Starting with Python

1 Installing Python with Miniconda

While a Python interpreter is usually available in Windows and Linux, the Miniconda package offers a simple way to install Python and add and update packages without (usually) having to worry much about issues of package dependencies. To install Python this way ferform the following steps:

- Download the Miniconda installer for your operating system, for instance from Latest Miniconda Installer Links.
- Run the program or shell script you have downloaded. Usually it is enough to accept the default options for a correct installation.
- Check that everything went well by running for instance the command

  `conda update conda`

  to update the `conda` package and installation manager.

With this you have a basic Python installation on your system and you can run

```
conda list | more
```

to get a listing of the installed packages. But probably there will be other packages that you will need and are not installed. A typical example is `numpy` that you can install by running

```
conda install numpy
```

After these commands, if you are working with Linux, you can start writing Python programs with any text editor and running them from the Linux shell command line. If you are working with Windows, listing your applications you should find a folder named `Anaconda 3` or something similar with an program named Anaconda PowerShell Prompt. Running it a console will open where you can again start running Python programs.

The first steps in Python are much easier if you work with notebooks, that combine a Python interpreter, simple text edition and an easy way to run Python programs. Another package you will need to work with notebooks will be JupyterLab, that you can install with

```
conda install -c conda-forge jupyterlab
```

where you are telling conda to get the package from the `conda-forge` channel, a repository with many packages. Once installed, you can run it, for instance, with

```
jupyter lab --no-browser --notebook-dir="D:"
```

This will open a notebook server that you can access from a web browser such as Chrome in which you can access all notebooks hanging from the starting folder given, `D:` here. To access the server, just copy in your browser's url box a line such as

```
http://localhost:8888/lab?token=90a0ed0eeab9142cee4f40769f45d3d554cd383f9a144e64
```

or something similar that you will find in your console after executing the `jupyer lab` command mentioned above.

We give next some indications on working with notebooks.

2 A Jupyter Notebook Environment

First of all, Jupyter notebooks are much more a tool to communicate results than a development environment. However, when starting with Python and dealing with small projects, notebooks can be useful to accelerate first steps and, in any case, they are a tool that every Python programmer should know about.

Here we will describe a minimal notebook environment for programming small projects in Python, Some things we almost always have to do are:

- Tell the notebook to draw pictures.
- Automatically reload some modules we may be working with.
- Move to some directory of our interest or list its contents.

We can do these inside a notebook using IPython's **magic functions** writing in a single cell commands such as:

```
#plot on the notebook
%matplotlib inline

#auotmatic module reloads
%load_ext autoreload
%autoreload 2

#some system commands
%cd D:\practicas_DAA_2017\datos_grafos
%ls
```

Next, we would import the standard modules we will work with and also the modules we are developing; it is also useful to enlarge Python's path with the folders where our own modules may be. This should be done in another notebook cell with commands such as:

```
import sys
import time
import matplotlib.pyplot as plt
import random
import numpy as np

from sklearn.linear_model import LinearRegression

sys.path.append(r"D:\algoritmia\practicas")
import grafos as gr
```

After this we are ready to start working with cells for either code or documentation. In code cells we will

- Edit sentences or functions.
- Execute them with `Ctrl+Intro`.
- Debug, re–edit and re–execute until OK.
- Draw pictures with matplotlib commands.s

Text cells have to be marked as Markdown cells with `Esc+m`. In them we can format text with Markdown syntax for headings, lists and other typesetting actions. They also admit formulas with LaTeX notation

Finally, notebooks can be saved as such, downloaded as plain html files or converted to, say, LaTeX using `nbconvert` (and then, say, to pdf). Their Python code can also be downloaded as a `.py` file. We can find more on Jupyter Notebooks in The Jupyter notebook link.

The Jupyter Notebook interface has a number of useful commands. One particularly handy is

```
Kernel | Restart & Run All
```

The reason is that cells are not stateless and whatever it is executed in one cell affects all others. Cell numbers help to control this and they should always be in consecutive order. A simple way to ensure this is to run `Kernel | Restart & Run All`, that restarts the Python interpreter and runs all cells in consecutive order. If this stops before its end, something was wrong somewhere!

3 Matplotlib and Pyplot

`matplotlib` is a 2D plotting library to generate plots, histograms, power spectra, bar charts, error charts, scatterplots, etc. The `pyplot` submodule combines standard plotting with functions to plot histograms, autocorrelation functions, error bars, ... We import it as `import matplotlib.pyplot as plt`.

Notebooks make plotting and visusalizing figures quite easy. To have the pictures shown inside the notebook we must use the magic command `matplotlib inline`. Basic plotting is done with

```
plt.plot(x, y, str)
```

where `x, y` are arrays or sequences that are plotted one against the other. Single plots are made with `plt.plot(x, str)`. The string `str` controls color and style with many options available; some examples are:

- `'b-'` : solid blue line (solid line is the default)
- `'g--'` : dashed green line
- `'r-.'` : red dash–dot line

In `plot` there can be several array–sequence groups, such as

```
plt.plot(x1, y1, 'g:', x2, y2, 'g-')
```

The following basic commands allow to particularize a plot's elements:

- Title: `plt.title(str)`
- Axis labels: `plt.xlabel('variable %d' % v)`
- Axis limits: `plt.xlim(xmin, xmax), plt.ylim(ymin, ymax)`
- Legends: `plt.legend(handles, labels, loc)` assigns the strings in `labels` to the lines in `handles` and draws them in a position according to `loc`
- Ticks: `plt.xticks, plt.yticks` show x, y axes ticks:

  ```
  plt.xticks(range(len(l_ticks)), l_ticks, rotation=90)
  ```

Finally `plt.show()` displays a plot and `plt.close()` closes it so that the next one is displayed separatedly.

Plots are saved with

```
plt.savefig(fname, dpi=None, orientation='portrait', format=None),
```

where `format` is one of the file extensions supported: `pdf, png, ps, eps, ...`. It can be inferred from the extension in `fname` We can also create figures with subplots with the command

```
plt.figure(num=None, figsize=None, dpi=None, ...)
```

which creates a figure referenced as `num` with width and height in inches determined by the tuple in `figsize` Its basic use is `plt.figure(figsize=(XX, YY))`.

The `subplot` commmand is used to create a subplot within a figure and to refer to that particular subplot. A typical use is

```
subplot(nrows, ncols, plot_number).
```

Here the figure is notionally split in a grid with `nrows * ncols` subaxes and `plot_number` identifies the

current plot in that grid **starting from 1**. If `nrows, ncols, plot_number` are ≤ 9, a 3–digit version can be used such as

```
subplot(311)
```

Finally, as an example of most of the above we have

```
d = { 'x':np.random.rand(100),
      'y': np.random.rand(100)}

plt.figure( figsize = (12, 5) )
plt.subplot(1, 2, 1)
plt.title("%s and %s" % ('x', 'y'))
plt.ylabel("%s, %s" % ('x', 'y'))
_ = plt.plot(d['x'], '*b', label='x')
_ = plt.plot(d['y'], '*r', label='y')
plt.legend(loc='best')

plt.subplot(1, 2, 2)
plt.title("%s vs %s" % ('x', 'y') )
plt.xlabel("x")
plt.ylabel("y")
_ = plt.plot(d['x'], d['y'], '*b', label='x')
plt.show()
```