Python Programming with uv

Hackers Cafe presentation
Gavin Wiggins https://gavinw.me
August 5, 2025

What is uv?

- A fast Python package and project manager tool written in Rust
- Developed by Astral who also created ruff (linter, formatter) and ty (type checker)
- Replaces tools like pip, tox, conda, pipenv, poetry, pdm, pipx, and more for Python projects
- Easily manage different versions of Python with uv
- Uses a global cache for dependency deduplication
- Install and run tools that are published as Python packages
- The most simple and easiest way to install Python and run Python code and tools

Why not use system Python?

- **Risk of Breaking System Tools:** Many operating systems use their system Python installation to run essential tools. Installing or upgrading packages globally can break these tools or even destabilize your OS.

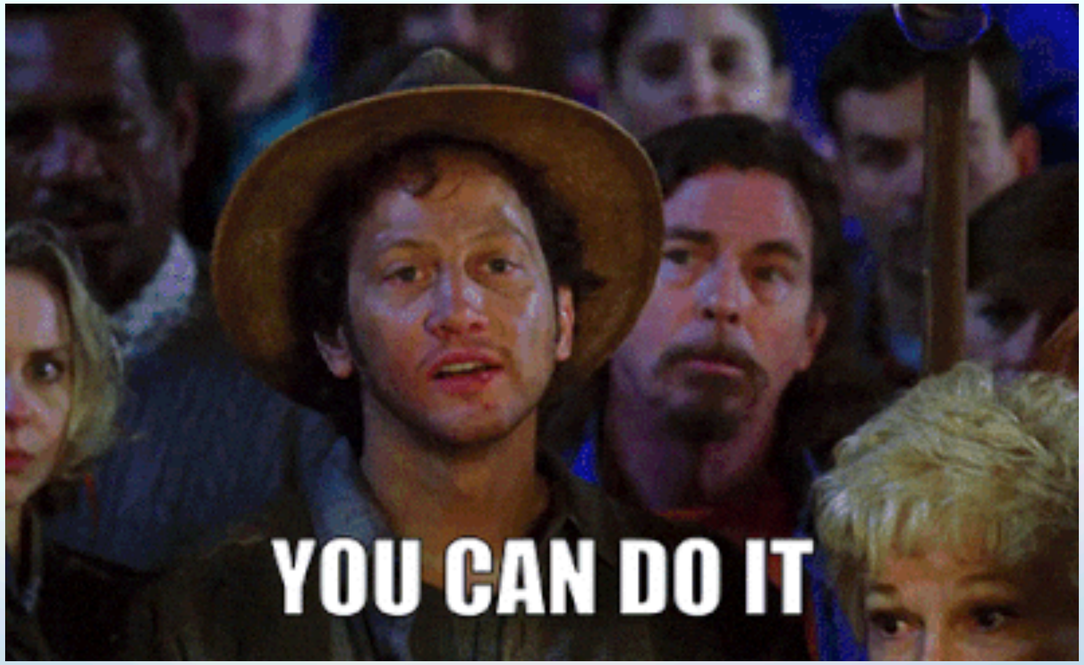
- **Permission Issues:** Modifying the system Python usually requires administrator or root privileges, which is a security risk and an inconvenience.

- **Version Conflicts:** The system Python version is often outdated and may not match the requirements of your project. Different projects may require different Python or package versions, which system Python cannot flexibly support.

- **Reproducibility Problems:** Sharing code or deploying to other machines becomes unreliable because system Python versions and installed packages can vary widely across platforms.

- **Lack of Flexibility and Control:** You have little control over the version of Python installed with your OS, and you might not be able to use newer features unless the OS itself updates.

Always use version management tools to create isolated virtual environments for each Python project. This approach protects your system, simplifies dependency management, and provides a reproducible development environment.



Create a Python project with uv

(a directory of Python files)

Create a Python project

```
$ uv init my-project

$ cd my-project

$ uv add numpy

$ uv run main.py

Hello from my-project!
a is [1 2 3 4 5]
```

```
my-project/
```

Initial project structure

```
$ uv init my-project

$ cd my-project

$ uv add numpy

$ uv run main.py

Hello from my-project!
a is [1 2 3 4 5]
```

```
my-project/
├── .git/
├── .gitignore
├── .python-version
├── main.py
├── pyproject.toml
└── README.md
```

Add a dependency

```
$ uv init my-project

$ cd my-project

$ uv add numpy

$ uv run main.py

Hello from my-project!
a is [1 2 3 4 5]
```

```
my-project/
├── .git/
├── .gitignore
├── .python-version
├── .venv/
├── main.py
├── pyproject.toml
├── README.md
└── uv.lock
```

Add a dependency

```
$ uv init my-project

$ cd my-project

$ uv add numpy

$ uv run main.py

Hello from my-project!
a is [1 2 3 4 5]
```

```
my-project/
├── .git/
├── .gitignore
├── .python-version
├── .venv/
├── main.py
├── pyproject.toml
├── README.md
└── uv.lock
```

```toml
# pyproject.toml

[project]
name = "my-project"
version = "0.1.0"
description = "Add your description here"
readme = "README.md"
requires-python = ">=3.13"
dependencies = [
    "numpy>=2.3.0",
]
```

Run the main file

```
$ uv init my-project

$ cd my-project

$ uv add numpy

$ uv run main.py

Hello from my-project!
a is [1 2 3 4 5]
```

```python
# main.py

import numpy as np


def main():
    print("Hello from my-project!")

    a = np.array([1, 2, 3, 4, 5])
    print("a is", a)


if __name__ == "__main__":
    main()
```

Compare uv to standard Python

```
$ uv init my-project

$ cd my-project

$ uv add numpy

$ uv run main.py
```

With uv

Without uv

```
$ mkdir my-project

$ cd my-project

$ touch README.md main.py pyproject.toml

$ git init

$ python -m venv .venv

$ source .venv/bin/activate

$ pip install numpy

$ python main.py

$ deactivate
```

Run Python scripts with uv

(a single Python file)

Script with no dependencies

```
$ cd my-scripts

$ uv run example.py

Script using standard Python
Random number is 0.716128781864
```

```python
# my-scripts/example.py

import random


def main():
    print("Script using standard Python")
    print("Random number is", random.random())


if __name__ == "__main__":
    main()
```

Run the script

```
$ cd my-scripts

$ uv run example.py

Script using standard Python
Random number is 0.716128781864
```

```python
# my-scripts/example.py

import random


def main():
    print("Script using standard Python")
    print("Random number is", random.random())


if __name__ == "__main__":
    main()
```

Script with dependencies

```
$ cd my-scripts

$ uv add --script example2.py numpy

$ uv run example2.py

Hello from NumPy!
a is [1 2 3 4 5]
```

```python
# my-scripts/example2.py

import numpy as np


def main():
    print("Hello from NumPy!")

    a = np.array([1, 2, 3, 4, 5])
    print("a is", a)


if __name__ == "__main__":
    main()
```

Script with dependencies

```
$ cd my-scripts

$ uv add --script example2.py numpy

$ uv run example2.py

Hello from NumPy!
a is [1 2 3 4 5]
```

```python
# /// script
# requires-python = ">=3.13"
# dependencies = [
#     "numpy",
# ]
# ///

# my-scripts/example2.py

import numpy as np


def main():
    print("Hello from NumPy!")

    a = np.array([1, 2, 3, 4, 5])
    print("a is", a)


if __name__ == "__main__":
    main()
```

Run script with NumPy dependency

```
$ cd my-scripts

$ uv add --script example2.py numpy

$ uv run example2.py

Hello from NumPy!
a is [1 2 3 4 5]
```

```python
# /// script
# requires-python = ">=3.13"
# dependencies = [
#     "numpy",
# ]
# ///

# my-scripts/example2.py

import numpy as np


def main():
    print("Hello from NumPy!")

    a = np.array([1, 2, 3, 4, 5])
    print("a is", a)


if __name__ == "__main__":
    main()
```

Run Python tools with uv

(a Python package with a command-line interface)

Run a Python tool

```
$ uv tool run pycowsay

  ------------
< hello world! >
  ------------

   \   ^__^
    \  (oo)\_______
       (__)\       )\/\
           ||----w |
           ||     ||
```

Run a Python tool

```
$ uv tool run pycowsay hello world!

  ------------
< hello world! >
  ------------

   \   ^__^
    \  (oo)\_______
       (__)\       )\/\
           ||----w |
           ||     ||
```

equivalent

```
$ uvx pycowsay hello world!

  ------------
< hello world! >
  ------------

   \   ^__^
    \  (oo)\_______
       (__)\       )\/\
           ||----w |
           ||     ||
```

Install a Python tool

```
$ uv tool install genja

$ genja -version

25.3

$ uv tool list

genja v25.3.1
- genja
```

More uv commands

Run a Jupyter notebook	`$ uvx jupyter notebook`
View tree of project dependencies	`$ uv tree`
List available Python installations	`$ uv python list`
Add a development dependency	`$ uv add --dev ruff`
Run a Python REPL	`$ uv run python`
Sync lock file with project	`$ uv sync`
Upgrade uv	`$ uv self update`
Load environment variables	`$ uv run --env-file .env app.py`

With uv…

- There is no extra step to install Python
- There is no manual activation of the virtual environment
- There is no pip installation of packages
- There is no manual virtual environment deactivation
- There is just uv

To run a Python file…

```
$ uv run example.py
```

Do this

Not this

```
$ source .venv/bin/activate

$ python example.py

$ deactivate
```

Comparing uv and pixi project creation

Using new Lambda Cloud instance with A100 GPU and no cache

```
$ uv init myproject  [0.073s]

$ cd myproject

$ uv add matplotlib ruff torch torchvision torchaudio  [27.395s]
```

Total = 27 seconds

```
$ pixi init myproject2  [0.017s]

$ cd myproject2

$ pixi add matplotlib ruff  [5.180s]

$ pixi add --pypi torch torchvision torchaudio  [26.912s]
```

Total = 32 seconds

uv is awesome!

https://docs.astral.sh/uv/ for uv installation and usage instructions

https://discord.com/invite/astral-sh for Astral Discord community

https://github.com/astral-sh/uv for uv GitHub repository