First Steps with Python

Scientific Programming and Data Analysis with Python

Ana Matran Fernandez & Viola Fanfani

Table of contents

Managing Python Packages

Package Managers

All the libraries we have seen until now and with them all those that are needed to run a specific piece of code need to be installed on top of Python.

One can download and install the package directly from source, but considering that we are talking about dozens of libraries that are constantly updated, this approach could lead to multiple problems.

To tackle this issue, Python itself provides a package manager, **pip**, that allows to download, install and update the packages directly from shell.

Package Managers

While pip solves the problem of installing specific versions of Python packages, we might still have a problem with their interactions.

The solution to this issue are virtual environments: a self-contained directory tree that contains a Python installation for a particular version of Python, plus a number of additional packages.

Environments are helpful to keep software updated and consistent: they avoid conflicts between packages, they allow to keep track of updates that could change the behaviour of the code, they make the code portable over multiple machines.

Conda and Anaconda

Conda is an agnostic package manager, non-specific to python, that can take care not only of environments and packages, but also of dependencies.

Miniconda is essentially an installer for an empty conda environment, containing only Conda and its dependencies, so that you can install what you need from scratch.

Anaconda is an open source, easy-to-install high performance Python and R distribution, with the conda package and environment manager and collection of 1,000+ open source packages with free community support.

Conda or pip/virtualenv?

The full Anaconda distribution is a fast and secure way to install almost all the data-analysis packages, provides an easy to use GUI (Navigator) and has the conda environment working below.

Conda is non python-specific and is built to prevent multi-language pipelines from having conflicts.

Pip and virtualenv (the Python environment manager) are flexible publication and distribution platforms for Python packages.

Different goals, different tools...

https://towardsdatascience.com/which-python-package-manager-should-you-use-d0fd0789a250

https://jakevdp.github.io/blog/2016/08/25/conda-myths-and-misconceptions/

Jupyter Notebooks

Jupyter Notebooks

”The Jupyter Notebook is an open-source web application that allows you to create and share documents that contain live code, equations, visualizations and narrative text.”

Jupyter notebooks run code on a python kernel, allowing the user to run python code as if they were on the console. Notebooks are perfect for courses and result displaying: they can incorporate cells of markdown text, they show figures inline, they can be exported in a variety of formats (.pdf, .html ...)

`https://github.com/jupyter/jupyter/wiki/A-gallery-of-interesting-Jupyter-Notebooks#machine-learning-statistics-and-probability`

Jupyter Notebooks

Jupyter resources are extremely popular and, with the proper knowledge, one can create beautiful and powerful Jupyter notebooks.

A few examples to see how diffuse and powerful notebooks are:

Plotting in the Notebook, pyplot introduction

Probabilistic Programming

Signal Processing: filtering

Bokeh quickstart

Jupyter Notebooks

Now, let's get started with the actual lab!

- ☐ open Jupyter Notebook
- ☐ go to the `lab1/` folder, have you copied it into your repo?
- ☐ open the `intro_to_scientific_programming.ipynb` notebook
- ☐ go through the notebook, solve the excercises and don't forget to commit your changes