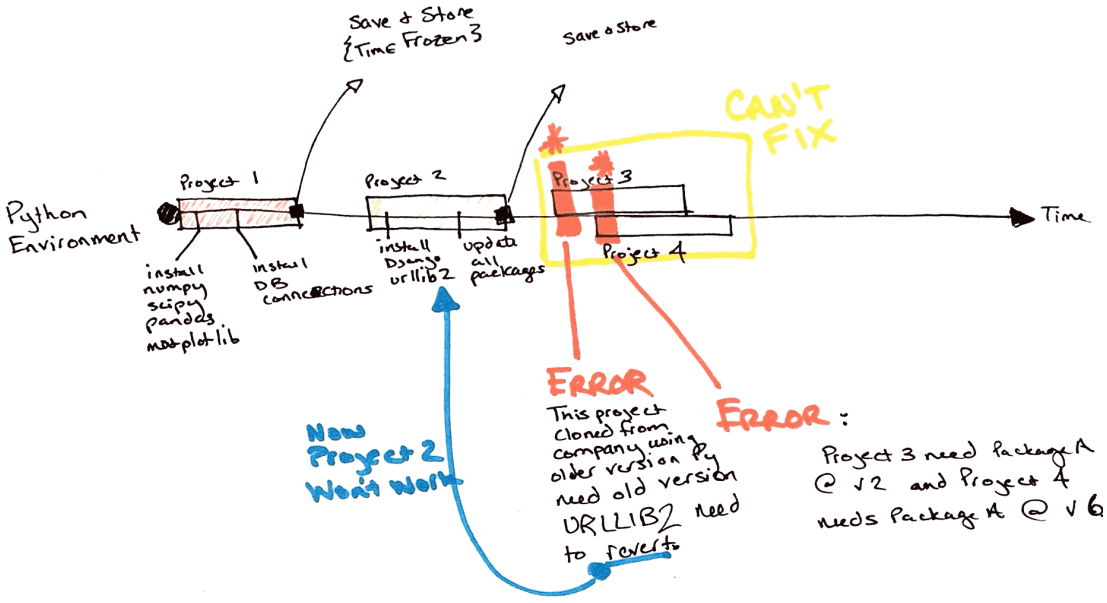
PUG: Python Virtual Environment

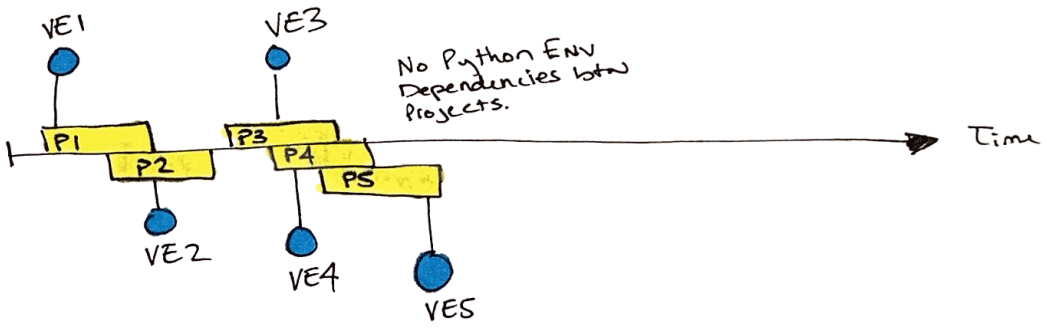
Why use a VE?

By default your computer will download and install all packages (3rd party libraries) to the same directories to store and retrieve them. This means that for every individual and specific project you build you will be altering the entire python environment on your computer. Over time your Python Environment will change as each project requires different packages and alters dependencies through updates, upgrades, or replacement. A problem then arises, all your past projects depend on a snapshot state of your Python Environment that is always changing.

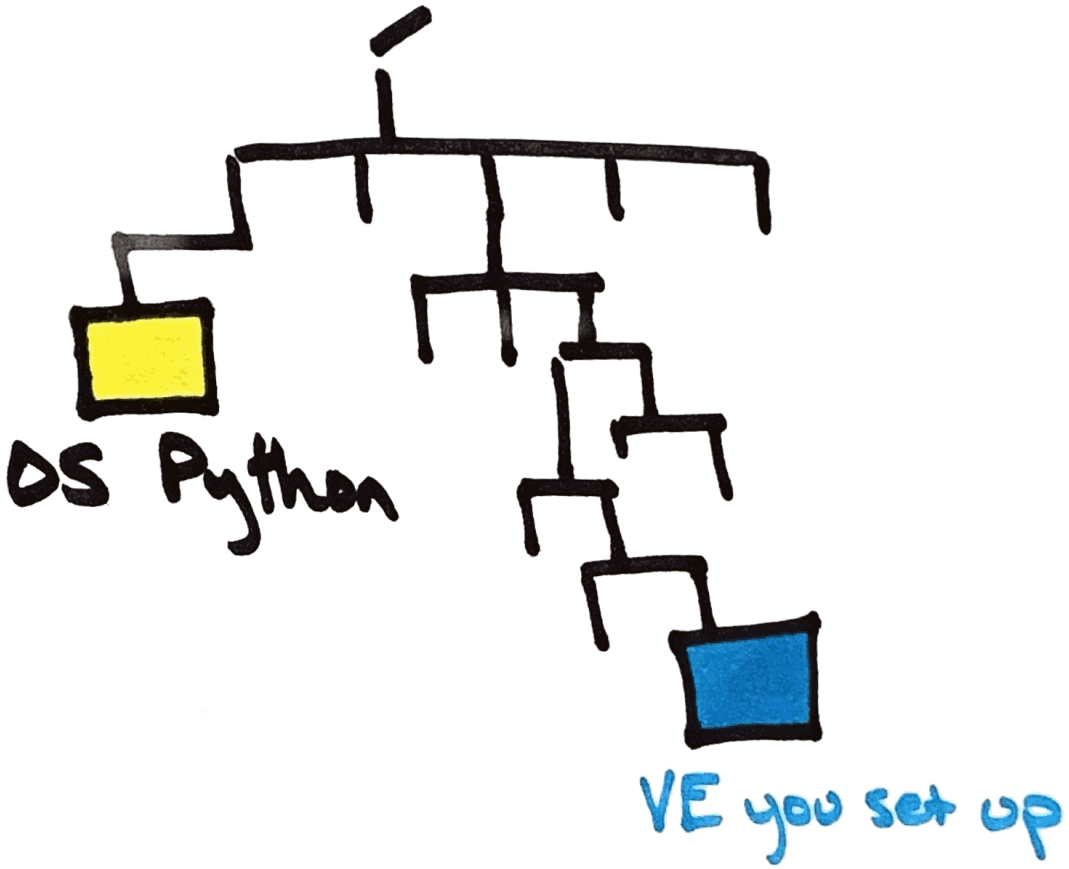


What is a VE?

An isolated environment for Python. This means that each project can have its own dependencies, regardless of what dependencies every other project has. So for the illustration above one could have made environments Projects 1, 2, 3, and 4. Therefore the errors shown in orange would not occur because each Python Environment is unique to the project and its requirements.



What's Happening?



When you tell your computer to use Python by default it goes to the one associate with the System. For a Mac Python is automatically installed. In our case you use a VE and it directs your computer to use the Python instance stored elsewhere (Blue box). In this diagram it shows two Python instances, but you could have many.

Each Python instance has a directory structure similar to below.

```
├── bin
│   ├── activate
│   ├── activate.csh
│   ├── activate.fish
│   ├── easy_install
│   ├── easy_install-3.5
│   ├── pip
│   ├── pip3
│   ├── pip3.5
│   ├── python -> python3.5
│   ├── python3 -> python3.5
│   └── python3.5 ->
/Library/Frameworks/Python.framework/Versions/3.5/bin/python3.5
├── include
├── lib
│   └── python3.5
│       └── site-packages
└── pyvenv.cfg
```

Where the following folders mean:

- **bin**: files that interact with the virtual environment
- **include**: C headers that compile the Python packages
- **lib**: a copy of the Python version along with a site-packages folder where each dependency is installed

How's This Happening?

When Python is starting up, it looks at the path of its binary. In a virtual environment, it is actually just a copy of, or symlink* to, your system's Python binary. It then sets the location of sys.prefix and sys.exec_prefix based on this location, omitting the bin portion of the path.

The path located in sys.prefix is then used for locating the site-packages directory by searching the relative path lib/pythonX.X/site-packages/, where X.X is the version of Python you're using.

The path is stored in the sys.path array, which contains all of the locations where a package

can reside.

**is a term for any file that contains a reference to another file or directory in the form of an absolute or relative path and that affects pathname resolution*

System Default Python:

```
$ echo $PATH
/usr/local/bin:/usr/bin:/bin:/usr/sbin:/sbin:
```

Your Virtual Environment Python

```
$ source env/bin/activate
(env) $ echo $PATH
/Users/jesse/python-virtual-
environments/env/bin:/usr/local/bin:/usr/bin:/bin:/usr/sbin:/sbin:
```

Much of the content above was taken from: Python Virtual Environments: A Primer. by Real Python

---

Installation and Setup

This setup is based on macOS High Seirra.

Make sure Homebrew is installed.

PYTHON

Install latest version of Python using Homebrew

Why bother, you ask, when Apple includes Python along with macOS? Here are some reasons:

- When using the bundled Python, macOS updates can nuke your Python packages, forcing you to re-install them.
- As new versions of Python are released, the Python bundled with macOS will become out-of-date. Homebrew always has the most recent version.
- Apple has made significant changes to its bundled Python, potentially resulting in hidden bugs.
- Homebrew's Python includes the latest versions of Pip and Setuptools(Python package management tools)

Use the following command to install Python 3.x via Homebrew:

```
> brew install python
```

Install Virtualenv

```
> pip3 install virtualenv
```

Create some directories to store our projects, virtual environments, and Pip configuration file, respectively:

```
$ mkdir -p ~/Projects ~/Virtualenvs ~/Library/Application\ Support/pip
```

We'll then open Pip's configuration file (which may be created if it doesn't exist yet)…

```
$ vim ~/Library/Application\ Support/pip/pip.conf
```

… and add some lines to it:

```
[install]
require-virtualenv = true

[uninstall]
require-virtualenv = true
```

Now we have Virtualenv installed and ready to create new virtual environments, which we will store in ~/Virtualenvs. New virtual environments can be created via:

```
$ cd ~/Virtualenvs
$ virtualenv foobar
```

If you have both Python 2.x and 3.x and want to create a Python 3.x virtualenv:

```
$ virtualenv -p python3 foobar-py3
```

Restricting Pip to virtual environments

What happens if we think we are working in an active virtual environment, but there actually is no virtual environment active, and we install something via

```
pip3 install foobar
```

Well, in that case the

```
foobar package
```

gets installed into our global site-packages, defeating the purpose of our virtual environment isolation.

This won't happen since in pip.conf above we required virtualenv for an install and uninstall.

Wait, I need to install a Global!

We can temporarily turn off this restriction by defining a new function in ~/.bashrc:

```
gpip(){
   PIP_REQUIRE_VIRTUALENV="0" pip3 "$@"
}
```

You must run for the change to take effect.

```
source ~/.bash_profile
```

If in the future we want to upgrade our global packages, the above function enables us to do so via:

```
gpip install --upgrade pip setuptools wheel virtualenv
```

My First Virtual Environment

Creating virtual environments

Let's create a virtual environment for ***Pelican***, a Python-based static site generator:

```
$ cd ~/Virtualenvs
$ virtualenv pelican
```

Change to the new environment and activate it via:

```
$ cd pelican
$ source bin/activate
```

To install Pelican into the virtual environment, we'll use pip:

```
$ pip3 install pelican markdown
```

Once you are done and want to stop this:

```
$ deactivate
```

Content for installation and making a VE from: Python Development Environment on macOS High Sierra

Further Things to Consider

virtualenvwrapper

Managing Virtual Environments With virtualenvwrapper

A few of the more useful features of virtualenvwrapper are that it:

- Organizes all of your virtual environments in one location
- Provides methods to help you easily create, delete, and copy environments
- Provides a single command to switch between environments

Using Different Versions of Python

Unlike the old virtualenv tool, pyvenv doesn't support creating environments with arbitrary versions of Python, which means you're stuck using the default Python 3 installation for all of the environments you create. While you can upgrade an environment to the latest system version of Python (via the --upgrade option), if it changes, you still can't actually specify a particular version.

This is where pyenv comes in to play - Despite the similarity in names (pyvenv vs pyenv), pyenv is different in that its focus is to help you switch between Python versions on a system-level as well as a project-level. While the purpose of pyvenv is to separate out modules, the purpose of pyenv is to separate Python versions.

SUBLIME3 and VE

https://packagecontrol.io/packages/Virtualenv

Features

- Reusable build system. Execute code with a virtualenv without editing your paths manually.
- Virtualenv search. Finds virtualenvs in the open folders or anywhere in your system.
- Activation/Deactivation. Select or disable the current virtualenv easily.
- Create and delete virtualenvs. With target python selection. Supports both the standard virtualenv package and the built-in venv module from Python 3.3.
- Integration with other packages. SublimeREPL.

Point it to your VE Directories.

References

- https://realpython.com/setting-up-sublime-text-3-for-full-stack-python-development/
- https://www.granneman.com/webdev/editors/sublime-text/packages/how-to-install-and-use-package-control
- https://realpython.com/python-virtual-environments-a-primer/
- https://hackercodex.com/guide/mac-development-configuration/
- https://hackercodex.com/guide/python-development-environment-on-mac-osx/