

# Anaconda

Welcome to this lesson on using [Anaconda \(https://www.anaconda.com/what-is-anaconda/\)](https://www.anaconda.com/what-is-anaconda/) to manage packages and environments for use with Python. With Anaconda, it's simple to install the packages you'll often use in data science work. You'll also use it to create virtual environments that make working on multiple projects much less mind-twisting. Anaconda has simplified my workflow and solved a lot of issues I had dealing with packages and multiple Python versions.

Anaconda is actually a distribution of software that comes with `conda`, Python, and over 150 scientific packages and their dependencies. The application `conda` is a package and environment manager. Anaconda is a fairly large download (~500 MB) because it comes with the most common data science packages in Python. If you don't need all the packages or need to conserve bandwidth or storage space, there is also `Miniconda`, a smaller distribution that includes only `conda` and Python. You can still install any of the available packages with `conda`, it just doesn't come with them.

Conda is a program you'll be using exclusively from the command line, so if you aren't comfortable using it, check out this [command prompt tutorial for Windows \(https://www.lynda.com/-tutorials/Windows-command-line-basics/497312/513424-4.html\)](https://www.lynda.com/-tutorials/Windows-command-line-basics/497312/513424-4.html) or [Linux Command Line Basics course for OSX/Linux. \(https://www.youtube.com/watch?v=CpTfQ-q6MPU\)](https://www.youtube.com/watch?v=CpTfQ-q6MPU)

You probably already have Python installed and wonder why you need this at all. First, since Anaconda comes with a bunch of data science packages, you'll be all set to start working with data. Secondly, using `conda` to manage your packages and environments will reduce future issues dealing with the various libraries you'll be using.

## Managing Packages

```

python3 ~/anaconda/bin/conda install numpy
[~] $ conda install numpy
Fetching package metadata .....
Solving package specifications: .....
Using Anaconda Cloud api site https://api.anaconda.org

Package plan for installation in environment /Users/mat/anaconda:

The following packages will be downloaded:

package | build | size
-----|-----|-----
conda-env-2.6.0 | 0 | 601 B
numpy-1.11.2 | py34_0 | 2.7 MB
ruamel_yaml-0.11.14 | py34_0 | 179 KB
conda-4.2.11 | py34_0 | 387 KB
-----|-----|-----
Total: | 3.3 MB

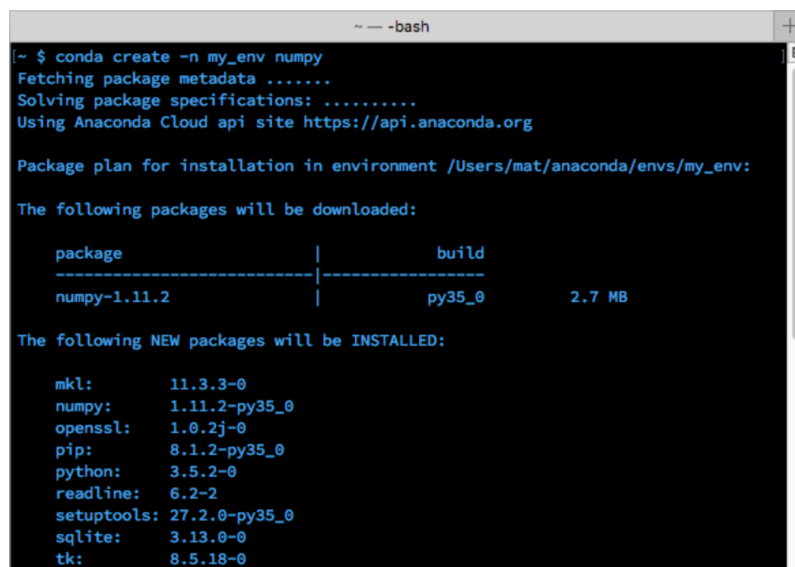
The following packages will be UPDATED:

conda: 4.1.12-py34_0 --> 4.2.11-py34_0
conda-env: 2.5.2-py34_0 --> 2.6.0-0
numpy: 1.11.1-py34_0 --> 1.11.2-py34_0
ruamel_yaml: 0.11.7-py34_0 --> 0.11.14-py34_0
  
```

Package managers are used to install libraries and other software on your computer. You're probably already familiar with `pip`, it's the default package manager for Python libraries. Conda is similar to `pip` except that the available packages are focused around data science while `pip` is for general use. However, conda is not Python specific like `pip` is, it can also install non-Python packages. It is a package manager for any software stack. That being said, not all Python libraries are available from the Anaconda distribution and conda. You can (and will) still use `pip` alongside conda to install packages.

Conda installs precompiled packages. For example, the Anaconda distribution comes with Numpy, Scipy and Scikit-learn compiled with the [MKL library \(https://docs.continuum.io/mkl-optimizations/\)](https://docs.continuum.io/mkl-optimizations/), speeding up various math operations. The packages are maintained by contributors to the distribution which means they usually lag behind new releases. But because someone needed to build the packages for many systems, they tend to be more stable (and more convenient for you).

## Managing Environments



```

~ -bash
$ conda create -n my_env numpy
Fetching package metadata .....
Solving package specifications: .....
Using Anaconda Cloud api site https://api.anaconda.org

Package plan for installation in environment /Users/mat/anaconda/envs/my_env:

The following packages will be downloaded:

package | build
-----|-----
numpy-1.11.2 | py35_0 2.7 MB

The following NEW packages will be INSTALLED:

mkl: 11.3.3-0
numpy: 1.11.2-py35_0
openssl: 1.0.2j-0
pip: 8.1.2-py35_0
python: 3.5.2-0
readline: 6.2-2
setuptools: 27.2.0-py35_0
sqlite: 3.13.0-0
tk: 8.5.18-0

```

Along with managing packages, Conda is also a virtual environment manager. It's similar to [virtualenv \(https://virtualenv.pypa.io/en/stable/\)](https://virtualenv.pypa.io/en/stable/) and [pyenv \(https://github.com/yyuu/pyenv\)](https://github.com/yyuu/pyenv), other popular environment managers.

Environments allow you to separate and isolate the packages you are using for different projects. Often you'll be working with code that depends on different versions of some library. For example, you could have code that uses new features in Numpy, or code that uses old features that have been removed. It's practically impossible to have two versions of Numpy installed at once. Instead, you should make an environment for each version of Numpy then work in the appropriate environment for the project.

This issue also happens a lot when dealing with Python 2 and Python 3. You might be working with old code that doesn't run in Python 3 and new code that doesn't run in Python 2. Having both installed can lead to a lot of confusion and bugs. It's much better to have separate environments.

You can also export the list of packages in an environment to a file, then include that file with your code. This allows other people to easily load all the dependencies for your code. Pip has similar functionality with `pip freeze > requirements.txt`.

## Where we go from here

Next, I'll get into the details of using Anaconda. First I'll cover installing it, then using the package manager, and finally creating and managing environments.