Virtual Environments and Packages

Introduction

Python applications will often use packages and modules that don't come as part of the standard library.

As a popular open source development project, Python has an active supporting community of contributors and users that also make their software available for other Python developers to use under open source license terms.

This section will cover:

1. How to search/install/uninstall third part librairies using **pip**
2. How to use **virtual environments**

Read more https://docs.python.org/3.7/tutorial/venv.html (https://docs.python.org/3.7/tutorial/venv.html)

Managing Packages with pip

You can install, upgrade, and remove packages using a program called pip. By default pip will install packages from the Python Package Index, https://pypi.python.org/pypi (https://pypi.python.org/pypi). You can browse the Python Package Index by going to it in your web browser, or you can use pip's limited search feature:

You can try any search

```
$ pip search wrangling
```

In [1]:
```
!pip search wrangling
```

```
ERROR: XMLRPC request failed [code: -32500]
RuntimeError: PyPI no longer supports 'pip search' (or XML-RPC search). Pleas
e use https://pypi.org/search (https://pypi.org/search) (via a browser) inste
ad. See https://warehouse.pypa.io/api-reference/xml-rpc.html#deprecated-metho
ds (https://warehouse.pypa.io/api-reference/xml-rpc.html#deprecated-methods)
for more information.
```

pip has a number of subcommands: "search", "install", "uninstall", "freeze", etc.

```
$ pip install requests
```

In [2]:
```
import requests
```

To install a third part library, you need root priviledges.

Indeed, third part libraries are installed in site-packages folder. On my macbook it is here:

```
/opt/local/Library/Frameworks/Python.framework/Versions/3.5/lib/python
3.5/site-packages/
```

On your Ubuntu, it will be in

```
/usr/local/lib/pythonX.Y/site-packages/
```

Check running

```
$ python -m site
```

It is possible to install packages (modules, third part librairies) without any priviledges thanks to virtual environments. Let's dive into **virtual environments**, we will come back to pip after

pip is the preferred installer program. Starting with Python 3.4, it is included by default with the Python binary installers.

Virtual Environments

Motivations

The main motivation for virtual environments:

- **To allow multiple Python projects that have different requirements, to coexist on the same computer.**

Example: application A runs with version 1. of the *request* package. Project for a new application B also needs *request* but with a feature introduced in version 2.2. Then upgrading request will upgrade for all applications on the machine. This will eventually cause a bug in application A if the new feature has no backward compatibility. So, it is important to test a project with different Python versions and packages' versions, hence the need for different "virtual" environments that are independent and do not overlap.

Creating Virtual Environments

1. venv

The module used to create and manage virtual environments is called venv (https://docs.python.org/3.6/library/venv.html#module-venv).

A virtual environment is a self-contained directory tree that contains a Python installation for a particular version of Python, plus a number of additional packages.

NOTE: *venv* is part of Python since Python 3.3. It is the standard tool for creating virtual environments. *virtualenv (https://virtualenv.pypa.io)* is a third party alternative (and predecessor) to venv. It allows virtual environments to be used on versions of Python prior to 3.4.

To create a virtual environment, open your command-line window and run:

```
$ python3 -m venv path/to/your/virtual/environment
```

If you are a Windows user, just open your command prompt, and type:

```
> python -m venv your_virtual_env_name
```

You can specify absolute or relative paths for your virtual environment. Notice that we use `python` instead of `python3` in Windows.

The python -m module trick

```
python -m <module-name>
```

Search sys.path for the named module and execute its contents as the **main** module

Useful examples:

- `python -m http.server`

launches a web service that serves the content of the current directory

- `python -m json.tool file.json`

pretty prints (indent) the content of a json file

- `python -m venv`

creates a virtual environment

- `python -m pip`

same as pip, we will come back to this one

Create a virtual environment

```
$ python3 -m venv dw
```

(This is valid in Windows as well, but use `python` instead of `python3`

```
> python -m venv dw
```

What happened?

Browse `dw` directory and figure out what happened

A directory tree has been created.

bin/ contains a copy of binaries python and pip

lib/ contains a directory pythonX.Y/site-packages where packages will be installed

Use the new environment

```
$ source dw/bin/activate
(dw) $
```

In Windows, navigate to the newly created virtual environment and type:

```
> activate
(dw) >
```

Note: If you are using Anaconda Prompt, you might want to deactivate the default loaded base environment, which was loaded by conda. Then, proceed to activate your new environment.

```
(base) > conda deactivate
> activate
(dw) >
```

A prefix with the name of the virtual environment appears before the prompt. In this case, you will see a `(dw)` in front of the prompt.

```
# to leave the virtual env
```

```
$ deactivate
```

```
# to delete virtual env, simply remove directory. In Windows, you can just
remove the entire directory without any issue.
$ rm -rf path/to/your/virtual/environment
```

2. conda

Conda is an interface for managing installations of various packages, a.k.a. a package manager. Conda also allows creation of new virtual environments with a pre-installed set of packages.

To list down all existing environments in Anaconda:

```
> conda info --envs
```

In [6]:
```
!conda info --envs
```

```
# conda environments:
#
base                     D:\Anaconda3
python-cvcourse          D:\Anaconda3\envs\python-cvcourse
python-dscourse       *  D:\Anaconda3\envs\python-dscourse
web_crawler              D:\Anaconda3\envs\web_crawler
```

The environment marked with an asterisk (*) indicates the current loaded environment.

To create a new environment named `saturn` :

```
> conda create --name saturn
```

This creates a new environment with some basic packages (and the same Python version as the base).

Sometimes we want more control over which version of Python or which packages we want to be in the environment when created. So, we can use other options, such as creating an environment with a specific package:

```
> conda create --name neptune scipy matplotlib
```

Sometimes we want an environment with Python 2 along with other specific versions of other packages. This can be done:

```
> conda create --name neptune_py2 python=2.7 scipy=0.15
```

After you have created these environments, to use them, you need to activate an environment of your choice:

```
> conda activate neptune
```

Now, to verify if the packages in this environment are correct as to what you have specified, examine the packages installed by:

```
> conda list
```

In [7]:
```
!conda list
```

```
# packages in environment at D:\Anaconda3\envs\python-dscourse:
#
# Name                    Version                   Build  Channel
argon2-cffi               20.1.0           py38he774522_1
async_generator           1.10                       py_0
attrs                     20.2.0                     py_0
backcall                  0.2.0                      py_0
beautifulsoup4            4.9.3              pyhb0f4dca_0
blas                      1.0                         mkl
bleach                    3.2.1                      py_0
brotlipy                  0.7.0           py38he774522_1000
bs4                       4.9.3                        0
ca-certificates           2020.7.22                    0
certifi                   2020.6.20                py38_0
cffi                      1.14.3           py38h7a1dbc1_0
chardet                   3.0.4                 py38_1003
click                     7.1.2                      py_0
colorama                  0.4.3                      py_0
cryptography              3.1.1            py38h7a1dbc1_0
```

To deactivate the environment,

```
> conda deactivate
```

Conda will fall back on the base environment when you do this.

If you wish to clone an existing environment to a new one (for practical reasons sometimes), you can do this:

```
> conda create --name neptune_clone --clone neptune
```

You can go to https://conda.io/docs/user-guide/tasks/manage-environments.html (https://conda.io/docs/user-guide/tasks/manage-environments.html) for the full guide on how to use `conda` to manage your virtual environments within Anaconda.

3. virtualenv

There's another way of creating virtual environments for Python, that is to use the virtualenv package:

```
> pip install virtualenv
```

To test your installation:

virtualenv --version

and you should see the version number for the virtualenv package.

Let's create a project folder:

```
> virtualenv my_project_folder
```

This will create a folder in the current directory which will contain the Python executable files, and a copy of the pip library which you can use to install other packages. The name of the virtual environment (in this case, it was `my_project_folder`) can be anything; without providing any specific directory path will place the files in the current directory instead.

For more info, look at [http://docs.python-guide.org/en/latest/dev/virtualenvs/ (http://docs.python-guide.org/en/latest/dev/virtualenvs/)](http://docs.python-guide.org/en/latest/dev/virtualenvs/)

You can also use the Python interpreter of your choice (like python2.7). For example,

```
virtualenv -p C:\Python\bin\python_27 my_project
```

But this is not possible if you had installed Anaconda based on a specific Python version, say 3.6. Then you do not have a copy of Python 2.7 in your machine.

Hence, this brings us to the third way of creating virtual environments (particularly suitable if you are using the full Anaconda suite), that is using `conda` .

Installing packages with conda and pip

As a popular open source development project, Python has an active supporting community of contributors and users that also make their software available for other Python developers to use under open source license terms.

This allows Python users to share and collaborate effectively, benefiting from the solutions others have already created to common (and sometimes even rare!) problems, as well as potentially contributing their own solutions to the common pool.

pip and **conda** are the preferred package installer programs. **pip** started with Python 3.4 and Python 2.7.9, and it is included by default with most Python binary installers. As for **conda**, it is definitely available when Anaconda is installed, but not necessarily in other distributions.

> NOTE: For older **python versions only**, a script is available at https://bootstrap.pypa.io/get-pip.py (https://bootstrap.pypa.io/get-pip.py)

```
$ wget https://bootstrap.pypa.io/get-pip.py $ python get-pip.py
```

** PyPI - the Python Package Index ** The Python Package Index is a repository of software for the Python programming language. https://pypi.python.org/pypi (https://pypi.python.org/pypi) PyPI is the default Package Index for the Python community. It is open to all Python developers to consume and distribute their distributions.

`pip` searches and installs packages from this repository by default.

To run pip, you can simply call `pip` command or `python -m pip` .

```
> pip install beautifulsoup4
```

In conda, you can also even search for packages (whether they exist or not, or if you forgot part of the name):

```
> conda search scipy
```

...and proceed to install them:

```
> conda install scipy
```

You will notice that conda tells you explicitly and transparently, which packages will be installed, and if they are overwriting a previous version, what versions are those. It also traces back all the necessary dependencies (other packages) and gets them installed and upgraded if needed. Very convenient!

Note: `pip` also does this actually. But it outputs very verbosely you usually miss it.

To update packages,

```
> conda update scipy
> conda update conda
```

or to update all...

```
> conda update all
```

Use the above with utmost caution, especially if you are already in the middle of a project. This is best done at the beginning or after you have created a new environment. The chances of your existing code failing or running into errors is quite high.

Likewise, to upgrade packages with pip:

```
> pip install numpy --upgrade
```