Introducción a Python para Científicos

Fermin Huarte
Ramon Crehuet

Barcelona, noviembre 2023

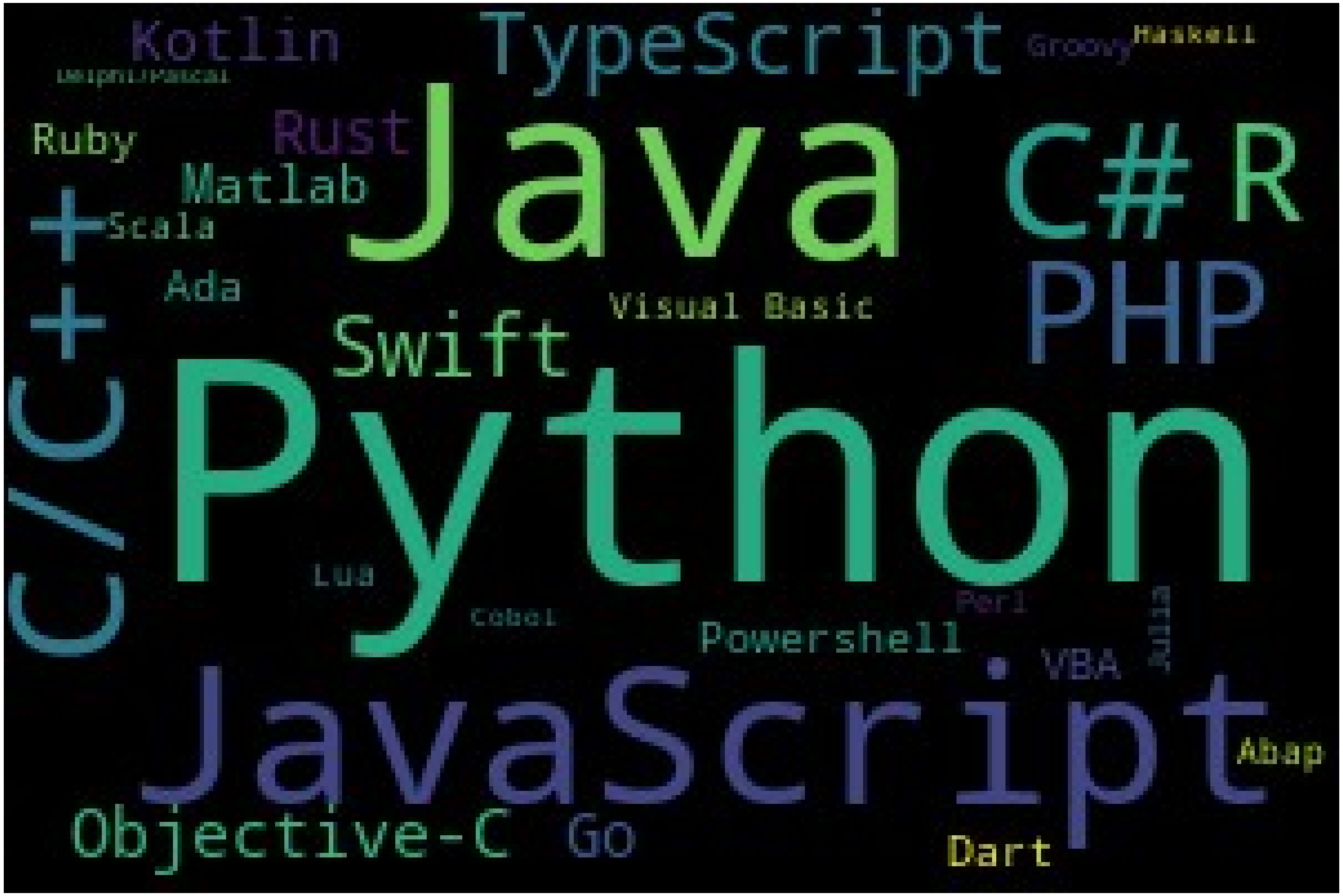
You?

- Research institute?
- Research field?
- Known languages?
- Kind of data?
- Expectations?

Overview

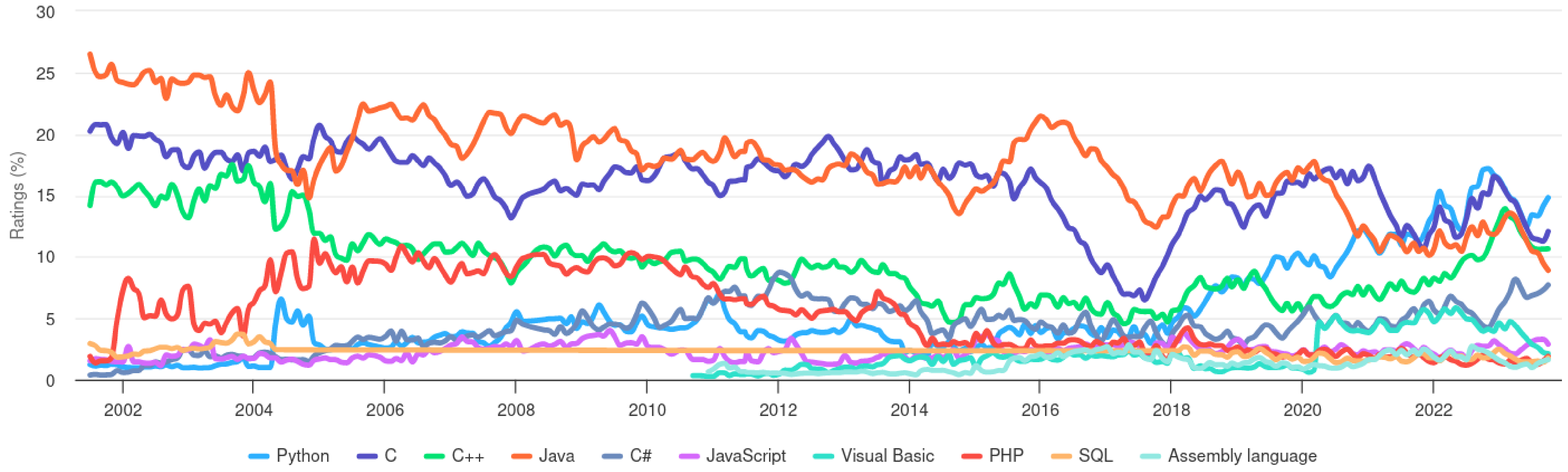
- Why Python
- Language basics
  - Lists, sets, dictionaries, control structures
- interative development: Jupyter notebooks
- Working with files and strings
- Working with arrays: Numpy
- Data visualization: matplotlib
- data analysis with Pandas
- Functions and modules
- Scientific modules. Scipy
- Other scientific modules: sckikit-learn, biopython...

Language popularity



http://pypl.github.io/PYPL.html

Language popularity



https://www.tiobe.com/tiobe-index/

Python for science

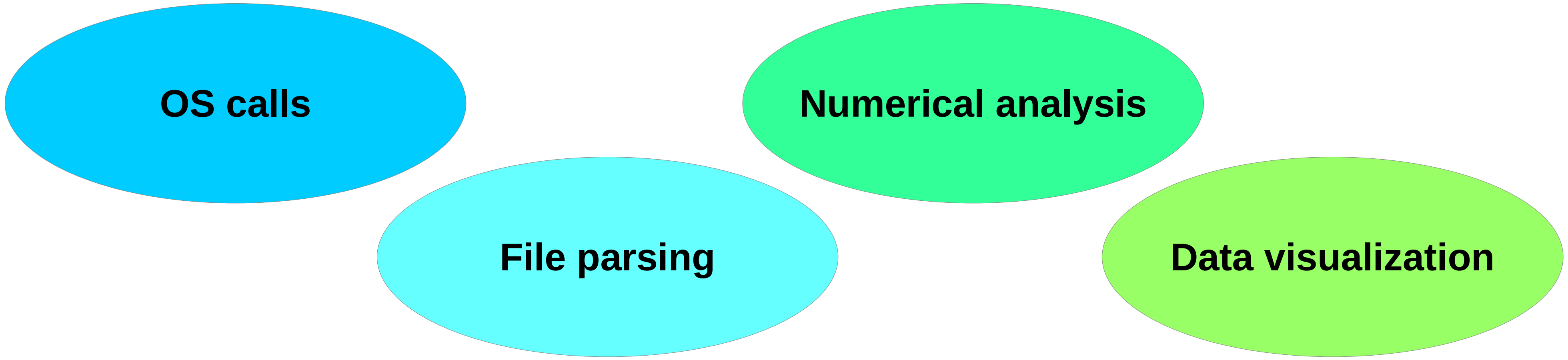
- A high level language gives more time to more complex problems
  - At the expense of hiding important details
- Example:
  - A reaction mechanism
  - Optimisation of an energy function
    - Steepest descent, conjugate gradients, quasi-Newton
  - Implementation of BFGS quasi-Newton
    - Memory issues, diagonalization, matrix inversion...
  - Calculation of numerical gradients or hessians:
    - machine precision, central differences, etc.

http://fperez.org/py4science/why_python.html

Python for science

“We then generated 1000 random sequences with randomly specified $\langle H \rangle$ and $\langle Q \rangle$ values, and conducted molecular-dynamics simulations to calculate the $\langle Rg \rangle$ for each chain. The obtained $\langle Rg \rangle$ values were combined with the two-state formalism to determine whether the chain was ordered (globule) or disordered (coil)”

Biophysical Journal, **104**, 2013, 488–495



Python for science

Compiled languages
Fast
Difficult
non-interactive

Matlab, Mathematica, Octave
Slow
Rich libraries
Nice development environment
Restricted base language
Expensive (some)

Python
Rich libraries (less than matlab)
Other libraries
Free
Active community
Harder than Matlab

Matlab, Mathematica?

- Scientific computing:
  - ipython + scipy + matplotlib
- Free
- Open source
- Extensible

- Bioinformatics
  - Biopython
- Molecular Dynamics
  - OpenMM
- Efficiency
  - Numba, Cython, Fortran, C
- Server control
- XML parser

Python for data science

- Python or R for Data Analysis: Which Should I Learn?
- Python vs R for Data Science: Which Should You Learn?
- Python vs. R: What’s the Difference?
- Python, Machine Learning, and Language Wars

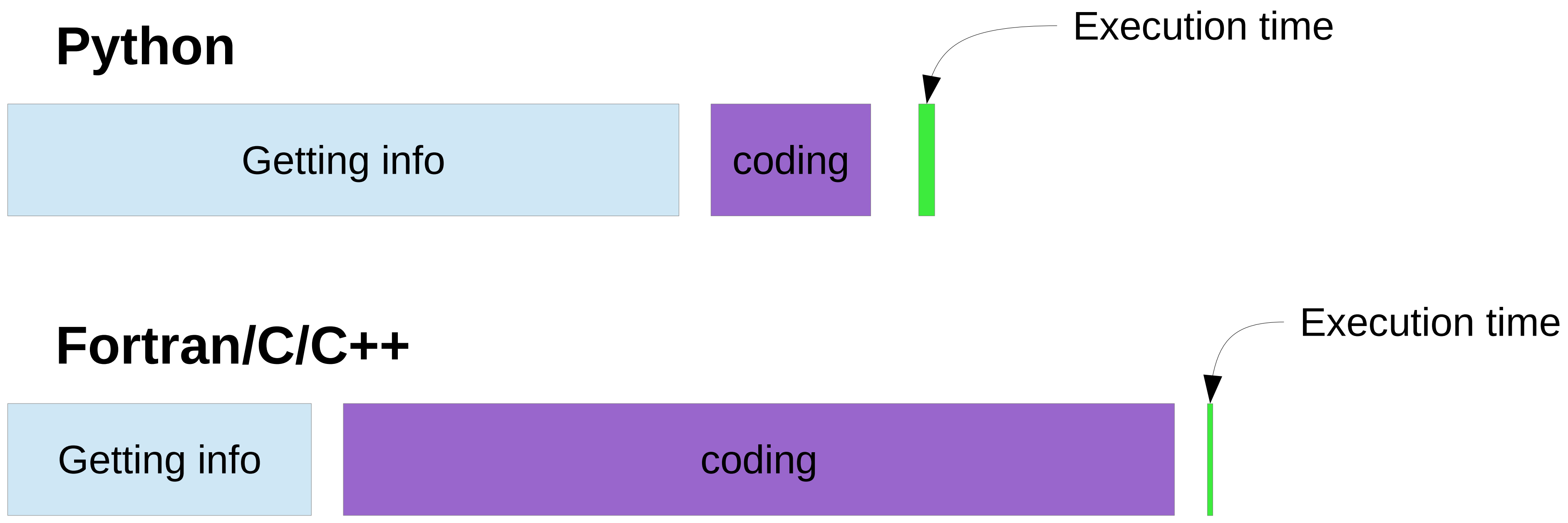
Low level vs. high level

- Python is a high level language
- You can focus on:
  - Low level issues
  - Higher complexity of problems

- Low level issues
  - Variable types
  - Machine precision
- But also
  - Extend
  - Mantain
  - Document code

Python vs. Fortran/C

Different time distribution to get a task done



Python for science

- The homogenization of scientific computing, or why Python is steadily eating other languages' lunch
  http://www.talyarkoni.org/blog/2013/11/18/the-homogenization-of-scientific-computing-or-why-python-is-steadily-eating-other-languages-lunch/
- 10 Reasons Python Rocks for Research (And a Few Reasons it Doesn't)
  http://www.stat.washington.edu/~hoytak/blog/whypython.html

Hello World program

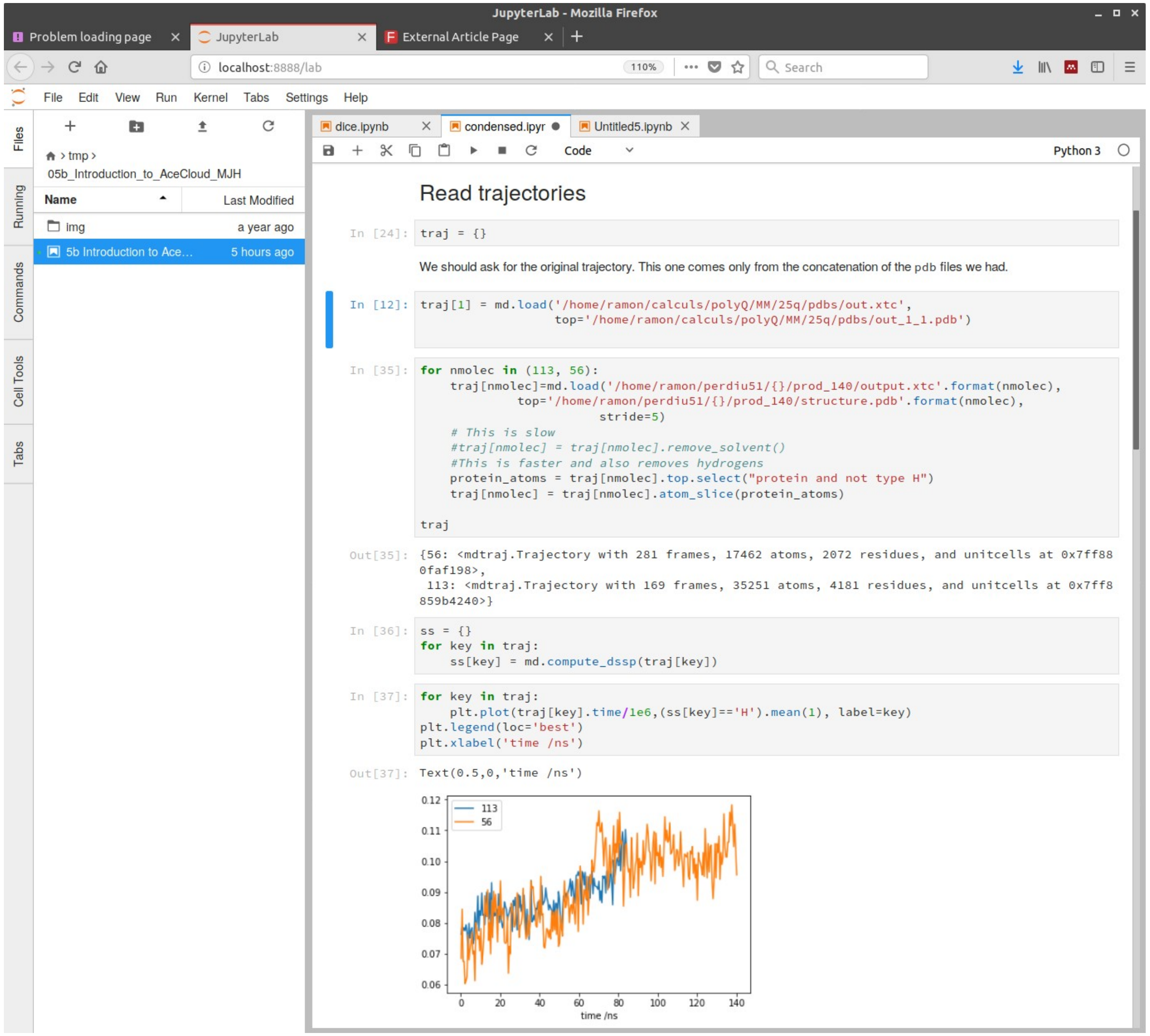
```
print(“Hello World!”)
```

```
print(“Hello World!”)
```

```
$ python hello.py
```

Interactive environments

- python
- Visual Studio
- **Jupyter Lab**
- spyder
- Rodeo (for data science)
- Sage



Read trajectories

```
In [24]: traj = {}
```

We should ask for the original trajectory. This one comes only from the concatenation of the pdb files we had.

```
In [12]: traj[1] = md.load('/home/ramon/calculs/polyQ/MM/25q/pdbs/out.xtc',
                           top='/home/ramon/calculs/polyQ/MM/25q/pdbs/out_1_1.pdb')

In [35]: for nmolec in (113, 56):
             traj[nmolec]=md.load('/home/ramon/perdiu51/{}/prod_140/output.xtc'.format(nmolec),
                      top='/home/ramon/perdiu51/{}/prod_140/structure.pdb'.format(nmolec),
                                stride=5)
             # This is slow
             #traj[nmolec] = traj[nmolec].remove_solvent()
             #This is faster and also removes hydrogens
             protein_atoms = traj[nmolec].top.select("protein and not type H")
             traj[nmolec] = traj[nmolec].atom_slice(protein_atoms)

         traj

Out[35]: {56: <mdtraj.Trajectory with 281 frames, 17462 atoms, 2072 residues, and unitcells at 0x7ff88
         0faf198>,
          113: <mdtraj.Trajectory with 169 frames, 35251 atoms, 4181 residues, and unitcells at 0x7ff8
         859b4240>}

In [36]: ss = {}
         for key in traj:
             ss[key] = md.compute_dssp(traj[key])

In [37]: for key in traj:
             plt.plot(traj[key].time/1e6,(ss[key]=='H').mean(1), label=key)
         plt.legend(loc='best')
         plt.xlabel('time /ns')

Out[37]: Text(0.5,0,'time /ns')
```

Python distributions

- Anaconda
  - https://www.anaconda.com/download
- Others are gone

Dynamically typed

```
>>> a = 4
>>> type(a)
<class 'int'>
>>> b = 7.6
>>> type(b)
<class 'float'>
>>> type(a+b)
<class 'float'>
>>> c = 'Hola'
>>> c + ' Que tal?'
'Hola Que tal?'
>>> c + a
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: Can't convert 'int' object to str implicitly
```

Language elements

Numbers

Integers:

```
> i = 5
> j = i**i**i
```

Limited by amount of memory:

```
>>> i.bit_length()
3
>>> j = i**i**i
>>> j.bit_length()
7257
>>> 9 % 5 #modulo
4
```

Floating point:

```
>>> x = 5.
>>> y = x**x**x
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
OverflowError: (34, 'Numerical result
   out of range')
```

Division vs integer division (Python 3):

```
>>> 3/2
1.5
>>> 3//2
1
>>> j/i #Returns a Float
```

Assignments

Explicit notation:

```
> i = i+1
> j = j / 10.
```

Short notation:

```
> i += 1
> j /= 10.
```

Floating point:

```
>>> x = 5.
>>> y = x**x**x
Traceback (most recent call
   last):
  File "<stdin>", line 1, in
   <module>
OverflowError: (34, 'Numerical
   result out of range')
```

strings

Strings:

```
> str(6.7)
> c ='Hola'
```

Operations:

```
> s='numeric ' +'python'
> len(s)
> s[5]
'i'
> s.split()
['numeric', 'python']
```

```
> print('Result: %5.3f' % (11./3.))
3.667
```

Non mutable:

```
> s[6]
> s[6]='7' #Error!
```

Regular expressions

```
import re
```

Lists, sets and tuples

- Lists can be heterogeneous:

```
> l=[6, 'a', [5,[9,8,7,6]], -
  6.5, (True, True)]
> [1,2]+[3,4]
> l.append(6)
```

- sets:

```
s=set([4,3,2,3])
> 4 in s
True
> s
set([2, 3, 4])
```

- Tuples are unmutable lists

```
t=(1,2,3)
```

Lists, sets and tuples

List indexing and methods:

```
> l = list(range(10))
> l[4] = 20
> l[4:]
> l[-4]
> l[:]
> l[::-1] #reverse
> l.reverse()
> l.pop()
> l.extend([3,4,5])
> l.sort()
```

First index is 0

Set methods:

```
> s1=set([1,2,3,4])
> s2=set([3,4,5,6,7])
> s1.union(s2)
> s1.intersection(s2)
> s1.difference(s2)
> s2.difference(s1)
> s1.intersection(s2) == s2 & s1
True
> s1 - s2 == s2 - s1
False
```

Uses of lists, sets and tuples

- Calculate and keep all the primes < 1000
- Given a coordinate file, calculate for each atom a list of all the atoms that are at less than 0.2nm.
- Get the solutions of a quadratic equation (0,1,2) or (real vs. complex).
- http://docs.python.org/3/tutorial/datastructures.html

Copying and looping over lists

lists are treated as pointers:

- copying lists, makes a copy of the pointer.

```
> l=[1,2,3,4]
> l2=l
> l[2]=1000
l
[1, 2 , 1000, 4]
```

Looping over lists:

- Fortran/C style:

```
num=[2,3,2,3,4,5,5]
for i in range(len(num)):
    print(num[i])
```

- Pythonic style:

```
for item in num:
    print(item)
```

- This can be used for sets, dictionaries, and tuples.

Dictionaries

Setting elements:

```
    > phone={}
    > phone['Ramon']='1242'
    > phone['Joan']='1323'
    > phone['Quique']='1242'
    > phone.keys()
['Quique', 'Joan', 'Ramon']
> d2 = dict(Ramon=1242, Joan=1323,
   Quique=1242)
```

Dictionaries are not ordered

Getting elements:

```
> for key in phone:
...  print(key, phone[key])
Quique 1242
Joan 1323
Ramon 1242
```

Removing elements:

```
> del(phone['Ramon'])
```

The beauty of Python blocks

We are usually told to indent blocks for clarity.

Python makes this the syntax rule to identify blocks.

The code has to be nice!

Convention:

- Use 4 spaces
- Use spaces, not tabs.

```
while iter < maxIter:
    x = f(x)
    iter = iter + 1

if i>0:
    print("i is positive")
elif i==0:
    print("i is zero")
else:
    print("i is negative")
```

Execution control: if

if... elif... else

```
if <condition>:
  <block>
elif <condition>:
  <block>
else:
  <block>
```

```
4==4 #True
5!=4 #True
4>=5 #False
4 in [4,5] #True
result=True
if result: print('yes')
```

Execution control

Conditions can be combined with:

`and or not ( )`

Object identity:

```
> a=[1,2,3]
> b=a
> b is a
True
```

Any non-zero number or non-empty string is True:

```
> if []: print ('yes')
     else: print('no')
no
> if 5 and 'result':
    print('yes')
else:
    print('no')
yes
> if 5 or 1/0:  print('yes')
yes
```

for and while loops

For loops

```
> dict={4:'a',3:'b', 2:'c',
  1:'d'}
> for i in dict:
      print(i, dict[i])
```

While

```
while <condition>:
   <block>
```

Break continue pass

```
> pass # does nothing
```

`break`: Exit loop

```
if x>0:
  pass
else:
  break
```

`continue`: Continue with the next iteration

list comprehension and enumerate

simple way to create lists:

```
> l=[x**2 for x in range(8)]
[0, 1, 4, 9, 16, 25, 36, 49]
```

with conditionals:

```
l2= [(i, -2*i+3) for i in l if i % 3 == 0]
[(0, 3), (9, -15), (36, -69)
```

Nested lists:

```
> [(x, y) for x in [1,2,3] for y in [3,1,4] if x != y]
[(1, 3), (1, 4), (2, 3), (2, 1), (2, 4), (3, 1), (3, 4)]
```

list comprehension and enumerate

Enumerate indexes lists:

```
line='how do you do?'
line=line.split()
for i, word in enumerate(line):
  print(i, word.upper())
0 HOW
1 DO
2 YOU
3 DO?
```

Enumerate returns an iterator

```
> enumerate(['a', 'b', 'c'])
<enumerate object at 0x1ebeaa50>
```

Be pythonic

Convert the negative elements of a list to positive

```
>>> x = [1, 2, -4, -5, 3, -5]

j = 0
while j < len(x):
   x[j] = abs(x[j])
   j += 1

for j in range(len(x)):
    x[i] = abs(x[i])
```

Or with list comprehensions

```
x = [abs(j) for j in x]
```

Or with functional programming

```
x= map(abs, x) #returns an iterator
```

http://docs.python-guide.org/en/latest/writing/style/

More python functions

```
print(3,4,5, sep='o', end='<<<<\n')
zip([1,2,3], ['a', 'b', 'c', 'd'])
a = input('Write a number: ')
len([1,2,3])
list(range(5))
range(20,10,-1)
sorted([5,4,3,5])
sum([5,4,3,5])
```

http://docs.python.org/3.3/library/functions.html

Mutable and immutable

- Mutable objects can be mutated.
  - Their identity remains the same
- Immutable objects are “mutated” by creating a new object

```
>>> a = 4
>>> id(a)
9157088
>>> a += 2
>>> id(a)
9157152
>>> s = 'Hola'
>>> id(s)
140165884365656
>>> s = s+ ' que tal?'
>>> id(s)
140165884365712
>>> ll = [3,4,5]
>>> id(ll)
140165884674416
>>> ll.append(6)
>>> id(ll)
140165884674416
```

Identity and equality

```
>>> 1.0 is 1.0
True
>>> 1.0 == 1.0
True
>>> 1 == 1.0
True
>>> 1 is 1.0
False
```

```
>>> a = 4
>>> b = a
>>> a is b
True
>>> id(a)
9157088
>>> id(b)
9157088
>>> l1 = [1,2,3,]
>>> l2 = l1
>>> l2 = l1[:]
>>> l2 is l1
False
>>> l2 == l1
True
```

Objects: everything

```
>>> a = 5
>>> isinstance(a, int)
True
>>> object
<class 'object'>
>>> int
<class 'int'>
>>> isinstance(a, object)
True
>>> issubclass(int, object)
True
```

Objects have variables:

```
> c = 4+5j
> c.real
```

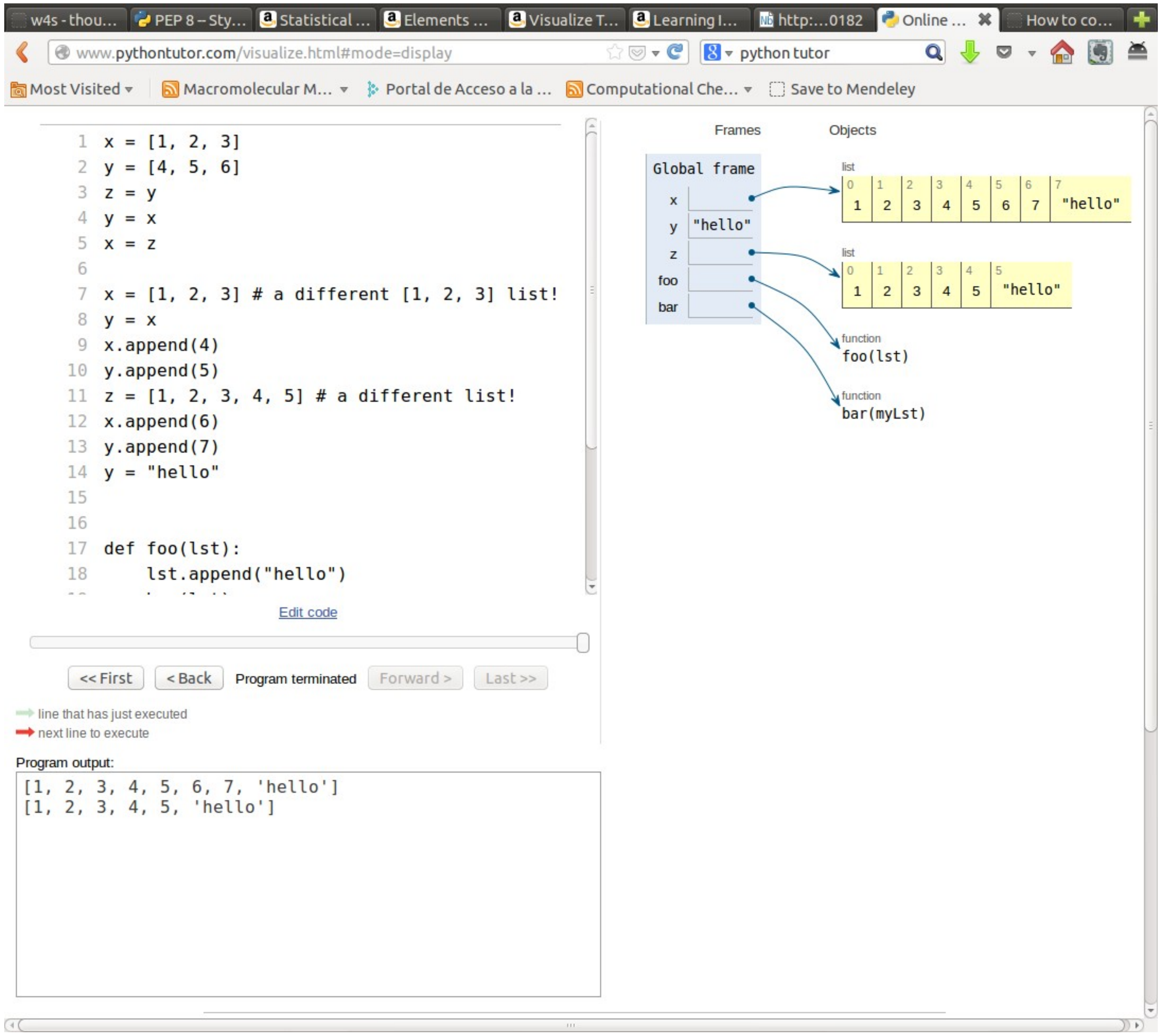
Objects have methods:

```
> c.conjugate #the method
> c.conjugate() #its call
```

And we can apply functions to objects:

```
> abs(c)
```

Python flow with pythontutor



```
x = [1, 2, 3]
y = [4, 5, 6]
z = y
y = x
x = z

x = [1, 2, 3] # a different [1, 2, 3] list!
y = x
x.append(4)
y.append(5)
z = [1, 2, 3, 4, 5] # a different list!
x.append(6)
y.append(7)
y = "hello"


def foo(lst):
    lst.append("hello")
```

0	1	2	3	4	5	6	7
1	2	3	4	5	6	7	"hello"

0	1	2	3	4	5
1	2	3	4	5	"hello"

Program output:

```
[1, 2, 3, 4, 5, 6, 7, 'hello']
[1, 2, 3, 4, 5, 'hello']
```

try... except

"Look before you leap":

```
def safe_divide_1(x, y):
  if y==0:
    print("Divide-by-0 attempt detected")
    return None
  else:
    return x/y
```

"It's easier to ask forgiveness than permission":

```
def safe_divide_2(x, y):
  try:
    return x/y
  except ZeroDivisionError:
    print("Divide-by-0 attempt detected")
    return None
```

Short jupyter-lab tutorial

Magic functions

```
%timeit x=10     : time the 'x=10' statement with high precision.
%%timeit x=2**100
x*100           : time 'x*100' with a setup of 'x=2**100'; setup code is not
                   counted.  This is an example of a cell magic.

%history
%load_ext
%run
%pdb: Control the automatic calling of the pdb interactive debugger.
%timeit
%pwd
%cd
%%bash
```

Also tab autocompletion

http://ipython.org/ipython-doc/dev/interactive/tutorial.html

Running scripts

`%run script.py`

`import script.py`

are not the same!

`%run script.py` is like `python3 script.py`

Imports are only “imported” once in a session (see later `%autoreload` magic function)

ipython notebook

- Nice presentation
- Allows parallel execution
- Combines text and code
- Executable or exportable to:
  - html
  - LaTeX
  - python
- Start with: `jupyter lab`
- Examples:
  https://github.com/jrjohansson/scientific-python-lectures

Files

Files

- Files can be *text* or *binary*
- Files can be opened for read, write or append
  - `'r', 'w', 'a+'`
- `with open('name') as filein:`
  - Allows automatic file closure
  - Explanation of the `with` statement: http://effbot.org/zone/python-with-statement.htm

Reading / Writing Files

```
file_in=open('indata.txt','r')
file_out=open('outdata.txt','w')
for line in file_in:
    # Take some information (split() method is very useful!)
    x = float(line.split()[0])
    # Apply a given function (fact)
    fx = fact(x)
    # Write the result in an output file with a defined format
    file_out.write('{:010.3f}\n'.format(fx))
```

But for loading numerical data **Numpy** is more efficient. And **pandas** even more.

File parsing

- The basic:

```
for line in filein:
    do something
```

- Common things:

```
if 'optimized' in line:
    do something
line = line.split()
if line.upper().startswith('GEOM'): …
energy = float(line[2])
```

skipping lines

- Lines can be skipped by calling `next()` to a file:

```
for line in filein:
    if 'Optimized' in line:
        next(filein); next(filein) #skip two lines
        do something...
```

Formatting

- There are several function:

```
'12'.rjust(5), '12'.zfill(5)
```

- But `format` is more general:

```
print('{0:2d} {1:3d}'.format(x, x*x))
print("{:10.3f} {:10.3f} {:10.3f}".format(x,y,z))
```

- List of unkown length (use argument unpacking):

```
vals = np.linspace(0,1,11)
print((len(vals)*"{:10.2e} ").format(*vals))
```

http://docs.python.org/3/library/string.html#formatspec

Useful modules

- Similar to `ls`:

```
import glob
files = glob.glob(pattern)
```

- Working with shell-like commands:

```
import os
os.rename(src, dst)
os.mkdir(path)
os.chown(path, uid, gid)
os.getenv(key)
os.walk(directory)
```

http://docs.python.org/3/library/os.html

Useful modules

- Reading Excel files http://www.python-excel.org/:

```
import xlrd
```

  - Pandas uses this library
- Working with image files
  - http://scikit-image.org/
  - http://pillow.readthedocs.io/en/latest/

Numpy

Why Numpy / Scipy?

- Python (alone) is not efficient for numerical calculations
- Python (alone) is not practical for array manipulation
- Numpy provides the data types and methods for arrays
- Scipy provides more elaborate numerical methods
  - Optimization
  - Fast Fourier Transform
  - Linear algebra, etc

```
import numpy as np

import scipy.optimization

import scipy.stats as stats
```

numpy arrays

- without numpy:

```
> a=[[1,2],[3,4]]
> b=[[10,20], [30,40]]
> a+b
[[1, 2], [3, 4], [10, 20], [30, 40]]
```

- with numpy:

```
> a=np.array(a)
> b=np.array(b)
> a+b
array([[11, 22],[33, 44]])
```

- Array creation

```
a=np.array([1,2,3,4]).reshape([2,2])
a=np.array([[1,2], [3,4]])
a=np.zeros([2,2], dtype=int)
a[0,0]=1.
a=np.ones((4,4))
a=np.arange(10)
a=np.diag([1,2,3,4])
a=np.tile(a, (10,2))
a=np.identity(3)
a=np.linspace(-5,5, 20)
```

Ufuncs

- Unary:

```
a.min()
a.sum()
a.cumsum()
a.mean()
np.argmin(a)
np.exp(-a)
np.cov(a)
a.tolist()
```

- Binary:

```
a + b
np.dot(a, b)
```

- Applying to parts of an array:

```
> a=np.array([[1,2], [3,4]])
> a.min(axis=0)
array([1, 2])
a.sum(axis=1)
array([3, 7])
```

- `Python functions are less efficient than numpy functions:`

  `a.sum()` better than `sum(a)`

  `np.min(a)` better than `min(a)`

many implemented as methods and functions

Accessing array elements

- Slicing:

```
> a[2:5]


> b[:, ::5]


> a[1:4, ...]
```

- Fancy indexing:
  - Boolean arrays (masks):

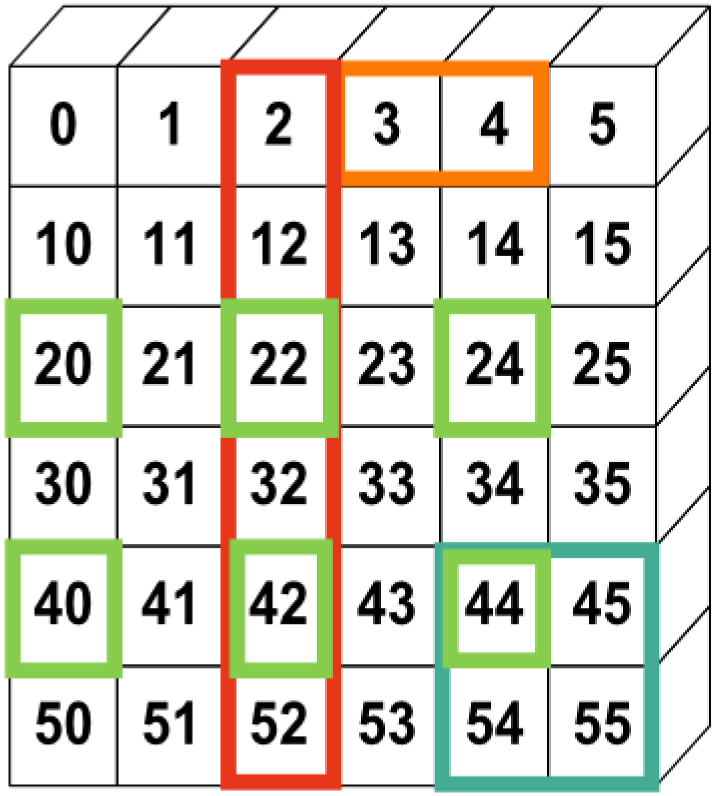
```
> a = np.arange(10,15)
> indices = (a**2 > 115) & (a < 14)
> a[indices]
array([11, 12, 13])
```

  - With lists:

```
> a = np.arange(10,15)
> y=a[[4,4,1]]
> y
array([14, 14, 11])
> a[[4,4,1]] = [-2, -4, 5]
> a
array([10,  5, 12, 13, -4])
```

Accessing array elements

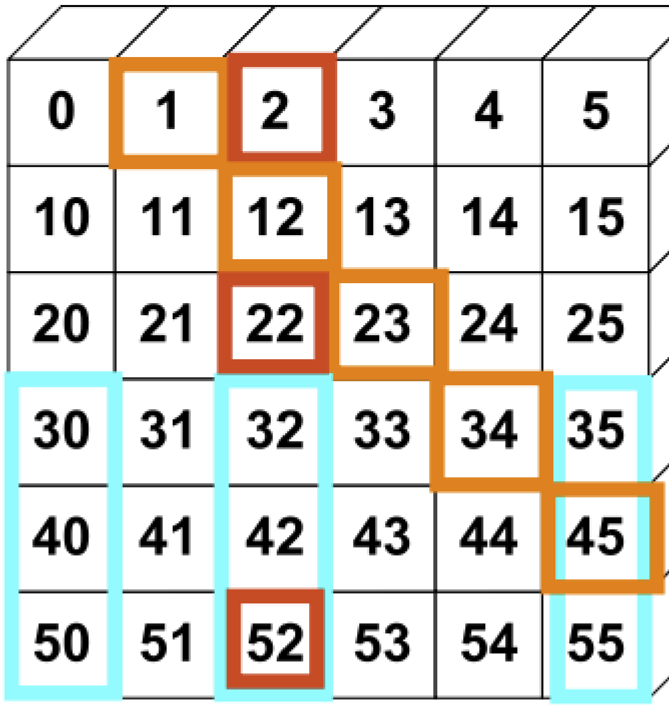
```
>>> a[0,3:5]
array([3,4])


>>> a[4:,4:]
array([[44, 45],
       [54, 55]])


>>> a[:,2]
array([2,12,22,32,42,52])


>>> a[2::2,::2]
array([[20,22,24]
       [40,42,44]])
```



From: https://scipy-lectures.github.io/intro/numpy/array_object.html

Accessing array elements

```
>>> a[(0,1,2,3,4),(1,2,3,4,5)]
array([ 1, 12, 23, 34, 45])

>>> a[3:,[0, 2, 5]]
array([[30, 32, 35],
       [40, 42, 45]])
       [50, 52, 55]])

>>> mask = array([1,0,1,0,0,1],
                 dtype=bool)
>>> a[mask,2]
array([2,22,52])
```



From: https://scipy-lectures.github.io/intro/numpy/array_object.html

Accessing array elements

- Slices return views

```
> a = np.arange(5)
> y=a[2:5]
> y *= -1
> a
array([ 0,  1, -2, -3, -4])
> y.flags.owndata
False
```

- np.where

```
> np.where((a>=2)&(a<4), a**2, -1)

Array([-1, -1,  4,  9, -1])
```

- np.choose
  - Powerful, but complex!
- np.nonzero

- Boolean arrays return copies

```
> a = np.arange(5)
> y = a[a>1]
> y *= -1
> a
array([0, 1, 2, 3, 4])
> y.flags.owndata
True
```

- Fancy indexing returns copies:

```
> a = np.arange(5)
> y=a[[2,3,4]]
> y *= -1
> a
array([0, 1, 2, 3, 4])
> y.flags.owndata
True
```

Broadcasting

```
> a = 4.
> b = np.array([1,2,3])
> c = np.array([[1,2,3], [4,5,6]])
> b+a, c+a
(array([ 5.,  6.,  7.]), array([[  5.,   6.,   7.],
        [  8.,   9.,  10.]]))
> b+c
array([[2, 4, 6],
       [5, 7, 9]])
> c.dot(b)
> b.dot(c)
ValueError: objects are not aligned
> b[1:]*c
ValueError: operands could not be broadcast together with shapes (2) (2,3)
> b[1:]*c.T
```

- Use `matrix` if you want more algebra-like behaviour

Broadcasting

- Change the shape to allow for broadcasting:

```
> c = np.array([[1,2,3], [4,5,6]])
> b = c.mean(axis=1)
> c+b[:,np.newaxis] #or c+b[:,None]
> c+b.reshape((-1,1))
```

- Or keep the shape:

```
> b = c.mean(axis=1, keepdims=True)
> c+b
```

- See also:
  - `np.atleast_2d`, `np.atleast_1d` and `np.atleast_3d`

Broadcasting

- **Broadcasting rules:**

  When operating on two arrays, NumPy compares their shapes element-wise. It starts with the trailing dimensions, and works its way forward. Two dimensions are compatible when

  1) they are equal, or
  2) one of them is 1

- More examples and longer explanation here:
  - https://docs.scipy.org/doc/numpy-1.13.0/user/basics.broadcasting.html

np.einsum

- Complex but powerful function to avoid the use of loops
  - Dot product, outer product, and others can be written as `einsum`

```
> c = np.array([[1,2,3], [4,5,6]])
> np.allclose(c.dot(c.T), np.einsum('ij, kj->ik',c,c))
True
```

- See numpy documentation and the following blog entry:
  - http://docs.scipy.org/doc/numpy-1.10.0/reference/generated/numpy.einsum.html
  - http://ajcr.net/Basic-guide-to-einsum/

array functions and methods

- Array reduction and logical operations:

```
> a=np.arange(5)
> np.all(a>3)
False
> np.any(a>3)
True
> a > 3
array([False, False, False, False,
 True], dtype=bool)
> (a > 3) & (a < 5)
array([False, False, False, False,
 True], dtype=bool)
```

- Some details of memory use:
- `> a.flags`

```
 C_CONTIGUOUS : True
  F_CONTIGUOUS : True
  OWNDATA : True
  WRITEABLE : True
  ALIGNED : True
  UPDATEIFCOPY : False
```

Loading and saving data

- Pickle is the usual way to save and restore data in Python
- We often have data file in text format:

```
#Dist Energy
1.0 34.
1.2 38.
2.4 42.
```

- `f=np.loadtxt(“energies.dat”)`
- `f=np.genfromtxt(“energies.dat”)`

```
> f
array([[  1. ,   34. ],
       [  1.2,   38. ],
       [  2.4,   42. ]])
```

- Save single arrays with:

```
> np.save('result_y', y)
```

- Save in text mode with:

```
> np.savetxt('result_y', y)
```

- and multiple arrays with (saves a dictionary):

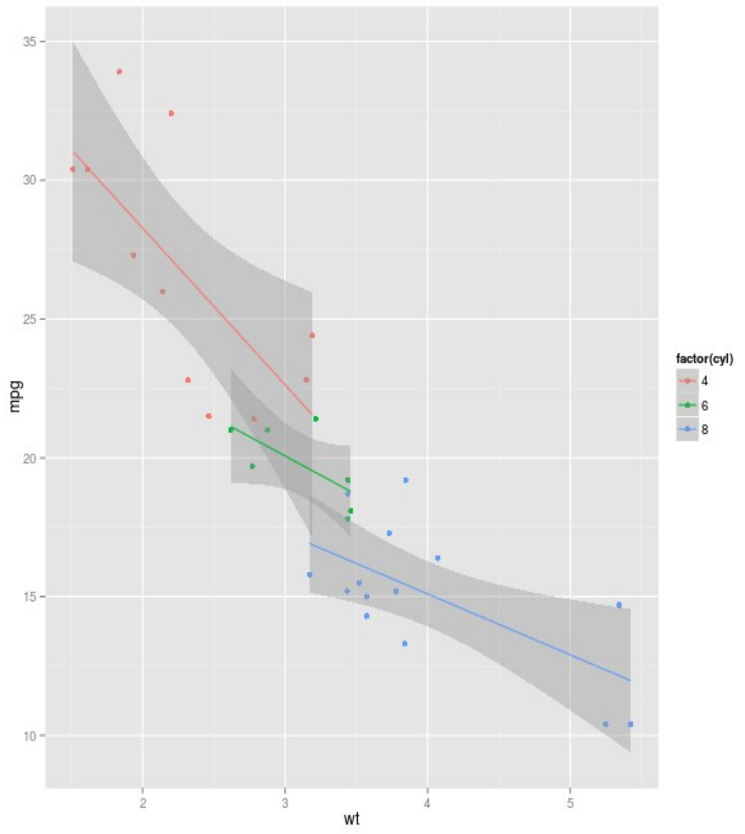
```
> np.savez('results', x, y)
```

- Recover them with load:

```
> y=np.load('results_y.npy')

> npz=np.load('results.npz')
```

Acess R from python

- Use the rpy2 module.
- From the documentation:



```
import math, datetime
import rpy2.robjects.lib.ggplot2 as ggplot2
import rpy2.robjects as ro
from rpy2.robjects.packages import importr
base = importr('base')
datasets = importr('datasets')


mtcars = datasets.data.fetch('mtcars')['mtcars']
pp = ggplot2.ggplot(mtcars) + \
     ggplot2.aes_string(x='wt', y='mpg', col='factor(cyl)') + \
     ggplot2.geom_point() + \
     ggplot2.geom_smooth(ggplot2.aes_string(group = 'cyl'),
                         method = 'lm')
pp.plot()
```

Other tutorials

- Take a look at these tutorials:
  - http://wiki.scipy.org/Tentative_NumPy_Tutorial
  - From: http://jrjohansson.github.io/
    - Lecture-2-Numpy.ipynb
    - Lecture-3-Scipy.ipynb

matplotlib

Matplotlib

- A module for plotting 2D and 3D data
- Combines well with numpy
- Starts with

```
import matplotlib.pyplot as plt
%matplotlib inline
```

`import pylab` or similar is deprecated.

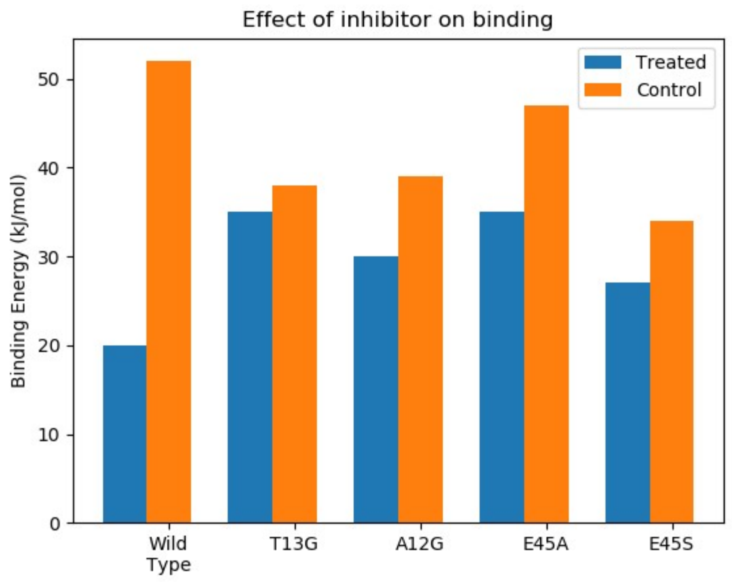
Matplotlib

Simplest plots:

```
> plt.plot([1,2,3], [1,4,9])
> plt.plot(x, sin(x), '--') #where x is a numpy array
> plt.figure() # creates new figure
> plt.clf() # Clears current figure
> plt.matshow(m) # m is a 2D array
> plt.imshow(m) # m is a 2D array. Same as matshow.
> d = np.loadtxt('data.txt')
> plt.plot(d[:,0], d[:,1], 's') #just slightly longer than
gnuplot
```

Matplotlib

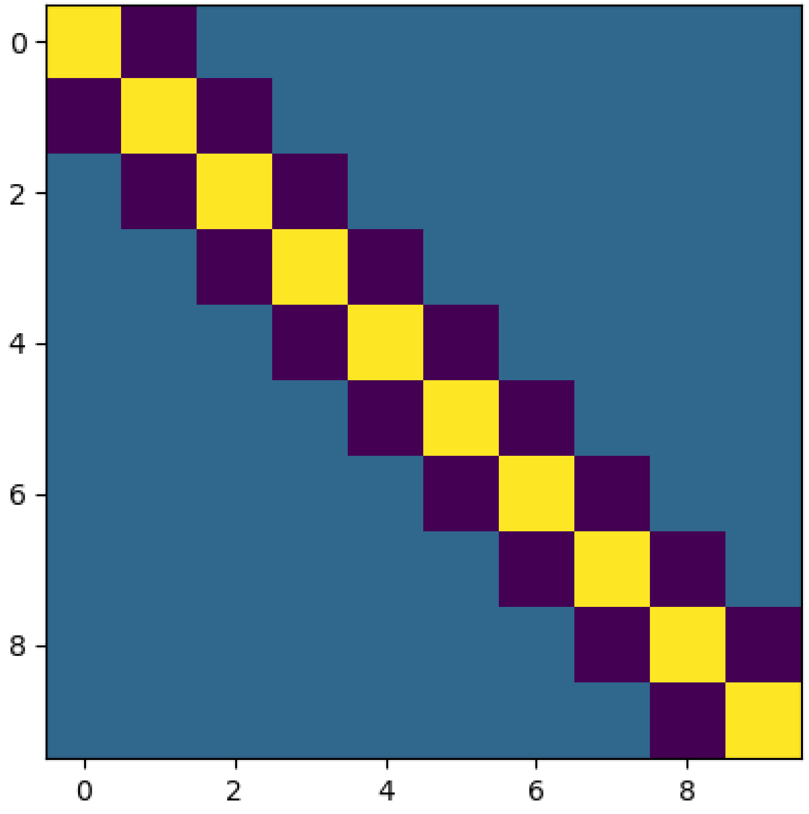
Totally reproducible figures



```
N = 5
treated = (20, 35, 30, 35, 27)
control = (52, 38, 39, 47, 34)
ind = np.arange(N)  # the x locations for the groups
width = 0.35       # the width of the bars

fig, ax = plt.subplots()
rects1 = ax.bar(ind, treated, width, label='Treated')
rects2 = ax.bar(ind+width, control, width, label = 'Control')

# add some
ax.set_ylabel('Binding Energy (kJ/mol)')
ax.set_title('Effect of inhibitor on binding')
ax.set_xticks(ind+width)
ax.set_xticklabels( ('Wild\nType', 'T13G', 'A12G', 'E45A', 'E45S') )
ax.legend()
```

Plotting matrices

```
m=np.diag(2*np.ones(10))+np.diag(-1*np.ones(9),1)+np.diag(-1*np.ones(9), -1)

plt.imshow(m)            # plt.matshow(m) is very similar
```

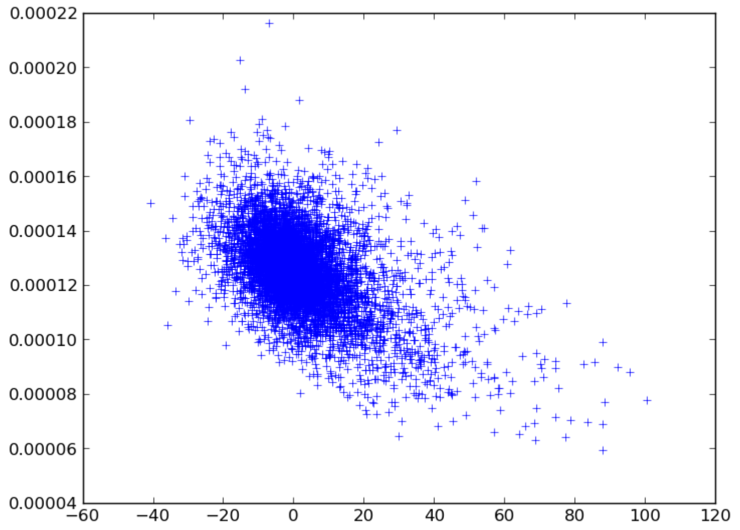


Matplotlib styles

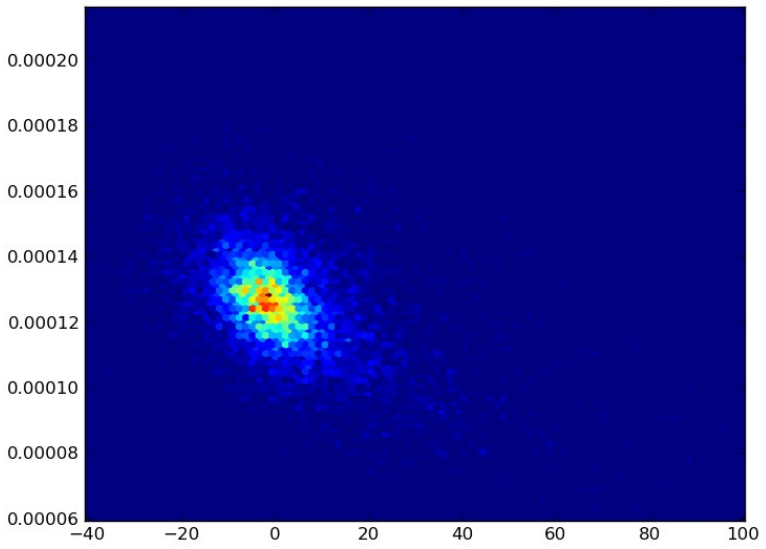
- Since version 1.5 several default styles.
- Try the following code

```
x= np.linspace(0, np.pi, 100)
for s in plt.style.available:
    with plt.style.context(s):
        plt.figure()
        plt.title(s)
        plt.plot(x,np.sin(x)*np.cos(x**2), label='A')
        plt.plot(x,np.sin(x)*np.cos(x**2)*np.cos(x), label='B')
        plt.plot(x,np.sin(x)-np.cos(x)*np.sin(x), label='C')
        plt.legend(loc='best')
```

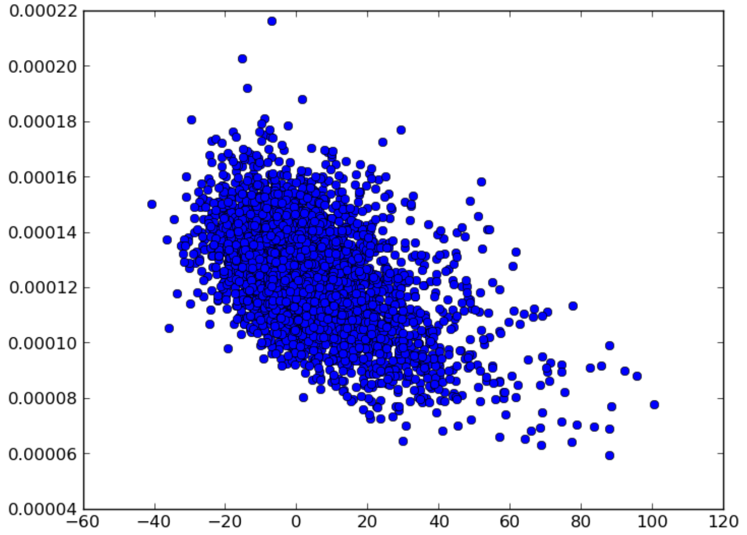
Plotting lots of points:hexbin



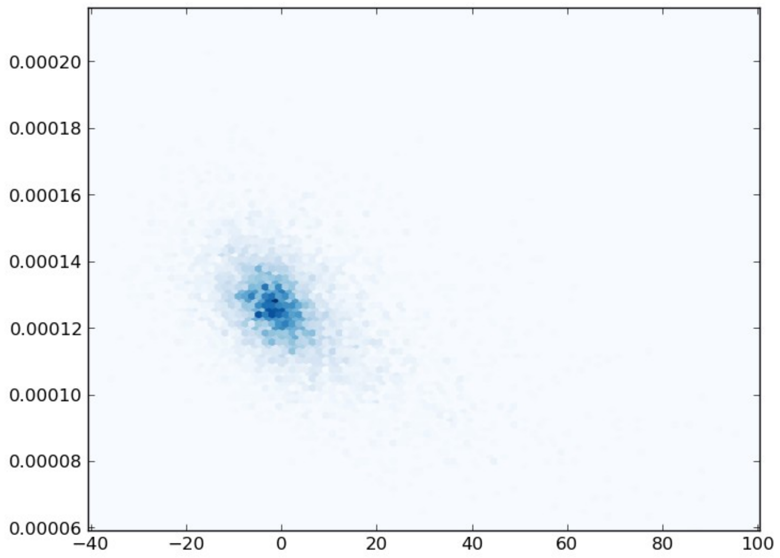
`plt.plot(x, y, '+')`



`plt.hexbin(x, y)`

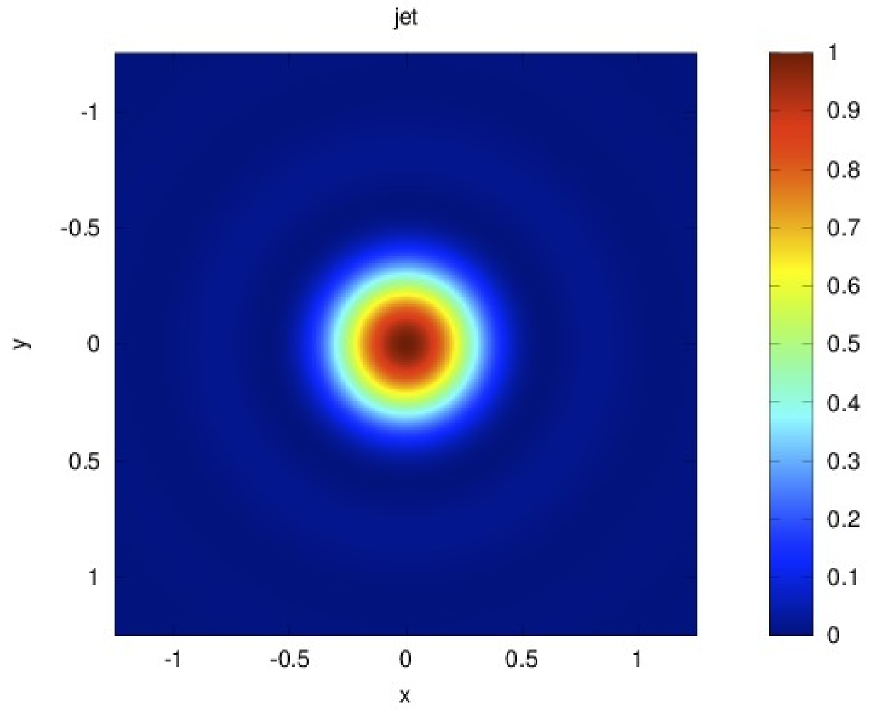


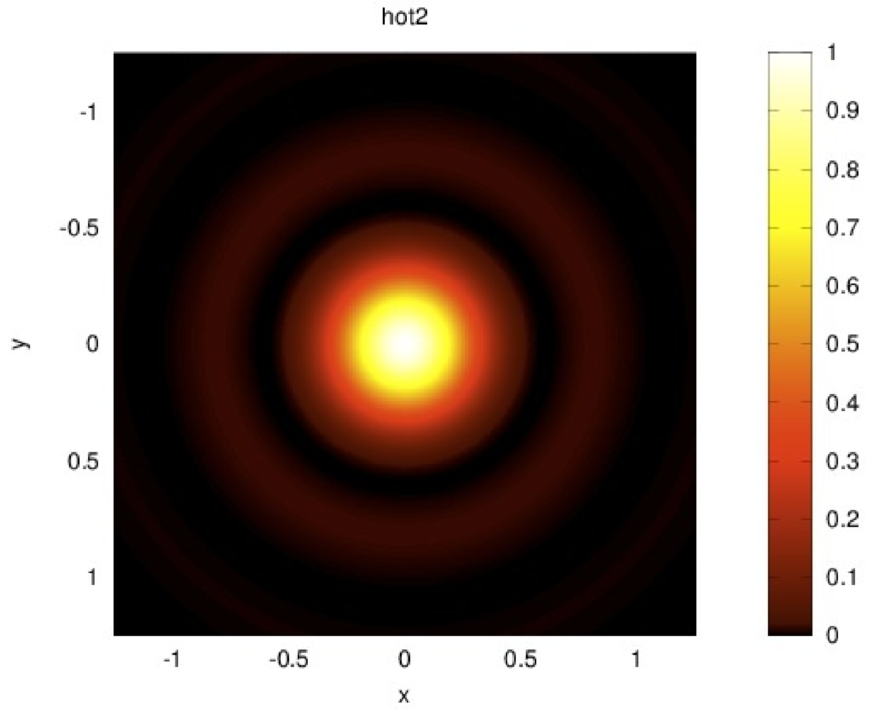
`plt.plot(x, y, 'o')`

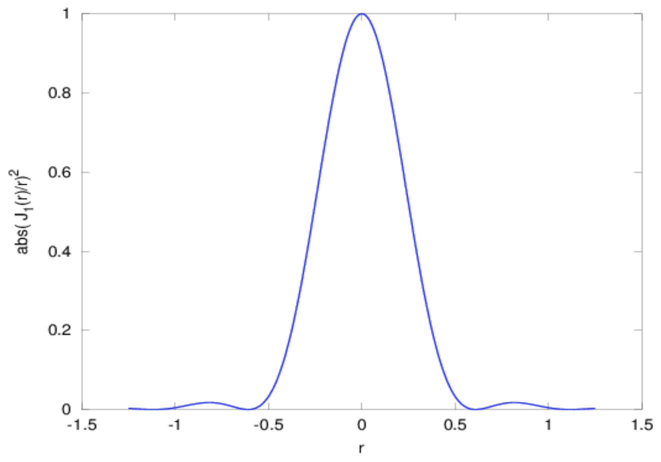


`plt.hexbin(x, y, cmap=pylab.cm.Blues)`

Jet is not a good colormap







http://cresspahl.blogspot.com.es/2012/03/expanded-control-of-octaves-colormap.html
https://jakevdp.github.io/blog/2014/10/16/how-bad-is-your-colormap/

Matplotlib

- Do Lecture-4-Matplotlib.ipynb from http://jrjohansson.github.io/
  - Other interesting material there...
- Check matplotlib gallery
  - http://matplotlib.org/gallery.html
- Quick reference of symbols and colours:
  - http://www.loria.fr/~rougier/teaching/matplotlib/#quick-references

    (part of a larger tutorial)
- Some more tricks and examples:
  - http://wiki.scipy.org/Cookbook/Matplotlib

Extensions

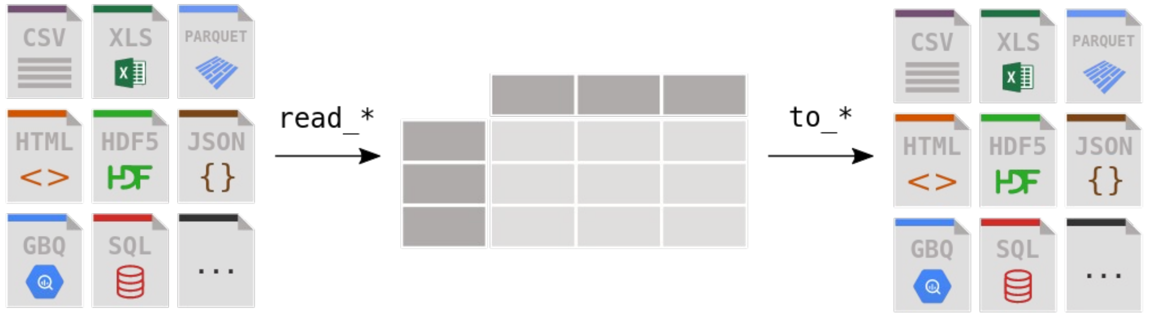
- Seaborn
  - Data visualization
  - Nice color palettes from http://colorbrewer2.org/
  - https://stanford.edu/~mwaskom/software/seaborn/
- Bokeh:
  - http://bokeh.pydata.org/en/latest/
- Plotly:
  - https://plot.ly/
- http://pbpython.com/visualization-tools-1.html

Pandas

Pandas

- Pandas: `import pandas as pd`
  - Dataframes with named columns and rows
  - Similar to a spreadsheet
  - Great for data analysis
  - a dataframe is to a numpy array what a dictionary is to alist
  - https://pandas.pydata.org/
- xarray: `import xarray as xr`
  - labeled multidimensional arrays: n-dimensional pandas-like dataframes
  - https://docs.xarray.dev/en/stable/

Dataframe creation



- Very powerful reading: `read_csv`, `read_fwf`, `read_xml`, `read_excel`...
- From lists or dictionaries:

```
df2 = pd.DataFrame(
       {
           "A": 1.0,
            "B": pd.Timestamp("20130102"),
            "C": np.array([3] * 4, dtype="int32"),
            "D": pd.Categorical(["test", "train", "test", "train"]),
        }
    )

     A          B   C      D
0  1.0 2013-01-02   3   test
1  1.0 2013-01-02   3  train
2  1.0 2013-01-02   3   test
3  1.0 2013-01-02   3  train

df = pd.DataFrame(np.random.randn(6, 4), columns=list("ABCD"))
```

dataframe manipulation

- Selection and sorting
- `df['A']`
- `df.loc[:, ["A", "B"]]`
- `df[df["A"] > 0]`
- `df.sort_index()`
- `df_sort_values(by='A')`

- Poweful joining/merging methods
- Powerful pivoting
- Grouping:

```
     A      B         C         D
0  foo    one  1.346061 -1.577585
1  bar    one  1.511763  0.396823
2  foo    two  1.627081 -0.105381
3  bar  three -0.990582 -0.532532
4  foo    two -0.441652  1.453749
5  bar    two  1.211526  1.208843
6  foo    one  0.268520 -0.080952
7  foo  three  0.024580 -0.264610


df.groupby("A")[["C", "D"]].sum()

            C         D
A
bar  1.732707  1.073134
foo  2.824590 -0.574779
```

seaborn: dataframe plotting

- `import seaborn as sns`
- https://seaborn.pydata.org/

```
dots = sns.load_dataset("dots")
sns.relplot(
    data=dots, kind="line",
    x="time", y="firing_rate", col="align",
    hue="choice", size="coherence", style="choice",
    facet_kws=dict(sharex=False),
)

fmri = sns.load_dataset("fmri")
sns.relplot(
    data=fmri, kind="line",
    x="timepoint", y="signal", col="region",
    hue="event", style="event",
)

tips = sns.load_dataset("tips")
sns.catplot(data=tips, kind="swarm", x="day", y="total_bill",
hue="smoker")
```

Machine Learning

Pandas

- library providing high-performance, easy-to-use **data structures** and data analysis tools for Python
- Watch the video:
  - https://www.youtube.com/watch?v=0CFFTJUZ2dc
- Download the notebook and the data:
  - https://github.com/jonathanrocher/pandas_tutorial
- Check this book and its downloadable notebooks:
  - https://jakevdp.github.io/PythonDataScienceHandbook/

Functions and modules

Functions

defined by def and a colon:

```
def add(x,y):
    return x+y
```

Remember indentation!

Automatic (and recommended) documentation:

```
def add(x,y):
    ””” Returns the
      sum of 2 numbers”””
    return x+y
```

Functions can be seen as both Fortran procedures and functions but...

Arguments are passed by reference

there is access to global variables:

```
> def x_val(): print(x)
> x=60
> x_val()
60
```

Functions II

Function variables are local :

```
> def x_val():
...  x=40
...  print(x)
> x=60
> x_val()
40
> x
60
```

to assign variables, use `return`

```
def x_val():
...  x=40
...  print(x)
...  return x
> x = xval()
40
> x
40
```

Functions III

Mutable objects are passed by reference:

```
> def square_0(lst):
...  lst[0]*=lst[0]
> a=[3,2,1]
> square_0(a)
> a
[9,2,1]
```

Copy variables that need to be preserved:

```
> a_copy=a[:]
> square_0(a)
> import copy
> a_copy=copy.deepcopy(a)
```

Functions IV

Functions can have default arguments :

```
> def submit(job, priority=10,
  nprocs=1):
...  pass
> submit('job1.sh')
```

Function arguments do not have explicit types.

```
> add('Python ', 'summerschool')
Python summerschool
```

Functions can be recursive

```
def fact(n):
  if n == 1:
      return 1
  else:
      return n * fact(n-1)
```

Argument unpacking

Starred arguments are tuples that collect positional arguments :

```
> def prod(*args): ...
> prod(2,3,4)
> x = (4, 5, 6)
> prod(*x)
```

In `prod, args=(2,3,4)`

Keword arguments can be passed as a dictionary:

```
> options = dict(paper='A4', color =
  True)
print_setup(options)
```

Unpacking can be a convenient way to print a list:

```
> vals = [1,2,3,4,5]
> print((4*'{:03d} ').format(*vals))
001 002 003 004
```

https://docs.python.org/3/tutorial/controlflow.html#unpacking-argument-lists

Lists or iterators?

- Lists are iterable objects
- Iterators generate objects on-the-fly
- Iterators can be created with a generator function
  - Uses `yield` satement
- Relevant for efficiency

```
def rang_llista(n):
    result = []
    i = 0
    while i<n:
        result.append(i)
         i += 1
    return result

def rang_gen(n):
    i = 0
    while i<n:
        yield i
        i += 1
```

Modules

- Modules allow packing libraries or extensions
- There are built-in and external modules
- When imported modules are executed
- Modules can be written in C or Fortran!

```
> import math

> m = math

> import math as m

> from math import cos, sin

> from math import * #dangerous. All into the same namespace
```

Modules

- Python checks if a module is already loaded.
  - The interpreter does not reload a module already imported
  - This can cause unexpected behaviour interactively
- Ipython has a more versatile module loading

```
%load_ext autoreload

autoreload 2 #Will reload a module if it changes
```

Some useful modules

- sys — System-specific parameters and functions
- os — Miscellaneous operating system interfaces
- os.path — Common pathname manipulations
- glob — Unix style pathname pattern expansion
- re — regular expressions
- copy — Shallow and deep copy operations
- argparse — Parser for command-line options, arguments and sub-commands
- subprocess — Subprocess management
- inspect — Inspect live objects

Some useful modules

```
if len(sys.argv!=3):
    print('Error: Use two arguments.')
    sys.exit()

method = sys.argv[1]
filelist = glob.glob('/home/ramon/*')
for fileName in filelist:
    if os.path.isfile(fileName): print(fileName)
```

Modules: too many…

```
>>> import math
>>> import cmath
>>> import numpy.lib.scimath as scimath
>>> math.sqrt(4)
2.0
>>> math.sqrt(-4)
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
ValueError: math domain error
>>> cmath.sqrt(4)
(2+0j)
>>> cmath.sqrt(-4)
2j
>>> scimath.sqrt(4)
2.0
>>> scimath.sqrt(-4)
2j
```

Working with your modules

- Import reads from local directory and from the directories in `sys.path` (`import sys` first)
- Put your modules in a directory and add it to the environtment variable `$PYTHONPATH`.
- Python will add the directories in `$PYTHONPATH` to `sys.path`
- Document your modules and the functions therein.
- Use `if __name__=='__main__':` to execute code only if Python is running the module, and not if it is imported.
  - http://stackoverflow.com/questions/419163/what-does-if-name-main-do

Installing external Modules

- Use conda distribution. Then `$ conda install module`
- Many come as part of the linux distributions (usually older versions that those in conda or PyPI)
  - ipython, numpy, biopython...
- For modules in the PyPI repository(most of them) https://pypi.python.org/pypi
  - `(sudo) pip3 install module`
- Manual installation (dependencies have to be also manually installed):

```
$ python setup.py build
$ (sudo) python setup.py install
```

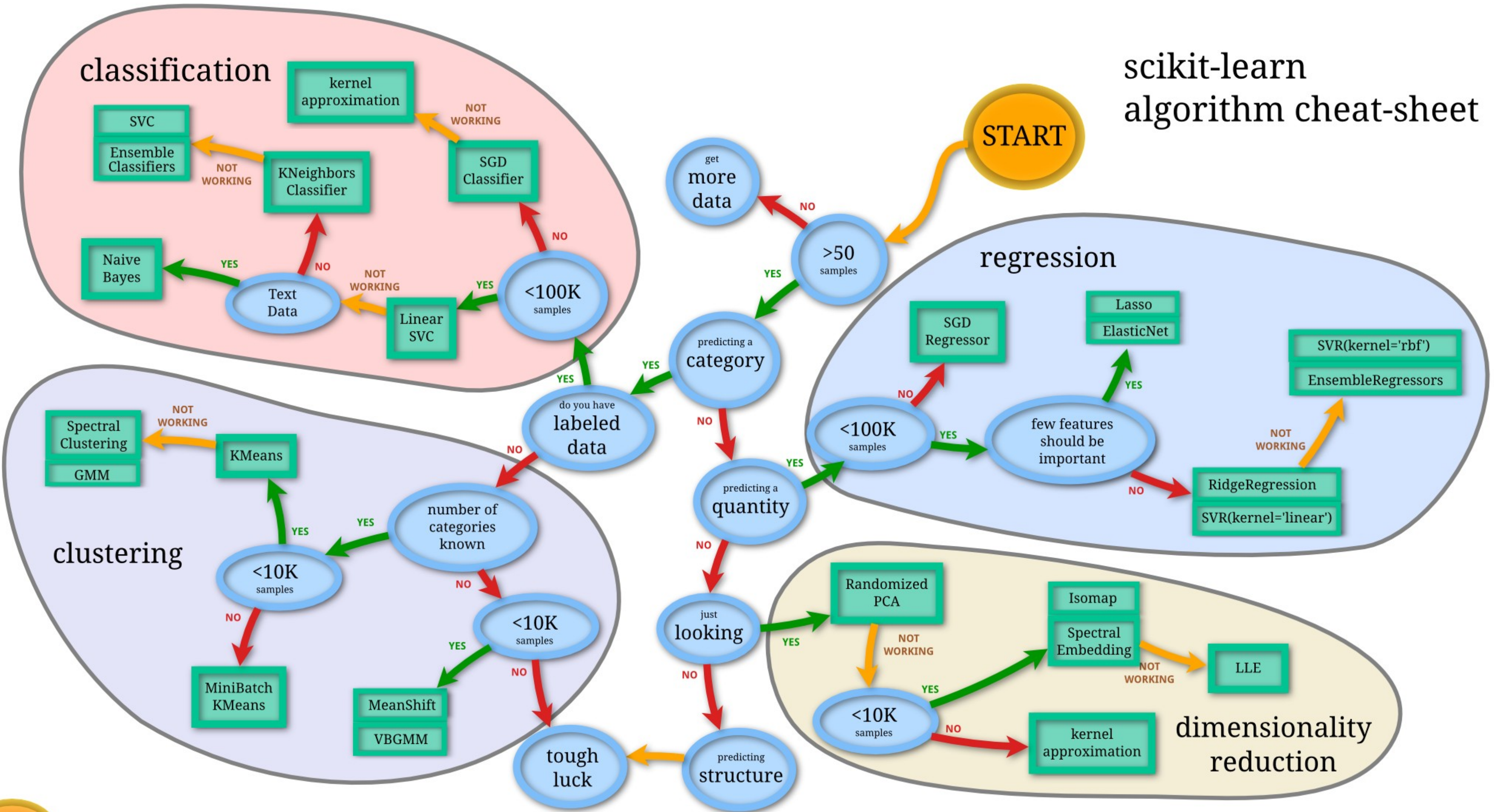
Updating external Modules

- With conda `$ conda update module`
- For modules in the PyPI repository(most of them) https://pypi.python.org/pypi
  - `pip3 install -U module`
- `pip` can also be used in the conda installation.
- Remeber that modules are installed for a given version of python. If you have python 2.x and 3.x you need to check for which version you are installing. For example using `pip3` or

```
$ which pip
```

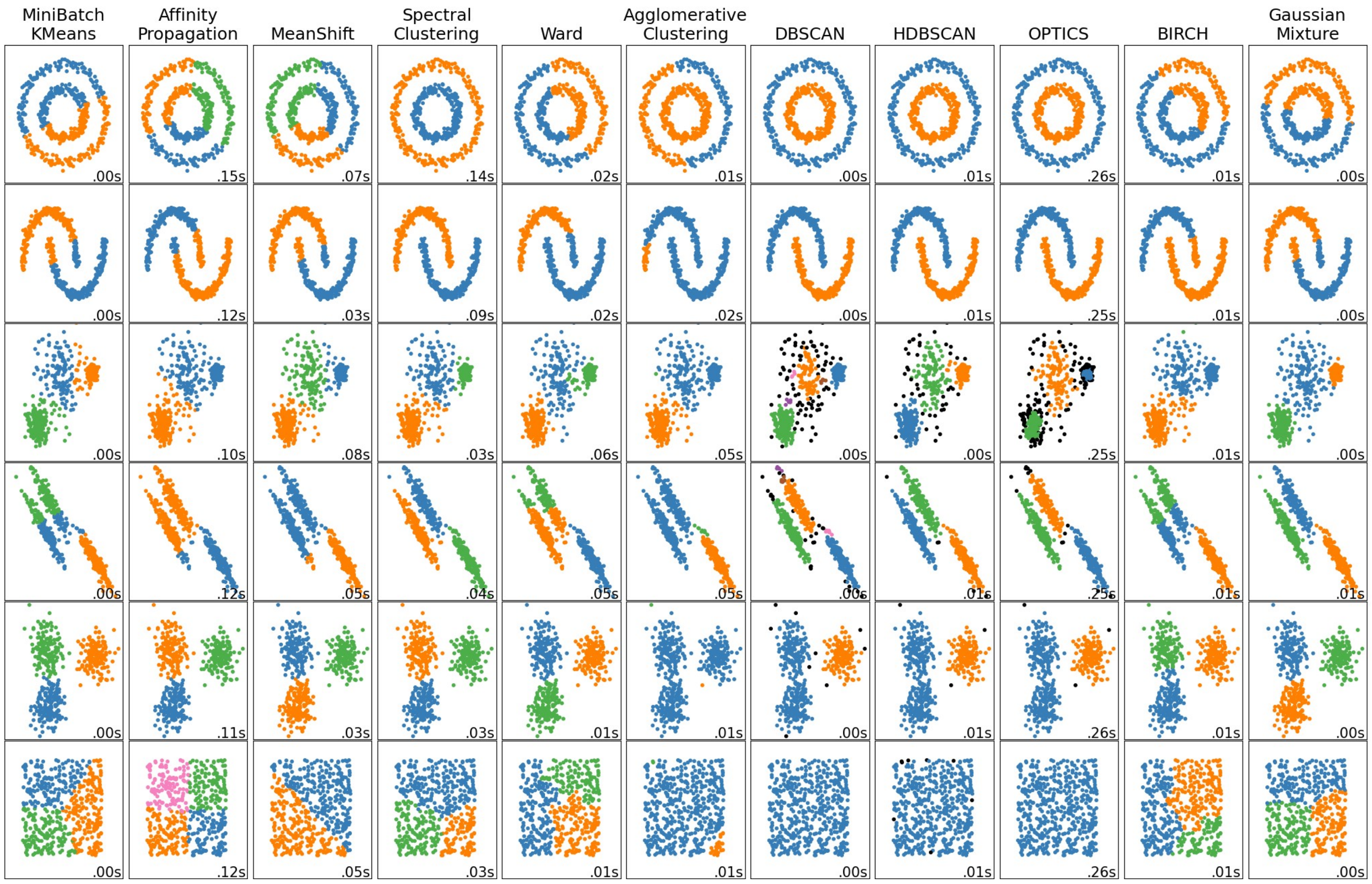
Machine Learning

scikit-learn



Back

```
>>> from sklearn import linear_model
>>> reg = linear_model.LinearRegression()
>>> reg.fit([[0, 0], [1, 1], [2, 2]], [0, 1, 2])
LinearRegression()
>>> reg.coef_
array([0.5, 0.5])
```



Scipy

Linear algebra

- Support for LAPACK, BLAS and ATLAS

```
> A=np.array(np.random.rand(5,5))
> linalg.inverse(A)
> linalg.det(A)
> linalg.eigvals(A)
> linalg.eig(A)
> linalg.svd(A)
> linalg.cholesky(A)
```

- **A.x=b**

```
> b=np.array(random.rand(5)).reshape((5,1))
> linalg.solve(A,b)
```

`scipy.linalg` contains all the functions in `numpy.linalg.` plus some other more advanced ones not contained in `numpy.linalg`

Optimization

- There are different optimization methods:

```
> import scipy.optimize as so
```

- Some only need the function value:

```
> fmin, fmin_powell
```

- Some need the gradient or the hessian:

```
> fmin_cg, fmin_bfgs, fmin_ncg
```

- Some look for global minima:

```
> anneal
```

- Remember:

```
> scipy.info('optimize')
```

- Pedagogical documentation:
- http://docs.scipy.org/doc/scipy/reference/tutorial/optimize.html
- http://docs.scipy.org/doc/scipy/reference/optimize.html

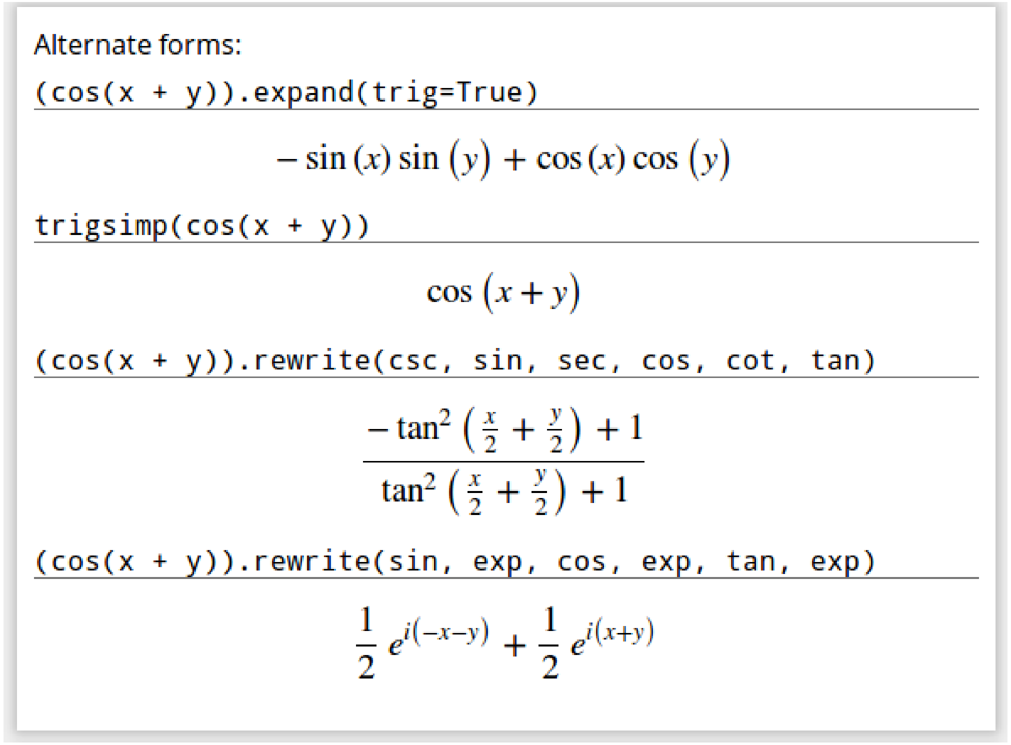
Big data, big memory

- Numpy arrays are meant to live in memory
- If that is not possible:
  - Use op= operations (they use half the memory):
    - `p *=alpha` is better than `p = p*alpha`
  - Use `scipy.sparse` matrices
    - http://docs.scipy.org/doc/scipy/reference/sparse.html
  - Use `PyTable` to store (compressed) matrices on disk
    - http://www.pytables.org/
  - Modify your algorithm to work with submatrices

Sympy: Symbolic math

- Symbolic algebra
- Analytic solution of equations
- Integration, derivation
- Polynomials
- Limits

http://sympy.org/en/index.html



Alternate forms:

```
(cos(x + y)).expand(trig=True)
```

$$-\sin(x)\sin(y) + \cos(x)\cos(y)$$

```
trigsimp(cos(x + y))
```

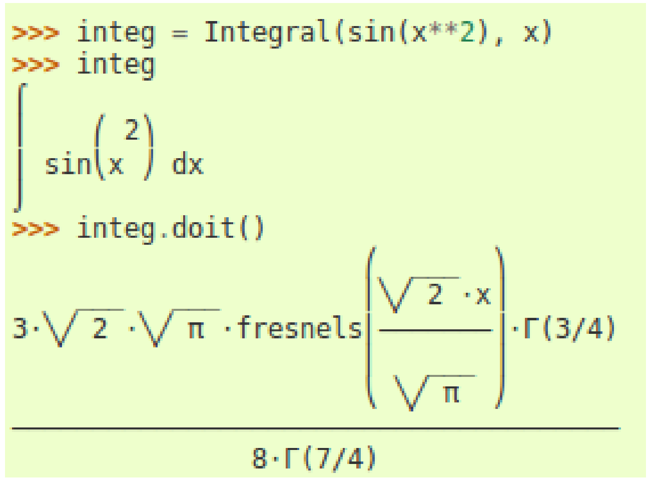
$$\cos(x+y)$$

```
(cos(x + y)).rewrite(csc, sin, sec, cos, cot, tan)
```

$$\frac{-\tan^2\left(\frac{x}{2}+\frac{y}{2}\right)+1}{\tan^2\left(\frac{x}{2}+\frac{y}{2}\right)+1}$$

```
(cos(x + y)).rewrite(sin, exp, cos, exp, tan, exp)
```

$$\frac{1}{2}e^{i(-x-y)} + \frac{1}{2}e^{i(x+y)}$$



```
>>> integ = Integral(sin(x**2), x)
>>> integ
⌠
⎮    ⎛ 2⎞
⎮ sin⎝x ⎠ dx
⌡
>>> integ.doit()
                       ⎛√2 ·x⎞
3·√2 ·√π ·fresnels⎜─────⎟·Γ(3/4)
                       ⎝ √π  ⎠
─────────────────────────────────
             8·Γ(7/4)
```

Add-ons

Add ons: Biopython

Biopython

```
from Bio.PDB import *
p=PDBParser(PERMISSIVE=1)
s=p.get_structure('1OJR', filename)
```

Print out the coordinates of all CA atoms with B factor > 50:

```
for model in s.get_list():
  for chain in model.get_list():
    for residue in chain.get_list():
      if residue.has_id("CA"):
        ca=residue["CA"]
        if ca.get_bfactor()>50.0:
          print ca.get_coord()
```

http://biopython.org

Add ons: Machine learning and statistics

- Basic statistics in scipy.stats
  - Tutorial: http://docs.scipy.org/doc/scipy/reference/tutorial/stats.html
  - Reference: http://docs.scipy.org/doc/scipy/reference/stats.html
- Machine learning with sklearn
  - http://scikit-learn.org/stable/
  - Choosing the method: http://scikit-learn.org/stable/tutorial/machine_learning_map/
- More algorithms (and a textbook) with AstroML
  - http://www.astroml.org/

Add ons: itertools

```
> import itertools
> perms = itertools.permutations('ABC', 3)
> list(perms)
[('A', 'B', 'C'),
 ('A', 'C', 'B'),
 ('B', 'A', 'C'),
 ('B', 'C', 'A'),
 ('C', 'A', 'B'),
 ('C', 'B', 'A')]
> list(itertools.combinations('ABC',2))
[('A', 'B'), ('A', 'C'), ('B', 'C')]
```

Optimization and debugging

Optimization

- “Premature optimization is the root of all evil” Knuth
- `%timeit a=np.random.random(100000)`
-
```
a=np.random.random(100000)
n_dim=3
%%timeit
x=np.zeros(shape=(100000,n_dim),order='F')
for j in range(0,n_dim):
    x[:,j]=a*j
```
- Evaluated in a separate enviroment

Exceptions and errors

Although the language is interpreted there are some syntax errors that prevent execution:

```
def safe_divide_1(x, y)


File"/home/ramon/python/prova.py",
   line 1

    def safe_divide_1(x, y)

                           ^

SyntaxError: invalid syntax
```

Exceptions leave a trace easy to follow.

Easy debugging with

`%pdb`

`%debug`

pdb: python debugger

```
In [1]: pdb
Automatic pdb calling has been turned ON

In [4]: run foo.py
NameError: name 'b' is not defined
> /home/ramon/python/foo.py(2)<module>()
      1 a = 3
----> 2 print(b)

ipdb> ?
```

pdb: python debugger

```
In [9]: run foo.py
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
/home/ramon/python/foo.py in <module>()
      1 a = 3
----> 2 print(b)

NameError: name 'b' is not defined

In [10]: %debug
> /home/ramon/python/foo.py(2)<module>()
      1 a = 3
----> 2 print(b)
ipdb>
```

Numba, CuPy, JAX, Julia

- Numba:
  - just in time compiler. Can use CUDA (GPU's)
- Cupy:
  - library for GPU-accelerated computing with Python
- JAX
  - NumPy on the CPU, GPU, and TPU, with great automatic differentiation for high-performance machine learning research
- Julia is a (my definitions, probably not accurate):
  - Compiled Python
  - Interactive Fortran
- Cython:
  - C in Python

https://gist.github.com/jpivarski/da343abd8024834ee8c5aaba691aafc7#file-mandelbrot-on-all-accelerators-ipynb

Numba

- Numba compiles in a virtual machine.
- Developped by Continuum analytics, so easiest install from conda.
- `$ conda install numba`

Numba

- Use Numba Just in time compiler:

  -
    ```
    @numba.jit(nopython=True, parallel=True)
    def f1(a, b):
       return a**2+b**2
    ```
- Or vectorize in cpu or gpu:
  ```
  @numba.vectorize(['float64(float64, float64)'],nopython=True)
  def f1b(a, b):
     return a**2+b**2
  @numba.vectorize(['float64(float64, float64)'],target='cuda')
  def f1c(a, b):
     return a**2+b**2
  ```

Linear algebra & FFT on GPUs

When to use it?

- https://www.anaconda.com/blog/getting-started-with-gpu-computing-in-anaconda

- GPU use is complex. Input output is slow.
  - Memory management is complex
- All are Machine Learning Oriented.
  - Tensorflow
  - Pytorch
- Others even more machine learning oriented.
- https://docs.anaconda.com/anaconda/user-guide/tasks/gpu-packages/

Resources

Resources

On-line Official documentation (contains Tutorial in PDF or HTML):

http://www.python.org/doc

General introductory books (also in paper):

http://diveintopython.org/ (This one is simpler!)

http://www.greenteapress.com/thinkpython/thinkpython.html

Comparison of codes in different languages:

http://rosetacode.org

http://www.codecodex.com

Python package index: where to find modules

http://pypi.python.org/pypi

Resources

- Interactive tutorial
  - http://pythonmonk.com/
- A Crash Course in Python for Scientists (with applications in Quantum chemistry)
  - http://nbviewer.ipython.org/5920182
  - Written in an ipython notebook
- Python Scientific Lecture notes
  - http://scipy-lectures.github.io/
- Python flow with Pythontutor
  - http://www.pythontutor.com

Resources

- Python for geosciences
  - http://earthpy.org/tag/python-for-geosciences.html
- Bioinformatics with BioPython:
  - http://biopython.org/

Resources: Books

- Rossant, C, *Learning Ipython for Interactive Computing and Data Visualization.*
  - Basic level. Covers several subjects, includng matplotlib and parallelism. Recipes book.
- Vanderplas, J. *Python Data Science Handbook.*
  - Online: https://jakevdp.github.io/PythonDataScienceHandbook/
- DeCaria A. J. *Python Programming and Visualization for Scientists*
  - DeCaria teaches Python programming and visualization for meteorology and ocean sciences majors.
- Packt Publishing. Wide variety, lots on GIS and Python.
- https://wiki.python.org/moin/AdvancedBooks

Resources: Video Tutorials

- Check: https://www.youtube.com/user/EnthoughtMedia
- Check: http://www.pyvideo.org/
- Check Scipy Conference and Euroscipy:
  - https://conference.scipy.org/proceedings/scipy2023/
  -

Resources: Teaching

- On teaching programming with Python 3

  http://www.comp.leeds.ac.uk/nde/papers/teachpy3.html
- Online Syntax Highlighting

  http://tohtml.com/python/
- Style Guide for Python Code:
- www.python.org/dev/peps/pep-0008/

K. Hinsen views

- “NumPy has introduced incompatible changes with almost every new version over the last years”
- “Given the importance of NumPy in the scientific Python ecosystem, I consider its lack of stability alarming”.
- “What makes me hesitate to recommend not using Python is that there is no better alternative”.
- https://khinsen.wordpress.com/2014/09/12/the-state-of-numpy/

Jake VanderPlas

- Great blog about python with applications in
  - Science
  - Statistics
  - Cycling...
  - All entries are jupyter notebooks.
  - https://jakevdp.github.io/
  - See also his book and library on machine learning:
  - http://www.astroml.org/
  - http://press.princeton.edu/titles/10159.html