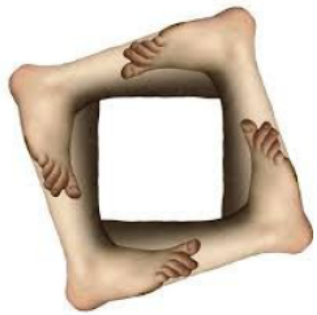
Una introducción a la programación en python

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`

WTPC workshop en técnicas de programación científica

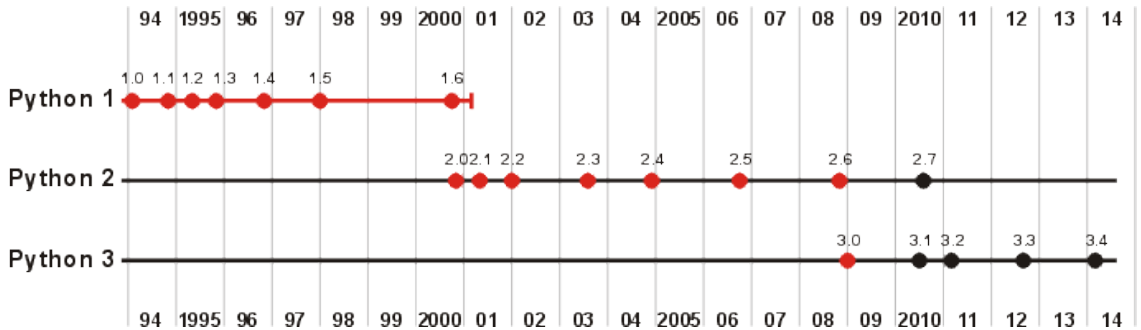
Motivos para aprender Python

- 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.

WTPC
Workshop en técnicas de programación científica

Un poquito de historia

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



Sobre el diseño

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 + 1
2 2
3 >>> a = range(10)
4 >>> print(a)
5 [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.

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

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`

WTPC workshop en técnicas de programación científica

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
```

WTPC workshop en técnicas de programación científica

Sintaxis de Python

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

```
1 if(a>b)
2     foo();
3     bar();
4 baz();
```

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

```
1 if(a>b):
2     foo()
3     bar()
4 baz()
```

Sintaxis de Python

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
```

Sintaxis de Python

Cómo definir una función:

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

Algunas funciones en Python que devuelven valores:

```
1 >> longitud = len('La casa de la pradera')
2 >> print (longitud)
3 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)
```

WTPC workshop en técnicas de programación científica

Recursividad en Python

Un ejemplo simple:

```
1
2 def jugar(intento=1):
3     respuesta = raw_input("De que color es una naranja? ")
4     if respuesta != "naranja":
5         if intento < 3:
6             print "\nFallaste Intentalo de nuevo"
7             intento += 1
8             jugar(intento) # Llamada recursiva
9         else:
10            print "\nPerdiste"
11    else:
12        print "\nGanaste!"
13 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

```
1  a= "Soy Cecilia"
2  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

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

- Tuple

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

- Dict

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

- Set

```
1 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

WTPC workshop en técnicas de programación científica

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.

WTPC workshop en técnicas de programación científica

Numpy

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

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

WTPC workshop en técnicas de programación científica

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.

WTPC workshop en técnicas de programación científica

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.

WTPC
workshop en técnicas de programación científica

Como usarlas?

Es necesario importar las librerias:

```
1 import numpy
2 import scipy
3 import matplotlib.pyplot
```

Se puede hacer de distintas maneras:

```
1 import numpy
2 import numpy as np
3 from numpy import *
```

Yo uso:

```
1 import numpy as np
2 import scipy
3 import matplotlib.pyplot as pp
```

WTPC
Workshop en técnicas de programación científica

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.

WTPC workshop en técnicas de programación científica

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).

WTPC workshop en técnicas de programación científica

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. ])
```

WTPC Workshop en técnicas de programación científica

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.]]
```

WTPC Workshop en técnicas de programación científica

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.])
```

WTPC Workshop en técnicas de programación científica

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]]
```

WTPC Workshop en técnicas de programación científica

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))
```

WTPC 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]]"
```

Como crear un dataset y salvarlo en un .txt

```
1  from math import *
2  import numpy as np
3  import matplotlib.pyplot as pp
4  #File creation
5  f_out_max = open('tabla.txt', 'w')
6  #data generation
7  x    = np.arange(441)
8  Sin1 = 1*np.sin(2*pi*(25/441.0)*x)
9  Sin2 = 0.25*np.sin(2*pi*((25./2)/441.0)*x)
10 Sin  = Sin1+Sin2
11 Vec  = np.c_[x, Sin]
12 #printing
13 print ('Vec: ', Vec.shape)
14 #save in .txt file
15 np.savetxt(f_out_max, Vec, fmt='%f', delimiter='\t', header="x #f(x)")
16 f_out_max.close()
17 #plot figure
18 pp.figure()
19 pp.plot(x*1./44100., Sin, color='r', label='Sin vs x')
20 pp.xlabel('Time (s)')
21 pp.savefig("fig_01.jpg",dpi=200)
```

Como crear un dataset y salvarlo en un .txt

```
1  from math import *
2  import numpy as np
3  import matplotlib.pyplot as pp
4  #File creation
5  f_out_max = open('tabla.txt', 'w')
6  #data generation
7  x    = np.arange(441)
8  Sin1 = 1*np.sin(2*pi*(25/441.0)*x)
9  Sin2 = 0.25*np.sin(2*pi*((25./2)/441.0)*x)
10 Sin  = Sin1+Sin2
11 Vec  = np.c_[x, Sin]
12 #printing
13 print ('Vec: ', Vec.shape)
14 #save in .txt file
15 np.savetxt(f_out_max, Vec, fmt='%f', delimiter='\t', header="x #f(x)")
16 f_out_max.close()
17 #plot figure
18 pp.figure()
19 pp.plot(x*1./44100., Sin, color='r', label='Sin vs x')
20 pp.xlabel('Time (s)')
21 pp.savefig("fig_01.jpg",dpi=200)
```

Como crear un dataset y salvarlo en un .txt

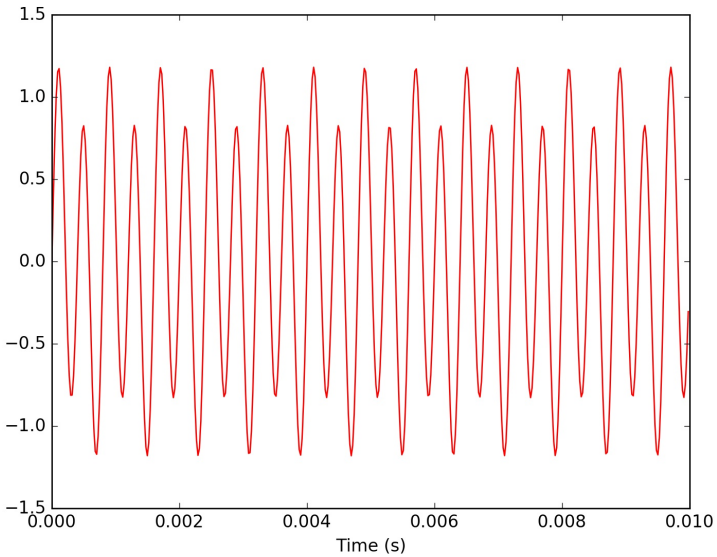
```
1  from math import *
2  import numpy as np
3  import matplotlib.pyplot as pp
4  #File creation
5  f_out_max = open('tabla.txt', 'w')
6  #data generation
7  x    = np.arange(441)
8  Sin1 = 1*np.sin(2*pi*(25/441.0)*x)
9  Sin2 = 0.25*np.sin(2*pi*((25./2)/441.0)*x)
10 Sin  = Sin1+Sin2
11 Vec  = np.c_[x, Sin]
12 #printing
13 print ('Vec: ', Vec.shape)
14 #save in .txt file
15 np.savetxt(f_out_max, Vec, fmt='%f', delimiter='\t', header="x #f(x)")
16 f_out_max.close()
17 #plot figure
18 pp.figure()
19 pp.plot(x*1./44100., Sin, color='r', label='Sin vs x')
20 pp.xlabel('Time (s)')
21 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)
```

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:

```
1  import numpy as np
2
3  file_name_you_want        = np.loadtxt(fname,delimiter=" ")
4
5  print "First column element: ",file_name_you_want[0]
6  #to get the full column:
7
8  Transpose_your_file       = file_name_you_want.T
9
10 print "First column: ", Transpose_your_file[0]
```

Como hacer un ajuste y graficarlo:

```
1  import numpy as np
2  import matplotlib.pyplot as pp
3  from scipy.optimize import curve_fit
4
5  def fitFunc(t, a, b, c):
6      return a*np.exp(-b*t) + c
7
8  t                          = np.linspace(0,4,50)
9  temp                       = fitFunc(t, 2.5, 1.3, 0.5)
10 noisy                      = temp + 0.25*np.random.normal(size=len(temp))
11 fitParams, fitCovariances = curve_fit(fitFunc, t, noisy)
12
13 pp.figure(figsize=(12, 6))
14 pp.ylabel('Temperature (C)', fontsize = 16)
15 pp.xlabel('time (s)', fontsize = 16)
16 pp.xlim(0,4.1)
17 pp.errorbar(t, noisy, fmt = 'ro', yerr = 0.2)
18 sigma = [fitCovariances[0,0], fitCovariances[1,1], fitCovariances[2,2] ]
19 pp.plot(t, fitFunc(t, fitParams[0], fitParams[1], fitParams[2]))
20 pp.plot(t, fitFunc(t, fitParams[0] + sigma[0], fitParams[1] - sigma[1], fitParams[2] + sigma[2]))
21 pp.plot(t, fitFunc(t, fitParams[0] - sigma[0], fitParams[1] + sigma[1], fitParams[2] - sigma[2]))
22 pp.savefig('dataFitted.pdf', bbox_inches=0, dpi=600)
23 pp.show()
```

WTPC Workshop en técnicas de programación científica

Como hacer un ajuste y graficarlo:

```
1  import numpy as np
2  import matplotlib.pyplot as pp
3  from scipy.optimize import curve_fit
4
5  def fitFunc(t, a, b, c):
6      return a*np.exp(-b*t) + c
7
8  t                          = np.linspace(0,4,50)
9  temp                       = fitFunc(t, 2.5, 1.3, 0.5)
10 noisy                      = temp + 0.25*np.random.normal(size=len(temp))
11 fitParams, fitCovariances = curve_fit(fitFunc, t, noisy)
12
13 pp.figure(figsize=(12, 6))
14 pp.ylabel('Temperature (C)', fontsize = 16)
15 pp.xlabel('time (s)', fontsize = 16)
16 pp.xlim(0,4.1)
17 pp.errorbar(t, noisy, fmt = 'ro', yerr = 0.2)
18 sigma = [fitCovariances[0,0], fitCovariances[1,1], fitCovariances[2,2] ]
19 pp.plot(t, fitFunc(t, fitParams[0], fitParams[1], fitParams[2]))
20 pp.plot(t, fitFunc(t, fitParams[0] + sigma[0], fitParams[1] - sigma[1], fitParams[2] + sigma[2]))
21 pp.plot(t, fitFunc(t, fitParams[0] - sigma[0], fitParams[1] + sigma[1], fitParams[2] - sigma[2]))
22 pp.savefig('dataFitted.pdf', bbox_inches=0, dpi=600)
23 pp.show()
```

WTPC Workshop en técnicas de programación científica

Como hacer un ajuste y graficarlo:

```
1  import numpy as np
2  import matplotlib.pyplot as pp
3  from scipy.optimize import curve_fit
4
5  def fitFunc(t, a, b, c):
6      return a*np.exp(-b*t) + c
7
8  t                          = np.linspace(0,4,50)
9  temp                       = fitFunc(t, 2.5, 1.3, 0.5)
10 noisy                      = temp + 0.25*np.random.normal(size=len(temp))
11 fitParams, fitCovariances = curve_fit(fitFunc, t, noisy)
12
13 pp.figure(figsize=(12, 6))
14 pp.ylabel('Temperature (C)', fontsize = 16)
15 pp.xlabel('time (s)', fontsize = 16)
16 pp.xlim(0,4.1)
17 pp.errorbar(t, noisy, fmt = 'ro', yerr = 0.2)
18 sigma = [fitCovariances[0,0], fitCovariances[1,1], fitCovariances[2,2] ]
19 pp.plot(t, fitFunc(t, fitParams[0], fitParams[1], fitParams[2]))
20 pp.plot(t, fitFunc(t, fitParams[0] + sigma[0], fitParams[1] - sigma[1], fitParams[2] + sigma[2]))
21 pp.plot(t, fitFunc(t, fitParams[0] - sigma[0], fitParams[1] + sigma[1], fitParams[2] - sigma[2]))
22 pp.savefig('dataFitted.pdf', bbox_inches=0, dpi=600)
23 pp.show()
```

WTPC Workshop en técnicas de programación científica

Como hacer un ajuste y graficarlo:

```
1  import numpy as np
2  import matplotlib.pyplot as pp
3  from scipy.optimize import curve_fit
4
5  def fitFunc(t, a, b, c):
6      return a*np.exp(-b*t) + c
7
8  t                          = np.linspace(0,4,50)
9  temp                       = fitFunc(t, 2.5, 1.3, 0.5)
10 noisy                      = temp + 0.25*np.random.normal(size=len(temp))
11 fitParams, fitCovariances = curve_fit(fitFunc, t, noisy)
12
13 pp.figure(figsize=(12, 6))
14 pp.ylabel('Temperature (C)', fontsize = 16)
15 pp.xlabel('time (s)', fontsize = 16)
16 pp.xlim(0,4.1)
17 pp.errorbar(t, noisy, fmt = 'ro', yerr = 0.2)
18 sigma = [fitCovariances[0,0], fitCovariances[1,1], fitCovariances[2,2] ]
19 pp.plot(t, fitFunc(t, fitParams[0], fitParams[1], fitParams[2]))
20 pp.plot(t, fitFunc(t, fitParams[0] + sigma[0], fitParams[1] - sigma[1], fitParams[2] + sigma[2]))
21 pp.plot(t, fitFunc(t, fitParams[0] - sigma[0], fitParams[1] + sigma[1], fitParams[2] - sigma[2]))
22 pp.savefig('dataFitted.pdf', bbox_inches=0, dpi=600)
23 pp.show()
```

WTPC Workshop en técnicas de programación científica

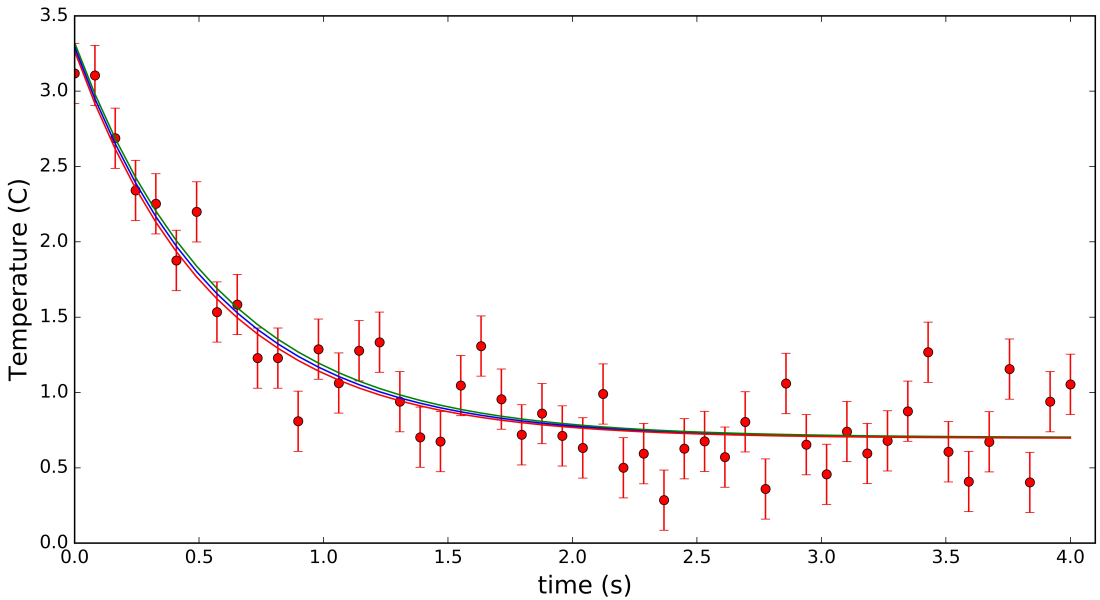
Como hacer el ajuste

We obtain the fit parameters:

```
[ 2.595658    1.74438726  0.69809511]
```

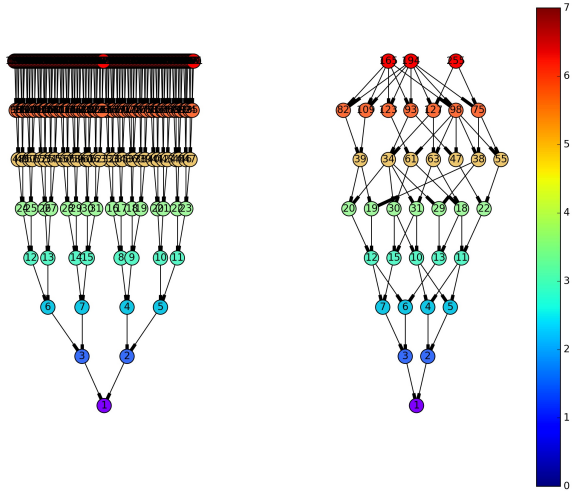
The covariance matrix:

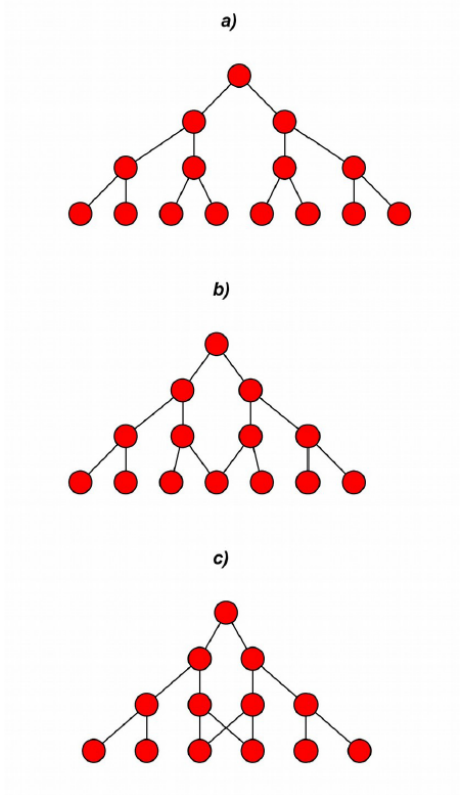
```
[[ 0.02506636  0.01490486 -0.00068609]
 [ 0.01490486  0.04178044  0.00641246]
 [-0.00068609  0.00641246  0.00257799]]
```



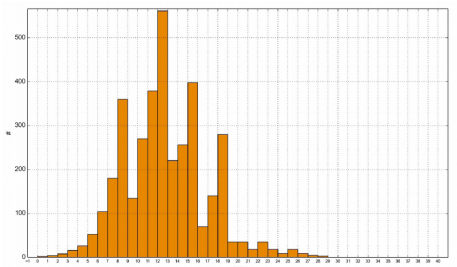
Otros tipos de gráficos

```
1 import networkx as nx
2 import pygraphviz
3 from graphviz import *
```



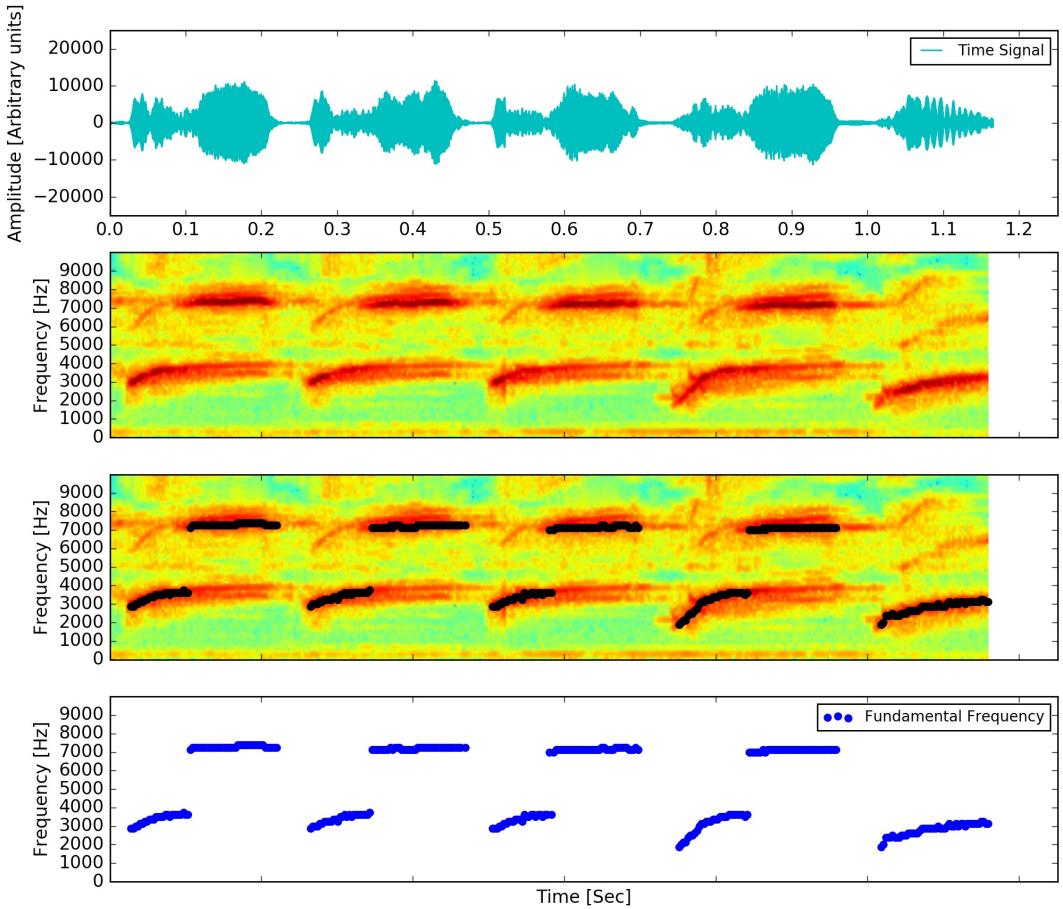



```
1 import matplotlib.pyplot as pp
2 pp.hist([vector,bins=bins)
```



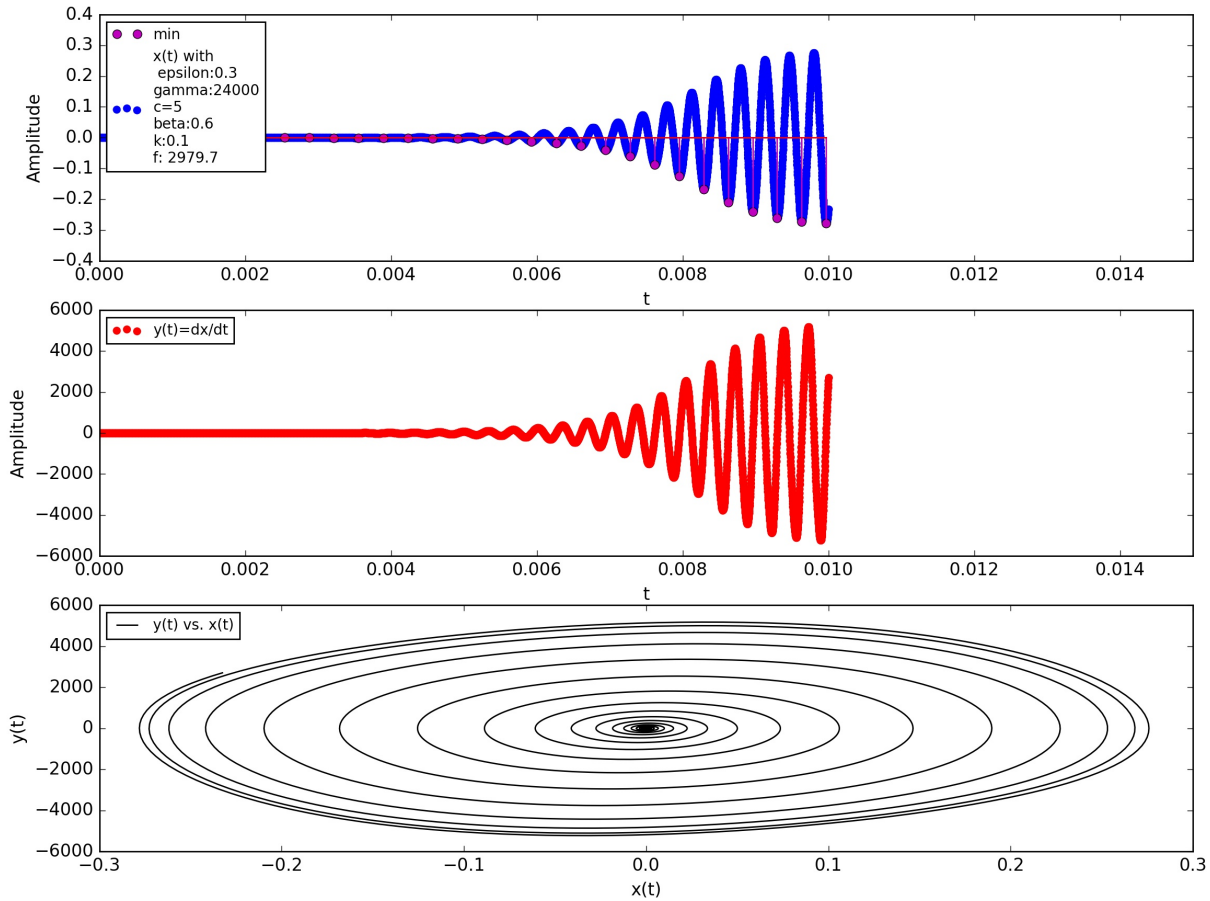
Aplicando Transformada de fourier y un sonograma

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



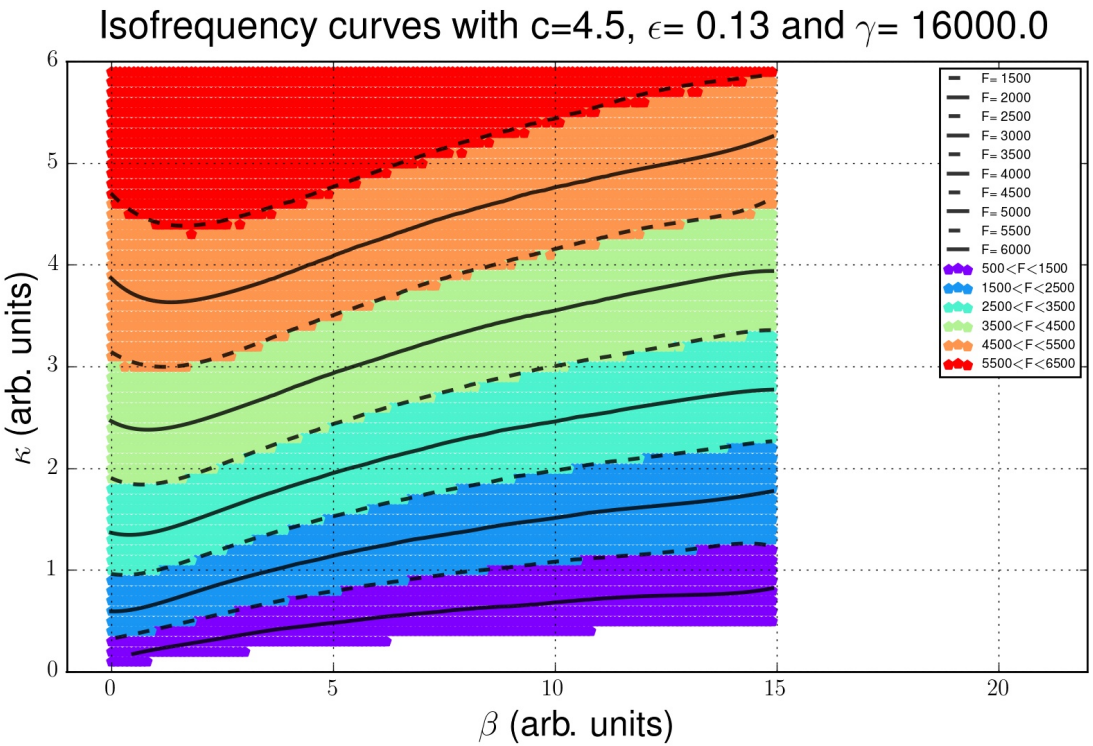
Ejemplo de integración numérica

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



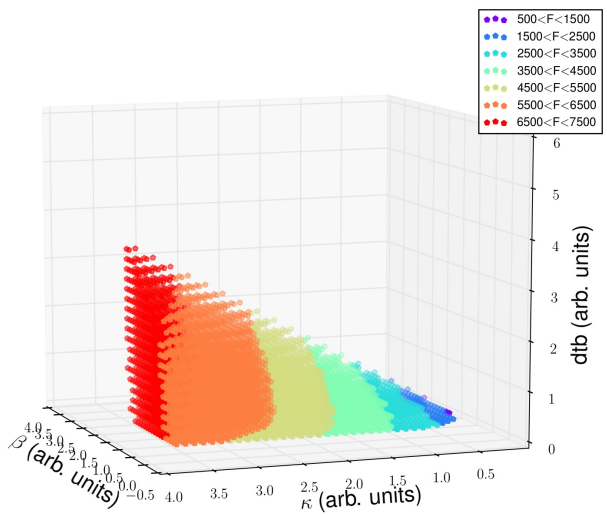
Scatter plot e integración numérica múltiple

- Cada punto representa un valor obtenido por la integración numérica de la ecuación anterior.
- Los colores representan regiones de frecuencia similar.

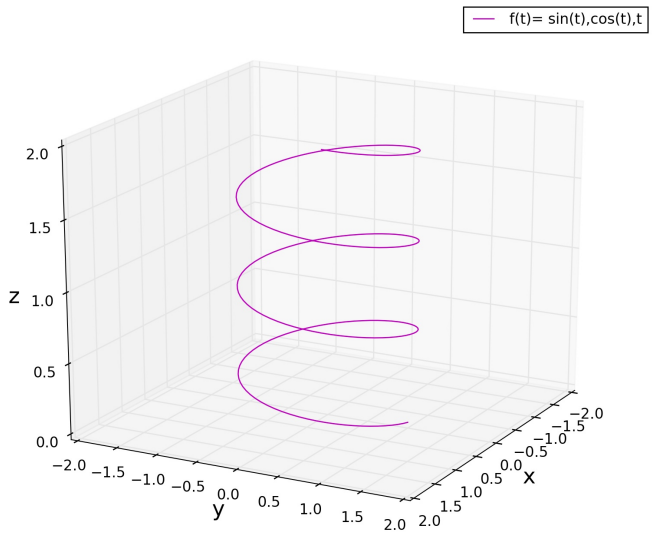


Se pueden hacer gráficos 3d y también videos

El ejemplo anterior estudiando una variable extra:



Curva paramétrica:



Conclusiones

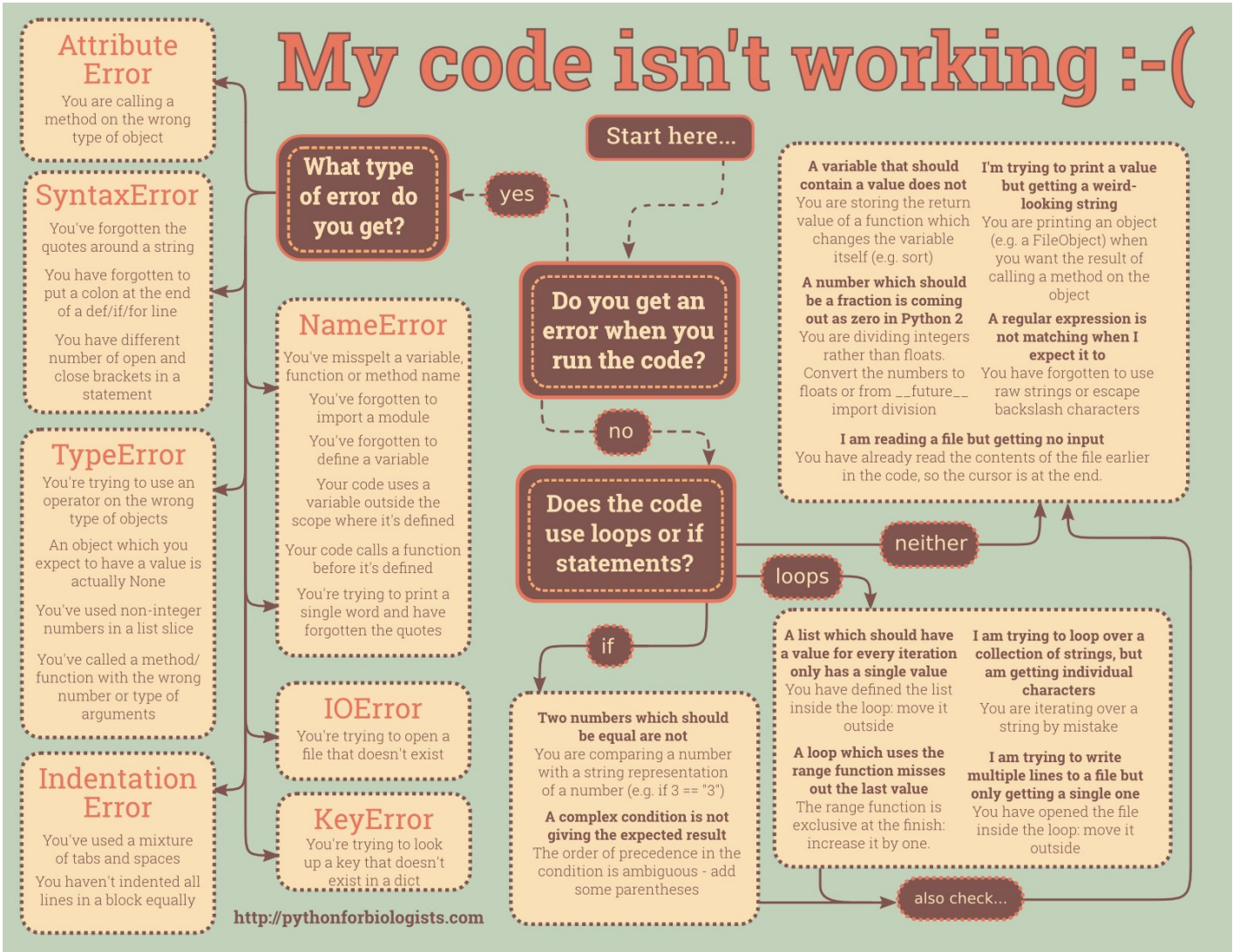
- Les dejamos una colección infinitamente grande de herramientas y recursos recombinables y reutilizables para poner en práctica.
- El salto es mas pequeño de lo que parece.
- Posibilidades de reutilizar y adaptar código a nuestras necesidades para crear soluciones prácticas.
- Una guia de estilo en: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

WTPC workshop en técnicas de programación científica

Más librerías para análisis de datos:

- Scikit learn:
  `http://scikit-learn.org`
- Pandas
  `http://pandas.pydata.org/` **Ver la Charla del Martes 6/3!!!!!!!**

Otras librerías para análisis de datos:



Algo más

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

WTPC
workshop en técnicas de programación científica