

What is Anaconda Distribution?

Anaconda is a program to manage (*install, upgrade, or uninstall*) packages and environments to use with Python. It's simple to install packages with Anaconda and create virtual environments to work on multiple projects conveniently.

Even if you already have Python installed, it will be beneficial to use Anaconda/Miniconda because:

1. Anaconda comes with a bunch of data science packages; you'll be all set to start working with data.
2. Using conda to manage your packages and environments will reduce future issues dealing with the various libraries you'll be using.

Python Packages

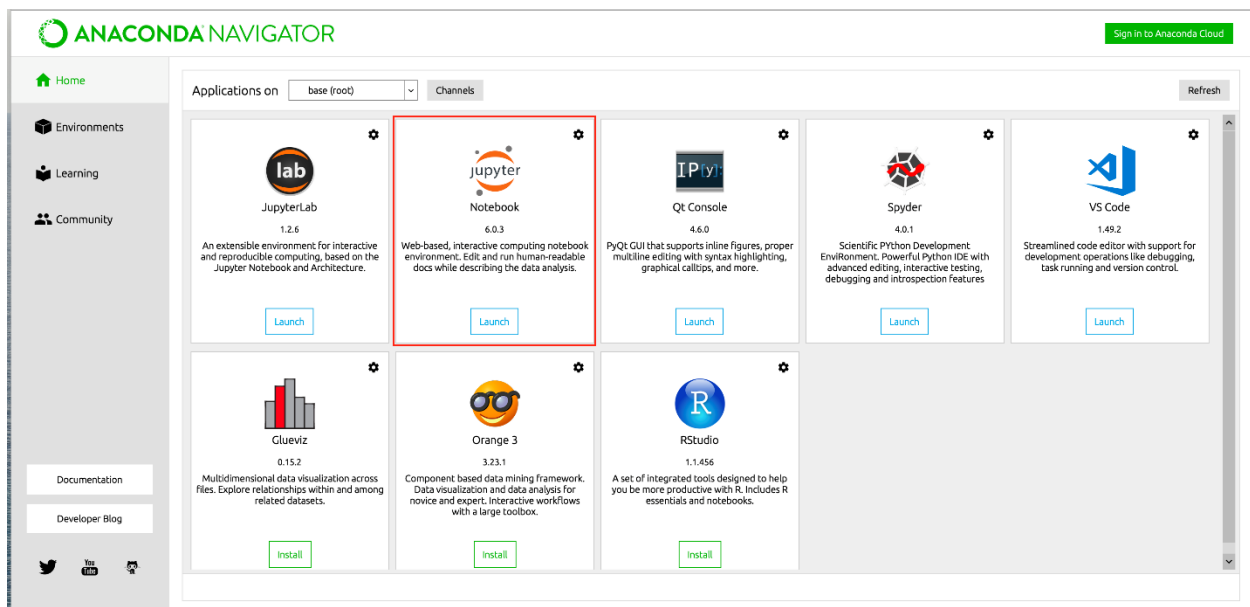
A package is a bunch of modules, where each module consists of a set of classes and function definitions. After installing a particular package, you can import and use the functions defined in that package.

If we install Anaconda, then a basic few packages are installed by default. However, you can install any more packages, if needed.

Anaconda Distribution

Anaconda is a fairly large download (~500 MB) because it comes with Python's most common data science packages. [Anaconda](#) is a software distribution that includes the following:

- **Anaconda Navigator** - It is a graphical user interface that helps open up any installed applications, such as Jupyter notebook or VS code editor. We will learn more about the notebook in the next lesson. See a snapshot of Anaconda Navigator below:



Anaconda Navigator GUI, same in both macOS/Linux and Windows. We will learn to code in the Jupyter Notebook in the next lesson.

- **conda** - A command-line utility for package and environment management. Mac/Linux users can use the Terminal, and Windows users can use the "**Anaconda Prompt**" to execute conda commands. Windows users must run the Anaconda Prompt as an Administrator. Your first command should be
- **conda --version**

If you are not comfortable on the command line, check out the [command prompt tutorial for Windows](#) or our [Linux Command Line Basics](#) course for OSX/Linux.

- **Python** - The latest version of Python gets installed as an individual package.
- Over 160 scientific packages and their dependencies are also installed.

If you don't need all the packages or need to conserve bandwidth or storage space, there is an option for you - **Miniconda**.

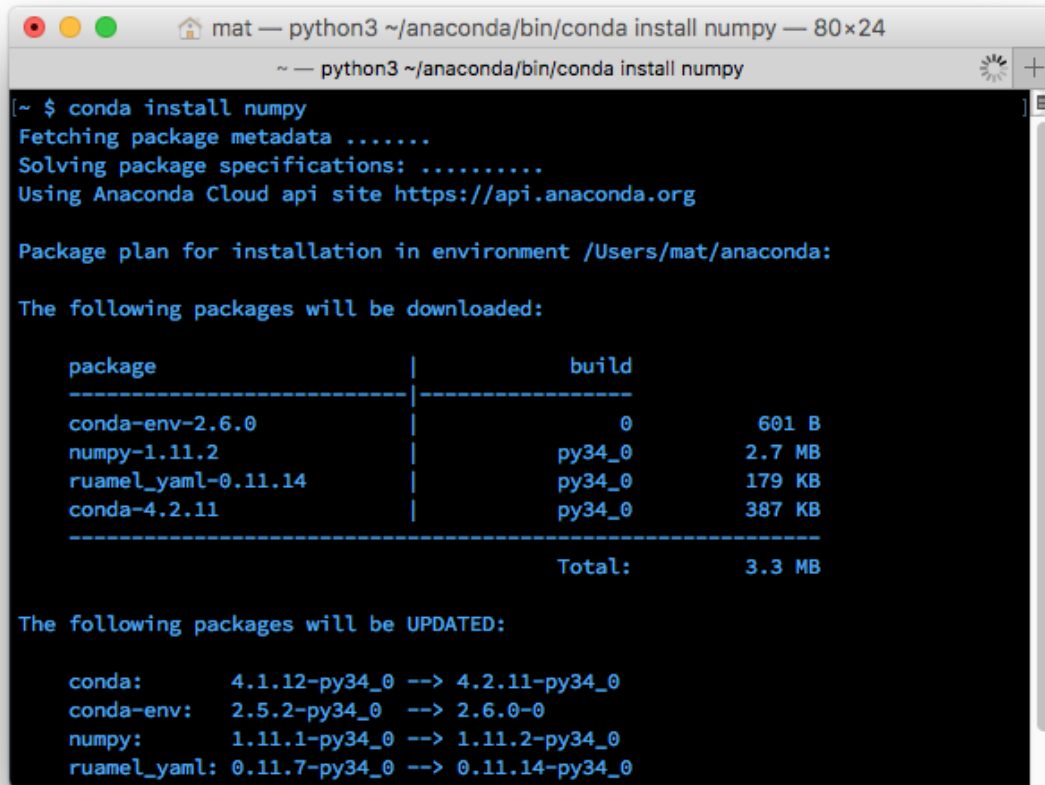
Miniconda

Either Anaconda or Miniconda is adequate for this course. [Miniconda](#) is a smaller distribution as compared to Anaconda, which includes only conda and Python. Miniconda can do everything Anaconda is capable of, but **doesn't have the preinstalled packages**. You can still install any of the available packages using `conda install PACKAGENAME` on the terminal/Anaconda Prompt. Interestingly, you can anytime upgrade from Miniconda to Anaconda by using the command:

```
conda install anaconda
```

In the command above, did you notice that we followed the same conda install PACKAGENAME syntax? See an example below to install the Numpy package with conda on the Terminal/Anaconda Prompt. You will see a demonstration soon after understanding the setup.

Overview - Managing Packages using either pip or conda



```
mat — python3 ~/anaconda/bin/conda install numpy — 80×24
~ — 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
-----|-----
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
```

Installing numpy with conda

The conda and pip both are the Python package managers. Package managers are used to installing libraries and other software on your computer. pip is the default package manager for Python libraries, whereas conda focuses only on the packages that are available from the Anaconda distribution.

Update Note

In the newer version of Anaconda/Miniconda, both pip and conda package managers are included by default, so you do not need to install them separately.

Both pip and conda gets installed when you install either Anaconda or Miniconda. On the next page, we will see details to install Anaconda/Miniconda. However, the pip also comes preinstalled with the Python 2 >=2.7.9 or Python 3 >=3.4.

In case if you have Anaconda/Miniconda already installed, and don't have pip in your system, there are two ways:

1. Refer to the [Pip installation instructions](#)

2. Install conda first and then install pip using conda. We will see commands for this step on the very next page because we will learn to install Anaconda/Miniconda next.

Which one should I prefer - pip or conda?

There are two points you can consider before making a choice:

1. The available packages available from the Anaconda distribution in conda focus on data science, whereas pip is for general use. Conda installs precompiled packages. For example, the Anaconda distribution comes with Numpy, Scipy, and Scikit-learn compiled with the [MKL library](#), speeding up various math operations. **But, sometimes, you may need packages other than the ones listed on the Anaconda distribution.**
2. Pip can install both Python and non-Python packages. Pip can install any package listed on the [Python Package Index](#) (PyPI).

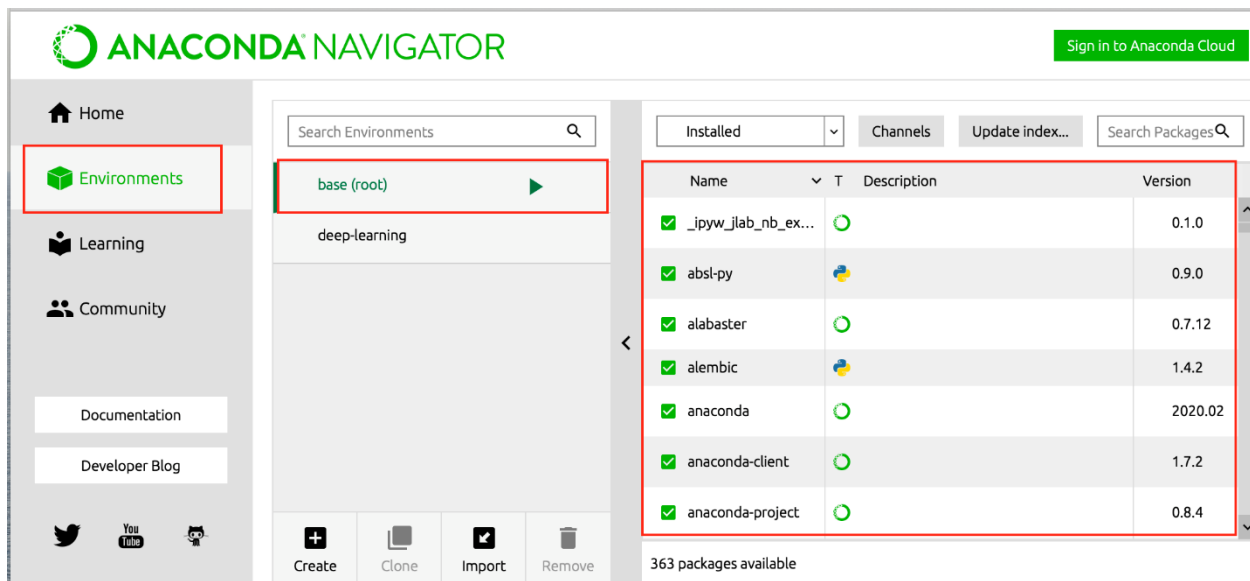
You can (and will) still use pip alongside conda to install packages.

Environments

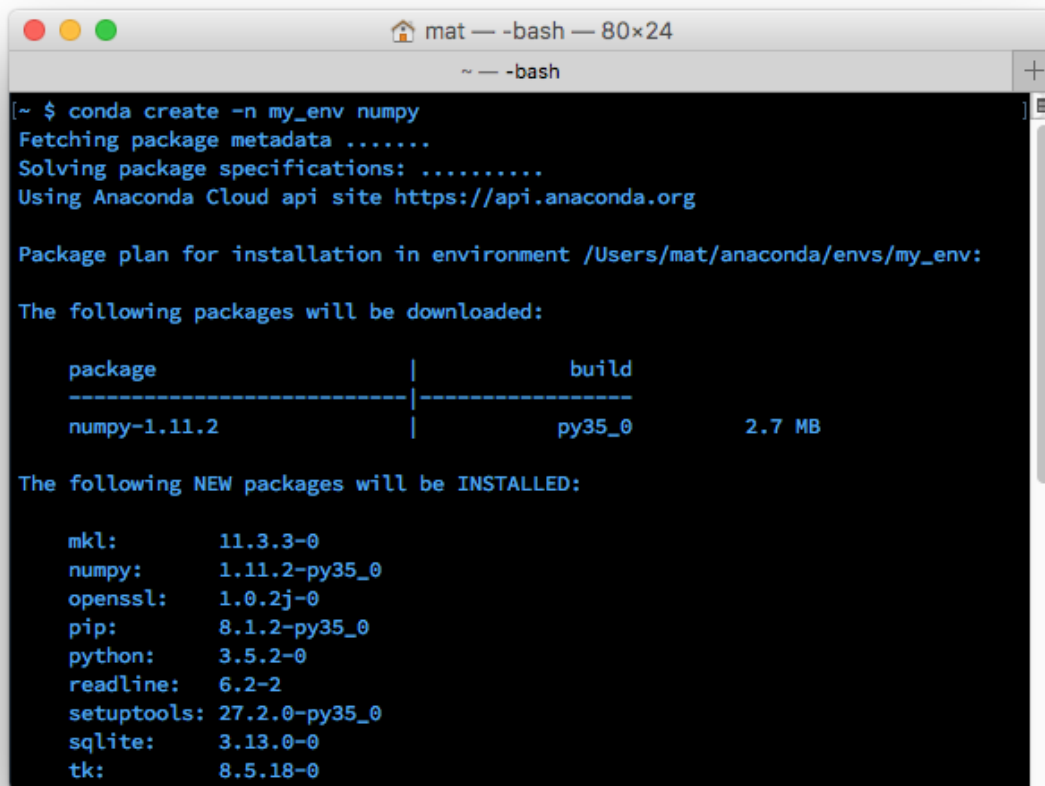
A Python environment comprises a particular version of each of the following:

- Python interpreter,
- Python-packages, and
- The utility scripts, such as pip.

It is possible to have two or more environments residing on the same computer *virtually*. If you are using Anaconda, you are in the base(root) environment by-default.



The default base(root) environment in Anaconda

A terminal window titled 'mat - bash - 80x24' showing the output of a conda command. The user has run 'conda create -n my_env numpy'. The terminal shows progress bars for fetching metadata and solving specifications. It then displays the package plan for installation in the environment '/Users/mat/anaconda/envs/my_env'. A table lists the packages to be downloaded: numpy-1.11.2 (py35_0, 2.7 MB). Below this, it lists the new packages to be installed: mkl, numpy, openssl, pip, python, readline, setuptools, sqlite, and tk, each with their respective versions.

```
[~ $ 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
```

Creating an environment with conda

Along with managing packages, Conda is also a virtual environment manager. It's similar to [virtualenv](#) and [pyenv](#), other popular environment managers.

Why do you need a Virtual Environment?

Each virtual environment remains isolated from other virtual environments, and the default “system” environment. **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`.