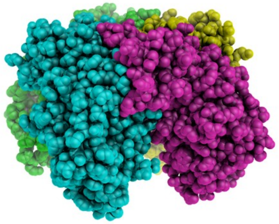
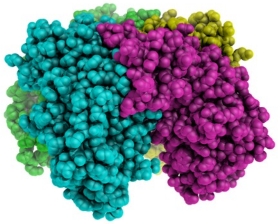


Eines Informàtiques: Python

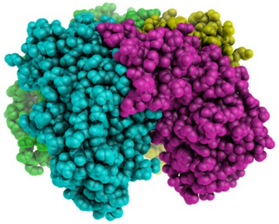
Ramon Crehuet

Curs 2020-2021

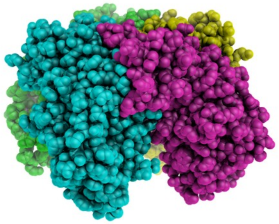


Overview

- Why Python
- Language basics
- Functions and modules
- Working with files
- Classes and objects (bare minimum!)
- Working with arrays: Numpy
- Data visualization
- Scientific modules. Scipy
- Other scientfic modules: Pandas, sckikit-learn, biopython
- Profiling and optimization
- Beyond Python

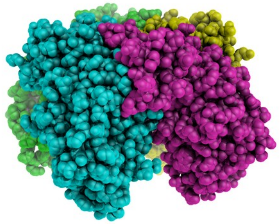


Introduction



Overview

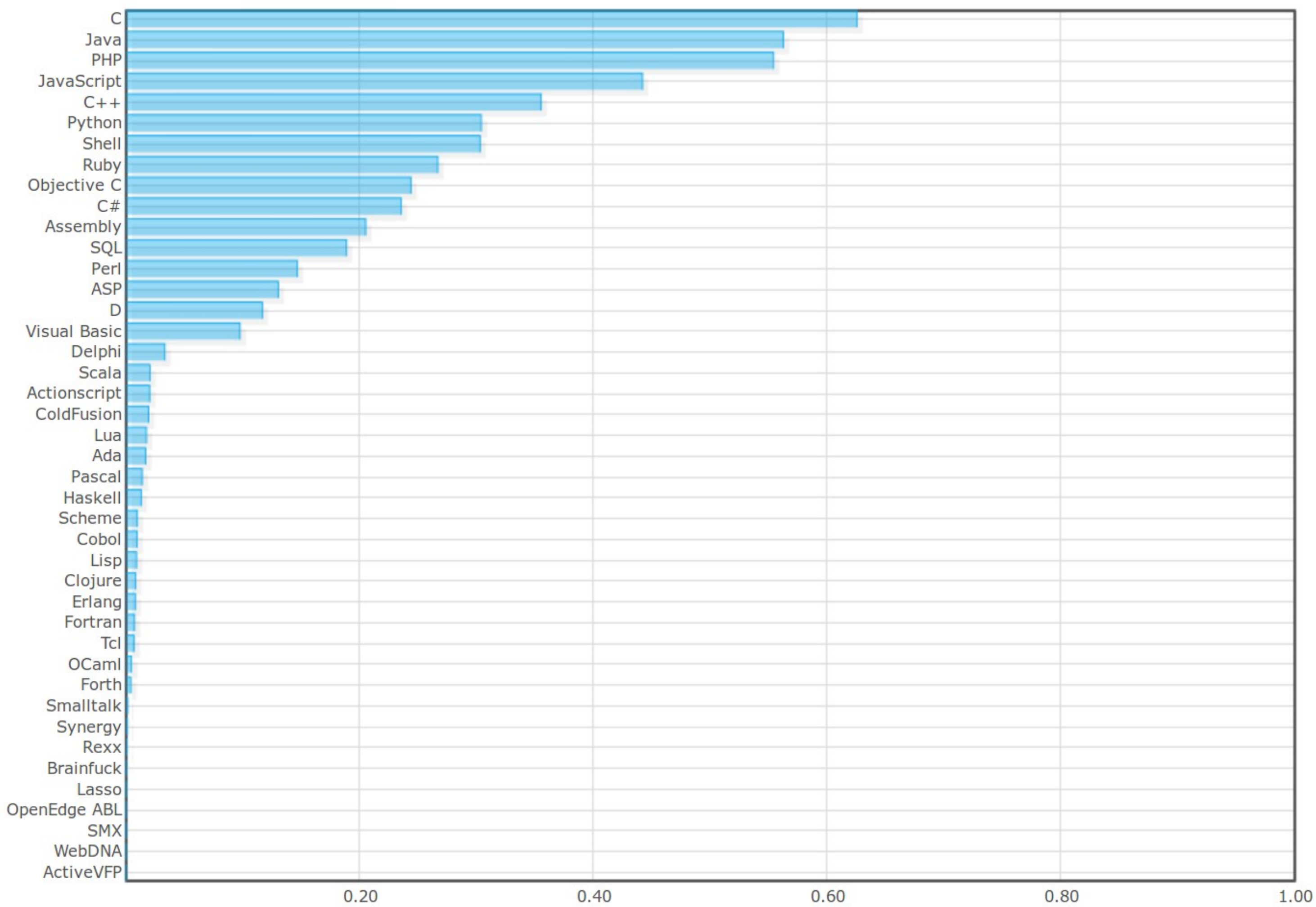
- Why Python
- Language basics
- Functions and modules
- Working with files
- Classes and objects (bare minimum!)
- Working with arrays: Numpy
- Data visualization
- Scientific modules. Scipy
- Other scientfic modules: Pandas, sckikit-learn, biopython
- Profiling and optimization
- Beyond Python



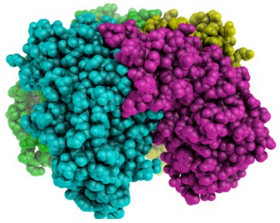
Language popularity

Normalized Comparison

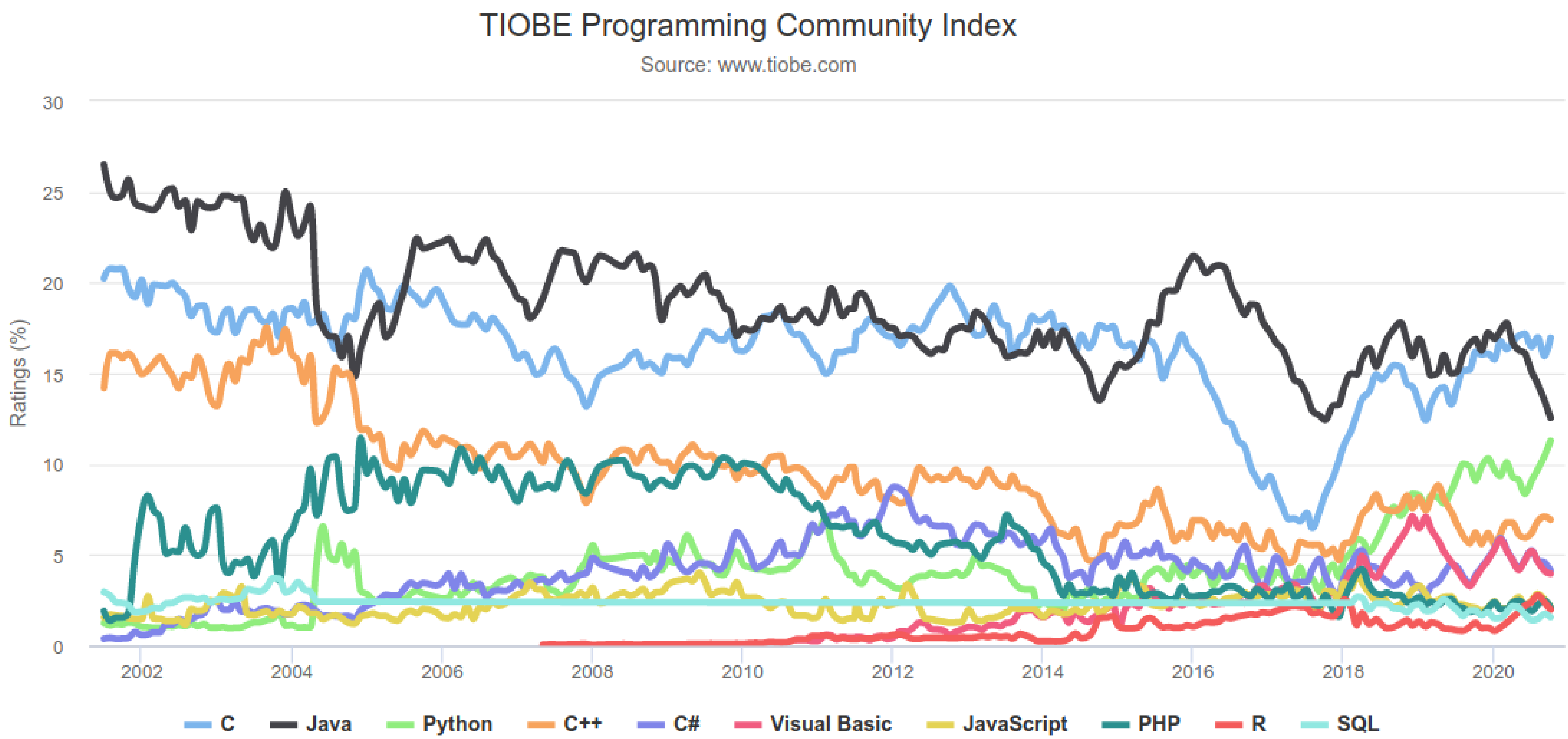
This is a chart showing combined results from all data sets, listed individually below.



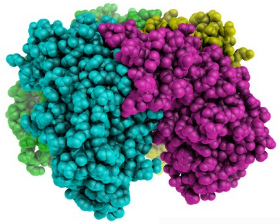
http://langpop.com/



Language popularity



http://www.tiobe.com/index.php/content/paperinfo/tpci/index.html



Hammerprinciple.com

THIS LANGUAGE ENCOURAGES WRITING CODE THAT IS EASY TO MAINTAIN.

93% 7%

Python — 54 out of 58 picked Python over R — R

THIS IS A MAINSTREAM LANGUAGE

92% 8%

Python — 62 out of 67 picked Python over R — R

THIS LANGUAGE IS GOOD FOR BEGINNERS

92% 8%

Python — 71 out of 77 picked Python over R — R

I WOULD USE THIS LANGUAGE AS A SCRIPTING LANGUAGE EMBEDDED INSIDE A LARGER APPLICATION

91% 9%

Python — 62 out of 68 picked Python over R — R

I WOULD USE THIS LANGUAGE FOR WRITING SERVER PROGRAMS

90% 10%

Python — 57 out of 63 picked Python over R — R

I WOULD USE THIS LANGUAGE FOR MOBILE APPLICATIONS

90% 10%

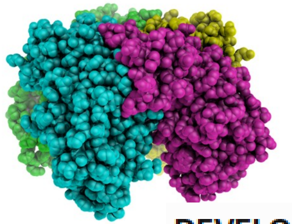
Python — 55 out of 61 picked Python over R — R

I CAN IMAGINE THIS WILL BE A POPULAR LANGUAGE IN TWENTY YEARS TIME

http://hmrp.pl/x79RTk#73

90% 10%

Python — 64 out of 71 picked Python over R — R



Hammerprinciple.com

DEVELOPERS WHO PRIMARILY USE THIS LANGUAGE OFTEN BURN OUT AFTER A FEW YEARS

23% 77%

Python | 43 out of 56 picked R over Python | R

THE SEMANTICS OF THIS LANGUAGE ARE MUCH DIFFERENT THAN OTHER LANGUAGES I KNOW.

22% 78%

Python | 52 out of 67 picked R over Python | R

WRITING CODE IN THIS LANGUAGE IS A LOT OF WORK

18% 82%

Python | 64 out of 79 picked R over Python | R

THIS LANGUAGE HAS A NICHE IN WHICH IT IS GREAT

18% 82%

Python | 60 out of 74 picked R over Python | R

THIS LANGUAGE HAS AN ANNOYING SYNTAX

17% 83%

Python | 48 out of 58 picked R over Python | R

THIS LANGUAGE IS UNUSUALLY BAD FOR BEGINNERS

16% 84%

Python | 68 out of 81 picked R over Python | R

I OFTEN FEEL LIKE I AM NOT SMART ENOUGH TO WRITE THIS LANGUAGE

13% 87%

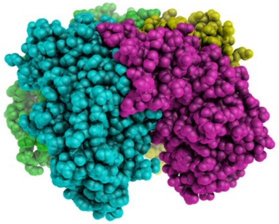
Python | 38 out of 44 picked R over Python | R

THIS LANGUAGE HAS A NICHE OUTSIDE OF WHICH I WOULD NOT USE IT

http://hmrp.pl/x79RTk#27

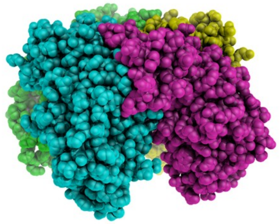
8% 92%

Python | 66 out of 72 picked R over Python | R



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.



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)”

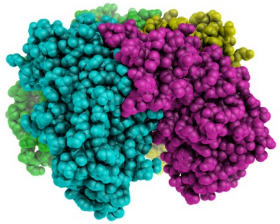
Biophysical Journal, **104**, 2013, 488–495

OS calls

File parsing

Numerical analysis

Data visualization

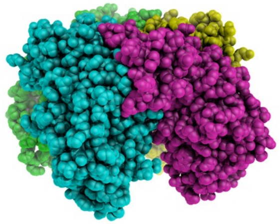


Python for science

Compiled languages
Fast
Difficult
non-interactive

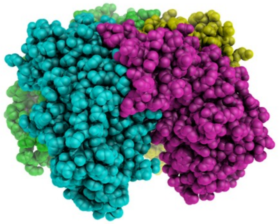
Matlab, Mathematica, Octave
Slow
Rich libraries
Nice development environment
Restricted base language
Expensive (some)

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



Matlab, Mathematica?

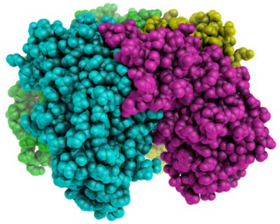
- Scientific computing:
  - ipython + scipy + matplotlib
- Free
- Open source
- Extensible
- Bioinformatics
  - Biopython
- Molecular Dynamics
  - MMTK
- Efficiency
  - Numba, Cython, Fortran, C
- Server control
- XML parser



Low level vs. high level

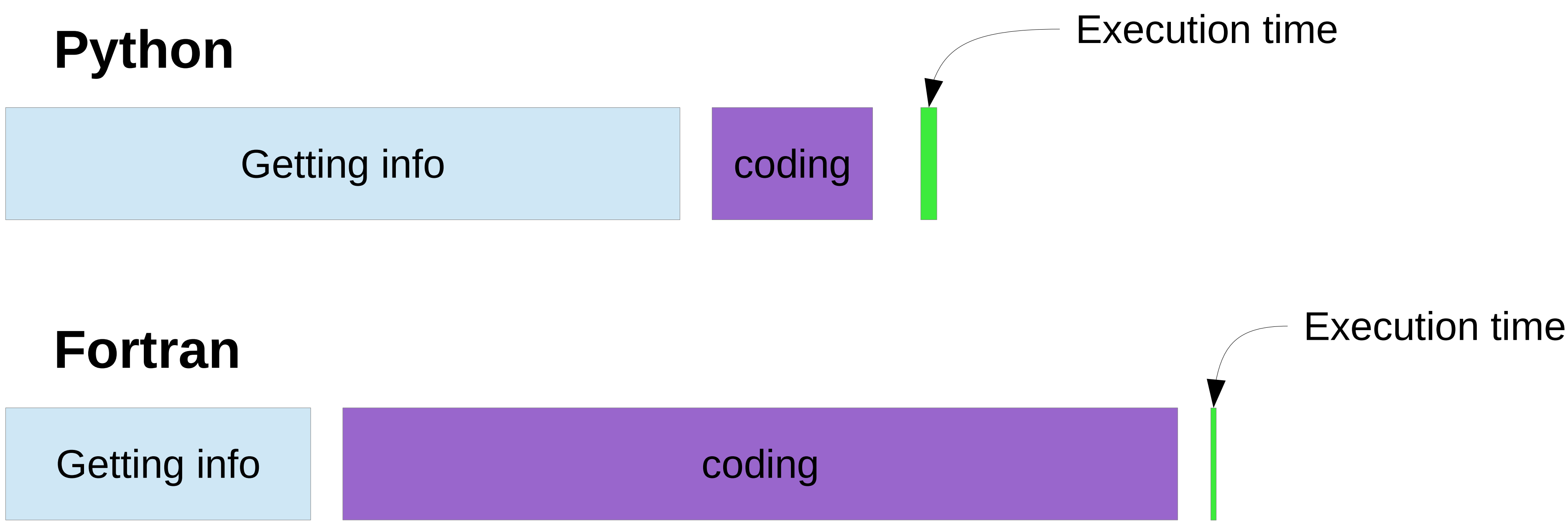
- Python is a high level language
- You can focus on:
  - Low level issues
  - Higher complexity of problems

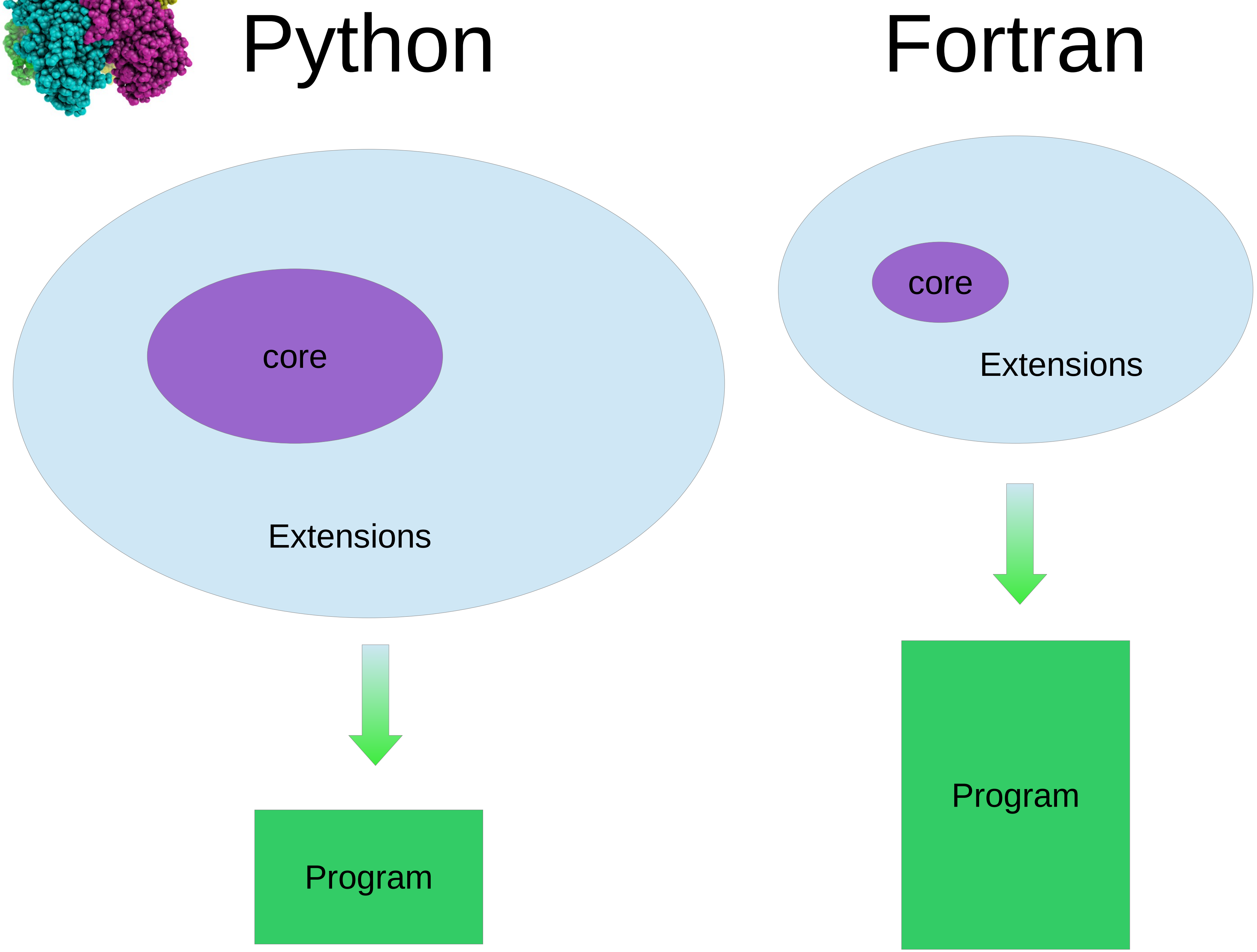
- Low level issues
  - Variable types
  - Machine precision
- But also
  - Extend
  - Mantain
  - Document code

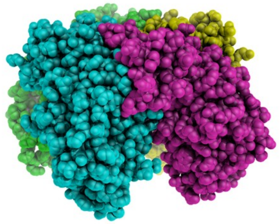


Python vs. Fortran

Different time distribution to get a task done




Python
Fortran
core
Extensions
Program
core
Extensions
Program



Python for science

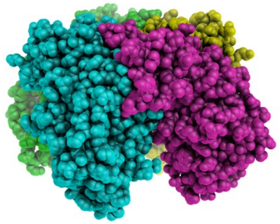
- The homogenization of scientific computing, or why Python is steadily eating other languages' lunch
  http://www.talyarkoni.org/blog/2013/11/18/the-homogenization-of-scientific-computing-or-why-python-is-steadily-eating-other-languages-lunch/
- 10 Reasons Python Rocks for Research (And a Few Reasons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

Hello World program

```
print(“Hello World!”)
```

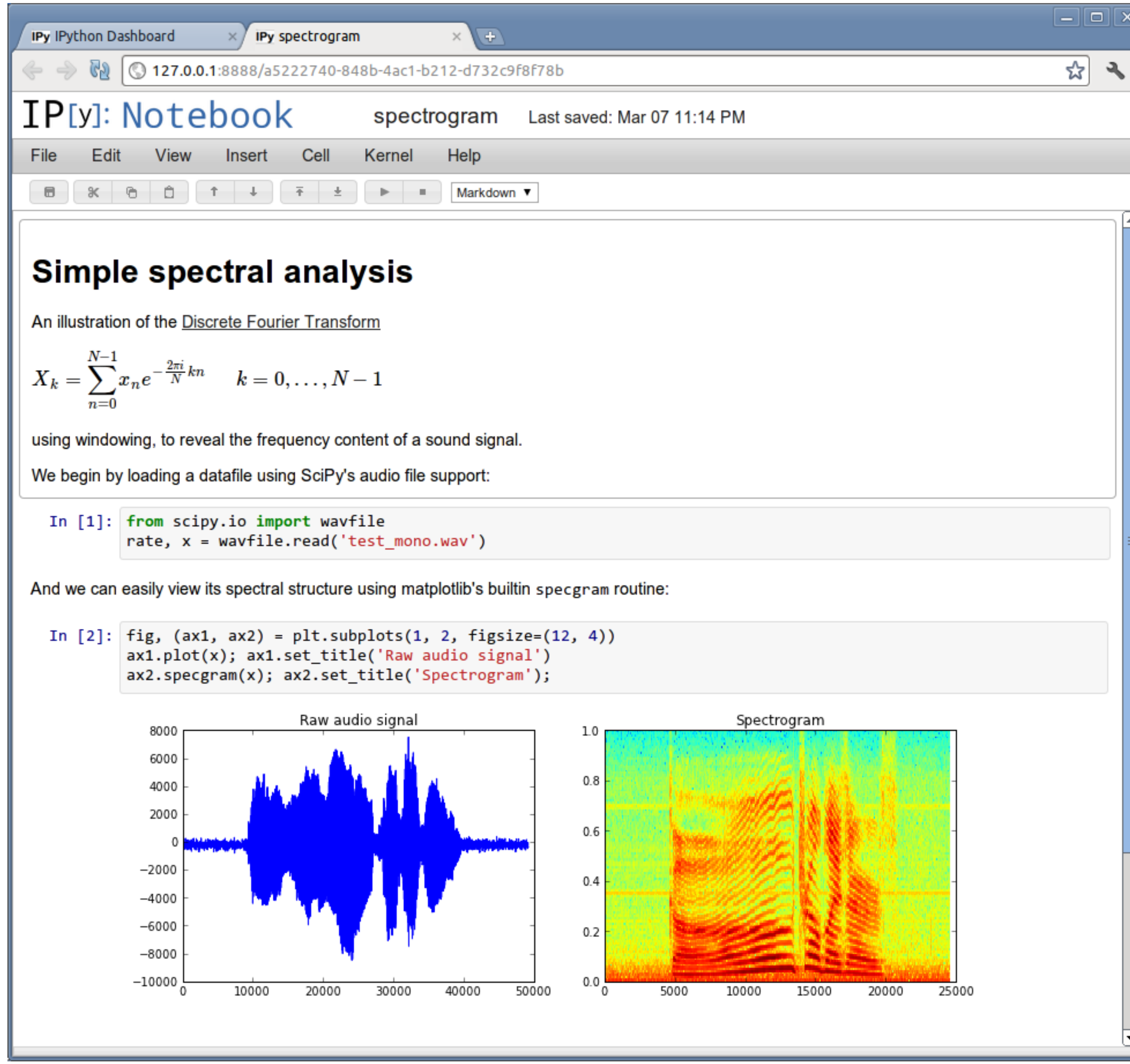
```
print(“Hello World!”)
```

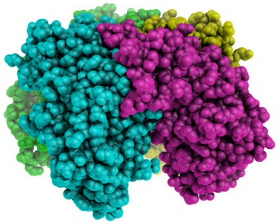
```
$ python3 hello.py
```



Interactive shells

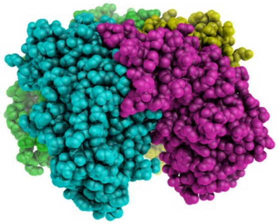
- python
- IDLE
- **ipython**
  - shell
  - notebook
- spyder
- visual studio code
- PIDA
- Sage





Dynamically typed

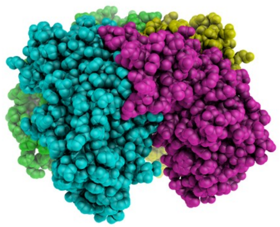
```
>>> a = 4
>>> type(a)
<class 'int'>
>>> b = 7.6
>>> type(b)
<class 'float'>
>>> type(a+b)
<class 'float'>
>>> c = 'Hola'
>>> c + ' Que tal?'
'Hola Que tal?'
>>> c + a
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: Can't convert 'int' object to str implicitly
```



Which python version?

- Language is fast evolving
- 2 versions now coexist: 3.x and 2.x
- These versions are not completely compatible
- 3.x is better and continued
- 2.x has some software still not ported
- Both can safely coexisit
  - Packages and shells are for a specific version
- `2to3 -w hello.py`

Language elements



Numbers

Integers:

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

Limited by amount of memory:

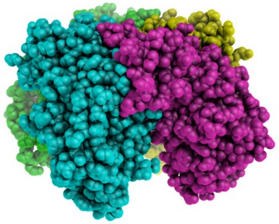
```
>>> i.bit_length()
3
>>> j = i**i**i
>>> j.bit_length()
7257
>>> 9 % 5 #modulo
4
```

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:

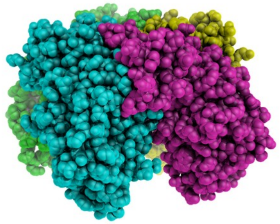
```
> i = i+1
> j = j / 10.
```

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']
```

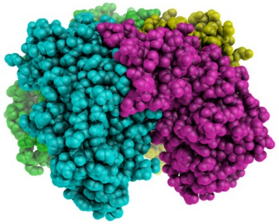
```
> print('Result: %5.3f' % (11./3.))
3.667
```

Non mutable:

```
> s[6]
> s[6]='7'
```

Regular expressions

```
import re
```



Lists, sets and tuples

- Fortran **`dimension`**:

  much more flexible

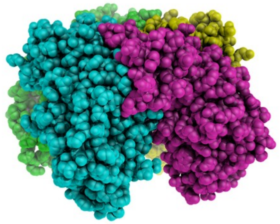
  ```
  > l=[6, 'a', [5,[9,8,7,6]], -
     6.5, (True, True)]
  > [1,2]+[3,4]
  > l.append(6)
  ```

- 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

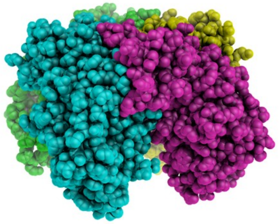
List indexing and methods:

```
> 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

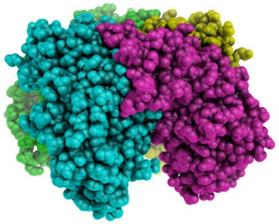
Set methods:

```
> 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
l1
[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)
```

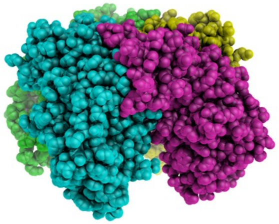
Dictionaries are not ordered

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.

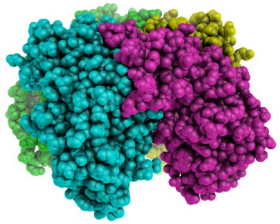
Python makes this the syntax rule to identify blocks.

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")
```

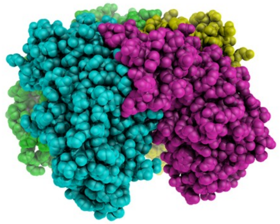


Execution control: if

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:

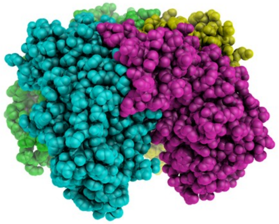
```
and or not ( )
```

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>
```

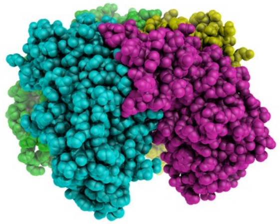
Break continue pass

```
> pass # does nothing
```

`break`: Fortran `EXIT`

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

`cycle`: Fortran `CONTINUE`



list comprehension and enumerate

simple way to create lists:

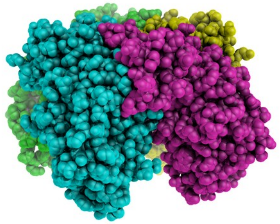
```
> l=[x**2 for x in range(8)]
[0, 1, 4, 9, 16, 25, 36, 49]
```

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)]
```



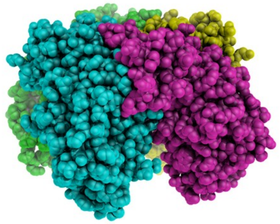
list comprehension and enumerate

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])
```

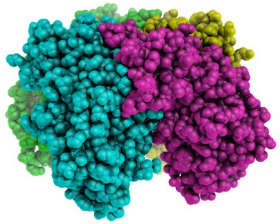
Or with list comprehensions

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

Or with functional programming

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

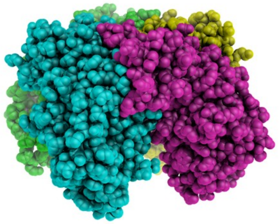
http://docs.python-guide.org/en/latest/writing/style/



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])
```

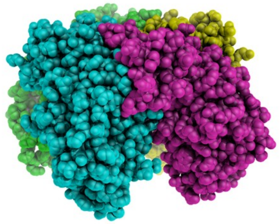
http://docs.python.org/3.3/library/functions.html



Mutable and immutable

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

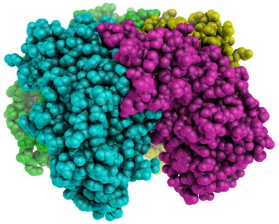
```
>>> a = 4
>>> id(a)
9157088
>>> a += 2
>>> id(a)
9157152
>>> s = 'Hola'
>>> id(s)
140165884365656
>>> s = s+ ' que tal?'
>>> id(s)
140165884365712
>>> ll = [3,4,5]
>>> id(ll)
140165884674416
>>> ll.append(6)
>>> id(ll)
140165884674416
```



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:

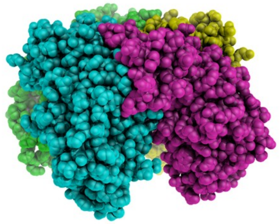
```
> c = 4+5j
> c.real
```

Objects have methods:

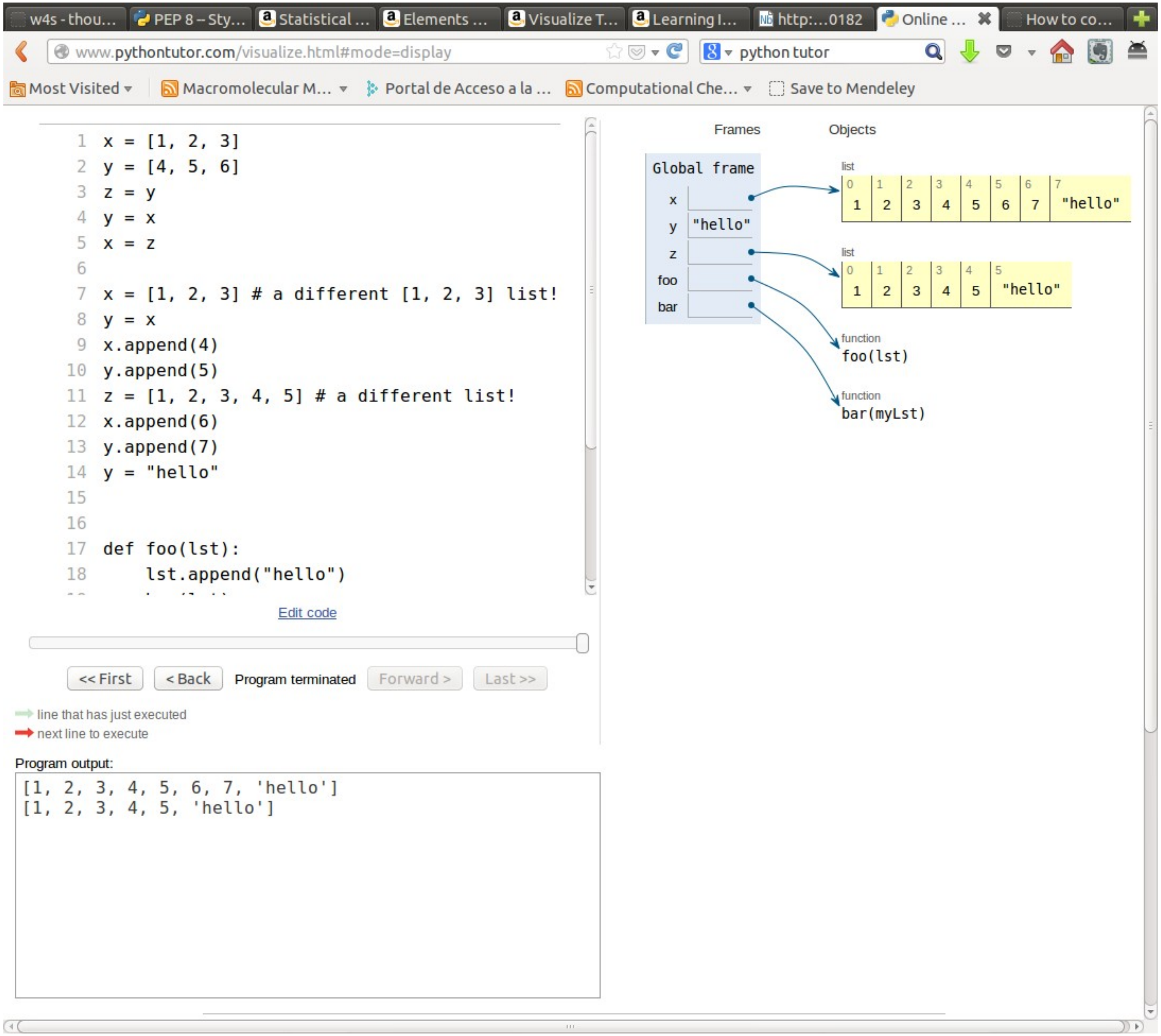
```
> c.conjugate #the method
> c.conjugate() #its call
```

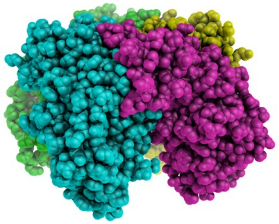
And we can apply functions to objects:

```
> abs(c)
```



Python flow with pythontutor





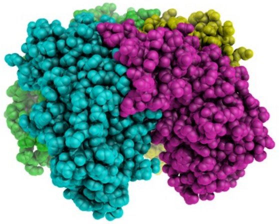
try... except

“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
```

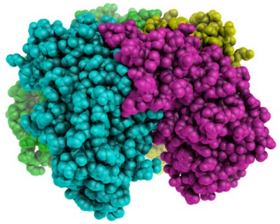
“It's easier to ask forgiveness than permission”:

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



Short notebook tutorial

But watch “I don’t like notebooks”:
http://ipython.org/ipython-doc/dev/interactive/tutorial.html



beyond python

TAB autocomplete:

- functions
- methods
- files
- …

`reload` command

cursor keys get history:

- even previous sessions!
- text + keys: previous match

?: intro to ipython

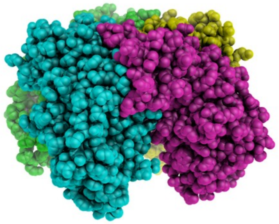
`%quickref`

`Ctrl-r` : previous commands

Without ipython:

`python3 -u script.py` enters interactive mode

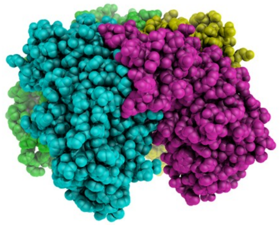
```
>>> import rlcompleter, readline
>>> readline.parse_and_bind('tab:complete')
```



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.
%cpaste, %paste: Paste & execute a pre-formatted code block from clipboard.
%history
%load_ext
%run
%pdb: Control the automatic calling of the pdb interactive debugger.
%pylab
%timeit
%pwd
%cd
%%bash
```

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



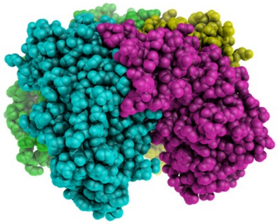
running scripts

`%run script.py`

`import script.py`

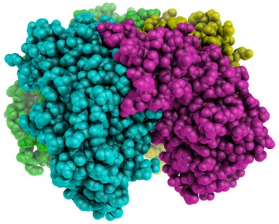
are not the same!

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

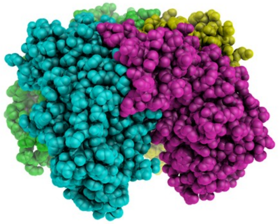


ipython notebook

- Nice presentation
- Allows parallel execution
- Combines text and code
- Executable or exportable to:
  - html
  - LaTeX
  - python
- Start with: `ipython3 notebook`
- Examples:
  https://github.com/jrjohansson/scientific-python-lectures

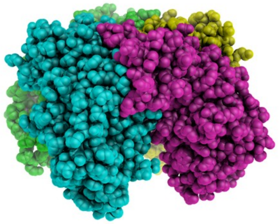


Files



Files

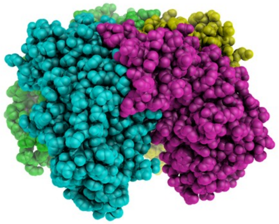
- Files can be text of binary files
- Files can be opened for read, write or append
  - `'r', 'w', 'a+'`
- `with open('name') as filein:`
  - Allows automatic file closure



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...



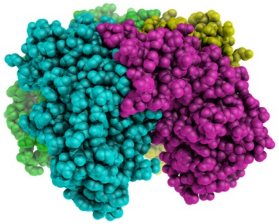
File parsing

- The basic:

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

- Common things:

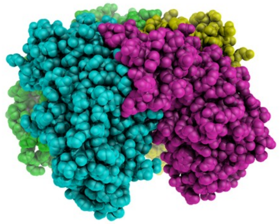
```
if 'optimized' in line: do something
line = line.split()
if line.upper().startswith('GEOM'): …
energy = float(line[2])
```



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)
```

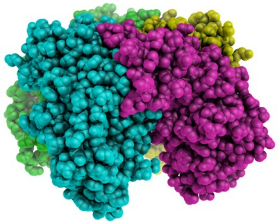
- But format is more general:

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

- List of unkown length:

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

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



Useful modules

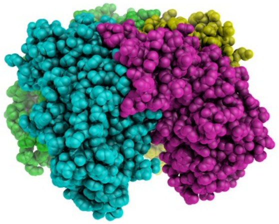
- Similar to `ls`:

```
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:

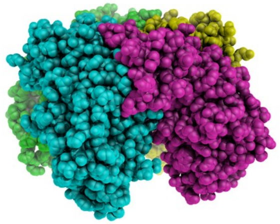
```
import xlrd
```

- Working with image files:

```
from PIL import Image
```

http://www.python-excel.org/
http://pillow.readthedocs.org/en/latest/

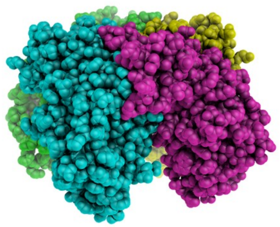
Numpy



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:

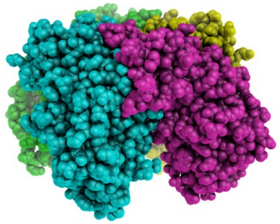
```
> a=[[1,2],[3,4]]

> 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:

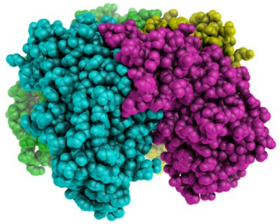
```
> a=np.array([[1,2], [3,4]])
> a.min(axis=0)
array([1, 2])
a.sum(axis=1)
array([3, 7])
```

- 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):

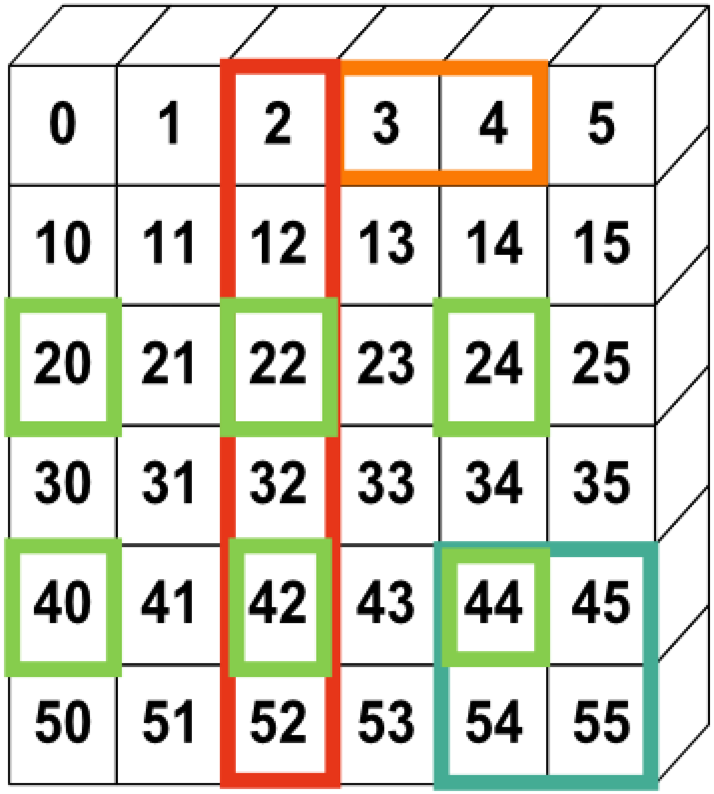
```
> a = np.arange(10,15)
> indices = (a**2 > 115) & (a < 14)
> a[indices]
array([11, 12, 13])
```

  - With lists:

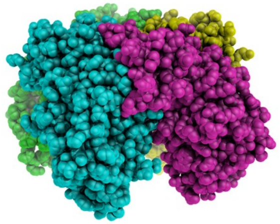
```
> a = np.arange(10,15)
> y=a[[4,4,1]]
> y
array([14, 14, 11])
> a[[4,4,1]] = [-2, -4, 5]
> a
array([10,  5, 12, 13, -4])
```

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]])
```

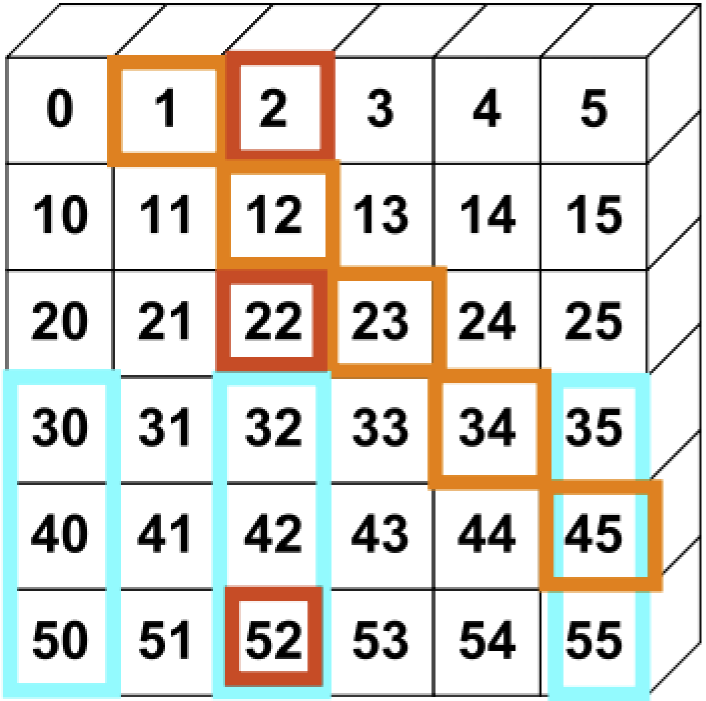


From: https://scipy-lectures.github.io/intro/numpy/array_object.html

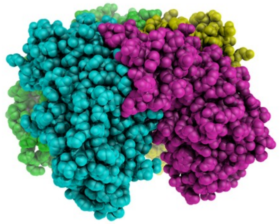


Accessing array elements

```
>>> 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])
```

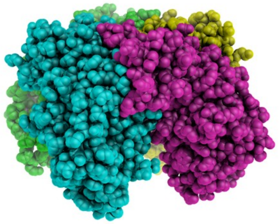
- np.choose
  - Powerful, but complex!
- np.nonzero

- Boolean arrays return copies

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

- Fancy indexing returns copies:

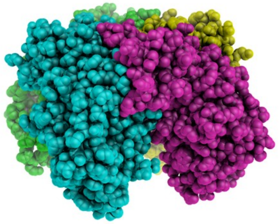
```
> a = np.arange(5)
> y=a[[2,3,4]]
> y *= -1
> a
array([0, 1, 2, 3, 4])
> y.flags.owndata
True
```



Broadcasting

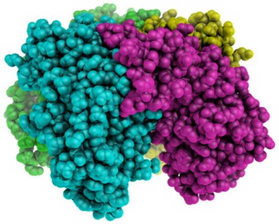
```
> a = 4.
> b = np.array([1,2,3])
> c = np.array([[1,2,3], [4,5,6]])
> b+a, c+a
(array([ 5.,  6.,  7.]), array([[  5.,   6.,   7.],
        [  8.,   9.,  10.]]))
> b+c
array([[2, 4, 6],
       [5, 7, 9]])
> c.dot(b)
> b.dot(c)
ValueError: objects are not aligned
> b[1:]*c
ValueError: operands could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

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



Broadcasting

- *The size of the trailing axes for both arrays in an operation must either be the same size or one of them must be one.*
- 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
  - they are equal, or
  - one of them is 1



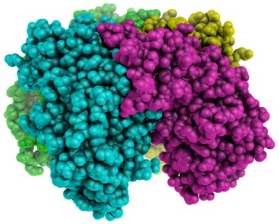
array functions and methods

- 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.iscontiguous()`
- Useful when interfacing with fortran / C:
- `a.is_c_array()`

  `a.is_f_array()`



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

- Save single arrays with:

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

- Save in text mode with:

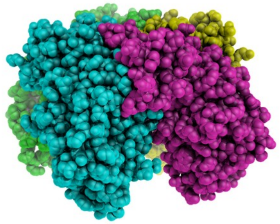
```
> np.savetxt('result_y', y)
```

- and multiple arrays with:

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

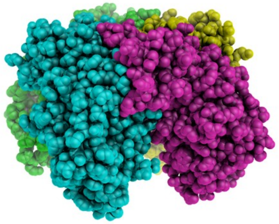
- Recover them with load:

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

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

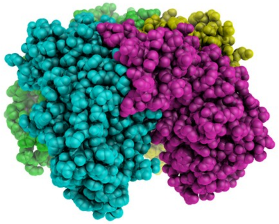


Beyond numpy

- Pandas:
  - Dataframes with named columns and rows
  - Similar to a spreadsheet
  - Great for data analysis
  - https://pandas.pydata.org/
- xarray
  - labeled multidimensional arrays: n-dimensional pandas-like dataframes
  - http://xarray.pydata.org/en/stable/



matplotlib

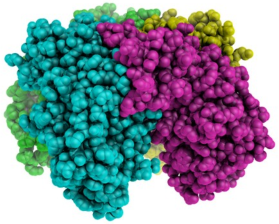


Matplotlib

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

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

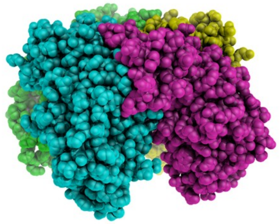
`import pylab` or similar is deprecated.



Matplotlib

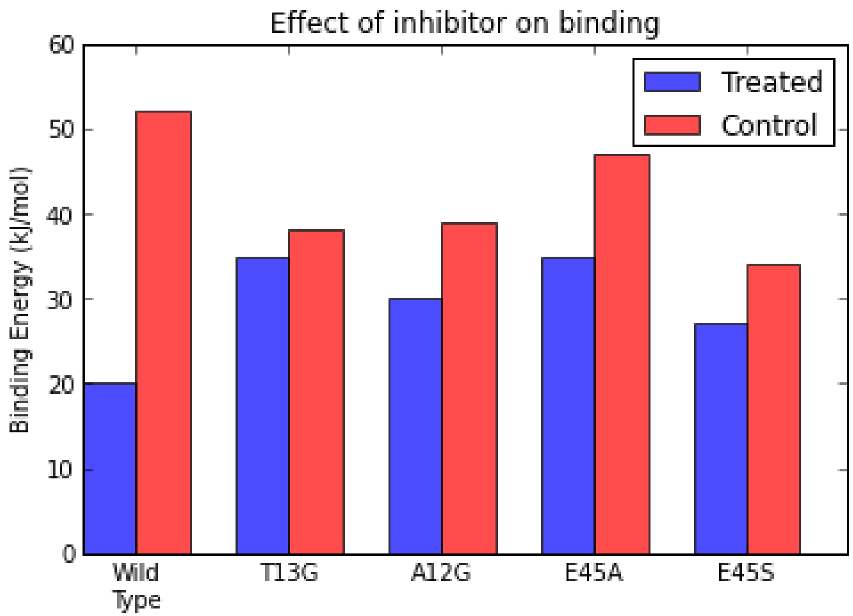
Simplest plots:

```
> 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. Similar to matshow.

> d = np.loadtxt('data.txt')

> plt.plot(d[:,0], d[:,1], 's') #just slightly longer than
gnuplot
```

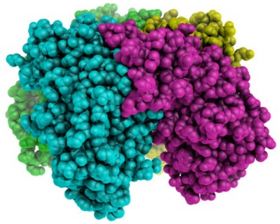


Matplotlib

Totally reproducible figures



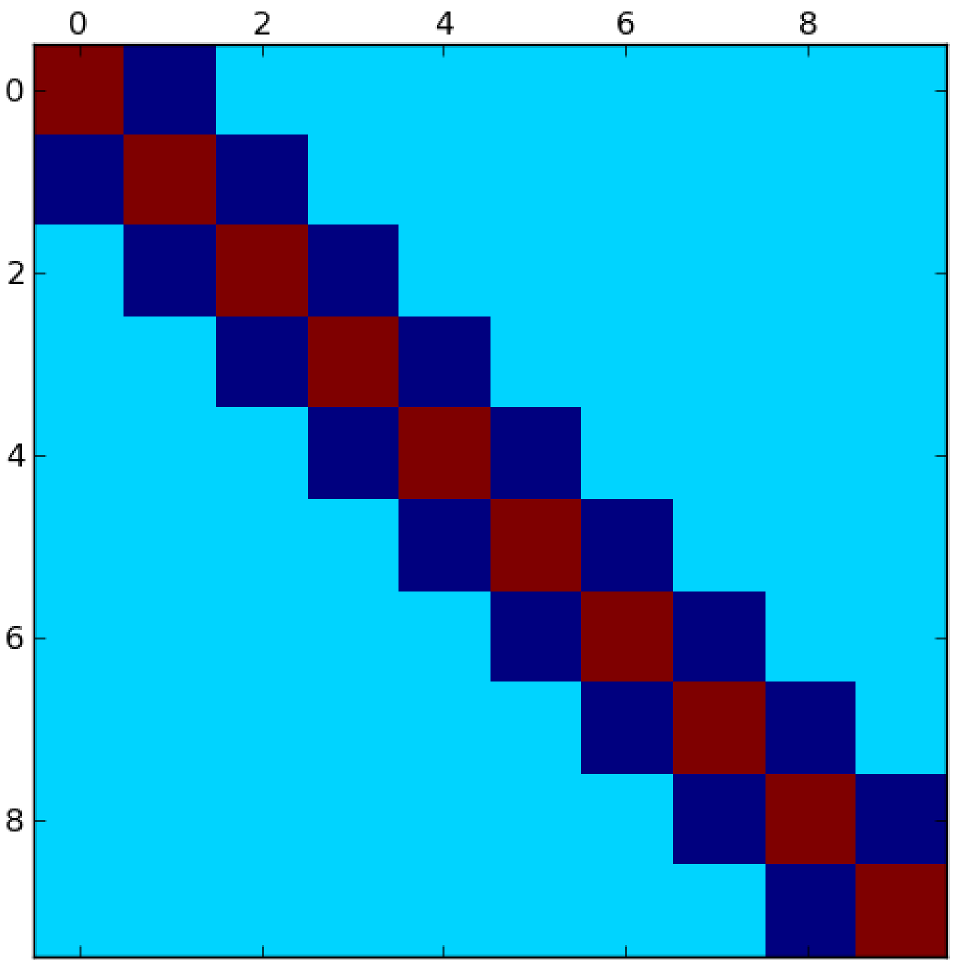

```
N = 5
treated = (20, 35, 30, 35, 27)
control = (52, 38, 39, 47, 34)
ind = np.arange(N)  # the x locations for the groups
width = 0.35       # the width of the bars

fig, ax = plt.subplots()
rects1 = ax.bar(ind, treated, width, color='b', alpha=0.7, label='Treated')
rects2 = ax.bar(ind+width, control, width, color='r', alpha=0.7, 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()
```



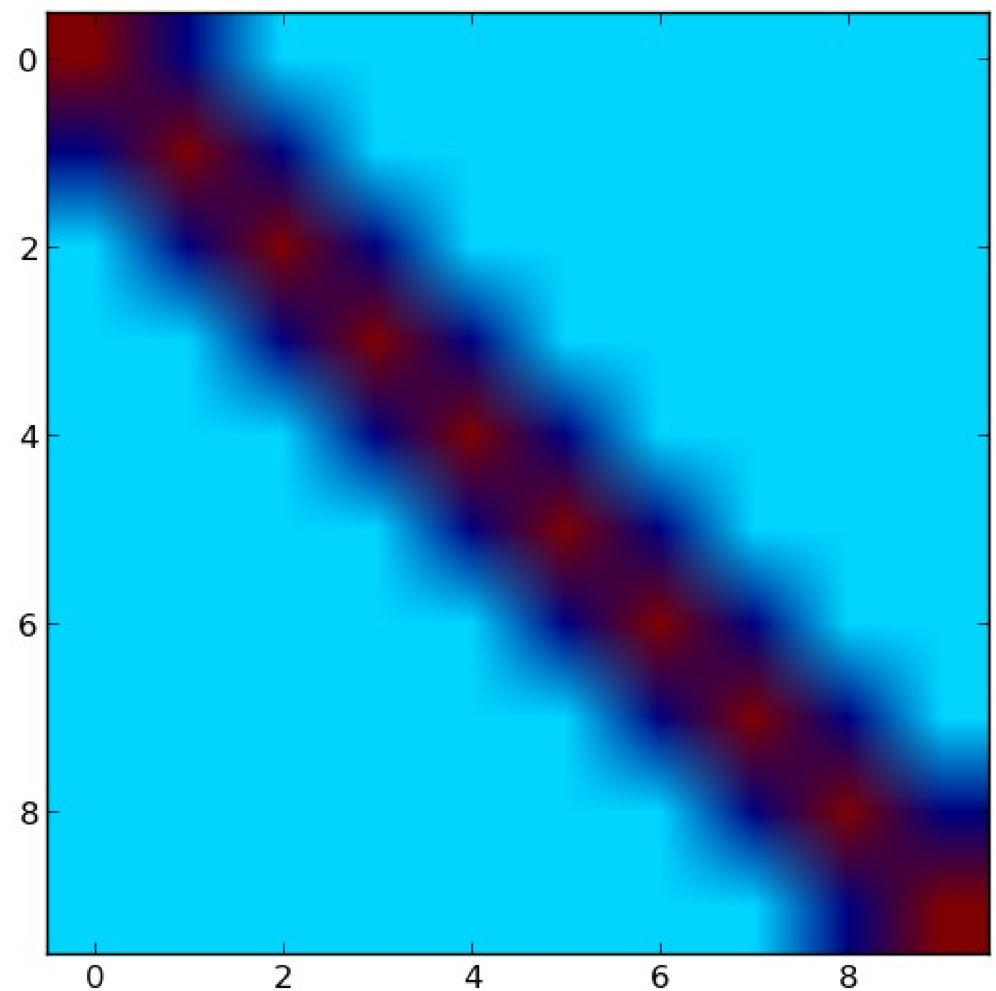
Plotting matrices

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

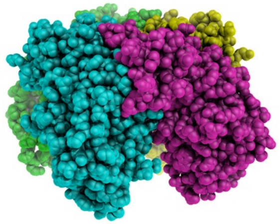
`matshow(m)`

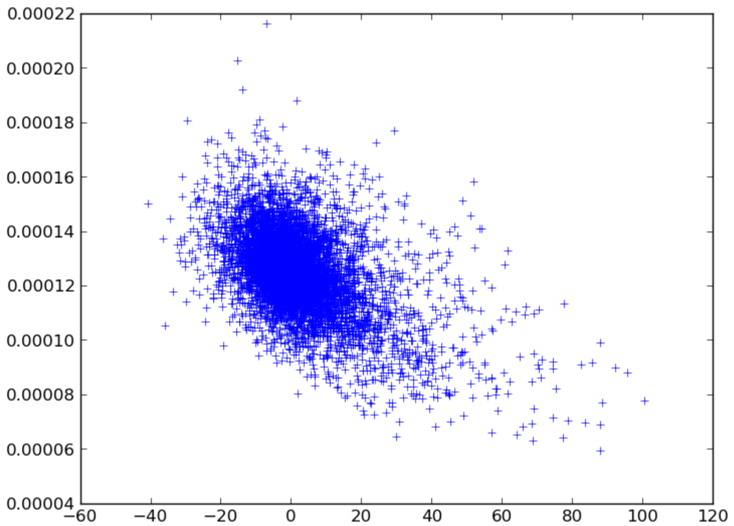


`imshow(m)`

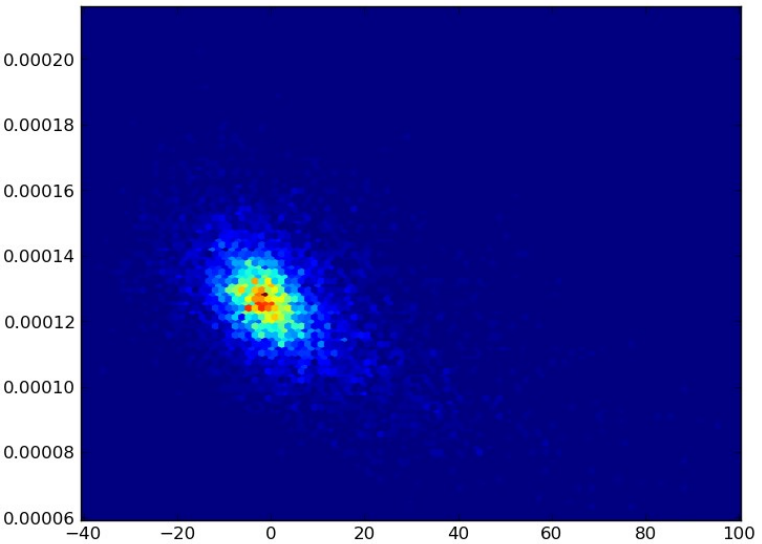


Plotting lots of points:hexbin

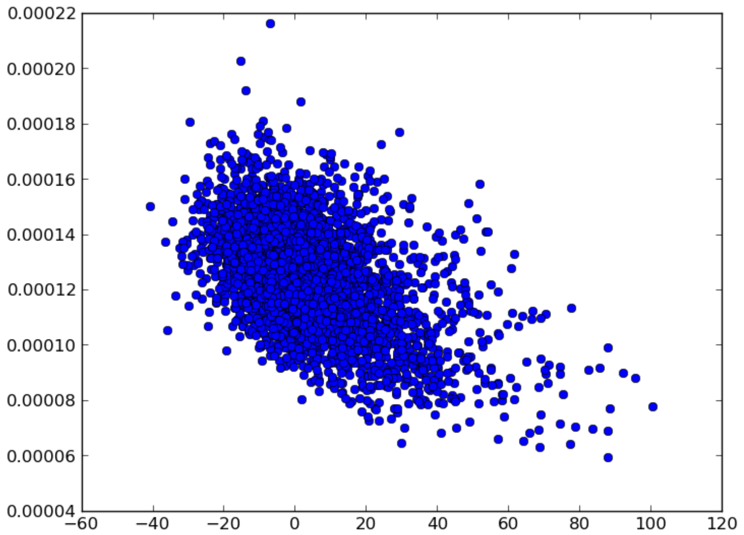




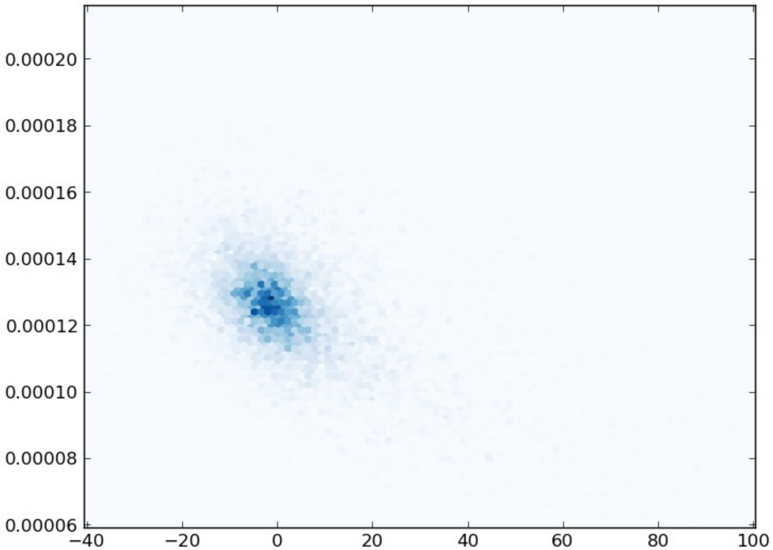
`plt.plot(x, y, '+')`



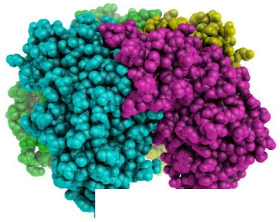
`plt.hexbin(x, y)`



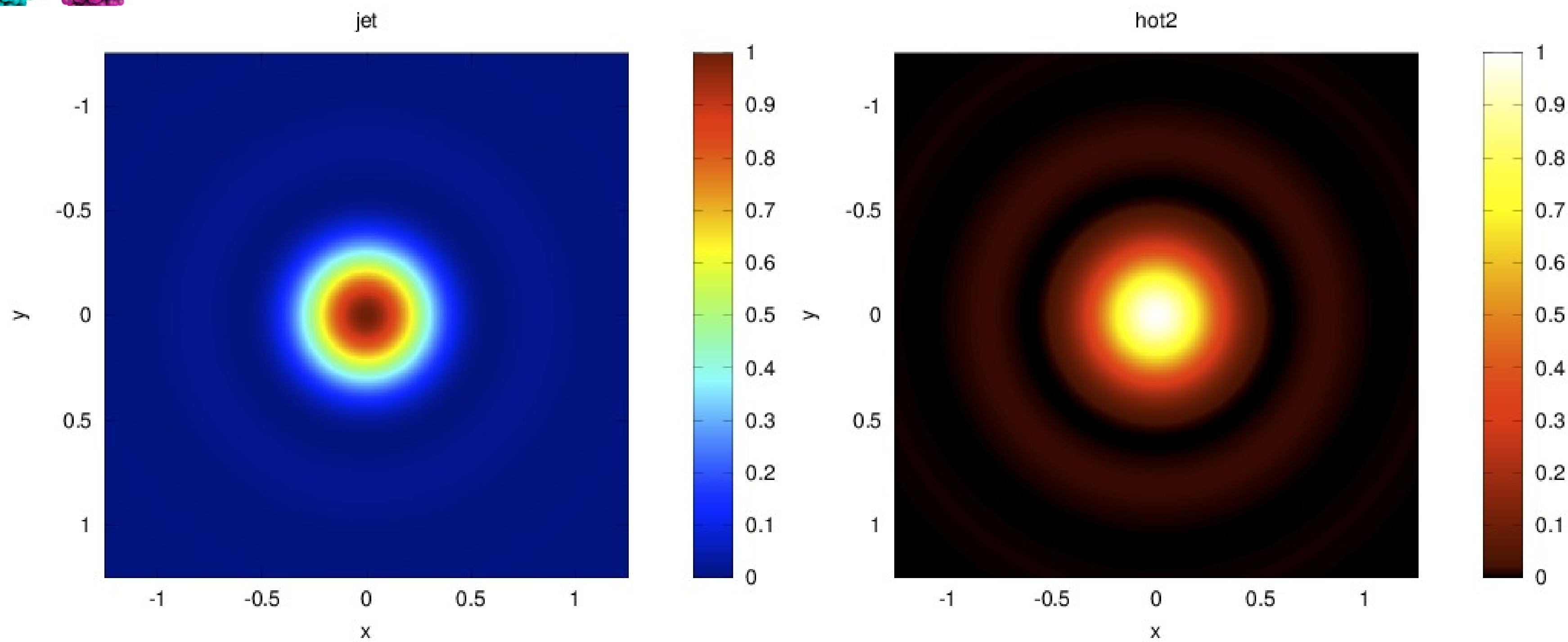
`plt.plot(x, y, 'o')`

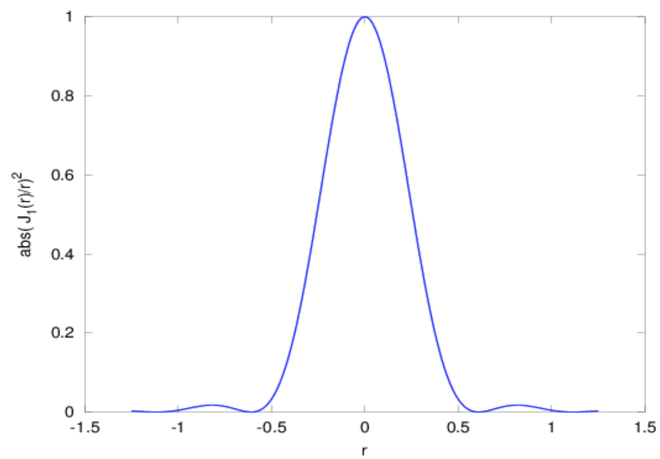


`plt.hexbin(x, y, cmap=pylab.cm.Blues)`

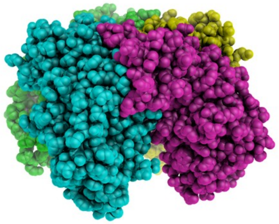


Jet is not a good colormap





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/

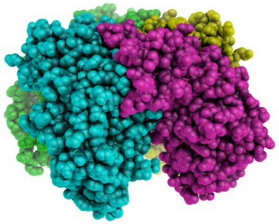


Matplotlib

- 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

Functions and modules



Exceptions and errors

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

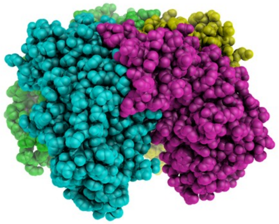
```
def safe_divide_1(x, y)


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



Functions

defined by def and a colon:

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

Remember indentation!

Automatic (and recommended) documentation:

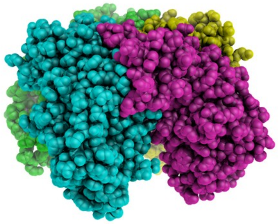
```
def add(x,y):
    ””” Returns the
     sum of 2 numbers”””
    return x+y
```

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
```



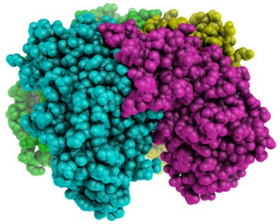
Functions II

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
```



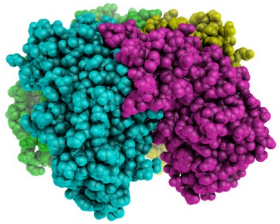
Functions III

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
> import copy
> a_copy=copy.deepcopy(a)
```



Functions IV

Functions can have default arguments :

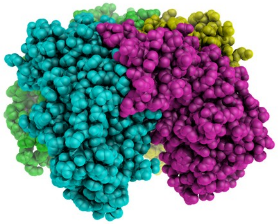
```
> def submit(job, priority=10,
  nprocs=1):
...   pass
> submit('job1.sh')
```

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)
```

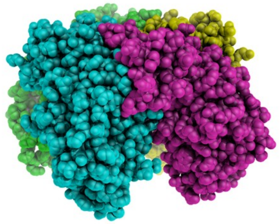
In `prod, args=(2,3,4)`

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:

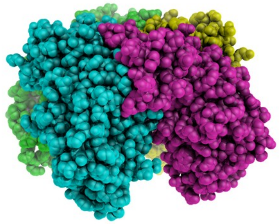
```
> vals = [1,2,3,4,5]
> print((4*'{:03d} ').format(*vals))
001 002 003 004
```



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

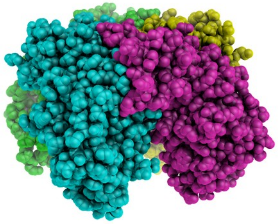
```
def rang_llista(n):
    result = []
    i = 0
    while i<n:
        result.append(i)
         i += 1
    return result


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!

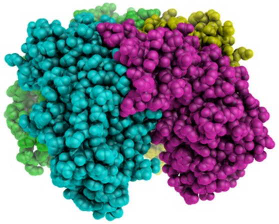
```
> import math

> m = math

> import math as m

> from math import cos, sin

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



Modules

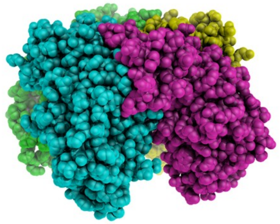
- Python checks if a module is already loaded.
  - The interpreter does not reload a module already imported
  - This can cause unexpected behaviour interactively
- Ipython has a more versatile module loading

```
%load_ext autoreload

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



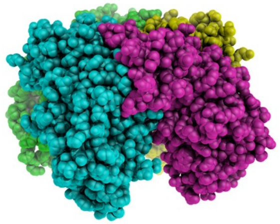
Some useful modules

- 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)
```

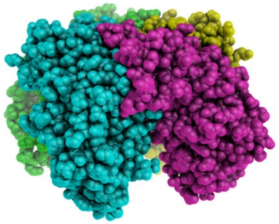


Modules: too many...

From the python documentation:

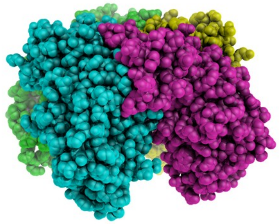
It is also possible to use a list as a queue, where the first element added is the first element retrieved (“first-in, first-out”); however, lists are not efficient for this purpose. While appends and pops from the end of list are fast, doing inserts or pops from the beginning of a list is slow (because all of the other elements have to be shifted by one).

To implement a queue, use `collections.deque` which was designed to have fast appends and pops from both ends.



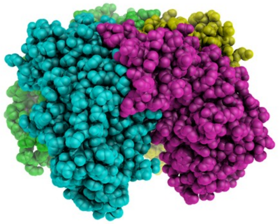
Modules: too many...

```
>>> 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

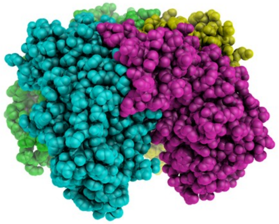
- Import reads from local directory and from the directories in `sys.path` (`import sys` first)
- Put your modules in a directory and add it to the environtment variable `$PYTHONPATH`.
- Python will add the directories in `$PYTHONPATH` to `sys.path`
- Document your modules and the functions therein.
- 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



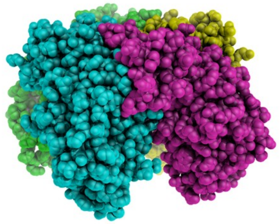
Installing external Modules

- Use conda distribution. Then `` `conda install module` ``
- Many come as part of the linux distributions (usually older versions)
  - ipython, numpy, biopython...
- For modules in the PyPI repository(most of them) https://pypi.python.org/pypi
  - Use `pip-3` or `pip3`
- Use:

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



Scipy



Linear algebra

- Support for LAPACK, BLAS and ATLAS
  - Can make Scipy compilation more involved

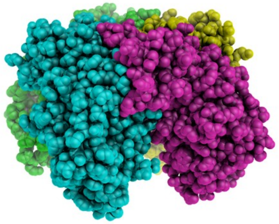
```
> A=matrix(random.rand(5,5))
> A.I
> linalg.det(A)
> linalg.eigvals(A)
> linalg.eig(A)
> linalg.svd(A)
> linalg.cholesky(A)
```

- Solving linear systems:
  - **A.x=b**

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

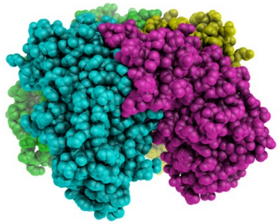
- LAPACK, BLAS wrappers

```
> from scipy.lib import lapack
> from scipy.lib import blas
blas.fblas.sdot?
```



Linear regression

- There are different ways to perform linear regression (still surprised?)
- See the notebook.



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
```

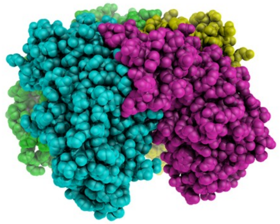
- Some look for global minima:

```
> 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



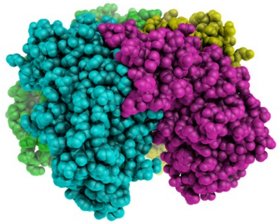
f2py

- Many things are fast with Numpy
- Iterative algorithms over **array values** are slow
- You can import Fortran functions and subroutines with f2py
- You could also call external fortran programs with

  `> subprocess.call(<program>, shell=True)`

  – but data exchange has to be through files (slower)

- f2py finds your fortran compiler. Works with gfortran, ifort,...
- f2py creates a module you can import in python
- As simple as:
- `$ f2py -c <file> -m <module>`
  – Tip: first compile it to check it works



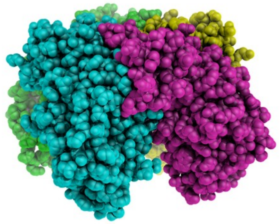
f2py II

```
module funcs
implicit none
contains
function f1(x,y)
  real,intent(in):: x,y
  real:: f1
  f1=x+y**2
end function f1

function f2(x,y)
  real,intent(in):: x,y
  real, dimension(3):: f2
  f2(1)=x+y**2
  f2(2)=sin(x*y)
  f2(3)=2*x-y
end function f2
end module
```

```
$ f2py -c test.f90 -m test
```

- go to ipython:

```
> import test
> test.funcs.f1(1,2)
5.0
> test.funcs.f2(1,2)
array([ 5., 0.90929741, 0.],
dtype=float32)
```



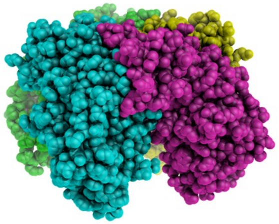
f2py III

Using ipython magicfunctions:

```
sudo pip3 install -U fortran-magic
```

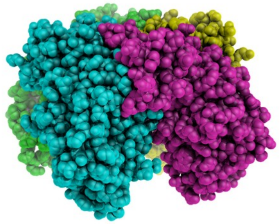
Useful for performing long array operations

```
In [5]: %load_ext fortranmagic

In [6]: %%fortran
        subroutine f1(x, y, z)
            real, intent(in) :: x,y
            real, intent(out) :: z

            z = sin(x+y)

        end subroutine f1

In [7]: f1(1.0, 2.1415)

Out[7]: 9.26574066397734e-05
```



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
  - Use `PyTable` to store (compressed) matrices on disk
  - Modify your algorithm to work with submatrices



Sympy: Symbolic math

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

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)`

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

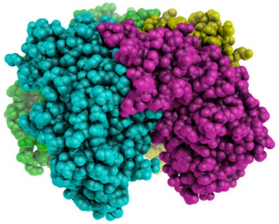
`(cos(x + y)).rewrite(sin, exp, cos, exp, tan, exp)`

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

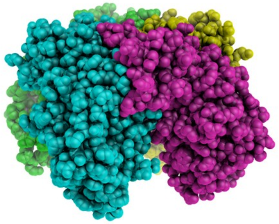
```
>>> integ = Integral(sin(x**2), x)
>>> integ
⌠
⎮    ⎛ 2⎞
⎮ sin⎝x ⎠ dx
⌡
>>> integ.doit()
```

$$\frac{3\cdot\sqrt{2}\cdot\sqrt{\pi}\cdot\mathrm{fresnels}\left(\frac{\sqrt{2}\cdot x}{\sqrt{\pi}}\right)\cdot\Gamma(3/4)}{8\cdot\Gamma(7/4)}$$

http://sympy.org/en/index.htm
l



Classes



Classes and objects

Most simple examples are quite stupid...

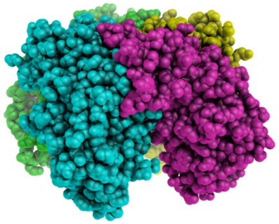
Everything is an object with python.

Objects have methods

- `sort` is a method of list objects
- They have to be called with ()
- Objects have attributes
- `c.real` is an attribute

Example: you have a set of sensors

- Sensors have position
- Sensors have a value
- Sensors can be reset



Classes and objects II

Sensor is an object

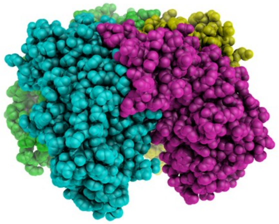
- Each object has methods (and attributes)

```
sensor[i].position
sensor[i].value
sensor[i].reset(0.0)
```

Fortran / C:

- Generate arrays for each property:

```
position[i]
value[i]
subroutine reset(i, val)
```



Classes and objects III

Better Fortran / C:

- use struct (in C)
- use type (in Fortran)

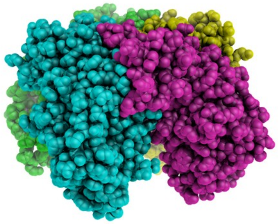
```
type sensor
       real, dimension(3) :: position
    real :: value
end type sensor
sensor[i]%position
sensor[i]%value
reset ?
```

But now you also have clocks:

- clocks have `values`
- clocks can be `reset`

What does `value` refer to?

What is the argument of `reset`?



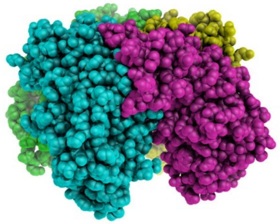
Objects and inheritance

New classes are defined as:

```
class Point:
  def __init__(self,x,y):
    self.x=x
    self.y=y
```

Classes allow the definition of elaborate personal objects

```
class Num_List(list):
  def __init__(self, lst=None)
    list.__init__(self,lst)
def square(self):
  for i in range(len(self)):
    self[i]=self[i]**2
> a=num_list([1,2,3])
> a.square()
> a
[1, 4, 9]
```



A periodic table of objects

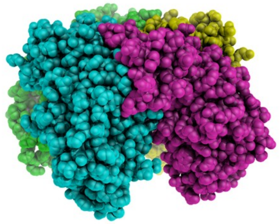
Each chemical element can be defined as a object.

Attributes such as:

- Name
- atomic symbol
- atomic weight
- Exercice: generate element objects and iterate over them to print:

```
The atomic number of hydrogen is 1
The atomic number of lithium is 3
The atomic number of helium is 2
```

- https://ramoncrehuet.wordpress.com/2014/11/12/a-periodic-table-of-objects/



A practical application

Line search in one dimension:

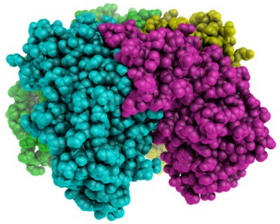
f(x) becomes f($\mathbf{x}_0$+s $\mathbf{d}$)

Implicitly depends on $\mathbf{x}_0$ and $\mathbf{d}$.

How can we use our line_search algorithm?

```
def f1(x):

    return x[0]**2+2*x[1]**2+4*\

          x[0]*x[1]-3*x[1]+2*x[1]


x0 = np.asarray([3,-1.])

r=cg(f1, x0)

print(r)
```

```
import numpy as np
import scipy.optimize as so
def cg(func, x0):
    "Conjugate Gradient"
    class Scalar:
        'A class to hold scalar functions'
        def __init__(self, func, x0, d):
            self.func = func
            self.x0 = x0
            self.d = d
        def value(self,s):
            'function evaluation'
            return self.func(x0+s*d)
    d = # from conjugate gradient algorithm
    f=Scalar(func, x0, d)
    result = so.minimize_scalar(f.value)
    return result
```



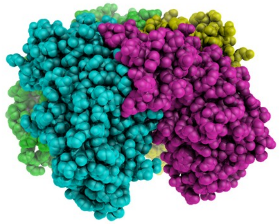
A practical application

As usual, there are already solutions for this common task:

```
scipy.optimize.fmin(func, x0, args=())
Parameters
----------
func : callable func(x,*args)
    The objective function to be minimized.
x0 : ndarray
    Initial guess.
args : tuple, optional
    Extra arguments passed to func, i.e. ``f(x,*args)``.
```

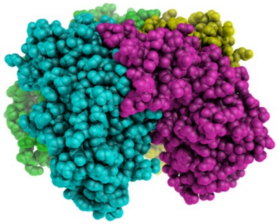
Or use (needs gradient):

```
scipy.optimize.line_search
```

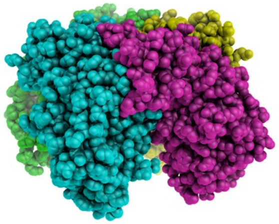


Add ons: itertools

```
> 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')]
```



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

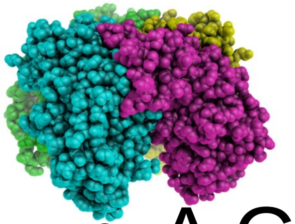
Comparison of codes in different languages:

http://rosetacode.org

http://www.codecodex.com

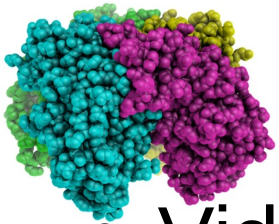
Python package index: where to find modules

http://pypi.python.org/pypi



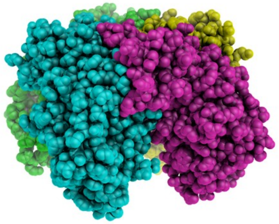
Resources

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



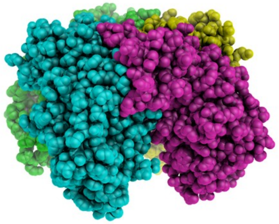
Resources

- Videos from the Scipy conference:
  - https://www.youtube.com/watch?v=V0D2mhVt7NE&t=0s&list=PLYx7XA2nY5Gd-tNhm79CNMe_qvi35PgUR&index=15
  - https://www.youtube.com/watch?v=Gzun8PpyBCo&t=0s&list=PLYx7XA2nY5Gd-tNhm79CNMe_qvi35PgUR&index=93
  -



Resources: Books

- Langtangen, *A Premier on Scientific Programming with Python*.
  - Good for learning scientific programming. Starts from scratch. Unfortunately does not use ipython nor numpy. Python 2.
- Langtangen, *Python Scripting for Computational Science.*
  - Good book for programmers. Advanced level. Explains how to interface with C and Fortran and how to optimize code. Python 2.
- Rossant, C, *Learning Ipython for Interactive Computing and Data Visualization.*
  - Basic level. Covers several subjects, includng matplotlib and parallelism.

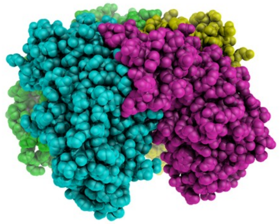


Resources: Teaching

- On teaching programming with Python 3

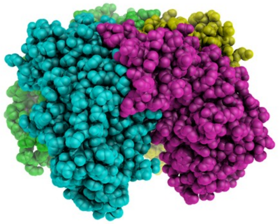
  http://www.comp.leeds.ac.uk/nde/papers/teachpy3.html
- Online Syntax Highlighting

  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
- iven 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/



Software in python

- Molecular visualization:
  - VMD: http://www.ks.uiuc.edu/Research/vmd/
  - pymol: http://www.pymol.org/
- Molecular Dynamics
  - Espresso http://espressomd.org/wordpress/
  - HOOMD-Blue http://glotzerlab.engin.umich.edu/hoomd-blue/
  - QM/MM with pDynamo: http://www.pdynamo.org
- QM calculation with
  - pyQuante: http://pyquante.sourceforge.net/
  - NWChem: http://www.nwchem-sw.org/index.php/Python
- Protein structure with pyRosetta: http://pyrosetta.org/
- Bioinformatics with BioPython: http://biopython.org/