An introduction to the Zoo of libraries

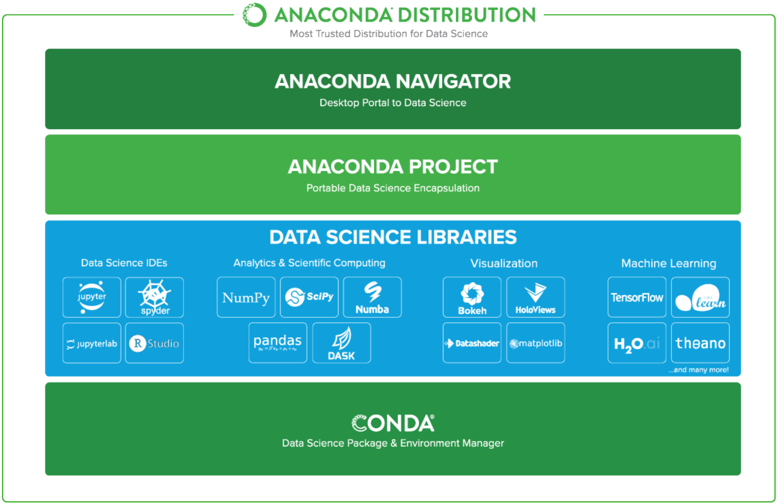
Cecilia Jarne

Twitter: @ceciliajarne

What is this talk about?

- A brief review on the use of some python scientific libraries.
- What is Machine Learning?: Introduction and basic libraries.
- Good programming practices: How to code and be understood.

What is Anaconda for?



Python basics for data analysis:

Python For Data Science *Cheat Sheet*

Python Basics

Learn More Python for Data Science Interactively at www.datacamp.com

Variables and Data Types

Variable Assignment

```
>>> x=5
>>> x
5
```

Calculations With Variables

`>>> x+2` → `7`	Sum of two variables
`>>> x-2` → `3`	Subtraction of two variables
`>>> x*2` → `10`	Multiplication of two variables
`>>> x**2` → `25`	Exponentiation of a variable
`>>> x%2` → `1`	Remainder of a variable
`>>> x/float(2)` → `2.5`	Division of a variable

Types and Type Conversion

`str()`	`'5', '3.45', 'True'`	Variables to strings
`int()`	`5, 3, 1`	Variables to integers
`float()`	`5.0, 1.0`	Variables to floats
`bool()`	`True, True, True`	Variables to booleans

Asking For Help

```
>>> help(str)
```

Strings

```
>>> my_string = 'thisStringIsAwesome'
>>> my_string
'thisStringIsAwesome'
```

String Operations

```
>>> my_string * 2
 'thisStringIsAwesomethisStringIsAwesome'
>>> my_string + 'Innit'
 'thisStringIsAwesomeInnit'
>>> 'm' in my_string
 True
```

String Operations — Index starts at 0

```
>>> my_string[3]
>>> my_string[4:9]
```

String Methods

`>>> my_string.upper()`	String to uppercase
`>>> my_string.lower()`	String to lowercase
`>>> my_string.count('w')`	Count String elements
`>>> my_string.replace('e', 'i')`	Replace String elements
`>>> my_string.strip()`	Strip whitespaces

Lists — Also see NumPy Arrays

```
>>> a = 'is'
>>> b = 'nice'
>>> my_list = ['my', 'list', a, b]
>>> my_list2 = [[4,5,6,7], [3,4,5,6]]
```

Selecting List Elements — Index starts at 0

Subset	
`>>> my_list[1]`	Select item at index 1
`>>> my_list[-3]`	Select 3rd last item
Slice	
`>>> my_list[1:3]`	Select items at index 1 and 2
`>>> my_list[1:]`	Select items after index 0
`>>> my_list[:3]`	Select items before index 3
`>>> my_list[:]`	Copy my_list
Subset Lists of Lists	
`>>> my_list2[1][0]`	my_list[list][itemOfList]
`>>> my_list2[1][:2]`	

List Operations

```
>>> my_list + my_list
['my', 'list', 'is', 'nice', 'my', 'list', 'is', 'nice']
>>> my_list * 2
['my', 'list', 'is', 'nice', 'my', 'list', 'is', 'nice']
>>> my_list2 > 4
True
```

List Methods

`>>> my_list.index(a)`	Get the index of an item
`>>> my_list.count(a)`	Count an item
`>>> my_list.append('!')`	Append an item at a time
`>>> my_list.remove('!')`	Remove an item
`>>> del(my_list[0:1])`	Remove an item
`>>> my_list.reverse()`	Reverse the list
`>>> my_list.extend('!')`	Append an item
`>>> my_list.pop(-1)`	Remove an item
`>>> my_list.insert(0,'!')`	Insert an item
`>>> my_list.sort()`	Sort the list

Libraries

Import libraries

```
>>> import numpy
>>> import numpy as np
```

Selective import

```
>>> from math import pi
```

pandas — Data analysis

learn — Machine learning

NumPy — Scientific computing

matplotlib — 2D plotting

Install Python

ANACONDA — Leading open data science platform powered by Python

spyder — Free IDE that is included with Anaconda

jupyter — Create and share documents with live code, visualizations, text, ...

Numpy Arrays — Also see Lists

```
>>> my_list = [1, 2, 3, 4]
>>> my_array = np.array(my_list)
>>> my_2darray = np.array([[1,2,3],[4,5,6]])
```

Selecting Numpy Array Elements — Index starts at 0

Subset	
`>>> my_array[1]` → `2`	Select item at index 1
Slice	
`>>> my_array[0:2]` → `array([1, 2])`	Select items at index 0 and 1
Subset 2D Numpy arrays	
`>>> my_2darray[:,0]` → `array([1, 4])`	my_2darray[rows, columns]

Numpy Array Operations

```
>>> my_array > 3
 array([False, False, False,  True], dtype=bool)
>>> my_array * 2
 array([2, 4, 6, 8])
>>> my_array + np.array([5, 6, 7, 8])
 array([6, 8, 10, 12])
```

Numpy Array Functions

`>>> my_array.shape`	Get the dimensions of the array
`>>> np.append(other_array)`	Append items to an array
`>>> np.insert(my_array, 1, 5)`	Insert items in an array
`>>> np.delete(my_array,[1])`	Delete items in an array
`>>> np.mean(my_array)`	Mean of the array
`>>> np.median(my_array)`	Median of the array
`>>> my_array.corrcoef()`	Correlation coefficient
`>>> np.std(my_array)`	Standard deviation

DataCamp
Learn Python for Data Science Interactively

WTPC
Workshop en técnicas de programación científica

To start (very quick!):

- Install the Python interpreter on your computer.
  (Linux distributions also frequently include Python and it is upgraded.)
- Call interpreter:

```
python
```

- It has an interactive mode to introduce instructions one by one and see the result.

```
>>> 1 + 1
2
>>> a = range(10)
>>> print a
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
```

Basic ideas:

- In a statically typed language, every variable name is bound:
  - to a type (at compile time, by means of a data declaration).
  - to an object.
- In a dynamically typed language, every variable name is (unless it is null) is bound only to an object.

```
x = 1
x = "text" # dynamic typing :)
```

Data types:

A list of the most common data types:

Class	Description	Immutable?
bool	Boolean value	✓
int	integer (arbitrary magnitude)	✓
float	floating-point number	✓
list	mutable sequence of objects	
tuple	immutable sequence of objects	✓
str	character string	✓
set	unordered set of distinct objects	
frozenset	immutable form of set class	✓
dict	associative mapping (aka dictionary)	

https://docs.python.org/2/library/types.html

Data types:

A list of the most common data types:

Class	Description	Immutable?
bool	Boolean value	✓
int	integer (arbitrary magnitude)	✓
float	floating-point number	✓
list	mutable sequence of objects	
tuple	immutable sequence of objects	✓
str	character string	✓
set	unordered set of distinct objects	
frozenset	immutable form of set class	✓
dict	associative mapping (aka dictionary)	

https://docs.python.org/2/library/types.html

Differences in the Python Syntax

:

In C or C++: use of ; at the end of statements

```
if(a>b)
    foo();
    bar();
baz();
```

In python: indentation level of your statements is significant! Last statement is executed out of the conditional:

```
if(a>b):
    foo()
    bar()
baz()
```

Python Syntax: control flow instructions

Control flow

```
for i in list:
    baz(i)
```

```
if a>b:
    foo()
elif b!=c:
    bar()
else:
    baz()
```

```
while a>b:
    foo()
    bar()
```

```
pass
```

```
break
continue
```

Python Syntax

Function definition:

```
def function(x,y,z):
    x=3*y
    return x+y-z
```

Also functions can be pass as values:

```
def func(x, p1,p2):
    return p1*(x)+p2
```

Python Boolean and Math operators

Math operators:

Name	Function	symbol
+ Addition	Adds values on either side of the operator.	a + b = 30
- Subtraction	Subtracts right hand operand from left hand operand.	a − b = -10
* Multiplication	Multiplies values on either side of the operator	a * b = 200
/ Division	Divides left hand operand by right hand operand	b / a = 2
% Modulus	Divides left hand operand by right hand operand and returns remainder	b % a = 0
** Exponent	Performs exponential (power) calculation on operators	a**b =10 to the power 20

Boolean:

Operation	Result
x or y	if x is false, then y, else x
x and y	if x is false, then x, else y
not x	if x is false, then True, else False

It is possible to use ’ or “ on string definition:

```
a= "I am Cecilia"
b= 'I am from Argentina'
```

To print on screen we use the print function:

```
print ('---All rigth!!!----')
```

Lists

- Very useful structure on python.
- Index and slice access.

```
a[0] a[1] a[2:5] a[2:10:2]
```

- Comprehension list:

```
[x**2 for x in range(1,11)]
```

List Methods

```
list.append(obj)
```

Appends object obj to list.

```
list.count(obj)
```

Returns count of how many times obj occurs in list.

```
list.extend(seq)
```

Appends the contents of seq to list.

```
list.index(obj)
```

Returns the lowest index in list that obj appears.

```
list.insert(index, obj)
```

Inserts object obj into list at offset index.

More List Methods

```
list.pop(obj=list[-1])
```

Removes and returns last object or obj from list.

```
list.remove(obj)
```

Removes object obj from list.

```
list.reverse()
```

Reverses objects of list in place.

Three main libraries:

- NumPy: http://www.numpy.org/
- SciPy: http://www.scipy.org/
- MatPlotLib: http://matplotlib.org/

All are open source!

Main reason to use python in scientific programing

- NumPy provides functionality to create, delete, manage and operate on large arrays of type "raw" data (like Fortran and C/C++ arrays).
- SciPy extends NumPy with a collection of useful algorithms like minimization, Fourier transforms, regression and many other applied mathematical techniques.
- Both packages are add-on packages (not part of the Python standard library) containing Python code and compiled with (fftpack, BLAS).
- MatPlotLib is a nice library to plot (but there are others as Seaborn or Bokeh).

Numpy Basics:

Python For Data Science *Cheat Sheet*

NumPy Basics

Learn Python for Data Science **Interactively** at www.DataCamp.com

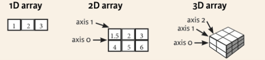
NumPy

The **NumPy** library is the core library for scientific computing in Python. It provides a high-performance multidimensional array object, and tools for working with these arrays.

Use the following import convention:

```
>>> import numpy as np
```

NumPy Arrays



Creating Arrays

```
>>> a = np.array([1,2,3])
>>> b = np.array([(1.5,2,3), (4,5,6)], dtype = float)
>>> c = np.array([[(1.5,2,3), (4,5,6)], [(3,2,1), (4,5,6)]],
                  dtype = float)
```

Initial Placeholders

`>>> np.zeros((3,4))`	Create an array of zeros
`>>> np.ones((2,3,4),dtype=np.int16)`	Create an array of ones
`>>> d = np.arange(10,25,5)`	Create an array of evenly spaced values (step value)
`>>> np.linspace(0,2,9)`	Create an array of evenly spaced values (number of samples)
`>>> e = np.full((2,2),7)`	Create a constant array
`>>> f = np.eye(2)`	Create a 2X2 identity matrix
`>>> np.random.random((2,2))`	Create an array with random values
`>>> np.empty((3,2))`	Create an empty array

I/O

Saving & Loading On Disk

```
>>> np.save('my_array', a)
>>> np.savez('array.npz', a, b)
>>> np.load('my_array.npy')
```

Saving & Loading Text Files

```
>>> np.loadtxt("myfile.txt")
>>> np.genfromtxt("my_file.csv", delimiter=',')
>>> np.savetxt("myarray.txt", a, delimiter=" ")
```

Data Types

`>>> np.int64`	Signed 64-bit integer types
`>>> np.float32`	Standard double-precision floating point
`>>> np.complex`	Complex numbers represented by 128 floats
`>>> np.bool`	Boolean type storing `TRUE` and `FALSE` values
`>>> np.object`	Python object type
`>>> np.string_`	Fixed-length string type
`>>> np.unicode_`	Fixed-length unicode type

Inspecting Your Array

`>>> a.shape`	Array dimensions
`>>> len(a)`	Length of array
`>>> b.ndim`	Number of array dimensions
`>>> e.size`	Number of array elements
`>>> b.dtype`	Data type of array elements
`>>> b.dtype.name`	Name of data type
`>>> b.astype(int)`	Convert an array to a different type

Asking For Help

```
>>> np.info(np.ndarray.dtype)
```

Array Mathematics

Arithmetic Operations

`>>> g = a - b`
`array([[-0.5, 0. , 0. ],`
`[-3. , -3. , -3. ]])`
	Subtraction

`>>> np.subtract(a,b)`	Subtraction

`>>> b + a`
`array([[ 2.5, 4. , 6. ],`
`[ 5. , 7. , 9. ]])`
	Addition

`>>> np.add(b,a)`	Addition

`>>> a / b`
`array([[ 0.66666667, 1. , 1. ],`
`[ 0.25 , 0.4 , 0.5 ]])`
	Division

`>>> np.divide(a,b)`	Division

`>>> a * b`
`array([[ 1.5, 4. , 9. ],`
`[ 4. , 10. , 18. ]])`
	Multiplication

`>>> np.multiply(a,b)`	Multiplication
`>>> np.exp(b)`	Exponentiation
`>>> np.sqrt(b)`	Square root
`>>> np.sin(a)`	Print sines of an array
`>>> np.cos(b)`	Element-wise cosine
`>>> np.log(a)`	Element-wise natural logarithm

`>>> e.dot(f)`
`array([[ 7., 7.],`
`[ 7., 7.]])`
	Dot product

Comparison

`>>> a == b`
`array([[False, True, True],`
`[False, False, False]], dtype=bool)`
	Element-wise comparison

`>>> a < 2` `array([True, False, False], dtype=bool)`	Element-wise comparison
`>>> np.array_equal(a, b)`	Array-wise comparison

Aggregate Functions

`>>> a.sum()`	Array-wise sum
`>>> a.min()`	Array-wise minimum value
`>>> b.max(axis=0)`	Maximum value of an array row
`>>> b.cumsum(axis=1)`	Cumulative sum of the elements
`>>> a.mean()`	Mean
`>>> b.median()`	Median
`>>> a.corrcoef()`	Correlation coefficient
`>>> np.std(b)`	Standard deviation

Copying Arrays

`>>> h = a.view()`	Create a view of the array with the same data
`>>> np.copy(a)`	Create a copy of the array
`>>> h = a.copy()`	Create a deep copy of the array

Sorting Arrays

`>>> a.sort()`	Sort an array
`>>> c.sort(axis=0)`	Sort the elements of an array's axis

Subsetting, Slicing, Indexing

Also see Lists

Subsetting

`>>> a[2]` `3`	Select the element at the 2nd index
`>>> b[1,2]` `6.0`	Select the element at row 0 column 2 (equivalent to `b[1][2]`)

Slicing

`>>> a[0:2]` `array([1, 2])`	Select items at index 0 and 1
`>>> b[0:2,1]` `array([ 2., 5.])`	Select items at rows 0 and 1 in column 1
`>>> b[:1]` `array([[1.5, 2., 3.]])`	Select all items at row 0 (equivalent to `b[0:1, :]`)

`>>> c[1,...]`
`array([[[ 3., 2., 1.],`
`[ 4., 5., 6.]]])`
	Same as `[1,:,:]`

`>>> a[ : :-1]` `array([3, 2, 1])`	Reversed array `a`

Boolean Indexing

`>>> a[a<2]` `array([1])`	Select elements from `a` less than 2

Fancy Indexing

`>>> b[[1, 0, 1, 0],[0, 1, 2, 0]]` `array([ 4. , 2. , 6. , 1.5])`	Select elements `(1,0)`,`(0,1)`,`(1,2)` and `(0,0)`

`>>> b[[1, 0, 1, 0]][:,[0,1,2,0]]`
`array([[ 4. ,5. , 6. , 4. ],`
`[ 1.5, 2. , 3. , 1.5],`
`[ 4. , 5. , 6. , 4. ],`
`[ 1.5, 2. , 3. , 1.5]])`
	Select a subset of the matrix's rows and columns

Array Manipulation

Transposing Array

`>>> i = np.transpose(b)`	Permute array dimensions
`>>> i.T`	Permute array dimensions

Changing Array Shape

`>>> b.ravel()`	Flatten the array
`>>> g.reshape(3,-2)`	Reshape, but don't change data

Adding/Removing Elements

`>>> h.resize((2,6))`	Return a new array with shape (2,6)
`>>> np.append(h,g)`	Append items to an array
`>>> np.insert(a, 1, 5)`	Insert items in an array
`>>> np.delete(a,[1])`	Delete items from an array

Combining Arrays

`>>> np.concatenate((a,d),axis=0)` `array([ 1, 2, 3, 10, 15, 20])`	Concatenate arrays

`>>> np.vstack((a,b))`
`array([[ 1. , 2. , 3. ],`
`[ 1.5, 2. , 3. ],`
`[ 4. , 5. , 6. ]])`
	Stack arrays vertically (row-wise)

`>>> np.r_[e,f]`	Stack arrays vertically (row-wise)

`>>> np.hstack((e,f))`
`array([[ 7., 7., 1., 0.],`
`[ 7., 7., 0., 1.]])`
	Stack arrays horizontally (column-wise)

`>>> np.column_stack((a,d))`
`array([[ 1, 10],`
`[ 2, 15],`
`[ 3, 20]])`
	Create stacked column-wise arrays

`>>> np.c_[a,d]`	Create stacked column-wise arrays

Splitting Arrays

`>>> np.hsplit(a,3)` `[array([1]),array([2]),array([3])]`	Split the array horizontally at the 3rd index

`>>> np.vsplit(c,2)`
`[array([[[ 1.5, 2. , 1. ],`
`[ 4. , 5. , 6. ]]]),`
`array([[[ 3., 2., 3.],`
`[ 4., 5., 6.]]])]`
	Split the array vertically at the 2nd index

DataCamp

Learn Python for Data Science Interactively

WTPC
Workshop en técnicas de programación científica

NumPy

NumPy is the fundamental package for scientific computing in Python.

- A powerful N-dimensional array object.
- Sophisticated (broadcasting) functions.
- Tools for integrating C/C++ and Fortran code.
- Useful linear algebra, Fourier transform, and random number capabilities.
- Operations on matrices and vectors in NumPy are very efficient because they are linked to compiled in BLAS/LAPACK code.

Things from Numpy

NumPy functionality:

- Polynomial mathematics.
- Statistical computations.
- Pseudo random number generators.
- Discrete Fourier transforms.
- Size / shape / type testing of arrays.

Fundamental things from Numpy: np.array

There are 5 general mechanisms for creating arrays:

- Conversion from other Python structures (e.g., lists, tuples).
- Intrinsic numpy array creation objects (e.g., arange, ones, zeros, etc.)
- Reading arrays from disk, either from standard or custom formats.
- Creating arrays from raw bytes through the use of strings or buffers.
- Use of special library functions (e.g., random).

Python For Data Science *Cheat Sheet*

SciPy - Linear Algebra

Learn More Python for Data Science Interactively at www.datacamp.com

SciPy

The **SciPy** library is one of the core packages for scientific computing that provides mathematical algorithms and convenience functions built on the NumPy extension of Python.

SciPy

Interacting With NumPy

Also see NumPy

```
>>> import numpy as np
>>> a = np.array([1,2,3])
>>> b = np.array([(1+5j,2j,3j), (4j,5j,6j)])
>>> c = np.array([[(1.5,2,3), (4,5,6)], [(3,2,1), (4,5,6)]])
```

Index Tricks

`>>> np.mgrid[0:5,0:5]`	Create a dense meshgrid
`>>> np.ogrid[0:2,0:2]`	Create an open meshgrid
`>>> np.r_[3,[0]*5,-1:1:10j]`	Stack arrays vertically (row-wise)
`>>> np.c_[b,c]`	Create stacked column-wise arrays

Shape Manipulation

`>>> np.transpose(b)`	Permute array dimensions
`>>> b.flatten()`	Flatten the array
`>>> np.hstack((b,c))`	Stack arrays horizontally (column-wise)
`>>> np.vstack((a,b))`	Stack arrays vertically (row-wise)
`>>> np.hsplit(c,2)`	Split the array horizontally at the 2nd index
`>>> np.vpslit(d,2)`	Split the array vertically at the 2nd index

Polynomials

`>>> from numpy import poly1d` `>>> p = poly1d([3,4,5])`	Create a polynomial object

Vectorizing Functions

```
>>> def myfunc(a):
        if a < 0:
          return a*2
        else:
          return a/2
>>> np.vectorize(myfunc)
```

Vectorize functions

Type Handling

`>>> np.real(b)`	Return the real part of the array elements
`>>> np.imag(b)`	Return the imaginary part of the array elements
`>>> np.real_if_close(c,tol=1000)`	Return a real array if complex parts close to 0
`>>> np.cast['f'](np.pi)`	Cast object to a data type

Other Useful Functions

`>>> np.angle(b,deg=True)`	Return the angle of the complex argument
`>>> g = np.linspace(0,np.pi,num=5)`	Create an array of evenly spaced values (number of samples)
`>>> g [3:] += np.pi`	
`>>> np.unwrap(g)`	Unwrap
`>>> np.logspace(0,10,3)`	Create an array of evenly spaced values (log scale)
`>>> np.select([c<4],[c*2])`	Return values from a list of arrays depending on conditions
`>>> misc.factorial(a)`	Factorial
`>>> misc.comb(10,3,exact=True)`	Combine N things taken at k time
`>>> misc.central_diff_weights(3)`	Weights for Np-point central derivative
`>>> misc.derivative(myfunc,1.0)`	Find the n-th derivative of a function at a point

Linear Algebra

Also see NumPy

You'll use the `linalg` and `sparse` modules. Note that `scipy.linalg` contains and expands on `numpy.linalg`.

```
>>> from scipy import linalg, sparse
```

Creating Matrices

```
>>> A = np.matrix(np.random.random((2,2)))
>>> B = np.asmatrix(b)
>>> C = np.mat(np.random.random((10,5)))
>>> D = np.mat([[3,4], [5,6]])
```

Basic Matrix Routines

Inverse	
`>>> A.I`	Inverse
`>>> linalg.inv(A)`	Inverse
Transposition	
`>>> A.T`	Tranpose matrix
`>>> A.H`	Conjugate transposition
Trace	
`>>> np.trace(A)`	Trace
Norm	
`>>> linalg.norm(A)`	Frobenius norm
`>>> linalg.norm(A,1)`	L1 norm (max column sum)
`>>> linalg.norm(A,np.inf)`	L inf norm (max row sum)
Rank	
`>>> np.linalg.matrix_rank(C)`	Matrix rank
Determinant	
`>>> linalg.det(A)`	Determinant
Solving linear problems	
`>>> linalg.solve(A,b)`	Solver for dense matrices
`>>> E = np.mat(a).T`	Solver for dense matrices
`>>> linalg.lstsq(F,E)`	Least-squares solution to linear matrix equation
Generalized inverse	
`>>> linalg.pinv(C)`	Compute the pseudo-inverse of a matrix (least-squares solver)
`>>> linalg.pinv2(C)`	Compute the pseudo-inverse of a matrix (SVD)

Creating Sparse Matrices

`>>> F = np.eye(3, k=1)`	Create a 2X2 identity matrix
`>>> G = np.mat(np.identity(2))`	Create a 2x2 identity matrix
`>>> C[C > 0.5] = 0`	
`>>> H = sparse.csr_matrix(C)`	Compressed Sparse Row matrix
`>>> I = sparse.csc_matrix(D)`	Compressed Sparse Column matrix
`>>> J = sparse.dok_matrix(A)`	Dictionary Of Keys matrix
`>>> E.todense()`	Sparse matrix to full matrix
`>>> sparse.isspmatrix_csc(A)`	Identify sparse matrix

Sparse Matrix Routines

Inverse	
`>>> sparse.linalg.inv(I)`	Inverse
Norm	
`>>> sparse.linalg.norm(I)`	Norm
Solving linear problems	
`>>> sparse.linalg.spsolve(H,I)`	Solver for sparse matrices

Sparse Matrix Functions

`>>> sparse.linalg.expm(I)`	Sparse matrix exponential

Matrix Functions

Addition	
`>>> np.add(A,D)`	Addition
Subtraction	
`>>> np.subtract(A,D)`	Subtraction
Division	
`>>> np.divide(A,D)`	Division
Multiplication	
`>>> A @ D`	Multiplication operator (Python 3)
`>>> np.multiply(D,A)`	Multiplication
`>>> np.dot(A,D)`	Dot product
`>>> np.vdot(A,D)`	Vector dot product
`>>> np.inner(A,D)`	Inner product
`>>> np.outer(A,D)`	Outer product
`>>> np.tensordot(A,D)`	Tensor dot product
`>>> np.kron(A,D)`	Kronecker product
Exponential Functions	
`>>> linalg.expm(A)`	Matrix exponential
`>>> linalg.expm2(A)`	Matrix exponential (Taylor Series)
`>>> linalg.expm3(D)`	Matrix exponential (eigenvalue decomposition)
Logarithm Function	
`>>> linalg.logm(A)`	Matrix logarithm
Trigonometric Functions	
`>>> linalg.sinm(D)`	Matrix sine
`>>> linalg.cosm(D)`	Matrix cosine
`>>> linalg.tanm(A)`	Matrix tangent
Hyperbolic Trigonometric Functions	
`>>> linalg.sinhm(D)`	Hypberbolic matrix sine
`>>> linalg.coshm(D)`	Hyperbolic matrix cosine
`>>> linalg.tanhm(A)`	Hyperbolic matrix tangent
Matrix Sign Function	
`>>> np.signm(A)`	Matrix sign function
Matrix Square Root	
`>>> linalg.sqrtm(A)`	Matrix square root
Arbitrary Functions	
`>>> linalg.funm(A, lambda x: x*x)`	Evaluate matrix function

Decompositions

Eigenvalues and Eigenvectors	
`>>> la, v = linalg.eig(A)`	Solve ordinary or generalized eigenvalue problem for square matrix
`>>> l1, l2 = la`	Unpack eigenvalues
`>>> v[:,0]`	First eigenvector
`>>> v[:,1]`	Second eigenvector
`>>> linalg.eigvals(A)`	Unpack eigenvalues
Singular Value Decomposition	
`>>> U,s,Vh = linalg.svd(B)`	Singular Value Decomposition (SVD)
`>>> M,N = B.shape`	
`>>> Sig = linalg.diagsvd(s,M,N)`	Construct sigma matrix in SVD
LU Decomposition	
`>>> P,L,U = linalg.lu(C)`	LU Decomposition

Sparse Matrix Decompositions

`>>> la, v = sparse.linalg.eigs(F,1)`	Eigenvalues and eigenvectors
`>>> sparse.linalg.svds(H, 2)`	SVD

Asking For Help

```
>>> help(scipy.linalg.diagsvd)
>>> np.info(np.matrix)
```

DataCamp
Learn Python for Data Science Interactively

WTPC
Workshop en técnicas de programación científica

Scipy

One of the core packages. SciPy is built on top of NumPy and implements many specialized scientific computation tools:

- Manly user-friendly.
- Efficient numerical routines such as routines for numerical integration and optimization.

    - Clustering.
    - Fourier transforms.
    - numerical integration, interpolations.
    - data I/O, LAPACK.
    - sparse matrices, linear solvers, optimization.
    - signal processing.
    - statistical functions.

Matplotlib basics and How do we prepare data in general?

Python For Data Science *Cheat Sheet*

Matplotlib

Learn Python **Interactively** at www.DataCamp.com

Matplotlib

Matplotlib is a Python 2D plotting library which produces publication-quality figures in a variety of hardcopy formats and interactive environments across platforms.

matplotlib

1 Prepare The Data

Also see Lists & NumPy

1D Data

```
>>> import numpy as np
>>> x = np.linspace(0, 10, 100)
>>> y = np.cos(x)
>>> z = np.sin(x)
```

2D Data or Images

```
>>> data = 2 * np.random.random((10, 10))
>>> data2 = 3 * np.random.random((10, 10))
>>> Y, X = np.mgrid[-3:3:100j, -3:3:100j]
>>> U = -1 - X**2 + Y
>>> V = 1 + X - Y**2
>>> from matplotlib.cbook import get_sample_data
>>> img = np.load(get_sample_data('axes_grid/bivariate_normal.npy'))
```

2 Create Plot

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

Figure

```
>>> fig = plt.figure()
>>> fig2 = plt.figure(figsize=plt.figaspect(2.0))
```

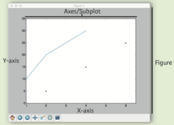
Axes

All plotting is done with respect to an `Axes`. In most cases, a subplot will fit your needs. A subplot is an axes on a grid system.

```
>>> fig.add_axes()
>>> ax1 = fig.add_subplot(221) # row-col-num
>>> ax3 = fig.add_subplot(212)
>>> fig3, axes = plt.subplots(nrows=2,ncols=2)
>>> fig4, axes2 = plt.subplots(ncols=3)
```

Plot Anatomy & Workflow

Plot Anatomy



Workflow

The basic steps to creating plots with matplotlib are:

1 Prepare data 2 Create plot 3 Plot 4 Customize plot 5 Save plot 6 Show plot

```
>>> import matplotlib.pyplot as plt
>>> x = [1,2,3,4]                          Step 1
>>> y = [10,20,25,30]
>>> fig = plt.figure()                     Step 2
>>> ax = fig.add_subplot(111)              Step 3
>>> ax.plot(x, y, color='lightblue', linewidth=3)   Step 3, 4
>>> ax.scatter([2,4,6],
               [5,15,25],
               color='darkgreen',
               marker='^')
>>> ax.set_xlim(1, 6.5)
>>> plt.savefig('foo.png')
>>> plt.show()                             Step 6
```

4 Customize Plot

Colors, Color Bars & Color Maps

```
>>> plt.plot(x, x, x, x**2, x, x**3)
>>> ax.plot(x, y, alpha = 0.4)
>>> ax.plot(x, y, c='k')
>>> fig.colorbar(im, orientation='horizontal')
>>> im = ax.imshow(img,
                   cmap='seismic')
```

Markers

```
>>> fig, ax = plt.subplots()
>>> ax.scatter(x,y,marker=".")
>>> ax.plot(x,y,marker="o")
```

Linestyles

```
>>> plt.plot(x,y,linewidth=4.0)
>>> plt.plot(x,y,ls='solid')
>>> plt.plot(x,y,ls='--')
>>> plt.plot(x,y,'--',x**2,y**2,'-.')
>>> plt.setp(lines,color='r',linewidth=4.0)
```

Text & Annotations

```
>>> ax.text(1,
            -2.1,
            'Example Graph',
            style='italic')
>>> ax.annotate("Sine",
                xy=(8, 0),
                xycoords='data',
                xytext=(10.5, 0),
                textcoords='data',
                arrowprops=dict(arrowstyle="->",
                                connectionstyle="arc3"),)
```

Mathtext

```
>>> plt.title(r'$sigma_i=15$', fontsize=20)
```

Limits, Legends & Layouts

Limits & Autoscaling

```
>>> ax.margins(x=0.0,y=0.1)                        Add padding to a plot
>>> ax.axis('equal')                               Set the aspect ratio of the plot to 1
>>> ax.set(xlim=[0,10.5],ylim=[-1.5,1.5])          Set limits for x-and y-axis
>>> ax.set_xlim(0,10.5)                            Set limits for x-axis
```

Legends

```
>>> ax.set(title='An Example Axes',                Set a title and x-and y-axis labels
           ylabel='Y-Axis',
           xlabel='X-Axis')
>>> ax.legend(loc='best')                          No overlapping plot elements
```

Ticks

```
>>> ax.xaxis.set(ticks=range(1,5),                 Manually set x-ticks
                 ticklabels=[3,100,-12,"foo"])
>>> ax.tick_params(axis='y',                       Make y-ticks longer and go in and out
                   direction='inout',
                   length=10)
```

Subplot Spacing

```
>>> fig3.subplots_adjust(wspace=0.5,               Adjust the spacing between subplots
                         hspace=0.3,
                         left=0.125,
                         right=0.9,
                         top=0.9,
                         bottom=0.1)
>>> fig.tight_layout()                             Fit subplot(s) in to the figure area
```

Axis Spines

```
>>> ax1.spines['top'].set_visible(False)           Make the top axis line for a plot invisible
>>> ax1.spines['bottom'].set_position(('outward',10))   Move the bottom axis line outward
```

3 Plotting Routines

1D Data

`>>> fig, ax = plt.subplots()`	
`>>> lines = ax.plot(x,y)`	Draw points with lines or markers connecting them
`>>> ax.scatter(x,y)`	Draw unconnected points, scaled or colored
`>>> axes[0,0].bar([1,2,3],[3,4,5])`	Plot vertical rectangles (constant width)
`>>> axes[1,0].barh([0.5,1,2.5],[0,1,2])`	Plot horiontal rectangles (constant height)
`>>> axes[1,1].axhline(0.45)`	Draw a horizontal line across axes
`>>> axes[0,1].axvline(0.65)`	Draw a vertical line across axes
`>>> ax.fill(x,y,color='blue')`	Draw filled polygons
`>>> ax.fill_between(x,y,color='yellow')`	Fill between y-values and 0

Vector Fields

`>>> axes[0,1].arrow(0,0,0.5,0.5)`	Add an arrow to the axes
`>>> axes[1,1].quiver(y,z)`	Plot a 2D field of arrows
`>>> axes[0,1].streamplot(X,Y,U,V)`	Plot a 2D field of arrows

Data Distributions

`>>> ax1.hist(y)`	Plot a histogram
`>>> ax3.boxplot(y)`	Make a box and whisker plot
`>>> ax3.violinplot(z)`	Make a violin plot

2D Data or Images

```
>>> fig, ax = plt.subplots()
>>> im = ax.imshow(img,
                   cmap='gist_earth',
                   interpolation='nearest',
                   vmin=-2,
                   vmax=2)
```

Colormapped or RGB arrays

`>>> axes2[0].pcolor(data2)`	Pseudocolor plot of 2D array
`>>> axes2[0].pcolormesh(data)`	Pseudocolor plot of 2D array
`>>> CS = plt.contour(Y,X,U)`	Plot contours
`>>> axes2[2].contourf(data1)`	Plot filled contours
`>>> axes2[2]= ax.clabel(CS)`	Label a contour plot

5 Save Plot

Save figures

```
>>> plt.savefig('foo.png')
```

Save transparent figures

```
>>> plt.savefig('foo.png', transparent=True)
```

6 Show Plot

```
>>> plt.show()
```

Close & Clear

`>>> plt.cla()`	Clear an axis
`>>> plt.clf()`	Clear the entire figure
`>>> plt.close()`	Close a window

DataCamp

Learn Python for Data Science Interactively

WTPC
Workshop en técnicas de programación científica

Matplotlib

matplotlib

Is a Python 2D plotting library which produces publication quality figures in a variety of hardcopy formats and interactive environments across platforms. Can be used in:

- Python scripts.
- The Python and IPython shell.
- The jupyter notebook.
- Web application servers.
- Graphical user interface toolkits.

WTPC
Workshop en técnicas de programación científica

How to start?

It is necessary to import libraries:

```
import numpy
import scipy
import matplotlib.pyplot
```

There are different way to import libraries:

```
import numpy
import numpy as np
from numpy import *
```

I use:

```
import numpy as np
import scipy as sp
import matplotlib.pyplot as pp
import matplotlib.pyplot as plt
```

A quick example:

How to create data set, plot and save in .txt

```
from math import *
import numpy as np
import matplotlib.pyplot as pp
#File creation
f_out_max = open("tabla.txt", "w")
#data generation
x    = np.arange(441)
Sin1 = 1*np.sin(2*pi*(25/441.0)*x)
Sin2 = 0.25*np.sin(2*pi*((25./2)/441.0)*x)
Sin  = Sin1+Sin2
Vec  = np.c_[x,Sin]
#printing
print ("Vec: ",Vec.shape)
#save in .txt file
np.savetxt(f_out_max,Vec,fmt=" %f",delimiter="\t",header="x #f(x)")
f_out_max.close()
#plot figure
pp.figure()
pp.plot(x*1./44100.,Sin, color="r",label="Sin vs. x")
pp.xlabel("Time [s]")
pp.savefig("fig_01.jpg",dpi=200)
```

How to create data set, plot and save in .txt

```
from math import *
import numpy as np
import matplotlib.pyplot as pp
#File creation
f_out_max = open("tabla.txt", "w")
#data generation
x    = np.arange(441)
Sin1 = 1*np.sin(2*pi*(25/441.0)*x)
Sin2 = 0.25*np.sin(2*pi*((25./2)/441.0)*x)
Sin  = Sin1+Sin2
Vec  = np.c_[x,Sin]
#printing
print ("Vec: ",Vec.shape)
#save in .txt file
np.savetxt(f_out_max,Vec,fmt="%f",delimiter="\t",header="x #f(x)")
f_out_max.close()
#plot figure
pp.figure()
pp.plot(x*1./44100.,Sin, color="r",label="Sin vs. x")
pp.xlabel("Time [s]")
pp.savefig("fig_01.jpg",dpi=200)
```

A quick example:

How to create data set, plot and save in .txt

```
from math import *
import numpy as np
import matplotlib.pyplot as pp
#File creation
f_out_max = open("tabla.txt", "w")
#data generation
x    = np.arange(441)
Sin1 = 1*np.sin(2*pi*(25/441.0)*x)
Sin2 = 0.25*np.sin(2*pi*((25./2)/441.0)*x)
Sin  = Sin1+Sin2
Vec  = np.c_[x,Sin]
#printing
print ("Vec: ",Vec.shape)
#save in .txt file
np.savetxt(f_out_max,Vec,fmt=" %f",delimiter="\t",header="x #f(x)")
f_out_max.close()
#plot figure
pp.figure()
pp.plot(x*1./44100.,Sin, color="r",label="Sin vs. x")
pp.xlabel("Time [s]")
pp.savefig("fig_01.jpg",dpi=200)
```

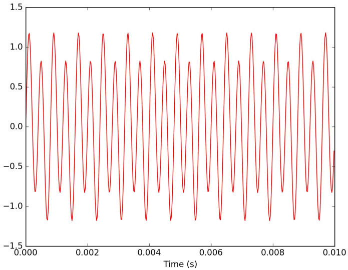
A quick example:

How to create data set, plot and save in .txt

```
from math import *
import numpy as np
import matplotlib.pyplot as pp
#File creation
f_out_max = open("tabla.txt", "w")
#data generation
x    = np.arange(441)
Sin1 = 1*np.sin(2*pi*(25/441.0)*x)
Sin2 = 0.25*np.sin(2*pi*((25./2)/441.0)*x)
Sin  = Sin1+Sin2
Vec  = np.c_[x,Sin]
#printing
print ("Vec: ",Vec.shape)
#save in .txt file
np.savetxt(f_out_max,Vec,fmt=" %f",delimiter="\t",header="x #f(x)")
f_out_max.close()
#plot figure
pp.figure()
pp.plot(x*1./44100.,Sin, color="r",label="Sin vs. x")
pp.xlabel("Time [s]")
pp.savefig("fig_01.jpg",dpi=200)
```

We obtained

A basic figure and table in a .txt file:



# x	#f(x)
0.000000	0.000000
1.000000	0.392994
2.000000	0.740813
3.000000	1.003819
4.000000	1.152764
5.000000	1.172342
6.000000	1.062998
7.000000	0.840786
8.000000	0.535279
9.000000	0.185801
10.000000	-0.163538
11.000000	-0.469389
12.000000	-0.694586
13.000000	-0.812779
14.000000	-0.811673
15.000000	-0.694499
16.000000	-0.479506
17.000000	-0.197559
18.000000	0.111860 ...

A very easy way is using Numpy:

```
import numpy as np

file_name_you_want        = np.loadtxt(fname,delimiter=" ")

print "First column element: ",file_name_you_want[0]
#to get the full column:

Transpose_your_file       = file_name_you_want.T

print "First column: ", Transpose_your_file[0]
```

How to perform a simple fit and nicely plot it

```python
import numpy as np
import matplotlib.pyplot as pp
from scipy.optimize import curve_fit

def fitFunc(t, a, b, c):
    return a*np.exp(-b*t) + c

t                             = np.linspace(0,4,50)
temp                          = fitFunc(t, 2.5, 1.3, 0.5)
noisy                         = temp + 0.25*np.random.normal(size=len(temp))
fitParams, fitCovariances = curve_fit(fitFunc, t, noisy)

pp.figure(figsize=(12, 6))
pp.ylabel("Temperature (C)", fontsize = 16)
pp.xlabel("time (s)", fontsize = 16)
pp.xlim(0,4.1)
pp.errorbar(t, noisy, fmt = "ro", yerr = 0.2)
sigma = [fitCovariances[0,0], fitCovariances[1,1], fitCovariances[2,2] ]
pp.plot(t, fitFunc(t, fitParams[0], fitParams[1], fitParams[2]))
pp.plot(t, fitFunc(t, fitParams[0] + sigma[0], fitParams[1] - sigma[1], fitParams[2] + sigma[2]))
pp.plot(t, fitFunc(t, fitParams[0] - sigma[0], fitParams[1] + sigma[1], fitParams[2] - sigma[2]))
pp.savefig("dataFitted.pdf", bbox_inches=0, dpi=600)
pp.show()
```

WTPC
Workshop en técnicas de programación científica

How to perform a simple fit and nicely plot it

```python
import numpy as np
import matplotlib.pyplot as pp
from scipy.optimize import curve_fit

def fitFunc(t, a, b, c):
    return a*np.exp(-b*t) + c

t                           = np.linspace(0,4,50)
temp                        = fitFunc(t, 2.5, 1.3, 0.5)
noisy                       = temp + 0.25*np.random.normal(size=len(temp))
fitParams, fitCovariances = curve_fit(fitFunc, t, noisy)

pp.figure(figsize=(12, 6))
pp.ylabel("Temperature (C)", fontsize = 16)
pp.xlabel("time (s)", fontsize = 16)
pp.xlim(0,4.1)
pp.errorbar(t, noisy, fmt = "ro", yerr = 0.2)
sigma = [fitCovariances[0,0], fitCovariances[1,1], fitCovariances[2,2] ]
pp.plot(t, fitFunc(t, fitParams[0], fitParams[1], fitParams[2]))
pp.plot(t, fitFunc(t, fitParams[0] + sigma[0], fitParams[1] - sigma[1], fitParams[2] + sigma[2]))
pp.plot(t, fitFunc(t, fitParams[0] - sigma[0], fitParams[1] + sigma[1], fitParams[2] - sigma[2]))
pp.savefig("dataFitted.pdf", bbox_inches=0, dpi=600)
pp.show()
```

Workshop en técnicas de programación científica

How to perform a simple fit and nicely plot it

```python
import numpy as np
import matplotlib.pyplot as pp
from scipy.optimize import curve_fit

def fitFunc(t, a, b, c):
    return a*np.exp(-b*t) + c

t                          = np.linspace(0,4,50)
temp                       = fitFunc(t, 2.5, 1.3, 0.5)
noisy                      = temp + 0.25*np.random.normal(size=len(temp))
fitParams, fitCovariances = curve_fit(fitFunc, t, noisy)

pp.figure(figsize=(12, 6))
pp.ylabel("Temperature (C)", fontsize = 16)
pp.xlabel("time (s)", fontsize = 16)
pp.xlim(0,4.1)
pp.errorbar(t, noisy, fmt = "ro", yerr = 0.2)
sigma = [fitCovariances[0,0], fitCovariances[1,1], fitCovariances[2,2] ]
pp.plot(t, fitFunc(t, fitParams[0], fitParams[1], fitParams[2]))
pp.plot(t, fitFunc(t, fitParams[0] + sigma[0], fitParams[1] - sigma[1], fitParams[2] + sigma[2]))
pp.plot(t, fitFunc(t, fitParams[0] - sigma[0], fitParams[1] + sigma[1], fitParams[2] - sigma[2]))
pp.savefig("dataFitted.pdf", bbox_inches=0, dpi=600)
pp.show()
```

How to perform a simple fit and nicely plot it

```python
import numpy as np
import matplotlib.pyplot as pp
from scipy.optimize import curve_fit

def fitFunc(t, a, b, c):
    return a*np.exp(-b*t) + c

t                          = np.linspace(0,4,50)
temp                       = fitFunc(t, 2.5, 1.3, 0.5)
noisy                      = temp + 0.25*np.random.normal(size=len(temp))
fitParams, fitCovariances = curve_fit(fitFunc, t, noisy)

pp.figure(figsize=(12, 6))
pp.ylabel("Temperature (C)", fontsize = 16)
pp.xlabel("time (s)", fontsize = 16)
pp.xlim(0,4.1)
pp.errorbar(t, noisy, fmt = "ro", yerr = 0.2)
sigma = [fitCovariances[0,0], fitCovariances[1,1], fitCovariances[2,2] ]
pp.plot(t, fitFunc(t, fitParams[0], fitParams[1], fitParams[2]))
pp.plot(t, fitFunc(t, fitParams[0] + sigma[0], fitParams[1] - sigma[1], fitParams[2] + sigma[2]))
pp.plot(t, fitFunc(t, fitParams[0] - sigma[0], fitParams[1] + sigma[1], fitParams[2] - sigma[2]))
pp.savefig("dataFitted.pdf", bbox_inches=0, dpi=600)
pp.show()
```

WTPC Workshop en técnicas de programación científica

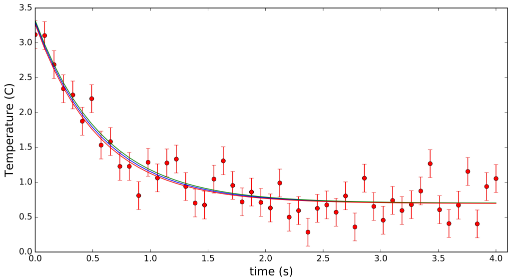
How to perform a simple fit

We obtained the fit parameters:

```
[ 2.595658    1.74438726  0.69809511]
```

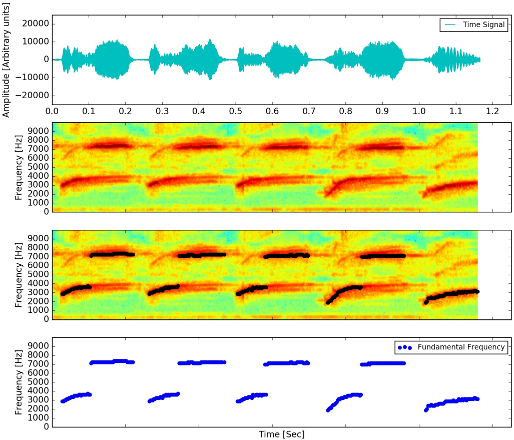
The covariance matrix:

```
[[ 0.02506636  0.01490486 -0.00068609]
 [ 0.01490486  0.04178044  0.00641246]
 [-0.00068609  0.00641246  0.00257799]]
```



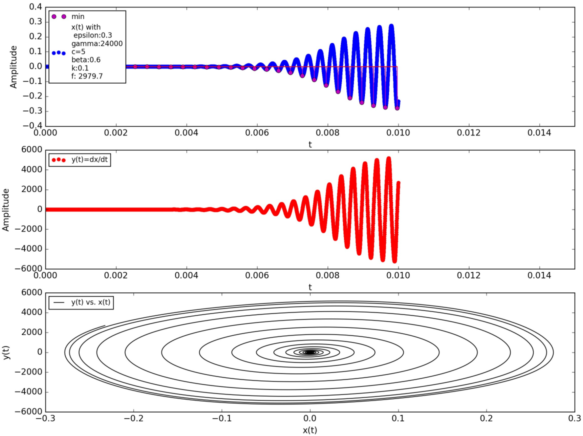
Fourier transform and Spectrograms

```python
import matplotlib.pyplot as pp
pp.specgram(signal, NFFT=nfft, Fs= sample_rate, noverlap=par, cmap='jet')
```



Numerical integration

```
from scipy.integrate import odeint
dy/dt = func(y, t0, ...)
sol   = odeint(func, X0, t)
```



What else do we need todo with data?

What can we do with data?

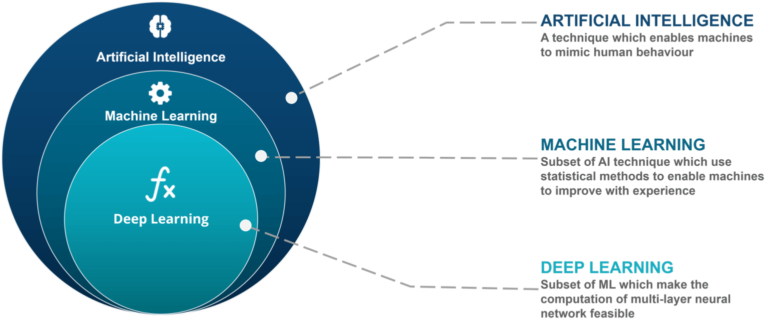
We are interested in:

- Data Visualization.
- Data Analysis (Identify features from the data).
- Data Classification.
- Implementation of different algorithms as intelligent as we can get.

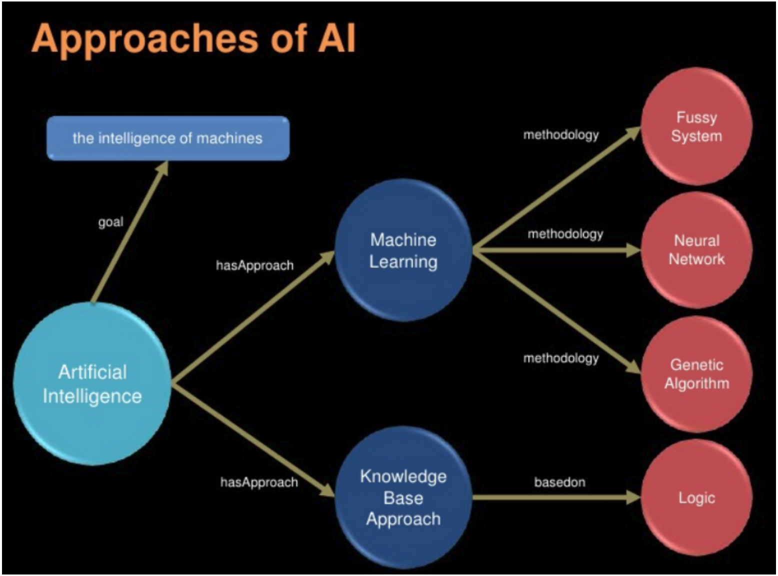
Which are the available algorithms?

TYPE	NAME	DESCRIPTION	ADVANTAGES	DISADVANTAGES
Linear	Linear regression	The **"best fit" line** through all data points. Predictions are numerical.	Easy to understand -- you clearly see what the biggest drivers of the model are.	**X** Sometimes too simple to capture complex relationships between variables. **X** Tendency for the model to "overfit".
Linear	Logistic regression	The adaptation of **linear regression** to problems of classification (e.g., yes/no questions, groups, etc.)	Also easy to understand.	**X** Sometimes too simple to capture complex relationships between variables. **X** Tendency for the model to "overfit".
Tree-based	Decision tree	A graph that uses a **branching method** to match all possible outcomes of a decision.	Easy to understand and implement.	**X** Not often used on its own for prediction because it's also often too simple and not powerful enough for complex data.
Tree-based	Random Forest	Takes the average of many decision trees, each of which is made with a sample of the data. Each tree is weaker than a full decision tree, but **by combining them we get better overall performance.**	A sort of "wisdom of the crowd". Tends to result in very high quality models. Fast to train.	**X** Can be slow to output predictions relative to other algorithms. **X** Not easy to understand predictions.
Tree-based	Gradient Boosting	Uses even weaker decision trees, that are increasingly **focused on "hard" examples.**	High-performing.	**X** A small change in the feature set or training set can create radical changes in the model. **X** Not easy to understand predictions.
Neural networks	Neural networks	Mimics the behavior of the brain. Neural networks are interconnected neurons that pass messages to each other. Deep learning uses several **layers of neural networks put one after the other.**	Can handle extremely complex tasks - no other algorithm comes close in image recognition.	**X** Very, very slow to train, because they have so many layers. Require a lot of power. **X** Almost impossible to understand predictions.

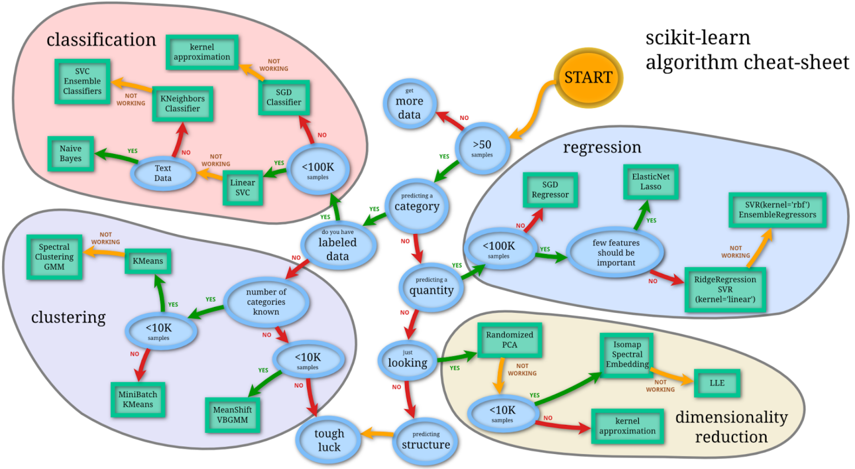
What about machine learning?



What we have?



Approaches of AI
the intelligence of machines
goal
Artificial Intelligence
hasApproach
Machine Learning
hasApproach
Knowledge Base Approach
methodology
Fussy System
methodology
Neural Network
methodology
Genetic Algorithm
basedon
Logic


classification
regression
clustering
dimensionality reduction
scikit-learn algorithm cheat-sheet
START
>50 samples
get more data
predicting a category
do you have labeled data
predicting a quantity
just looking
predicting structure
tough luck
<100K samples
SGD Classifier
kernel approximation
KNeighbors Classifier
SVC Ensemble Classifiers
Linear SVC
Text Data
Naive Bayes
number of categories known
<10K samples
KMeans
Spectral Clustering GMM
MiniBatch KMeans
MeanShift VBGMM
SGD Regressor
ElasticNet Lasso
few features should be important
RidgeRegression SVR (kernel='linear')
SVR(kernel='rbf') EnsembleRegressors
Randomized PCA
Isomap Spectral Embedding
LLE
kernel approximation
YES
NO
NOT WORKING

Scikit-learn Basics:

Python For Data Science *Cheat Sheet*

Scikit-Learn

Learn Python for data science **Interactively** at www.DataCamp.com

Scikit-learn

Scikit-learn is an open source Python library that implements a range of machine learning, preprocessing, cross-validation and visualization algorithms using a unified interface.

A Basic Example

```
>>> from sklearn import neighbors, datasets, preprocessing
>>> from sklearn.cross_validation import train_test_split
>>> from sklearn.metrics import accuracy_score
>>> iris = datasets.load_iris()
>>> X, y = iris.data[:, :2], iris.target
>>> X_train, X_test, y_train, y_test= train_test_split(X, y, random_state=33)
>>> scaler = preprocessing.StandardScaler().fit(X_train)
>>> X_train = scaler.transform(X_train)
>>> X_test = scaler.transform(X_test)
>>> knn = neighbors.KNeighborsClassifier(n_neighbors=5)
>>> knn.fit(X_train, y_train)
>>> y_pred = knn.predict(X_test)
>>> accuracy_score(y_test, y_pred)
```

Loading The Data

Also see NumPy & Pandas

Your data needs to be numeric and stored as NumPy arrays or SciPy sparse matrices. Other types that are convertible to numeric arrays, such as Pandas DataFrame, are also acceptable.

```
>>> import numpy as np
>>> X = np.random.random((10,5))
>>> y = np.array(['M','M','F','F','M','F','M','M','F','F','F'])
>>> X[X < 0.7] = 0
```

Training And Test Data

```
>>> from sklearn.cross_validation import train_test_split
>>> X_train, X_test, y_train, y_test = train_test_split(X,
                                                          y,
                                                          random_state=0)
```

Create Your Model

Supervised Learning Estimators

Linear Regression

```
>>> from sklearn.linear_model import LinearRegression
>>> lr = LinearRegression(normalize=True)
```

Support Vector Machines (SVM)

```
>>> from sklearn.svm import SVC
>>> svc = SVC(kernel='linear')
```

Naive Bayes

```
>>> from sklearn.naive_bayes import GaussianNB
>>> gnb = GaussianNB()
```

KNN

```
>>> from sklearn import neighbors
>>> knn = neighbors.KNeighborsClassifier(n_neighbors=5)
```

Unsupervised Learning Estimators

Principal Component Analysis (PCA)

```
>>> from sklearn.decomposition import PCA
>>> pca = PCA(n_components=0.95)
```

K Means

```
>>> from sklearn.cluster import KMeans
>>> k_means = KMeans(n_clusters=3, random_state=0)
```

Model Fitting

Supervised learning

```
>>> lr.fit(X, y)
>>> knn.fit(X_train, y_train)
>>> svc.fit(X_train, y_train)
```

Fit the model to the data

Unsupervised Learning

```
>>> k_means.fit(X_train)
>>> pca_model = pca.fit_transform(X_train)
```

Fit the model to the data
Fit to data, then transform it

Prediction

Supervised Estimators

```
>>> y_pred = svc.predict(np.random.random((2,5)))
>>> y_pred = lr.predict(X_test)
>>> y_pred = knn.predict_proba(X_test)
```

Predict labels
Predict labels
Estimate probability of a label

Unsupervised Estimators

```
>>> y_pred = k_means.predict(X_test)
```

Predict labels in clustering algos

Preprocessing The Data

Standardization

```
>>> from sklearn.preprocessing import StandardScaler
>>> scaler = StandardScaler().fit(X_train)
>>> standardized_X = scaler.transform(X_train)
>>> standardized_X_test = scaler.transform(X_test)
```

Normalization

```
>>> from sklearn.preprocessing import Normalizer
>>> scaler = Normalizer().fit(X_train)
>>> normalized_X = scaler.transform(X_train)
>>> normalized_X_test = scaler.transform(X_test)
```

Binarization

```
>>> from sklearn.preprocessing import Binarizer
>>> binarizer = Binarizer(threshold=0.0).fit(X)
>>> binary_X = binarizer.transform(X)
```

Encoding Categorical Features

```
>>> from sklearn.preprocessing import LabelEncoder
>>> enc = LabelEncoder()
>>> y = enc.fit_transform(y)
```

Imputing Missing Values

```
>>> from sklearn.preprocessing import Imputer
>>> imp = Imputer(missing_values=0, strategy='mean', axis=0)
>>> imp.fit_transform(X_train)
```

Generating Polynomial Features

```
>>> from sklearn.preprocessing import PolynomialFeatures
>>> poly = PolynomialFeatures(5)
>>> poly.fit_transform(X)
```

Evaluate Your Model's Performance

Classification Metrics

Accuracy Score

```
>>> knn.score(X_test, y_test)
>>> from sklearn.metrics import accuracy_score
>>> accuracy_score(y_test, y_pred)
```

Estimator score method
Metric scoring functions

Classification Report

```
>>> from sklearn.metrics import classification_report
>>> print(classification_report(y_test, y_pred))
```

Precision, recall, f1-score and support

Confusion Matrix

```
>>> from sklearn.metrics import confusion_matrix
>>> print(confusion_matrix(y_test, y_pred))
```

Regression Metrics

Mean Absolute Error

```
>>> from sklearn.metrics import mean_absolute_error
>>> y_true = [3, -0.5, 2]
>>> mean_absolute_error(y_true, y_pred)
```

Mean Squared Error

```
>>> from sklearn.metrics import mean_squared_error
>>> mean_squared_error(y_test, y_pred)
```

R^2 Score

```
>>> from sklearn.metrics import r2_score
>>> r2_score(y_true, y_pred)
```

Clustering Metrics

Adjusted Rand Index

```
>>> from sklearn.metrics import adjusted_rand_score
>>> adjusted_rand_score(y_true, y_pred)
```

Homogeneity

```
>>> from sklearn.metrics import homogeneity_score
>>> homogeneity_score(y_true, y_pred)
```

V-measure

```
>>> from sklearn.metrics import v_measure_score
>>> metrics.v_measure_score(y_true, y_pred)
```

Cross-Validation

```
>>> from sklearn.cross_validation import cross_val_score
>>> print(cross_val_score(knn, X_train, y_train, cv=4))
>>> print(cross_val_score(lr, X, y, cv=2))
```

Tune Your Model

Grid Search

```
>>> from sklearn.grid_search import GridSearchCV
>>> params = {"n_neighbors": np.arange(1,3),
              "metric": ["euclidean", "cityblock"]}
>>> grid = GridSearchCV(estimator=knn,
                        param_grid=params)
>>> grid.fit(X_train, y_train)
>>> print(grid.best_score_)
>>> print(grid.best_estimator_.n_neighbors)
```

Randomized Parameter Optimization

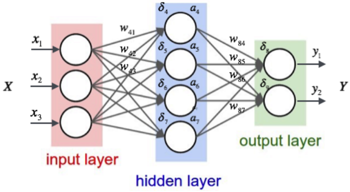
```
>>> from sklearn.grid_search import RandomizedSearchCV
>>> params = {"n_neighbors": range(1,5),
              "weights": ["uniform", "distance"]}
>>> rsearch = RandomizedSearchCV(estimator=knn,
                                 param_distributions=params,
                                 cv=4,
                                 n_iter=8,
                                 random_state=5)
>>> rsearch.fit(X_train, y_train)
>>> print(rsearch.best_score_)
```

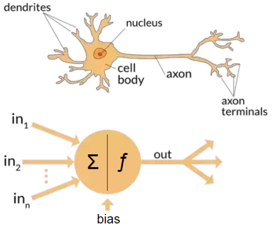
DataCamp
Learn Python for Data Science Interactively

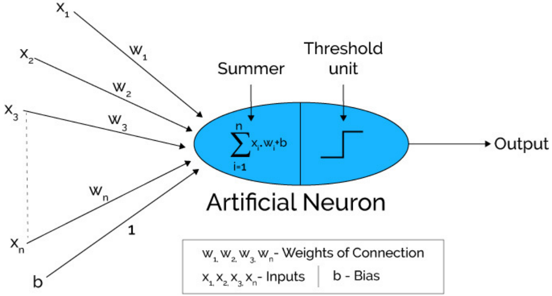
Workshop en técnicas de programación científica

Hands on in the afternoon to learn some basic use of a set of functions!

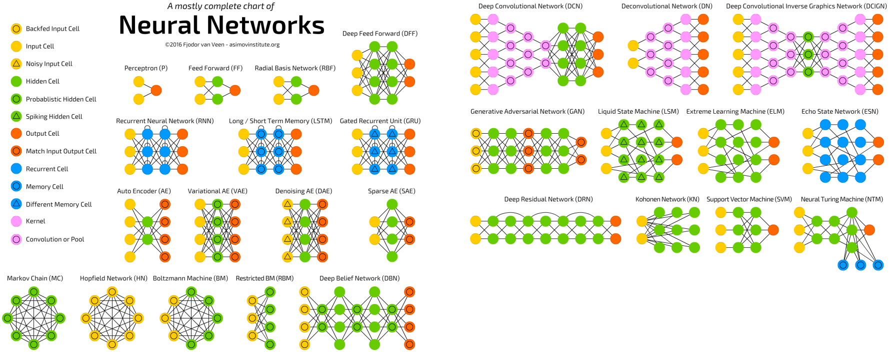
Neural Networks:







Neural Networks:



How do we implement this algorithms?

- From zero with math libraries and python.
- Using dedicated open source frameworks:
  - Tensorflow.
  - Keras.

Keras Basics:

Keras: A high-level neural networks API, written in Python and capable of running on top of TensorFlow, CNTK, or Theano. It was developed with a focus on enabling fast experimentation. Being able to go from idea to result with the least possible delay is key to doing good research.

Keras Basics:

Python For Data Science *Cheat Sheet*

Keras

Learn Python for data science **Interactively** at www.DataCamp.com

Keras

Keras is a powerful and easy-to-use deep learning library for Theano and TensorFlow that provides a high-level neural networks API to develop and evaluate deep learning models.

A Basic Example

```
>>> import numpy as np
>>> from keras.models import Sequential
>>> from keras.layers import Dense
>>> data = np.random.random((1000,100))
>>> labels = np.random.randint(2,size=(1000,1))
>>> model = Sequential()
>>> model.add(Dense(32,
                    activation='relu',
                    input_dim=100))
>>> model.add(Dense(1, activation='sigmoid'))
>>> model.compile(optimizer='rmsprop',
                  loss='binary_crossentropy',
                  metrics=['accuracy'])
>>> model.fit(data,labels,epochs=10,batch_size=32)
>>> predictions = model.predict(data)
```

Data

Also see NumPy, Pandas & Scikit-Learn

Your data needs to be stored as NumPy arrays or as a list of NumPy arrays. Ideally, you split the data in training and test sets, for which you can also resort to the `train_test_split` module of `sklearn.cross_validation`.

Keras Data Sets

```
>>> from keras.datasets import boston_housing,
                               mnist,
                               cifar10,
                               imdb
>>> (x_train,y_train),(x_test,y_test) = mnist.load_data()
>>> (x_train2,y_train2),(x_test2,y_test2) = boston_housing.load_data()
>>> (x_train3,y_train3),(x_test3,y_test3) = cifar10.load_data()
>>> (x_train4,y_train4),(x_test4,y_test4) = imdb.load_data(num_words=20000)
>>> num_classes = 10
```

Other

```
>>> from urllib.request import urlopen
>>> data = np.loadtxt(urlopen("http://archive.ics.uci.edu/
ml/machine-learning-databases/pima-indians-diabetes/
pima-indians-diabetes.data"),delimiter=",")
>>> X = data[:,0:8]
>>> y = data [:,8]
```

Preprocessing

Also see NumPy & Scikit-Learn

Sequence Padding

```
>>> from keras.preprocessing import sequence
>>> x_train4 = sequence.pad_sequences(x_train4,maxlen=80)
>>> x_test4 = sequence.pad_sequences(x_test4,maxlen=80)
```

One-Hot Encoding

```
>>> from keras.utils import to_categorical
>>> Y_train = to_categorical(y_train, num_classes)
>>> Y_test = to_categorical(y_test, num_classes)
>>> Y_train3 = to_categorical(y_train3, num_classes)
>>> Y_test3 = to_categorical(y_test3, num_classes)
```

Train and Test Sets

```
>>> from sklearn.model_selection import train_test_split
>>> X_train5,X_test5,y_train5,y_test5 = train_test_split(X,
                                                         y,
                                                         test_size=0.33,
                                                         random_state=42)
```

Standardization/Normalization

```
>>> from sklearn.preprocessing import StandardScaler
>>> scaler = StandardScaler().fit(x_train2)
>>> standardized_X = scaler.transform(x_train2)
>>> standardized_X_test = scaler.transform(x_test2)
```

Model Architecture

Sequential Model

```
>>> from keras.models import Sequential
>>> model = Sequential()
>>> model2 = Sequential()
>>> model3 = Sequential()
```

Multilayer Perceptron (MLP)

Binary Classification

```
>>> from keras.layers import Dense
>>> model.add(Dense(12,
                    input_dim=8,
                    kernel_initializer='uniform',
                    activation='relu'))
>>> model.add(Dense(8,kernel_initializer='uniform',activation='relu'))
>>> model.add(Dense(1,kernel_initializer='uniform',activation='sigmoid'))
```

Multi-Class Classification

```
>>> from keras.layers import Dropout
>>> model.add(Dense(512,activation='relu',input_shape=(784,)))
>>> model.add(Dropout(0.2))
>>> model.add(Dense(512,activation='relu'))
>>> model.add(Dropout(0.2))
>>> model.add(Dense(10,activation='softmax'))
```

Regression

```
>>> model.add(Dense(64,activation='relu',input_dim=train_data.shape[1]))
>>> model.add(Dense(1))
```

Convolutional Neural Network (CNN)

```
>>> from keras.layers import Activation,Conv2D,MaxPooling2D,Flatten
>>> model2.add(Conv2D(32,(3,3),padding='same',input_shape=x_train.shape[1:]))
>>> model2.add(Activation('relu'))
>>> model2.add(Conv2D(32,(3,3)))
>>> model2.add(Activation('relu'))
>>> model2.add(MaxPooling2D(pool_size=(2,2)))
>>> model2.add(Dropout(0.25))

>>> model2.add(Conv2D(64,(3,3), padding='same'))
>>> model2.add(Activation('relu'))
>>> model2.add(Conv2D(64,(3, 3)))
>>> model2.add(Activation('relu'))
>>> model2.add(MaxPooling2D(pool_size=(2,2)))
>>> model2.add(Dropout(0.25))

>>> model2.add(Flatten())
>>> model2.add(Dense(512))
>>> model2.add(Activation('relu'))
>>> model2.add(Dropout(0.5))
>>> model2.add(Dense(num_classes))
>>> model2.add(Activation('softmax'))
```

Recurrent Neural Network (RNN)

```
>>> from keras.klayers import Embedding,LSTM
>>> model3.add(Embedding(20000,128))
>>> model3.add(LSTM(128,dropout=0.2,recurrent_dropout=0.2))
>>> model3.add(Dense(1,activation='sigmoid'))
```

Inspect Model

`>>> model.output_shape`	Model output shape
`>>> model.summary()`	Model summary representation
`>>> model.get_config()`	Model configuration
`>>> model.get_weights()`	List all weight tensors in the model

Compile Model

MLP: Binary Classification

```
>>> model.compile(optimizer='adam',
                  loss='binary_crossentropy',
                  metrics=['accuracy'])
```

MLP: Multi-Class Classification

```
>>> model.compile(optimizer='rmsprop',
                  loss='categorical_crossentropy',
                  metrics=['accuracy'])
```

MLP: Regression

```
>>> model.compile(optimizer='rmsprop',
                  loss='mse',
                  metrics=['mae'])
```

Recurrent Neural Network

```
>>> model3.compile(loss='binary_crossentropy',
                   optimizer='adam',
                   metrics=['accuracy'])
```

Model Training

```
>>> model3.fit(x_train4,
               y_train4,
               batch_size=32,
               epochs=15,
               verbose=1,
               validation_data=(x_test4,y_test4))
```

Evaluate Your Model's Performance

```
>>> score = model3.evaluate(x_test,
                            y_test,
                            batch_size=32)
```

Prediction

```
>>> model3.predict(x_test4, batch_size=32)
>>> model3.predict_classes(x_test4,batch_size=32)
```

Save/ Reload Models

```
>>> from keras.models import load_model
>>> model3.save('model_file.h5')
>>> my_model = load_model('my_model.h5')
```

Model Fine-tuning

Optimization Parameters

```
>>> from keras.optimizers import RMSprop
>>> opt = RMSprop(lr=0.0001, decay=1e-6)
>>> model2.compile(loss='categorical_crossentropy',
                   optimizer=opt,
                   metrics=['accuracy'])
```

Early Stopping

```
>>> from keras.callbacks import EarlyStopping
>>> early_stopping_monitor = EarlyStopping(patience=2)
>>> model3.fit(x_train4,
               y_train4,
               batch_size=32,
               epochs=15,
               validation_data=(x_test4,y_test4),
               callbacks=[early_stopping_monitor])
```

DataCamp
Learn Python for Data Science **Interactively**

Workshop en técnicas de programación científica

A second Afternoon tutorial!

To Finish: How to “code” to be understood?

What is this part of the talk about?

- Try to remove bad habits of coding alone.
- Adopt useful tools that can make the collaborative development task easier.
- At least allow you to crosscheck your work with others!

Summary of the general ideas:

- Group work.
- Re-usable code.
- Version control Software.
- Clear documentation.
- Do not reinvent gunpowder (that is, use tested libraries for standard calculus).

The actual problem:

First step: My code, our code... Whose is the code?

Need to consider all levels for different audiences:

- Code annotations: formatting, comments.
- Structure-level documentation (functions, classes).
- End-user reference material.
- Introduction for new users...
- ... and new developers (often left out!)

Code writing

Code will be read much more often than written.

- Clarity wins over cleverness.
- Choose a style and stick with it: block structure, indentation, line length, variable naming, ...
- Know the conventions of the language community (in Python, look for PEP 8).

Be consistent with others!

WTPC Workshop en técnicas de programación científica

Code comments

In python:

```
# Here my nice and elegant comment
    of my code in Python
```

In fortran 90:

```
!* All characters after an
    exclamation mark are
    considered as comments *
```

In C:

```
/* Here my nice and elegant comment
     of my code in C*/

//also like this if is only one
    line
```

Code comments

Explain the intention and not the work:

```python
def tripleTuple(x):
# assign x to y
y = x
# assign x to z
z = x
# double y
y = 2
# triple z
z = 3
# create tuple
t = (x, y, z)
# return the tuple
return t
```

```python
def tripleTuple(x):
    y = z = x
    # apply foo scaling, see [34] eq (2.3)
    y = 2
    z = 3
    return (x, y, z)

```

- Deviations from standard.
- Unexpected choices of implementation.

Documentation:

- Dual audience: developers and end users.
- Describe role of function.
- Intended input / output.
- Mention side effects!

Automatic tools

Extract documentation from code

- Pydoc
- Doxygen
- Sphinx

Documentation: Pydoc

Pydoc extracts docstrings defined in
`http://www.python.org/dev/peps/pep-0257/`

- A docstring is a string literal that occurs as the first statement in a module, function, class, or method definition.
- Such a docstring becomes the *__doc__* special attribute of that object.
- Every Python installation has pydoc, no extra work required.

Documentation: Doxygen

- Supports C++, C, Obj-C, C#, PHP, Java, Python.
- IDL, Fortran, VHDL, Tcl.
- Can extract structure from undocumented files.
- Graphical visualization of dependencies.
- Can write general pages, too.

Documentation: Sphinx

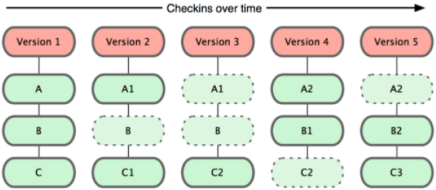
- Automatic extraction of Python docstrings conversion from Doxygen.
- Great support for additional structure.
- Used in standard library, matplotlib, scipy, numpy.
- Like most Python projects, excellent support for beginners:

```
sphinx-quickstart
```

Some considerations

- New developers need an intro.
- Very different requirements than users.
- Doxygen alone is not enough.
- At minimum, prepare some paths to follow through the code docs.

Version control system: git



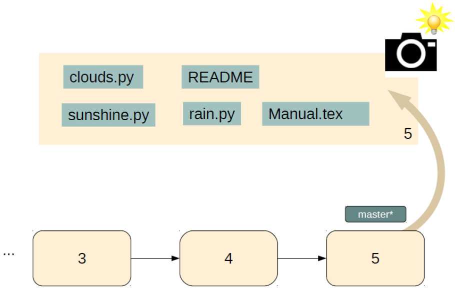
www.git-scm.com

Other good practice: Version control software

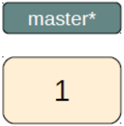
- Storing Versions Properly (Allows to document changes on the code).
- Allows a team of people to work programing together, all using the same files.
- Ordered way of multiple people editing the same files.
- Allows restoring Previous Versions.
- Allows shear the changes for all contributors.

Other nice practice: version control software

An example of a project:



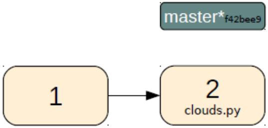
```
> git init
```




```
> git status
On branch master
Initial commit
nothing to commit (create/copy files and use "git add" to track)
```

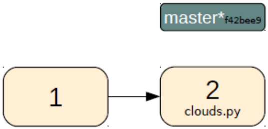
An example for version control system: git

```
> vim clouds.py
> git add clouds.py
> git commit -m 'test cases added'
[master (root-commit) f42bee9] test case
1 file changed, 1 insertion(+)
create mode 100644 clouds.py
```



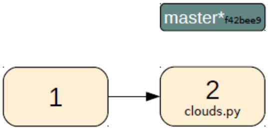
An example for version control system: git

```
> git log
commit f28ef8b10951a4ecc95dc911bbe7018e3c0225fc
Author: Cecilia Jarne <cecilia.jarne@unq.edu.ar>
Date: Mon Oct 26 12:36:46 2015 -0300
test cases added
```

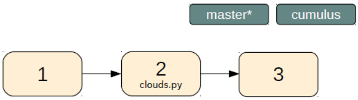


An example for version control system: git

```
> vim wind.py
> git status
On branch master
Untracked files:
(use "git add <file>..." to include in what will be committed)
wind.py
nothing added to commit
```

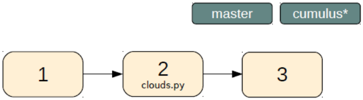


```
> git branch cumulus
```



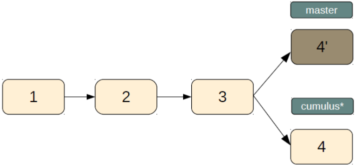
It is a pointer (does not copy the code!)

```
> git checkout cumulus
```



An example for version control system: git

```
>vim clouds.py #to modify clouds.py
> git add clouds.py
> git commit -m 'added parametrization [cumulus]'
> git commit -m 'added parametrization
```

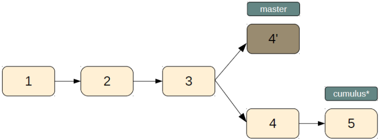


An example for version control system: git

```
> git merge cumulus master
```

or

```
> git merge master
```



An example for version control system: git

To delete files

```
> rm wind.py
> git status
On branch master
Changes not staged for commit:
(use "git add/rm <file>..." to update what will be committed)
(use "git checkout -- <file>..." to discard changes in working directory
    )
deleted: wind.py
no changes added to commit (use "git add" and/or "git commit -a")
```

An example for version control system: git

Some commands on workspace, staging area, local repository

- init #to create repository
- status # repository status
- add #add to the stage
- commit #save in the repository what is in stage
- branch #new branch
- checkout #switch branch
- diff #show differences between branches
- log #show commits

Remote work

```
> git clone https://github.com/<USER>/HOgit.git
```

```
> git push
```

```
> git pull
```

So ....

Final remarks

- Python is a very versatile language to write scripts or complex programs combining C, or Fortran, and other Scientific libraries.
- Scientific libraries with a lot of examples are well doc. Think on what do we need?
- Possibility of re-use and improve existing code.
- A guide of style in: `https://google.github.io/styleguide/py` guide.html
- Matplotlib examples: `http://matplotlib.org/gallery.html`

More about our work and who we are:

https://wtpc.github.io/

Classes and material will be at:

http://ceciliajarne.web.unq.edu.ar/investigacion/