

# Scripting for Scientific Computing: the **Python** example

Pascal Poizat

LaMI UMR 8042 CNRS/Univ. Evry, Genopole

<http://www.lami.univ-evry.fr/~poizat>

# What is a script ?

- **short** program
- used (mainly) to **glue** existing components
- written in a **high-level** scripting language
- **interpreted & portable** [only interpreter needed]
- examples: Unix shells, Tcl, Perl, PHP, **Python**, Ruby, Rexx, Scheme, VisualBasic, ...
- more and more popular (open source community)

# Short programs ...

Consider a data file `f.data` to be read:

```
1.1 9      5.2  
1.762543E-02  
0 0.1 0.01 0.001  
9 3 7
```

Python solution:

```
F = open("f.data", 'r')  
n = F.read().split()
```

In C/C++/Java: more complex

... have advantages

Scripting yields **shorter** code

Shorter code gives **less** bugs

**Faster** development/ debugging cycle  
(edit + test, no compiling)

[see J. K. Ousterhout article for time/LoC comparisons]

# Pro/Cons of Scripting

## Pros

- connecting components
- interface/GUI over an existing module
- extensive text processing
- LoL (lots of lists)
- communication with web
- prototyping

## Cons

- static typing advantages needed
- complicated data structures & algorithms
- large datasets

# (Unix) Shells vs Python

Python is a **real programming language** with:

- High-level basic data types
- Object-orientation
- Exception handling (`try ... except`)
- Lots of available modules
- Shells' basic gluing mechanisms (`>` `>>` `<` `|`)  
can be mimicked in Python  
(`os.system`, `os.exec*`, `os.popen`, ...)
- Same for basic file system operations  
(`cp`, `mv`, `rm`, `ls`, `chmod`, file related tests, ...)

# Perl vs Python

```
@list = ([1,2,3],["a","b","c"]);
for $i ( 0 .. $#list) {
for $j ( 0 .. ${$list[$i]} ) {
printf "list[%d][%d] is %s\n", $i, $j, $list[$i][$j];
```

```
list = [[1,2,3],["a","b","c"]]
for i in range(len(list)):
    for j in range(len(list[i])):
        print("list[%d][%d] is %s" % (i,j,list[i][j]))
```

# Python basics (1)

No (static) types + strong typing (basic C-like types)

```
x = 1
x = "foo"    # ok
x = 4 + x
# TypeError: cannot concatenate 'str' and 'int'
# objects
```

## Lists, tuples and dictionaries

```
l = (1, "abc", 1.0, 1.0E-10, 2-4j)
t = [1, "abc", 1.0, 1.0E-10, 2-4j]
h = {1: "result1.eps", 0.1: "result2.eps"}
m = {"June": 30, "July": 31, "August": 31}
```

## Lists of Lists of ...

```
people = {"code1": ('John', 'Smith', 32),
          "code2": ('John', 'Doe', 40) }
```

# Slicing (overview)

```
>>> a = range(4)
>>> a[:] # enables copy by value, eg in b=a
[0, 1, 2, 3, 4, 5, 6, 7, 8]
>>> a[2:4]
[2, 3]
>>> a[2:]
[2, 3, 4, 5, 6, 7, 8]
>>> b = [range(0,3),range(3,6),range(6,9)]
>>> b
[[0, 1, 2], [3, 4, 5], [6, 7, 8]]
>>> b[1][1]
4
>>> b[1,1]
Traceback (most recent call last):
  File "<stdin>", line 1, in ?
TypeError: list indices must be integers
```

# Python basics (2)

## conditional

```
if condition:
```

```
    block if
```

```
elif condition:
```

```
    block if
```

```
else:
```

```
    block else
```

## while

```
while condition:
```

```
    block while
```

## loop

```
for element in list:
```

```
    block loop
```

## list comprehensions

```
l=range(3)    # [0,1,2]
```

```
plus2=[e+2 for e in l]
           # [2,3,4]
```

```
evens=[e for e in l \
        if e%2==0]
```

```
           # [0,2]
```

# Python basics (3)

- functions

```
def f(x,y):  
    return x-y
```

```
f(3,2) # 1
```

- named arguments

```
f(y=2,x=3) # 1
```

- default arguments

```
def f(x,y=1):  
    return x-y
```

```
f(x=3) # 2
```

- variable size arg. list

```
def f(x,y,*rest):  
    sum = x-y  
    for e in rest:  
        sum = sum + e  
    return sum
```

```
f(3,2,4,5) # 10
```

- works with kw arg.

```
def f(x,y,**kw):  
    work with kw[key]
```

```
f(3,2,add1=4,add2=5)
```

# Speed ?

Scripts are said to be slower (than compiled code)

**but:**

- bytecode generation (.py -> .pyc)
- specific (compiled C) external modules exist
- **extension**: possible use of inlined / external efficient code (C, Fortran)
- **embedding**: possible use of python within C code

# Python modules for Scientific Prg.

Modules interface powerful compiled libraries

- **Numeric** / Numarray : fast/compact array data type
- Scientific Python module (**scipy**) on top of them
- Provide a strong basis with : linear algebra, integration, interpolation, FFT, signal and image processing, stats, graphics and plotting, genetic algorithms, ...
- Other (domain dedicated) modules extend this basis

# Arrays

```
>>> from Numeric import *
>>> a = zeros(6)
>>> a
array([0, 0, 0, 0, 0, 0])
>>> b = identity(4)
>>> b
array([[1, 0, 0, 0],
       [0, 1, 0, 0],
       [0, 0, 1, 0],
       [0, 0, 0, 1]])
>>> c = arange(0, pi, pi/6)
>>> c
array([ 0.          ,  0.52359878,  1.04719755,
        1.57079633,  2.0943951 ,  2.61799388,
        3.14159265])
```

# Arrays from functions

```
>>> def f(x,y):  
...     return 2*(x**2) + y  
...  
>>> a = fromfunction(f, (5,5))  
>>> a  
array([[ 0,  1,  2,  3,  4],  
       [ 2,  3,  4,  5,  6],  
       [ 8,  9, 10, 11, 12],  
       [18, 19, 20, 21, 22],  
       [32, 33, 34, 35, 36]])  
>>> # a[3,1] = 19 = 2*(3**2) + 1
```

# Arrays and types

```
>>> zeros(3,Float)
array([ 0.,  0.,  0.] )
```

```
>>> zeros(3,Complex)
array([ 0.+0.j,  0.+0.j,  0.+0.j])
```

```
>>> array([0, 0., 0+1j])
array([ 0.+0.j,  0.+0.j,  0.+1.j])
```

# Shapes and reshaping

```
>>> shape(b)
(4, 4)
>>> shape(a)
(6, )
>>> reshape(a, (3, 2))
array([[0, 0],
       [0, 0],
       [0, 0]])
```

# Slicing (revisited)

```
>>> a = reshape(arange(9), (3,3))
>>> a
array([[0, 1, 2],
       [3, 4, 5],
       [6, 7, 8]])
>>> a[1,1] # now it works !
4
>>> a[1:,1:]
array([[4, 5],
       [7, 8]])
```

other slicing operators: ... ::

# Universal functions and resizing

work on arrays

```
>>> sin(identity(2))  
array([[ 0.84147098,  0.          ],  
       [ 0.          ,  0.84147098]])
```

**! array resizing when needed !**

```
>>> array([1,2]) + array([3,4])  
array([4, 6])  
>>> array([1,2]) + array([3])  
array([4, 5])  
>>> # same than: array([1,2]) + array([3,3])  
>>> array([1,2]) * array([2,3])  
array([2, 6])  
>>> dot(array([1,2]),array([2,3]))  
8
```

# Python & external code

Possible solutions :

- use `os.system`, `os.popen`, ...
- wrapping (C/C++/Fortran):
  - SWIG (define an interface using an IDL)
  - F2PY (fortran code -> .so library)
  - Numerous others

see [www.scipy.org/wikis/topical\\_software/TopicalSoftware](http://www.scipy.org/wikis/topical_software/TopicalSoftware)

# Speed Comparisons: the benchmark

## Laplace.py benchmark

[www.scipy.org/documentation/weave/weaveperformance.html](http://www.scipy.org/documentation/weave/weaveperformance.html)

unknown function  $u(x,y)$  such that  $\nabla^2 u=0$

```
for i in range(1, nx-1):
    for j in range(1, ny-1):
        u[i,j] = ((u[i-1, j] + u[i+1, j])*dy**2 +
                  (u[i, j-1] + u[i, j+1])*dx**2)
                  / (2.0*(dx**2 + dy**2))
```

100 iterations on a 500x500 grid, eps: 1E-16

Compares: pure python, pure numeric, scipy weave  
inlined C (blitz/inline), fortran external module via  
f2py, pyrex. [C++, given separately takes 1s]

# Speed: benchmark outputs

```
[poizat@korrigan laplace]$ python laplace.py
Doing 100 iterations on a 500x500 grid
numeric took 8.18 seconds
blitz took 2.59 seconds
inline took 0.81 seconds
fastinline took 0.51 seconds
fortran took 0.78 seconds
pyrex took 0.48 seconds
slow (1 iteration) took 2.77 seconds
100 iterations should take about 277.000000
seconds
slow with Psyco (1 iteration) took 1.89 seconds
100 iterations should take about 189.000000
seconds
[poizat@korrigan laplace]$
```

# Speed Comparisons: pb. & solution

## Problem:

- launch **x** times the benchmark to compute min, max and average completion times
- generate an HTML report (via redirection)
- usage : `tester 20 > report.html`

Solution: **use python !**

# Speed: benchmark outputs

```
[poizat@korrigan laplace]$ python laplace.py
Doing 100 iterations on a 500x500 grid
numeric took 8.18 seconds
blitz took 2.59 seconds
inline took 0.81 seconds
fastinline took 0.51 seconds
fortran took 0.78 seconds
pyrex took 0.48 seconds
slow (1 iteration) took 2.77 seconds
100 iterations should take about 277.000000
seconds
slow with Psyco (1 iteration) took 1.89 seconds
100 iterations should take about 189.000000
seconds
[poizat@korrigan laplace]$
```

# Speed: benchmark outputs

```
[poizat@korrigan laplace]$ python laplace.py
Doing 100 iterations on a 500x500 grid
numeric took 8.18 seconds
blitz took 2.59 seconds
inline took 0.81 seconds
fastinline took 0.51 seconds
fortran took 0.78 seconds
pyrex took 0.48 seconds
slow (1 iteration) took 2.77 seconds
100 iterations should take about 277.000000
seconds
slow with Psyco (1 iteration) took 1.89 seconds
100 iterations should take about 189.000000
seconds
[poizat@korrigan laplace]$
```

# Python solution (1)

```
#!/usr/bin/python
```

```
import sys      # get arguments
import os       # O.S. operations
import sre      # regular expressions
import string   # string operations
import scipy    # average computation
```

```
pattern = '(.*)took(.*)seconds'
command = "python laplace.pyc"
tmp_file = "/tmp/results"
```

```
tests = ["numeric", "blitz", "inline", "fastinline",
         "fortran", "pyrex", "slow (1 iteration)",
         "slow with Psyco (1 iteration)"]
results = {}
```

```
nbtests = int(sys.argv[1])
```

# Python solution (2)

```
for test in tests: # initializing
    results[test] = []

for testnumber in range(nbttests): # testing
    # running a test
    os.system(command+" >%s"%tmp_file)
    # getting results
    F = open(tmp_file,'r')
    lines = F.read().split('\n')
    for line in lines:
        m = sre.match(pattern,line)
        if m is not None:
            com, time = m.group(1,2) # multi. assign.
            com, time = string.strip(com),
                float(string.strip(time))
            results[com].append(time)
    F.close()
    # removing result file
    os.unlink(tmp_file)
```

# Python solution (3)

```
# post-processing

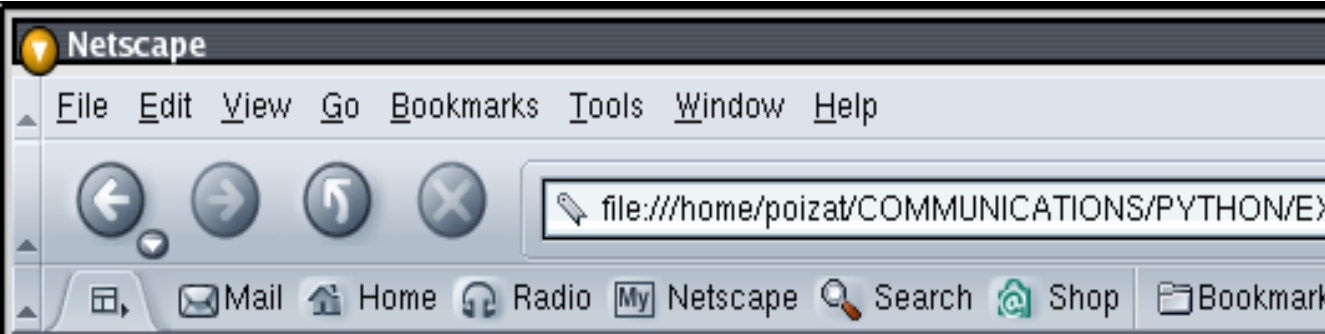
print "<HTML><BODY>"
print "Running tests on %d iterations" % nbtests
print "<TABLE BORDER=1>"
print """<TR><TH>category</TH>
        <TH>min time (sec)</TH>
        <TH>avg time (sec)</TH>
        <TH>max time (sec)</TH></TR>"""

for test in tests:
    print """<TR><TD>%s</TD>
            <TD>%.2f</TD><TD>%.2f</TD><TD>%.2f</TD>
            </TR>""" \
        % (test,
           min(results[test]),
           scipy.average(results[test]),
           max(results[test]))

print "</TABLE>"
print "</BODY></HTML>"
```

# Speed Comparison (final results)

```
tester.py 10 > report10.html
```

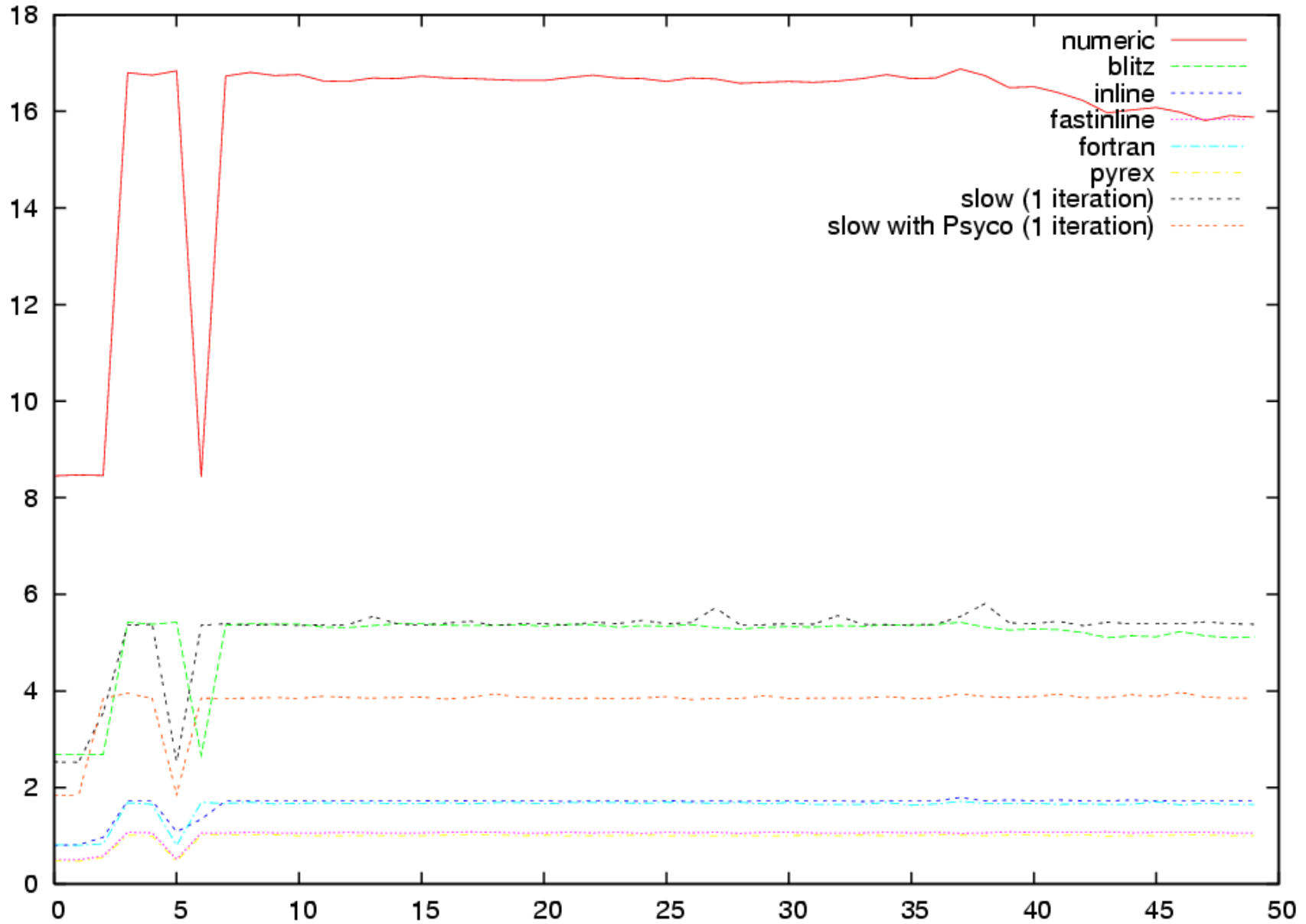


The screenshot shows a Netscape browser window with the following interface elements: a title bar with the Netscape logo, a menu bar (File, Edit, View, Go, Bookmarks, Tools, Window, Help), a toolbar with navigation buttons (back, forward, home, stop) and a search bar, and a status bar with icons for Mail, Home, Radio, My Netscape, Search, Shop, and Bookmarks. The main content area displays the text "Running tests on 10 iterations" above a table.

category	min time (sec)	avg time (sec)	max time (sec)
numeric	7.23	13.12	14.79
blitz	2.37	4.37	4.90
inline	0.81	1.53	1.71
fastinline	0.50	0.94	1.06
fortran	0.75	1.41	1.60
pyrex	0.47	0.89	0.99
slow (1 iteration)	2.49	4.72	5.28
slow with Psyco (1 iteration)	1.84	3.47	3.89

# Enhancing Outputs ?

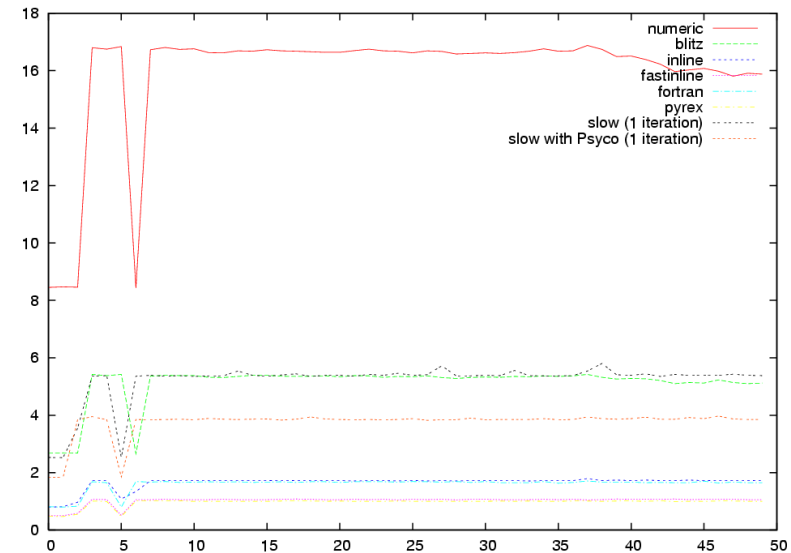
# Enhancing Outputs ?



# Plotting API(s)

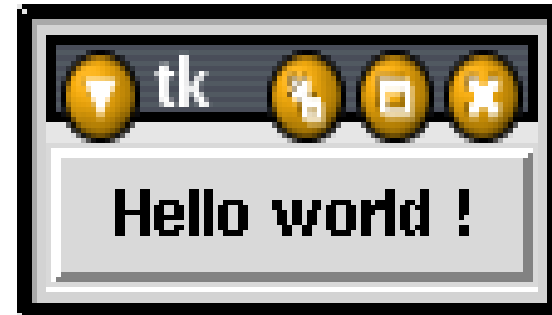
**Gnuplot**, [xg]plt, dislin, ...

```
ids_tests = range(nbttests)
import Gnuplot
g = Gnuplot.Gnuplot()
g('set data style lines')
g.plot(Gnuplot.Data(ids_tests,
                    results[tests[0]],
                    title=tests[0]))
for test in tests[1:]:
    g.replot(Gnuplot.Data(ids_tests,
                          results[test],
                          title=test))
g.hardcopy("results.eps", color=1, enhanced=1)
```



# Building a GUI

```
from Tkinter import *  
  
def sayHello():  
    print "hello"  
  
root = Tk()  
hello = Button(root, text='Hello world !',  
               command = sayHello)  
hello.pack()  
root.mainloop()
```



# More & more modules/libraries

- Hinsen's Scientific Python :  
lots of extensions for geometry (vectors, tensors, ...), stats, signal processing, netCDF, MPI and BSPlib interfaces, ...
- distributed parallel programming (MPI) :  
pyMPI, pythonMPI, pyPAR, Scientific Python, ...
- Grid computing: PyGlobus, PEG
- Interfaces for R, BioPython, Matlab, ...

# References (python / scripting)

[www.tcl.tk/doc/scripting.html](http://www.tcl.tk/doc/scripting.html)

J. K. Ousterhout. Scripting: Higher Level Programming for the 21st Century. IEEE Computer. March 1998.

[www.python.org](http://www.python.org)

python web site

[www.ifi.uio.no/in228/lecsplit](http://www.ifi.uio.no/in228/lecsplit)

H.P. Langtangen. Scripting for Computational Science course.

# References (python + scientific)

[www.python.org/moin/NumericAndScientific](http://www.python.org/moin/NumericAndScientific)

Wiki on numeric & scientific python

[www.scipy.org](http://www.scipy.org)

Scientific Tools for Python

[www.enthought.com](http://www.enthought.com)

Python scientific distribution for Windows

[www.logilab.net/cups](http://www.logilab.net/cups)

Club des Utilisateurs de Python dans le domaine

Scientifique (sorry, in French only)