Introducing uv: Next-Gen Python Package Manager

codemaker2016.medium.com/introducing-uv-next-gen-python-package-manager-b78ad39c95d7

Vishnu Sivan December 15, 2024

Python evolution has been closely tied to advancements in package management, from manual installations to tools like `pip` and `poetry`. However, as projects grow in complexity, traditional tools often fall short in speed and efficiency.

uv is a cutting-edge Python package and project manager built with Rust, aims to change that. Combining the functionality of tools like `pip`, `poetry`, and `virtualenv`, uv streamlines tasks like dependency management, script execution, and project building—all with exceptional performance. Its seamless compatibility with `pip` commands, requiring no additional learning curve.

In this tutorial, we will explore how to install uv and make the most of its features. From setting up a project and managing dependencies to running scripts and leveraging its enhanced pip interface.

Table of contents

pip limitations

Pip is a widely used package management system written in Python, designed to install and manage software packages. However, despite its popularity, it is often criticized for being one of the slowest package management tools for Python. Complaints about “pip install being slow” are so common that they frequently appear in developer forums and threads.

One significant drawback of pip is its susceptibility to dependency smells, which occur when dependency configuration files are poorly written or maintained. These issues can lead to serious consequences, such as increased complexity and reduced maintainability of projects.

Another limitation of pip is its inability to consistently match Python code accurately when restoring runtime environments. This mismatch can result in a low success rate for dependency inference, making it challenging to reliably recreate project environments.

What is uv

uv is a modern, high-performance Python package manager, developed by the creators of ruff and written in Rust. Designed as a drop-in replacement for `pip` and `pip-tools`, it delivers exceptional speed and compatibility with existing tools.

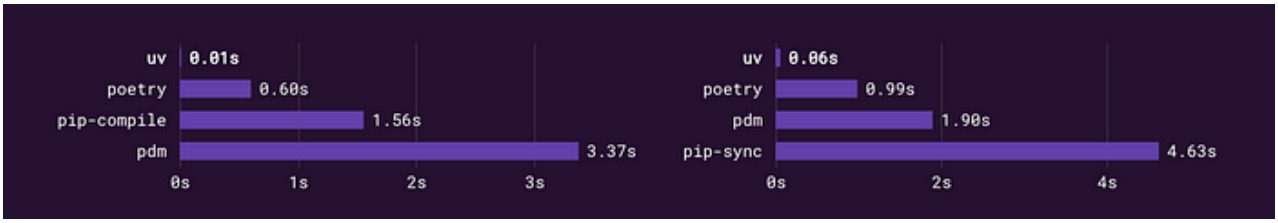
Key features include support for editable installs, Git and URL dependencies, constraint files, custom indexes, and more. uv's standards-compliant virtual environments work seamlessly with other tools, avoiding lock-in or customization. It is cross-platform, supporting Linux, Windows, and macOS, and has been tested extensively against the PyPI index.

Focusing on simplicity, speed, and reliability, uv addresses common developer pain points like slow installations, version conflicts, and complex dependency management, offering an intuitive solution for modern Python development.

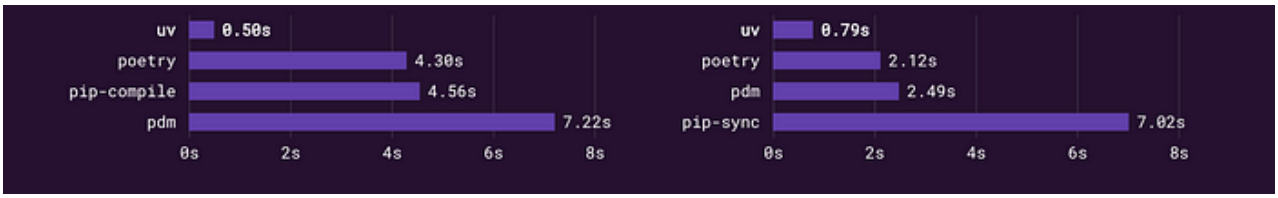
Key features of uv

- : Seamlessly replaces `pip`, `pip-tools`, `virtualenv`, and other tools with full compatibility.
- : 10–100x faster than traditional tools like `pip`, `pip-compile`, and `pip-sync`.
- : Utilizes a global cache for dependency deduplication, saving storage.
- : Installable via `curl`, `pip`, or `pipx` without requiring Rust or Python.
- : Proven performance at scale with the top 10,000 PyPI packages.
- : Fully compatible with macOS, Linux, and Windows.
- : Features include dependency version overrides, alternative resolution strategies, and a conflict-tracking resolver.
- : Best-in-class error handling ensures developers can resolve conflicts efficiently.
- : Supports editable installs, Git dependencies, direct URLs, local dependencies, constraint files, and more.
- : Combines the functionality of tools like `pip`, `pipx`, `poetry`, `pyenv`, `twine`, and more into a single solution.
- : Installs and manages Python versions, runs scripts with inline dependency metadata, and supports comprehensive project workflows.
- : Simplifies project management with consistent and portable lockfiles.
- : Handles scalable projects with Cargo-style workspace management.

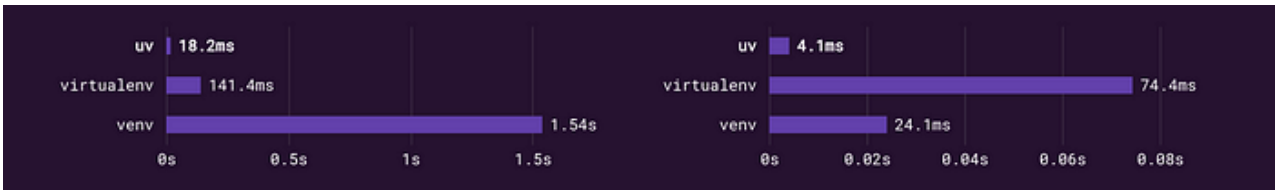
Benchmarks



Resolving (left) and installing (right) dependencies using a warm cache, simulating the process of recreating a virtual environment or adding a new dependency to an existing project.



Resolving (left) and installing (right) dependencies with a cold cache simulate execution in a clean environment. Without caching, **uv** is 8–10x faster than `pip` and `pip-tools`, and with a warm cache, it achieves speeds 80–115x faster.



Creating a virtual environment with (left) and without (right) seed packages like `pip` and `setuptools`. **uv** is approximately 80x faster than `python -m venv` and 7x faster than `virtualenv`, all while operating independently of Python.

> Installing is quick and straightforward. You can opt for standalone installers or install it directly from PyPI.

```
pacman -S uv
```

```
curl -LsSf https://astral.sh/uv/install.sh | sh
```

```
powershell -c
```

```
pip install uv
```

```
pipx install uv
```

```
brew install uv
```

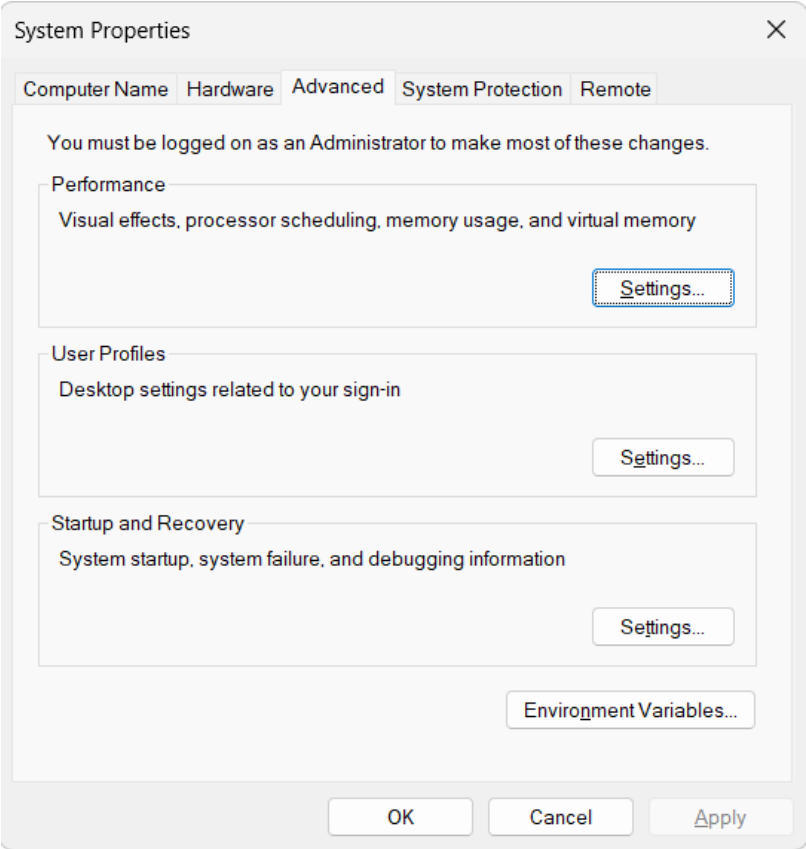
```
C:\Windows\System32\cmd.e
Microsoft Windows [Version 10.0.26100.2454]
(c) Microsoft Corporation. All rights reserved.

E:\Experiments\uv>pip install uv
Defaulting to user installation because normal site-packages is not writeable
Collecting uv
  Downloading uv-0.5.9-py3-none-win_amd64.whl (16.2 MB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 16.2/16.2 MB 3.8 MB/s eta 0:00:00
Installing collected packages: uv
Successfully installed uv-0.5.9
```

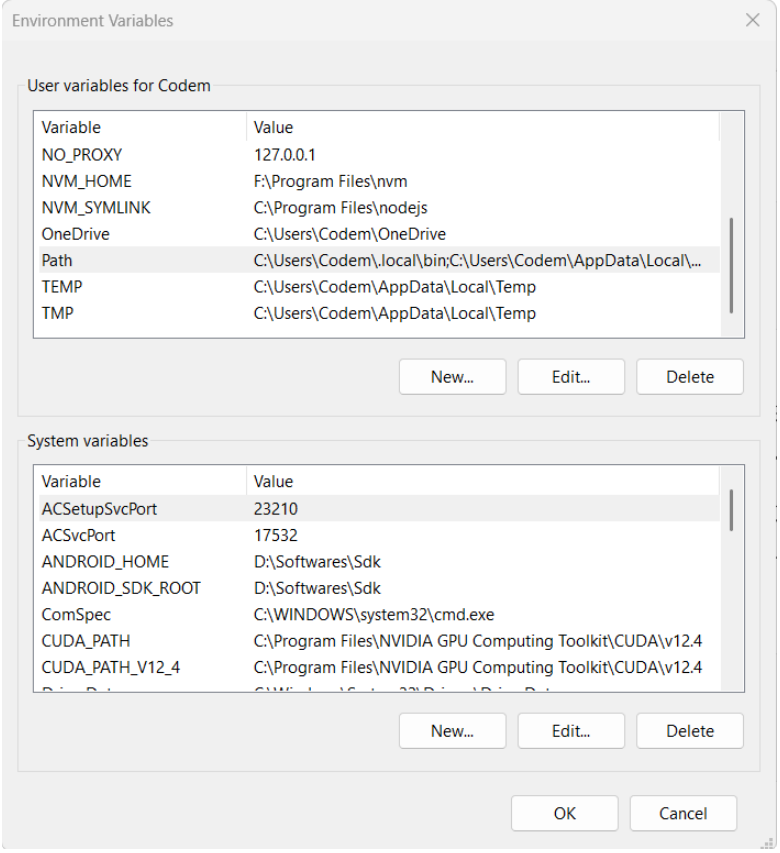
```
Windows PowerShell
Windows PowerShell
Copyright (C) Microsoft Corporation. All rights reserved.

Install the latest PowerShell for new features and improvements! https://aka.ms/PSWindows

PS C:\Users\Codem> powershell -c "irm https://astral.sh/uv/install.ps1 | iex"
Downloading uv 0.5.9 (x86_64-pc-windows-msvc)
Installing to C:\Users\Codem\.local\bin
  uv.exe
  uvx.exe
everything's installed!

To add C:\Users\Codem\.local\bin to your PATH, either restart your system or run:

    set Path=C:\Users\Codem\.local\bin;%Path%   (cmd)
    $env:Path = "C:\Users\Codem\.local\bin;$env:Path"   (powershell)
```

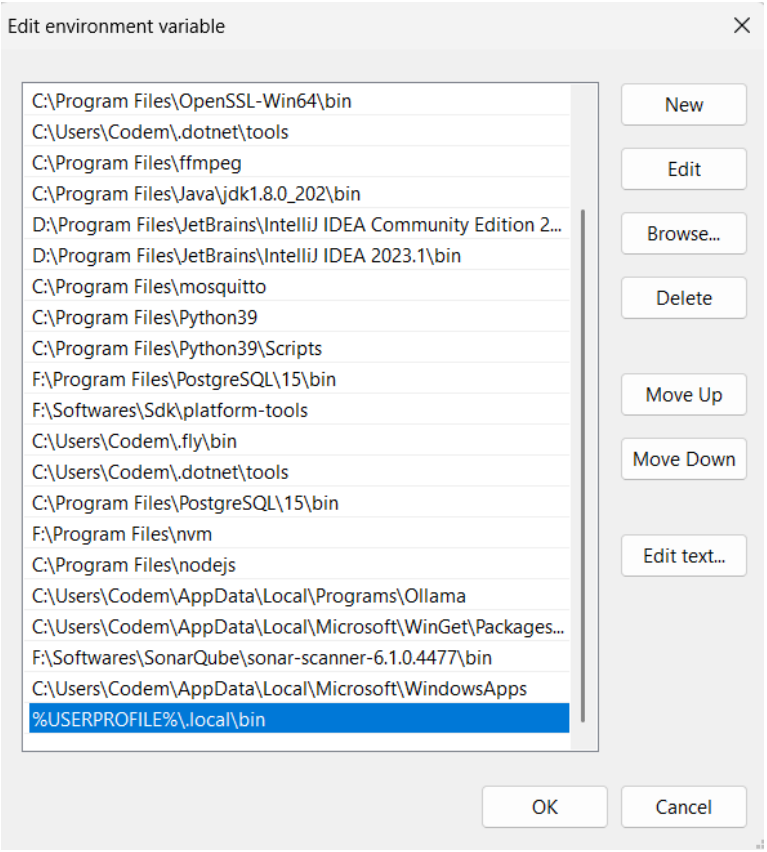
- Before using uv, we have to add the uv path to environment variables.
- , modify the `PATH` environment variable using the following command in the terminal:

PATH=

, To add a directory to the PATH environment variable for both user and system on Windows, search for Environment Variables in the search panel. Under User variables / System variables, select the Path variable, click Edit, then click New and add the desired path.







After the installation, run the uv command in the terminal to verify that it has been installed correctly.

```
C:\WINDOWS\system32\cmd.
Microsoft Windows [Version 10.0.26100.2454]
(c) Microsoft Corporation. All rights reserved.

C:\Users\Codem>uv
An extremely fast Python package manager.

Usage: uv [OPTIONS] <COMMAND>

Commands:
  run      Run a command or script
  init     Create a new project
  add      Add dependencies to the project
  remove   Remove dependencies from the project
  sync     Update the project's environment
  lock     Update the project's lockfile
  export   Export the project's lockfile to an alternate format
  tree     Display the project's dependency tree
  tool     Run and install commands provided by Python packages
  python   Manage Python versions and installations
  pip      Manage Python packages with a pip-compatible interface
  venv     Create a virtual environment
```

Creating virtual environments

Creating a virtual environment with **uv** is simple and straightforward. Use the following command, along with your desired environment name, to create it.

```
uv venv
```

Run the following command to activate the virtual environment.

```
.venv\Scripts\activate
```

```
 .venv/bin/activate
```

```
C:\Windows\System32\cmd.e
Microsoft Windows [Version 10.0.26100.2454]
(c) Microsoft Corporation. All rights reserved.

E:\Experiments\uv>uv venv
Using CPython 3.10.6 interpreter at: C:\Program Files\Python310\python.exe
Creating virtual environment at: .venv
Activate with: .venv\Scripts\activate

E:\Experiments\uv>.venv\Scripts\activate
```

Installing packages

Installing packages into the virtual environment follows a familiar process. The various installation methods are given below.

```
uv pip install flask                uv pip install -r requirements.txt  uv pip
install -e .                 uv pip install         uv pip install
```

```
C:\Windows\System32\cmd.e

(uv) E:\Experiments\uv>uv pip install flask
Resolved 8 packages in 1.16s
Prepared 8 packages in 235ms
[0/8] Installing wheels...
warning: Failed to hardlink files; falling back to full copy. This may lead to degraded performance
.
         If the cache and target directories are on different filesystems, hardlinking may not be s
upported.
         If this is intentional, set `export UV_LINK_MODE=copy` or use `--link-mode=copy` to suppre
ss this warning.
Installed 8 packages in 168ms
 + blinker==1.9.0
 + click==8.1.7
 + colorama==0.4.6
 + flask==3.1.0
 + itsdangerous==2.2.0
 + jinja2==3.1.4
 + markupsafe==3.0.2
 + werkzeug==3.1.3
```

To synchronize the locked dependencies with the virtual environment, use the following command:

```
uv pip  requirements.txt
```

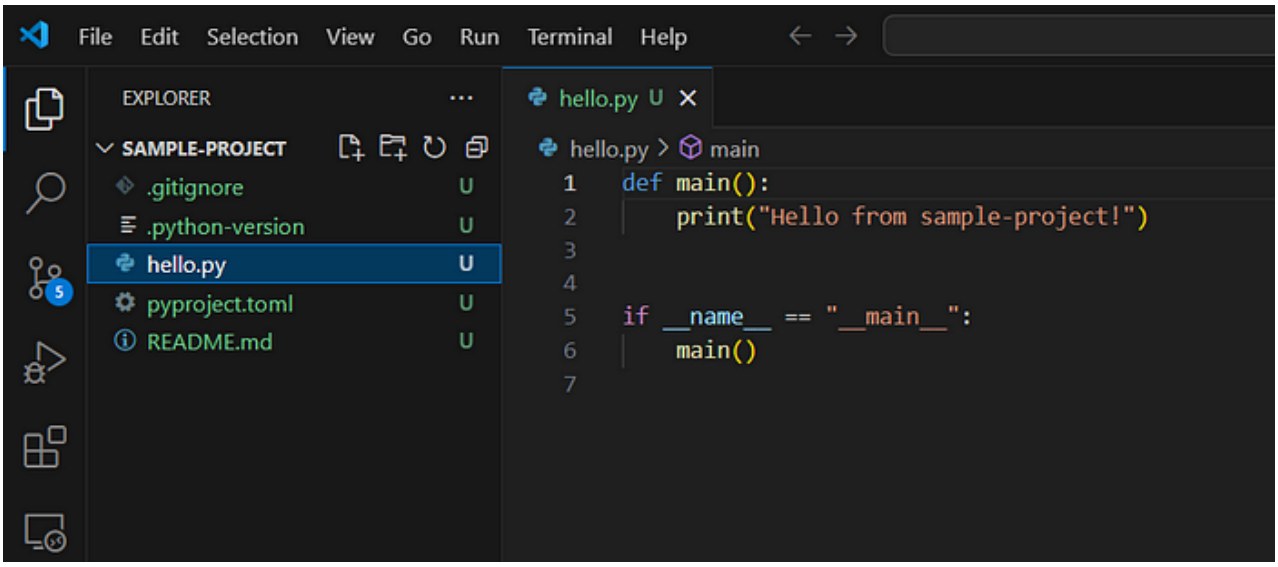
> uv supports a variety of command-line arguments similar to those of existing tools, including `-r requirements.txt`, `-c constraints.txt`, `-e .`, `--index-url`, and more.

Building a flask app using uv

Let's explore some project-related commands with **uv**. Start by initializing a Python project named "sample-project."

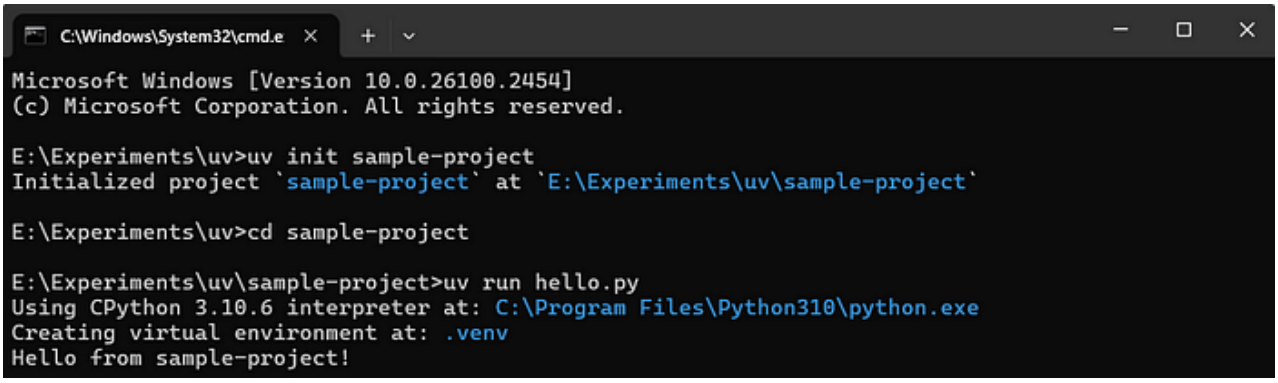
```
uv  sample-project
```

Navigate to the `sample-project` directory. **uv** initializes the project with essential files such as `app.py`, `requirements.txt`, `README.md`, and more.



Use the `run` command to execute the sample Python file. This process first creates the virtual environment folder and then runs the Python file.

```
uv run hello.
```



Install flask

Add Flask to your project dependencies.

```
uv  flask
```

Create the Flask Application

Create a new one and write the following code.

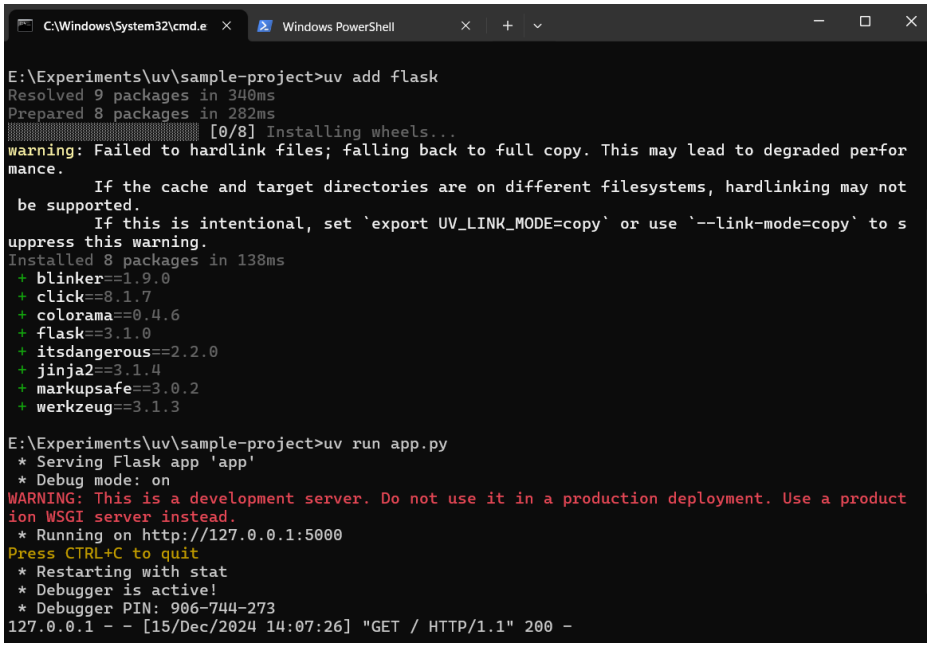
```
      __name__ == :     app.run(debug=)

 flask  Flask



app = Flask(__name__)



(): {: },
```

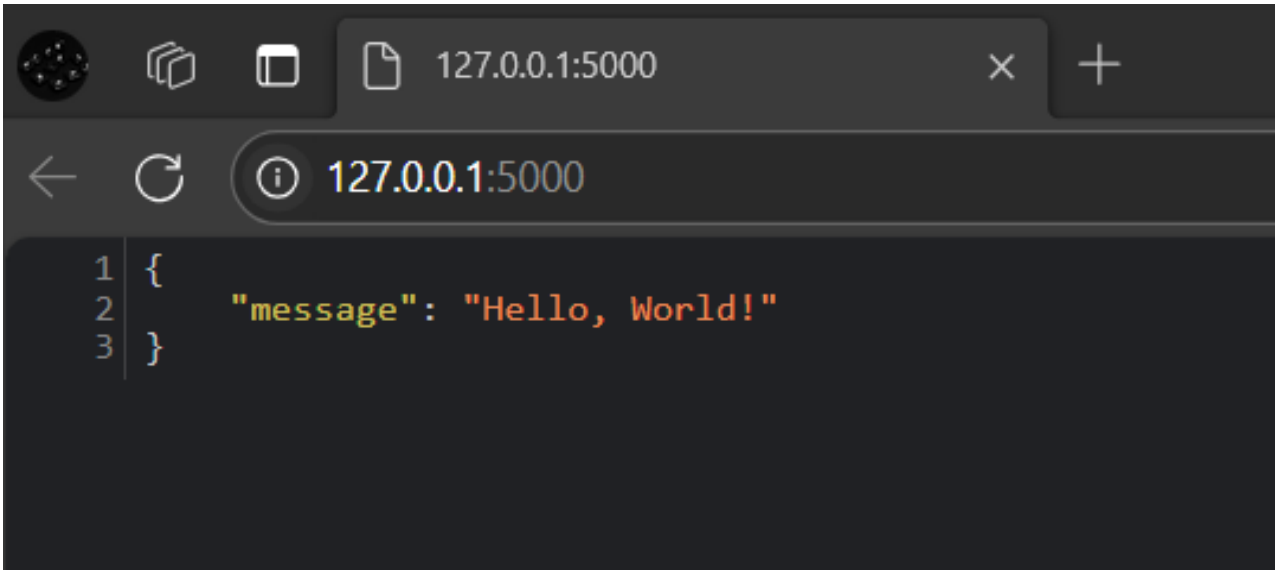
Run the app

Use the `uv run` command to execute the application.

```
uv run app
```

Open a browser or use a tool like `curl` or Postman to send a GET request.





Installing python with uv

Using uv to install Python is optional, as it works seamlessly with existing Python installations. However, if installing Python through uv is preferred, it can be done with a straightforward command:

```
uv python install
```

```
E:\Experiments\uv\sample-project>uv python install 3.12
Installed Python 3.12.8 in 12.68s
 + cpython-3.12.8-windows-x86_64-none
```

This approach is often more convenient and reliable compared to traditional methods, as it avoids the need for managing repositories or downloading installers. Simply execute the command, and the setup is ready to use.

Tools

CLI tools can be installed and used with the `uv` command. For example, the `huggingface_hub` tools can be installed to enable pulling and pushing files to Hugging Face repositories.

Use the following command to install huggingface_hub using uv.

```
uv tool install huggingface_hub
```

```
E:\Experiments\uv\sample-project>uv tool install huggingface_hub
Resolved 13 packages in 522ms
Prepared 12 packages in 771ms
Installed 13 packages in 72ms
 + certifi==2024.12.14
 + charset-normalizer==3.4.0
 + colorama==0.4.6
 + filelock==3.16.1
 + fsspec==2024.10.0
 + huggingface-hub==0.26.5
 + idna==3.10
 + packaging==24.2
 + pyyaml==6.0.2
 + requests==2.32.3
 + tqdm==4.67.1
 + typing-extensions==4.12.2
 + urllib3==2.2.3
Installed 1 executable: huggingface-cli.exe
```

The following command displays all the installed tools:

```
uv tool list
```

```
E:\Experiments\uv\sample-project>uv tool list
huggingface-hub v0.26.5
- huggingface-cli.exe
```

Cheatsheet

Here is a quick cheatsheet for performing common operations with uv:

	uv version	Explanation
Project dependency file	pyproject.toml	Base or core dependencies are specified in this file.
Project lock file	uv.lock	Derived dependencies are managed through a universal lockfile.
Installing Python	uv python install version or uv sync or uv run	uv will locate or install Python as needed when syncing or running code within the environment.
Creating project	uv init projectname	uv handles project dependencies and environments, offering features like lockfile management and workspace support, similar to pip.
Creating virtual environments	uv venv or uv sync or uv run	A virtual environment is automatically created by uv the first time you use it if one doesn't already exist.
Installing packages	uv pip install packagename or uv sync or uv run	uv installs all required packages into the environment whenever it is used.
Building dependencies	uv sync or uv run	The lockfile is rebuilt from dependencies each time uv is run.
Add a package	uv add	Adding a package will update pyproject.toml, uv.lock, and synchronize the environment.
Remove a package	uv remove	Removing a package will update pyproject.toml, uv.lock, and synchronize the environment.
Add a tool	uv tool install toolname	uv runs and installs command-line tools from Python packages, much like pipx.

Current Limitations

Even thoughuv offers a fast and efficient solution for Python package management, it has some limitations:

- Although uv supports a substantial portion of the `pip` interface, it does not yet cover the entire feature set. Some of these differences are intentional design choices, while others stem from uv still being in its early stages of development. For a detailed comparison, consult the pip compatibility guide.
- Like `pip-compile`, uv generates platform-specific `requirements.txt` files. This contrasts with tools such as Poetry and PDM, which create platform-agnostic `poetry.lock` and `pdm.lock` files. Consequently, 's `requirements.txt` files may lack portability across different platforms and Python versions.

Thanks for reading this article !!

Thanks Gowri M Bhatt for reviewing the content.

If you enjoyed this article, please click on the clap button 👏 and share to help others find it!

uv

docs.astral.sh