



WORKING WITH CONDA

MODULES, PACKAGES, AND ENVIRONMENTS

BUILT-IN MODULES

Python comes with numerous **built-in modules** providing extensive functionality for handling of many typical programming problems, including:

- Text and HTML processing
- Maths and statistics
- File formats and access
- Internet communication
- Testing and debugging

THE STANDARD LIBRARY

The collection of built-in modules is known as the [Python Standard Library](#).

These modules are available via import without further installation requirements, e.g.:

```
import json
import datetime
import math
```

PACKAGES

Additional **packages** (also known as **libraries**) can be installed:

- To re-use Python code previously created by yourself or others
- To simplify the creation of complex programming procedures

For example:

```
import pandas as pd
```

The `pd` above is an **alias** which can be used in subsequent code to refer to the given package.

The `pandas` package contains numerous **modules** and functionality to help with data analysis.

MODULES

Packages can contain numerous **modules** and **sub-modules**

- We can import the whole package or just a specific module or submodule

For example:

```
from sklearn.metrics import consensus_score
```

The above code will import the `consensus_score` sub-module from the `metrics` module of the `sklearn` package.

DEPENDENCIES

Packages may have [dependencies](#) on other packages:

- These dependencies must also be installed for the package to work

For example, `pandas` is dependent on a handful of other packages to work:

- `setuptools`
- `NumPy`
- `python-dateutil`
- `pytz`

The [Anaconda Distribution](#) which you may have installed includes all of these packages; to use `pandas` all you need to do is `import` it.

ENVIRONMENTS

Environments allow users to have **multiple combinations** of package and version installations on a single machine.

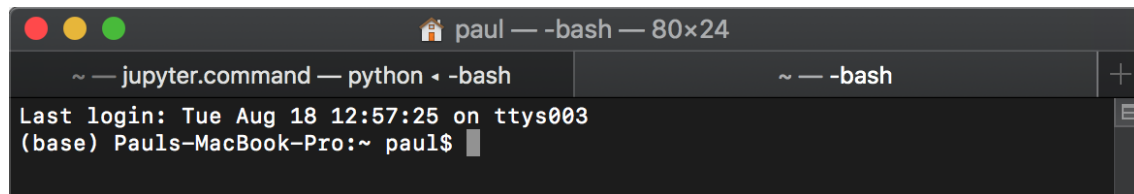
- ensures that the execution of code is **reproducible** in future or on other machines
- allows work on multiple projects where **incompatible dependencies** are required

With such a wide range of packages available, most of which have different versions (the [Python Package Index](#) lists 250,000+ projects with 2,000,000+ releases), it is not hard to see how different users or use cases can end up requiring quite a different collection of packages and versions to be installed.

CONDA

Conda is an **environment manager** which simplifies the process of **creating**, **updating**, and **switching** between environments.

The default conda environment is referred to as **base**. If you have installed Anaconda, you should see this in the **Terminal** (Mac) or **Anaconda Prompt** (Windows):



```
paul — -bash — 80x24
~ — jupyter.command — python - -bash
~ — -bash
Last login: Tue Aug 18 12:57:25 on ttys003
(base) Pauls-MacBook-Pro:~ paul$
```

In Windows the Anaconda Prompt should be available from the Start Menu. See [here](#) for help.

CONDA COMMANDS OVERVIEW

`conda create --name mynewenv`

- creates a new environment with the given name (here, `mynewenv`)

`conda activate mynewenv`

- activates the environment on your computer (i.e. will enable the use of any packages and the particular versions installed in that environment, and any installed packages will be put into the currently active environment)

`conda install flask`

- installs a new package and its dependencies to the environment (here, `flask`)

`conda install flask=1.1.2`

- install the specific version of a package to the environment

`conda list`

- shows the packages and versions installed in the current environment

`conda env export > environment.yml`

- creates a YAML (a type of configuration) file which lists the packages in the active environment

`conda env create -f environment.yml`

- creates an environment from the `environment.yml` file (execute the command in the same location as the file)

See the [documentation](#) for more commands.

virtualenv & pip

virtualenv is an alternative to conda for the creation of environments.

- In both cases they can be considered 'virtual', in that if the same package is in use in numerous environments, it will not have multiple installations on the given computer
- Whereas conda can handle both environment management and package installation, virtualenv only handles environments and is typically used in conjunction with pip
- pip is a package installer for Python (and can also be used to install into conda environments)

WHICH SHOULD I USE?

You will often see instructions for the installation of packages using `pip` alongside instructions for `conda` (see [pandas-profiling](#) as an example).

There are no hard and fast rules but in general try to do the following:

- create a new environment with `conda` whenever working on a new project (assignment, piece of work, etc)
- use `conda` for package installations when it is available
- use `pip` if there is no `conda` command