Useful pythonic practices

Alexander Belikov
Knowledge Lab, Computation Institute, University of Chicago, Chicago IL 60637, USA

I. INTRODUCTION

Python is an interpreted language with an extensive suite of libraries. It is expressive and flexible from the point of view performance tuning. Various extensions allow usage of pre-compiled code (Cython, numba, etc). Python is a fun language to learn. An unprecedented amount of literature and resources are dedicated to python lore is available online.

II. DEVELOPMENT PRACTICES

Even if you are starting to use a language, like python, recreationally you might end developing multiple large projects. So it is good to maintain certain practices from the very beginning.

A. Virtual environments

While developing your own set of scripts or a library, it is useful to keep the version of external libraries fixed. For instance, if you are using a function *foo(a, b)* from library *meerkats* version 1.0 and in version 2.0 the signature of *foo()* changes to *foo(c, a, b)* your current code would become incompatible if *meerkats* were updated. Different projects might rely also on different version of libraries. In a number of cases a user simply does not have the privileges to install or change version of python system packages.

There are two main ways of working with python virtual environments:

- using *virtualenv* python package and *pip* for installing packages
- Anaconda python distribution

It is preferable to use Anaconda distribution for the following reasons:

- conda installs binary packages from its own repositories (so they are not compiled locally during install)
- a set of packages with version-specific dependencies forms a graph. Fixing versions of several packages may result in a solution, where some packages would have be downgraded or upgraded. Or perhaps some of the version constraints would have to be violated, so there would not be a solution in the strict sense. Conda uses SAT solver for dependency resolution [1].

B. Are virtual environments necessary?

Why would one want or need to use virtual environments? While they are not necessary for starting with python, at a certain time their use becomes almost imperative. Here we list situations where you would want to use virtual envs.

- you work on a server, where you don't have admin rights (you can't install python libraries system-wide) and you want to install a specific library *meerkats*. (in that case you could still install anaconda in your home directory)
- you work on a server and want a specific version of library *meerkats*, for example 0.3 (and 0.4 when comes out, because you want to test your scripts with most modern scripts), while the system-wide version is 0.2 and likely to remain 0.2. Quite the opposite is possible: while developing your scripts you want to be pinned to meerkats 0.3 (for obvious reasons - you want to isolate the behavior of your code from the possible different behaviors due to different version of external libraries) while your mettlesome admin keeps updating the version of *meerkats*.

- you want to study the effect of running your code under two versions of the same library (and conclude that the newer version is not harmful).
- you are in the process of developing your own library, which you would like to be available for importing 'globally' (within the environment) and place a reference to it (the so-called '.egg-link') back to the project source code directory. In that way the changes you would be making to your library would be actual to any code that imports your library.

C. Basic conda commands

To create an enviroment *env*:

```
$conda create --name env
```

To activate environment *env*:

```
$source activate env
```

Morpheme (*env*) preceding shell specifying info (such as hostname and shell separator, e.g. $ in case of bash) will indicate that the user is indeed in the environment To deactivate current environment *env*:

```
(env) hostname:curr_dir username$
```

To deactivate current environment *env*:

```
(env) $source deactivate
```

To list all environments:

```
$conda info --envs
```

To update all packages:

```
$conda update --all
```

To remove environmet *env*:

```
$conda-env remove -n env
```

For more information on command equivalence between *virtualenv-pip* pair and *conda* please refer to [2]. For more information on managing conda environments refer to [3].

III. PYTHON GUIDES

A multitude of python guide are available that choose to introduce python from different angles. We mentioned Fast Lane to Python by Norm Matloff, as an example of a basic and concise introduction [4].

[1] I. Schnell (2013), URL `https://www.continuum.io/blog/developer/new-advances-conda-0`.
[2] (2016), URL `http://conda.pydata.org/docs/_downloads/conda-pip-virtualenv-translator.html`.
[3] B. V. de Ven (2015), URL `https://www.continuum.io/blog/developer-blog/python-packages-and-environments-conda`.
[4] N. Matloff (2012), URL `http://heather.cs.ucdavis.edu/~matloff/Python/PLN/FastLanePython.pdf`.