cfg
```

It Adapts the Prefix-Finding Process / 프리픽스(경로) 찾기 과정을 위한 준비

1. sys.base_prefix: venv를 생성할 때 참조한 python exe의 경로
2. sys.prefix: pyvenv.cfg를 포함한 디렉토리의 경로

It Links Back to Your Standard Library / 표준 라이브러리와 연결한다

home = /Library/Frameworks/Python.framework/Versions/3.12/bin
include-system-site-packages = false
...

```
>>> import sys
>>> sys.path
['',
 '/Library/Frameworks/Python.framework/Versions/3.12/lib/python312.zip',
 '/Library/Frameworks/Python.framework/Versions/3.12/lib/python3.12',
 '/Library/Frameworks/Python.framework/Versions/3.12/lib/python3.12/lib-dynload',
 '/Users/name/path/to/venv/lib/python3.12/site-packages']
```

It Modifies Your PYTHONPATH / PYTHONPATH를 수정한다

```
>>> import sys
>>> sys.path
['',
 '/Library/Frameworks/Python.framework/Versions/3.12/lib/python312.zip',
 '/Library/Frameworks/Python.framework/Versions/3.12/lib/python3.12',
 '/Library/Frameworks/Python.framework/Versions/3.12/lib/python3.12/lib-dynload',
 '/Library/Frameworks/Python.framework/Versions/3.12/lib/python3.12/site-packages']
```

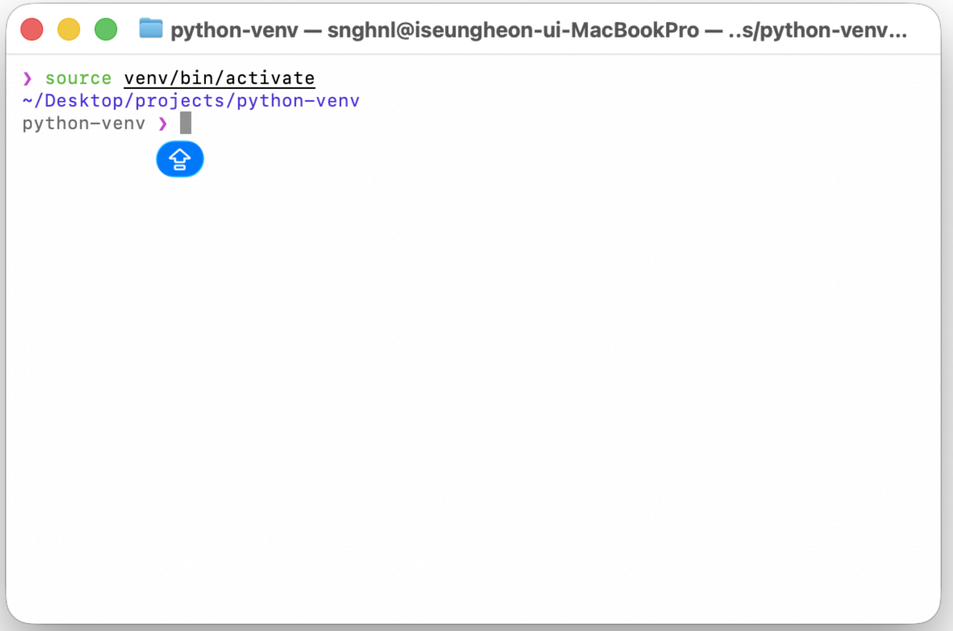
before activation

```
>>> import sys
>>> sys.path
['',
 '/Library/Frameworks/Python.framework/Versions/3.12/lib/python312.zip',
 '/Library/Frameworks/Python.framework/Versions/3.12/lib/python3.12',
 '/Library/Frameworks/Python.framework/Versions/3.12/lib/python3.12/lib-dynload',
 '/Users/name/path/to/venv/lib/python3.12/site-packages']
```

after activation

It Changes Your Shell PATH Variable on Activation / 활성화 시 셸 PATH 변수를 바꾼다

```
>>> import sys
>>> sys.path
['',
 '/Library/Frameworks/Python.framework/Versions/3.12/lib/python312.zip',
 '/Library/Frameworks/Python.framework/Versions/3.12/lib/python3.12',
 '/Library/Frameworks/Python.framework/Versions/3.12/lib/python3.12/lib-dynload',
 '/Users/name/path/to/venv/lib/python3.12/site-packages']
```



1. PATH

2. Command prompt

It Runs From Anywhere With Absolute Paths / 절대 경로로 어디서든 동작한다

```
$ /Users/name/path/to/venv/bin/python
```

그래도 불편하다

```
argcomplete
# testing
build~=1.2.0
click<8.2
hypothesis
mypy==1.11.0
# lint
pre-commit
# (FIXME: 2024/01/28) The latest pytest-asyncio 0.23.3 is not compatible with pytest 8.0.0
pytest<8.0.0
pytest-asyncio
pytest-benchmark
pytest-cov
pytest-httpserver
pytest-pyodide==0.58.6
setuptools; python_version>='3.12'
sphinx-click
```

```
pip freeze > requirements.txt

pip install -r requirements.txt
```

종속성 관리와 종속성 설치의 불편함

Third-Party Packages의 등장

Poetry

uv

Third-Party Packages: Poetry

```
poetry new <project>  // 프로젝트 시작
cd <project>

poetry add <package>  // 의존성 설치
poetry install // 의존성 설치 (pyproject.toml)

poetry run python main.py  // 가상환경에서 실행

poetry shell  // 가상환경 접속
```

Third-Party Packages: uv

```
uv init <project>
cd <project>

uv add <package> // 의존성 설치

uv run python main.py // 가상환경에서 실행

uv venv // 가상환경 생성

uv sync //의존성 설치 (pyproject.toml)
```

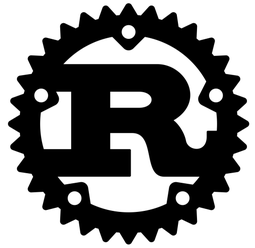
Python 종속성 관리의 미래?

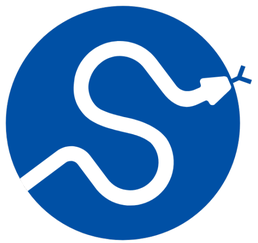
개발환경 분리 종속성 설치 종속성 관리 패키지 빌드 ...

Python 종속성 관리의 미래?

개발환경 분리 종속성 설치 종속성 관리 패키지 빌드 ...

Python Packages는 다양한 언어로 개발된다





Python은 다양한 곳에서 사용된다

OS
Python

브라우저
Pyodide

임베디드
MicroPython

WheelNext

Welcome to WheelNext

What is WheelNext and Why should you participate !?

WheelNext is an open-source initiative (https://github.com/wheelnext & https://wheelnext.dev/) aiming to improve the user experience in the Python packaging ecosystem, specifically around the scientific computing and machine/deep learning space. We also anticipate benefits in other domains that heavily rely on performance of compiled Python extension modules - the benefit of utilizing one's hardware more optimally is not exclusive to any single domain.

The current state of Python packaging was designed and implemented at a time predating the increasingly complex needs these use cases demand were prevalent. Through WheelNext, we aim to evolve Python packaging standards to better address the needs of this important segment, while retaining its effectiveness for the overall ecosystem.

Many of the problems and gaps are summarized on the following website: https://pypackaging-native.github.io/ The WheelNext objective is to improve the state of packaging for Python native code, making it easier to build, distribute, and install Python packages with complex native dependencies.

E.O.D