Python Introduction Course: simplifying python programming

With emphasis on data-science problems
Geant4 Course at the 16th Seminar on Software for Nuclear, Sub-nuclear and Applied Physics, Porto Conte, Alghero (Italy), 26-31 May 2019. (https://agenda.infn.it/event/17240/)
This course is available on gitlab (https://gitlab.com/andreadotti/pyalghero2019)
Contact me: (mailto:andrea.dotti@gmail.com)

Packages location

We have seen that a module or package can be used in a python session via:

In [2]:
```
import numpy as np
```

But where does the file(s) of a package actually reside?
When an import statement is executed there are several paths where the package is searched for (similarly to how `PATH` or `LD_LIBRARY_PATH` search paths work for binaries and libraries on linux).

In [4]:
```
import sys
sys.path
```

Out[4]:
```
['/mnt/d/Andrea/Work/PyAlghero2019/Slides',
 '/home/adotti/anaconda3/envs/pycourse/lib/python37.zip',
 '/home/adotti/anaconda3/envs/pycourse/lib/python3.7',
 '/home/adotti/anaconda3/envs/pycourse/lib/python3.7/lib-dynload',
 '',
 '/home/adotti/anaconda3/envs/pycourse/lib/python3.7/site-packages',
 '/home/adotti/anaconda3/envs/pycourse/lib/python3.7/site-packages/IPython/ext
ensions',
 '/home/adotti/.ipython']
```

In [5]:
```
np.__file__
```

Out[5]:
```
'/home/adotti/anaconda3/envs/pycourse/lib/python3.7/site-packages/numpy/__init
__.py'
```

A module, when *imported*, is searched in order in the list of paths. The current directory is by default added as the first search path. The directory `site-packages` usually contains the distribution modules and packages. Note that often packages can come in `egg` format (all files of a packaged are *zipped* together with meta-data files).

Changing search path

You can add or modify the path search in two ways, directly from a python program, manipulating the `sys.path` list:

In [11]:
```
import sys
from os.path import join
sys.path.append(join('home','adotti','work'))
sys.path[-1]
```

Out[11]:
```
'home/adotti/work'
```

On *NIX systems You can also define the environment variable `PYTHONPATH` *before* starting a python session to extend the search path.

Installing packages

pip and virtualenv

The PyPI (https://pypi.org/) (Python Package Index) is a repository of published python packages (currently more than 180.000 projects) that can be easily installed.
The oldest way to install a package is to use `easy_install` that comes with the python `setuptools`. For example, to install the python package `pip` for the whole system you can do:

```
#Don't do that
sudo easy_install pip
```

`pip` is a more flexible way to interact with PyPI. It usually comes with all python distributions and thus you do not need to install it. The command line utility allows for the installation/removal of packages, for example to install the package `numpy` for the whole system you can do:

```
#Don't do this
sudo pip install numpy
```

`pip` will take care of dependencies installing them for you.

The most appreciated feature of `pip` is the possibility to specify a [requirements file (https://pip.readthedocs.io/en/1.1/requirements.html)](https://pip.readthedocs.io/en/1.1/requirements.html) that contains the list of packages and versions you need to be installed in one go:

```
cat requirements.txt
MyApp
Framework==0.9.4
Library>=0.2

pip install -r requirements.txt
```

A python environment can be reproduced:

```
pip freeze > requirements.txt
```

virtualenv

virtualenv solves a very specific problem: it allows multiple Python projects that have different (and often conflicting) requirements, to coexist on the same computer.
It also allows to install packages without the need to have super-user privileges (i.e. no `sudo` needed).

```
sudo pip install virtualenv
cd ~/myproject
virtualenv myenv
```

This will create an *environment* (a directory) called `myenv` that contains a python distribution that can be *activated*:

```
cd ~/myproject
source myenv/bin/activate
pip install -r requirements.txt
```

Now the specified packages are installed in a subdirectory of `myenv` creating an isolated environment. You can *deactivate* the environment with:

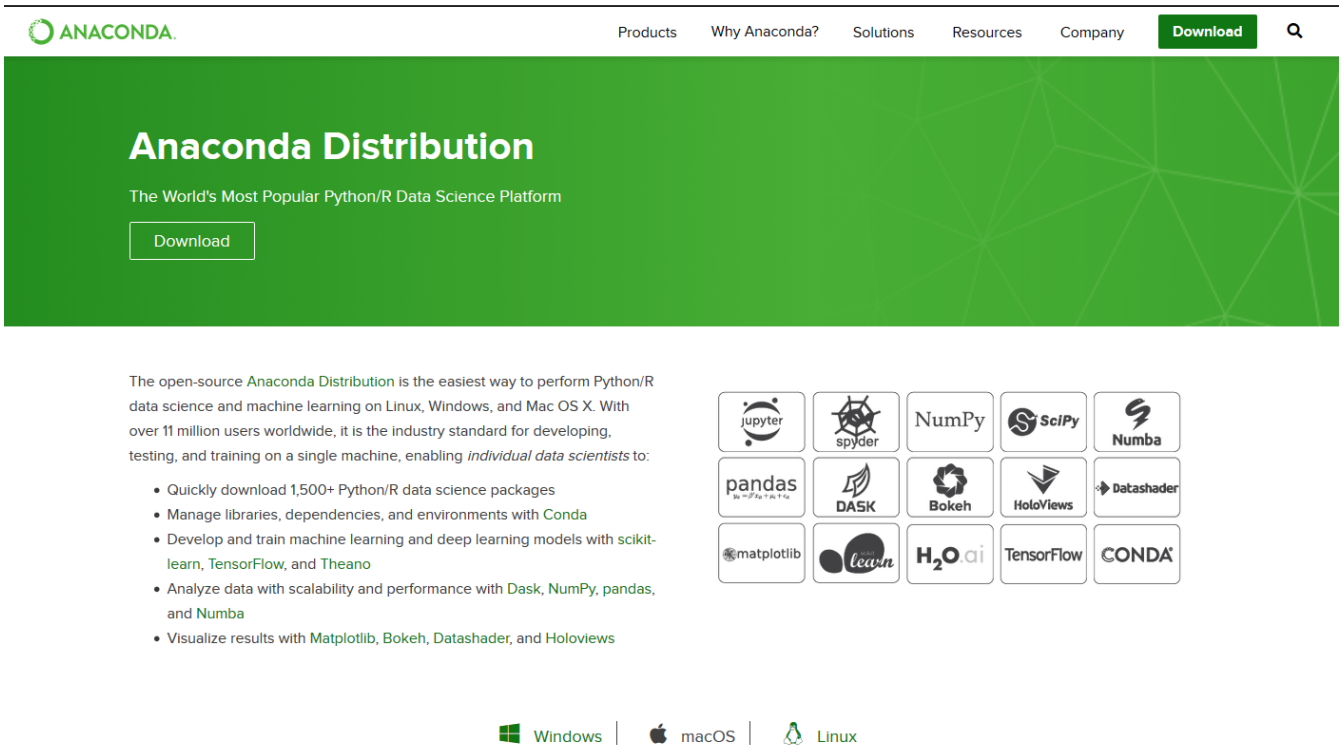
```
myenv/bin/deactivate
```

Anaconda distribution

The [Anaconda distribution (https://anaconda.org/)](https://anaconda.org/) is maintained by a private company (Anaconda Inc.), it provides a free and open-source distribution tailored to data science.

Similarly to pip/virtualenv it provides a package and environment manager.

- Linux, MacOS and Windows are all supported
- The support is not limited to python, but also to notably R and in general any binary package (e.g. Qt, GCC,...)



After installing anaconda distribution, similarly to `pip` packages can be installed (globally) with:

```
conda install numpy
```

However usually packages are installed in *environments*:

```
conda env create myenv
conda activate myenv
conda install numpy
conda deactivate
```

Similarly to `pip` all needed packages can be specified via a file (in YAML format):

```
cat environment.yml
name: myenv
dependencies:
- python=3
- numpy

conda env create -f environment.yml
conda activate myenv
...
conda deactivate
```

This tutorial

For this tutorial we have pre-installed anaconda on the school VM. We have also created an environment with all python code that is needed. Remember to activate the environment (https://gitlab.com/andreadotti/pyalghero2019/blob/master/environment.yml) with:

```
conda activate course
```

This should be done in each new terminal. Note the name of the environment, prefixed to the terminal prompt.

IPython interpreter

Instead of the default interpreter, `ipython` provides additional features, very useful in interactive sessions:

- Improved command line navigation (similar to a shell/terminal)
- Syntax highlight
- Auto completion: press `Tab`-key with an incomplete word/command to see suggestions
- Call system program from interpreter with `!` (e.g.: `!pwd`). Note the form `mydir = !pwd`
- Improved history handling. Including: type the first characters of an old command, press `Up`-key to auto complete line to most recent matching line
- Retrieve the last computed result with _ or with _<N> for output *N*
- *Magic* functions, extensions to IPython that can improve interactive sessions. Some examples:
    - `%magic` help on magic subsystem itself
    - `%timeit python-code-goes-here` will time the python line, repeating it a large number of times to improve precision
    - `%bookmark` create *favorite* folders to easily cd into them
    - `%cd` change the current directory
    - `%logstart`/`%logstop` start/stop logging of interactive session and save it to a file
    - `%pycat` similar to `cat` but syntax highlight as python code

Jupyter notebooks

Jupyter

A GUI, served in a browser, to operate on *notebook* style documents: interactive cells where code can be written and executed dynamically.

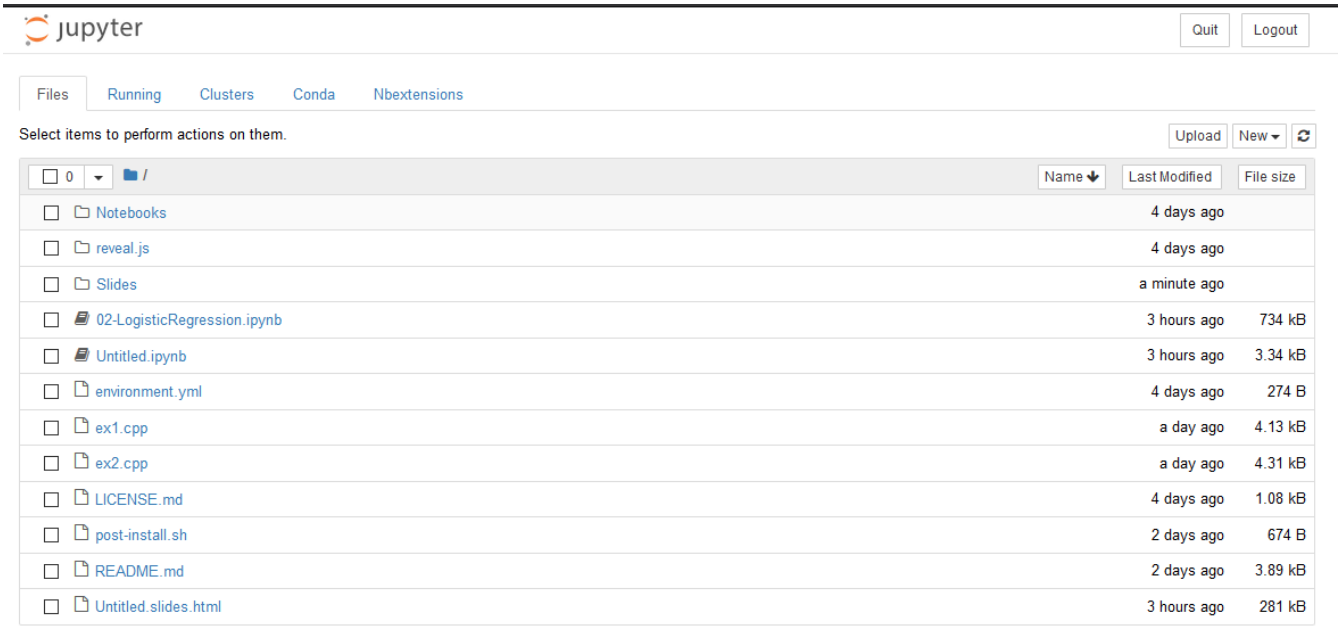
Initially developed for python, now supports many programming languages. The *kernels* run the code (it's a `ipython` interpreter in our case), receive output from the browser input and send back output.

Installation via conda:

```
conda activate <env>
conda install jupyter
#Other useful packages
conda install jupyter_contrib_nbextensions nbconvert nb_conda nb_conda_kernels
```

Start jupyter with:

```
conda activate <env> #If needed
jupyter notebook
```


jupyter
Quit
Logout
Files
Running
Clusters
Conda
Nbextensions
Select items to perform actions on them.
Upload
New
0
/
Name
Last Modified
File size
Notebooks 4 days ago
reveal.js 4 days ago
Slides a minute ago
02-LogisticRegression.ipynb 3 hours ago 734 kB
Untitled.ipynb 3 hours ago 3.34 kB
environment.yml 4 days ago 274 B
ex1.cpp a day ago 4.13 kB
ex2.cpp a day ago 4.31 kB
LICENSE.md 4 days ago 1.08 kB
post-install.sh 2 days ago 674 B
README.md 2 days ago 3.89 kB
Untitled.slides.html 3 hours ago 281 kB

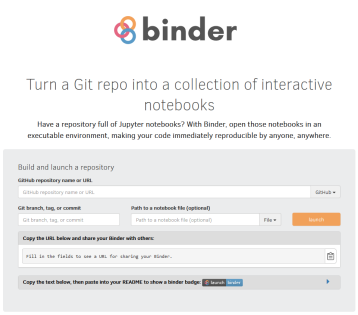
Demo

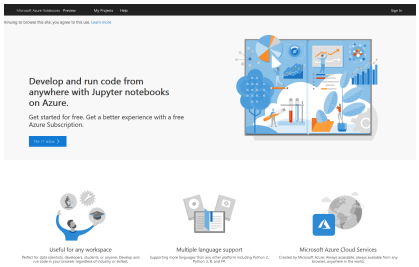
Sharing notebooks

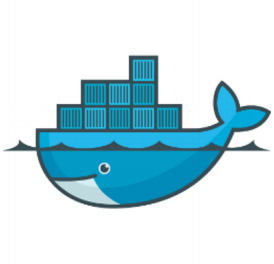
Jupyter is very popular and several ways to share notebooks exist. It should be noted that when a notebook is executed the output of code cells is stored in meta-data, thus it can be rendered:

- Gitlab and github render a notebook as expected: [example (https://gitlab.com/andreadotti/pyalghero2019/blob/master/Slides/Exercise-01-Solution.ipynb)](https://gitlab.com/andreadotti/pyalghero2019/blob/master/Slides/Exercise-01-Solution.ipynb)
- They are based on [nbviewer (https://nbviewer.jupyter.org/)](https://nbviewer.jupyter.org/)

- Online services provide interactive execution of notebooks on premise/cloud resources ([MyBinder (https://mybinder.org/)](https://mybinder.org/), [Microsoft Azure (https://notebooks.azure.com/)](https://notebooks.azure.com/), [Google Colaboratory (https://colab.research.google.com/notebooks/welcome.ipynb)](https://colab.research.google.com/notebooks/welcome.ipynb))







Sharing of notebooks often requires writing and using containers. Check out [this project (https://jupyter-docker-stacks.readthedocs.io/en/latest/index.html)](https://jupyter-docker-stacks.readthedocs.io/en/latest/index.html) if you need them.