Commands that helped me:

Brew install python@3.10

brew unlink python@3.10

brew rmtree python@3.10

ln -s ~/.pyenv/versions/3.10.9 $(brew --cellar python)/3.10.9

or

ln -s ~/.pyenv/versions/3.10.9 $(brew --cellar python@3.10)/3.10.9

and

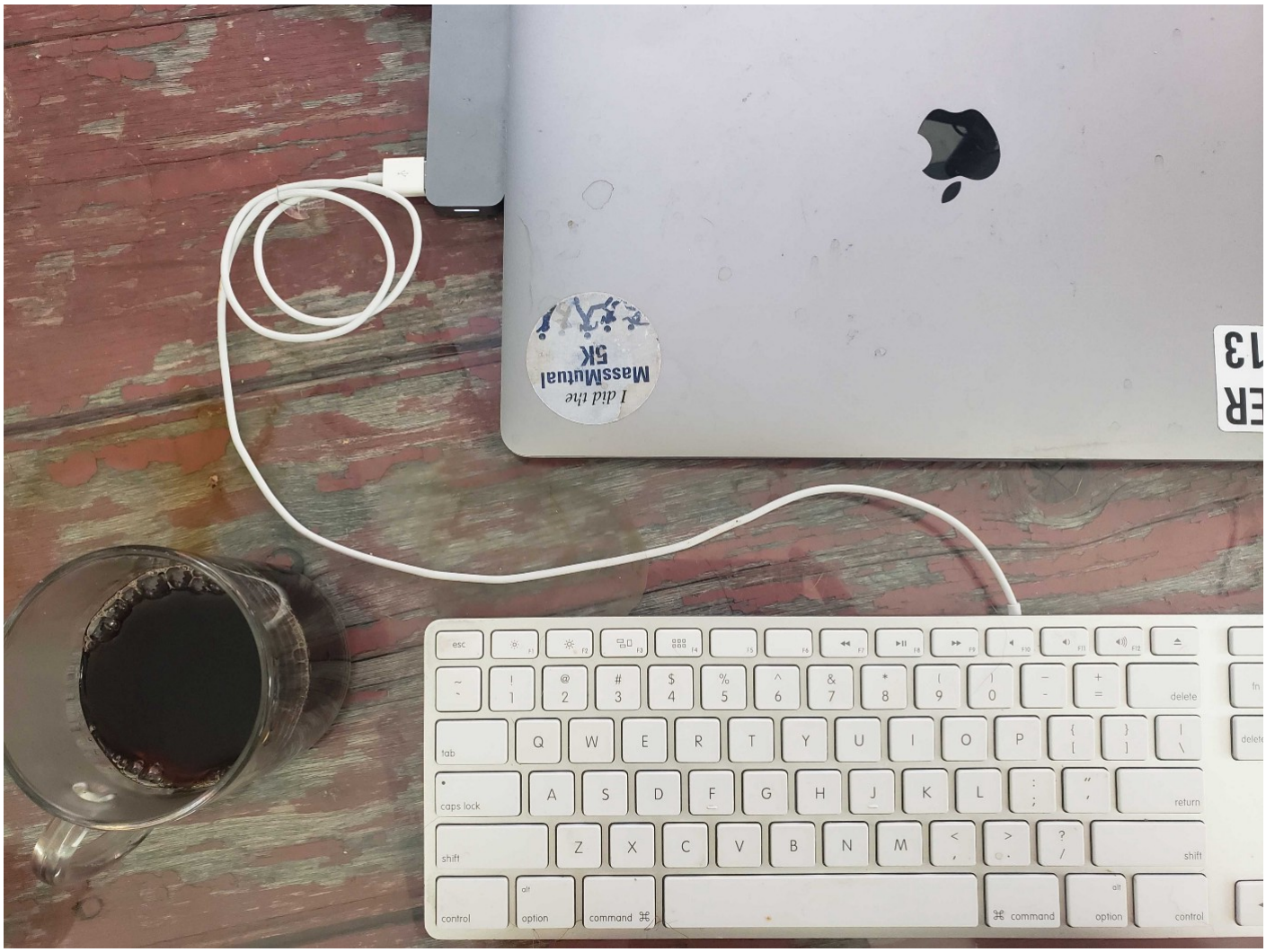
ln -s ~/.pyenv/versions/3.11.1 $(brew --cellar python@3.11)/3.11.1

But the thing is, that it must be installed the latest minor version of python for every major version of python via pyenv

Homebrew and Pyenv Python Playing Pleasantly in Partnership

https://towardsdatascience.com/homebrew-and-pyenv-python-playing-pleasantly-in-partnership-3a342d86319b

How to use pyenv's Python to satisfy Homebrew dependencies so only pyenv manages Python 3 installations on your machine



Next, I'll clean my desk! Image by Author.

Like many data scientists and Python developers before me, I've given up on managi ng my own Python builds and turned to pyenv (links to chronological posts in Towards Data Science). At various points in time, I built Python versions from source myself or used the prebuilt Framework installers on OSX, all while managing the links

in `/usr/local/bin` by hand. With either of those methods, I had no problem spinning virtual environments from any of the installed versions, but most recently my PATH and me got mixed up on whether we were using Homebrew's `python@3.8` or the Framework version I had installed. Enter pyenv. If you're looking for how to set up pyenv, there are no shortage of articles or blog posts on how to install it, however, I'd recommend sticking to the official documentation. As some of these articles mention, there are many ways to get Python on your OSX systems:

- There's the built in Python 2.7.x (note: Python 2 is deprecated as of Dec 2019).
- Homebrew will provide a Python 3 build to satisfy it's own dependencies.
- You could have the Framework versions.
- You could have built it from source.
- Or as many would recommend, you can use pyenv to manage the versions.

Your system needs the built-in, and Homebrew is really only going to keep a single version of Python 3 for you, so you need to pick between the last three if you want to manage specific versions of Python 2 or 3. I omitted Anaconda since I haven't needed it or taken the time to really understand how it works, but I have heard success stories for Windows users. Here, we'll choose pyenv to manage python versions, but in doing so, we'll initially have multiple versions of both Python 2 and Python 3 on our system (those from pyenv in addition to the builtin version 2 and homebrew version 3). Again, there's nothing we can do with the Python 2 builtin, OSX needs it. **For Python 3, we can have Homebrew *use* one of pyenv's Python versions to eliminate that source of redundancy.** Before we get started, it's worth noting that you're unlikely to experience conflicts between pyenv and homebrew if you use your `PATH` like pyenv recommends, and Homebrew warns against it; you're wading into the territory of relying on your own understanding to support this use case (as Homebrew states in their post).

Use pyenv's Python to satisfy Homebrew dependencies

These Stack Overflow answers have the right idea of using symbolic links (symlinks), but from what I can tell they don't get the links quite right. Let's get them right!

If you already have any traces of Homebrew's python3, first get rid of them (`brew unlink python3` and `brew uninstall python3`). If you already had python@3.8 from Homebrew, when uninstalling it, note the packages that depended on it (for example, `ffmpeg`), and reinstall them after.

As of this post, Homebrew is expected Python 3.8.6 for it's python@3.8, so first install that version with pyenv following their documentation:

```
pyenv install 3.8.6
```

That will put (by default) the actual Python install in

```
~/.pyenv/versions/3.8.6
```

Now we just need to add one link, and then let brew do the rest. I'll use full paths here, so you can run this from anywhere (and remember to read `ln -s ... ...` in your head as "link -symbolic [target] [linkname]"):

```
ln -s ~/.pyenv/versions/3.8.6 $(brew --cellar python)/3.8.6
```

With the `-f` flag, you could have omitted the trailing `/3.8.6`, as `ln` will use the name of the target. To be as explicit as possible on the link, you should have

```
ln -s ~/.pyenv/versions/3.8.6 /usr/local/Cellar/python@3.8/3.8.6
```

Here's what the result should look like:

```
➜  ~ ll $(brew --cellar python)
total 0
lrwxr-xr-x  1 [my username]  admin    36B Oct 14 16:52 3.8.6 ->
/Users/[my username]/.pyenv/versions/3.8.6
```

Finally, let Homebrew manage the rest of necessary linking:

```
brew link python3
```

This article was inspired by my own exploration of how to make it work and that the answer to this question on Stack Overflow went unanswered (I answered it, I think!).