



Python para el cálculo científico

Daniel Lubián Arenillas

12 de febrero de 2018



Presentando Python

Librerías importantes

Programación orientada a objetos

Sintaxis básica de Python

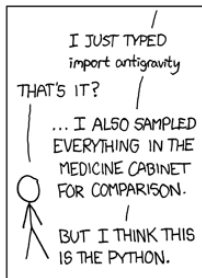
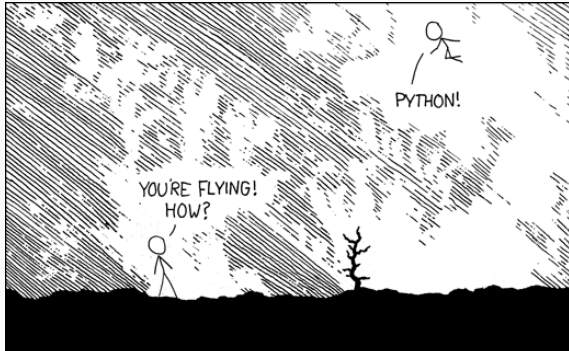
Numpy: el `ndarray`

Scipy

Presentando Python



- Creado por Guido van Rossum en 1991.
- Lenguaje de propósito general:
 - Cálculo científico
 - Desarrollo web
 - Administración de sistemas
 - GUIs
 - Inteligencia artificial
 - Todo es posible
- Lenguaje multiparadigma, permite programación estructurada, orientada a objetos, funcional,...
- Lenguaje interpretado, no compilado.





- Dos versiones: 2.7 y **3.6**
- Libre, abierto y gratuito, con una comunidad enorme → todo a golpe de Google.
- Mil y una librerías abiertas y gratuitas.
- Rápido y fácil de escribir, puede ser lento de ejecutar.

Para escribirlo: Spyder, cuadernos Jupyter, Pycharm, VS Code, Atom, Geany, Notepad++...



IP[y]:
IPython

Numpy: cálculo numérico

Scipy: cálculo científico

Matplotlib: graficado

Pandas: análisis de datos

IPython: consola interactiva



Un **objeto** tiene **métodos** (“funciones”) y **atributos** (“variables”) que lo constituyen.

- `A.shape`
- `z.conjugate()`
- `v.reshape((3,4))`

Todo Python trabaja con objetos¹

¹(hasta donde yo sé)

Sintaxis básica de Python



Entero	integer	1
Con coma flotante	float	1.
Complejo	complex	1. + 2j
Booleano	boolean	True

División de enteros

```
1 print(3 / 2)
2 print(3 / 2.)
3 print(3.0 // 2)
4
5 ## SALIDA
6 # 1.5
7 # 1.5
8 # 1
```



Cadenas $\rightarrow s = \text{"perro"}$ inmutable

Listas $\rightarrow l = [1, \text{'perro'}, \text{True}]$

Tuplas $\rightarrow t = (1, \text{'perro'}, \text{True})$ inmutable

Diccionarios $\rightarrow d = \{ \text{'O': 16, 'H2O': 18} \}$

Ojo: los índices van de 0 a n-1, como en C

- $s[0]$ devuelve 'p'
- $l[-1]$ devuelve True
- $t[3]$ no existe
- $d[\text{'H2O'}]$ devuelve 18
- $l[0:2]$ devuelve $[1, \text{'perro'}]$ ($0 \leq i < 2$)



Condición: if-elif-else

```
1 if 1 in x:
2     haz esto
3 elif 2 < 7 and 2 > 1:
4     haz esto otro
5 else:
6     o esto
```



Bucle: for

```
1 x = [2, "muse"]
2 n = 2
3
4 for i in range(0, n):
5     for x_i in x:
6         print(i, x_i)
7
8 ## SALIDA:
9 # 0 2
10 # 0 muse
11 # 1 2
12 # 1 muse
```



Bucle: while

```
1 z = 1 + 1j
2
3 while abs(z) < 100:
4     if z.imag == 0:
5         break
6     z = z**2 + 1
```



Iteración avanzada

```
1 words = ('cool', 'powerful', 'readable')
2
3 for index, item in enumerate(words):
4     print((index, item))
5
6 # (0, 'cool')
7 # (1, 'powerful')
8 # (2, 'readable')
```



List comprehensions

```
1 x = [i**2 for i in range(6)]  
2 print(x)  
3  
4 ## SALIDA  
5 # [0, 1, 4, 9, 16, 25]
```




Definición

```
1 # funciones normales
2 def mi_funcion(x, z=2):
3     return x**2 + z
4
5 # funciones anonimas
6 f = lambda x: x**3 + 5
```

Paso por referencia

Los argumentos de las funciones se pasan por referencia, no por parámetro. Se mete la variable, no una copia, por lo que si se hace alguna modificación a `x` en el ámbito de la función, `x` cambiará fuera de la función.



Importar

```
1 import function
2 import numpy as np
3 from scipy.linalg import inv as hazme_inversa
4 import matplotlib.pyplot as plt
5 from math import *
6
7 e = function.mi_funcion(5, z=3)
8 z = np.linspace(0, 1, 3)
9 i = hazme_inversa(np.eye(3))
10 plt.plot(z)
11 e = sqrt(e)
```

Numpy: el ndarray



- Es un clase eficiente para computación en *arrays*.
- Almacenan cualquier tipo, pero en todos los elementos el mismo.
- Atributos y métodos más útiles:
 - **ndims**: entero con el número de dimensiones.
 - **shape**: tupla con la forma.
 - **reshape()**: cambia la forma según la tupla introducida.
 - **T**: traspuesta
 - <https://docs.scipy.org/doc/numpy/reference/generated/numpy.ndarray.html>



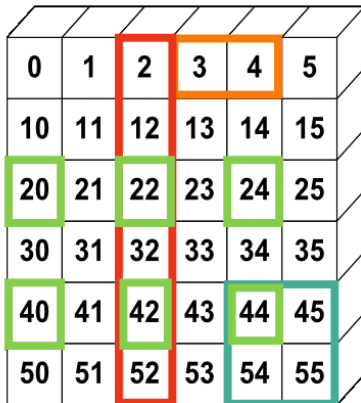
- `np.array([1, 2, 3])`
- `np.zeros((2,))`
- `np.ones((2, 1))`
- `np.zeros_like(a)`
- `np.empty((3, 4))`
- `np.linspace(0, 1, 6)`

```
>>> a[0,3:5]  
array([3,4])
```

```
>>> a[4:,4:]  
array([[44, 45],  
       [54, 55]])
```

```
>>> a[:,2]  
array([2,12,22,32,42,52])
```

```
>>> a[2::2,::2]  
array([[20,22,24]  
       [40,42,44]])
```



0	1	2	3	4	5
10	11	12	13	14	15
20	21	22	23	24	25
30	31	32	33	34	35
40	41	42	43	44	45
50	51	52	53	54	55



- `+`, `-`, `*`, `**`... funcionan *element-wise*
- El producto interno se puede hacer de varias maneras:
`A.dot(B)`, `np.dot(A,B)`, `A @ B` (multiplicación de matrices de toda la vida)

Scipy



- `scipy.linalg`
Álgebra lineal.
- `scipy.integrate`
Integración de integrales y EDOs.
- `scipy.optimize`
solvers para ecuaciones no lineales y optimizadores.
- `scipy.sparse`
Matrices dispersas.
- `scipy.constants`
Constantes.

API Reference

The exact API of all functions and classes, as given by the docstrings. The API documents expected types and allowed features for all functions, and all parameters available for the algorithms.

- Clustering package (`scipy.cluster`)
- Constants (`scipy.constants`)
- Discrete Fourier transforms (`scipy.fftpack`)
- Integration and ODEs (`scipy.integrate`)
- Interpolation (`scipy.interpolate`)
- Input and output (`scipy.io`)
- Linear algebra (`scipy.linalg`)
- Miscellaneous routines (`scipy.misc`)
- Multi-dimensional image processing (`scipy.ndimage`)
- Orthogonal distance regression (`scipy.odr`)
- Optimization and root finding (`scipy.optimize`)
- Signal processing (`scipy.signal`)
- Sparse matrices (`scipy.sparse`)
- Sparse linear algebra (`scipy.sparse.linalg`)
- Compressed Sparse Graph Routines (`scipy.sparse.csgraph`)
- Spatial algorithms and data structures (`scipy.spatial`)
- Special functions (`scipy.special`)
- Statistical functions (`scipy.stats`)
- Statistical functions for masked arrays (`scipy.stats.mstats`)
- Low-level callback functions



Integral de una función

$$I = \int_0^{4,5} J_{2,5}(x) dx$$

```
1 import scipy.integrate as integrate
2 import scipy.special as special
3 result = integrate.quad(
4     lambda x: special.jv(2.5,x),
5     0, 4.5)
6 print(result)
7
8 # resultado                error absoluto estimado
9 # (1.1178179380783249, 7.8663172481899801e-09)
```



Integral de una serie de datos

```
1 import scipy.integrate as integrate
2 import scipy.special as special
3 import numpy as np
4
5 x = np.linspace(0, 4.5, 50)
6 y = special.jv(2.5, x)
7 result1 = integrate.trapz(y, x)
8 result2 = integrate.simps(y, x)
9 ## resultado
10 # quad:  1.1178179380783249
11 # trapz: 1.11767339787
12 # simps: 1.11781225777
```



$$\frac{d^2w}{dz^2} - zw(z) = 0$$
$$w(0) = \frac{1}{\sqrt[3]{3^2} \cdot \Gamma\left(\frac{2}{3}\right)}$$
$$\left. \frac{dw}{dz} \right|_{z=0} = -\frac{1}{\sqrt[3]{3} \cdot \Gamma\left(\frac{1}{3}\right)}$$

Solución: $w(z) = \text{Ai}(z)$



$$\mathbf{y} = \begin{Bmatrix} w \\ \frac{dw}{dz} \end{Bmatrix}$$
$$\frac{d}{dz} \begin{Bmatrix} y_0 \\ y_1 \end{Bmatrix} = \begin{bmatrix} 0 & 1 \\ z & 0 \end{bmatrix} \begin{Bmatrix} y_0 \\ y_1 \end{Bmatrix}$$



Vamos a probarlo

- Ir a uno de los repositorios de ese seminario:
 - https://gitlab.com/muse-dlubian/seminario_vida_moderna
 - https://github.com/danbul/seminario_vida_moderna
- Descargar el cuaderno Jupyter: `python/ode.ipynb`
- Subirlo a <https://try.jupyter.org/>

$$\varepsilon = \frac{\Gamma(\gamma)}{P^{\frac{1}{\gamma}} \sqrt{\frac{2\gamma}{\gamma-1} \left(1 - P^{\frac{\gamma-1}{\gamma}}\right)}}$$

Algunas posibilidades

- Funciones escalares
 - `brentq`: método de Brent
 - `newton`: Newton-Raphson
- Multidimensional
 - `root`
 - `fsolve`
 - `broyden1`