Ludovic Dutoit, September 2016

Tutorial: A numerical Python

This tutorial is a simple introduction into the word of vectors and numbers in python (i.e. arrays). In the second part we will focus on the plotting possibilities in python using the *matplotlib.pyplot* submodule. We will limit ourselves to the basics of arrays and their properties using *numpy* so that we can spend some time plotting!

Write, do not copy/paste!
Look at your output, think at what the computer did.

Numpy

Numpy stands for numerical python, it has many functions that are useful to scientists. Let's first import it as *np* so that everytime we want to call something in *numpy* we do not have to type "*numpy.*" but just *"np."*.

```
import numpy as np # WRITE,  do not copy paste
```

Numpy's main object is the homogeneous multidimensional array. It is a table of elements (usually numbers), all of the same type.

```
a = np.arange(15)
a  #look at it
```

arange works like the basic function range. Try to create an array *c* from 10 to 20 every 2, so that you get 10,12...20.

Another useful function is the function *linspace(start, stop, num)* which creates an array from *start* to *stop* with *num* values evenly spaced. Try to create a vector *b* from 1,10 with 100 values.

What if you wanted 0,0.1,0.2,0.3? You would have needed 101 values. Achieving given intervals is much easier with *arange* which behaves like *range(start,stop,interval)*. Try to use *range* to get a list from 0 to 10 by 0.1.

You can always take that list and transform it in an array.

```
l = [1 ,5 ,7 ,3 ,4]
a = np.array(l)
```

You may want to define an array from scratch, but you have to include brackets because they are useful to define dimensions.

```
a = np.array(1,2,3,4)    # WRONG
a = np.array([1,2,3,4])  # RIGHT
```

Why do you need the brackets? Because using brackets you can also create a multidimensional array. Let's try 3D by creating 1 big array with 3 small arrays inside.

```
a = np.array([ [1, 2, 3, 4], [2, 6, 8, 6, 4], [1, 5, 7, 8, 9] ])
a
```

What differentiates an *array* from a *list*? Try a few basic operations on an array and on a list from one to ten, operations like +,-,/,*:

```
a = np.arange(1,10)
l = range(1,10)
```

One very useful property of *numpy* is the ability to create arrays that come from distributions.

Try to understand the functions *np.random.rand, np.random.uniform or np.random.normal*. There are many more! They will be useful later today when we try to make plots!

1. Try to make a normal sample of 1000 values of a normal distribution with mean 1 and standard deviation 2.
2. Try to make a sample of values distributed in a uniform manner between 5 and 15.

Try a few more functions of *numpy* using np.≤TAB> to see what there is around!

Pyplot

Pyplot is a submodule of *matplotlib*. It allows you to easily create high definition figures for your Publications in many formats!
Pyplot is actually closely modeled after Matlab. It has a lot of documentations and many options.

Basic Figures

Let's import *matplotlib.pyplot* as *plt*.

```
import matplotlib.pyplot as plt
```

Let's define easy x and y values,

```
x = range(1, 100, 2)
```

Now that you have *x*, define two different y variables:
y1 should be the squares of *x*.
y2 should be its cubic values.

Try to define both *y1* and *y2* using list comprehensions first. Once you have done it, try recreating *y1* and *y2* transforming *x* into a *numpy* array first and then using the square "**2" and cubic "**3" operations.

Time for our first plot:

```
plt.plot(x, y1)
# It does not show up immediately? No, you do need to call plt.show()
plt.show()
```

You can play with the graphical interface to modify some parameters but also to save the plot. Save it to see how it looks. Now you can see that your python command line is blocked into this "show" mode. You have to exit it using CTRL+C.

You probably do not want to do that every time. You can work with plt.ion() "plt interactive mode on"

```
plt.ion()
plt.plot(x,y1) # look at your plot
plt.plot(x,y2) # look at your plot again
```

Did you see that? By calling plot again we added a line, we did not create a new plot! The *y axis* dynamically adjusted to include all the cubic values. Something R will not do.

In general, *matplotlib* has great default settings, but let's see if we can do more than that!

```
?plt.plot
```

There are many parameters, let's first close our last plot using *plt.close()* or the closing button on the figure window.Let's try a few more parameters

```
plt.plot(x, y2, "b>")  #b stands for blue and > for triangles. It goes just after x and y
plt.plot(x, y1, "ro",markersize=10)# we add big red (r) circles (o)
```

Close your plot. Let's explore more attributes, try to make a plot with red plus markers with greenmarker edge and quite large markers.

```
plt.plot? # some help
```

Close your plot. Let's make a legend. Here we made several lines of codes for you. Go through them one by one and ask yourself if you understand them.

```
plt.plot(x,y1,"r-",label="size1") # here we give labels to our line
plt.plot(x,y2,"m--",label="size2")# a dashed blue line
plt.legend(loc = "upper left",frameon=False)# we the call plt.legend on the plot
```

What is the *frameon* argument about? And *loc*? Try changing them!

Let's also see what we can do with axes:

```
plt.xticks(range(0, 100, 10)) # x ticks from 0 to 100 every 10
plt.yticks([]) # remove y ticks
plt.yticks([0, 1000000], ["A little", "A lot"], size = 20) # add two y ticks with given texts
plt.xlabel("x label", fontname ="Comic Sans MS") # add a x label with a special font
plt.ylabel("y label",size=20) # add a big y label
plt.ylim(-1e5,1.2e6) #change the limits of the yaxis; python supports this numbers as well
plt.ylimits(-100,10000000)
```

You may have noticed that for axes parameters, they replace each other not like the data that you can keep adding!
Let's now save the plot, you can do it in many different formats.

```
plt.savefig("Figure1.pdf",dpi=300) # dpi is a resolution (“dots per inches”)
plt.savefig("Figure1.png") # Python adapts the format from the extension
```

Different plots

Now these plots were not a lot of fun but it got us familiar with the way pyplot works! Let's try to make some more interesting plots.

Barplot

A classic plot is a barplot. The plot below is a barplot representing the percentage of survivor within every class when titanic sank.

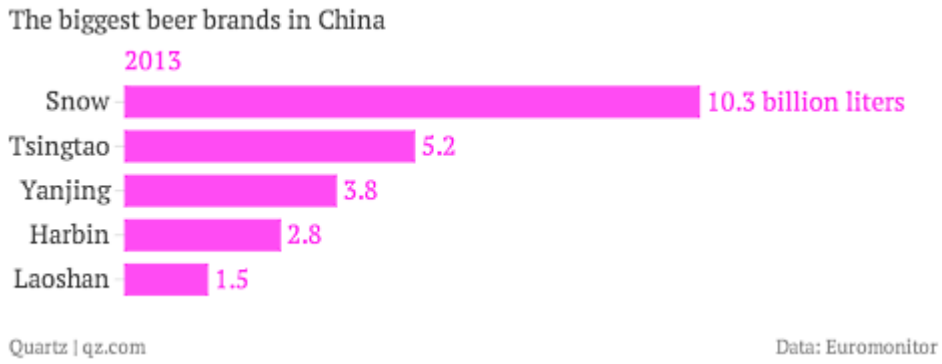
categories = ('1st Class', '2nd class', '3rd Class',"Crew" ,'Total')
survivors = (60.5,41.7,24.5,23.8,31.8) # survivors percentages per class

```
plt.bar(range(5), survivors, color="black",align="center") #look at barplot help!

plt.xticks(range(5), categories)# add xticks with names per category
plt.xlabel('Category')# add an xlabel
plt.ylabel("Percentage that survived")# add a ylabel
plt.xlim(-1,5 ) # recenter the x axis to make it look nice
```

Exercise

We would like you to bring attention to the beer market in China in 2013. Here is an unattractive barplot. Try to make a nice one using python. Use at least one function we did not use in our example above!



Pie chart

Another classical example is a pie chart:

```
labels = ['Pizzas', 'Wheel of fortunes', 'Pies' ] # labels for categories
sizes = [0.3, 0.4,  0.3] # proportiob by category or %
colors = ['yellowgreen', 'red', 'lightblue']
explode = (0.1, 0, 0)  # only "explode" the 1stnd slice
plt.pie(sizes, explode=explode, labels=labels, colors=colors,
shadow=True, startangle=90) # make a sadow, incline the plot
plt.title("Things that look like pie charts")
```

Exercise

Look at “funny pie charts” on google images, find one that is good enough for you and make it. Again, try to use one function we did not use in our example.

Scatter plot

Let’s look at a classic scatter plot.

```
N = 50 # number of observations
x = np.random.rand(N) # extract 50 values between 0 and 1
y = np.random.rand(N) # extract 50 values between 0 and 1
colors = np.random.rand(N) # colors also  have values between 0 and 1 in python
area = np.pi * (15 * np.random.rand(N))**2  # 0 to 15 point radiuses
plt.scatter(x, y, s = area, c=colors, alpha = 0.8) # create the scatterplot
```

There are many more possibilities using *pyplot*. If you have time left, combine the knowledge you got these 3 last days to read some of our own data and make it into a python plot. If you want some inspiration, look at the *pyplot* library, which contains dozens of plots and their associated code, and maybe try a few:
http://matplotlib.org/gallery.html