Una introducción a la programación en python

Cecilia Jarne

cecilia.jarne@unq.edu.ar

CONICET

Universidad Nacional de Quilmes

WTPC Workshop en técnicas de programación científica

Qué es Python?

Es un lenguaje de programación interpretado, que permite tipeado dinámico y es multiplataforma.

- Soporta orientación a objetos.
- Programación imperativa.
- Programación funcional.

- Se pueden usar distintos paradigmas de programación.
- Tiene una sintaxis clara.
- Se puede extender su funcionalidad a través de distintas librerías.

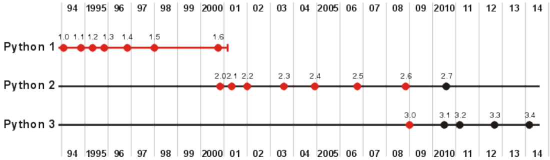
`https://www.python.org`

Motivos para aprender Python

- Fácil de aprender.
- Un conjunto gigante de librerías.
- Soporte científico excelente!!
- Se puede desarrollar software bastante rápido.
- Posee una licencia de código abierto.
- Una comunidad gigante desarrollando con la cual realmente se puede contar.

Un poquito de historia

- Desarrollado desde 1989
- Autor: Guido van Rossum (si lo queres seguir: @gvanrossum)



Zen de Python, por Tim Peters

- Bello es mejor que feo.
- Explícito es mejor que implícito.
- Simple es mejor que complejo.
- Complejo es mejor que complicado.
- Si la implementación es difícil de explicar, es una mala idea.

Tomado de: `https://www.python.org/dev/peps/pep-0020/`

Uso del intérprete

- Sobre la instalación de python
  (Las distribuciones de Linux generalmente incluyen python)
- Para llamar al intérprete:

```
python
```

- Para ver la versión que tienen instalada:

```
python -V o python --version
```

- Posibilidad de probar porciones de código en el modo interactivo antes de integrarlo como parte de un programa.

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

WTPC Workshop en técnicas de programación científica

Ideas básicas

- En un lenguaje estático, el nombre de la variable está ligado a
  - un tipo.
  - un objeto.
- En python puede tomar distintos valores en otro momento, incluso de un tipo diferente al que tenía previamente.

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

Una lista de los tipos de datos más comunes:

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`

Tipos de datos

Una lista de los tipos de datos más comunes:

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`

Algunas ideas para entender que es mutable e inmutable

```
>> x = 'foo'
>> y = x
>> print(x)
foo
>> y += 'bar'
>> print(x)
foo
```

```
>> def func(val):
>>     val += 'bar'
>> x = 'foo'
>> print x
foo
>> func(x)
>> print(x)
foo
```

```
>> x = [1, 2, 3]
>> y = x
>> print (x)
[1, 2, 3]
>> y += [3, 2, 1]
>> print (x)
[1, 2, 3, 3, 2, 1]
```

```
>> def func(val):
>>     val += [3, 2, 1]

>> x = [1, 2, 3]
>> print(x)
[1, 2, 3]
>> func(x)
>> print(x)
[1, 2, 3, 3, 2, 1]
```

```
>> a="ejemplo"
>> a="otro"
>> a[2]="c"
Traceback (most recent call last):
    File "<stdin>", line 1, in <module>
TypeError: 'str' object does not support item
      assignment
```

Sintaxis de Python

En C o C++: se usa ; al final de cada linea

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

En python: El nivel de indentación es significativo! Aca la ultima linea se ejecuta fuera del condicional.

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

WTPC Workshop en técnicas de programación científica

Sintaxis de Python

Control flow

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

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

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

```python
pass
```

```python
break
continue
```

WTPC Workshop en técnicas de programación científica

Cómo definir una función:

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

Algunas funciones en Python que devuelven valores:

```
>> longitud = len('La casa de la pradera')
>> print (longitud)
21
```

Sintaxis de Python ejemplo

En C la indentation es opcional:

```c
int factorial(int x)
{
if (x == 0)
return 1;
else
return x * factorial(x - 1);
}
```

En python es obligatoria:

```python
def factorial(x):
    if x == 0:
        return 1
    else:
        return x * factorial(x - 1)
```

Un ejemplo simple:

```python

def jugar(intento=1):
    respuesta = raw_input("De que color es una naranja? ")
    if respuesta != "naranja":
        if intento < 3:
            print "\nFallaste Intentalo de nuevo"
            intento += 1
            jugar(intento) # Llamada recursiva
        else:
            print "\nPerdiste"
    else:
        print "\nGanaste!"
jugar()
```

Operadores en Python

Matemáticos:

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

Booleanos:

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

Strings

Se puede usar ' o " para la definición de los string

```
a= "Soy Cecilia"
b= 'Soy de Argentina'
```

Para imprimir en la pantalla:

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

Strings: ejemplos

Un ejemplo:

```
'Yo comi %d %s hoy' %(12,'manzanas')
```

Otro ejemplo: I ate   today.format(12,'apples')

structures

- List

```
a=[1,'apple',1.2]
```

- Tuple

```
a=(1,'apple',1.2)
```

- Dict

```
a={'name':'Giovanni', 'age':42}
```

- Set

```
a={1,'apple',1.2}
```

Listas

- Son estructuras de python super útiles
- Se pueden acceder usando un índice y se puede hacer slicing

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

Para obtener el ultimo elemento de la lista:

```
a[-1]
```

- Se pueden definir por comprensión

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

Métodos útiles asociados a las listas

```
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

Métodos útiles asociados a las listas

```
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

Estas librerías:

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

Son todas open source!

Mis Motivos (y los de muchos) para usar Python

- NumPy provee funcionalidad para crear, borrar, manejar y operar en grandes arrays de datos crudos (como los de Fortran and C/C++ arrays).
- SciPy extiende NumPy con una colección de algoritmos útiles como minimimización, transformada de Fourier, regresión y otras herramientas.
- Ambos paquetes son add-on packages (no son parte de python en si. Continen código en Python y compilado con (fftpack, BLAS).
- MatPlotLib es una librería para hacer gráficos.

Numpy

NumPy es un paquete fundamental para python de programación científica. Entre otras cosas contiene:

- Potentes arrays N-dimensionales.
- Funciones sofisticadas.
- Herramientas para integración con código C/C++ y Fortran.
- Herramientas útiles de alegra lineal, transformada de fourier, y generadores de números aleatorios.

Scipy

Es un paquete que extiende la funcionalidad de Numpy con una colección substancial de algoritmos por ejemplo de minimización, cálculo y procesamiento de señales.

- Es user-friendly
- Tiene rutinas eficientes de integration and optimization. Permiten trabajar con:
  - Clustering.
  - Fourier transforms.
  - numerical integration, interpolations.
  - data I/O, LAPACK.
  - sparse matrices, linear solvers, optimization.
  - signal processing.
  - statistical functions.

matplotlib

Es una librería de python para gráficos 2D (y 3D también un poquito) que produce imágenes de alta calidad en una gran diversidad de formatos y entornos o plataformas interactivas.

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

Como usarlas?

Es necesario importar las librerias:

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

Se puede hacer de distintas maneras:

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

Yo uso:

```
import numpy as np
import scipy as sp
import matplotlib.pyplot as pp (o tambien) plt
```

Cosas sobre Numpy

Funcionalidad de NumPy :

- Funciones polinómicas.
- Cómputo estadístico.
- Generadores de números aleatoreos.
- Transformada de Fourier discreta.
- Cambio de tamaño, forma, testeo y cálculo con arrays.

Algo fundamental de Numpy: np.array

Hay 5 mecanismos generales para crear arrays:

- Conversión desde otras estructuras de Python (e.g., lists, tuples).
- Directamente como Numpy array (e.g., arange, ones, zeros, etc.).
- Leer los arrays del disco.
- Crearlos a partir de strings o datos en buffers.
- Usar las librerías especiales (e.g., random).

Numpy: ejemplos

(>>> significa input)

```
import numpy as np
>>> x = np.array([2, 3, 1, 0])
>>> print(x)
[2 3 1 0]

>>> np.arange(10)
array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9])

>>> np.arange(2, 10, dtype=np.float)
array([ 2., 3., 4., 5., 6., 7., 8., 9.])

>>> np.arange(2, 3, 0.1)
array([ 2. , 2.1, 2.2, 2.3, 2.4, 2.5, 2.6, 2.7, 2.8, 2.9])

>>> np.linspace(1., 4., 6)
array([ 1. ,  1.6,  2.2,  2.8,  3.4,  4. ])
```

Cambiando la forma del array

```
>>> a = np.zeros((5, 2))
>>> print(a)
[[ 0.  0.]
 [ 0.  0.]
 [ 0.  0.]
 [ 0.  0.]
 [ 0.  0.]]
```

reshape:

```
>>> b = a.reshape((2, 5))
>>> print(b)
[[ 0.  0.  0.  0.  0.]
 [ 0.  0.  0.  0.  0.]]
```

Trasponer y llenar:

Algunas rutinas de ejemplo:

```
>>> x = np.array([1.,2.,3.,4.])
>>> x
array([ 1.,  2.,  3.,  4.])
>>> x.T
array([ 1.,  2.,  3.,  4.])
```

```
>>> a = np.array([1, 2])
>>> a.fill(0)
>>> a
array([0, 0])
>>> a = np.empty(2)
>>> a.fill(1)
>>> a
array([ 1.,  1.])
```

Operaciones elementales con Numpy:

```
x = np.array([[1,2],[3,4]], dtype=np.float64)
y = np.array([[5,6],[7,8]], dtype=np.float64)

# Elementwise sum; both produce the array

>>>print(x + y)
>>>print(np.add(x, y))
 [[ 6.0  8.0]
 [10.0 12.0]]

# Elementwise difference; both produce the array

>>>print(x - y)
>>>print(np.subtract(x, y))
 [[-4.0 -4.0]
 [-4.0 -4.0]]
```

Operaciones elementales con Numpy:

```
# Elementwise product; both produce the array
>>>print(x * y)
>>>print(np.multiply(x, y))
[[ 5.0 12.0]
[21.0 32.0]]

# Elementwise division; both produce the array
>>>print(x / y)
>>>print(np.divide(x, y))
[[ 0.2         0.33333333]
[ 0.42857143  0.5       ]]

# Elementwise square root; produces the array
>>>print(np.sqrt(x))
[[ 1.          1.41421356]
[ 1.73205081  2.        ]]
```

Más operaciones con arrays

```
x = np.array([[1,2],[3,4]])
y = np.array([[5,6],[7,8]])

v = np.array([9,10])
w = np.array([11, 12])

# Inner product of vectors; both produce 219
print(v.dot(w))
print(np.dot(v, w))

# Matrix / vector product; both produce the rank 1 array [29 67]
print(x.dot(v))
print(np.dot(x, v))

# Matrix / matrix product; both produce the rank 2 array

print(x.dot(y))
print(np.dot(x, y))
```

Workshop en técnicas de programación científica

Sobre creación de vectores

```
a = np.zeros((2,2))   # Create an array of all zeros
print(a)              # Prints "[[ 0.  0.]
                      #          [ 0.  0.]]"
b = np.ones((1,2))    # Create an array of all ones
print(b)              # Prints "[[ 1.  1.]]"
c = np.full((2,2), 7) # Create a constant array
print(c)              # Prints "[[ 7.  7.]
                      #          [ 7.  7.]]"
d = np.eye(2)         # Create a 2x2 identity matrix
print(d)              # Prints "[[ 1.  0.]
                      #          [ 0.  1.]]"
e = np.random.random((2,2))  # Create an array filled with random values
print(e)                     # Might print "[[ 0.91940167  0.08143941]
                             #               [ 0.68744134  0.87236687]]"
```

Numpy Chart:

Python For Data Science *Cheat Sheet*

NumPy Basics

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

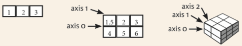
NumPy

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

Scipy Basics:

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.

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

Workshop en técnicas de programación científica

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.

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))
```

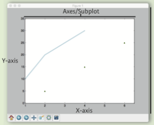
Axes

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,                          Colormapped or RGB arrays
                   cmap='gist_earth',
                   interpolation='nearest',
                   vmin=-2,
                   vmax=2)
```

```
>>> 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**

Workshop en técnicas de programación científica

Como crear un dataset y salvarlo en un .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)
```

Como crear un dataset y salvarlo en un .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)
```

Como crear un dataset y salvarlo en un .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)
```

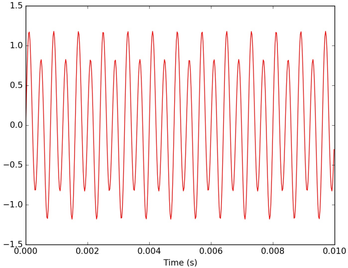
Como crear un dataset y salvarlo en un .txt

```
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)
```

Workshop en técnicas de programación científica

Obtenemos:

Una tabla en txt y una figura:



# 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 ...

Como abrir .txt or cvs

Un modo super facil es usar 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]
```

Como hacer un ajuste y graficarlo:

```
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()
```

Como hacer un ajuste y graficarlo:

```
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

Como hacer un ajuste y graficarlo:

```
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()
```

Como hacer un ajuste y graficarlo:

```
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

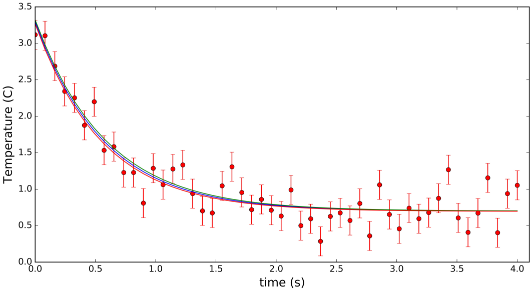
Como hacer el ajuste

We obtain the fit parameters:

```
[ 2.595658    1.74438726  0.69809511]
```

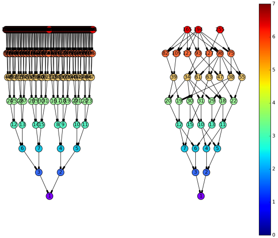
The covariance matrix:

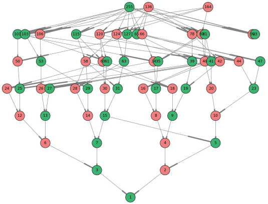
```
[[ 0.02506636  0.01490486 -0.00068609]
 [ 0.01490486  0.04178044  0.00641246]
 [-0.00068609  0.00641246  0.00257799]]
```



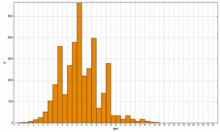
Otros tipos de gráficos

```python
import networkx as nx
import pygraphviz
from graphviz import *
```



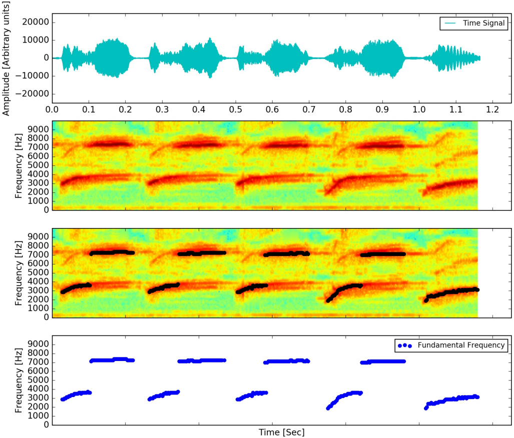


```python
import matplotlib.pyplot as pp
pp.hist([vector,bins=bins)
```



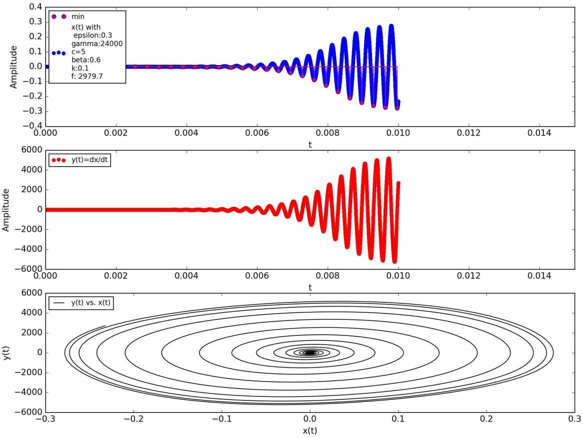
Aplicando Transformada de fourier y un sonograma

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



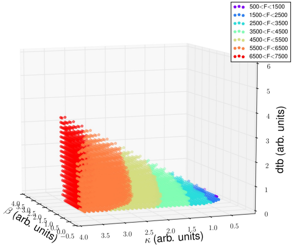
Ejemplo de integración numérica

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

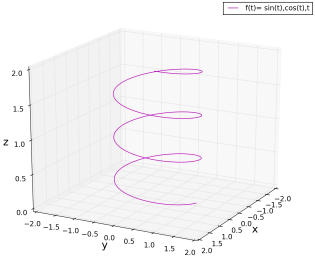


Se pueden hacer gráficos 3d y también videos

El ejemplo anterior estudiando una variable extra:



Curva paramétrica:



What else do we need todo with data?

What can we do with data?

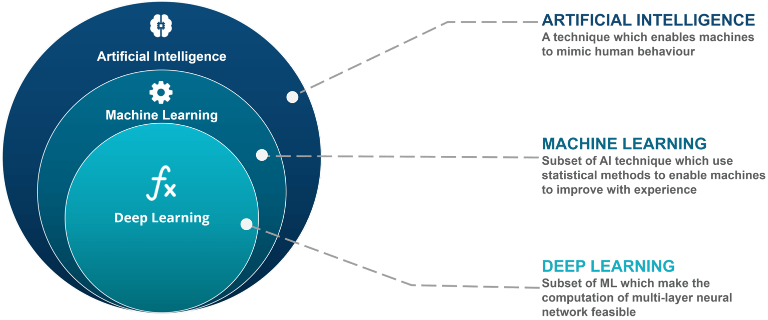
We are interested in:

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

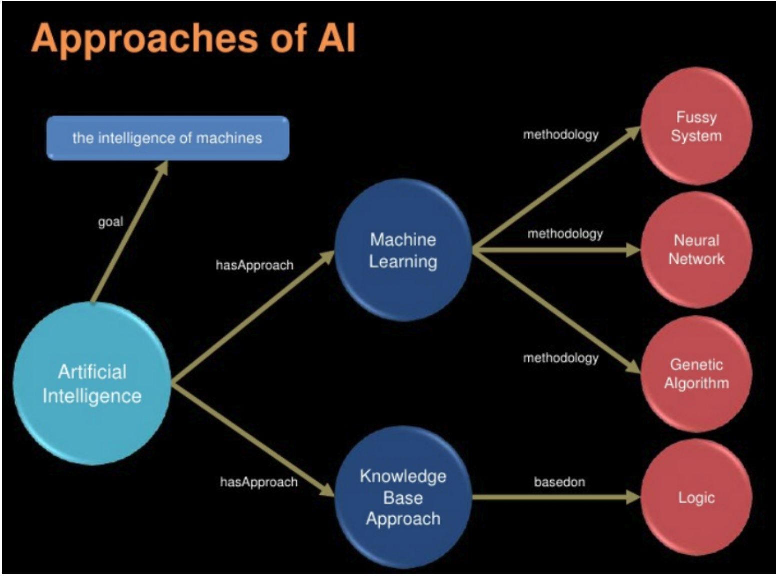
Which are the available algorithms?

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.**

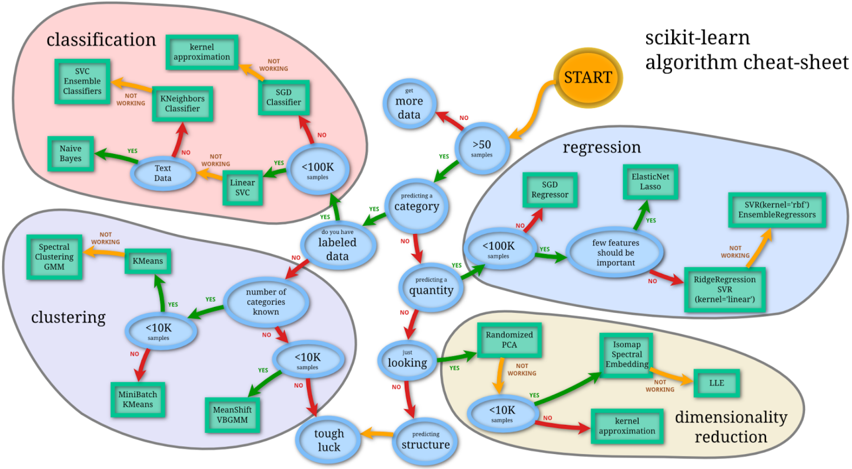
What about machine learning?



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

We can use Scikit learn



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)
```

Estimator score method

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

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_)
```

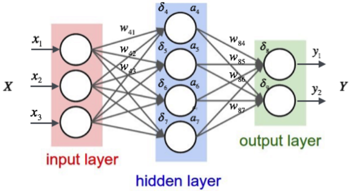
DataCamp
Learn Python for Data Science **Interactively**

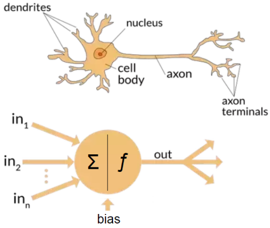
Workshop en técnicas de programación científica

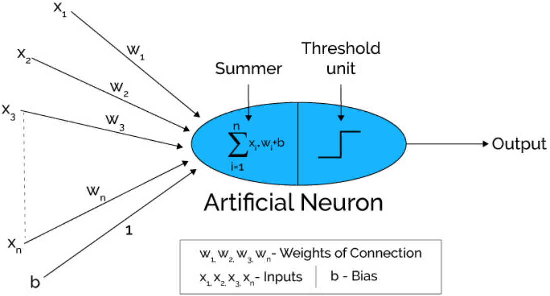
Scikit-learn Basics:

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

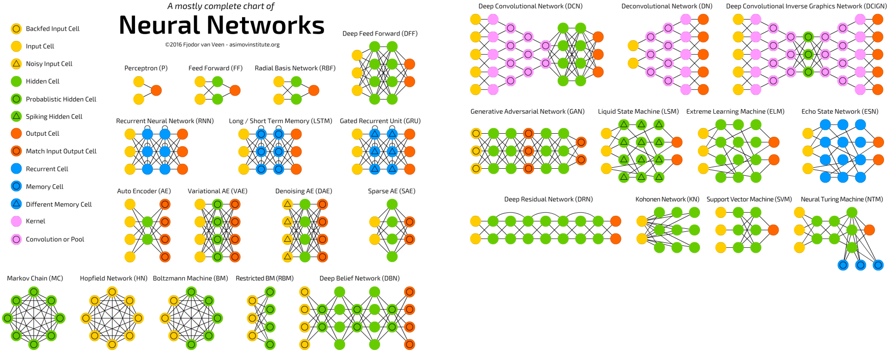
Neural Networks:







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]
```

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'))
```

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)
```

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

WTPC
Workshop en técnicas de programación científica

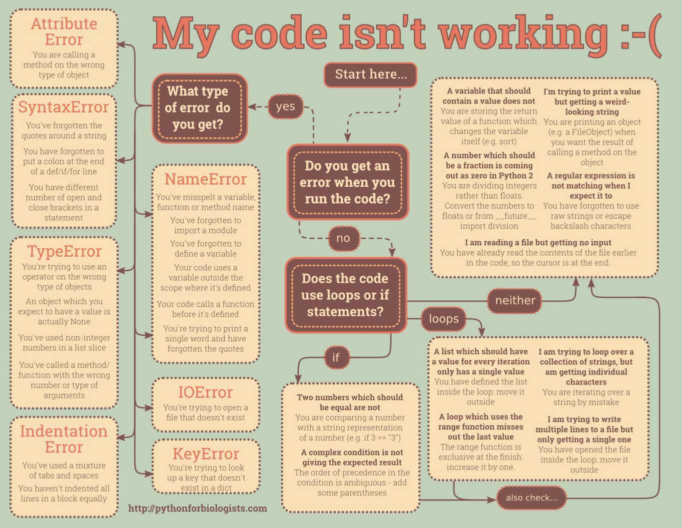
Conclusiones

- Les dejamos una colección infinitamente grande de herramientas y recursos recombinables y reutilizables para poner en práctica.
- El salto es mas pequeño de lo que parece.
- Posibilidades de reutilizar y adaptar código a nuestras necesidades para crear soluciones prácticas.
- Una guia de estilo en: `https://google.github.io/styleguide/pyguide.html`
- ejemplos de Matplotlib: `http://matplotlib.org/gallery.html`

Más librerías para análisis de datos:

- Scikit learn:
  `http://scikit-learn.org`
- Pandas
  `http://pandas.pydata.org/`

Otras librerías para análisis de datos:



Algo más

Invitación: http://www.python.org.ar/wiki/PyCamp/2018