**Virtual Environment (Youtube video – Corey Shafar)**

* It is a way that you can separate different python environments for different projects
* If you have multiple projects and they all rely on a single package (ex Flask) and each of these projects may be a using a different version of that package. It would be better if each of those projects had an isolated environment where they have only the dependencies and packages they need and the specific versions they need. Venv allows us to make the different python environments.

**Why the Need for Virtual Environments (realpython.com)**

It’s important to know this because, by default, every project on your system will use these same directories to store and retrieve site packages (third party libraries). At first glance, this may not seem like a big deal, and it isn’t really, for system packages (packages that are part of the standard Python library), but it does matter for site packages.

Consider the following scenario where you have two projects: ProjectA and ProjectB, both of which have a dependency on the same library, ProjectC. The problem becomes apparent when we start requiring different versions of ProjectC. Maybe ProjectA needs v1.0.0, while ProjectB requires the newer v2.0.0, for example.

This is a real problem for Python since it can’t differentiate between versions in the site-packages directory. So both v1.0.0 and v2.0.0 would reside in the same directory with the same name:

/System/Library/Frameworks/Python.framework/Versions/3.5/Extras/lib/python/ProjectC

Since projects are stored according to just their name, there is no differentiation between versions. Thus, both projects, ProjectA and ProjectB, would be required to use the same version, which is unacceptable in many cases.

This is where virtual environments and the virtualenv/venv tools come into play…

**What Is a Virtual Environment? (realpython.com)**

At its core, the main purpose of Python virtual environments is to create an isolated environment for Python projects. This means that each project can have its own dependencies, regardless of what dependencies every other project has.

In our little example above, we’d just need to create a separate virtual environment for both ProjectA and ProjectB, and we’d be good to go. Each environment, in turn, would be able to depend on whatever version of ProjectC they choose, independent of the other.

The great thing about this is that there are no limits to the number of environments you can have since they’re just directories containing a few scripts. Plus, they’re easily created using the virtualenv or pyenv command line tools.