**Managing Packages**

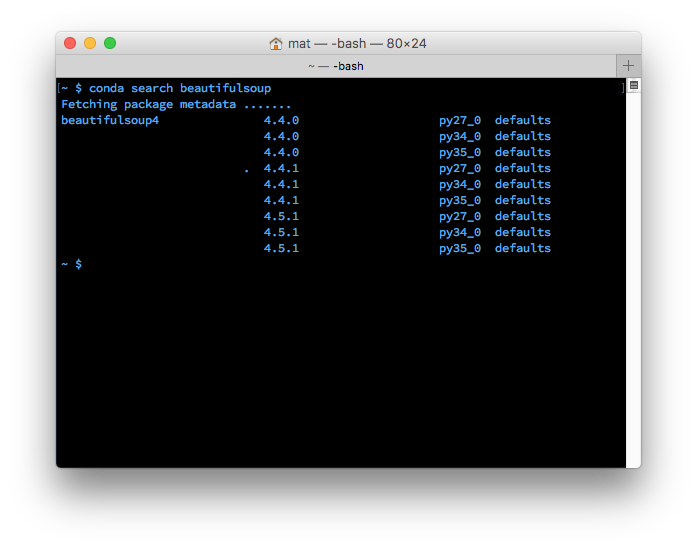
Once you have Anaconda installed, managing packages is fairly straightforward. To install a package, type conda install package\_name in your terminal. For example, to install numpy, type conda install numpy.

You can install multiple packages at the same time. Something like conda install numpy scipy pandaswill install all those packages simultaneously. It's also possible to specify which version of a package you want by adding the version number such as conda install numpy=1.10.

Conda also automatically installs dependencies for you. For example scipy depends on numpy, it uses and requires numpy. If you install just scipy (conda install scipy), Conda will also install numpy if it isn't already installed.

Most of the commands are pretty intuitive. To uninstall, use conda remove package\_name. To update a package conda update package\_name. If you want to update all packages in an environment, which is often useful, use conda update --all. And finally, to list installed packages, it's conda list which you've seen before.

If you don't know the exact name of the package you're looking for, you can try searching with conda search search\_term. For example, I know I want to install [**Beautiful Soup**](https://www.crummy.com/software/BeautifulSoup/), but I'm not sure of the exact package name. So, I try conda search beautifulsoup.

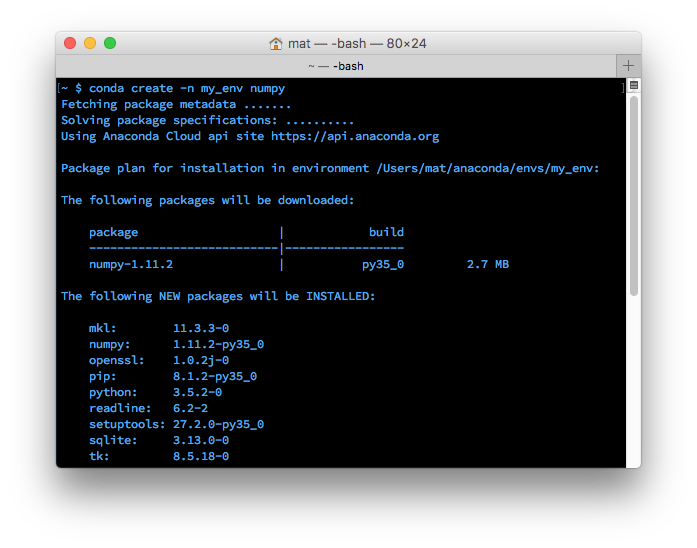


Searching for beautifulsoup

It returns a list of the Beautiful Soup packages available with the appropriate package name, beautifulsoup4.

# Managing environments

As I mentioned before, conda can be used to create environments to isolate your projects. To create an environment, use conda create -n env\_name list of packages in your terminal. Here -n env\_namesets the name of your environment (-n for name) and list of packages is the list of packages you want installed in the environment. For example, to create an environment named my\_env and install numpy in it, type conda create -n my\_env numpy.



When creating an environment, you can specify which version of Python to install in the environment. This is useful when you're working with code in both Python 2.x and Python 3.x. To create an environment with a specific Python version, do something like conda create -n py3 python=3 or conda create -n py2 python=2. I actually have both of these environments on my personal computer. I use them as general environments not tied to any specific project, but rather for general work with each Python version easily accessible. These commands will install the most recent version of Python 3 and 2, respectively. To install a specific version, use conda create -n py python=3.3 for Python 3.3.

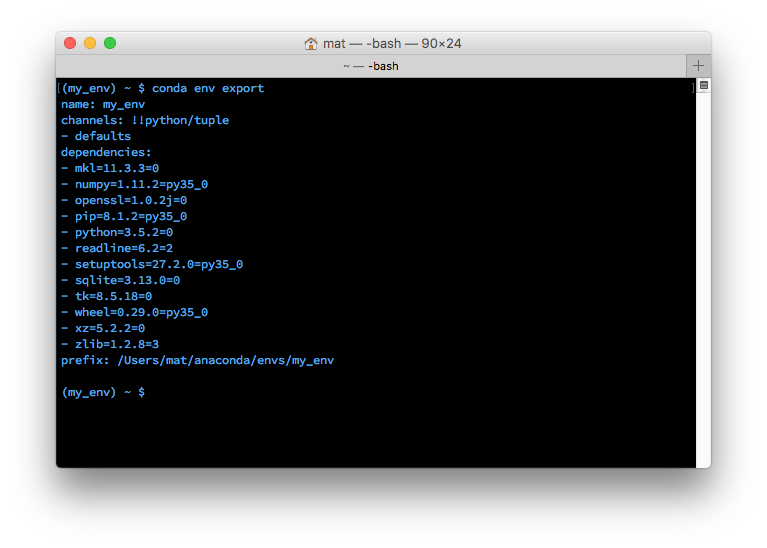
## Entering an environment

Once you have an environment created, use source activate my\_env to enter it on OSX/Linux. On Windows, use activate my\_env.

When you're in the environment, you'll see the environment name in the terminal prompt. Something like (my\_env) ~ $. The environment has only a few packages installed by default, plus the ones you installed when creating it. You can check this out with conda list. Installing packages in the environment is the same as before: conda install package\_name. Only this time, the specific packages you install will only be available when you're in the environment. To leave the environment, type source deactivate (on OSX/Linux). On Windows, use deactivate.

## Saving and loading environments

A really useful feature is sharing environments so others can install all the packages used in your code, with the correct versions. You can save the packages to a [**YAML**](http://www.yaml.org/) file with conda env export > environment.yaml. The first part conda env export writes out all the packages in the environment, including the Python version.



Exported environment printed to the terminal

Above you can see the name of the environment and all the dependencies (along with versions) are listed. The second part of the export command, > environment.yaml writes the exported text to a YAML file environment.yaml. This file can now be shared and others will be able to create the same environment you used for the project.

To create an environment from an environment file use conda env create -f environment.yaml. This will create a new environment with the same name listed in environment.yaml.

## Listing environments

If you forget what your environments are named (happens to me sometimes), use conda env list to list out all the environments you've created. You should see a list of environments, there will be an asterisk next to the environment you're currently in. The default environment, the environment used when you aren't in one, is called root.

## Removing environments

If there are environments you don't use anymore, conda env remove -n env\_name will remove the specified environment (here, named env\_name).

# Best practices

## Using environments

One thing that’s helped me tremendously is having separate environments for Python 2 and Python 3. I used conda create -n py2 python=2 and conda create -n py3 python=3 to create two separate environments, py2 and py3. Now I have a general use environment for each Python version. In each of those environments, I've installed most of the standard data science packages (numpy, scipy, pandas, etc.)

I’ve also found it useful to create environments for each project I’m working on. It works great for non-data related projects too like web apps with Flask. For example, I have an environment for my personal blog using [**Pelican**](http://docs.getpelican.com/en/stable/).

## Sharing environments

When sharing your code on GitHub, it's good practice to make an environment file and include it in the repository. This will make it easier for people to install all the dependencies for your code. I also usually include a pip requirements.txt file using pip freeze ([**learn more here**](https://pip.pypa.io/en/stable/reference/pip_freeze/)) for people not using conda.

## More to learn

To learn more about conda and how it fits in the Python ecosystem, check out this article by Jake Vanderplas: **[Conda myths and misconceptions](https://jakevdp.github.io/blog/2016/08/25/conda-myths-and-misconceptions/" \t "_blank)**. And here's the **[conda documentation](http://conda.pydata.org/docs/using/index.html" \t "_blank)** you can reference later.