Reproducible research

Python

G. Durif (CNRS – IMAG – Univ Montpellier)
Charles Elie Rabier (Univ Montpellier – IMAG)
Christelle Pierkot (CNRS – IR Data Terra)
Sonia Tieo (CNRS – CEFE – Univ Montpellier)

June 23th 2021, Montpellier

Montpellier Bio-Stats (`https://groupes.renater.fr/wiki/montpellier-biostat`)

Outline

1. Intro
2. Backward compatibility problem
3. Management of virtual environments
4. Literate Programming

Intro

Python

Programming language created in 1989 by Guido van Rossum in the Netherlands.
The name Python comes from an homage to the TV series Monty Python's Flying Circus.
The first public version of this language was published in 1991.

- **Python runs on an interpreter system** meaning that code can be executed as soon as it is written $\rightarrow$ prototyping can be very quick.
- **Python works on different platforms** Windows, Mac, Linux, Raspberry Pi, etc
- **Python can be treated in a procedural way, an object-oriented way or a functional way.**

Backward compatibility problem

Python 2 vs Python 3

In February 1991, the first public version of Python (0.9) was made available to the community. In 2000, the first Python 2.0 version is available.
Then in 2008, Python 3.0 and its syntax break is made available to the public along with a new update of the 2.X branch (Python 2.6.1). And since then, the two versions continue to coexist.
After more than 10 years of coexistence, Python 2 is on its way out. Moreover, the python software foundation has announced the end of python 2.7 maintenance for 2020 (`https://www.python.org/dev/peps/pep-0373/`).

Python 2 vs Python 3 : Exemple 1

The function **print**:

```
$ python2
Python 2.7.14 (default, Oct 31 2017, 21:12:13)
[GCC 6.4.0] on cygwin
Type "help", "copyright", "credits" or "license" for more
   information.
>>> x = 5
>>> print "value=", x
value= 5
```

In Python 2, **print** is a command.

Python 2 vs Python 3 : Exemple 1

```
$ python3
Python 3.6.4 (default, Jan  7 2018, 15:53:53)
[GCC 6.4.0] on cygwin
Type "help", "copyright", "credits" or "license" for more
   information.
>>> x = 5
>>> print( "value=", x )
value= 5
```

In Python 2, **print** is a function.

Parentheses become mandatory. Being a function, it can now be used in functions by passing it as a parameter.

Easier to redirect the flow to a file rather than to the console.

Python 2 vs Python 3 : Exemple 2

Division of integers :

```
$ python2
Python 2.7.14 (default, Oct 31 2017, 21:12:13)
[GCC 6.4.0] on cygwin
Type "help", "copyright", "credits" or "license" for more
   information.
>>> print( 2/5 )
0
>>>
```

Python 2 vs Python 3 : Exemple 2

```
$ python3
Python 3.6.4 (default, Jan  7 2018, 15:53:53)
[GCC 6.4.0] on cygwin
Type "help", "copyright", "credits" or "license" for more
   information.
>>> print( 2/5 )
0.4
>>>
```

In Python 2, the division of two integers returns the Euclidean division of these two integers.

In Python 3, we get the result of the floating-point division.

From Python to Python 3

Other exemples:

https://python.sdv.univ-paris-diderot.fr/21_remarques_complementaires/

Porting Guide:

https://docs.python.org/3/howto/pyporting.html

https://portingguide.readthedocs.io/en/latest/

https://py3c.readthedocs.io/en/latest/index.html

Python 3.x

Status of Python branches

Branch	Schedule	Status	First release	End-of-life
master	PEP 619	features	*2021-10-04*	*TBD*
3.9	PEP 596	bugfix	*2020-10-05*	*TBD*
3.8	PEP 569	bugfix	2019-10-14	*2024-10*
3.7	PEP 537	security	2018-06-27	*2023-06-27*
3.6	PEP 494	security	2016-12-23	*2021-12-23*
3.5	PEP 478	security	2015-09-13	*2020-09-13*

Spacing between new versions seems to have stabilized at one year intervals and their lifespan is 5 years.

According to the timeline, all language versions below 3.6 have now reached their end of life.

When and Why Upgrade Python 3.9?

Some troubles:

1. Missing Packages;
2. Many bug fixes;

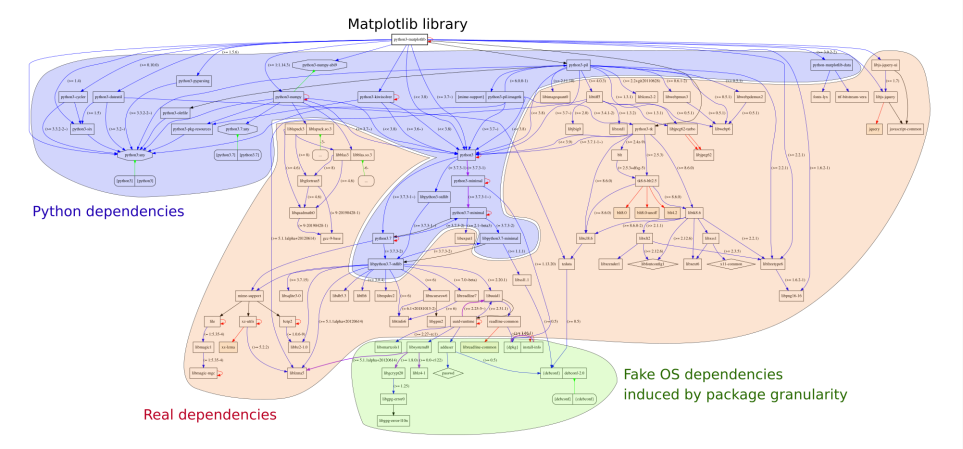
Source: `https://medium.com/analytics-vidhya/when-and-why-upgrade-python-3-9-2b2476daaddb`

Backward Compatibility Problem in the same package

Give exemple (Matplotlib, Pandas...)

Package dependencies

For e.g : Matplotlib - Credit: A.Legrand Talk



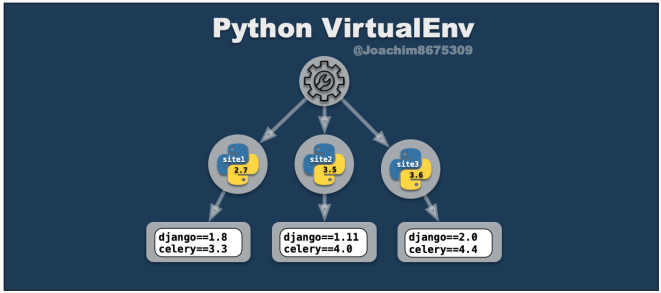
Management of virtual environments

Python virtual env

It can happen that you work on several projects at the same time, each requiring different version libraries. E.g: for a project it will be Django 1.8 and the other Django 2.1 .

So how to work on the same lib but with different versions?

It is possible thanks to the virtualenv package.



Python virtual env

```
# Create the project directory
$ mkdir myproject

# Go into the project's folder
$ cd myproject

# Create a virtualenv named "env"
$ virtualenv env

# Activate the environment using source
$ source env/bin/activate

# Install project dependencies using a requirements file
$ pip install -r requirements.txt
```

Python virtualenv

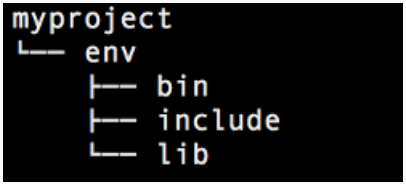
By default, the command takes a single argument: the path to where the environment should be created. Additional options allow for many aspects of the environment to be configured, including the version of Python and whether packages installed on the system should be linked into the environment.

```
# Create a new virtual environment using Python 3
$ virtualenv --python=python3 path/to/env
```

Python virtualenv: Layout and Structure

Several conventions as to where to place environments, but one common approach is to put it alongside the source folder for a program. For example, a project might be checkout out to /myproject/project-src. If following the convention, the environment would be placed at /myproject/env.

- **bin** is where Python and other environment executables are located
- **lib** is where Python packages will be installed



Python virtualenv: Activating the virtual environment

```
# Activate a virtual environment
$ cd ~/myproject
$ source ~/myproject/env/bin/activate
# Check the active version of Python
(env)$ which python
# Output
/home/myuser/myproject/bin/python
```

Freeze environment

```
# Freeze environnement
$ pip freeze > requirements.txt‘
# Install with requirement.txt
$ pip install -r requirements.txt
```

Anaconda Virtual Environment


```
# To create an environment in /envs/. No packages will be
   installed in this environment.
$ conda create --name myenv

# To create an environment with a specific version of Python:
$ conda create -n myenv python=3.6

# To create an environment with a specific package:
$ conda create -n myenv scipy
# -- or
$ conda create -n myenv python
$ conda install -n myenv scipy
```

Anaconda Virtual Environment

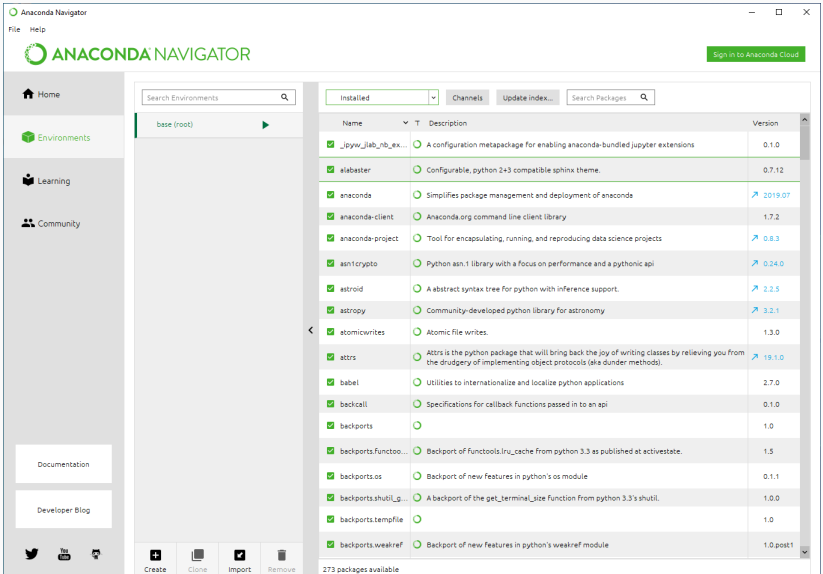

```
# To create an environment with a specific version of Python
   and multiple packages:
$ conda create -n myenv python=3.6 scipy=0.15.0 astroid babel

# To create an environment with yml file
$ conda env create -f environment.yml

# To activate an environment:
$ conda activate myenv

# Export your active environment to a new file:
$ conda env export > environment.yml
```

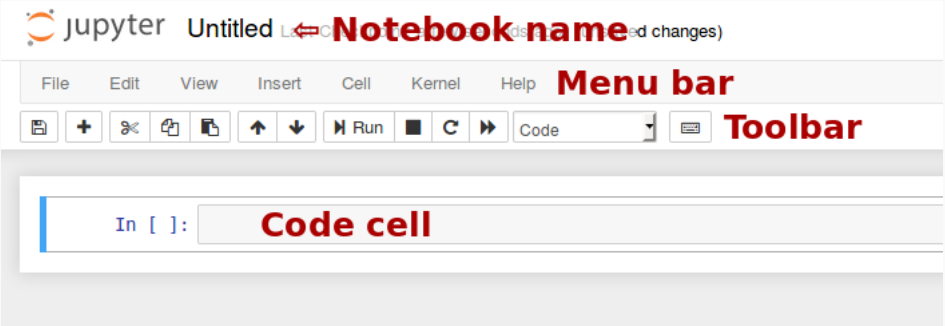
Anaconda Navigator



Literate Programming

Jupyter notebook

https://jupyter.org/



Jupyter notebook

Plotting data and linear model

Text

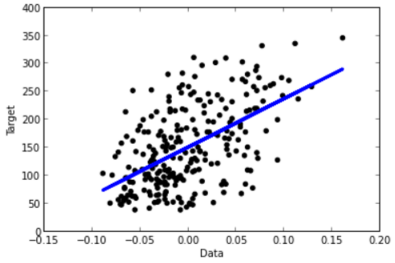
Now we want to plot the train data and teachers (marked as dots).

With line we represents the data and predictions (linear model that we found):

In [14]:
```
# Visualises dots, where each dot represent a data exaple and corresponding teacher
plt.scatter(X_train, y_train,  color='black')
# Plots the linear model
plt.plot(X_train, regr.predict(X_train), color='blue', linewidth=3);
plt.xlabel('Data')
plt.ylabel('Target')
```

Out[14]:
```
<matplotlib.text.Text at 0xb101b0cc>
```

Code



Plot

References