WTPC
Workshop en técnicas de programación científica

1 al 11 de Marzo @Tucumán
Argentina

Introducción a la programación en python

Cecilia Jarne

cecilia.jarne@unq.edu.ar

Ideas básicas de Python

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

Es un lenguaje multiparadigma:

- soporta orientación a objetos.
- programación imperativa.
- programación funcional.

https://www.python.org

Ideas básicas de Python

¿Por qué aprender python si yo ya se programar en ***?**

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

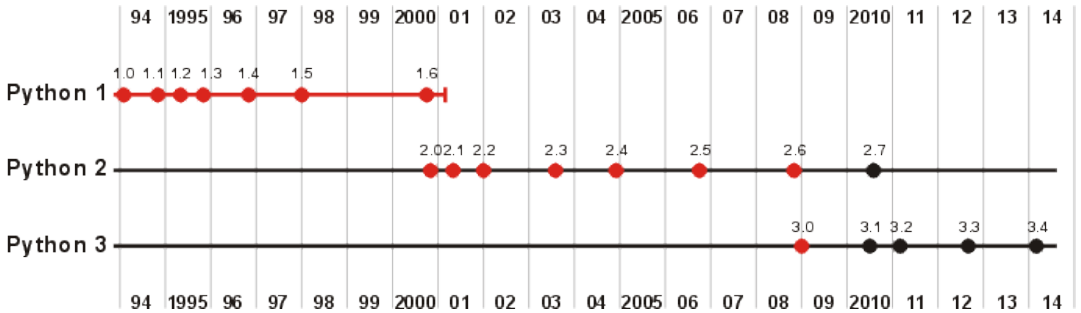
Ideas básicas de Python

Algo breve de historia...

- Desarrollado desde 1989 por Guido van Rossum.

- Versión Python 2.0: Octubre 2000 (now: 2.7.8)

- Versión Python 3.0: Diciembre 2008 (now 3.5.1)



Ideas básicas de Python

Python tiene alguna filosofía detrás, bien descripta por Tim Peters

-Bello es mejor que feo.
-Explícito es mejor que implícito.
-Simple es mejor que complejo.
-Complejo es mejor que complicado.
-Plano es mejor que anidado.
-Disperso es mejor que denso.
-La legibilidad cuenta.
-Los casos especiales no son tan especiales como para quebrantar las reglas.
-Lo práctico gana a lo puro.
-Los errores nunca deberían dejarse pasar silenciosamente.
-A menos que hayan sido silenciados explícitamente.
-Frente a la ambigüedad, rechaza la tentación de adivinar.
-Debería haber una -y preferiblemente sólo una- manera obvia de hacerlo.
-Aunque esa manera puede no ser obvia al principio a menos que usted sea holandés.
-Ahora es mejor que nunca.
-Aunque nunca es a menudo mejor que ya mismo.
-Si la implementación es difícil de explicar, es una mala idea.
-Si la implementación es fácil de explicar, puede que sea una buena idea.
-Los namespaces son una gran idea ¡Hagamos más de esas cosas!

Ideas básicas de Python

El intérprete estándar incluye un modo interactivo (intérprete de comandos):

-Las expresiones pueden ser introducidas una a una para ver el resultado de su evaluación inmediatamente.

- 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]
```

Ideas básicas de Python

Variables:

- Se definen de forma dinámica: - No se tiene que especificar cuál es su tipo de antemano.
- Puede tomar distintos valores en otro momento, incluso de un tipo diferente al que tenía previamente.
- Se usa el símbolo = para asignar valores.

```
x = 1
x = "texto" # Esto es posible porque
los tipos son asignados dinámicamente
```

Ideas básicas de Python

Tipos de datos:

Tipo	Clase	Notas	Ejemplo
`str`	Cadena	Inmutable	`'Cadena'`
`unicode`	Cadena	Versión Unicode de `str`	`u'Cadena'`
`list`	Secuencia	Mutable, puede contener objetos de diversos tipos	`[4.0, 'Cadena', True]`
`tuple`	Secuencia	Inmutable, puede contener objetos de diversos tipos	`(4.0, 'Cadena', True)`
`set`	Conjunto	Mutable, sin orden, no contiene duplicados	`set([4.0, 'Cadena', True])`
`frozenset`	Conjunto	Inmutable, sin orden, no contiene duplicados	`frozenset([4.0, 'Cadena', True])`
`dict`	Mapping	Grupo de pares clave:valor	`{'key1': 1.0, 'key2': False}`
`int`	Número entero	Precisión fija, convertido en *long* en caso de overflow.	`42`
`long`	Número entero	Precisión arbitraria	`42L` ó `456966786151987643L`
`float`	Número decimal	Coma flotante de doble precisión	`3.1415927`
`complex`	Número complejo	Parte real y parte imaginaria *j*.	`(4.5 + 3j)`
`bool`	Booleano	Valor booleano verdadero o falso	`True o False`

Ideas básicas de Python

La Sintaxis:

- Los espacios en blanco importan!!

C/C++

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

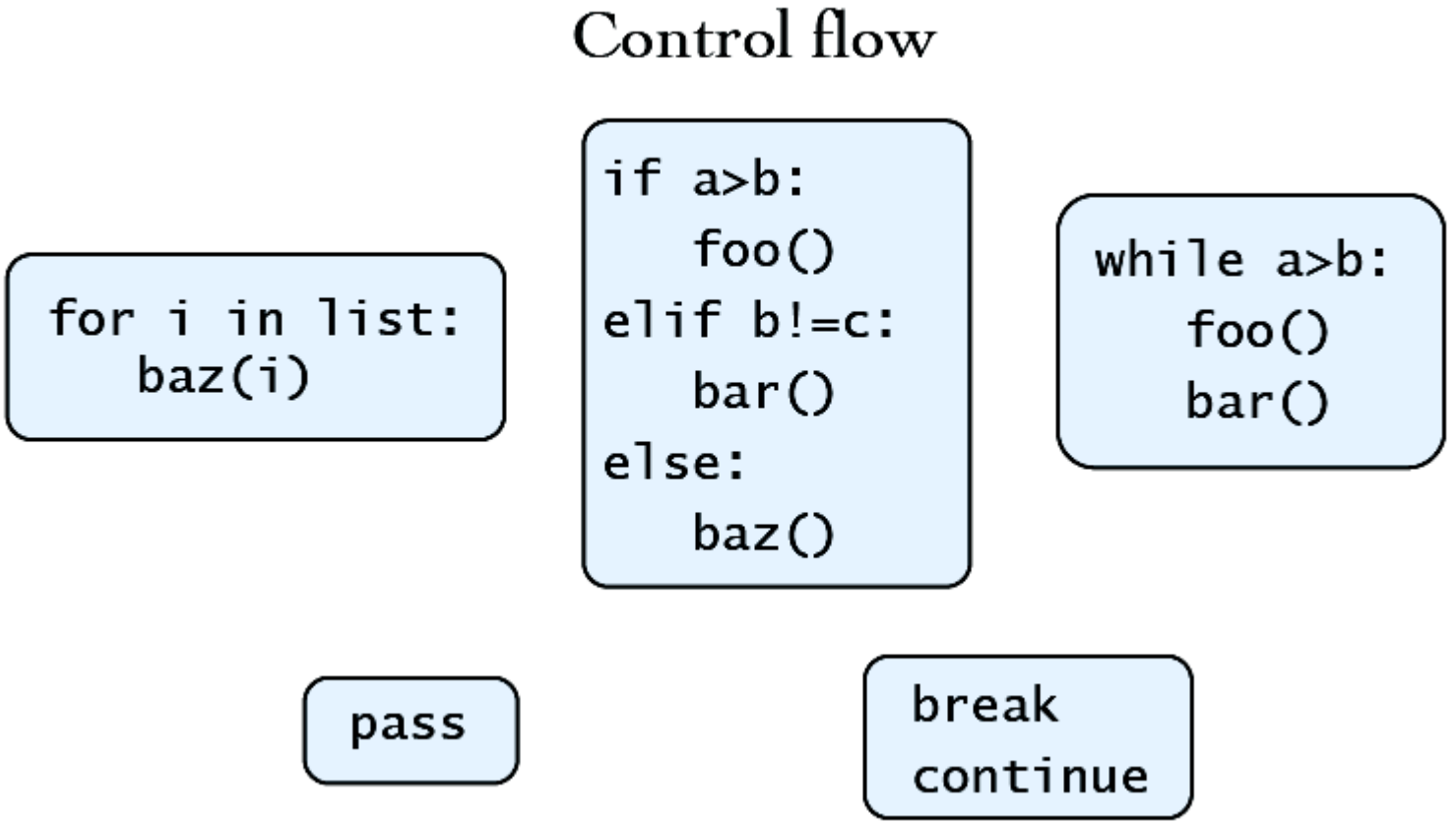
Python

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

Ideas básicas de Python

La Sintaxis:

-Posee varios comandos para el control de flujo:



Ideas básicas de Python

La Sintaxis: un ejemplo con la función factorial

(6!=6*5*4*3*2*1)

Función factorial en C (indentación opcional)

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

Función factorial en Python (indentación obligatoria)

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

Ideas básicas de Python

La Sintaxis:

Function definition

```
def stuff(a,b,c):
    a = 3*b
    return a+b-c
```

-Las funciones pueden ser pasadas como valores!

```
def timesN(N):
    def helper(x):
        return N*x
    return helper

times6 = timesN(6)
a = times6(7)
```

Las expresiones:

- Cuidado con la división en python 2!!

```
5/3 == 1
```

```
5./3. == 1.66666666667
```

- Los operadores booleanos se escriben explícitamente

```
and      or      not
   True   False
```

Ideas básicas de Python

Strings:

Se puede usar ' o “

```
a = ”Fred’s house”
b = ’He said “Hello!” to me’
```

Verbatim texts in triple quotes

```
”””can go
over several lines
like this
”””
```

Otra construcción de strings, no muy conocida: (sirve para partirlo en varias líneas)

```python
>>> s = ("hola, me llamo"
         "Pablo y estoy muy"
         "contento de verlos")
>>> print s
hola, me llamo Pablo y estoy muy contento de verlos
```

Formato para los Strings:

```
"I ate %d %s today" % (12,"apples")   (like printf())

"I ate {} {} today".format(12,"apples")
```

La sintaxis utilizando diccionarios:

```python
>>> s = "Me llamo {nombre} y tengo {edad} años.".format(nombre="Mafalda", edad=4)
>>> print s
Me llamo Mafalda y tengo 4 años.
```

Ideas básicas de Python

Colecciones:

list, tuple

```
[3, 1, 'foo', 12.]      mutable

(3, 1, 'foo')           immutable
```

```
a[0]   a[-1]   a[2:5]   a[2:10:2]      index / slice access

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

dict, set

```
d={'name':'Monty', 'age':42}
   d['name']    d['age']

{3, 1, 'foo', 12.}  unique elements, union, intersection, etc.
```

Ideas básicas de Python

list.append(x)

Agrega un ítem al final de la lista; equivale a a[len(a):] = [x].

list.extend(L)

Extiende la lista agregándole todos los ítems de la lista dada; equivale a a[len(a):] = L.

list.insert(i, x)

Inserta un ítem en una posición dada. El primer argumento es el índice del ítem delante del cual se insertará, por lo tanto a.insert(0, x) inserta al principio de la lista, y a.insert(len(a), x) equivale a a.append(x).

list.remove(x)

Quita el primer ítem de la lista cuyo valor sea x. Es un error si no existe tal ítem.

list.pop([i])

Quita el ítem en la posición dada de la lista, y lo devuelve. Si no se especifica un índice, a.pop() quita y devuelve el último ítem de la lista. (Los corchetes que encierran a i en la firma del método denotan que el parámetro es opcional, no que deberías escribir corchetes en esa posición. Verás esta notación con frecuencia en la Referencia de la Biblioteca de Python.)

list.index(x)

Devuelve el índice en la lista del primer ítem cuyo valor sea x. Es un error si no existe tal ítem.

list.count(x)

Devuelve el número de veces que x aparece en la lista.

list.sort()

Ordena los ítems de la lista, in situ.

list.reverse()

Invierte los elementos de la lista, in situ.

Ideas básicas de Python

Si aun la motivación no alcanza, aquí la verdadera razón de porqué me convertí a python:

NumPy:
http://www.numpy.org/

SciPy:
http://www.scipy.org/

MatPlotLib:
http://matplotlib.org/

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.

Ideas básicas de Python

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.

Ideas básicas de Python

MatPlotlib:

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

-Puede ser usada en scrips de python o en entorno interactivo al estilo de MATLAB®* or Mathematica® y también e aplicaciones web.

-Por supuesto también es open source!!!

Ideas básicas de Python

Para poder utilizar estas librerías hay que importarlas:

```
import numpy

import numpy as np

from numpy import *
```

```
import scipy

import scipy as sp

from scipy import *
```

Ideas básicas de Python

Una de las unidades mas importantes de numpy es el array:

```
>>> a = np.array([1, 4, 5, 8], float32)

>>> a
array([1., 4., 5., 8.])


>>> type(a)
<type 'numpy.ndarray'>
>>> a[:2]
Array([1., 4.])

>>> a[3]
8.0
```

Ideas básicas de Python

Con los array se pueden realizar diversas operaciones y pueden ser reordenados, o reformateados

```
>>> a = np.array(range(10), float32)
>>> a
array([ 0., 1., 2., 3., 4., 5., 6., 7., 8., 9.])

>>> a.reshape((5, 2))
>>> a
array([ 0., 1., 2., 3., 4., 5., 6., 7., 8., 9.])

>>> b = a.reshape((5,2))
array([[ 0., 1.],
       [ 2., 3.],
       [ 4., 5.],
       [ 6., 7.],
       [ 8., 9.]])

>>> b.shape
(5, 2)
```

b points at same data in memory, new "view"

Ideas básicas de Python

Llenados, transpuestos... etc

```
>>> a = np.array([1, 2, 3],float)

>>> a
array([1.0, 2.0, 3.0])

>>> a.fill(0)
array([0.0, 0.0, 0.0])

>>> a = np.array(range(6), float).reshape((2, 3))
>>> a
array([[ 0., 1., 2.],
       [ 3., 4., 5.]])

>>> a.transpose()
array([[ 0., 3.],
       [ 1., 4.],
       [ 2., 5.]])
```

Ideas básicas de Python

Y operados elemento a elemento:

```
>>> a = np.array([1,2,3], float)
>>> b = np.array([5,2,6], float)
>>> a + b
array([6., 4., 9.])
>>> a - b
array([-4., 0., -3.])
>>> a * b
array([5., 4., 18.])
>>> b / a
array([5., 1., 2.])
>>> a % b
array([1., 0., 3.])
>>> b**a
array([5., 4., 216.])
```

Ideas básicas de Python

También es fácil definir y operar con matrices:

```
>>> b = np.linalg.inv(a)
>>> b
array([[ 0.14814815, 0.07407407, -0.25925926],
       [ 0.2037037 , -0.14814815, 0.51851852],
       [-0.27777778, 0.11111111, 0.11111111]])
>>> np.dot(a, b)
array([[ 1.00000000e+00, 5.55111512e-17, 2.22044605e-16],
       [ 0.00000000e+00, 1.00000000e+00, 5.55111512e-16],
       [ 1.11022302e-16, 0.00000000e+00, 1.00000000e+00]])

>>> a = np.array([[1, 3, 4], [5, 2, 3]], float)
>>> U, s, Vh = np.linalg.svd(a)
>>> U
array([[-0.6113829 , -0.79133492],
       [-0.79133492, 0.6113829 ]])
>>> s
array([ 7.46791327, 2.86884495])
>>> Vh
array([[-0.61169129, -0.45753324, -0.64536587],
       [ 0.78971838, -0.40129005, -0.464.....
```

Ideas básicas de Python

Ejemplo práctico de almacenamiento de operar, graficar y alamacenar datos:

```
import numpy as np
import scipy
import matplotlib.pyplot as pp
from pylab import *

f_out_max = open('tabla.txt', 'w')

x=arange(441)
Sin1 = 1*sin(2*pi*(25/441.0)*x)
Sin2 = 0.25*sin(2*pi*((25./2)/441.0)*x)
Sig  = sin1+sin2

print 'x: ',x
print 'sig:', sig
Vec = np.c_[x,sig]

print 'vec: ',vec
np.savetxt(f_out_max,vec,fmt='%f',delimiter='\t',header="x #f(x)")
f_out_max.close()

pp.figure()
pp.plot(x*1./44100.,sig, color='r',label='Time signal VS')
pp.ylabel('Amplitude')
pp.grid(True)
pp.xlabel('Time (s)')
pp.show()
```

Importo las librerías

Ideas básicas de Python

Ejemplo práctico de almacenamiento de operar, graficar y alamacenar datos:

```
import numpy as np
import scipy
import matplotlib.pyplot as pp
from pylab import *

f_out_max = open('tabla.txt', 'w')

x=arange(441)
Sin1 = 1*sin(2*pi*(25/441.0)*x)
Sin2 = 0.25*sin(2*pi*((25./2)/441.0)*x)
Sig  = sin1+sin2

print 'x: ',x
print 'sig:', sig
Vec = np.c_[x,sig]

print 'vec: ',vec
np.savetxt(f_out_max,vec,fmt='%f',delimiter='\t',header="x #f(x)")
f_out_max.close()

pp.figure()
pp.plot(x*1./44100.,sig, color='r',label='Time signal VS')
pp.ylabel('Amplitude')
pp.grid(True)
pp.xlabel('Time (s)')
pp.show()
```

Importo las librerías

Archivo de salida+formato

Ideas básicas de Python

Ejemplo práctico de almacenamiento de operar, graficar y alamacenar datos:

```
import numpy as np
import scipy
import matplotlib.pyplot as pp
from pylab import *

f_out_max = open('tabla.txt', 'w')

x=arange(441)
Sin1 = 1*sin(2*pi*(25/441.0)*x)
Sin2 = 0.25*sin(2*pi*((25./2)/441.0)*x)
Sig  = sin1+sin2

print 'x: ',x
print 'sig:', sig
Vec = np.c_[x,sig]

print 'vec: ',vec
np.savetxt(f_out_max,vec,fmt='%f',delimiter='\t',header="x #f(x)")
f_out_max.close()

pp.figure()
pp.plot(x*1./44100.,sig, color='r',label='Time signal VS')
pp.ylabel('Amplitude')
pp.grid(True)
pp.xlabel('Time (s)')
pp.show()
```

Importo las librerías

Archivo de salida+formato

Genero un set de datos

Ideas básicas de Python

Ejemplo práctico de almacenamiento de operar, graficar y alamacenar datos:

```
import numpy as np
import scipy
import matplotlib.pyplot as pp
from pylab import *

f_out_max = open('tabla.txt', 'w')

x=arange(441)
Sin1 = 1*sin(2*pi*(25/441.0)*x)
Sin2 = 0.25*sin(2*pi*((25./2)/441.0)*x)
Sig  = sin1+sin2

print 'x: ',x
print 'sig:', sig
Vec = np.c_[x,sig]

print 'vec: ',vec
np.savetxt(f_out_max,vec,fmt='%f',delimiter='\t',header="x #f(x)")
f_out_max.close()

pp.figure()
pp.plot(x*1./44100.,sig, color='r',label='Time signal VS')
pp.ylabel('Amplitude')
pp.grid(True)
pp.xlabel('Time (s)')
pp.show()
```

Importo las librerías

Archivo de salida+formato

Genero un set de datos

Reordeno mi set

Ideas básicas de Python

Ejemplo práctico de almacenamiento de operar, graficar y alamacenar datos:

```
import numpy as np
import scipy
import matplotlib.pyplot as pp
from pylab import *

f_out_max = open('tabla.txt', 'w')

x=arange(441)
Sin1 = 1*sin(2*pi*(25/441.0)*x)
Sin2 = 0.25*sin(2*pi*((25./2)/441.0)*x)
Sig  = sin1+sin2

print 'x: ',x
print 'sig:', sig
Vec = np.c_[x,sig]

print 'vec: ',vec
np.savetxt(f_out_max,vec,fmt='%f',delimiter='\t',header="x #f(x)")
f_out_max.close()

pp.figure()
pp.plot(x*1./44100.,sig, color='r',label='Time signal VS')
pp.ylabel('Amplitude')
pp.grid(True)
pp.xlabel('Time (s)')
pp.show()
```

Importo las librerías

Archivo de salida+formato

Genero un set de datos

Reordeno mi set

Guardo mi set en el outfile

Ideas básicas de Python

Ejemplo práctico de almacenamiento de operar, graficar y alamacenar datos:

```
import numpy as np
import scipy
import matplotlib.pyplot as pp
from pylab import *

f_out_max = open('tabla.txt', 'w')

x=arange(441)
Sin1 = 1*sin(2*pi*(25/441.0)*x)
Sin2 = 0.25*sin(2*pi*((25./2)/441.0)*x)
Sig  = sin1+sin2

print 'x: ',x
print 'sig:', sig
Vec = np.c_[x,sig]

print 'vec: ',vec
np.savetxt(f_out_max,vec,fmt='%f',delimiter='\t',header="x #f(x)")
f_out_max.close()

pp.figure()
pp.plot(x*1./44100.,sig, color='r',label='Time signal VS')
pp.ylabel('Amplitude')
pp.grid(True)
pp.xlabel('Time (s)')
pp.show()
```

Importo las librerías

Archivo de salida+formato

Genero un set de datos

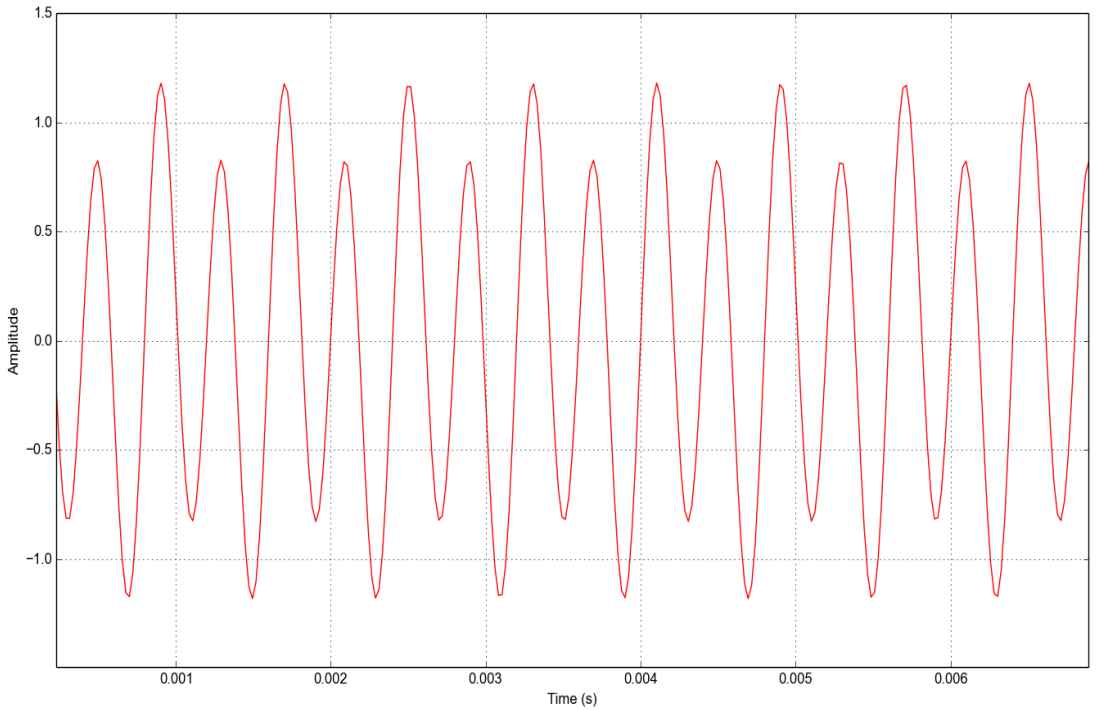
Reordeno mi set

Guardo mi set en el outfile

Grafico mis datos

Ideas básicas de Python

Python scrip.py y obtenemos:



# 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
.....	

Ideas básicas de Python

Como abro un archivo de texto con cierto formato?

```
# syllable
2.936735
0.005351
0.001111
0.001361
0.001
```

```
Content =[]

input_file=open("plots/all.txt")

for line in input_file:
    for numstr in line.split(","):
        if numstr:
            try:
                numFl = float(numstr)
          content.append(numFl)
                print(numFl)
            except ValueError as e:
                print(e)
input_file.close()
```

Genero una lista vacia

Abro el archivo

Loop sobre las lineas del archivo

cierro archivo

Ideas básicas de Python

Como hacer un ajuste con python?

Importo las librerías

```
import numpy as np
import matplotlib.pyplot as plt
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)
print fitParams
print fitCovariances

plt.ylabel('Temperature (C)', fontsize = 16)
plt.xlabel('time (s)', fontsize = 16)
plt.xlim(0,4.1)
# plot the data as red circles with errorbars in the vertical direction
plt.errorbar(t, noisy, fmt = 'ro', yerr = 0.2)
# now plot the best fit curve and also +- 3 sigma curves
# the square root of the diagonal covariance matrix element
# is the uncertianty on the corresponding fit parameter.
sigma = [fitCovariances[0,0], fitCovariances[1,1], fitCovariances[2,2] ]
plt.plot(t, fitFunc(t, fitParams[0], fitParams[1], fitParams[2]),\
         t, fitFunc(t, fitParams[0] + sigma[0], fitParams[1] - sigma[1], fitParams[2] + sigma[2]),\
         t, fitFunc(t, fitParams[0] - sigma[0], fitParams[1] + sigma[1], fitParams[2] - sigma[2])\
         )
plt.show()
# save plot to a file
savefig('dataFitted.pdf', bbox_inches=0, dpi=600)
```

Ideas básicas de Python

Como hacer un ajuste con python?

```
import numpy as np
import matplotlib.pyplot as plt
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)
print fitParams
print fitCovariances

plt.ylabel('Temperature (C)', fontsize = 16)
plt.xlabel('time (s)', fontsize = 16)
plt.xlim(0,4.1)
# plot the data as red circles with errorbars in the vertical direction
plt.errorbar(t, noisy, fmt = 'ro', yerr = 0.2)
# now plot the best fit curve and also +- 3 sigma curves
# the square root of the diagonal covariance matrix element
# is the uncertianty on the corresponding fit parameter.
sigma = [fitCovariances[0,0], fitCovariances[1,1], fitCovariances[2,2] ]
plt.plot(t, fitFunc(t, fitParams[0], fitParams[1], fitParams[2]),\
         t, fitFunc(t, fitParams[0] + sigma[0], fitParams[1] - sigma[1], fitParams[2] + sigma[2]),\
         t, fitFunc(t, fitParams[0] - sigma[0], fitParams[1] + sigma[1], fitParams[2] - sigma[2])\
         )
plt.show()
# save plot to a file
savefig('dataFitted.pdf', bbox_inches=0, dpi=600)
```

Importo las librerías

Defino la función

Ideas básicas de Python

Como hacer un ajuste con python?

```
import numpy as np
import matplotlib.pyplot as plt
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)
print fitParams
print fitCovariances

plt.ylabel('Temperature (C)', fontsize = 16)
plt.xlabel('time (s)', fontsize = 16)
plt.xlim(0,4.1)
# plot the data as red circles with errorbars in the vertical direction
plt.errorbar(t, noisy, fmt = 'ro', yerr = 0.2)
# now plot the best fit curve and also +- 3 sigma curves
# the square root of the diagonal covariance matrix element
# is the uncertianty on the corresponding fit parameter.
sigma = [fitCovariances[0,0], fitCovariances[1,1], fitCovariances[2,2] ]
plt.plot(t, fitFunc(t, fitParams[0], fitParams[1], fitParams[2]),\
         t, fitFunc(t, fitParams[0] + sigma[0], fitParams[1] - sigma[1], fitParams[2] + sigma[2]),\
         t, fitFunc(t, fitParams[0] - sigma[0], fitParams[1] + sigma[1], fitParams[2] - sigma[2])\
         )
plt.show()
# save plot to a file
savefig('dataFitted.pdf', bbox_inches=0, dpi=600)
```

Importo las librerías

Defino la función

Genero un set d datos

Ideas básicas de Python

Como hacer un ajuste con python?

```
import numpy as np
import matplotlib.pyplot as plt
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)
print fitParams
print fitCovariances

plt.ylabel('Temperature (C)', fontsize = 16)
plt.xlabel('time (s)', fontsize = 16)
plt.xlim(0,4.1)
# plot the data as red circles with errorbars in the vertical direction
plt.errorbar(t, noisy, fmt = 'ro', yerr = 0.2)
# now plot the best fit curve and also +- 3 sigma curves
# the square root of the diagonal covariance matrix element
# is the uncertianty on the corresponding fit parameter.
sigma = [fitCovariances[0,0], fitCovariances[1,1], fitCovariances[2,2] ]
plt.plot(t, fitFunc(t, fitParams[0], fitParams[1], fitParams[2]),\
         t, fitFunc(t, fitParams[0] + sigma[0], fitParams[1] - sigma[1], fitParams[2] + sigma[2]),\
         t, fitFunc(t, fitParams[0] - sigma[0], fitParams[1] + sigma[1], fitParams[2] - sigma[2])\
         )
plt.show()
# save plot to a file
savefig('dataFitted.pdf', bbox_inches=0, dpi=600)
```

Importo las librerías

Defino la función

Genero un set d datos

Importo las librerías

Ideas básicas de Python

Como hacer un ajuste con python?

```
import numpy as np
import matplotlib.pyplot as plt
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)
print fitParams
print fitCovariances

plt.ylabel('Temperature (C)', fontsize = 16)
plt.xlabel('time (s)', fontsize = 16)
plt.xlim(0,4.1)
# plot the data as red circles with errorbars in the vertical direction
plt.errorbar(t, noisy, fmt = 'ro', yerr = 0.2)
# now plot the best fit curve and also +- 3 sigma curves
# the square root of the diagonal covariance matrix element
# is the uncertianty on the corresponding fit parameter.
sigma = [fitCovariances[0,0], fitCovariances[1,1], fitCovariances[2,2] ]
plt.plot(t, fitFunc(t, fitParams[0], fitParams[1], fitParams[2]),\
         t, fitFunc(t, fitParams[0] + sigma[0], fitParams[1] - sigma[1], fitParams[2] + sigma[2]),\
         t, fitFunc(t, fitParams[0] - sigma[0], fitParams[1] + sigma[1], fitParams[2] - sigma[2])\
         )
plt.show()
# save plot to a file
savefig('dataFitted.pdf', bbox_inches=0, dpi=600)
```

Importo las librerías

Defino la función

Genero un set d datos

Importo las librerías

Ploteo elegantemente

Ideas básicas de Python

Como hacer un ajuste con python?

```
import numpy as np
import matplotlib.pyplot as plt
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)
print fitParams
print fitCovariances

plt.ylabel('Temperature (C)', fontsize = 16)
plt.xlabel('time (s)', fontsize = 16)
plt.xlim(0,4.1)
# plot the data as red circles with errorbars in the vertical direction
plt.errorbar(t, noisy, fmt = 'ro', yerr = 0.2)
# now plot the best fit curve and also +- 3 sigma curves
# the square root of the diagonal covariance matrix element
# is the uncertianty on the corresponding fit parameter.
sigma = [fitCovariances[0,0], fitCovariances[1,1], fitCovariances[2,2] ]
plt.plot(t, fitFunc(t, fitParams[0], fitParams[1], fitParams[2]),\
         t, fitFunc(t, fitParams[0] + sigma[0], fitParams[1] - sigma[1], fitParams[2] + sigma[2]),\
         t, fitFunc(t, fitParams[0] - sigma[0], fitParams[1] + sigma[1], fitParams[2] - sigma[2])\
         )
plt.show()
# save plot to a file
savefig('dataFitted.pdf', bbox_inches=0, dpi=600)
```

Importo las librerías

Defino la función

Genero un set d datos

Importo las librerías

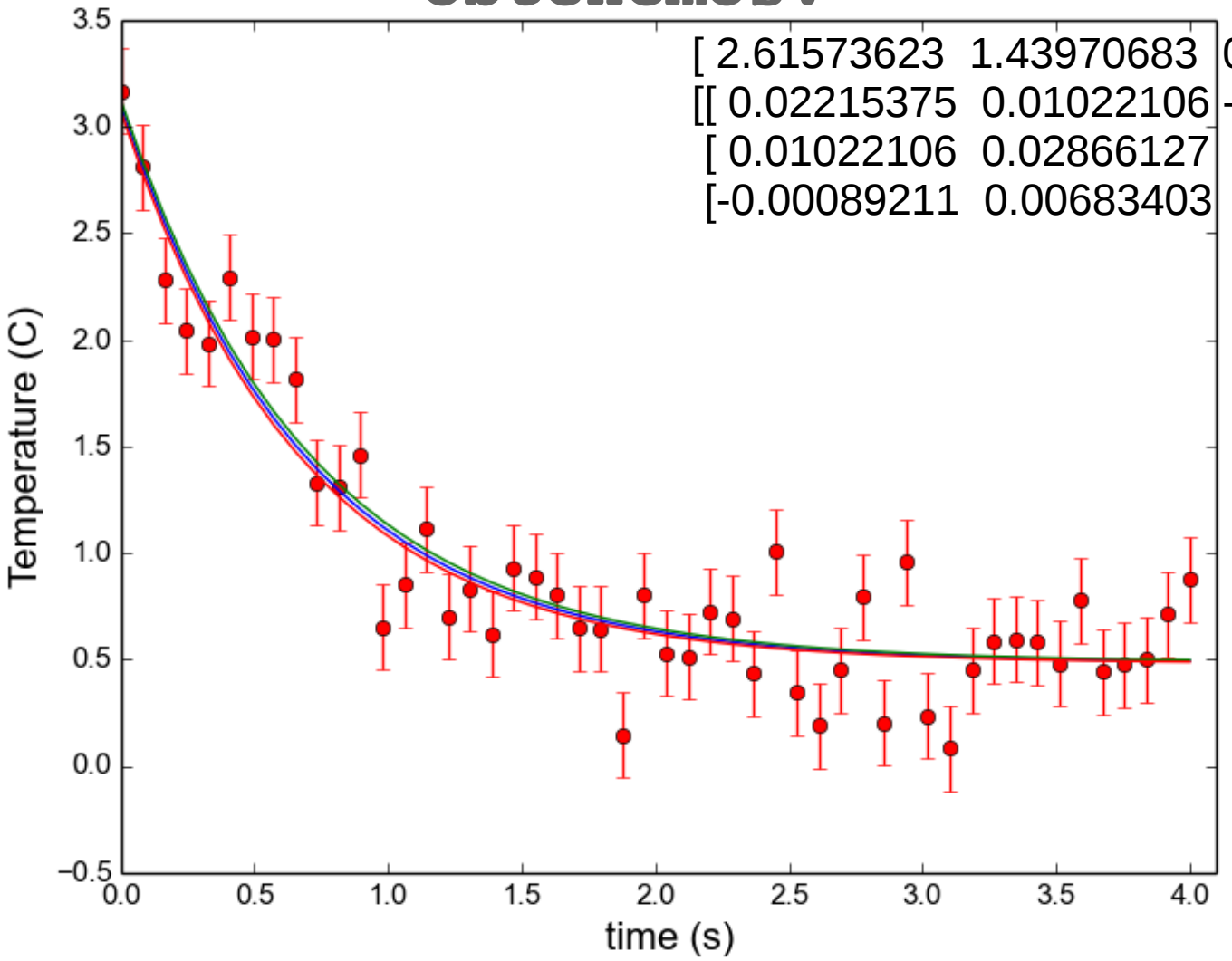
Ploteo elegantemente

Salvo mi plot

Ideas básicas de Python

Python scrip.py y obtenemos:

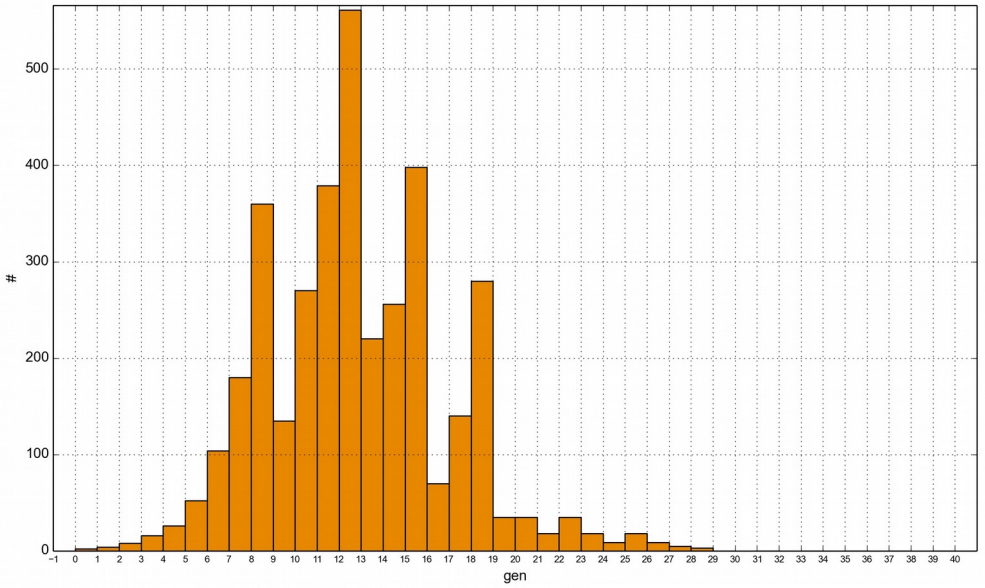
```
[ 2.61573623  1.43970683  0.48431117]
[[ 0.02215375  0.01022106 -0.00089211]
 [ 0.01022106  0.02866127  0.00683403]
 [-0.00089211  0.00683403  0.00339678]]
```

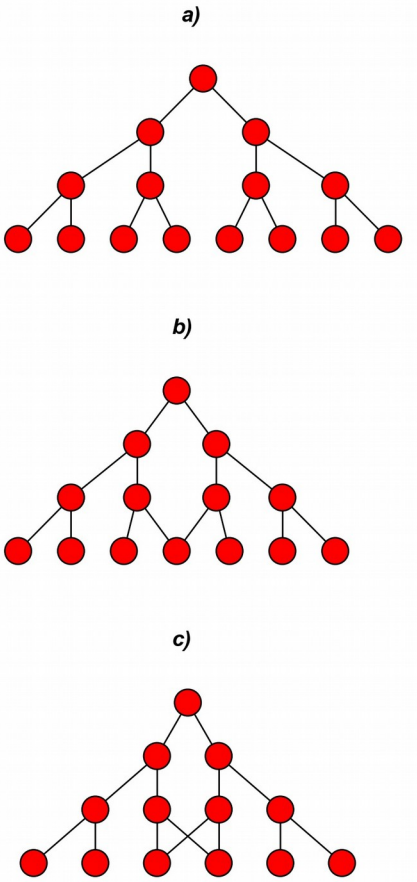


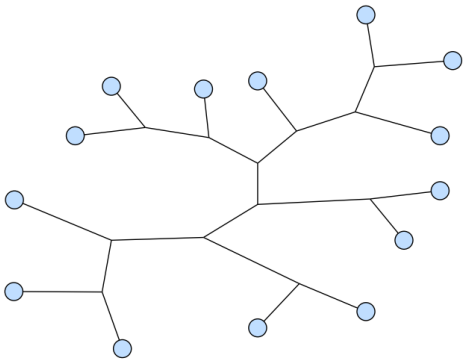
Ideas básicas de Python

Otros ejemplos: Histograma y redes

pp.hist([vector,bins=bins)





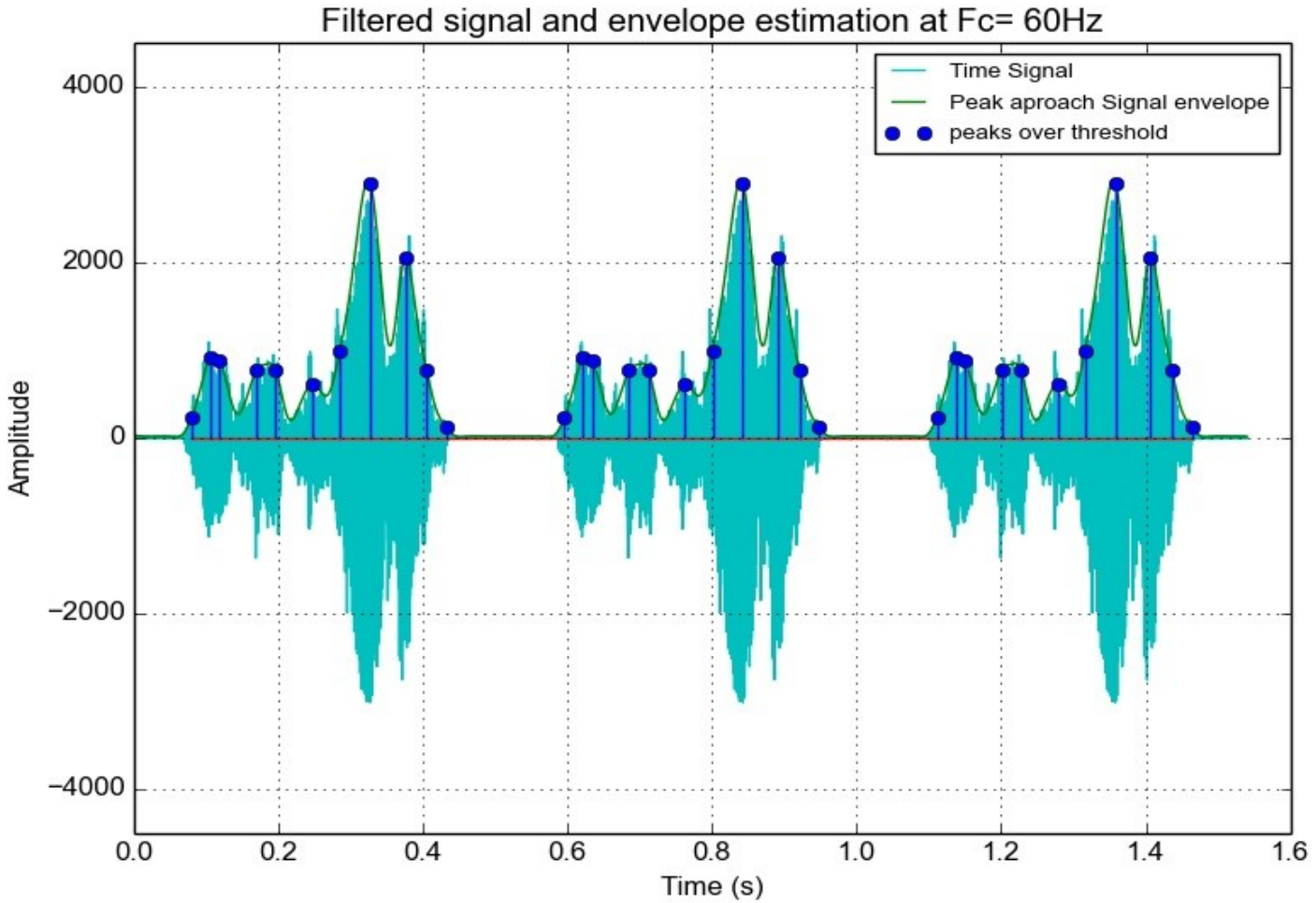


import networkx as nx
import pygraphviz

Ideas básicas de Python

Les dejo un ejemplo donde combino el uso de scipy, numpy y matplotlib:

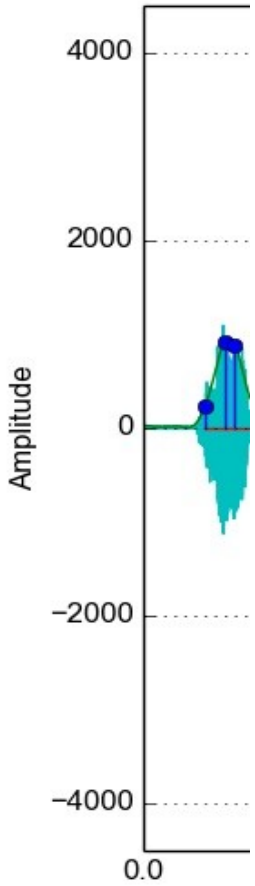
Ideas básicas de Python

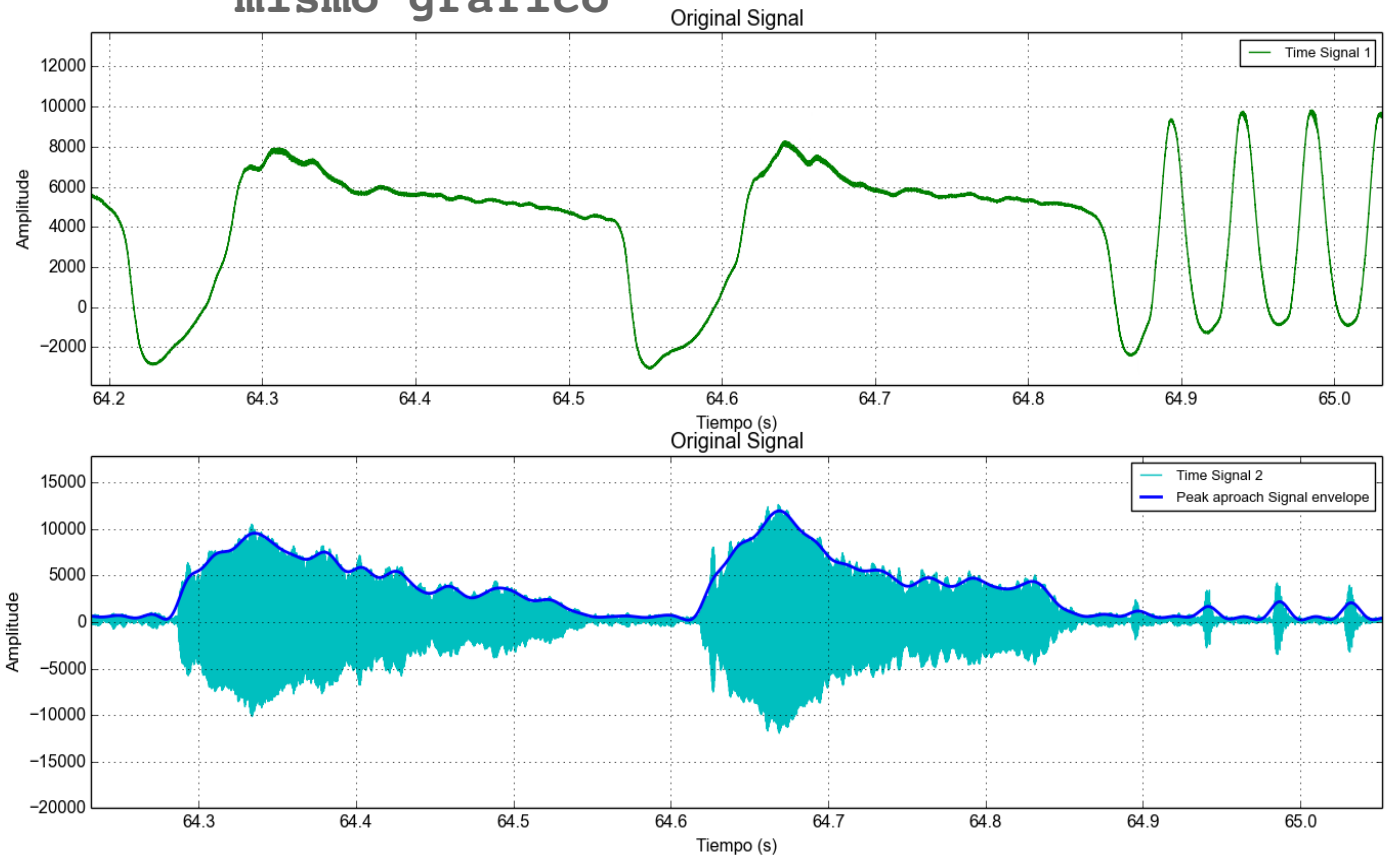


-Array 1 vs array 2 (originalmente una secuencia de sonido muestreada)
-Calculo de valor absoluto
-Filtrado
-Calculo de máximos y mínimos

Ideas básicas de Python

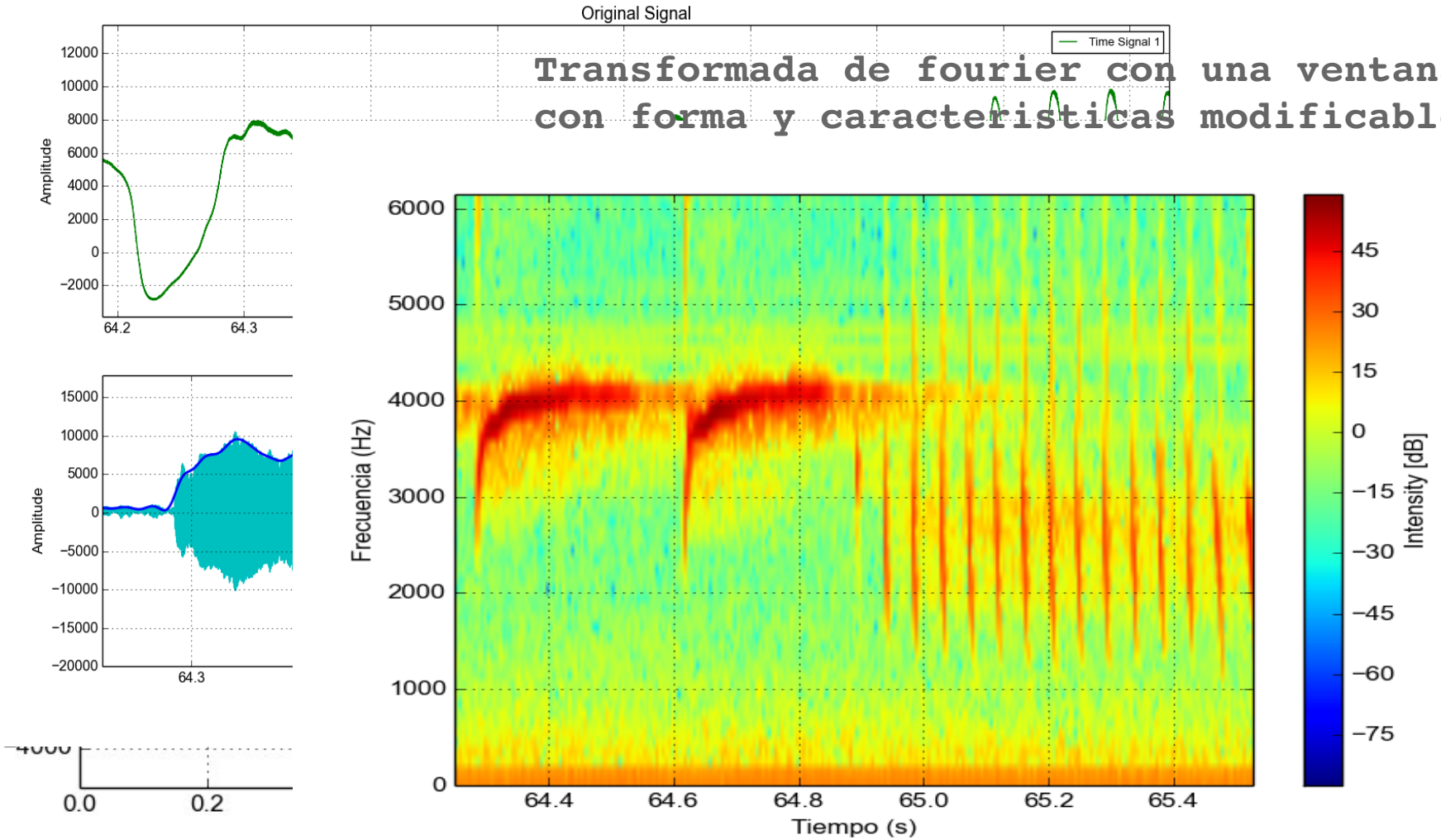
Array 1 vs Array 2 con Array 3 vs Array 2 en un mismo gráfico





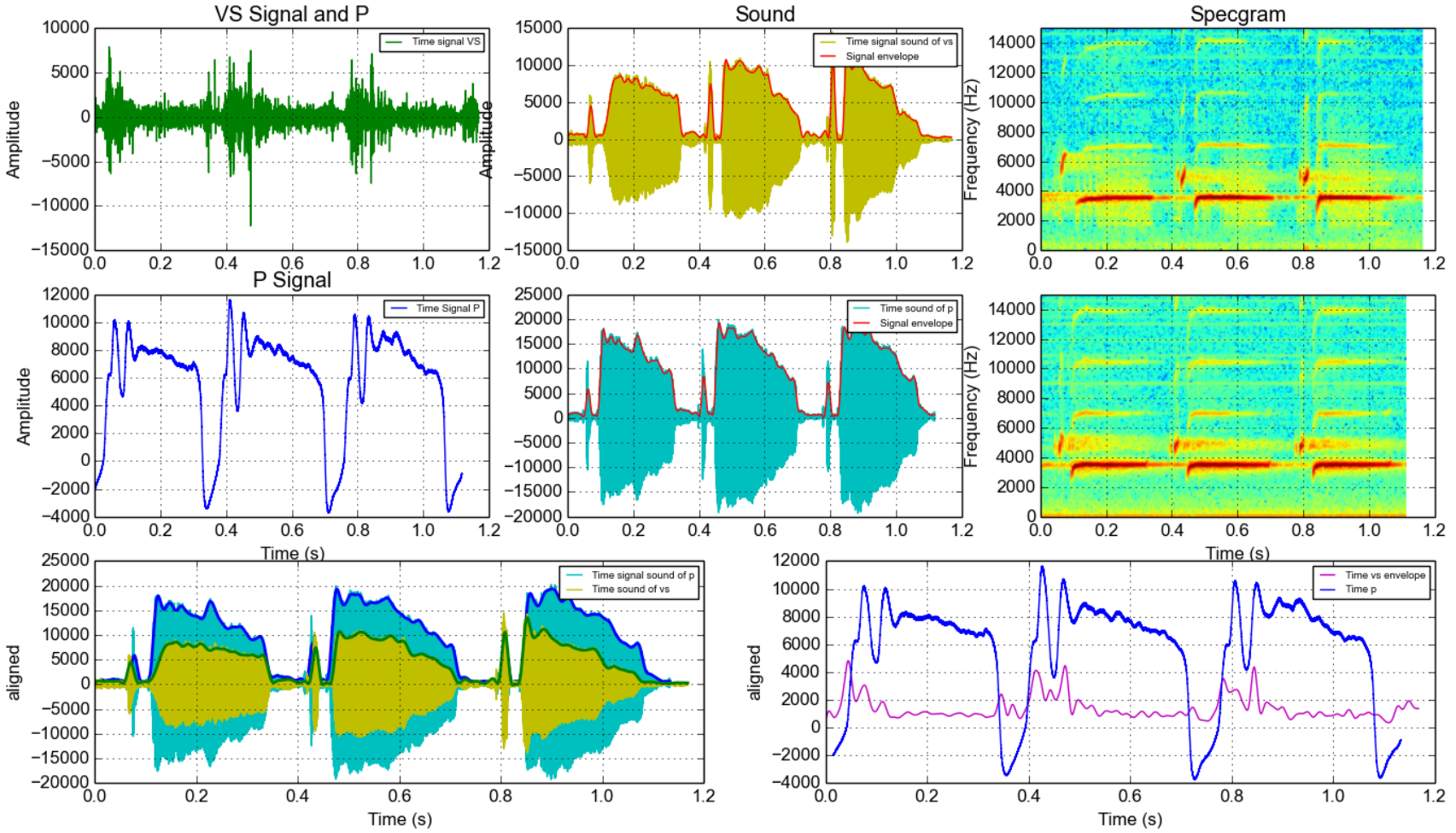
Ideas básicas de Python

Transformada de fourier con una ventana con forma y caracteristicas modificables



Ideas básicas de Python

Ejemplo alinear señales



Ideas básicas de Python

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.

Ideas básicas de Python

Backup:

https://google.github.io/styleguide/pyguide.html