Python for scientists

Lesson 9

NumPy, SciPy and Matplotlib

```
def complementary(seq):
    nt_comp = {
        'A' : 'T',
        'C' : 'G',
        'G' : 'C',
        'T' : 'A',
    }
    for nt in seq:
        compseq += nt_comp[nt]
    return compseq
```

Alvaro Sebastián Yagüe

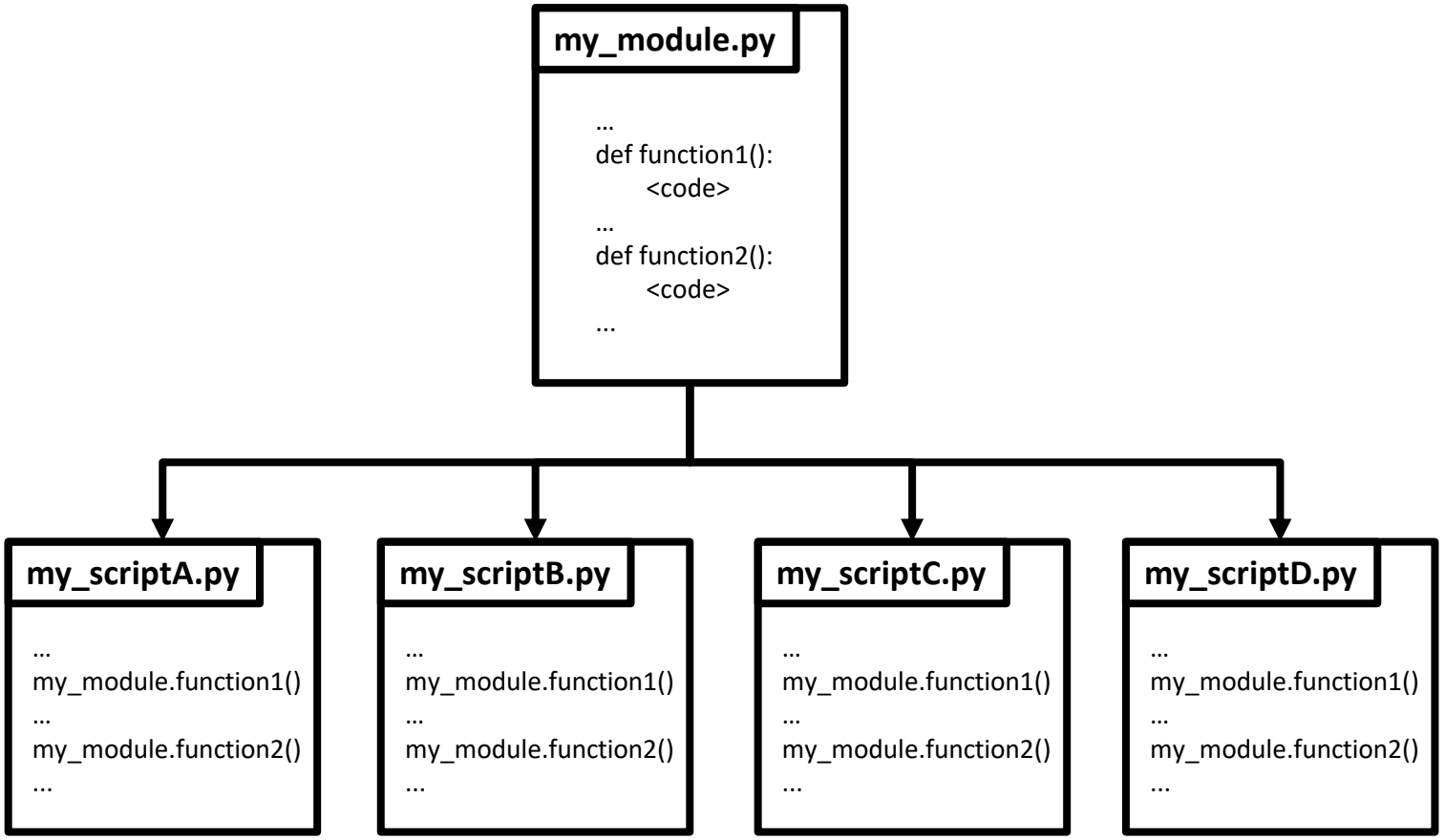
www.sixthresearcher.com

@SixthResearcher

Remember...

Python modules

A module is a Python file containing functions and variables used by multiple scripts.



Python modules

To use a module we have to import it with the 'import' command at the beginning of the script:

```
>>> import math
```

Functions included in the module can be called with the module name plus '.' (dot) and the function name:

```
>>> print('π:', math.pi)
π: 3.141592653589793
>>> print('e:', math.e)
e: 2.718281828459045
>>> values = [ 1, 2, 3, 4, 5 ]
>>> print('Sum of values:',math.fsum(values))
Sum of values: 15.0
>>> print('Factorial of 5:',math.factorial(5))
Factorial of 5: 120
>>> print('Square root of 5:',math.sqrt(5))
Square root of 5: 2.23606797749979
```

NumPy

NumPy

A Python package (or library) contains many modules which are separated by their uses.

NumPy is a Python package that adds support for large, multi-dimensional arrays and matrices, along with a large library of high-level mathematical functions to operate on these arrays.

```
In [1]: import numpy

In [2]: matrix = numpy.array([[ 0,  1,  2,  3,  4],
                              [ 5,  6,  7,  8,  9],
                              [10, 11, 12, 13, 14]])

In [3]: print(matrix)
[[ 0  1  2  3  4]
 [ 5  6  7  8  9]
 [10 11 12 13 14]]

In [4]: type(matrix)
Out[4]: numpy.ndarray

In [5]: matrix.shape
Out[5]: (3, 5)

In [6]: matrix.size
Out[6]: 15
```

‘matrix’ is a multidimensional array object

https://docs.scipy.org/doc/numpy-dev/user/quickstart.html

NumPy - basic operations

```
In [2]: A = numpy.arange(15).reshape(3,5)
        print(A)

        [[ 0  1  2  3  4]
         [ 5  6  7  8  9]
         [10 11 12 13 14]]

In [3]: A.min()

Out[3]: 0

In [4]: A.max()

Out[4]: 14

In [5]: A.sum()

Out[5]: 105

In [6]: numpy.exp(A)

Out[6]: array([[  1.00000000e+00,   2.71828183e+00,   7.38905610e+00,
                  2.00855369e+01,   5.45981500e+01],
               [  1.48413159e+02,   4.03428793e+02,   1.09663316e+03,
                  2.98095799e+03,   8.10308393e+03],
               [  2.20264658e+04,   5.98741417e+04,   1.62754791e+05,
                  4.42413392e+05,   1.20260428e+06]])

In [7]: numpy.sqrt(A)

Out[7]: array([[ 0.        ,  1.        ,  1.41421356,  1.73205081,  2.        ],
               [ 2.23606798,  2.44948974,  2.64575131,  2.82842712,  3.        ],
               [ 3.16227766,  3.31662479,  3.46410162,  3.60555128,  3.74165739]])
```

https://docs.scipy.org/doc/numpy-dev/user/quickstart.html

NumPy - basic operations

```
In [2]: A = numpy.array([[ 1, 2],
                         [ 3, 4]])

In [3]: B = numpy.array([[ 5, 6],
                         [ 7, 8]])

In [4]: A*B # Elementwise product

Out[4]: array([[ 5, 12],
               [21, 32]])

In [5]: numpy.dot(A,B) # Matrix product

Out[5]: array([[19, 22],
               [43, 50]])

In [6]: numpy.add(A,B) # Matrix Addition

Out[6]: array([[ 6,  8],
               [10, 12]])

In [7]: B += A # Adds A to B
        print(B)

        [[ 6  8]
         [10 12]]
```

https://docs.scipy.org/doc/numpy-dev/user/quickstart.html

SciPy

SciPy – polynomial functions

SciPy is an open source Python package used for scientific and technical computing.

SciPy contains modules for optimization, linear algebra, integration, interpolation, special functions, FFT, signal and image processing, ODE solvers and other tasks common in science and engineering.

```
In [1]: import scipy

In [2]: p = scipy.poly1d([3,4,5])
        print(p)

        2
3 x + 4 x + 5

In [3]: print(p*p) # Polynomial product

   4      3      2
9 x + 24 x + 46 x + 40 x + 25

In [4]: print(p.integ(k=6)) # Integration

   3     2
1 x + 2 x + 5 x + 6

In [5]: print(p.deriv()) # Differentiation

6 x + 4

In [6]: p([1,2,3,4]) # Evaluation for x=1,2,3,4
Out[6]: array([12, 25, 44, 69])
```

'p' is a 1D polynomial object

http://www.tau.ac.il/~kineret/amit/scipy_tutorial/

SciPy – integration

$$\int_0^1 x^2\,dx = \frac{1}{3}$$

```
In [1]: from scipy.integrate import quad

In [2]: def integrand(x):
            return x**2

In [3]: ans, err = quad(integrand, 0, 1)

In [4]: print(ans)
0.33333333333333337
```

$$\int_{x=\pi}^{2\pi} \int_{y=0}^{\pi} y\sin(x) + x\cos(y)\,dy\,dx = ?$$

```
In [1]: from scipy.integrate import dblquad
        import numpy

In [2]: def integrand(y, x):
            return y * numpy.sin(x) + x * numpy.cos(y)

In [3]: ans, err = dblquad(integrand, numpy.pi, 2*numpy.pi,
                           lambda x: 0,
                           lambda x: numpy.pi)

In [4]: print(ans)
-9.869604401089358
```

Matplotlib

Matplotlib

Matplotlib is a plotting library for Python. It provides both a very quick way to visualize data from Python and publication-quality figures in many formats.

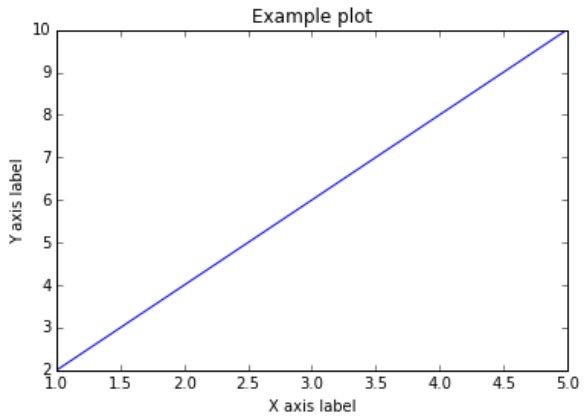
To activate Matplotlib sessions in Jupyter (IPython) with Matlab/Mathematica-like functionality, we use the **magic command: '%matplotlib inline'**.

```
In [1]: %matplotlib inline

In [2]: from matplotlib import pyplot

In [3]: pyplot.plot([1,2,3,4,5],[2,4,6,8,10])
        pyplot.title('Example plot')
        pyplot.xlabel('X axis label')
        pyplot.ylabel('Y axis label')

Out[3]: <matplotlib.text.Text at 0x5fcd4b0>
```



http://matplotlib.org/users/pyplot_tutorial.html

Matplotlib - functions

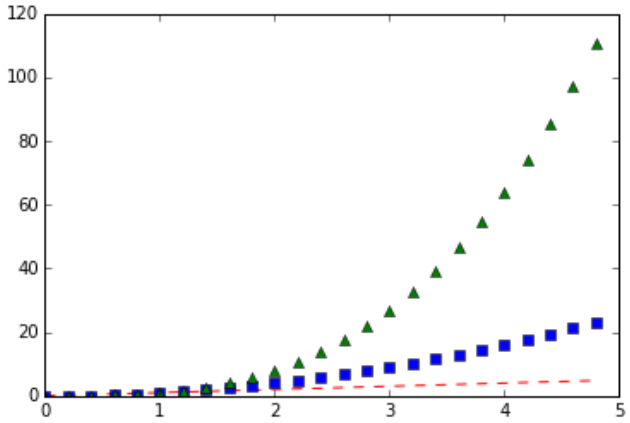
In [1]:
```
%matplotlib inline
```

In [2]:
```
import numpy
from matplotlib import pyplot
```

In [3]:
```
# Sample x from 0 to 5 at 0.2 intervals
x = numpy.arange(0, 5, 0.2)
```

In [4]:
```
# Plot linear function as red dashes
pyplot.plot(x, x, 'r--')
# Plot quadratic function as blue squares
pyplot.plot(x, x**2, 'bs')
# Plot cubic function as green triangles
pyplot.plot(x, x**3, 'g^')
```

Out[4]:
```
[<matplotlib.lines.Line2D at 0x5ff2a30>]
```



http://matplotlib.org/users/pyplot_tutorial.html

Matplotlib - functions

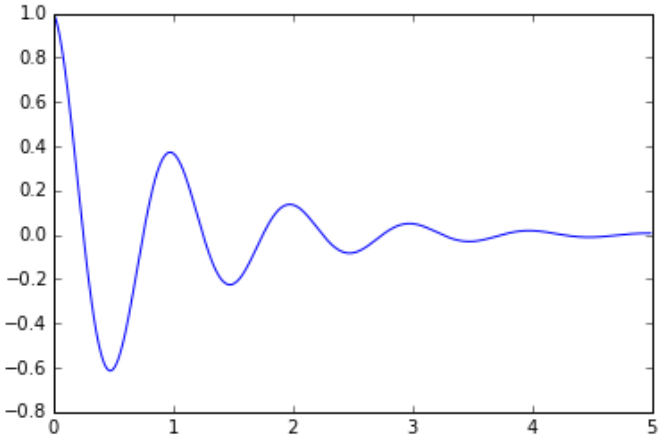
```
In [1]: %matplotlib inline

In [2]: import numpy
        from matplotlib import pyplot

In [3]: def f(t):
            return numpy.exp(-t) * numpy.cos(2*numpy.pi*t)

In [4]: t = numpy.arange(0.0, 5.0, 0.01)

In [5]: pyplot.plot(t, f(t))
Out[5]: [<matplotlib.lines.Line2D at 0x5ff4e90>]
```



$$e^{-t}\cos 2\pi t$$

http://matplotlib.org/users/pyplot_tutorial.html

Matplotlib – bar plots

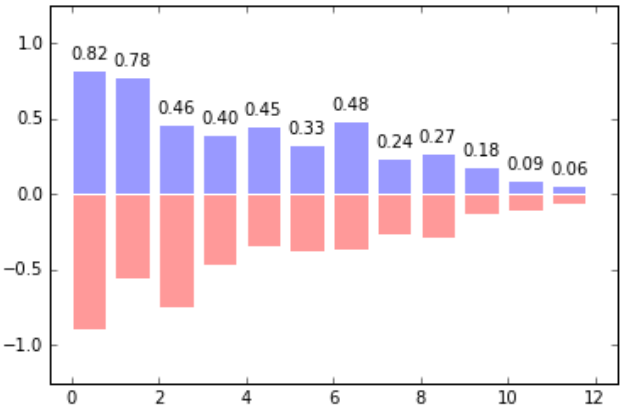
In [1]:
```
%matplotlib inline
```

In [2]:
```
import numpy
from matplotlib import pyplot
```

In [3]:
```
n = 12
X = numpy.arange(n)
Y1 = (1-X/float(n)) * numpy.random.uniform(0.5,1.0,n)
Y2 = (1-X/float(n)) * numpy.random.uniform(0.5,1.0,n)
```

In [12]:
```
pyplot.bar(X, +Y1, facecolor='#9999ff', edgecolor='white')
pyplot.bar(X, -Y2, facecolor='#ff9999', edgecolor='white')
for x,y in zip(X,Y1):
    pyplot.text(x+0.4, y+0.05, '%.2f' % y, ha='center', va= 'bottom')
pyplot.xlim(-0.5,+12.5)
pyplot.ylim(-1.25,+1.25)
```

Out[12]: (-1.25, 1.25)



https://www.labri.fr/perso/nrougier/teaching/matplotlib/

Matplotlib – contour plots

In [1]:
```
%matplotlib inline
```

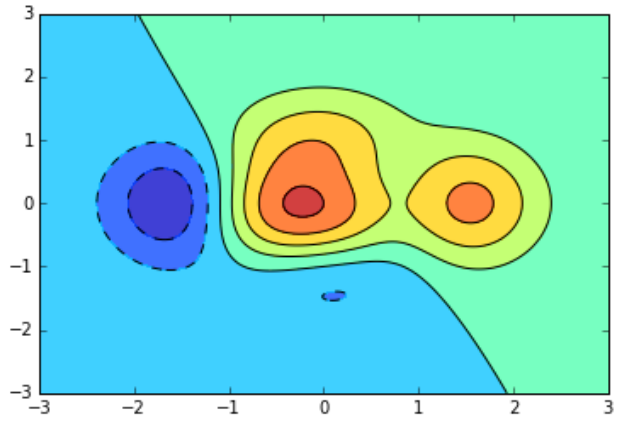
In [2]:
```
import numpy
from matplotlib import pyplot
```

In [3]:
```
def f(x,y): return (1-x/2+x**5+y**3)*numpy.exp(-x**2-y**2)
```

In [4]:
```
n = 256
x = numpy.linspace(-3,3,n)
y = numpy.linspace(-3,3,n)
X,Y = numpy.meshgrid(x,y)
```

In [6]:
```
pyplot.contourf(X, Y, f(X,Y), 8, alpha=.75, cmap='jet')
pyplot.contour(X, Y, f(X,Y), 8, colors='black', linewidth=.5)
```

Out[6]:
```
<matplotlib.contour.QuadContourSet at 0x615d370>
```



https://www.labri.fr/perso/nrougier/teaching/matplotlib/

NumPy+SciPy +Matplotlib

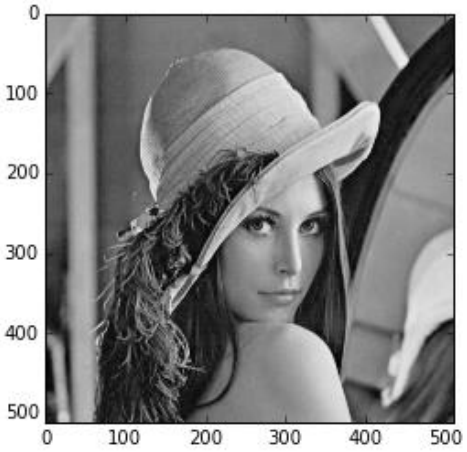
NumPy+SciPy+Matplotlib

```
In [1]: %matplotlib inline

In [2]: import numpy
        from scipy import misc
        from scipy import ndimage
        from matplotlib import pyplot

In [3]: lena = misc.lena()

In [4]: pyplot.imshow(lena, cmap=pyplot.cm.gray)
Out[4]: <matplotlib.image.AxesImage at 0x9efcfb0>
```

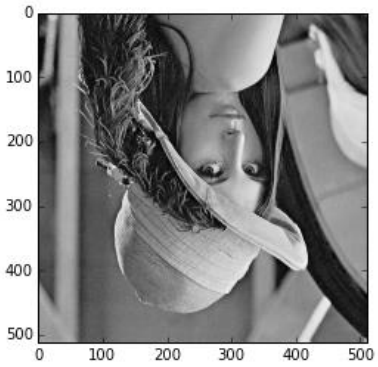


http://claudiovz.github.io/scipy-lecture-notes-ES/advanced/image_processing/index.html

NumPy+SciPy+Matplotlib

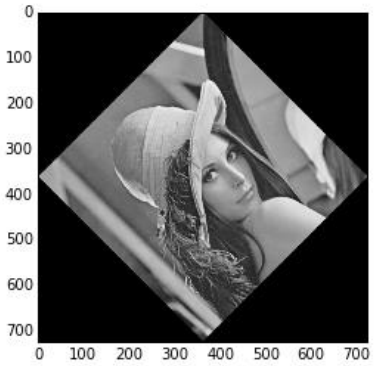
```
In [5]: flip_lena = numpy.flipud(lena)
        pyplot.imshow(flip_lena, cmap=pyplot.cm.gray)
```

```
Out[5]: <matplotlib.image.AxesImage at 0xa990130>
```




```
In [6]: rotate_lena = ndimage.rotate(lena, 45)
        pyplot.imshow(rotate_lena, cmap=pyplot.cm.gray)
```

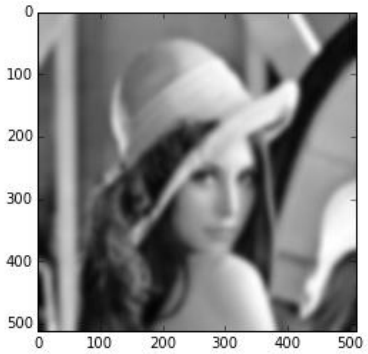
```
Out[6]: <matplotlib.image.AxesImage at 0xadeb030>
```



NumPy+SciPy+Matplotlib

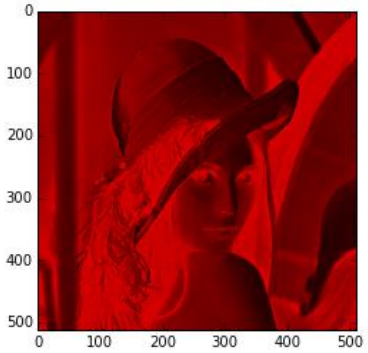
```
In [7]: blurred_lena = ndimage.gaussian_filter(lena, sigma=5)
        pyplot.imshow(blurred_lena, cmap=pyplot.cm.gray)
```

```
Out[7]: <matplotlib.image.AxesImage at 0xb335730>
```




```
In [8]: colored_lena = numpy.dstack((lena*1, lena*0, lena*0))
        pyplot.imshow(colored_lena)
```

```
Out[8]: <matplotlib.image.AxesImage at 0xb58c930>
```



Python for scientists

Next lesson...

Biopython

```
def complementary(seq):
    nt_comp = {
        'A' : 'T',
        'C' : 'G',
        'G' : 'C',
        'T' : 'A',
    }
    for nt in seq:
        compseq += nt_comp[nt]
    return compseq
```