

# Python Part I - Set-up

*Jean-Yves Sgro*

*February 16, 2017*

## Contents

<b>1</b>	<b>Introduction: Programming with Python 3</b>	<b>1</b>
1.1	python 3 on iMacs . . . . .	2
1.2	Tutorial source and license . . . . .	2
1.3	Tutorial scope . . . . .	2
1.4	Tutorial prerequisites . . . . .	2
1.5	Getting ready: Tutorial files . . . . .	2
<b>2</b>	<b>Starting Python</b>	<b>3</b>
2.1	Finder preferences: . . . . .	3
2.2	Jupyter Notebook . . . . .	4
<b>3</b>	<b>Notebook page</b>	<b>6</b>
3.1	Test that notebook is functional . . . . .	6
3.2	Save your notebook . . . . .	6
3.3	Verify python version . . . . .	6
<b>4</b>	<b>Libraries</b>	<b>7</b>
4.1	platform . . . . .	7
4.2	sys . . . . .	8
4.3	Additional libraries used . . . . .	9
<b>5</b>	<b>References and/or Footnotes</b>	<b>9</b>

## 1 Introduction: Programming with Python 3

There are two main versions of python currently in use:

- python **2** (subversion 2.7)
- python **3** (current subversion 3.6)

There is a “default” version installed for use of the operating system but other versions can be installed independently as well. So this can be very confusing.

For example, on my main Mac, I have the following installed as of April 5, 2016:

- Default “System” python for Mac OS X: version 2.7.10
- Added “independent version” with “brew”: 2.7.11 and 3.5
- Added Anaconda: version 2.7.11 and 3.5.1
- Added Canopy (by Enthought) : version 2.7.9 (version 3 not available - yet.)

Knowing which installation and which version one is using can be confusing. This will entirely depend on how you install your own python on your computer.

*Note:* Unlike most Unix systems and services, Windows does not include a system supported installation of Python.<sup>1</sup>

---

<sup>1</sup><https://docs.python.org/3/using/windows.html>

## 1.1 python 3 on iMacs

While python **2.x** may survive a few years, the transition is towards python **3.x**. Therefore this tutorial will be written for python version 3.x.

I have installed *Anaconda Python 3*<sup>2</sup> on all iMacs in the classroom for the purpose of this workshop.

*Note:* special workshops may be offered to install and use python on your own laptop.

## 1.2 Tutorial source and license

This workshop is based on the Software Carpentry tutorial *Programming with Python*<sup>3</sup>

The workshop is available under the Creative Commons Attribution license<sup>4</sup> and will be adapted for this workshop<sup>5</sup> with identical and edited content.

## 1.3 Tutorial scope

The method of the tutorial is to immediately *dive* into doing something useful: *data analysis*.

We are studying inflammation in patients who have been given a new treatment for arthritis, and need to analyze the first dozen data sets of their daily inflammation.

The data sets are stored in comma-separated values (CSV) format: each row holds information for a single patient, and the columns represent successive days. The first 3 rows of our first file look like this:

```
0,0,1,3,1,2,4,7,8,3,3,3,10,5,7,4,7,7,12,18,6,13,11,11,7,7,4,6,8,8,4,4,5,7,3,4,2,3,0,0
0,1,2,1,2,1,3,2,2,6,10,11,5,9,4,4,7,16,8,6,18,4,12,5,12,7,11,5,11,3,3,5,4,4,5,5,1,1,0,1
0,1,1,3,3,2,6,2,5,9,5,7,4,5,4,15,5,11,9,10,19,14,12,17,7,12,11,7,4,2,10,5,4,2,2,3,2,2,1,1
```

We want to:

- load that data into memory,
- calculate the average inflammation per day across all patients, and
- plot the result.

To do all that, we'll have to learn a little bit about programming.

## 1.4 Tutorial prerequisites

Learners need to understand the concepts of **files** and **directories** (including the **\*working directory**) and how to start a Python interpreter before tackling this lesson.

This lesson references the Jupyter (IPython) Notebook although it can be taught through any Python interpreter. The commands in this lesson pertain to Python 3.

## 1.5 Getting ready: Tutorial files

You need to download some files to follow this lesson:

1. Make a new folder in your **Desktop** called **python-novice-inflammation**.
2. Download python-novice-inflammation-data.zip<sup>6</sup> and move the file to this folder.

---

<sup>2</sup><https://www.continuum.io/downloads>

<sup>3</sup><http://swcarpentry.github.io/python-novice-inflammation/>

<sup>4</sup><https://creativecommons.org/licenses/by/4.0/>

<sup>5</sup><http://swcarpentry.github.io/python-novice-inflammation/LICENSE.html>

<sup>6</sup><https://swcarpentry.github.io/python-novice-inflammation/data/python-novice-inflammation-data.zip>

3. If it's not unzipped yet, double-click on it to unzip it.
4. You should end up with a new folder called **data**.
5. You can access this folder from the Unix shell with:

```
$ cd ~/Desktop/python-novice-inflammation/data
```

*Note:* the individual files as well as the zip file are also available at:  
<https://github.com/swcarpentry/python-novice-inflammation/tree/gh-pages/data>

## 2 Starting Python

Which python would start if you typed `python` at the Terminal command line may depend on local installation. By default on an iMac it would be the default version used by the operating system, currently version 2.7.11 for Mac OS (10.11.5 El Capitan.)

To make sure that we can access **python version 3** we will use the **Anaconda** graphical interface **Navigator** from the installation located on the Mac in `/anaconda`. But this may be invisible to you if it is the first time you log onto this specific Mac. Therefore we need to change the *Finder* preferences of Mac OS.

### 2.1 Finder preferences:

*Note:* If you have previously logged onto this iMac you may not need to do this.

By default nothing will show on the desktop. To change this state of affair, click anywhere on the desktop image and at the very top of the screen, next to the apple logo choose the menu cascade: **Finder > Preferences**

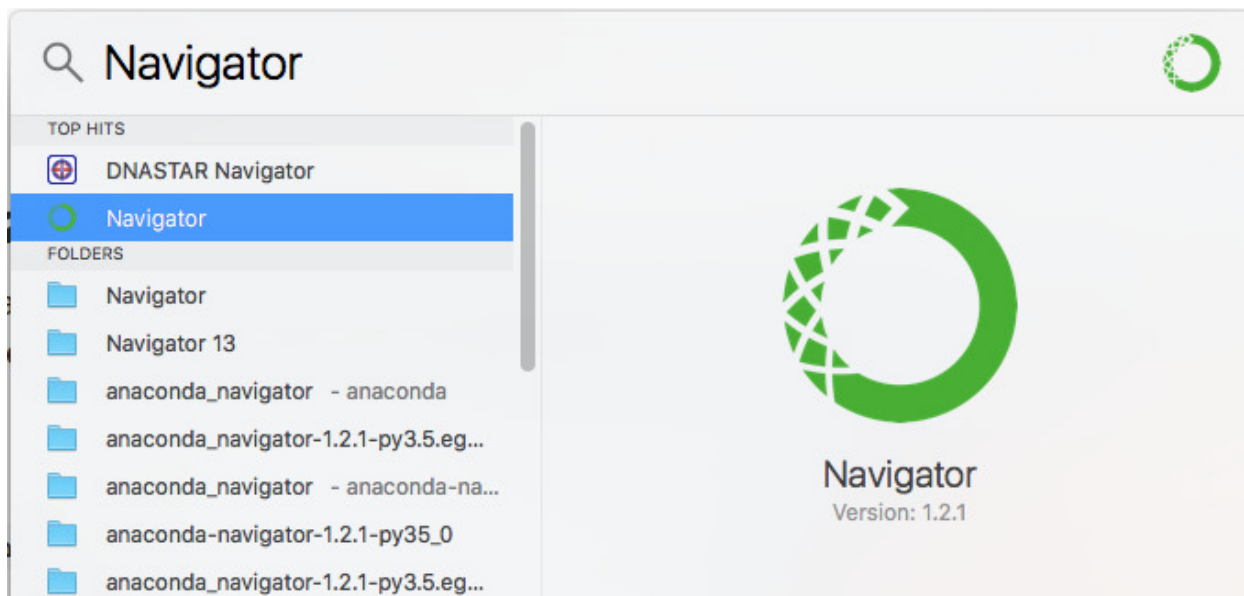
Within the small window that opens check **Hard disks** and **Connected server** which are by default unchecked. Now you will see the hard drive icon of the Mac appear at the top right of the desktop.

#### 2.1.1 Open Navigator: Method 1

- Double click the hard drive icon
- Double click on `anaconda`
- Double click on `Navigator`

#### 2.1.2 Open Navigator: Method 2

An alternate method to open the `Navigator` software is to use “Spotlight Search” that looks like a magnifying glass : Click the magnifying glass at the top right corner of your screen and type `Navigator` then press the `return` key.



Finding *Navigator* using *Spotlight*.

Figure 1.

*Note:* The newest version of the Anaconda Navigator is now called `anaconda-navigator`.

### 2.1.3 Trouble shooting at start-up

If Navigator seems to start but then immediately quit there might be a quick-fix for this problem: remove directories called `.anaconda` and/or `.continuum/` form your home directory:

- open a Terminal
- go to your home directory: `cd ~`
- remove the directories: `rm -rf .anaconda` and/or `rm -rf .continuum`

WARNING: the `rm -rf` command is UNDOABLE so make sure you type things correctly!

If the problem persists, then the administrator of the computer may have to intervene.

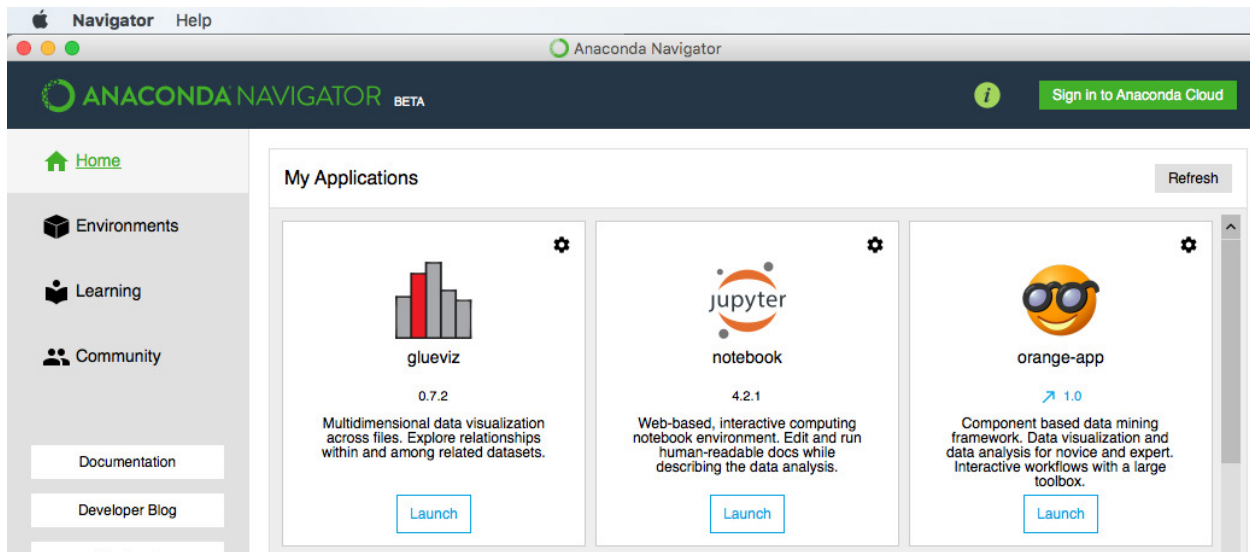
## 2.2 Jupyter Notebook

*Note:*Jupyter notebook is the new name of `ipython notebook`.

`Navigator` will open a large window with choice options.

For ease of use we will use the “notebook” method to enter python commands.

- Click on ‘Launch on the “Jupyter Notebook” panel



Click **Launch** on the **jupyter notebook** square.

Figure 2.

This will trigger a series of events:

- a text Terminal will open, don't interfere with that...
- Within the terminal a background local web server will run
- a web browser will open or a new window will open

The web browser will be used to navigate to the directory we want to use, and will also be the interface containing the Jupyter notebook session.

- The default URL will read `localhost:8888/tree`
- You can always navigate back to this URL by clicking on the little "home" button
- The content of your username will be listed
- Click on **Desktop**
- Click on `python-novice-inflammation` (created above)
- Click on `data` (added above from zip file)

The current directory which should be `'home logo' /Desktop/python-novice-inflammation/data` and you should see the data files `inflammation-xx.csv` and `small-0x.csv` in the page.

### 2.2.1 Create a new notebook for this session

It is now time to create a new notebook for typing commands.

On the right hand side of the Jupyter Notebook click on the pull down menu **New** and select **Python 3** or **Python [Root]** depending on the installation made.

This will create a new python notebook i.e. a new page in the browser running under python 3 within the current directory. The page is contained within a new web browser "TAB" called "**Untitled**" and will be given a name when we save it.

On the page should be a gray box preceded with `In [ ]:` which is the Python "prompt" to tell us that it's ready to receive a command.

We are now ready to start!

## 3 Notebook page

### 3.1 Test that notebook is functional

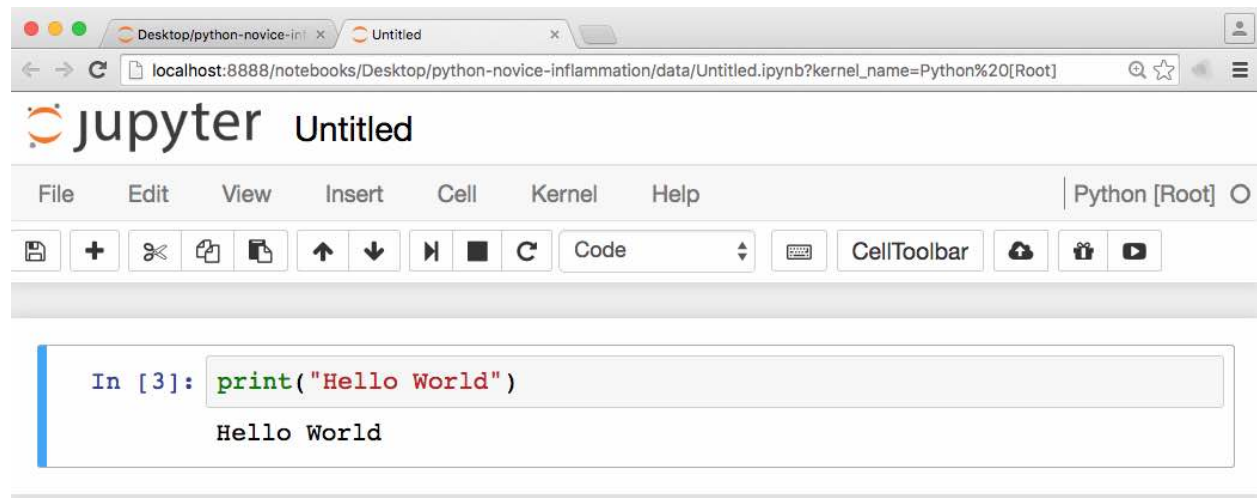
To make sure everything is in working order we'll write the simplest, most common program called `Hello World!`

Write the following command within the gray box line preceded by `In [ ]:` and press “**Control-Return**” (both keys at the same time) or press the step-forward button .

```
print("Hello World!")
```

Hello World!

At this point your notebook page should look something like this snapshot image:



Your **jupyter notebook** should be similar to this.

Figure 3. ipython notebook

*Note:* You may notice that there is a number 3 within the `In [3]` which simply means that this command was issued 3 times before the snapshot was taken. In other words, this number will increase by one each time the command is run. This may be important later.

### 3.2 Save your notebook

It is now time to save this “Untitled” notebook.

The notebook(s) are saved automatically by the Jupyter server with the default of “Untitled”.

There is no “Save As...” and to save this notebook we need to use the menu cascade:

**File > Rename...**

and give a name, for example **notebook1**. The file will be saved within the directory that we chose earlier and a filename extension will be added, therefore the final file will be named **notebook1.ipynb**.

### 3.3 Verify python version

As stated above there are many reasons why one should check the version of python we are running. There are also many ways to run python: we are running it within a python notebook, but in other cases it could

be on a Terminal and there are other options not described here.

### 3.3.1 Terminal

To test the python version running **within a Terminal** you could issue this command at the **bash** shell prompt `$`. This would be the result on the Mac Terminal under El Capitan currently:

```
/usr/bin/python -V
```

```
Python 2.7.10
```

*Note:* the absolute PATH is specified rather than just calling `python` to make sure we are calling the default system installed python.

If we just call `python` we might get a different result, and indeed we do!

```
python -V
```

```
Python 2.7.10
```

### 3.3.2 Your notebook

On **your notebook** we now want to add a cell for the next command. If you used the method given above of pressing “**Control-Return**” together the code executed but there is no new “cell” to write a new command.

If you pressed the step-forward button you already have a new cell.

If you used “**Control-Return**” you can add a new cell by pressing the large plus sign `+`, or, alternatively, use the menu cascade:

**Insert > Insert Cell Below**

We can now ask the question about which python version we are running within the notebook. We should theoretically run version **3** as stated in the introductory section.

How do we ask the question?

We need a command that will give us the answer!

The answer is “system library” (see below.)

## 4 Libraries

As other software usually do, python also can “load” external libraries.

The command to import a library is **import** followed by the name of the library. The caveat is that you should know what that name is!

Libraries may be built in the operating system installation or part of the python installed but not loaded when python is started. Other libraries such as libraries containing mathematical formulaes must be installed before they can be used. The installation method depends strongly on the python software used.

Here are 2 examples of libraries readily available: **sys** derived from the operating system and **platform**

### 4.1 platform

Within a new “cell” enter the following commands and either click the step-forward button or press “**Control-Return**” together.

We will add a “module” called `platform` temporarily to augment the python software. This addition is not permanent and is required each time the commands are needed, but should normally be called only one time at the top of a list of commands.

```
import platform
print(platform.python_version())
```

#### 2.7.10

The python version is available within the `platform` package and printed out. It is printed out as a set of numbers separated by dots.

The major version is the first number. We’ll explore below in the `sys` section how to request the value for each of the numbers separately.

*Note:* The word `print()` may or may not be required. **Test it out!**

(It is required within the method I use to create this document -*RStudio*- but does not seem required within an *ipython notebook* document.)

## 4.2 sys

We will now add another “module” called `sys` that contains information and commands related to the “system.”

Within a new “cell” enter the following commands and either click the step-forward button or press “**Control-Return**” together.

*Note:* it should be necessary to call the `import()` command only once in your session or in your *ipython notebook* document and it should be “remembered” for the remainder of the session. I have to keep the `import()` command each time below due to the software I use to create this document.

The command `sys.version` will print a large number of information that include all major and minor numbers of the python version, the software name (**Anaconda**) and its version number, with calendar dates of installation, and compilation compatibility and more.

```
import sys
print(sys.version)
```

```
2.7.10 (default, Oct 23 2015, 19:19:21)
[GCC 4.2.1 Compatible Apple LLVM 7.0.0 (clang-700.0.59.5)]
```

The command `sys.version_info` will print the *major*, *minor*, and *micro* versions of the python software that should match those revealed by the command `platform.python_version()` issued above.

```
import sys
print(sys.version_info)
```

```
sys.version_info(major=2, minor=7, micro=10, releaselevel='final', serial=0)
```

Before we start with the software carpentry exercises, let’s look at a little program gleaned online on stack overflow.<sup>7</sup>

If you don’t “get it” this first time it’s OK, we’ll learn about this with the exercises.

This little program does the following:

- Line 1: import `sys` as we did above. If you are continuing in the same session it may not be required this time.
- Line 2: a comment line defining the purpose of the program.

<sup>7</sup><http://stackoverflow.com/questions/446052/how-can-i-check-for-python-version-in-a-program-that-uses-new-language-features>



- Line 3: defining a *variable* named `python_version` (a name “made up” by the author) that will contain the value given by yet another command: `sys.version_info.major` which will *extract* the value (number) of the python version. From the above data we are expecting this number to be 3.
- Line 4: This is a conditional “IF” statement, testing if a proposition is true or false. If true it continues on the next line, if false it will jump to the ELSE statement.
- Line 5: execute the `print()` statement if the above line is true
- Line 6: Alternate option if the “IF” statement 2 lines above is false, in that case execute the next command on the next line.
- Line 7: execute the `print()` statement if the above line (ELSE) was activated.

The result will depend on the active python on your system, but should be 3 today.

```
import sys
# prints whether python is version 3 or not
python_version = sys.version_info.major
if python_version == 3:
    print("is python 3")
else:
    print("not python 3")
```

not python 3

*Caveat:* How to know which library to import in Python? There is no simple answer to that in spite of many resources that can help find if a module is available. (<sup>8</sup>,<sup>9</sup>)

### 4.3 Additional libraries used

In addition to some of the default libraries which are installed with Python, we will use the following libraries that need to be installed.

Installation for Anaconda is done by bash shell commands on a terminal:

```
conda install numpy
conda install matplotlib
```

If you are using a different Python installation you may need to use the `pip` shell program. See for example “Installing Python Modules.”<sup>10</sup>

## 5 References and/or Footnotes

<sup>8</sup><http://stackoverflow.com/questions/14050281/how-to-check-if-a-python-module-exists-without-importing-it>

<sup>9</sup><https://docs.python.org/3.6/tutorial/modules.html>

<sup>10</sup><https://docs.python.org/3.6/installing/index.html>