Understanding UV “Pictorially” for Python Development

Satya Komatineni, 8/4/25

Location: <https://github.com/SatyaKomatineni/articles-repo/tree/master/python/uv>

Audience: Developers going in to Python (for GenAI or otherwise)

Reading/Understanding time: Few hours (May save you a month or two of research)

Contents

[Abstract 3](#_Toc205287301)

[Audience and Benefits of reading this 3](#_Toc205287302)

[Benefits 3](#_Toc205287303)

[Research Journals 3](#_Toc205287304)

[Official Documentation References 3](#_Toc205287305)

[Background 3](#_Toc205287306)

[A modern learning map 4](#_Toc205287307)

[Primacy of Understanding in the age of “genai” 4](#_Toc205287308)

[UV, your Jeeves – The (indispensable) Agent 6](#_Toc205287309)

[Python Development: A set of Tools 6](#_Toc205287310)

[UV as a Single Tool 7](#_Toc205287311)

[Issue with libraries 7](#_Toc205287312)

[Python’s simple answer: Virtual Environments 8](#_Toc205287313)

[How does UV work? 9](#_Toc205287314)

[Basics: UV as an Agent 9](#_Toc205287315)

[UV Internals 10](#_Toc205287316)

[More UV Details 10](#_Toc205287317)

[How does UV improve upon .venv? 10](#_Toc205287318)

[Is the native python .venv approach not preferred? 11](#_Toc205287319)

[How big a conceptual difference between UV and Python .venv? 11](#_Toc205287320)

[What is common between UV and .Venv? 11](#_Toc205287321)

[What is different 11](#_Toc205287322)

[How do you use “uv” for your daily python work? 12](#_Toc205287323)

[Install python 12](#_Toc205287324)

[Create a repo -- to start your project with a root directory 13](#_Toc205287325)

[Write python files there, .py 13](#_Toc205287326)

[Install desired libraries needed by the .py files 13](#_Toc205287327)

[Run the python file 13](#_Toc205287328)

[Set the shell back to its original state if needed 13](#_Toc205287329)

[Look at what libs are installed 13](#_Toc205287330)

[How to uninstall and restart? 13](#_Toc205287331)

[References 13](#_Toc205287332)

[My Research Journal References 14](#_Toc205287333)

# Abstract

Understand uv architecturally to use in daily python development as you level up to GenAI using Python.

One line Lesson:  
  
Anything you like to do with python, ask “uv” to do it.

# Audience and Benefits of reading this

Few Benefits.

## Benefits

Good audience is an experienced programmer who wants to level up, to say GenAI, and hence needs to work with Python.

Very likely

1. They can benefit from understanding “uv” from the start
2. This should **shave a month or two** from your learning
3. Come to see UV as a friend and not a foe

## Research Journals

There are references here also to a couple of my research journals as I learned myself. These research journals have basic materials and questions I went through. Give them a quick read.

## Official Documentation References

I have also referenced here from the official uv documentation site. It is well laid out.

# Background

Lately folks wanting to use python for genai development run into instructions on the web that look like:

uv run <script.py>

uv venv

uv pip install <package>

uv pip uninstall <package>

uv pip sync

uv lock

uv pip freeze

uv cache dir

uv cache clear

uvx <tool>

Furthermore, one sees these commands arbitrarily in the middle of a longer article or instruction.

One doesn’t know what “uv” is or what it does. Even if that is understandable from the context, what is the whole story? When else could one use this “uv”? What are its limitations, its provisions? Not a simple incomplete explanation, as I said, what is the full story? Why am I just not using python tools?

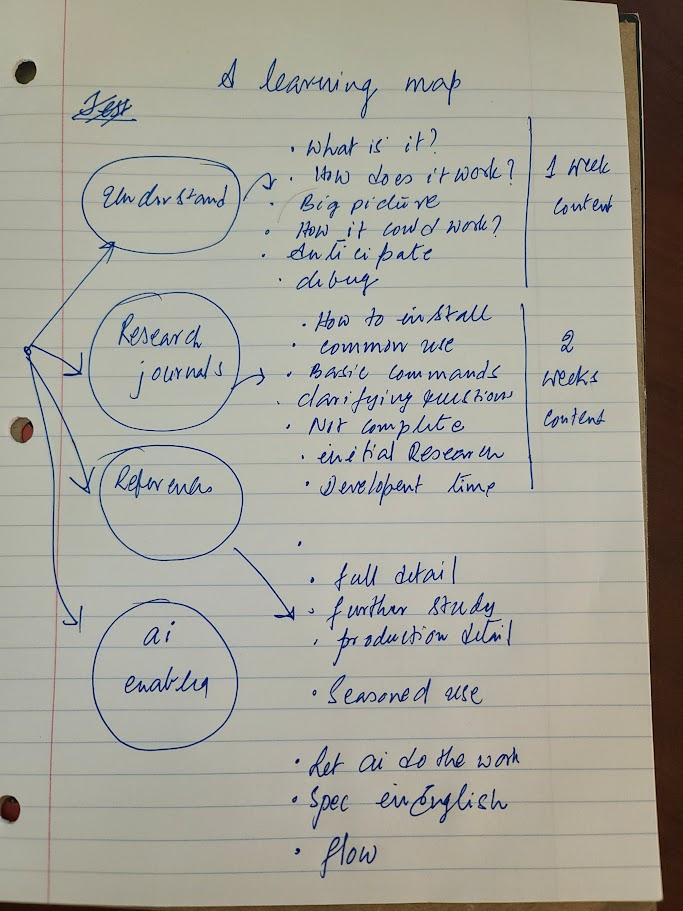
I have spent a lot of time with that dilemma.

Here is what I have found.

# A modern learning map

One document is not a good way to lead you along.

Here is a learning map that you can use.



This approach has 4 pathways for learning something new. They are

1. Understanding
2. Research Journals
3. References to learn the details of
4. Using “genai” to practice in flow

# Primacy of Understanding in the age of “genai”

Understanding fundamental principles gives you a lot more power because of genai.

Why? Because knowing those fundamental principles allows you to ask questions and specify requirements in English.

What this means is that the learning focus shifts to understanding, which is previously distributed between understanding, syntax, practice, and code.

**With this document you will have that understanding.**

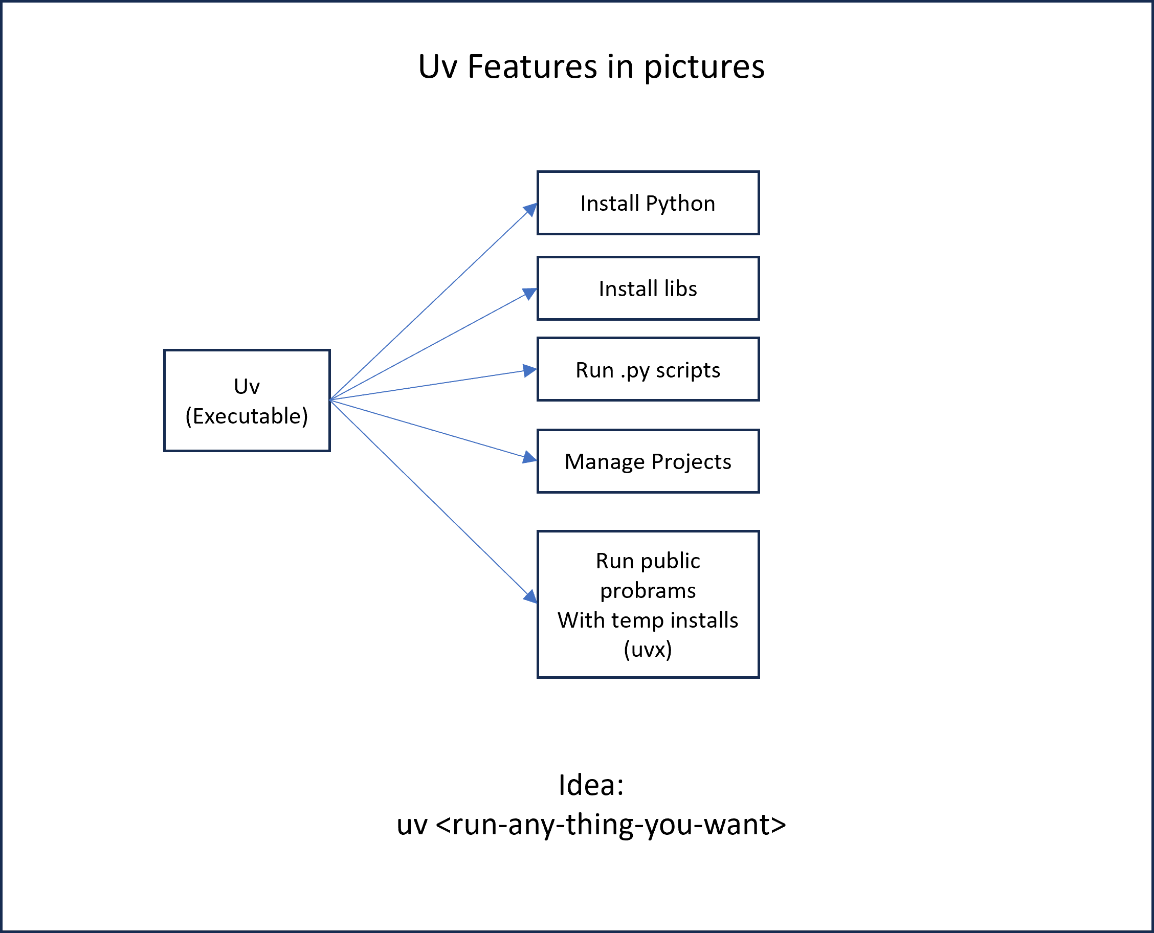
This document then lists references for the other pathways.

# UV, your Jeeves – The (indispensable) Agent

Consider the following diagram.

Anything you like to do with python, ask “uv” to do it.

“uv” is a command line tool developed by an independent company called “Astra”.



## Python Development: A set of Tools

Python has several command line tools. They do things like

1. Install and track libraries and versions
2. Linting
3. Formatters
4. Manage dependencies

Here are some examples of these tools

1. pip: install, upgrade, or uninstall Python packages from PyPI
2. poetry: Manage dependencies, virtual environments, versioning, and publishing via pyproject.toml
3. pipenv: combine pip + virtualenv; track deps in Pipfile / Pipfile.lock
4. hatch: all-in-one project manager for envs, builds, releases, and versioning
5. tox: automate testing across multiple Python versions and environment matrices
6. pytest: discover and run tests with rich assertions and plugin ecosystem
7. black: opinionated automatic code formatter
8. ruff: ultra-fast linter and formatter (covers flake8, isort, etc.)
9. flake8: pluggable linter for PEP 8 style and simple static analysis
10. mypy: static type checker enforcing type annotations
11. coverage: measure and report code-coverage percentages

## UV as a Single Tool

“uv” is one such tool but hoping to be the most frequently used and a one-stop-shop for “All needs”.

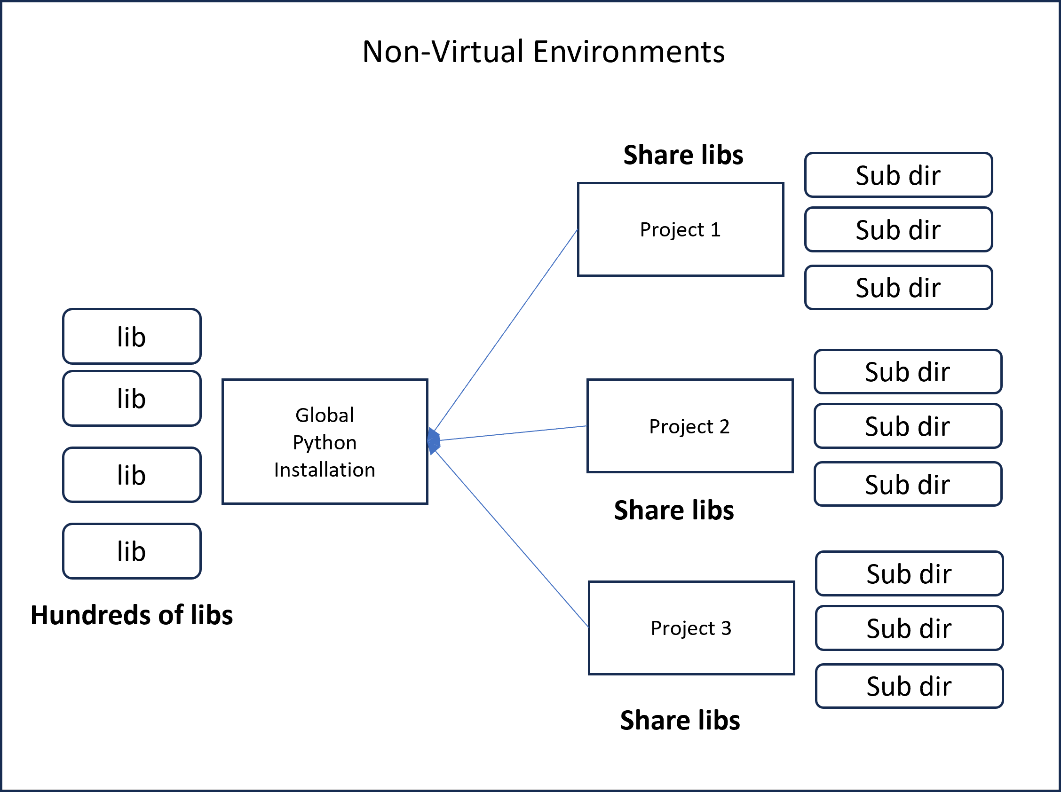
However, see the “uv” home page to see what is possible currently.

I will explain “uv as a tool” with the most obvious use for a tool in a python development environment, namely “**python version and library dependency management**”. Because this need is “intimately tied to” running python scripts. Why would you write a script, if you don’t want to run it?

# Issue with libraries

This is not specific to python. Most modern languages where “reuse” is so common library development is heavily “decentralized”. The libraries tend to be numerous, and one depends on another ad-infinitum.

Consider this picture of out of the box python development 101. A single python environment on installation.



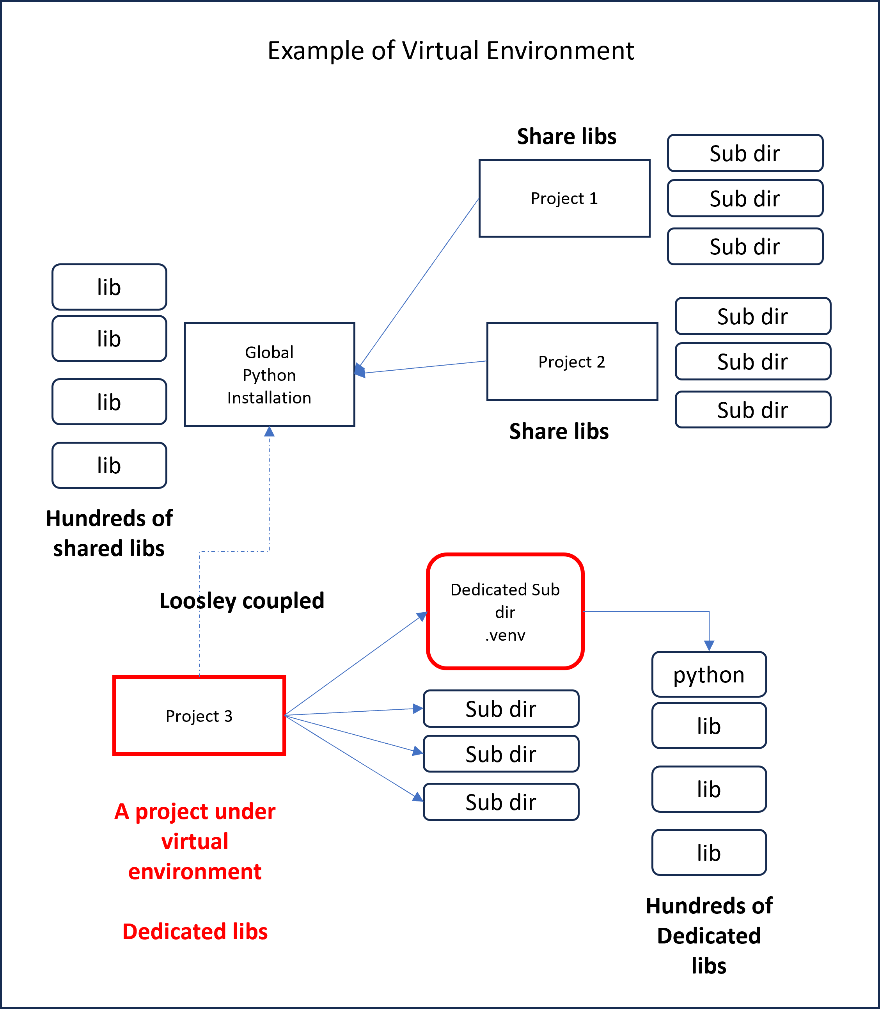
Here the libraries are global.

The projects 1, 2, and 3 have their own code sub directories but share the libraries and global python. They will step on each other.

# Python’s simple answer: Virtual Environments

Not just python, but most modern languages solve this problem similarly.

Here is how this design looks like.



Here Project 1 and 2 are using the global libraries and python.

Whereas Project 3 uses a virtual environment where its use of libraries are isolated to a sub directory in the project root.

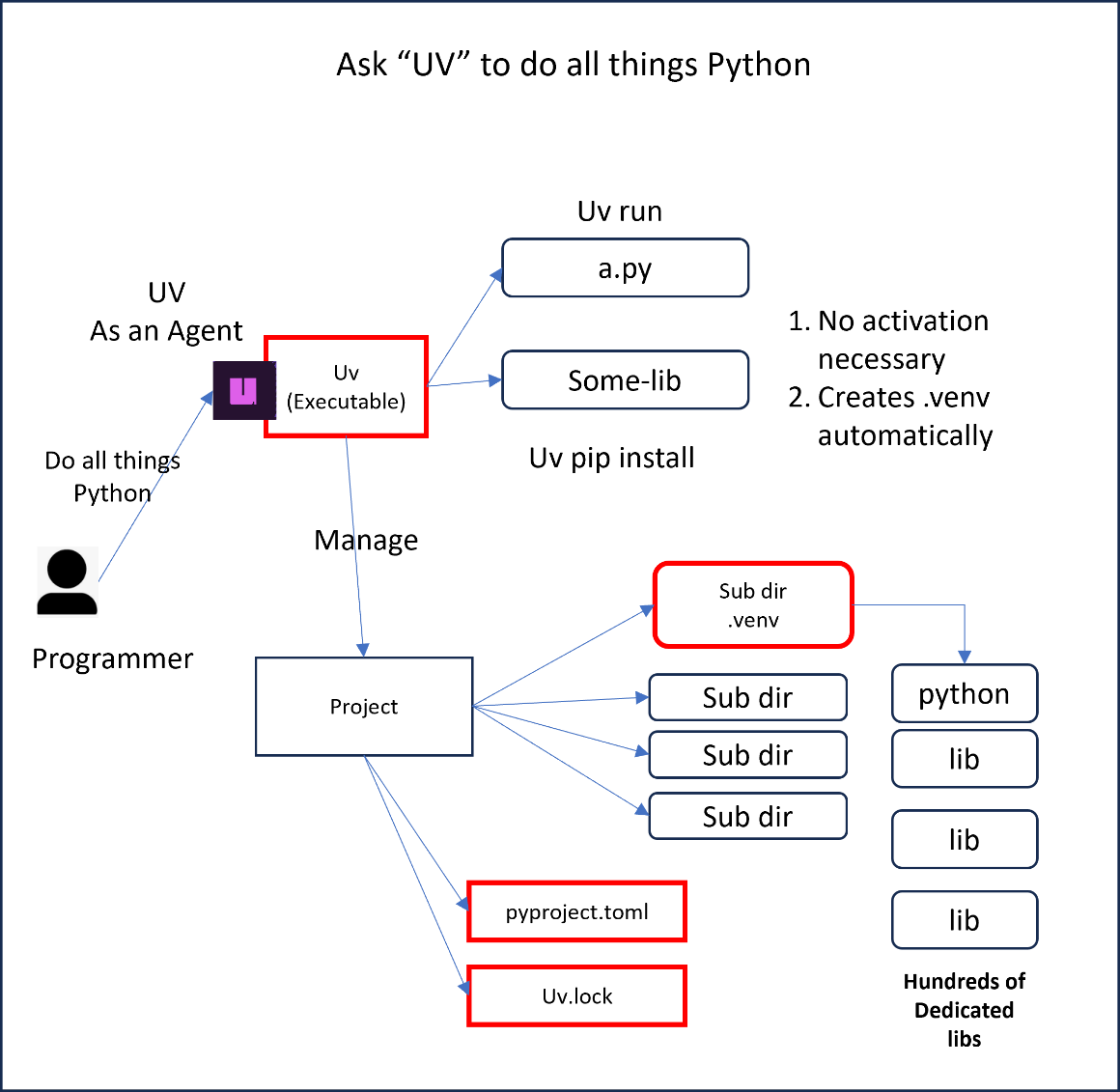
Here is how it works:

1. Natively Python answers this with a sub directory called .venv under the project root.
2. It relies on adjusting the command line paths and settings to store all newly installed libraries into that venv directory
3. It provides a batch file called .venv/activate.sh to set up environment variables in the shell
4. Once you activate it, all subsequent python executions will use the venv for path and library resolution.
5. The python library install tool “pip install”, will install the subsequent libs into the venv sub directory.
6. Tools like vscode also recognizes .venv and provides necessary type checking and library resolution during development.
7. When you no longer want to use the specialized environment in the shell you call deactivate or just close the shell.

See the references section for a full-length article on how the .venv works.

# How does UV work?

Start with a diagram to contrast the python .venv diagram before.



## Basics: UV as an Agent

Let me annotate the diagram first.

1. You, the programmer has an agent, the UV command line executable that you have installed
2. You can ask it to do a variety of things like install libs, or run python programs by resolving libs correctly

## UV Internals

So how does UV do its thing, from the picture:

1. UV uses a project file in the root called project.toml.
2. UV creates it if it doesn’t exist
3. UV also relies on the same .venv sub directory for isolation (It may use other schemes, but it seems to work with .venv by default). This also helps the tool chains like Vscode and their affinity to .venv
4. Every request to install libs, uv will place them in project.toml and place them in the .venv
5. If those libraries have additional dependencies to download, they are recorded in the uv.lock file.
6. When you run a script file anywhere in that project hierarchy it runs that python file with correct paths and dependencies.
7. You do not need to adjust the shell using activate

## More UV Details

1. Practically speaking you run every python script using "uv script args"
2. This includes pip install, a.py, or any script
3. You are telling "uv", like an agent, hey, run this python code for me!!
4. uv then decides to run that code using the python interpreter by picking the right path and the right libraires using a config file for that project context.
5. This project context is maintained in the root of the repo as "project.toml"
6. This file contains the project dependencies for all the script files in that project
7. uv will install all the dependencies from project.toml into the .venv sub directory, by creating one if it doesn't exist, the venv directory itself
8. It does not need to set the activation
9. It does it programmatically at run time just for that script
10. Your shell is untouched
11. Further it creates uv lock file that contains all the long list of resolved dependencies
12. this file can be checked in along with project.toml into github
13. the venv then can be recreated just using the project.toml spec
14. In other words the meta data about your project is captured in project.toml

## How does UV improve upon .venv?

1. At its core, you are temporarily changing the shell to simulate a virtual environment
2. There is no explicit registry that is managed to recreate the library environment when it is transported.
3. You typically do that by doing a pip freeze to list the total installed "libraries" in venv and check it in, usually something like a "requirements.txt", a list of all libraries
4. That is a manual step
5. Also, you have to remember to activate and deactivate the shells
6. It does not rely on a project definition file like your project.toml that explicitly controls how things and dependencies are setup.
7. In other words, you can have two projects that can share that venv and there isn't a way to know which shares what
8. So, the default .venv relies on the programmer to manage that meta data

## Is the native python .venv approach not preferred?

1. It is quite workable.
2. it is simple enough.
3. Most tools like vscode assume venv and python’s approach of running python programs. They are not yet fully programmed for "uv" like behavior
4. It works quite well
5. The need to go to "uv" is a small enhancement (or big if a particular aspect of uv makes your life simpler)

## How big a conceptual difference between UV and Python .venv?

Not big, a small adjustment if you understand how virtual environments and "uv" works.

## What is common between UV and .Venv?

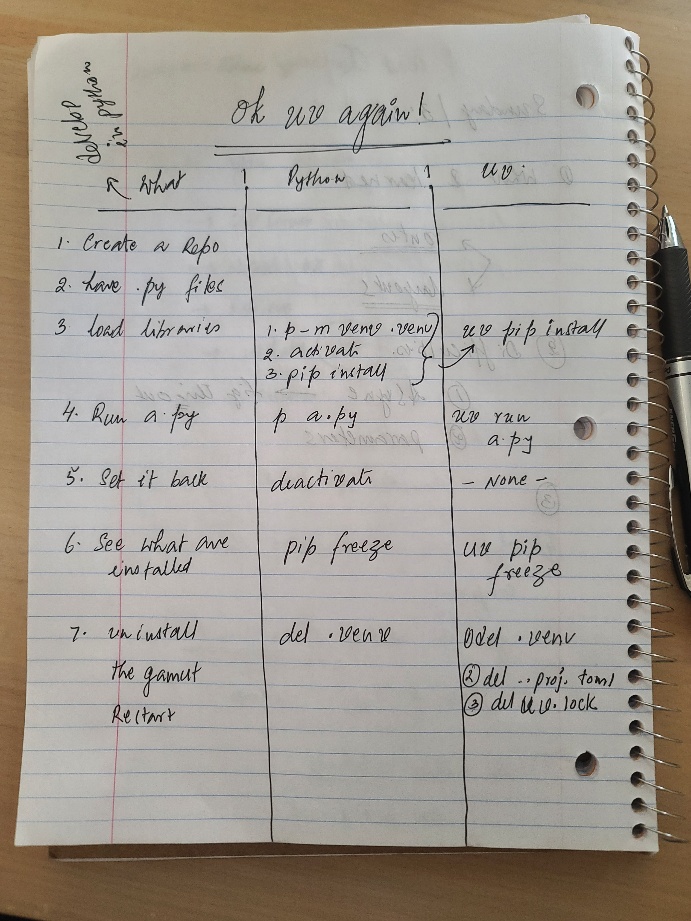
1. Both maintain .venv
2. Both resolve libraries correctly
3. Both help in running the python scripts
4. You can use one or the other

## What is different

1. uv uses project.toml to do this
2. uv also uses uv lock file to keep a record of installed dependencies
3. Every command or (uv enabled) tool is run with "uv" as the prefix or the main command

# How do you use “uv” for your daily python work?

I have a quick pencil drawing of when you bring in “uv” as you start working with python projects.



Consider the steps one would go through for a python project, be it first, or next.

1. Install python
2. Create a repo -- to start your project with a root directory
3. Write python files there, .py
4. Install desired libraries needed by the .py files
5. Run the python file
6. Set the shell back to its original state if needed
7. Look at what libs are installed
8. How to uninstall and restart?

Let’s take these step by step.

## Install python

Although I have not tried, I would explore if I can do this with “uv” itself! The official documentation seems to indicate this.

It is worth noting that and experimenting with.

## Create a repo -- to start your project with a root directory

Of course, you want to use github to start your project with a nice repo.

You can leave it as it is or use “uv init” to create the project.toml and other things that uv uses to manage further interaction.

## Write python files there, .py

Use vscode to author your python files.

## Install desired libraries needed by the .py files

You will use “uv pip install” as opposed to “pip install”.  
  
This allows you to do this with out activating environments. Also, it records the results in uv.lock so that it is faster next time to recreate the environment. In case of pip install no such recording takes place.

## Run the python file

Here also you use “uv” to run the python file.   
You don’t need to run activate if you are using uv.

Uv will resolve lib differences.

## Set the shell back to its original state if needed

In case of uv, the shell is untouched. So you don’t need to deactivate the shell.

## Look at what libs are installed

You can use uv to do this.

## How to uninstall and restart?

If something goes wrong, just delete the following

1. .venv
2. Project.toml
3. Uv.lock

# References

These are some references to understand virtual environments, common uv commands, official docs to start using uv in for your production work.

|  |  |  |
| --- | --- | --- |
| 1 | [Common UV Commands](https://github.com/SatyaKomatineni/articles-repo/blob/master/python/uv/uv-new-findings.md#everyday-uv-commands) | . Most used commands  . See this list to get a quick sense of what is possible with uv |
| 2 | [My Github article: Understanding Virtual Environments](https://github.com/SatyaKomatineni/articles-repo/blob/master/python/uv/python-virtual-environment-setup.md) | . What are virtual environments  . in pictures  . How to set them up  . How to activate them and deactivate them  . pip install  . Look up at what is installed  . References  . General understanding of virtual environments |
| 3 | February 15, 2024,  uv: Python packaging in Rust | <https://astral.sh/blog/uv>  it has   1. package manager in Rust 2. A replacement for Rye 3. Astral: Developer tools for the Python like Ruff, Python linter and formatter. 4. Single binary 5. Expected to start to look like "Cargo for Rust |
| 4 | Astral Official UV Docs | <https://docs.astral.sh/uv/getting-started/> |
| 5 | Features of UV.  Give it a quick read as it goes into “Other things” you can do with “uv” such as installing python itself.  Experiment with it. | <https://docs.astral.sh/uv/getting-started/features/#projects>  it has   1. How to install python itself :) 2. Run scripts 3. Manage project.toml 4. run global packages without installing (uvx) 5. Full backward compatibility for venv 6. .... 7. All in all it is beautifully done |
| 6 |  |  |

# My Research Journal References

These references have things like how to install, how to use, common commands, faqs, etc. They are progressively better. They are what they are, running journals. I go back to them as a first reference, and then I step back to official docs if I can’t find something there.

|  |  |  |
| --- | --- | --- |
| 1 | First attempt  <https://github.com/SatyaKomatineni/articles-repo/blob/master/python/uv/using-uv-for-python-development.md> | it has   1. What is uv 2. how to install 3. How to use it 4. More focus on using without venv 5. Speaks of some relationships with venv 6. ...... 7. Note: it is incomplete 8. Especially using venv naturally 9. Later article talks about this |
| 2 | Second attempt  <https://github.com/SatyaKomatineni/articles-repo/blob/master/python/uv/uv-new-findings.md> | It has   1. Focuses on compatibility with venv 2. Common useful commands 3. Brings it closed to how you use uv 4. As always references to public uv material 5. ..... 6. Still not a comprehensive one nor a simpler one 7. A good one will combine both |