

# Python para Análisis de datos: Introducción

## Sesión 1

Jesús Fernández (fernandez.cuesta@gmail.com)

16 Octubre 2018

Introducción a Python

Python es usado en

Python como lenguaje de programación

¿?

Instalación y entorno

Entornos virtuales

Conda

Instalación

Práctica I

Práctica II

Estructura del código

## Introducción a Python



Creado en 1990 por Guido van Rossum

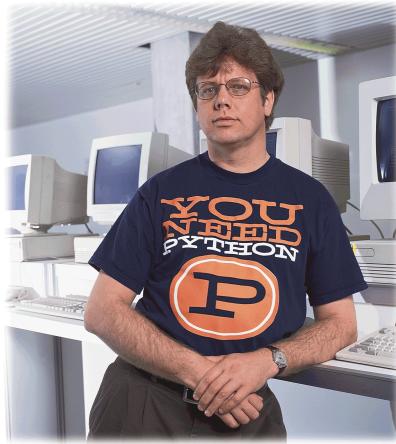


Figure 1: Guido van Rossum, 1999

- ▶ Énfasis en la productividad y legibilidad del código (PEP20)
- ▶ *Beautiful is better than ugly*
- ▶ *Explicit is better than implicit*
- ▶ *Simple is better than complex*
- ▶ *Complex is better than complicated*
- ▶ *Readability counts*

## Ejemplo:

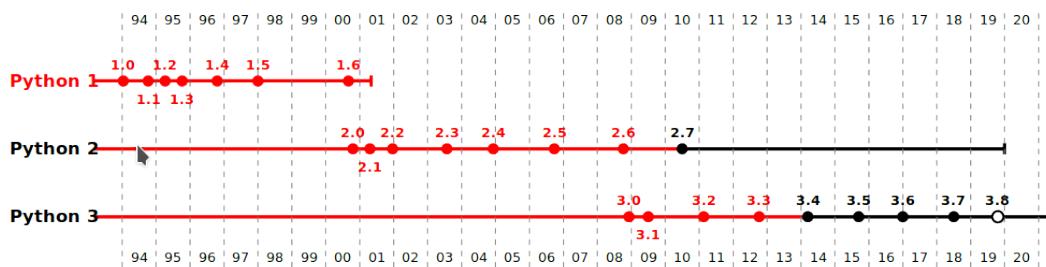
```
from math import factorial

try:
    num = input("Introduce un número [0, 1, 2, ...]: ")
    num = int(num)
    assert num >= 0
except ValueError:
    print('Debes introducir un número')
except AssertionError:
    print('Error, el número no puede ser negativo')
else:
    print('El factorial de {} es {}'.format(num, factorial(num)))
```

- ▶ Actualmente dos versiones principales activas:
  - ▶ Python2.7 (soporte hasta 1/1/2020)
  - ▶ Python3.x (3.6, 3.7) ←

python x.y.z  
x: versión principal, incompatibles entre sí [2, 3]  
y: versión secundaria, normalmente compatibles  
z: versión menor (errores y seguridad)

\$ python -V  
Python 3.7.0



Línea temporal de versiones, en rojo: versión obsoleta

- ▶ Lenguaje de alto nivel, interpretado, orientado a objetos
- ▶ Alta productividad, no compilado
- ▶ Gran cantidad de librerías incorporadas (*batteries included*)
- ▶ + librerías externas (>1M): Python Package Index (PyPI)

## Alto nivel

- ▶ Sencillo y comprensible
- ▶ Fácil de aprender
- ▶ Abstracción de datos
- ▶ Menos líneas de código
- ▶ Interfaces simples (*pythonic*)

## Código ejemplo en C++

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <netdb.h>
#include <arpa/inet.h>
#include <err.h>

char response[] = "HTTP/1.1 200 OK\r\n"
"Content-Type: text/html; charset=UTF-8\r\n\r\n\r\n"
"Hello, world!\r\n";

int main()
{
    int one = 1, client_fd;
    struct sockaddr_in svr_addr, cli_addr;
    socklen_t sin_len = sizeof(cli_addr);

    int sock = socket(AF_INET, SOCK_STREAM, 0);
    if (sock < 0)
        err(1, "can't open socket");

    setsockopt(sock, SOL_SOCKET, SO_REUSEADDR, &one, sizeof(int));

    int port = 8080;
```

## Equivalente python

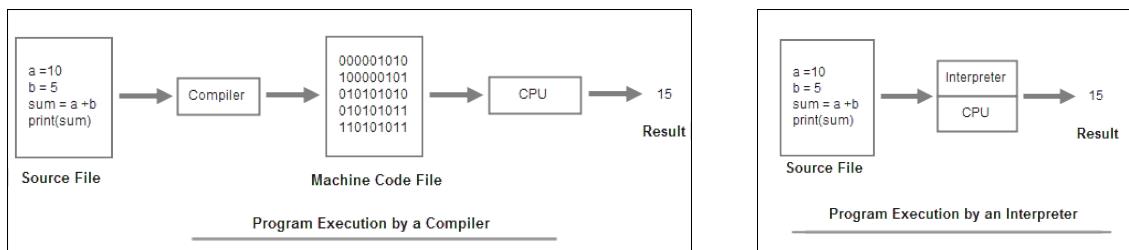
```
from flask import Flask
app = Flask(__name__)

@app.route("/")
def hello():
    return "Hello World!"

Flask.run(app, port=8080)
```

## Interpretado

- ▶ No es necesario compilar



Necesitamos tener instalado un intérprete python

## Interpretado

- ▶ Compatible (en general) en distintos sistemas operativos y arquitecturas
  - ▶ Linux
  - ▶ MacOS
  - ▶ Windows
  - ▶ otros (AIX, AS/400, z/OS, OpenVMS, ARM, ...)
- ▶ Depuración de errores desde intérprete
- ▶ Gestión automática de memoria

## Diferentes paradigmas de programación

- ▶ Orientado a objetos
- ▶ Procedural
- ▶ Imperativo
- ▶ Funcional (<100%)

Python es usado en

Data Science



Machine Learning



Desarrollo Web

(p.e. pinterest, instagram, . . . )

The Django logo features the word "django" in a large, white, lowercase, sans-serif font. It is set against a solid dark teal rectangular background.

Desarrollo software (scripts, prototipos, ...)

como “pegamento” entre componentes escritos en otros lenguajes



Fabric



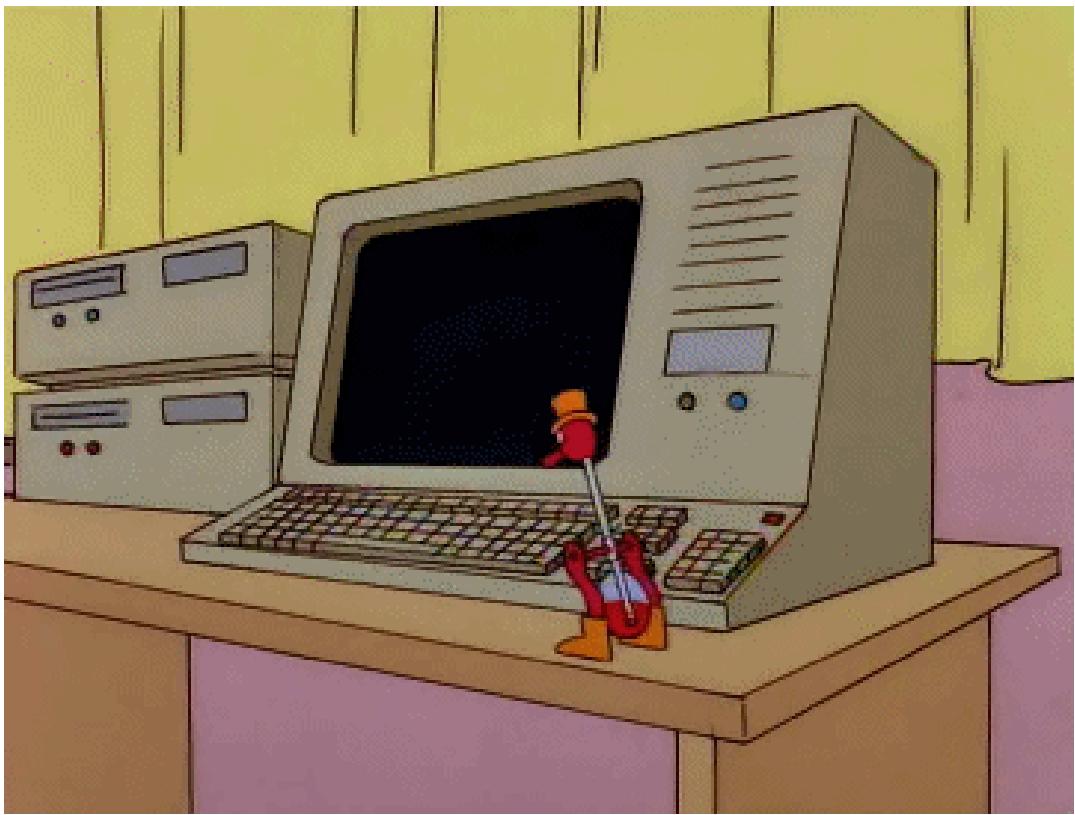
Celery

Automatización de procesos software



ANSIBLE



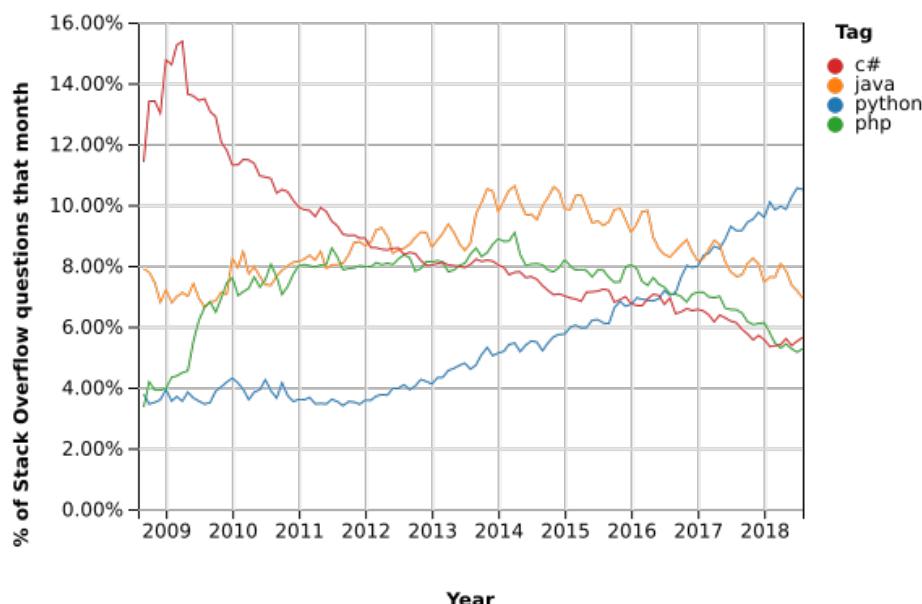


Python como lenguaje de programación

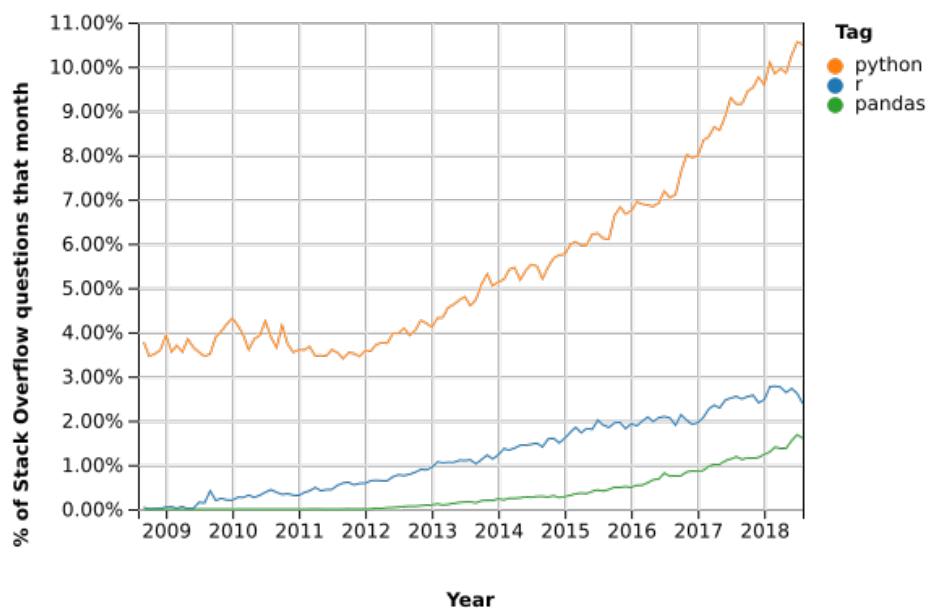
### The 2018 Top Programming Languages, IEEE

Language Rank	Types	Spectrum Ranking
1. Python	🌐💻⚙️	100.0
2. C++	📱💻⚙️	99.7
3. Java	🌐📱💻	97.5
4. C	📱💻⚙️	96.7
5. C#	🌐📱💻	89.4
6. PHP	🌐	84.9
7. R	💻	82.9
8. JavaScript	🌐📱	82.6
9. Go	🌐💻	76.4
10. Assembly	⚙️	74.1

gran soporte en foros, comunidades, conferencias, ...

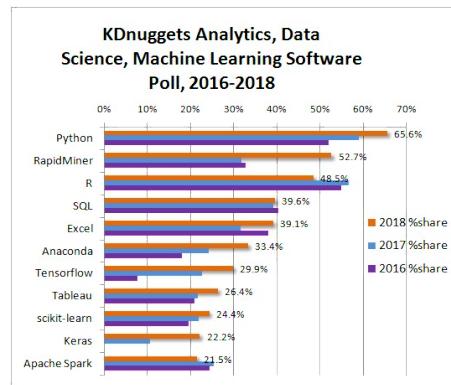


fuente: stackoverflow



fuente: stackoverflow

- ▶ R: muy enfocado en análisis estadístico
- ▶ Python: generalista, con librerías especializadas (pandas, scikit-learn, scipy, ... )
- ▶ Encuesta 2017
- ▶ Encuesta 2018
- ▶ Python Data Science



?  
?

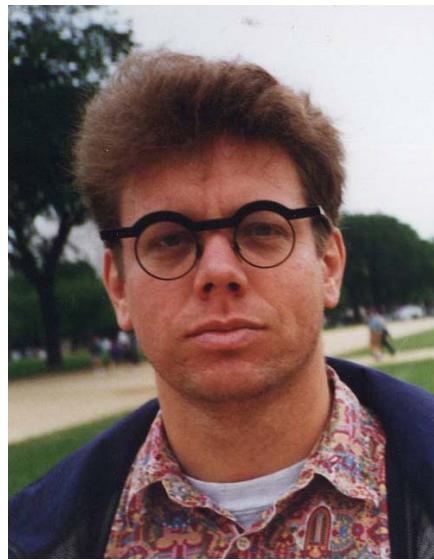


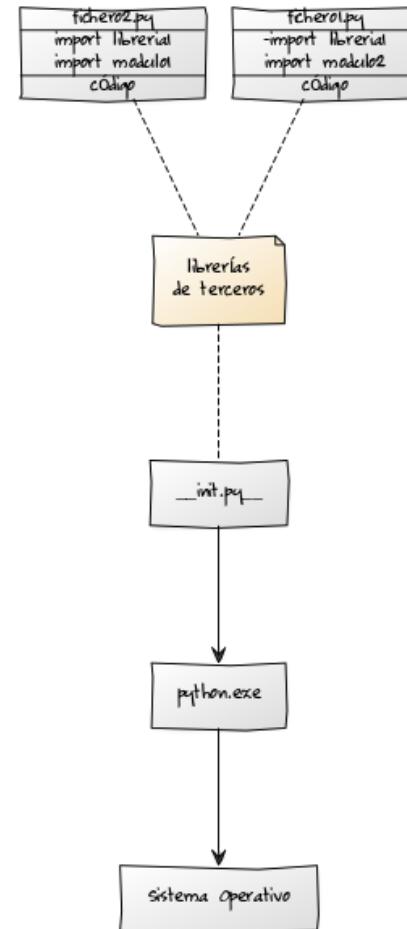
Figure 2: Guido van Rossum, 1995

### Instalación y entorno

Diferentes distribuciones para Windows:

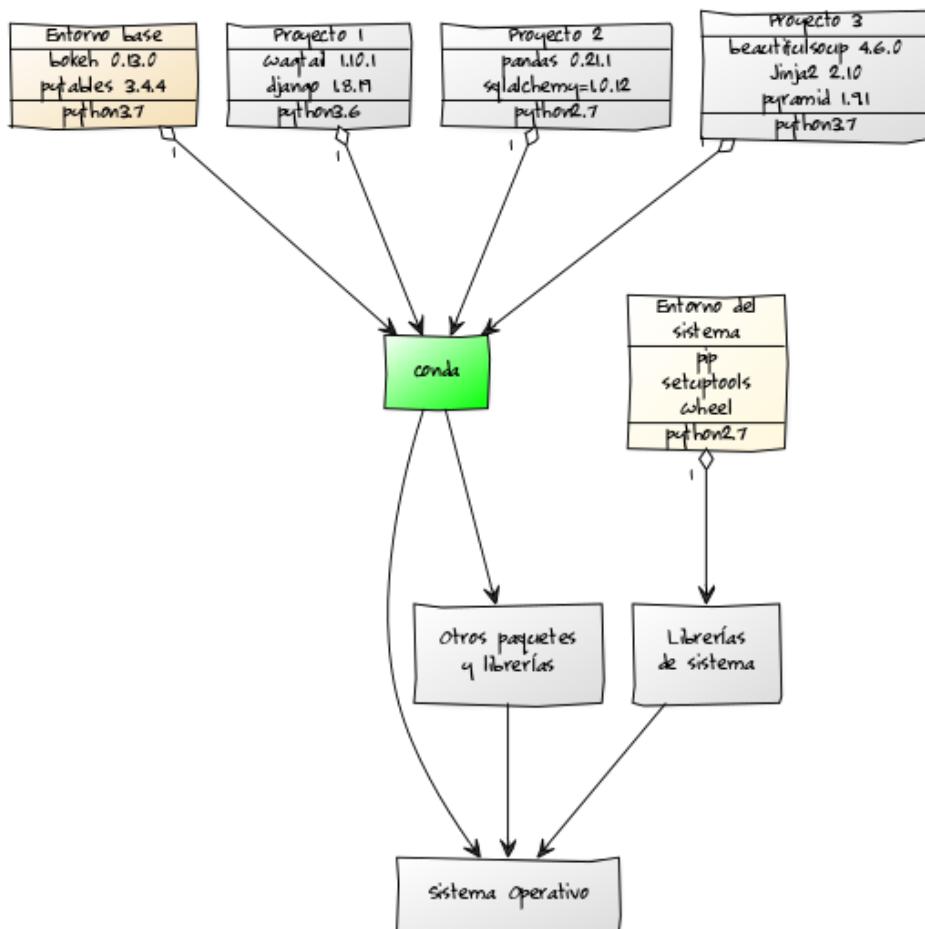
- ▶ python
- ▶ conda
- ▶ winpython
  - ▶ enfocado a sistemas Windows
  - ▶ no necesita ser instalado
  - ▶ incluye compilador C/C++
- ▶ activepython (comercial)

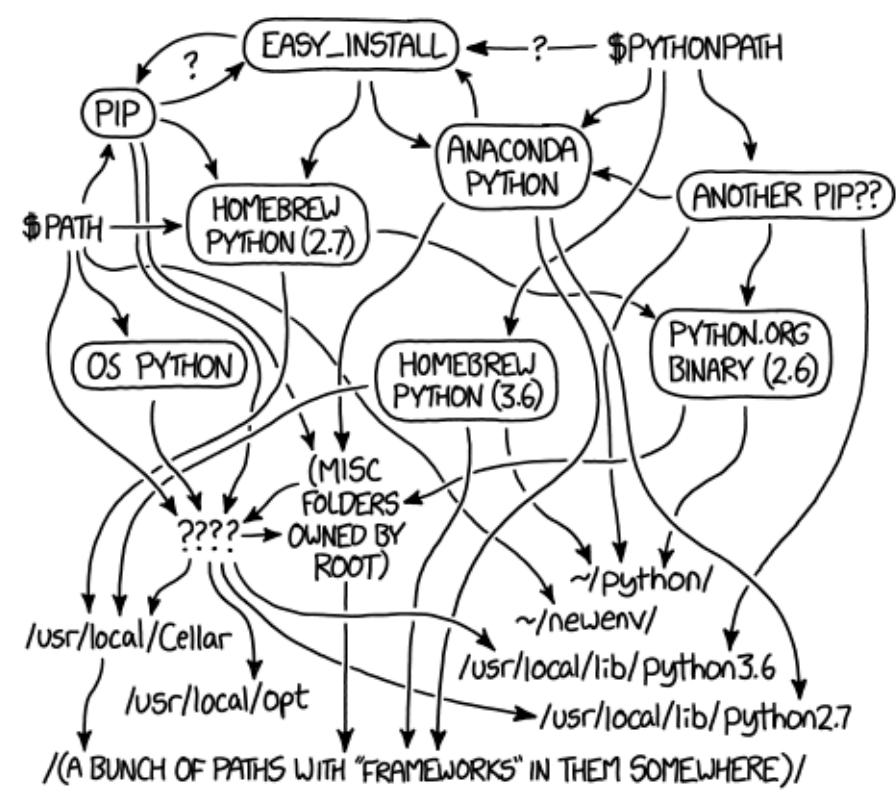
- ▶ Para ejecutar código python necesitaremos un **intérprete** (python.exe).
- ▶ Preinstalado en ciertos S.O. (p.e. Linux)
- ▶ Diferentes tipos de intérprete (CPython, PyPy, Jython, IronPython...)
- ▶ Podemos tener “n” intérpretes distintos instalados en el sistema, cada uno con diferentes librerías
- ▶ conda: instala por defecto un entorno (intérprete) base



**Regla general:** evitar usar el intérprete global del sistema y el entorno base

- ▶ Puede afectar a otros componentes
- ▶ Dependencias entre distintos proyectos
- ▶ Puede no ser la misma versión que la requerida en un proyecto



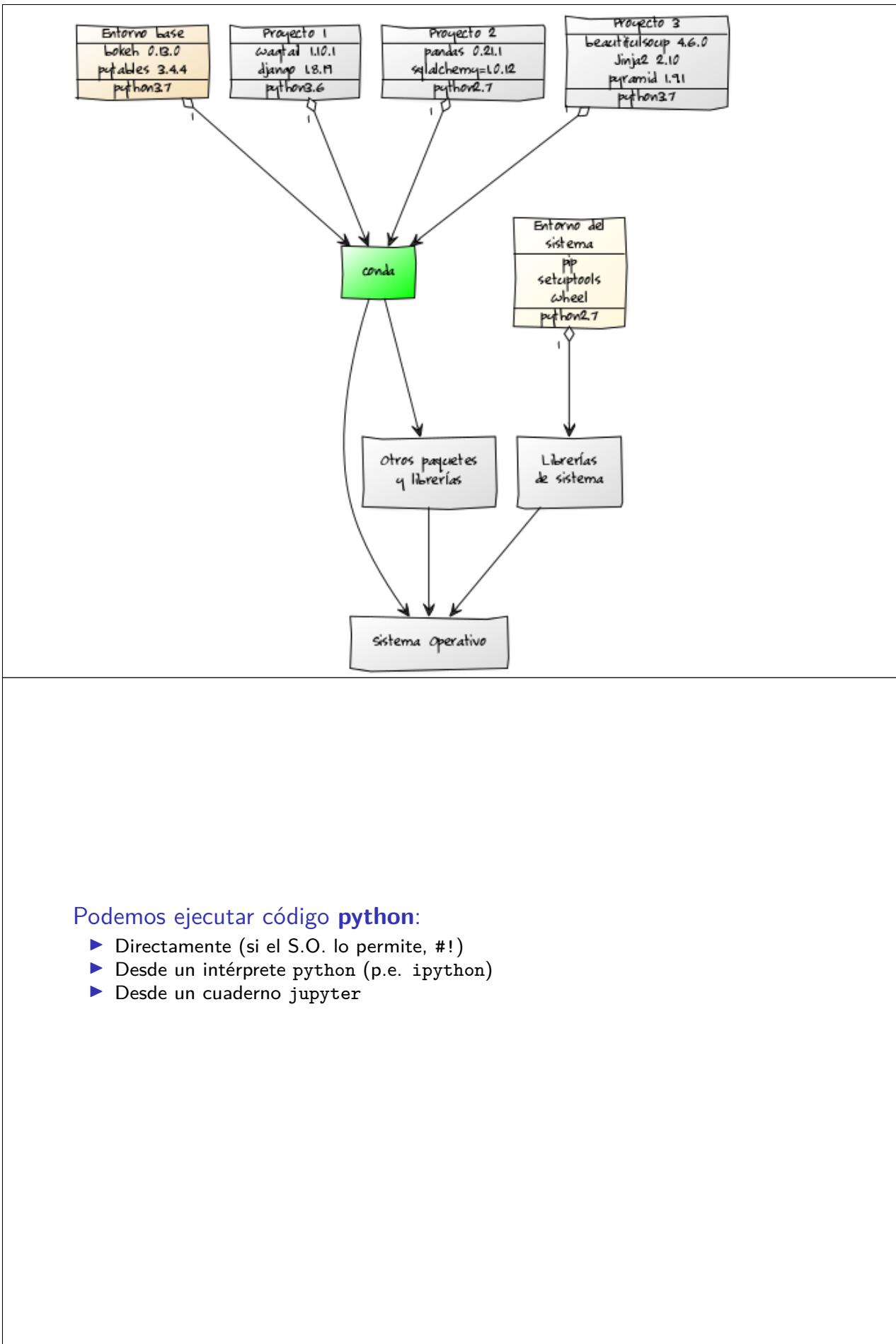


MY PYTHON ENVIRONMENT HAS BECOME SO DEGRADED  
THAT MY LAPTOP HAS BEEN DECLARED A SUPERFUND SITE.

¿Solución?

## Entornos virtuales

- ▶ Funcionan en un marco aislado y seguro
- ▶ Independientes entre sí
- ▶ Permiten especificar versiones de Python (2.7.10, 3.6.6, 3.6.7, ...)
- ▶ Facilmente reproducibles y exportables (**Portabilidad**)



### Podemos ejecutar código python:

- ▶ Directamente (si el S.O. lo permite, `#!`)
- ▶ Desde un intérprete python (p.e. ipython)
- ▶ Desde un cuaderno jupyter

## Conda

### Conda (anaconda/miniconda)

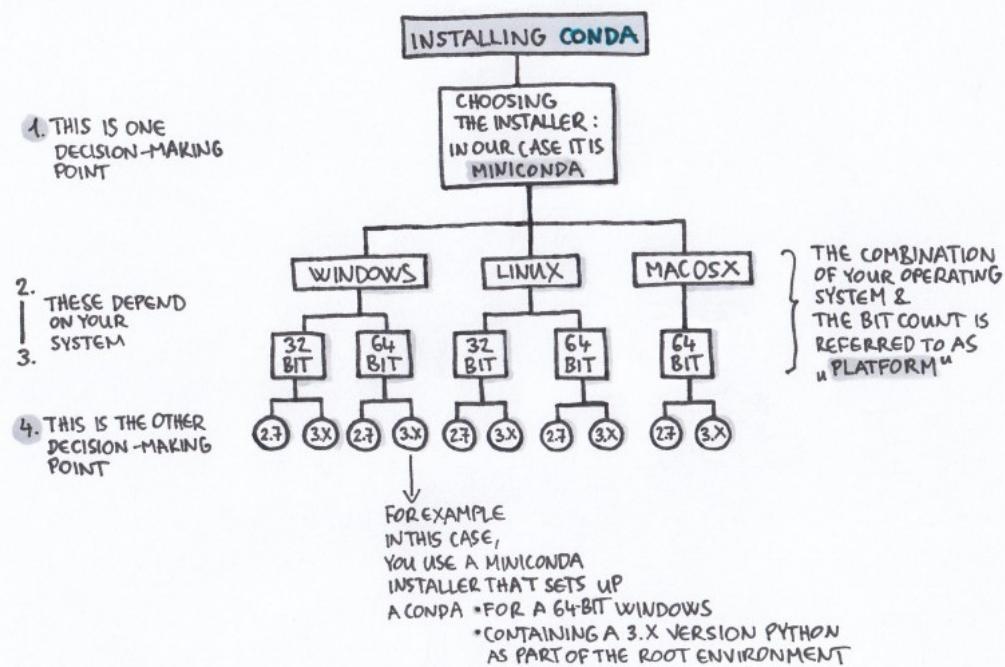
- ▶ Gestor de paquetes multi-lenguaje (python/R/...)
- ▶ Transparente: no instala ficheros fuera de su directorio
- ▶ Multipropósito, gestiona paquetes adicionales (p.e. git, librerías, ...)
- ▶ Coexistencia de entornos con diferentes librerías y versiones de python
- ▶ Por defecto nos encontraremos en el entorno base

Anaconda	Miniconda
~3GB disco	<400MB
> 200 librerías	base + dependencias
+ herramientas	ciclo distribución +rápido
IDE (Spyder + <b>VSCode</b> )	
Anaconda Navigator	sin interfaz gráfico
↑ tiempo instalación	

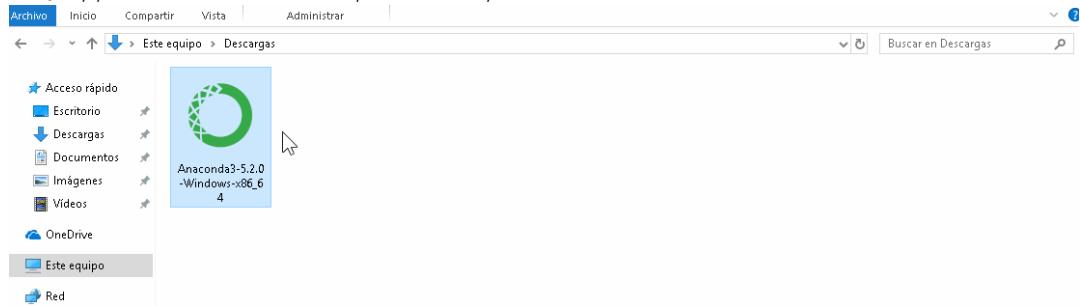
Alternativa: `python + pipenv`  
(más bajo nivel)

## Instalación

Determinar la plataforma sobre la que se va a instalar



<https://www.anaconda.com/download/>



La instalación puede tardar unos 5-10 minutos

## Práctica I

## Entorno virtual con Navigator

- ▶ Ejecutar Navigator y crear un nuevo entorno (distinto a base) con diferentes librerías instaladas, p.e.:

---

pandas	bs4
matplotlib	jupyter
scrapy	scipy
sqlalchemy	flask
jsonschema	bokeh

---

## Práctica II

### Entorno virtual en modo texto

- ▶ Ejecutar Anaconda prompt
- ▶ Verificar que conda está instalado:  
`(base) conda info`  
`(base) conda --help`

Crear un entorno virtual con pandas, matplotlib, jupyter y pyjstat.

- ▶ Inicializar el entorno virtual:

```
(base) conda create --name entorno-01  
(base) conda create --name entorno-02 python=2.7 --yes  
# crea entorno con paquetes preinstalados  
(base) conda create -n entorno-03 python=3.7 pandas scipy -y
```

- ▶ Acceder (activar) el entorno virtual:

```
(base) conda activate entorno-03  
(entorno-03) conda list # muestra paquetes instalados
```

Crear un entorno virtual con pandas, matplotlib, jupyter y pyjstat.

- ▶ Instalar librerías adicionales dentro del entorno

```
# instala librerías gestionadas por conda  
(entorno 03) conda install matplotlib jupyter --yes  
# instala paquetes no gestionados por conda desde PyPI (Python Package Index)  
(entorno-03) pip install -y pyjstat
```

- ▶ Exportar entorno virtual

```
# exportar definición del entorno  
(entorno-03) conda list --export > requirements.txt # solo dependencias  
(entorno-03) conda env export > entorno-03.yml # entorno + dependencias
```

**Distribuible y replicable**

- ▶ Acceder al entorno virtual y mostrar librerías instaladas

```
(base) conda activate entorno-03  
(entorno-03) conda list # muestra paquetes instalados
```

```
# instala paquetes adicionales  
(entorno-03) conda install jupyter -y
```

```
# instala paquetes desde PyPI, no gestionados por conda  
(entorno-03) pip install pyjstat  
(entorno-03) conda list
```

```
# instala paquetes desde github usando pip  
(entorno-03) pip install -e ^  
"git+https://github.com/predicador37/pyjstat.git#egg=pyjstat-git"
```

- ▶ Resumen comandos conda
- ▶ Guía de usuario pip

## Estructura del código

- ▶ No hay delimitadores de línea (tipo ';')
- ▶ Comentar código con #
- ▶ Jerarquía del código según nivel de indentación
- ▶ Indentar código con espacios
- ▶ Importar librerías al inicio
- ▶ Código agrupado con líneas en blanco
- ▶ **Recomendado:** Longitud de línea < 80 caracteres

- ▶ No hay delimitadores de línea (tipo ;)
- ▶ Comentar código con #

```
import os

home = os.path.expanduser('~/')  # comentario en linea
directorio_entorno = os.path.join(home,
                                   'Anaconda3',
                                   'envs',
                                   'entorno-03')

ficheros = []

# los comentarios comienzan con el caracter '#'
for f in os.listdir(directorio_entorno):
    if os.path.isfile(f):
        tamaño = os.stat(os.join(directorio_entorno), f).st_size
        ficheros.append((f, tamaño))
```

- ▶ Jerarquía del código según nivel de sangría (*indent*)
- ▶ Diferenciarlo con espacios <sup>1</sup>

```
import os

home = os.path.expanduser('~/') # directorio del usuario
directorio_entorno = os.path.join(
    home, 'Anaconda3', 'envs', 'entorno-03'
)
ficheros = []

# Busca ficheros y guarda (nombre, tamaño)
for f in os.listdir(directorio_entorno):
    if os.path.isfile(f):
        tamaño = os.stat(os.join(directorio_entorno), f).st_size
        ficheros.append((f, tamaño))
print(f)
print(tamaño)
```

<sup>1</sup> Generalmente con 4 espacios, no mezclar con TAB

- ▶ Importar librerías al inicio
- ▶ Código agrupado con líneas en blanco

```
import pathlib
from pathlib import Path

home = Path.home() # pathlib.Path.home()
directorio_entorno = home / 'Anaconda3' / 'envs' / 'entorno-03'
ficheros = [(f.name, f.stat().st_size)
            for f in directorio_entorno.iterdir()
            if f.is_file()]

cuaderno jupyter cuaderno jupyter (offline)
```

### Palabras reservadas

```
import builtins
import keyword

print(' ', '.join(keyword.kwlist))
False, None, True, and, as, assert, async, await, break, class,
continue, def, del, elif, else, except, finally, for, from,
global, if, import, in, is, lambda, nonlocal, not, or, pass,
raise, return, try, while, with, yield

print(dir(builtins))
['ArithmeticalError', 'AssertionError', 'AttributeError', 'BaseException',
'BlockingIOError', 'BrokenPipeError', 'BufferError', 'BytesWarning',
'ChildProcessError', 'ConnectionAbortedError', 'ConnectionError',
'ConnectionRefusedError', 'ConnectionResetError', 'DeprecationWarning',
'EOFError', 'Ellipsis', 'EnvironmentError', 'Exception', 'False',
'FileExistsError', 'FileNotFoundException', 'FloatingPointError', 'FutureWarning',
'GeneratorExit', 'IOError', 'ImportError', 'ImportWarning', 'IndentationError',
'IndexError', 'InterruptedError', 'IsADirectoryError', 'KeyError',
'KeyboardInterrupt', 'LookupError', 'MemoryError', 'ModuleNotFoundError',
'NameError', 'None', 'NotADirectoryError', 'NotImplemented',
'NotImplementedError', 'OSError', 'OverflowError', 'PendingDeprecationWarning',
'PermissionError', 'ProcessLookupError', 'RecursionError', 'ReferenceError',
'ResourceWarning', 'RuntimeError', 'RuntimeWarning', 'StopAsyncIteration',
'StopIteration', 'SyntaxError', 'SyntaxWarning', 'SystemError', 'SystemExit',
'TabError', 'TimeoutError', 'True', 'TypeError', 'UnboundLocalError',
'UnicodeDecodeError', 'UnicodeEncodeError', 'UnicodeError',
```

```
if __name__ == "__main__":
```

Se llama a `__main__()` cuando se ejecuta directamente:

```
(base) python circunferencia.py
import sys
import math
```

```
def area(radio):
    return math.pi * (radio ** 2)
```

```
def longitud(radio):
    return 2 * math.pi * radio
```

# entra aquí cuando se ejecuta directamente

```
if __name__ == "__main__":
    radio = float(sys.argv[1]) # sys.argv[] son los argumentos de entrada
    print(
        "La longitud de una circunferencia de radio {}cm es {:.2f}cm^2."
        .format(radio, longitud(radio)))
    print("El área de una circunferencia de radio {}cm es {:.2f}cm^2."
        .format(area(radio), radio))
```

- ▶ Un fichero python puede contener (entre otros) definiciones de constantes, variables, funciones o clases.
- ▶ Para organizar el código guardaremos los ficheros .py en un árbol de directorios:

```

principal/
    __init__.py
    practica.py
    recolector/
        __init__.py
        collector.py
        database.py
    conversor/
        __init__.py
        limpia.py
        procesa.py
    graficos/
        __init__.py
        graficos.py
        exportar.py
    string/
        __init__.py
        string.py

```

## Proyecto ejemplo

proyecto/	nombre del paquete
README.rst	descripción del proyecto
LICENSE	licencia
setup.py	distribución/empaquetado [2]
requirements.txt	descripción de las dependencias
entorno_conda.yml	descripción del entorno
ejemplo/__init__.py	el código en sí
ejemplo/core.py	" "
ejemplo/helpers.py	" "
docs/conf.py	documentación del proyecto
docs/index.rst	" "
tests/unitarios.py	funciones de test del proyecto
tests/funcionales.py	" "

<sup>2</sup> p.e. setuptools

## Práctica III

### Parte 1

- ▶ Abrir una consola ipython
  - ▶ In[n] identifica las entradas de comandos
  - ▶ comandos *mágicos* (solamente para ipython)
    - ▶ comienzan por el carácter '%'
    - ▶ ejemplo: %edit, %pylab
  - ▶ Ctrl+R activa la búsqueda en el historial
    - ▶ %hist, %history para visualizarlo
  - ▶ \_ guarda la salida del último comando <sup>3</sup>
- <sup>3</sup> \_ se usa también como variable de usar/tirar, p.e. cuando una función devuelve varios resultados, pero solamente estamos interesados en uno.
- consola online

- ▶ [TAB] completa (p.e. import st[TAB])
  - ▶ muy útil para ver los métodos/atributos de un módulo/clase

```
In[]: import st[TAB]  
In[]: import string
```

```
In[]: string.[TAB]  
      string.Formatter      string.ascii_uppercase  string.octdigits  
      string.Template       string.capwords        string.printable  
      string.ascii_letters   string.digits         string.punctuation  
      string.ascii_lowercase string.hexdigits     string.whitespace
```

```
In[1]: ? # muestra ayuda del intérprete ipython  
In[2]: ? dict # muestra ayuda breve de un método/clase/...  
In[3]: help(dict) # muestra ayuda completa  
In[4]: import math  
In[5]: math.sq[TAB]  
In[5]: math.sqrt(94)  
In[6]: a = _ # recupera última salida  
In[7]: dir(math) # muestra todos los métodos
```

- ▶ Convierte 67 grados a radianes
- ▶ Calcula la raíz cúbica del resultado
- ▶ Verifica si el resultado es mayor que  $14/13$

## Solución

```
import math

grados = math.radians(67)
math.pow(grados, 1/3) > 14/13

False
```

## Parte 2

- ▶ Importa el módulo numpy
  - ▶ import numpy as np
  - ▶ o bien (ipython): %pylab (no recomendado)
- ▶ Crea un array a de 1000 puntos entre [0, 1]
  - ▶ Ejecuta el método np.linspace
- ▶ Calcula el seno de  $2 \cdot \pi \cdot a$ 
  - ▶ Usa la constante np.pi
- ▶ Activa el modo gráfico integrado
  - ▶ %matplotlib inline
- ▶ Crea el gráfico de (a, seno(a))

## Solución

```
%matplotlib inline
import numpy as np
from matplotlib import pyplot as plt

a = np.linspace(0, 1, 1000)
b = sin(2*np.pi*a)

plt.plot(a, b)
```

## Práctica IV

Abrir el cuaderno de ejemplo y ejecutar línea por línea.  
cuaderno ejemplo offline

## Práctica V

### Exportar entorno virtual

```
# exportar definición del entorno

# solo dependencias
(entorno-01) conda list --export > requirements.txt

# entorno + dependencias
(entorno-01) conda env export > entorno-01.yml

# Desde otro sistema, restaurar el entorno
(base) conda env create -f entorno-01.yml

# O bien en la misma máquina, con otro nombre:
(entorno-01) conda list --export > requirements.txt
(entorno-01) deactivate
(base) conda env create --name entorno-11 --file requirements.txt
(base) activate entorno-11
(entorno-11) conda list

► Restaurar solo dependencias (requirements.txt): independiente de conda
```

## Recursos adicionales

- ▶ Guías de estilo:
  - ▶ PEP8
  - ▶ Guía de estilo de Google

#### Otros recursos en línea

- ▶ Librería standard de Python
- ▶ Guía de referencia de Python
- ▶ Guía para principiantes
- ▶ Stackoverflow
- ▶ Stackoverflow en español
- ▶ awesome

*Bonus: GIL y GC*

### *Reference counting*

- ▶ Usado por python para la gestión de memoria
- ▶ Cuenta las referencias a cada objeto creado
- ▶ Cuando la cuenta es 0, la memoria reservada se libera

```
import sys  
  
a = []  
b = a  
sys.getrefcount(a)  
3
```

### *GC: Garbage Collector*

Utiliza 2 algoritmos para la gestión de memoria:

- ▶ Reference counting: libera objetos sin referencias en un programa
  - ▶ ejecutado en “tiempo real”
- ▶ Referencias cíclicas: se ejecuta periódicamente

## Ejemplo

```
foo = []

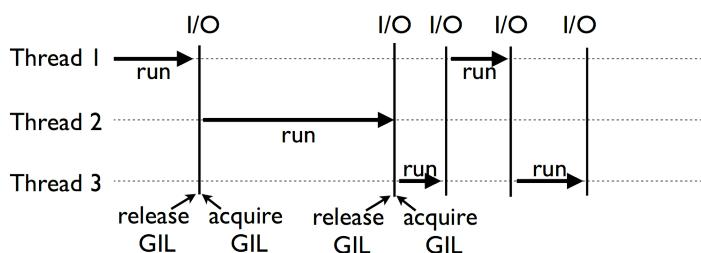
# 2 referencias, 1 de la variable foo y una de la llamada a getrefcount
print(sys.getrefcount(foo))

def bar(a):
    # 4 referencias
    # variable foo, argumento a función, getrefcount + pila interna de python
    print(sys.getrefcount(a))

bar(foo)
# 2 referencias, se limpian las propias de la función
print(sys.getrefcount(foo))
```

## GIL: Global Interpreter Lock

- ▶ Afecta a CPython
- ▶ Evita que la variable de conteo de referencias entre en condición de carrera
- ▶ Restringe la ejecución de código concurrente
  - ▶ ejecución secuencial dentro del intérprete
  - ▶ *multithreading*: aplicaciones no limitadas por CPU (p.e. I/O)
  - ▶ *multiprocessing*: varios intérpretes concurrentemente



### Ejemplo práctico: programa limitado por CPU, ejecución secuencial

```
# single_threaded.py
import time

COUNT = 50000000

def countdown(n):
    while n>0:
        n -= 1

start = time.time()
countdown(COUNT)
end = time.time()

print('Tiempo de ejecución {:.2f} segundos'.format(end - start))
$ python single_threaded.py
Tiempo de ejecución 2.30 segundos
```

### Programa limitado por CPU, ejecución concurrente (multithread)

```
# multi_threaded.py
import time
from threading import Thread

COUNT = 50000000

def countdown(n):
    while n>0:
        n -= 1

thread_1 = Thread(target=countdown, args=(COUNT//2,))
thread_2 = Thread(target=countdown, args=(COUNT//2,))
start = time.time()
thread_1.start()
thread_2.start()
thread_1.join()
thread_2.join()
end = time.time()

print('Tiempo de ejecución {:.2f} segundos'.format(end - start))
$ python multi_threaded.py
Tiempo de ejecución 2.85 segundos
```

## Programa limitado por CPU, ejecución concurrente (`multiprocess`)

```
# multi_process.py
import time
from multiprocessing import Pool

COUNT = 50000000

def countdown(n):
    while n>0:
        n -= 1

start = time.time()
with Pool(2) as p: # bypass del GIL
    p.map(countdown, 2* [COUNT//2])
end = time.time()

print('Tiempo de ejecución {:.2f} segundos'.format(end - start))
$ python multi_threaded.py
Tiempo de ejecución 1.32 segundos
```

cuaderno jupyter (offline)

cuaderno jupyter

# Python para Análisis de datos: Introducción

## Sesión 2

Miguel Expósito Martín ([exposito\\_m@cantabria.es](mailto:exposito_m@cantabria.es))

17 Octubre 2018

variables, estructuras de datos y de control, expresiones, sentencias y funciones

variables

expresiones

ejecución condicional

funciones

bucles

strings

lists

dicts

tuples & files

variables, estructuras de datos y de control, expresiones,  
sentencias y funciones

variables

¿qué es una constante?

un valor fijo, que nunca cambia

```
print(123)
print(15.2)
print('curso python')
pi = 3.14
print(pi)
```

recordad: ¡¡ojo con las palabras reservadas!!

```
False class return is finally None if for lambda continue
True def from while nonlocal and del global not with
as elif try or yield assert else import pass break except in raise
```

## ¿qué es una variable?

una variable es una posición de memoria a la que se asocia un nombre o etiqueta  
se les asigna valores con el operador =

```
x = 12.2
print(x)
y = 'variable'
x = 100
print(x)
```

## nomenclatura

- ▶ deben comenzar por una letra o guion bajo \_
- ▶ pueden contener letras, números y guiones bajos
- ▶ son sensibles a mayúsculas/minúsculas

## declaraciones

```
x = 1 # declaración de asignación  
x = x + 1 # asignación de expresión  
print(x) # declaración print  
¿cuáles son las variables, operadores, constantes, funciones?
```

## declaraciones de asignación

una declaración de asignación consiste en una expresión a la derecha del igual y una variable a la izquierda donde almacenar el resultado

```
x = 3.9 * x * ( 1 - x )
```

## expresiones

### expresiones numéricas

operador	operación
+	suma
-	resta
/	multiplicación
/	división
%	potenciación
	resto

## ejemplos

```
x = 2
x = x + 2
print(x)
y = 430 * 11
print(y)
z = y / 1000
print(z)
j = 26
k = j % 5
print(k)
```

## precedencia de operadores

1. paréntesis
2. potenciación
3. multiplicación, división y resto
4. suma y resta
5. de izquierda a derecha

```
x = 1 + 2 ** 3 / 4 * ( 5 + 4 )
```

## tipos

toda variable en Python tiene un tipo

Python sabe la diferencia entre `number` y `string`

```
x = 5 + 8
print(x)
type(x)
greeting = 'hola' + ' ' + 'Miguel'
print(greeting)
type(greeting)
```

con cadenas de texto, + concatena

## tipos

algunas operaciones están prohibidas...

```
greeting = 'hola' + ' ' + 'Miguel'
greeting = greeting + 1
¡no se puede sumar 1 a una cadena de texto!
```

## tipos numéricos

- ▶ enteros
- ▶ coma flotante

```
x = 1
type(x)
y = 48.9
type(y)
z = 1.0
type(z)
```

## conversiones de tipo

en una expresión, int se convierte implícitamente a float

conversiones explícitas: int() y float()

```
print(float(24) + 50)
i = 35
type(i)
f = float(i)
print(f)
type(f)
```

## conversiones de tipo

muy utilizada en logging:

```
msg = 'Número total de ocurrencias: '
count = 2
print(msg + str(count))
```

también se puede hacer lo contrario:

```
value= '555'
print(int(value) + 5)
```

## división entera

la división entera produce un resultado en punto flotante

```
print(10 / 2)
```

## entrada por teclado

```
input() devuelve un string
name = input('¿Cómo te llamas?')
print('Bienvenido', name)

age = input('Edad: ')
age_to_retirement = 65 - int(age)
print('Te quedan', age_to_retirement, 'años para jubilarte')
```

## comentarios

Python ignora cualquier cosa después de una almohadilla (#)

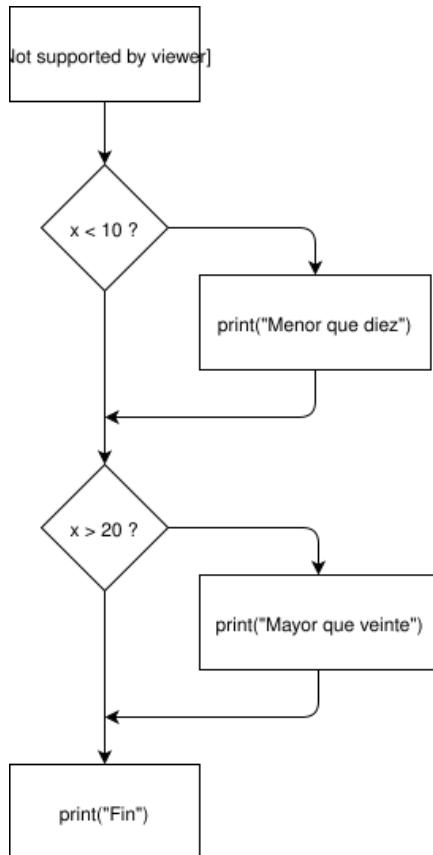
Los comentarios sirven para:

- ▶ describir lo que va a pasar en el código
- ▶ documentar quién escribió esa parte del código
- ▶ deshabilitar temporalmente una línea de código (\*)

## ejercicio

*escribir un programa que obtenga del usuario el número de horas trabajadas y el coste por hora y que calcule la paga de un empleado*

ejecución condicional



```

x = 7
if x < 10:
    print('Menor que diez')
if x > 20:
    print('Mayor que veinte')
print('Fin')

```

## operadores de comparación

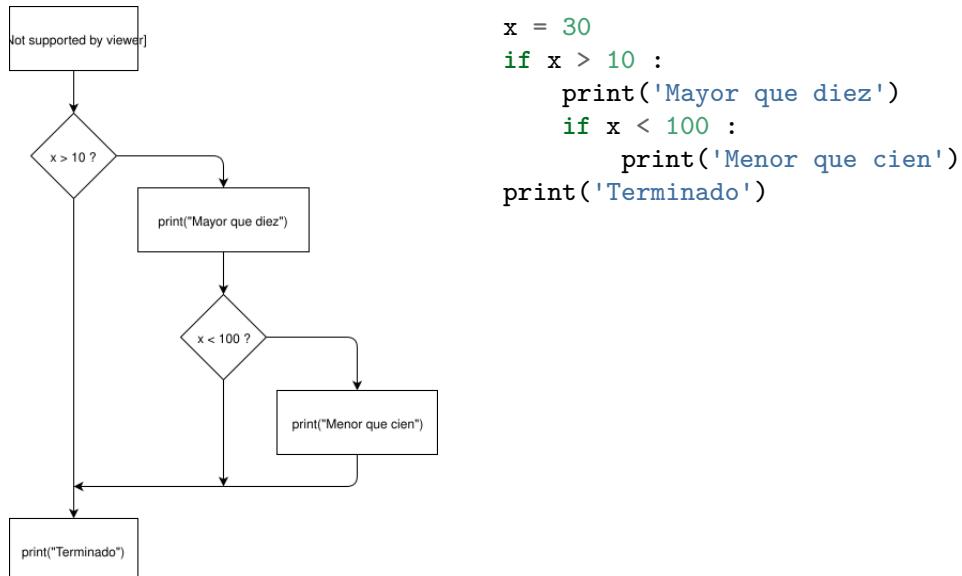
operador	operación
<	menor que
<=	menor o igual que
==	igual a
>=	mayor o igual que
>	mayor que
!=	distinto de

las expresiones que utilizan estas operaciones se evalúan a los valores True/False  
estos operadores no cambian el valor de las variables

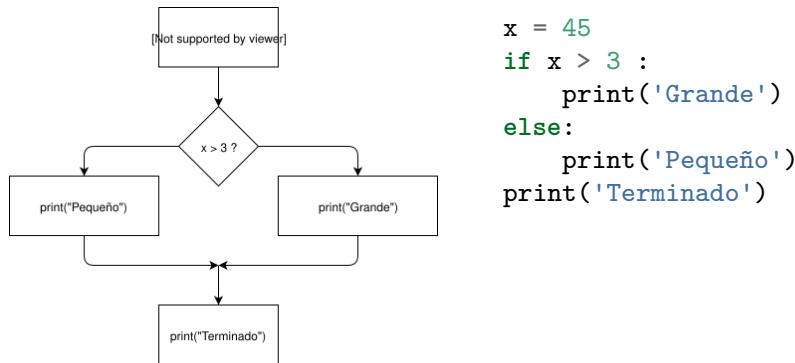
## sangría

- ▶ después de un `if` o un `for`, incrementar (después de `:`)
- ▶ mantener para indicar el ámbito del bloque (líneas afectadas por `if/for`)
- ▶ reducir al nivel del padre para indicar fin del bloque
- ▶ las líneas en blanco no afectan a la sangría
- ▶ los comentarios no afectan a la sangría

## decisiones anidadas

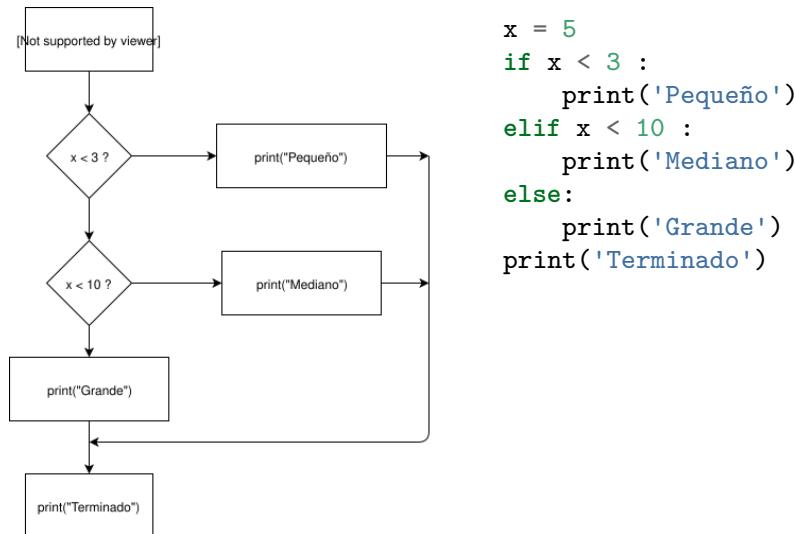


## decisiones con dos caminos



```
x = 45
if x > 3 :
    print('Grande')
else:
    print('Pequeño')
print('Terminado')
```

## decisiones multicamino



```
x = 5
if x < 3 :
    print('Pequeño')
elif x < 10 :
    print('Mediano')
else:
    print('Grande')
print('Terminado')
```

## ojo con liarse

```
if x < 2 :  
    print('Menor que 2')  
elif x >= 2 :  
    print('Mayor que 2')  
else :  
    print('Otra cosa')
```

## estructuras try / except

- ▶ se rodea una sección de código “peligrosa” con un `try / except`
- ▶ si el código en el `try` funciona, el `except` se ignora
- ▶ si el código en el `try` falla, se salta a la sección `except`

## ejemplo

```
astr = 'Hola Miguel'  
try:  
    istr = int(astr)  
except:  
    istr = -1  
print('Primero', istr)  
  
astr = '123'  
try:  
    istr = int(astr)  
except:  
    istr = -1  
print('Segundo', istr)
```

## ejercicio

*Reescribir el programa del cálculo de paga para pagarle todas las horas de más a partir de 40 a 1.5 veces el coste de la hora normal*

Ej: 45 horas y 10 €/hora: 475 €

## funciones

### definición

trozos de código reutilizable que toman argumentos de entrada, realizan algún tipo de cálculo y devuelven resultados

- ▶ se definen con `def` e indentando su contenido o cuerpo
- ▶ se invocan usando su nombre, paréntesis y argumentos

## ejemplo

```
def my_function():
    print('Hola')
    print('Adios')

my_function()
print('Pepe')
my_function()
```

## tipos

- ▶ built-in: parte de Python (print(), input(), int())
- ▶ definidas por el usuario

¡no usar nombre de función como nombre de variable!

## argumentos

- ▶ son valores que se pasan como entrada a la función en su llamada
- ▶ permiten dirigir a la función para que lleve a cabo distintos tipos de trabajo

```
print('Pepe')
```

## parámetros

- ▶ son variables utilizadas en la definición de la función
- ▶ permiten a la función acceder a sus argumentos para una llamada particular

```
def say_hello(lang):  
    if lang == 'es' :  
        print('Hola')  
    elif lang == 'fr' :  
        print('Salut')  
    else:  
        print('Hello')  
say_hello('fr')
```

## retorno

una función retorna un resultado con `return`

```
def say_hello(person):
    return "Hello " + str(person)

print(say_hello('Ana'))
```

## múltiples parámetros

- ▶ simplemente, se añaden
- ▶ ¡ojo con el número y orden de parámetros!

```
def add_three(a, b, c):
    sum = a + b + c
    return sum
x = add_three(3, 5, 7)
print(x)
```

## algunos consejos

- ▶ organizar el código
- ▶ DRY!
- ▶ divide y vencerás
- ▶ hacer bibliotecas con lo que más se usa

## bucles

## qué son

- ▶ los bucles permiten repetir la ejecución de un determinado paso
- ▶ contienen variables de iteración que cambian cada vez
- ▶ las variables de iteración suelen recorrer una secuencia de números

## ejemplo while (bucles indefinidos)

```
n = 5
while n > 0 :
    print(n)
    n = n - 1
print('Despegue!')
print(n)
```

¿qué hace esto?

```
n = 5
while n > 0 :
    print(n)
print('Despegue!')
print(n)
```

¿y esto?

```
n = 0
while n > 0 :
    print(n)
print('Despegue!')
print(n)
```

## salir de un bucle

```
break

while True :
    text= input('Introduce q para terminar: ')
    if text == 'q' :
        break
    print(text)
print('Hecho')
```

## salir de una iteración

```
continue

while True :
    text = input('Introduce q para terminar, c para otra oportunidad: ')
    if text == 'q' :
        break
    elif text == 'c':
        continue
    print(text)
print('Hecho')
```

## bucles definidos

- ▶ necesidad de recorrer una lista de items
- ▶ se ejecutan un número exacto de veces
- ▶ iteran sobre los miembros de un conjunto

```
for i in [5, 4, 3, 2, 1] :  
    print(i)  
print('Despegue!')
```

## también se pueden iterar strings

```
it_crowd = ['Izu', 'Saro', 'Seco']  
for it_guy in it_crowd :  
    print('Pregúntaselo a: ', it_guy)  
print('Hecho!')
```

de forma general

1. inicializar las variables
2. encontrar algo que hacer para cada ítem de forma separada, actualizando la variable
3. ver el resultado

ejercicio

*cuál es el máximo de la siguiente lista: 9, 41, 12, 3, 74, 15*

## conteos

```
numeros = [9, 41, 12, 3, 74, 15]
total = 0
for numero in numeros :
    total = total + 1
print('Hecho! el total es:', total)
```

## media aritmética

```
numeros = [9, 41, 12, 3, 74, 15]
total = 0
suma = 0
for numero in numeros :
    total = total + 1
    suma = suma + numero
print('Hecho! la media es:', suma / total)
```

## filtrado

```
numeros = [9, 41, 12, 3, 74, 15]
for numero in numeros :
    if numero > 20 :
        print('Mayor que 20')
print('Hecho! )
```

## is / is not

- ▶ utilizados en expresiones lógicas
- ▶ significa “es lo mismo que”
- ▶ similar a ==, pero más fuerte
- ▶ is devuelve True si dos variables apuntan al mismo objeto
- ▶ == devuelve True si los objetos a los que apuntan dos variables son iguales

```
x = None
if x is None :
    print('No hay valor para x')
```

## strings

### más sobre strings

- ▶ un string es una secuencia de caracteres
- ▶ se pueden usar comillas simples o dobles
- ▶ aunque contenga números, un string es un string
- ▶ el operador + significa concatenación
- ▶ se comparan con ==
- ▶ se pueden convertir strings a números con int() o float()
- ▶ en Python 3, todos los strings son unicode
- ▶ son **inmutables** (no se pueden modificar)

## indexado y rebanado

indexado: como un vector que empieza en 0

```
fruit = 'plátano'  
print fruit[3]  
print(len(fruit)) # longitud de la cadena  
print fruit[0:5] # slicing  
print fruit[1:] # slicing hasta el final
```

## iteración

```
fruit = 'plátano'  
for letra in fruit:  
    print(letra)
```

## in como operador lógico

```
fruit = 'plátano'  
'n' in fruit
```

## string library

- ▶ funciones por defecto
- ▶ se invocan añadiéndolas a la variable

**importante: no modifican el string original, sino que devuelven uno modificado**

## ejemplos

```
find() busca la primera ocurrencia de un substring  
fruit = plátano  
pos = fruit.find('ta')  
print(pos)
```

```
mayúsculas y minúsculas  
greeting = 'Hola Miguel'  
mins = greeting.lower()  
print(mins)  
mays = greeting.upper()  
print(mays)
```

## ejemplos

```
buscar reemplazar  
greeting = 'Hola Miguel'  
another_greeting = greeting.replace('Miguel', 'P')  
print(another_greeting)
```

```
quitar espacios en blanco  
(stripping)  
greeting = ' Hola Miguel '  
greeting.lstrip()  
greeting.rstrip()  
greeting.strip()
```

## ejemplos

prefijos

```
greeting = 'Que tenga un buen día'  
greeting.startswith('Que')
```

## ejercicio

```
From miguel.exposito@apps.cantabria.es Sat Jan 5 09:14:16 2008
```

## lists

### definición

[5, 89, 'Miguel']

- ▶ colecciones de elementos
- ▶ un elemento puede ser cualquier tipo de objeto
- ▶ una my\_list puede estar vacía
- ▶ son mutables: se puede cambiar un elemento usando el operador índice
- ▶ son **ordenadas**

## indexado, rebanado y longitud

```
my_list = [5, 12, 21, 40, 68]
print(my_list[1]) # indexado
my_list[1] = 7 # mutabilidad
print(my_list)
print(my_list[2:3]) # rebanado
print(len(my_list)) # longitud
```

## rangos

devuelven una my\_list de números en el intervalo [0,N-1]

```
print(range(4))
```

## concatenación

```
a = [1, 2, 3]
b = [4, 5, 6]
c = a + b
print(c)
```

## ordenación

```
a = [7, 4, 9, 1, 8]
print(a.sort())
```

## añadir y quitar elementos

```
my_list = list()
my_list.append('Miguel')
my_list.append('Paco')
my_list.append('Alberto')

pop quita el último elemento y lo devuelve
print(my_list.pop())
print(my_list)

también se puede especificar qué elemento (índice)
```

## otras funciones

```
len, max, min, sum

split rompe un string en una my_list de strings
my_list = 'my_list de cuatro palabras'
palabras = my_list.split()
print(palabras)

se puede especificar un delimitador
```

ejercicio

From miguel.exposito@apps.cantabria.es Sat Jan 5 09:14:16 2008

dicts

## definición

- ▶ también son colecciones de elementos
- ▶ permiten realizar operaciones similares a las de las bases de datos
- ▶ también llamados arrays asociativos, son la colección más potente en Python
- ▶ **no son ordenados**
- ▶ sus elementos no se indexan con un número, sino con una etiqueta de búsqueda

## ejemplo

```
ages = dict()
ages['miguel'] = 37
ages['pepito'] = 88
ages['carmen'] = 25
print(ages)

ages['pepito'] = ages['pepito'] + 2
print(ages)
```

## lists vs dicts

```
my_list = list()
my_list.append(21)
my_list.append(183)
print(my_list)
my_list[0] = 23
print(my_list)

my_dict = dict()
my_dict['edad'] = 21
my_dict['curso'] = 183
print(my_dict)
my_dict['edad'] = 23
print(my_dict)
```

## diccionarios con literales

```
shopping_list = {'patatas': 3, 'york': 100, 'yogur': 6}
```

se puede crear un my\_dict vacío así:

```
empty_dict = {}
```

## uso común: múltiples contadores

```
names = []
names['Paco'] = 1
names['Pepe'] = 1
print(names)
names['Paco'] = names['Paco'] + 1
print(names)
```

## cuidado cuando la clave no existe

si se utiliza una clave que no existe, aparecerá un error

Solución: get()

```
counts = []
counts['mesa'] = counts.get('mesa', 0) + 1
counts['silla'] = counts.get('silla', 0) + 1
print(counts)
counts['mesa'] = counts.get('mesa', 0) + 1
print(counts)
```

## iteración en diccionarios

```
counts = {'mesa': 2, 'silla': 4, 'reposapies': 3}
for key in counts:
    print(key, counts[key])

counts = {'mesa': 2, 'silla': 4, 'reposapies': 3}
for key, value in counts.items():
    print(key, value)
```

## obtener claves y valores

```
counts = {'mesa': 2, 'silla': 4, 'reposapies': 3}
print(counts.keys())
print(counts.values())
print(counts.items())
```

## contar palabras

- ▶ hacer split del texto en palabras
- ▶ iterar a lo largo de las palabras
- ▶ usar un my\_dict para contar cada palabra de forma independiente

## ejercicio

*En un lugar de La Mancha, de cuyo nombre no quiero acordarme...*

## ejercicio

```
providers = {
    'places': {
        'priority': 1,
        'eligible': False
    },
    'google': {
        'priority': 2,
        'eligible': True
    },
    'bing': {
        'priority': 4,
        'eligible': False
    },
    'mapbox': {
        'priority': 3,
        'eligible': True
    }
}
```

tuples & files

## tuples

un tipo de secuencia similar a una lista... pero inmutable

```
x = [9, 8, 7]           z = (5, 4, 3)  
x[2] = 6                z[2] = 0  
print(x)
```

## ¿qué no se puede hacer?

- ▶ sort
- ▶ append
- ▶ reverse

## lists vs tuples

- ▶ las tuplas son más eficientes (memoria y rendimiento)
- ▶ se pueden usar como un diccionario sin claves
- ▶ se suelen usar dentro de listas
- ▶ son comparables elemento a elemento

**no se pueden añadir ni eliminar elementos**

## asignación

```
(x,y) = (4, 'Pepe')  
print(y)
```

## ficheros

- ▶ un fichero de texto puede verse como una secuencia de líneas
- ▶ para abrirse, se crea un apuntador para manipularlo

```
my_file = open('prueba.txt')
for line in my_file:
    print(line.rstrip())
```

# Python para Análisis de datos: Introducción

## Sesión 3

Alejandro Villar ([avillar@ticnor.es](mailto:avillar@ticnor.es))

22 Octubre 2018

[Programación Orientada a Objetos](#)

[POO en Python](#)

[Ejercicios](#)

## Programación Orientada a Objetos

### Diferentes paradigmas de programación

- ▶ **Orientado a objetos**
- ▶ Procedural
- ▶ Imperativo
- ▶ Funcional (<100%)

## Qué es

- ▶ Concepto central: **objeto**
  - ▶ Datos (atributos o campos)
  - ▶ Comportamiento (métodos)

## Objeto: Coche de Pedro

- ▶ Datos:
  - ▶ Color: Gris
  - ▶ Año: 2012
- ▶ Comportamiento:
  - ▶ Arrancar
  - ▶ Parar
  - ▶ Acelerar
  - ▶ Frenar

## Clases y objetos

- ▶ Clase: Definición a partir de la que crear objetos (*contrato*).
  - ▶ Normalmente, mayúscula inicial
- ▶ Objeto: Instancia individual de una clase.
  - ▶ Normalmente, minúscula inicial (son variables)

## Características de OOP

- ▶ Herencia
- ▶ Encapsulamiento
- ▶ Abstracción
- ▶ Polimorfismo
- ▶ Composición

## Herencia

- ▶ Jerarquía: atributos y comportamiento
- ▶ La clase *Perro* hereda de *Mamífero* y ésta de *Animal*.
- ▶ Pero también de *Cuadrúpedo* <- Herencia múltiple
- ▶ Las subclases completan o modifican a sus clases ancestro

## Encapsulamiento

- ▶ Limitar acceso al estado interno del objeto
- ▶ Campos y métodos públicos y privados
  - ▶ ¡Python no limita el acceso! -> Convenio

## Abstracción

- ▶ No nos preocupamos de los detalles de funcionamiento interno
- ▶ *Confiamos* en que el objeto hará lo que le pedimos correctamente
- ▶ Ejemplo:
  - ▶ En Coche.arrancar() nos da igual el sistema de arranque interno, queremos que el coche pase a estar arrancado

## Polimorfismo

- ▶ Tratamos a una familia de objetos de la misma forma
- ▶ Ejemplo 1: lo que haga mensaje.enviar() dependerá de su clase
  - ▶ SMS
  - ▶ Email
  - ▶ MensajeWhatsApp
- ▶ Ejemplo 2: si la clase Mensaje tiene campos “asunto” y “contenido”, podemos implementar Bandeja.recibir(mensaje) sin importarnos la subclase concreta de mensaje.

## Composición

- ▶ Combinación de varios objetos con diferentes características
- ▶ Ejemplo:
  - ▶ Coche contiene objetos de clase Motor, Freno, Rueda, etc.
  - ▶ coche.frenar() llama a frenos\_delante.activar() y frenos\_detrás.activar()

POO en Python

## Primero, un repaso

- ▶ dict: estructura clave -> valor
  - ▶ Claves únicas
  - ▶ Valores: cualquier cosa
  - ▶ Permite anidamiento

```
midic = {  
    'clave1': 'valor1',  
    'clave2': 23,  
    'clave3': [1, 2, 3],  
    'clave4': {  
        'subclave1': [1, 2]  
    }  
}
```

La clase más simple:

```
class MiClase:  
    pass
```

Una clase más completa:

```
class Persona:  
    """Una clase para almacenar datos de personas"""  
  
    nombre = None  
    apellidos = None  
    email = None  
  
    def saludar(self):  
        print("Hola, {0} {1}".format(self.nombre, self.apellidos))  
  
    def linea_email(self):  
        return "{1}, {0} <{2}>".format(self.nombre, self.apellidos, self.email)
```

```
>>> p = Persona()
>>> p.nombre = 'Alejandro'
>>> p.apellidos = 'Villar'
>>> p.email = 'avillar@ticnor.es'
>>> p.saludar()
Hola, Alejandro Villar
>>> print(p.linea_email())
Villar, Alejandro <avillar@ticnor.es>
```

## self ()

- ▶ Los métodos dentro de una clase tienen un argumento inicial: `self`
- ▶ Ese argumento representa a la **instancia** (objeto) que lo llama
- ▶ Python automáticamente lo añade a los argumentos de llamada
- ▶ Necesario para modificar atributos de la instancia.

## self (II)

```
def saludar(self):
    print("Hola, {0} {1}".format(self.nombre, self.apellidos))
>>> p.saludar()  # No hay self
```

## Constructor

Realizar tareas de inicialización, según se crea la clase.

- ▶ Sin constructor (por defecto)
- ▶ Constructor sin argumentos.
- ▶ Constructor con argumentos.

Constructor sin argumentos

```
class Persona:  
    def __init__(self):  
        self.nombre = 'Pendiente'  
        print("Persona inicializada")  
  
>>> p = Persona()  
Persona inicializada  
>>> p.nombre  
'Pendiente'
```

Constructor con argumentos

```
class Persona:  
    def __init__(self, nombre, apellidos="Sin apellidos"):  
        self.nombre = nombre  
        self.apellidos = apellidos  
        print("Persona inicializada")  
  
    def saludar(self):  
        print("Hola, {0} {1}".format(self.nombre, self.apellidos))  
  
>>> p1 = Persona('Pedro')  
Persona inicializada  
>>> p2 = Persona('Juan', 'Pérez')  
Persona inicializada  
>>> p1.saludar()  
Hola, Pedro Sin apellidos  
>>> p2.saludar()  
Hola, Juan Pérez
```

## Variables de clase y de instancia

- ▶ De clase: compartida por todas las instancias
- ▶ De instancia: única para cada instancia

```
class Coche:  
    ruedas = 4          # Variable de clase  
  
    def __init__(self, marca):  
        self.marca = marca  # Variable de instancia
```

```
>>> honda = Coche("Honda")  
>>> honda.ruedas  
4  
>>> Coche.ruedas  
4  
>>> honda.marca  
'Honda'  
>>> Coche.marca  
Traceback (most recent call last):  
  File "<stdin>", line 1, in <module>  
AttributeError: type object 'Coche' has no attribute 'marca'
```

## Variables y métodos “privados”

```
class Empleado:  
    estado = 'Ocupado'    # Pública  
    __empresa = None      # Privada  
    __historial = []      # Privada  
  
    def set_empresa(self, empresa):    # Público  
        self.__empresa = empresa  
        self.__actualizar_historial()  
  
    def __actualizar_historial(self): # Privado  
        self.__historial.append(self.__empresa)
```

## Herencia (I)

```
class Persona: # Igual que class Persona(object)  
    pass  
  
class Empleado(Persona):  
    pass  
  
class Multifuncion(Impresora, Escaner):  
    pass
```

## Herencia (II)

- ▶ Todas las clases heredan, en última instancia, de `object`
- ▶ Las subclases heredan atributos y métodos de clases ancestro (superclases)
- ▶ Herencia múltiple: se puede heredar de varias clases
  - ▶ MRO (Method resolution order)

## Herencia (III)

```
>>> e = Empleado()
>>> isinstance(e, Empleado)
True
>>> isinstance(e, Persona)
True
>>> issubclass(Empleado, Persona)
True
>>> mf = Multifucion()
>>> isinstance(mf, Multifucion)
True
>>> isinstance(mf, Impresora)
True
>>> isinstance(mf, Persona)
False
```

## Herencia (IV)

### Overriding de métodos

```
class Transformador:  
    def transformar(self, texto):  
        return texto  
  
class Mayusculas(Transformador):  
    def transformar(self, texto):  
        return texto.upper()
```

## Herencia (V)

- ▶ `super()` permite llamar a métodos definidos en las clases ancestro

```
class Mamifero:  
    def __init__(self):  
        print("Soy un mamífero")  
    def comer(self, comida):  
        print("Como {}".format(comida))  
  
class Raton(Mamifero):  
    def __init__(self):  
        super().__init__()  
        print("Soy un ratón")  
    def comer(self, comida):  
        super().comer(comida)  
        print("y lo royo") # o roo o roigo
```

## Herencia (VI)

```
class A:  
    def ping(self):  
        print("Ping - A")  
  
class B:  
    def ping(self):  
        print("Ping - B")  
  
class BA(B, A):  
    def ping(self):  
        super().ping() # Llama a B.ping  
        A.ping(self) # Llama a A.ping
```

## Herencia (VII)

### *Linearization*

```
class A:  
    def ping(self):  
        print("Ping - A")  
  
class B(A):  
    def ping(self):  
        print("Ping - B")  
        super().ping()  
  
class C(A):  
    def ping(self):  
        print("Ping - C")  
        super().ping()  
  
class D(B, C):  
    def ping(self):  
        print("Ping - D")  
        super().ping()
```

## Ejercicios

1. Definir una jerarquía de clases para alimentos, partiendo de la clase `Alimento`.
2. Escribir una clase con un método para obtener el siguiente número de Fibonacci, habiéndole pasado los dos primeros números como argumentos del constructor.

3. Definir una clase **Forma** que tenga atributos de posición (**x** e **y**) y un método **descripcion()** que escribirá en pantalla la información de la forma.

  - ▶ Crear las subclases **Rectangulo** y **Circulo**, que aceptarán argumentos de su tamaño en el constructor, y reimplementarán **descripcion()**.
  - ▶ Crear un programa que genere varias formas y muestre sus descripciones.

# Python para Análisis de datos: Introducción

## Sesión 4

Alejandro Villar (avillar@ticnor.es)

23 Octubre 2018

Expresiones regulares

Expresiones regulares en Python

Web scraping

Brevísima intro a HTML

Extraer datos con Pandas

BeautifulSoup

Scrapy

## Expresiones regulares

### Qué es una Expresión regular

- ▶ Expresión regular (regex, regexp): Patrón de búsqueda de texto
- ▶ Ejemplo:
  - ▶ `b{2,5}` -> “Entre 2 y 5 veces ‘b’ ”
  - ▶ `\. +` -> “Un punto seguido de uno o más espacios”

## Conceptos básicos (I)

- ▶ Alternación (una cosa u otra)
  - ▶ Ejemplo: gris|verde
- ▶ Agrupación (juntas cosas)
  - ▶ Ejemplo: s(o|ó)lo
- ▶ Comodín (el punto):
  - ▶ Ejemplo: s.lo
- ▶ Caracteres especiales: ¡escapar con \'!

## Conceptos básicos (II)

- ▶ Cuantificación (una cosa n veces)
  - ▶ ? = "cero o una veces"
  - ▶ \* = "cero o más veces"
  - ▶ + = "una o más veces"
  - ▶ {n} = "n veces"
  - ▶ {n,} = "n o más veces"
  - ▶ {n,m} = "entre n y m veces"

$abc? = ab / abc$   
 $abc^+ = abc / abcc / abccc / \dots$   
 $abc^* = ab / abc / abcc / \dots$   
 $abc\{2,3\} = abcc / abccc$

### Conceptos básicos (III)

- ▶ Clases de caracteres:
  - ▶  $[abc]$  = “‘a’ o ‘b’ o ‘c’”
  - ▶  $[a-z]$  = “carácter de la ‘a’ a la ‘z’”
  - ▶  $[a-zA-Z]$  = “‘a’ a la ‘z’ o ‘0’ a ‘9’”
  - ▶  $[^a-zA-Z]$  = “carácter que no es de la ‘a’ a la ‘z’”

[0-2] [3-5]  
[a-f]?[jk1]  
[gq][^u]  
[ab]+[cd]?

## Conceptos básicos (V)

- ▶ Clases de caracteres:
  - ▶ \s = [ \t\n\r\f] = espacio
  - ▶ \d = [0-9] = dígito
  - ▶ \w = [A-Za-z0-9\_] = carácter de palabra
  - ▶ En mayúsculas -> negados
- ▶ Otros caracteres especiales:
  - ▶ ^ = "principio de línea"
  - ▶ \$ = "final de línea"
  - ▶ \b = "límite de palabra"

## Ejemplos

```
H[aäe]nse1
;Go+1!
(No m|M)e gusta Python
Mi [mp]adre tiene \d+ años
[a-zA-Z]{4,8}
^(python|java)$
```

## Grupos y capturas

- ▶ Los paréntesis crean grupos que el motor “captura”
- ▶ Ejemplo:
  - ▶ Nueva (York|Inglaterra|Montaña)
  - ▶ ((Facultad|Escuela( Técnica Superior)?)) de  
(Medicina|Enfermería|Ingeniería)
- ▶ Grupos capturados -> Extracción, reemplazo, etc.

## Problemas (I)

- ▶ Difíciles de entender para humanos (*write-only*)
  - ▶ <https://regex101.com/>
- ▶ No valen para todo
  - ▶ Sólo lenguajes regulares
  - ▶ Pueden llegar a ser muy complejas

## Problemas (II)

Validación de email:

```
\A(?:[a-z0-9!#$%&'*+\=/=?^_`{|}~-]+(?:\.\[a-z0-9!#$%&'*+\=/=?^_`{|}~-]+)*|"(\?:\[\\x0
```

## Expresiones regulares en Python

Python soporta nativamente expresiones regulares mediante el módulo `re`.

```
>>> import re
>>> r = re.compile('a[bc]')
>>> r.match('ab')
<_sre.SRE_Match object; span=(0, 2), match='ab'>
```

## Módulo re (I)

- ▶ `re.compile(patron, flags=0)` -> Compila expresiones
- ▶ `re.search(patron, string, flags=0)` -> Busca patron en cualquier sitio del string
- ▶ `re.match(patron, string, flags=0)` -> Busca patron **al principio** del string
- ▶ `re.fullmatch(patron, string, flags=0)` -> Comprueba que string concuerda completamente con patron

## Módulo re (II)

- ▶ `re.split(patron, string, max=0, flags=0)` -> Divide string cuando se encuentra patron
- ▶ `re.findall(patron, string, flags=0)` -> Devuelve una lista con todas las coincidencias de patron en string
- ▶ `re.sub(patron, reempl, string, flags=0)` -> Reemplaza las apariciones de patron en string

## Módulo re (III)

- ▶ Flags
  - ▶ `re.I` / `re.IGNORECASE`
  - ▶ `re.M` / `re.MULTILINE`
  - ▶ `re.S` / `re.DOTALL`
- ▶ Ejemplo de uso:  
`re.compile(r'[a-z]+', re.I | re.S)`

## Módulo re (IV)

- ▶ `re.search`, `re.match` y `re.fullmatch` devuelven `None` o un objeto `Match`
  - ▶ `Match.group(num)` -> grupo capturado (0 = todo el *match*, resto numerados)
  - ▶ `Match.groups()` -> todos los grupos
  - ▶ `Match.start()` y `Match.end()` -> Inicio y final de coincidencia (pueden aceptar el númerodel grupo).

## Ejemplos

```
re.sub(r'\. {2,}', '. ' texto)
re.sub(r'(Fernando|rancisco)', r'Julián (antes \1)', texto, re.I)
re.findall(r'((\d+),)*(\d+)(\.(\\d+))?', texto)
re.split(r'\\.\\s*', texto)
```

## Otras características avanzadas

- ▶ *Backreferences*
- ▶ Grupos con nombre
- ▶ *Lookahead* y *Lookbehind*
- ▶ Condicionales
- ▶ ...

<https://www.regular-expressions.info/>

## Ejercicios

1. Escribir una función que compruebe que una cadena de caracteres sólo tiene caracteres de la a a la f, con un mínimo de 8.
2. Escribir una función que compruebe que todas las iniciales de una frase son mayúscula.
3. Escribir una función que compruebe que una cadena empieza por a, termina por b, y por el medio tiene al menos una o.

## Ejercicios

4. Escribir una función que reemplace los espacios múltiples en una cadena de caracteres por un espacio sencillo.
5. Escribir una función que divida una cadena en varias, haciendo corte cuando aparezcan @, | o ; al menos 2 veces seguidas.
6. Escribir una función que encuentre minúsculas seguidas de punto y uno o varios espacios.

## Ejercicios

7. Escribir una función que encuentre todas las palabras que empiezan por a y terminan por o en un texto, sin importar mayúsculas/minúsculas.
8. Escribir una función que reemplace números con el formato 1,234,567.89 por 1.234.567,89 en un texto.

## Web scraping

## Qué es

Extracción de datos desde páginas web.

## Utilidades

- ▶ Extracción de datos no estructurados (p.e., sin APIs de datos) o con formatos incorrectos (páginas con errores de sintaxis)
- ▶ Indexación ( motores de búsqueda)
- ▶ Minería de datos
- ▶ Monitorización (precios, cambios, etc.)

## Ejemplo

1. Definir punto de entrada (p.e., portada).
2. Rastrear elementos con contenido a extraer (p.e., listado de productos)
3. Localizar enlaces a siguientes páginas -> Añadir a cola de rastreo
4. Pasar a siguiente página en la cola y volver a 2

The screenshot shows the Google Scholar search interface. The search term 'einstein' is entered in the search bar. The results are filtered to show 'Articles'. There are 1,810,000 results found in 0.07 seconds.

On the left, there are filters for time range (Any time, Since 2018, Since 2017, Since 2014, Custom range...), sorting options (Sort by relevance, Sort by date), and checkboxes for 'include patents' and 'include citations'. A 'Create alert' button is also present.

The main results area displays several search results, each with a title, author(s), publication details, and citation information. The first result is highlighted with a red border:

**Can quantum-mechanical description of physical reality be considered complete?**  
A Einstein, B Podolsky, N Rosen - Physical review, 1935 - APS  
In a complete theory there is an element corresponding to each element of reality. A sufficient condition for the reality of a physical quantity is the possibility of predicting it with certainty, without disturbing the system. In quantum mechanics in the case of two physical ...  
☆ 99 Cited by 17189 Related articles All 112 versions

Below this result, there are three more results, each with a red border:

[CITATION] A. Einstein, B. Podolsky, and N. Rosen, Phys. Rev. 47, 777 (1935).  
A Einstein - Phys. Rev., 1935  
☆ 99 Cited by 1409 Related articles

[book] Ideas and opinions  
A Einstein, C Seelig, S Bargmann, I Unna, B Wolff - 1954 - ias.ac.in  
" Bear in mind that the wonderful things you learn in your schools are the work of many generations, produced by enthusiastic effort and infinite labor in every country of the world. All this is put into your hands as your inheritance in order that you may receive it, honor it ...  
☆ 99 Cited by 1913 Related articles All 7 versions

[book] Investigations on the Theory of the Brownian Movement  
A Einstein - 1956 - books.google.com  
The "Brownian movement" was first described in 1828 by the botanist Robert Brown. While investigating the pollen of several different plants, he observed that pollen dispersed in water in a great number of small particles which he perceived to be in uninterrupted and ...  
☆ 99 Cited by 4282 Related articles All 4 versions

At the bottom right, there is a navigation bar with the text 'Goooooooooooooogle >' and page numbers 1 through 10, followed by a 'Next' button.

## Brevísima intro a HTML

- ▶ HTML: lenguaje de marcado (*markup language*)
- ▶ Especificación del W3C
- ▶ Base de la web
- ▶ Árbol de <elementos> anidados
- ▶ Los elementos pueden tener atributos, contenido textual y comentarios
- ▶ El espacio extra se ignora

```
<html>
  <head>
    <title>Mi HTML</title>
  </head>
  <body>
    <h1 id="titulo-pagina">Esto es un título</h1>
    <p class="texto-normal">Esto es un párrafo</p>
  </body>
</html>
```

## Elementos importantes

- ▶ html
  - ▶ head
    - ▶ title
  - ▶ body
    - ▶ div
    - ▶ span
    - ▶ p
    - ▶ a
    - ▶ img
    - ▶ h1, h2, h3..., h6

## Tablas

- ▶ **table** (tabla)
  - ▶ **thead** (grupo de cabecera, opcional)
  - ▶ **tbody** (grupo de cuerpo, opcional)
  - ▶ **tr** (fila)
  - ▶ **th** (celda de cabecera)
  - ▶ **td** (celda de contenido)

## Atributos importantes

- ▶ **id** (único en el documento)
- ▶ **class**
- ▶ **href**
- ▶ **src**

## Selectores CSS (I)

- ▶ Sintaxis:
  - ▶ elemento
  - ▶ #id
  - ▶ .clase
  - ▶ [atributo]
  - ▶ [atributo="valor"]

## Selectores CSS (II)

- ▶ Se pueden anidar y combinar:
  - ▶ body h1
  - ▶ p.texto-normal
  - ▶ body > p
  - ▶ p a[href]

```
<html>
  <head>
    <title>Mi HTML</title>
  </head>
  <body>
    <h1 id="titulo-pagina">Esto es un título</h1>
    <h2>Subtítulo</h2>
    <p class="texto-normal">
      Esto es un párrafo
      
      <a href="/" class="back-link"><span class="enfasis">Volver</span></a>
    </p>
    <div class="pie">
      Copyright Ticnor 2018.
    </div>
  </body>
</html>
```

## Extraer datos con Pandas

## Pandas

- ▶ Pandas es un módulo para trabajar con *tablas* (`DataFrame`) de datos.
- ▶ Ofrece métodos para crear `DataFrames` desde diversas fuentes (CSV, Excel, Web...).

## Ejemplo

```
>>> import pandas as pd
>>> dfs = pd.read_html('https://www.sepe.es/contenidos/personas/prestaciones/du
>>> type(dfs)
<class 'list'>
>>> type(dfs[0])
<class 'pandas.core.frame.DataFrame'>
```

```
>>> df = dfs[0]
>>> df
   Días de cotización  Días de prestación
0      de 360 a 539          120
1      de 540 a 719          180
2      de 720 a 899          240
3      de 900 a 1079         300
4      de 1080 a 1259         360
5      de 1260 a 1439         420
6      de 1440 a 1619         480
7      de 1620 a 1799         540
8      de 1800 a 1979         600
9      de 1980 a 2159         660
10     desde 2160            720
```

```
>>> df.loc[3]
Días de cotización    de 900 a 1079
Días de prestación      300
Name: 3, dtype: object
>>> df[df['Días de cotización'] == 'de 900 a 1079']
   Días de cotización  Días de prestación
3      de 900 a 1079          300
>>> df[df['Días de prestación'] > 300]
   Días de cotización  Días de prestación
4      de 1080 a 1259         360
5      de 1260 a 1439         420
6      de 1440 a 1619         480
7      de 1620 a 1799         540
8      de 1800 a 1979         600
9      de 1980 a 2159         660
10     desde 2160            720
```

## [BeautifulSoup](#)

BeautifulSoup es un módulo para extraer información (de cualquier tipo) de páginas HTML.

## Ejemplo

```
from bs4 import BeautifulSoup
import requests

r = requests.get("https://www.ticnor.es")
data = r.text
soup = BeautifulSoup(data)
for link in soup.find_all('a'):
    print(link.get('href'))
```

## find y find\_all

- ▶ `find` -> primer resultado; `find_all` -> todos
- ▶ Aceptan diversos argumentos para encontrar **elementos**: string, expresión regular (`re.compile(...)`), listas...

```
# Por nombre
soup.find_all(['th', 'td'])

# Por atributo
soup.find_all(href=re.compile('^http'))

# Combinando
soup.find_all('a', href=True)

# Usando dict de atributos
atributos = {'src': True}
soup.find_all(attrs=atributos)

# Por clase (cualquier clase)
soup.find_all(class_='menu__link')
```

## Moviéndonos por el documento

- ▶ Cuando tenemos un elemento localizado con `find` o `find_all`:
  - ▶ `find_parents()` / `find_parent()` (ancestros o ancestro)
  - ▶ `find_next_siblings()` / `find_next_sibling()` (hermanos a continuación)
  - ▶ `find_previous_siblings()` / `find_previous_sibling()` (hermanos antes)

## Selectores CSS

```
select() y select_one()  
soup.select("title")  
soup.select("a[href]")  
soup.select("h2")  
soup.select(".servicios a")
```

## Scrapy

## Qué es Scrapy

Scrapy es una librería de Python para hacer *web crawlers* (recolectores de información en HTML).

## Ejemplo

```
import scrapy

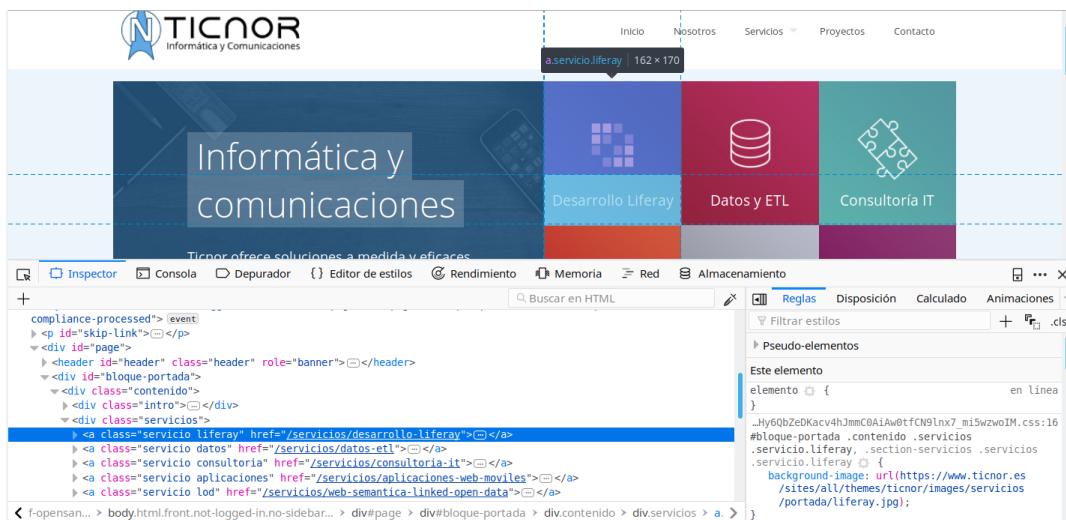
class BlogSpider(scrapy.Spider):
    name = 'blogspider'
    start_urls = ['https://blog.scrapinghub.com']

    def parse(self, response):
        for title in response.css('.post-header>h2'):
            yield {'title': title.css('a ::text').extract_first()}

        for next_page in response.css('div.prev-post > a'):
            yield response.follow(next_page, self.parse)

$ scrapy runspider blogspider.py
```

- ▶ Es necesario tener algo de conocimiento sobre la estructura del sitio que se quiere rastrear:
  - ▶ ¿Qué elementos / atributos / textos queremos extraer?
  - ▶ ¿Qué enlaces () debemos seguir (paginación, categorías...)?



Mejor método: Inspeccionar Elemento

## Procedimiento

1. Definir las URL iniciales
  - ▶ Ej: <https://scholar.google.es/scholar?q=einstein>
2. Localizar los elementos que queremos extraer
  - ▶ Ej: .gs\_ri h3 a ::text
3. Localizar los enlaces a las siguientes páginas a navegar
  - ▶ Ej: #gs\_n a

```
import scrapy

class EinsterSpider(scrapy.Spider):
    name = 'einstein_spider'
    start_urls = ['https://scholar.google.es/scholar?q=einstein']

    def parse(self, response):
        for texto in response.css('.gs_ri h3 a ::text'):
            yield {'publicacion': texto.extract()}

        for next_page in response.css('#gs_n a'):
            yield response.follow(next_page, self.parse)

scrapy runspider einstein.py -o resultados.json
```

## ¿Qué más ofrece scrapy?

- ▶ Extraer con métodos independientes distintas páginas (p.e., navegar listados, pero extraer datos de vistas individuales).
- ▶ Pasar argumentos desde la línea de comandos.
- ▶ Su consola (`scrapy shell https://www.ticnor.es`).

## Y más, y más...

- ▶ Exportar a distintos formatos (JSON, JSONL, CSV, XML).
- ▶ Seleccionar con XPath.
- ▶ Integrarlo en módulos (no sólo desde línea de comandos).
- ▶ ...

## Ejercicio

Extraer un listado de todas las calderas con su precio de <https://euroclimaonline.es/>

```
[  
  { "modelo": "XYZ", "precio": "335,30 €"},  
  { "modelo": "TRS", "precio": "299,99 €"},  
  ...  
]
```

# Python para Análisis de datos: Introducción

## Sesión 5

Miguel Expósito Martín ([exposito\\_m@cantabria.es](mailto:exposito_m@cantabria.es))

24 Octubre 2018

Python Data Analysis Library

numpy

introducción a pandas

series

dataframes

pandas desde excel

pyjstat

ejercicios

Python Data Analysis Library

numpy

## hoja de ruta

- ▶ numpy
- ▶ pandas

## numpy

es la biblioteca principal para computación científica en Python

- ▶ su principal estructura de datos es el array multidimensional
- ▶ recuerda a Matlab
- ▶ existe desde el año 2000
- ▶ es rápido, eficiente y tiene baja huella en memoria

## numpy

- ▶ almacena datos en bloques contiguos de memoria
- ▶ contiene funciones matemáticas sobre arrays sin necesidad de bucles
- ▶ implementa funcionalidades de álgebra lineal
- ▶ ofrece una API en C para conectar numpy con C o Fortran

## prueba de rendimiento

```
import numpy as np
my_array = np.arange(1000000)
my_list = list(range(1000000))

%time for _ in range(10): my_array_2 = my_arr * 2
%time for _ in range(10): my_list_2 = [x * 2 for x in my_list]
```

## tipos de datos precisos

tipo	descripción
int_	Default integer type (same as C long; normally either int64 or int32)
intc	Identical to C int (normally int32 or int64)
intp	Integer used for indexing (same as C ssize_t; normally either int32 or int64)
int8	Byte (-128 to 127)
int16	Integer (-32768 to 32767)
int32	Integer (-2147483648 to 2147483647)
int64	Integer (-9223372036854775808 to 9223372036854775807)

## tipos de datos precisos

tipo	descripción
float_	Shorthand for float64.
float16	Half precision float: sign bit, 5 bits exponent, 10 bits mantissa
float32	Single precision float: sign bit, 8 bits exponent, 23 bits mantissa
float64	Double precision float: sign bit, 11 bits exponent, 52 bits mantissa

## valores especiales

nan, inf

**¡ojo! no usar comparadores de igualdad**

usar np.isnan() o np.isinf()

## ndarrays

- ▶ contenedor flexible y rápido de datasets numéricos
- ▶ permite realizar operaciones matemáticas sobre bloques
- ▶ rank: número de dimensiones
- ▶ shape: tupla de enteros con el tamaño de cada dimensión
- ▶ datos numéricos y homogéneos (**mismo tipo**)

## ndarrays

1D array

1	2	3
---	---	---

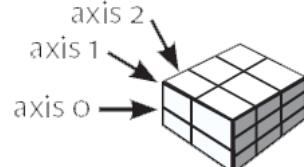
2D array

axis 1 →

axis 0 →

1.5	2	3
4	5	6

3D array



## ndarrays

```
data = np.random.randn(2, 3)
data
data * 10
data + data
```

## crear ndarrays

- ▶ forma más fácil: función array()
- ▶ acepta cualquier objeto de tipo secuencia

```
data1 = [6, 7.5, 8, 0, 1]
arr1 = np.array(data1)
arr1
```

## crear ndarrays

una lista de listas de la misma longitud se convierte a array multidimensional

```
data2 = [[1, 2, 3, 4], [5, 6, 7, 8]]
arr2 = np.array(data2)
arr2
arr2.ndim # dimension
arr2.shape # forma
arr2.dtype # tipo de datos
```

## crear ndarrays

- ▶ otras funciones: zeros, ones, empty, full, arange
- ▶ para varias dimensiones, pasar una tupla como argumento
- ▶ aceptan dtype como argumento

```
np.zeros(10)
np.zeros((3, 6))
np.ones(7)
np.ones(7, dtype=bool)
np.empty((2,3,2))
np.full((2,2), np.inf)
np.arange(15)
```

## crear ndarrays

valores aleatorios: np.random.randn()

```
arr = np.random.randn(4)
arr

matrix = np.random.randn(4,4)
matrix
```

## tipos en ndarrays

`dtype` es un objeto especial que contiene metadatos **importante**: `astype` para convertir tipos

```
arr1 = np.array([1, 2, 3], dtype=np.float64)
arr2 = np.array([1, 2, 3], dtype=np.int32)
```

```
arr1.dtype
arr2.dtype
```

```
float_arr = arr2.astype(np.float64)
float_arr.dtype
```

¿qué pasa al convertir de float a int?

## ejercicios

- ▶ crear un array 1D de números del 0 al 9
- ▶ crear un array booleano de 3x3 con todos los valores a True

## operaciones vectoriales

- ▶ sin bucles for
- ▶ entre arrays de la misma longitud, elemento a elemento

```
arr = np.array([[1., 2., 3.], [4., 5., 6.]])  
arr * arr  
1 / arr
```

## operaciones matriciales

```
A = np.array( [[1,1], [0,1]] )  
B = np.array( [[2,0], [3,4]] )  
  
A @ B  
A.dot(B)
```

## más álgebra lineal

```
from numpy.linalg import inv, qr

X = np.random.randn(5, 5)
mat = X.T.dot(X)
inv(mat)
mat.dot(inv(mat))

q, r = qr(mat)

linear algebra
```

## indexado y rebanado básicos

```
arr = np.arange(10)
arr[5]
arr[5:8]
arr[5:8] = 12
1 / arr
```

las rebanadas son vistas del array original; si se modifica una vista, se refleja en el original

**¡¡no se copian!!**

## indexado y rebanado básicos

con dos dimensiones

```
arr2d = np.array([[1, 2, 3], [4, 5, 6], [7, 8, 9]])
arr2d[2]
arr2d[0][2]

arr2d[:2]
arr2d[:, 1:]
```

## funciones universales

```
arr = np.arange(10)

np.sqrt(arr) # raíz cuadrada
np.exp(arr) # e elevado a

x = np.random.randn(8)
y = np.random.randn(8)

np.maximum(x, y) # máximo elemento a elemento
ufuncs
```

## funciones estadísticas

```
arr = np.random.randn(5, 4)

arr.mean()
arr.mean(axis=1)
arr.sum()

statistics
```

## lógica condicional

filtrar por condición

```
arr = np.arange(9)
arr[arr > 2] # elementos mayores que dos
arr[arr % 2 == 0] # elementos pares
```

## **lógica condicional**

ej: obtener x si cond es True; si no, obtener y

```
x = np.array([1.1, 1.2, 1.3, 1.4, 1.5])
y = np.array([2.1, 2.2, 2.3, 2.4, 2.5])
cond = np.array([True, False, True, True, False])

result = np.where(cond, xarr, yarr)
result
```

## **lógica condicional**

ejemplo:

*Generar una matriz de 4x4 con valores aleatorios. Reemplazar todos los valores positivos con 2 y los valores negativos con -2.*

## arrays booleanos

```
any(), all()  
bools = np.array([False, False, True, False])  
bools.any()  
bools.all()  
¿cómo contar True en un array booleano?  
arr = np.random.randn(100)  
(arr > 0).sum()
```

## ejercicio

- ▶ extraer todos los números pares de un array
- ▶ reemplazar números impares en array por -1

```
arr = np.array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9])
```

## introducción a pandas

### qué es

paquete Python con estructuras de datos flexibles, expresivas y eficientes para trabajar con datos relacionados o etiquetados

- ▶ datos tabulares con columnas heterogéneas (XLS, SQL)
- ▶ datos ordenados o no ordenados de series temporales
- ▶ datos matriciales arbitrarios con etiquetas de fila y columna
- ▶ cualquier otra forma de conjuntos de datos estadísticos observacionales

## características

- ▶ gestión de datos nulos
- ▶ tamaño mutable (borrado e inserción de columnas)
- ▶ funcionalidades de agregación (group by)
- ▶ rebanado inteligente basado en etiquetas
- ▶ fusión intuitiva de conjuntos de datos
- ▶ robustas herramientas de entrada/salida
- ▶ funcionalidad para series temporales
- ▶ open source desde 2010

## series

## qué son

objetos de una dimensión similares a un array que contienen una secuencia de valores y un array asociado de etiquetas (índice)

```
import pandas as pd  
obj = pd.Series([4, 7, -5, 3])  
obj  
obj.values  
obj.index
```

si no se especifica índice, se crea uno con enteros de 0 a N-1

## índices

a veces es deseable crear una serie con un índice etiquetado

```
import pandas as pd  
obj2 = pd.Series([4, 7, -5, 3], index=['Enero', 'Febrero', 'Marzo', 'Abril'])  
obj2
```

## índices

en contraposición a los arrays, pueden usarse las etiquetas para la selección de valores

```
import pandas as pd  
obj2['Marzo']  
obj2['Abril'] = 10  
obj2[['Enero', 'Marzo', 'Abril']]
```

## filtrado simple

la relación valor-índice se preserva

```
obj2[obj2 > 0]
```

## desde diccionario

la relación diccionario - series es directa

```
my_dict = {'visitantes': 2000, 'visitas': 50000, 'hits': 200000}
obj3 = pd.Series(my_dict)
obj3
```

## desde diccionario

se puede cambiar el orden de las claves

```
metrics = ['visitas', 'hits', 'visitantes', 'tpv']
obj4 = pd.Series(my_dict, index=metrics)
```

## ejercicio

crear una serie desde lista, array y dict

```
import numpy as np
mylist = list('abcdefghijklmnopqrstuvwxyz')
myarr = np.arange(26)
mydict = dict(zip(mylist, myarr))
```

## operaciones desde numpy

se pueden aplicar operaciones sin modificar la estructura

```
np.square(obj3)
```

alineación automática

```
obj3 + obj4
```

ejercicio

obtener los elementos de A que no están en B

usar ~ , isin

```
ser1 = pd.Series([1, 2, 3, 4, 5])
ser2 = pd.Series([4, 5, 6, 7, 8])
```

## ejercicio

obtener la media y la desviación estándar de una serie

```
s = pd.Series([1,2,3,4,5,6,7,8,9,5,3])
```

## dataframes

## qué son

*objetos que representan una tabla rectangular conteniendo una colección ordenada de columnas, cada una de las cuales puede ser de un tipo de datos distinto*

```
data = {'ca': ['Cantabria', 'Cantabria', 'Cantabria', 'Canarias', 'Canarias'],
        'year': [2000, 2001, 2002, 2001, 2002],
        'pop': [1.5, 1.7, 3.6, 2.4, 2.9]}
frame = pd.DataFrame(data) # inicialización con list of dicts
frame.head() # visualizar las 5 primeras filas
```

## ejemplo

	Trimestre	Sexo	...	Variables	value
0	2005	Ambos sexos	...	Población	NaN
1	2005	Ambos sexos	...	Activos	NaN
2	2005	Ambos sexos	...	Ocupados	NaN
3	2005	Ambos sexos	...	Parados	NaN
4	2005	Ambos sexos	...	Parados que buscan primer empleo	NaN
5	2005	Ambos sexos	...	Inactivos	NaN
6090	2018 - 2	Mujeres	...	Población	253.7
6091	2018 - 2	Mujeres	...	Activos	126.7
6092	2018 - 2	Mujeres	...	Ocupados	109.5
6093	2018 - 2	Mujeres	...	Parados	17.2
6094	2018 - 2	Mujeres	...	Parados que buscan primer empleo	2.2
6095	2018 - 2	Mujeres	...	Inactivos	127.1

## creación y manejo básico

- ▶ se puede especificar el orden de las columnas
- ▶ si una columna no existe, se creará con valores nulos
- ▶ una columna se puede recuperar como un objeto de tipo series
- ▶ también pueden recuperarse filas

```
pd.DataFrame(data, columns=['year', 'state', 'pop'])
frame2 = pd.DataFrame(data, columns=['year', 'ca', 'pop', 'debt'],
                      index=['uno', 'dos', 'tres', 'cuatro', 'cinco'])
frame2
frame2['ca'] # columna
frame2.loc['tres'] # fila
```

## lectura desde archivo

.read\_csv()

acepta:

- ▶ URLs
- ▶ sep: separador (,)
- ▶ delimiter: delimitador (None)
- ▶ header: cabecera (se infiere)

## escritura desde archivo

```
.write_csv()
```

## manejo básico

- ▶ se puede obtener una lista de columnas
- ▶ el atributo `dtypes` indica el tipo de dato de cada columna
- ▶ las dimensiones del dataframe se obtienen con `.shape`

```
frame2.columns  
frame2.dtypes  
frame2.shape
```

## manejo básico

la serie devuelta al recuperar una columna es una vista

```
vals = frame['pop']
vals[0] = 0
vals
data

usar copy() para eliminar el warning
```

## manejo básico

un dataframe se puede transponer

```
frame2.T
```

## manejo básico

cálculo de frecuencias

```
frame2.ca.value_counts()
```

## manejo básico

cálculo de valores únicos

```
frame2.ca.nunique()
```

## manejo básico

funciones estadísticas

```
frame2['pop'].mean()  
frame2['pop'].sum()  
frame2['pop'].max()  
frame2['pop'].min()  
frame2['pop'].median()
```

## manejo básico

valores nulos: isnull(): NaN, None/NaN, NaT

```
array = np.array([[1, np.nan, 3], [4, 5, np.nan]])  
pd.isnull(array)
```

## inspección

```
frame2.describe() # información estadística  
frame2.info() # información sobre la estructura de datos  
frame2.tail()
```

## selección de columnas

un dataframe puede verse como un conjunto de series que comparten un índice (las cabeceras de las columnas)

```
frame2[['year', 'pop']] # selección columnas  
frame2[frame2['pop'] > 2] # filtrado con condición  
frame2[(frame2['pop'] > 2) & (frame2['year'] != 2002)] # filtrado con condición  
frame2.iloc[:, 0:2] # todas las filas, columnas 0 a 2 no incluida  
frame2.iloc[:, :-1] # todas las filas, todas las columnas excepto la última
```

## indexado

```
frame4 = frame2.set_index('year')
frame4
```

## selección de filas

```
frame2.iloc[3] # posición
frame2.iloc[[3,4]] # posición
frame4.loc[2001] # etiqueta
```

## reseteo de índice

```
frame4.reset_index(inplace=True)
```

## resumen de selección

- ▶ loc para indexado basado en etiquetas
- ▶ iloc para indexado posicional

aunque hay otras formas...

para modificar el propio dataframe, usar el atributo `inplace=True`

## ejercicio inspección

importar datos desde:

```
DATA_URL = 'https://raw.githubusercontent.com/justmarkham/DAT8/master/data/u.user'
usuarios = pd.read_csv(DATA_URL, sep='|')
```

asignar a una variable llamada `users` y utilizar `user_id` como índice

## ejercicio inspección

- ▶ ver las 25 primeras filas
- ▶ ver las 10 últimas filas
- ▶ obtener el número de observaciones en el dataset
- ▶ obtener el número de columnas en el dataset
- ▶ mostrar los nombres de las columnas
- ▶ mostrar el índice del dataset

## ejercicio inspección

- ▶ mostrar los tipos de datos de cada columna
- ▶ mostrar sólo la columna de ocupación
- ▶ mostrar cuántas ocupaciones diferentes hay en el dataset
- ▶ mostrar la ocupación más frecuente
- ▶ resumir el dataframe
- ▶ calcular la edad media de los usuarios

## joins: merge

inner join

```
left_frame = pd.DataFrame({'key': range(5),
                           'left_value': ['a', 'b', 'c', 'd', 'e']})
right_frame = pd.DataFrame({'key': range(2, 7),
                            'right_value': ['f', 'g', 'h', 'i', 'j']})
left_frame
right_frame

pd.merge(left_frame, right_frame, on='key', how='inner')
```

## joins: merge

left, right, full outer join

```
pd.merge(left_frame, right_frame, on='key', how='left')
pd.merge(left_frame, right_frame, on='key', how='right')
pd.merge(left_frame, right_frame, on='key', how='outer')
```

## combinación: concat

```
pd.concat([left_frame, right_frame]) # vertical
pd.concat([left_frame, right_frame], axis=1) # horizontal
```

## añadir filas al final: append

los dos dataframes deben tener las mismas columnas

```
left_frame.append(left_frame)
```

## groupby

```
ANIMALS_URL='https://gist.githubusercontent.com/predicador37/bccac851999eaf10b6  
animals = pd.read_csv(ANIMALS_URL)
```

## groupby

¿cuál es el peso medio por animal?

```
# Agrupar por cada categoría de animal
animal_groups = animals.groupby("animal")
# Aplicar la media aritmética a la columna de peso
animal_groups['weight'].mean()
```

## agg

permite agregar por varias funciones según un eje

```
animal_groups['weight'].agg(['mean', 'median'])
```

## ejemplo

análisis de dataset de netflix

```
NETFLIX_URL='https://gist.githubusercontent.com/predicador37/d081821c1538cc6c26  
df = pd.read_csv(NETFLIX_URL)
```

## ejemplo: limpieza

- ▶ dropna(): elimina filas con valores nulos
- ▶ drop\_duplicates(): elimina filas duplicadas

```
df.dropna(inplace=True)  
df.drop_duplicates(inplace=True)
```

ejemplo: inspección

```
df.describe()
```

ejemplo: split

se dividen los datos en grupos, donde cada grupo es el conjunto de películas estrenadas en un año determinado

```
df_by_year = df.groupby('release year')
type(df_by_year)
```

## ejemplo: apply

```
aplicar .describe() a cada grupo  
df_by_year.describe().head()
```

## ejemplo: combine

obtener la media o la mediana de la puntuación de usuario por año

```
df_med_by_year = df_by_year.median()  
df_med_by_year.head()  
df_med_by_year['user rating score']
```

## ejercicio: agrupaciones

importar datos desde:

```
DATA_URL = 'https://raw.githubusercontent.com/justmarkham/DAT8/master/data/u.us  
¡inspeccionar separadores!
```

## ejercicio: agrupaciones

- ▶ inspeccionar dataset
- ▶ mostrar edad media por ocupación
- ▶ para cada ocupación, calcular las edades mínima y máxima
- ▶ para cada combinación de ocupación y sexo, calcular la edad media

## aplicar funciones a filas, columnas y elementos

- ▶ .apply(): función a arrays 1D a cada fila o columna
- ▶ .applymap(): elemento a elemento en dataframe
- ▶ .map(): elemento a elemento en series

## funciones anónimas

son funciones definidas sin nombre

```
lambda arguments: expression  
def square_root(x):  
    return math.sqrt(x)  
  
square_root = lambda x: math.sqrt(x)  
se usan mucho para aplicar funciones a dataframes
```

## ejemplo: aplicar operaciones en dataframes

```
import pandas as pd
import numpy as np

data = {'name': ['Pedro', 'Paco', 'Lorena', 'Juan', 'Patxi'],
        'year': [2012, 2012, 2013, 2014, 2014],
        'reports': [4, 24, 31, 2, 3],
        'coverage': [25, 94, 57, 62, 70]}
df = pd.DataFrame(data, index = ['Santander', 'Torrelavega', 'Laredo', 'Camargo',
                                 df
```

## ejemplo

```
crear función que pase a mayúsculas usando lambda
capitalizer = lambda x: x.upper()
```

## ejemplo

aplicar la función a la columna `name`

```
df['name'].apply(capitalizer)
```

## ejemplo

equivalente al anterior

```
df['name'].apply(lambda x: x.upper())
```

## ejemplo

```
mapear la función sobre cada elemento de la series name  
df['name'].map(capitalizer)
```

## ejemplo

```
aplicar una raiz cuadrada a todas las celdas del dataframe  
df = df.drop('name', axis=1) # eliminar la columna de texto  
df.applymap(np.sqrt)
```

## ejemplo

aplicar una función que multiplica por 100 los números

```
def times100(x):
    if type(x) is str:
        return x
    elif x:
        return 100 * x
    else:
        return
df.applymap(times100)
```

pandas desde excel

## cargar fichero de ejemplo

```
import pandas as pd
import numpy as np
DATA_URL = 'https://gist.githubusercontent.com/predicador37/24e8e4ee465956aa923'
df = pd.read_csv(DATA_URL)
df.head()
```

## añadir columna de totales

```
df["total"] = df["Jan"] + df["Feb"] + df["Mar"]
df.head()
```

## análisis básico a nivel de columna

- ▶ ¿total, media, mínimo, máximo del mes de enero?

```
df["Jan"].sum()  
df["Jan"].mean()  
df["Jan"].min()  
df["Jan"].max()
```

## añadir subtotales por mes y total general

```
sum_row = df[["Jan","Feb","Mar","total"]].sum() # suma para cada columna  
df_sum = pd.DataFrame(data=sum_row).T # crear un nuevo dataframe transpuesto  
df_sum  
df_sum = df_sum.reindex(columns=df.columns) # añadir las columnas que faltan  
df_sum  
df_final = df.append(df_sum, ignore_index=True)  
df_final.tail()
```

## añadir subtotales por estado

```
from money import Money

df_sub = df_final[["state", "Jan", "Feb", "Mar", "total"]].groupby('state').sum()
formatted_df = df_sub.applymap(lambda x: Money(x, currency='EUR'))
formatted_df

sum_row=df_sub[["Jan", "Feb", "Mar", "total"]].sum() # calcular subtotales como en
sum_row
df_sub_sum=pd.DataFrame(data=sum_row).T
def money(x):
    return Money(x, currency='EUR')
df_sub_sum = df_sub_sum.applymap(money)
df_sub_sum
```

## añadir subtotales por estado

```
final_table = formatted_df.append(df_sub_sum)
final_table

final_table = final_table.rename(index={0: "Total"})
final_table
```

## otro ejemplo

lectura del fichero

```
import pandas as pd
import numpy as np
DATA_URL = 'https://gist.githubusercontent.com/predicador37/29a4c89cc652d3b201e'
df = pd.read_csv(DATA_URL)
df.head()
df.dtypes
```

## otro ejemplo

conversión de objeto a fecha

```
df['date'] = pd.to_datetime(df['date'])
df.head()
df.dtypes
```

otro ejemplo

filtrar por número de cuenta 307599

```
df[df['account number'] == 307599].head()
```

otro ejemplo

filtrar por cantidad mayor a 22

```
df[df['quantity'] > 22].head()
```

otro ejemplo

filtrar por referencia (sku) que empiece por B1

```
df[df['sku'].map(lambda x: x.startswith('B1'))].head()
```

otro ejemplo

combinar los dos filtros anteriores

```
df[(df['quantity'] > 22) & (df['sku'].map(lambda x: x.startswith('B1')))].head()
```

otro ejemplo

encontrar todos los registros que incluyen dos números de cuenta específicos

```
df[df['account number'].isin([714466, 218895])].head()
```

otro ejemplo

query: requiere instalación previa de `numexpr` recuperar lista de clientes por nombre

```
df.query('name == ["Kulas Inc", "Barton LLC"]').head()
```

## otro ejemplo

trabajar con fechas

```
df = df.sort('date')
df.head()
df[df['date'] >='20140905'].head() # por fecha exacta
df[df['date'] >='2014-03'].head() # por mes
df[df['date'] >= 'Oct-2014'].head() # por mes en otro formato
df[df['date'] >= '10-10-2014'].head() # por fecha en otro formato
```

## otro ejemplo

series temporales: utilizar la fecha como índice con `set_index()`

```
df2 = df.set_index(['date'])
df2.head()
df2["20140101":"20140201"].head() # filtrado por rango de fechas
df2["2014"].head() # filtrado por año
```

## otro ejemplo

identificar referencias que contienen un determinado valor

```
df[df['sku'].str.contains('B1')].head()  
df[(df['sku'].str.contains('B1-531')) & (df['quantity']>40)].sort_values(by=['q
```

## otro ejemplo

obtener una lista de valores únicos

```
df['name'].unique()
```

## pyjstat

### qué es

paquete Python para convertir JSON-stat a pandas dataframe

<https://json-stat.org/>

se puede visualizar en:

- ▶ <http://jsonviewer.stack.hu/>
- ▶ <https://json-stat.org/format/browser/>
- ▶ <https://json-stat.org/format/viewer/>
- ▶ <http://json-stat.com/explorer/>

## ejemplo: lectura de archivo

```
from pyjstat import pyjstat
DATA_URL = 'http://www.icane.es/data/api/active-population-aged-16-more-gender-
dataset = pyjstat.Dataset.read(DATA_URL) # lee dataset de json-stat
df = dataset.write('dataframe') # genera un dataframe
print(df)
```

## ejemplo: consulta

```
query = [{'sexo': 'hombres'}, {'trimestre': '2016-1'}, {'grupo-de-edad': 'total'}
dataset.get_value(query)
obtener el total de activos mujeres para todos los grupos de edad en el segundo
trimestre de 2017
```

## ejercicios

### ejercicio: dataset ICANE

importar y explorar el archivo JSON-stat desde la siguiente URL:

```
DATA_URL = 'http://www.icane.es/data/api/municipal-register-annual-review-munic'
```

- ▶ crear un índice temporal
- ▶ filtrar los últimos 5 años de tu municipio de residencia en una variable df\_muni
- ▶ obtener una serie con la población para ese filtro

## ejercicio: dataset ICANE

consultar en el dataset original la variación interanual de la población total de Cantabria en 2017.

¡jojo! inspeccionar dataset en <http://jsonviewer.stack.hu/> para identificar nombres de dimensiones y variables

## ejercicio: nombres de niños en US

crear un dataframe desde la siguiente URL y asignárselo a la variable baby\_names

```
DATA_URL = 'https://raw.githubusercontent.com/guipsamora/pandas_exercises/master/01_Statistics_in_Python/titanic.csv'
```

## ejercicio: nombres de niños en US

- ▶ visualizar los primeros diez registros
- ▶ deshacerse de las columnas Unnamed: 0 e Id (se puede usar .drop())
- ▶ ¿hay más nombres de niñas o de niños?
- ▶ agrupar por nombre agregando con suma y asignarlo a la variable names
- ▶ ¿cuántos nombres diferentes hay en el dataset?
- ▶ ¿cuál es el nombre más frecuente?
- ▶ obtener un resumen con la media, mínimo, máximo, std y cuartiles

# Python para Análisis de datos: Introducción

## Sesión 6

Jesús Fernández (fernandez.cuesta@gmail.com)

25 Octubre 2018

Visualización de datos

Tipos de gráficos más importantes

Gráfico de dispersión

Gráfico de barras

Histograma

Diagrama circular

Elementos y parámetros

Estilos

Integración con pandas

Tipos de gráficos disponibles

Parámetros opcionales

## Visualización de datos

Diferentes librerías para visualización de datos

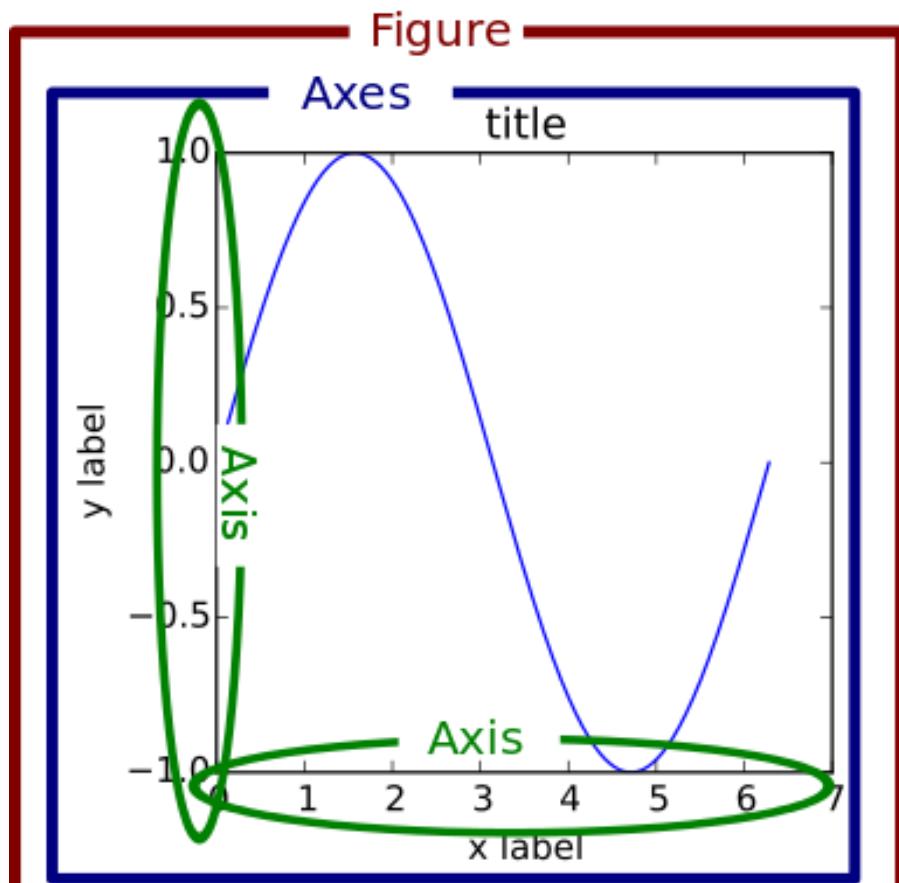
- ▶ Matplotlib
  - ▶ standard de-facto
  - ▶ hacer fácil tareas sencillas y posible tareas complejas
  - ▶ versátil, madura y extensa
  - ▶ integrado con pandas (<100% )
  - ▶ acepta paquetes de terceros como **extensiones**
- ▶ Algunas extensiones de Matplotlib:
  - ▶ Seaborn
    - ▶ funciona sobre `matplotlib`
    - ▶ enfocado a análisis estadístico
  - ▶ Cartopy, folium
    - ▶ visualización de datos en mapas
- ▶ Bokeh
  - ▶ enfocado a gráficos interactivos

## Matplotlib

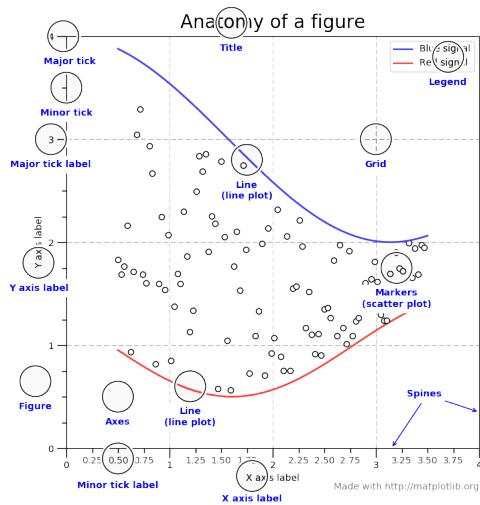
pyplot: módulo con interfaz parecida a MATLAB

```
import matplotlib.pyplot as plt  
# from matplotlib import pyplot as plt  
Importar las librerías necesarias
```

Partes de una figura



## Partes de una figura (II)



Los componentes más importantes son:

- ▶ Figura
- ▶ Ejes (axes)
  - ▶ región de la figura donde se visualizarán gráficos
- ▶ Eje (axis)
  - ▶ eje de coordenadas
- ▶ Artist
  - ▶ cualquier elemento (líneas, leyenda, etc., incluidos ejes)

Como regla general seguiremos los siguientes pasos:

1. Crear figura
2. Obtener ejes
3. Dibujar sobre los ejes

**Nota:** 1 y 2 se pueden combinar en un mismo comando

```
N = 1000
np.random.seed(2983) # reproducibilidad

fig = plt.figure() # crea figura
ax = plt.subplot(1, 1, 1) # crea ejes
t = pd.date_range('1/1/2018', periods=365)
y = np.random.randn(N).cumsum()
ax.plot(t, y) # dibuja sobre los ejes
```

De forma simplificada, los ejes y la figura se crean simultáneamente:

```
# fig = plt.figure() # crea figura
# ax = plt.subplot(1, 1, 1) # crea ejes
(fix, ax) = plt.subplots(1, 1) # (n_rows, n_cols)
t = pd.date_range('1/1/2018', periods=365)
y = np.random.randn(N).cumsum()
ax.plot(t, y) # dibuja sobre los ejes
```

## Importante

En un cuaderno los gráficos aparecerán automáticamente si la primera línea es:

```
%matplotlib inline
```

De lo contrario necesitaremos ejecutar:

```
# [...]
ax.plot(t, y, 'g.-')
plt.show() # muestra el gráfico
... para mostrar cada gráfico
```

matplotlib acepta los siguientes tipos de datos:

- ▶ Listas
- ▶ Diccionarios
- ▶ np.array
- ▶ pandas.DataFrame

El tipo de datos nativo es np.array.

El resto puede -o no- funcionar.

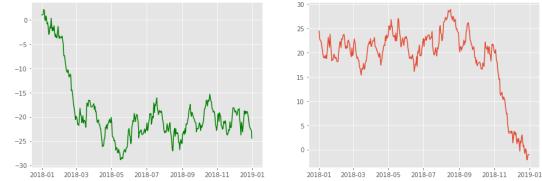
## plt.subplots()

Para crear una tupla (figura, ejes) usaremos:

- ▶ plt.subplots()
  - ▶ sin argumentos: crea una figura con 1 area de dibujo
  - ▶ (nrows, ncols): crea una figura con nrows\*ncols areas
  - ▶ devuelve la figura y todos los ejes

```
fig, (eje1, eje2) = plt.subplots(1, 2)
```

```
eje1.plot(t, y, 'g')
eje2.plot(t, -y[::-1])
```

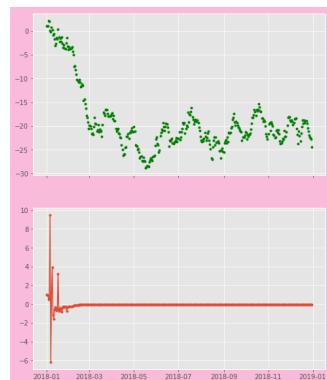


## plt.subplots()

Para crear una tupla (figura, ejes) usaremos:

- ▶ plt.subplots()
  - ▶ sin argumentos: crea una figura con 1 area de dibujo
  - ▶ (nrows, ncols): crea una figura con nrows\*ncols areas
  - ▶ devuelve la figura y todos los ejes

```
fig, ejes = plt.subplots(
    2, 1, sharex=True,
    figsize=(8, 10),
    facecolor="#fabada"
)
ejes[0].plot(t, y, 'g.')
ejes[1].plot(t, 1/y, '.-')
```



```

plt.subplot()
Ojo!: plt.subplot() != plt.subplots()
► (nrows, ncols, index): crea un eje
► devuelve 1 solo eje (al que se haga referencia)

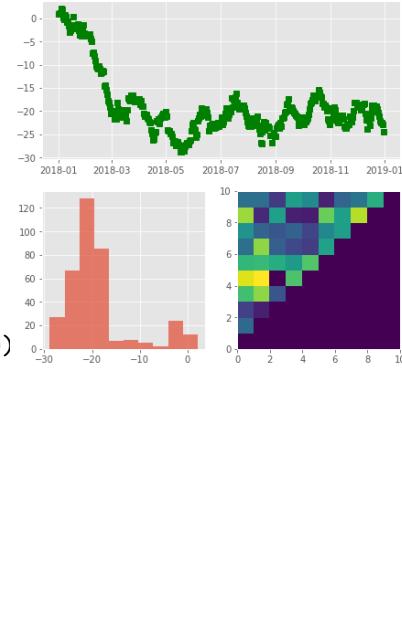
fig = plt.figure(figsize=(10, 10))

ax1 = plt.subplot(2, 1, 1)
ax1.plot(t, y, 'gs')

ax2 = plt.subplot(2, 2, 3)
ax2.hist(y, alpha=.7)

ax3 = plt.subplot(2, 2, 4)
ax3.pcolormesh(
    np.tril(np.random.uniform(size=(10, 10)
                               -1))
)

```



## Guardar una figura a fichero

```

from sklearn.datasets import load_iris

iris = load_iris()
plt.style.use('ggplot')
fig, ax = plt.subplots(figsize=(7, 6))
formatter = plt.FuncFormatter(lambda i, *args: iris.target_names[int(i)])

plt.scatter(iris.data[:, 0], iris.data[:, 1], c=iris.target)
plt.colorbar(ticks=[0, 1, 2], format=formatter)

figure.savefig('iris.pdf')

Tipos de fichero soportados, según backend:
print(fig.canvas.get_supported_filetypes())

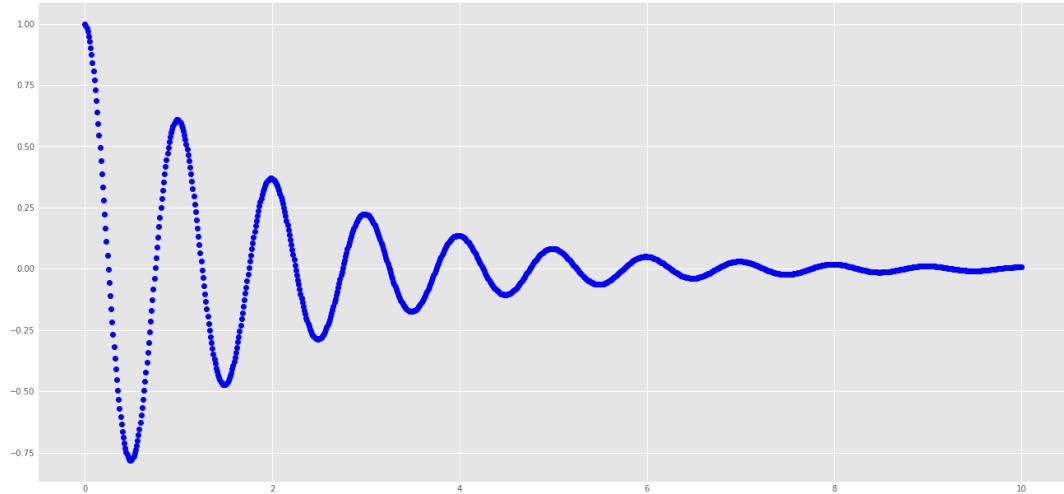
```

## Tipos de gráficos más importantes

### Lineplot (.plot())

Tipo de gráfico por defecto ## ~~~python (fix, ax) = plt.subplots(1, 1) # (n\_rows, n\_cols)

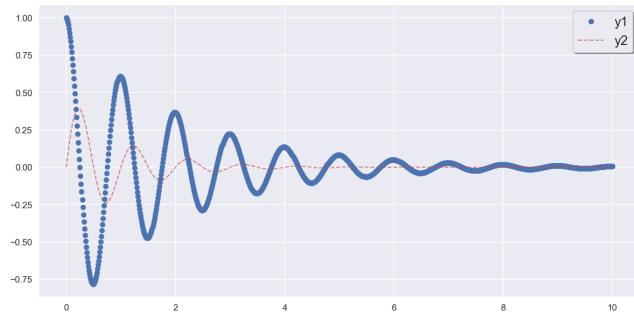
```
x = np.linspace(0, 10, 1000) y = np.exp(-x/2) * np.cos(2np.pix) ax.plot(x, y, 'bo') ~~~
```



Dos gráficos al mismo tiempo:

```
(fig, ax) = plt.subplots(1, 1) # (n_rows, n_cols)

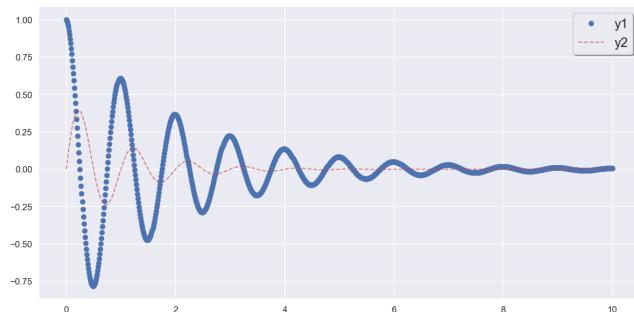
x = np.linspace(0, 10, 1000)
y1 = np.exp(-x / 2) * np.cos(2 * np.pi * x)
y2 = 0.5 * np.exp(-x) * np.sin(2 * np.pi * x)
ax.plot(x, y1, 'bo',
        x, y2, 'r--')
```



Dos gráficos al mismo tiempo:

```
(fig, ax) = plt.subplots(1, 1) # (n_rows, n_cols)

x = np.linspace(0, 10, 1000)
y1 = np.exp(-x / 2) * np.cos(2 * np.pi * x)
y2 = 0.5 * np.exp(-x) * np.sin(2 * np.pi * x)
ax.plot(x, y1, 'bo')
ax.plot(x, y2, 'r--')
```



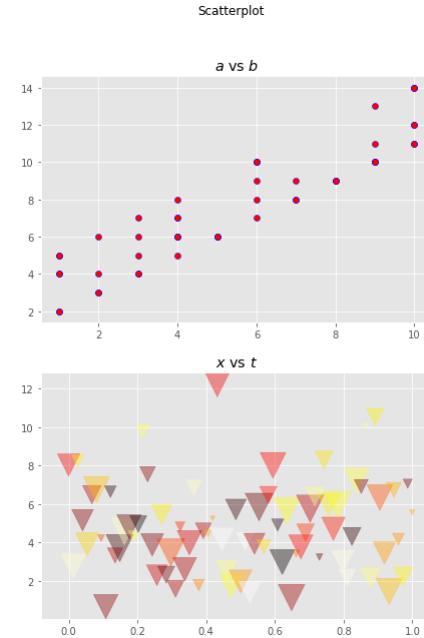
## Gráfico de dispersión

```
N = 75
np.random.seed(45987230)

fig, (ax1, ax2) = plt.subplots(2, 1,
                             figsize=(7, 10))
a = np.random.randint(low=1, high=11, size=50)
b = a + np.random.randint(1, 5, size=a.size)
x = np.linspace(0, 1, N)
t = np.random.gamma(5, size=N)
colors = np.random.rand(N)

ax1.scatter(x=a, y=b, marker='o', c='r',
            edgecolor='b')
ax1.set_title('$a$ vs $b$')
ax2.scatter(
    x, t,
    s=np.random.randint(10,800, N), # tamaño
    marker='v', # tipo de marcador
    c=colors, # colores
    alpha=0.4 # nivel de transparencia
)
ax2.set_title('$x$ vs $t$')

fig.suptitle("Scatterplot")
```



## Gráfico de barras

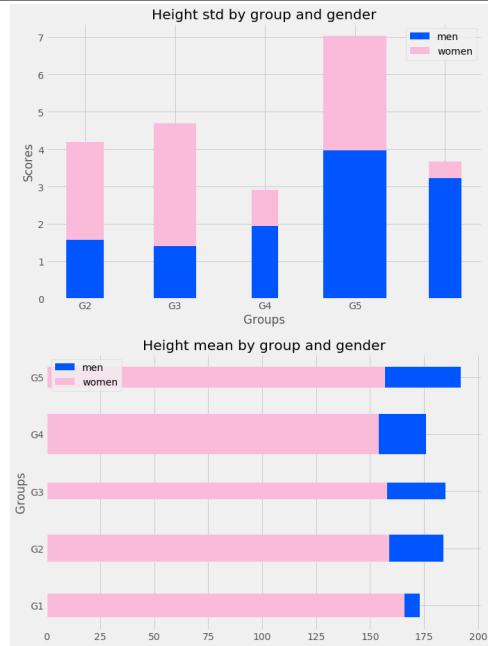
```
ax1.bar(ind,      # eje de abcisas
        men_std, # eje de ordenadas
        width,   # anchura
        color='#0055ff')
ax1.bar(ind,
        women_std,
        width,
        color='#fabada',
        bottom=men_std)

ax1.set_ylabel('Scores')
ax1.set_xlabel('Groups')
ax1.set_title('Height std by group+gender')
ax1.legend(['men', 'women'], loc='best')

# Barras horizontales

ax2.bard(ind,      # eje de ordenadas
          men_means, # eje de abcisas
          width,     # anchura
          color='#0055ff')
ax2.bard(ind,
          women_means,
          width,
          color='#fabada')

ax2.legend(['men', 'women'], loc='upper left')
ax2.set_title('Height mean by group+gender')
```

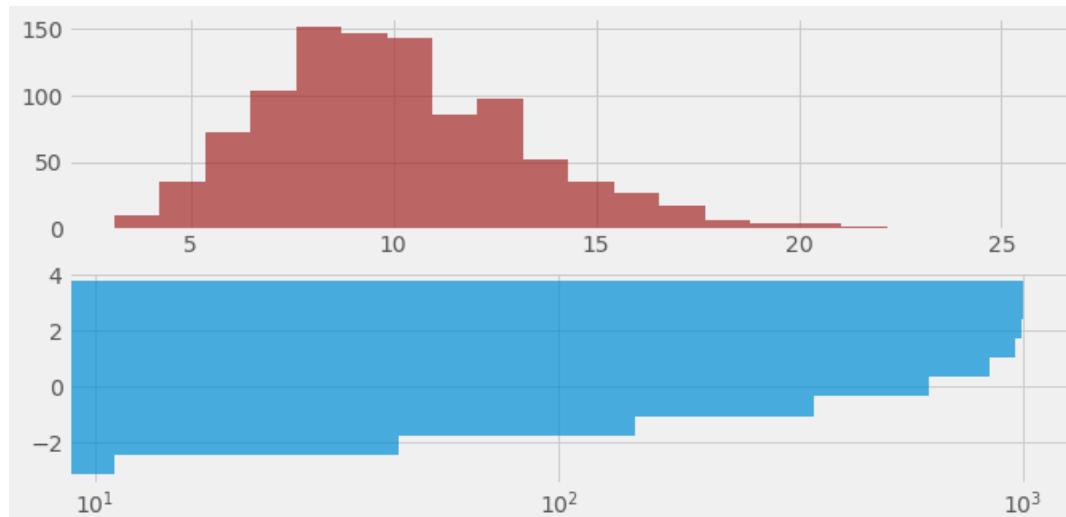


## Histogramma

```
fig, ax = plt.subplots(2, figsize=(10,5))
bins = 20
x1 = np.random.gamma(10, size=1000)
x2 = np.random.randn(1000)

ax1, ax2 = ax.flatten()

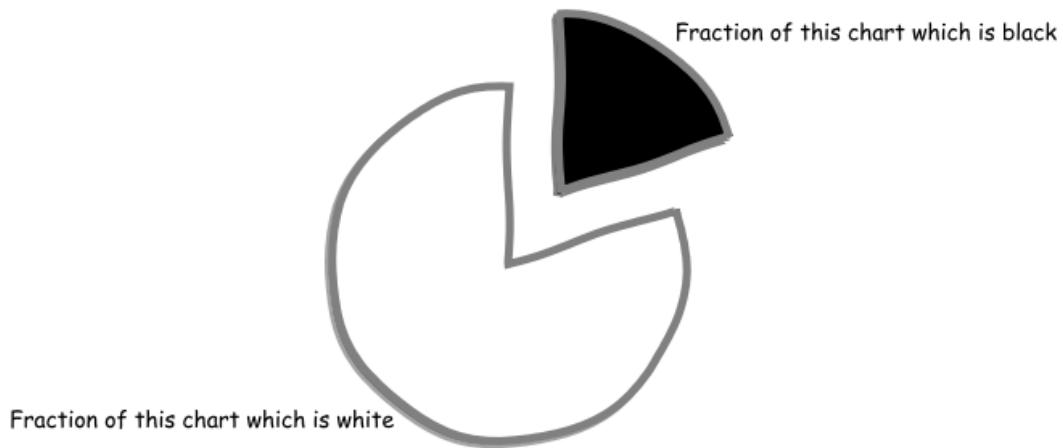
(ax1_values, _, _) = ax1.hist(x1, bins=bins, facecolor='brown', alpha=.7);
(_, ax2_bins, _) = ax2.hist(x2, alpha=.7, cumulative=True,
                           log=True, orientation='horizontal')
```



## Diagrama circular

```
colors = ['white', 'black']
labels = [f'Fraction of this chart which is {color}'
          for color in colors]

ax.pie(
    [80, 20], explode=(0, .5), labels=labels,
    colors=colors, shadow=True, startangle=90
)
# ejes iguales = asegurarnos de que se muestre
# como un círculo
ax.axis('equal');
```



## Elementos y parámetros

### Color/estilo de línea

El color de línea para un gráfico individual se puede controlar mediante una cadena de texto (p.e. 'r--'):

```
colors = {'b': 'blue', 'g': 'green', 'r': 'red', 'c': 'cyan', 'm': 'magenta',
          'y': 'yellow', 'k': 'black', 'w': 'white'}

lineStyles = {'-': '_draw_solid', '--': '_draw_dashed', '-.': '_draw_dot',
              ':': '_draw_dotted', 'None': '_draw_nothing',
              '': '_draw_nothing', ''': '_draw_nothing'}

markers = {
    '.': 'point', ',': 'pixel', 'o': 'circle', 'v': 'triangle_down',
    '^': 'triangle_up', '<': 'triangle_left', '>': 'triangle_right',
    '1': 'tri_down', '2': 'tri_up', '3': 'tri_left', '4': 'tri_right',
    '8': 'octagon', 's': 'square', 'p': 'pentagon', '*': 'star',
    'h': 'hexagon1', 'H': 'hexagon2', '+': 'plus', 'x': 'x', 'D': 'diamond',
    'd': 'thin_diamond', '|': 'vline', '_': 'hline', 'P': 'plus_filled',
    'X': 'x_filled', 0: 'tickleleft', 1: 'tickright', 2: 'tickup', 3: 'tickdown',
    4: 'caretleft', 5: 'caretright', 6: 'caretup', 7: 'caretdown',
    8: 'caretleftbase', 9: 'caretrightbase', 10: 'caretupbase',
    11: 'caretdownbase', 'None': 'nothing', None: 'nothing', '': 'nothing',
    ''': 'nothing'
}
```

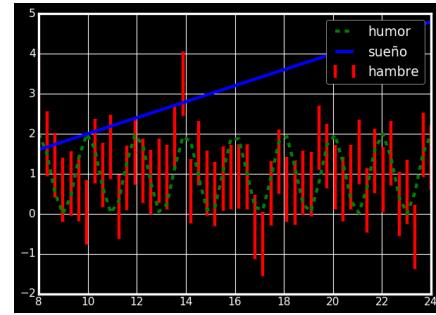
```

N = 50
np.random.seed(4873)

x = np.linspace(0, 10, N)
k = 0.8
y = k + np.sin(x) * np.random.randn(N)

(fig, ax) = plt.subplots(1)
ax.errorbar(x, y, yerr=k, fmt='r')
ax.plot(x, 1 + np.cos(np.pi*x), '--g')
ax.plot(x, x/5, 'b')

```

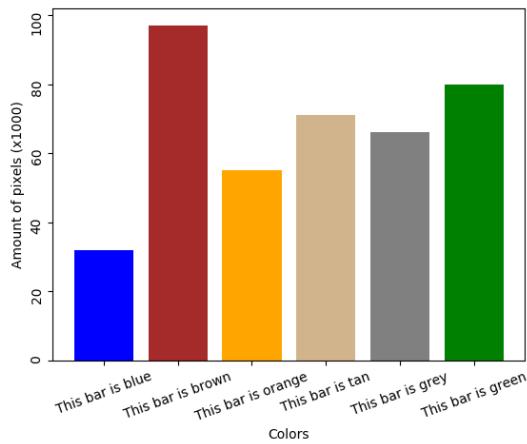


## xticks/yticks

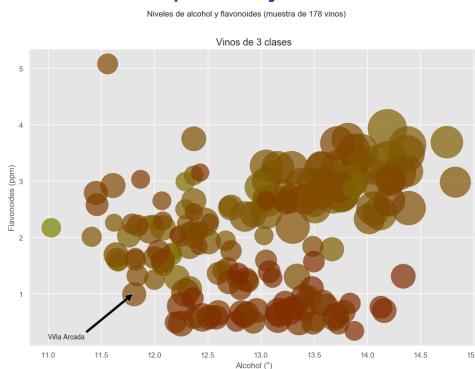
```

plt.yticks(rotation='vertical')
plt.xticks(rotation=20) # 20 grados en sentido antihorario

```



## Títulos, etiquetas y anotaciones



- ▶ Título de la figura
- ▶ Título de cada subplot (Axes)
- ▶ Título de cada eje (Axis)
- ▶ Anotaciones

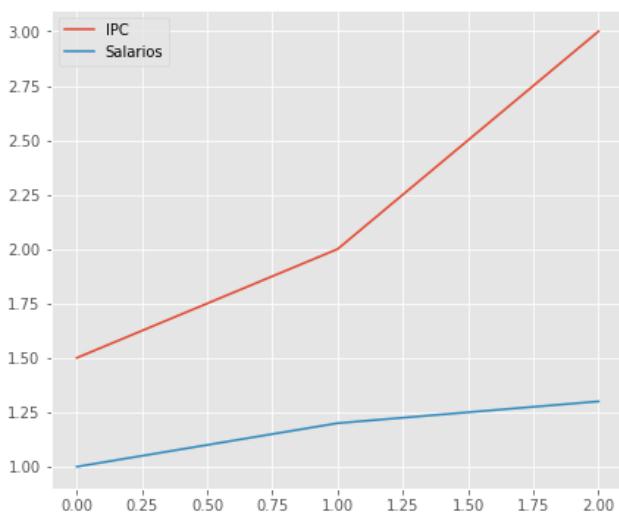
```
fig.suptitle('Niveles de alcohol y flavonoides',
             '(muestra de 178 vinos)')
ax.set_title('Vinos de 3 clases')
ax.set_xlabel('Alcohol (°)')
ax.set_ylabel('Flavonoides (ppm)')
ax.annotate(
    'Viña Arcada', xy=(11.8, 1),
    xytext=(11, .5),
    arrowprops=dict(facecolor='black', shrink=0.
)
```

## Leyenda

- ▶ Se genera automáticamente según datos inferidos de las etiquetas

```
(line1, ) = ax.plot([1.5, 2, 3],
                     label='IPC')
(line2, ) = ax.plot([1, 1.2, 1.3],
                     label='Salarios')
ax.legend(loc='upper left')
loc: define dónde emplazar la
leyenda
    best, upper right, upper
    left, lower left, lower right,
    right, center left, center
    right, lower center, upper
    center, center
```

:::



## Estilos

- ▶ Aplican a todos los gráficos generados
- ▶ Estilo por defecto en `matplotlib.rcParams`
  - ▶ pueden modificarse dinámicamente
- ▶ Podemos cambiar a estilos preconfigurados:

```
print(plt.style.available)
plt.style.use('ggplot')
```

- ▶ y/o modificar parámetros individualmente

```
plt.rcParams["figure.figsize"] = (20.0, 15.0)
plt.rcParams['font.family'] = ['monospace']
```

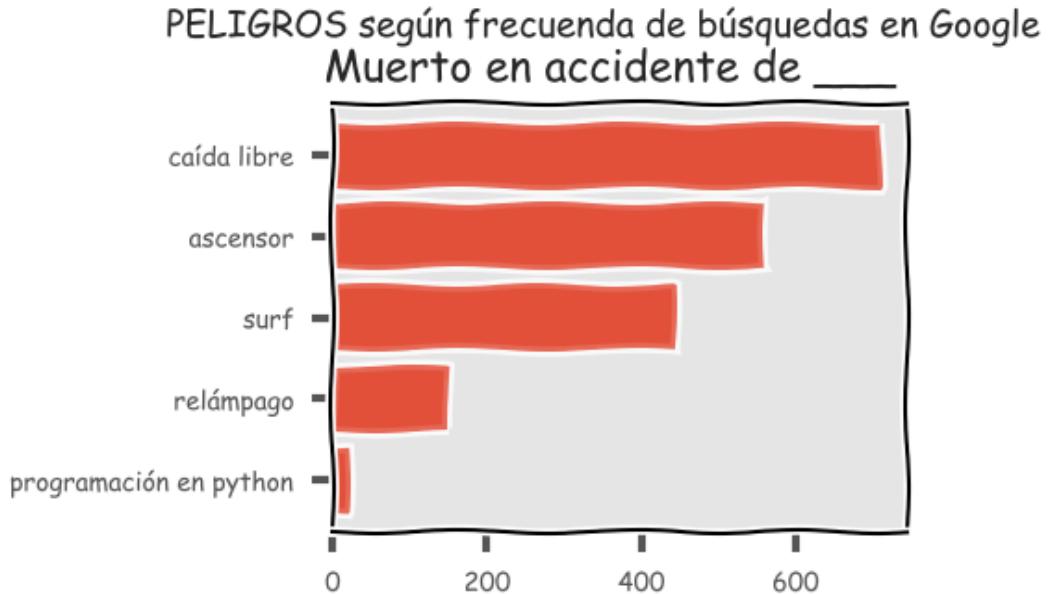
- ▶ Los cambios aplican a todos los gráficos

- ▶ salvo cambio temporal de estilos:

```
with plt.style.context('dark_background'):
    plt.plot(np.sin(np.linspace(0, 2 * np.pi)), 'r-o')
```

## Otros estilos

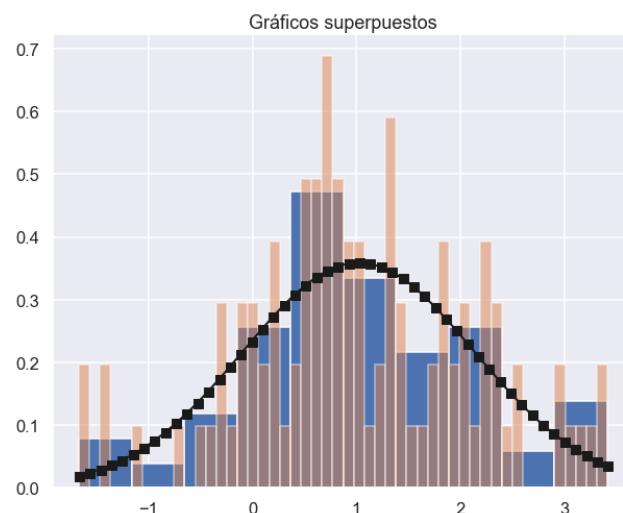
```
with plt.xkcd():
    ...
```



## Otros estilos

```
seaborn
import seaborn as sns
sns.set()
sns.set_context("talk")
```

```
...:
```



## Integración con pandas

pyplot está (parcialmente) integrado en pandas

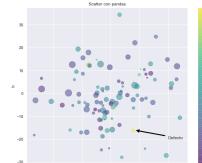
- ▶ Podemos dibujar directamente desde un dataframe de pandas
- ▶ se devuelve un objeto “*Axes*” sobre el que poder trabajar
  - ▶ no es necesario crear con antelación la figura y los ejes
  - ▶ ... aunque suele ser lo recomendable

```
# Crea la figura y los ejes
fig, ax = plt.subplots()

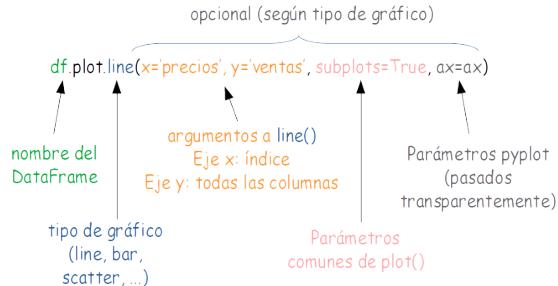
# siempre se devuelve referencia al eje
ax2 = df.plot.scatter(
    'a', 'b', c='c', s=df['d'],
    colormap='viridis', alpha=.5,
    title='Scatter con pandas', rot='vertical',
    ax = ax;
)

ax.annotate(
    'Defecto', xy=(1.9, -17),
    xytext=(2, -20),
    arrowprops=dict(facecolor='black', shrink=0.05)
)

ax2 == ax # dos referencias al mismo objeto
True
```



## Sintaxis



De forma alternativa:

```
df.plot(kind='line', x='precios', y='ventas', subplots=True, ax=ax)
```

Visualizar datos usando la interfaz de pandas es conveniente y mucho más sencillo:

```
provincias = ['Cantabria', 'Madrid', 'Murcia', 'León', 'Albacete']
index = np.arange(len(provincias)) + 0.3

y_offset = np.zeros(len(marriages.Total.columns))
cell_text = []

for row in marriages.Total.columns:
    _data = marriages.Total.loc[provincias, row]
    plt.bar(index, _data, bottom=y_offset, width=0.5)
    y_offset = y_offset + _data
    cell_text.append(["%1.1f" % (x / 1000.0) for x in y_offset])
cell_text.reverse()
tabla = plt.table(cellText=cell_text,
                  rowLabels=marriages.Total.columns,
                  colLabels=provincias,
                  loc='bottom')
plt.legend(marriages.Total.columns)
plt.title('Total Matrimonios en 2017')
```

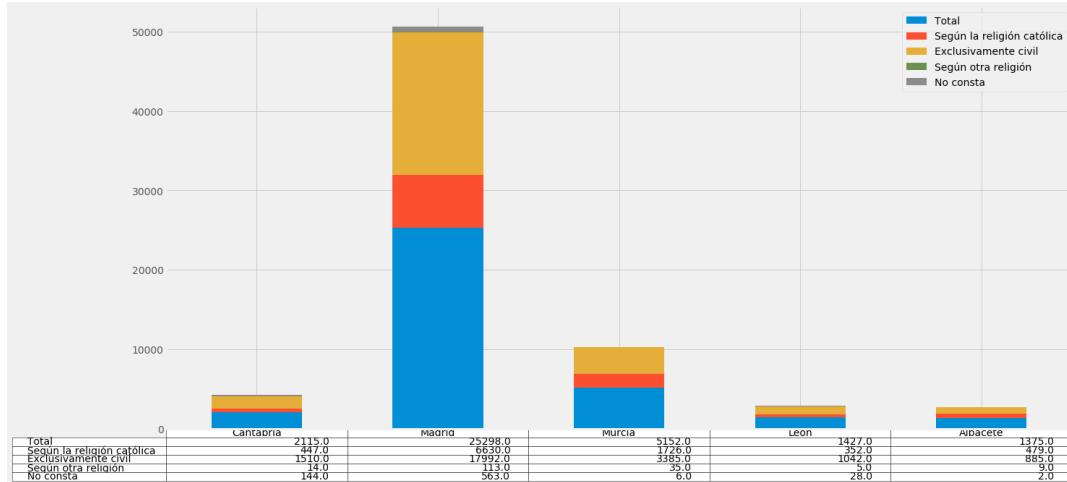
Equivale a:

```
provincias = ['Cantabria', 'Madrid', 'Murcia', 'León', 'Albacete']
df = ax.Total.loc[provincias]
ax = df.plot.bar(stacked=True,
                 table=True, title='Total Matrimonios en 2017')
```

```

provincias = ['Cantabria', 'Madrid', 'Murcia', 'León', 'Albacete']
df = ax.Total.loc[provincias]
ax = df.plot.bar(
    stacked=True,
    table=True,
    title='Total Matrimonios en 2017'
)

```



Generalmente las opciones más usadas en cada tipo de gráfico (título, ejes y posición, color, tamaño de línea, ...) son directamente accesibles desde `pandas.DataFrame.plot()`.

Para el resto, usar `matplotlib` refiriéndonos al objeto a modificar:

```

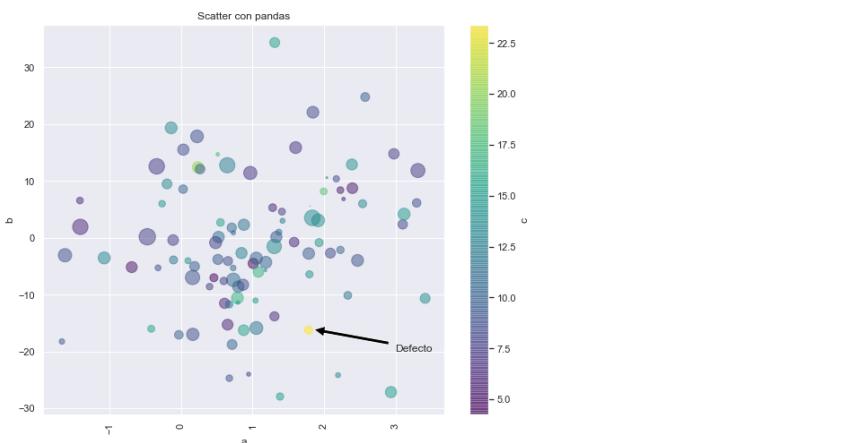
ax = df.plot(kind='scatter', ...) # df.plot.scatter(...)
ax.set_yticks(rotation='vertical')

```

## Tipos de gráficos disponibles

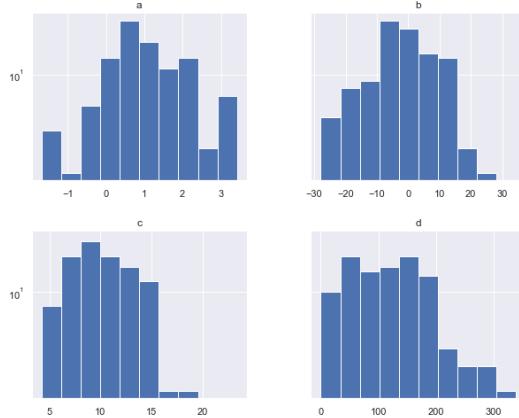
### ► Gráfico de dispersión (scatter)

```
df.plot.scatter(  
    x='a', y='b', # nombres de las columnas del dataframe  
    c='c', # columna con datos de color  
    s=df['d'], # tamaño de los puntos  
    colormap='viridis', # paleta de color  
    alpha=.5, # transparencia  
    title='Scatter con pandas', # título  
    rot='vertical', # rotar etiquetas del eje 'x'  
)
```



► Histograma

```
df.hist(sharey=True, log=True)
```

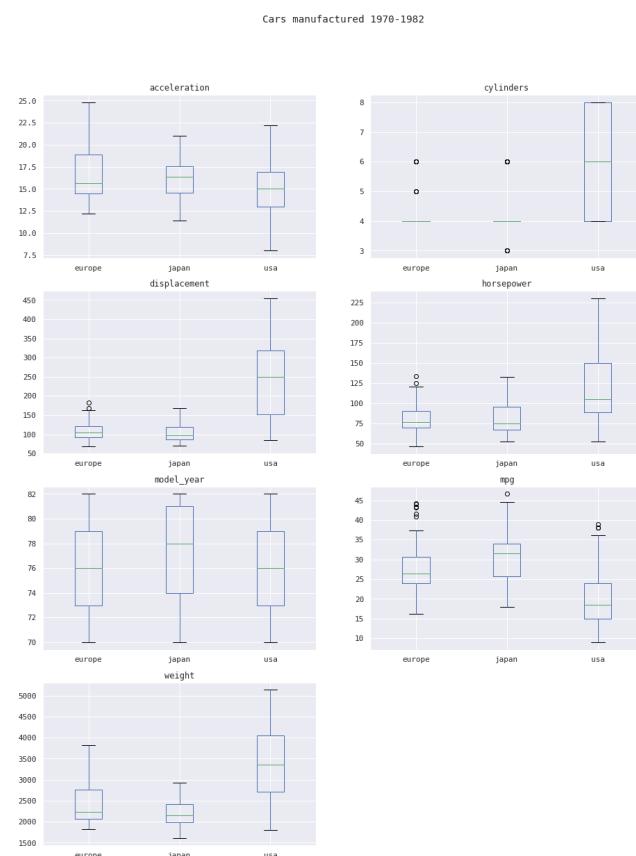


► **Series temporales:** aquellos donde el índice del DataFrame tiene propiedades de “índice temporal”



## ► Boxplots

```
df.boxplot(by='origin', ax=ax)
```



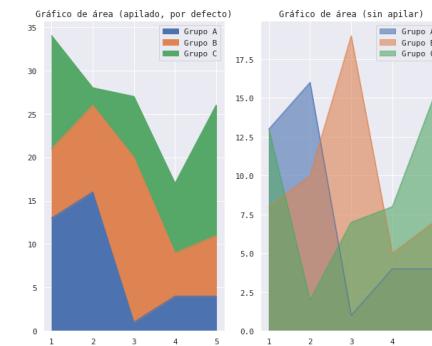
⋮

## ► Área

```
np.random.seed(0)
N = 5
data = pd.DataFrame(
    {'Grupo A': np.random.randint(1, 20, N),
     'Grupo B': np.random.randint(1, 20, N),
     'Grupo C': np.random.randint(1, 20, N)},
    index=range(1, N+1)
)

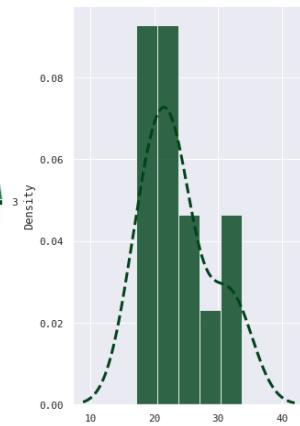
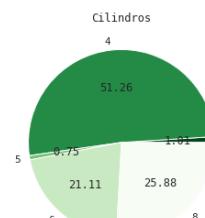
ax = plt.subplot(1, 2, 1)
data.plot.area(
    stacked=False,
    alpha=.6,
    title='Gráfico de área',
    ax=ax)

ax = plt.subplot(1, 2, 2)
data.plot.area(
    title='Gráfico de área (apilado)',
    ax=ax)
```



► Circular

```
ax = df1.plot.pie(cmap=cmap,
                   title='Cilindros',
                   autopct='%.2f', ax=ax)
```



## Parámetros opcionales

## `pd.DataFrame.plot()`

Es posible pasar argumentos para realizar personalizaciones rápidas.

- ▶ Existen opciones específicas para cada tipo de visualización
- ▶ También hay opciones **comunes** a todos ellos
- ▶ Además, podremos usar las primitivas de `matplotlib`, bien como argumentos adicionales o sobre los ejes

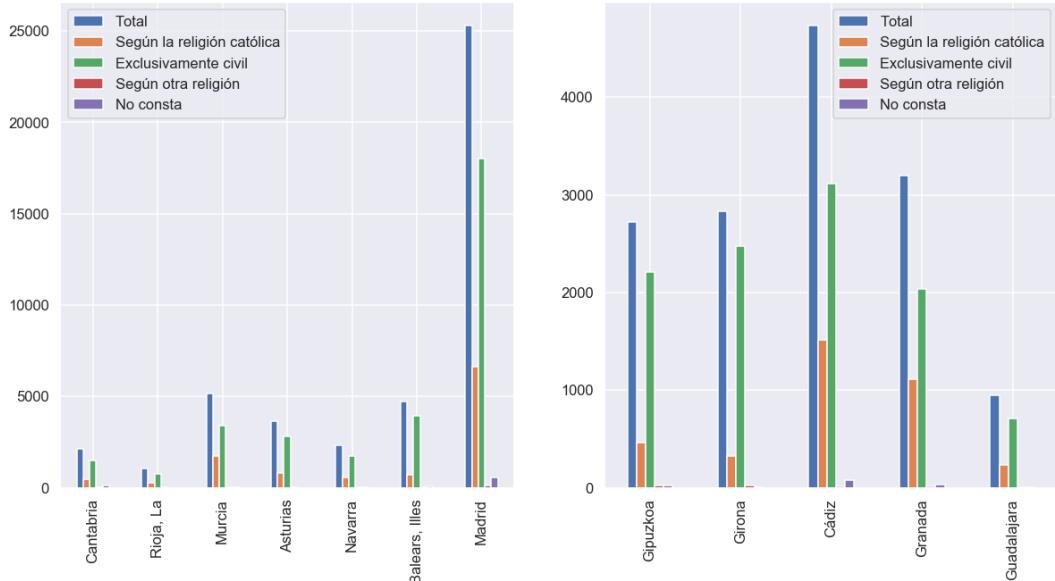
## NaN

- ▶ Por defecto ignora los datos ausentes (NaN)
- ▶ Podemos “rellenar los huecos”: `df.fillna()`
  - ▶ propagando el último valor
  - ▶ o con valores preestablecidos
- ▶ O interpolar `df.interpolate()`

Interactivo/Práctico

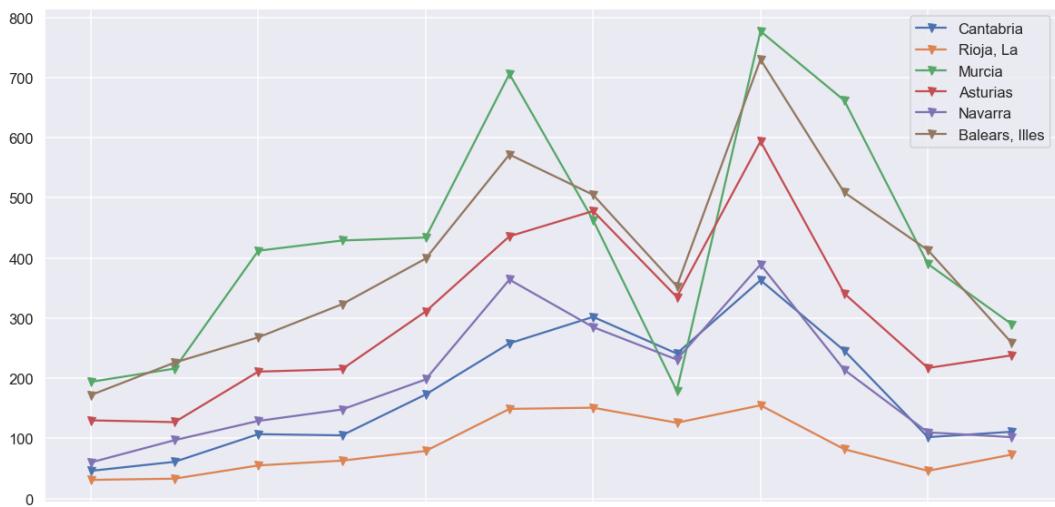
Abrir [cuaderno jupyter][nb\_pract]

## 1. ax: ejes sobre los que dibujar

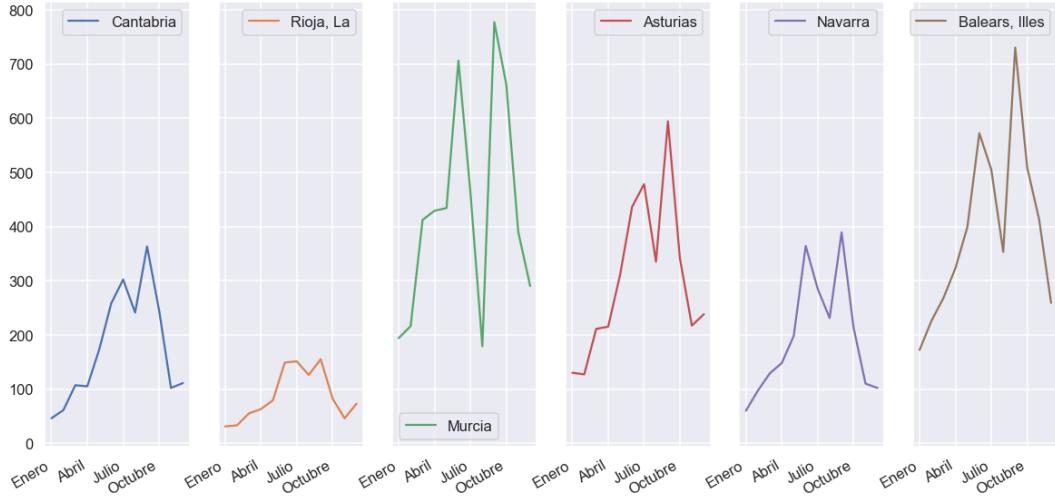


## 2. subplots [True|False]: subplot para cada columna dibujada

► layout: (n\_filas, n\_columnas), opcional con subplot=True

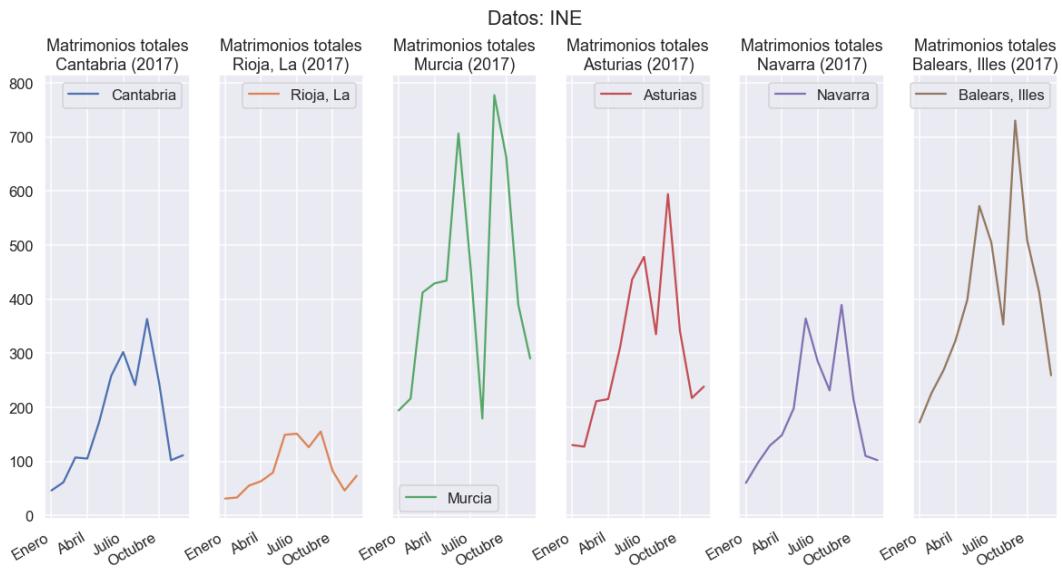


### 3. sharex/sharey: compartir ejes (subplots)

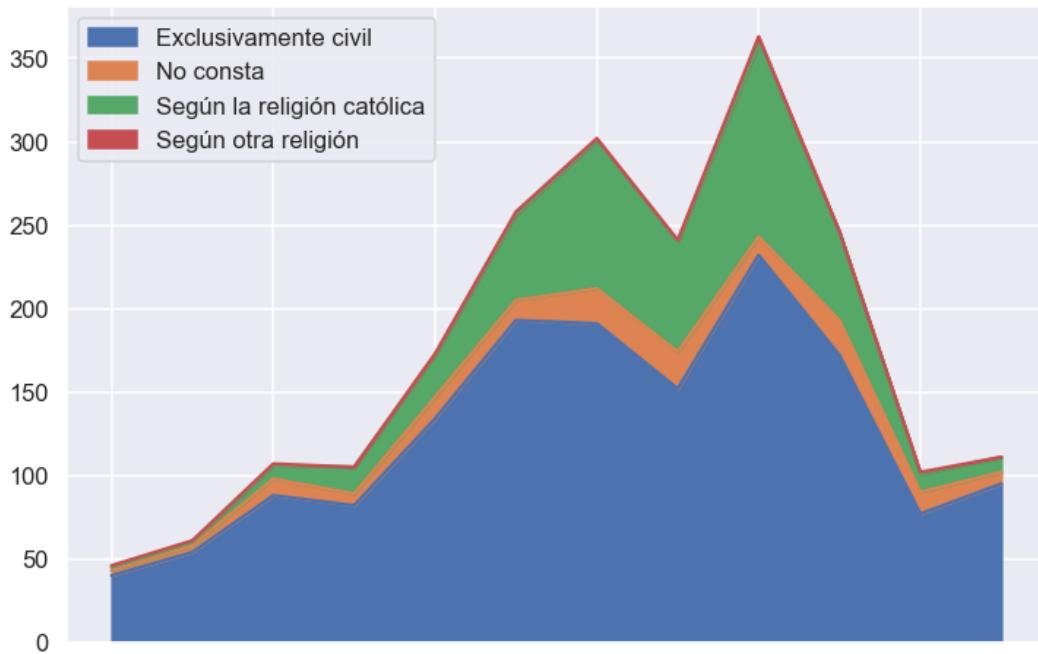


### 4. title:

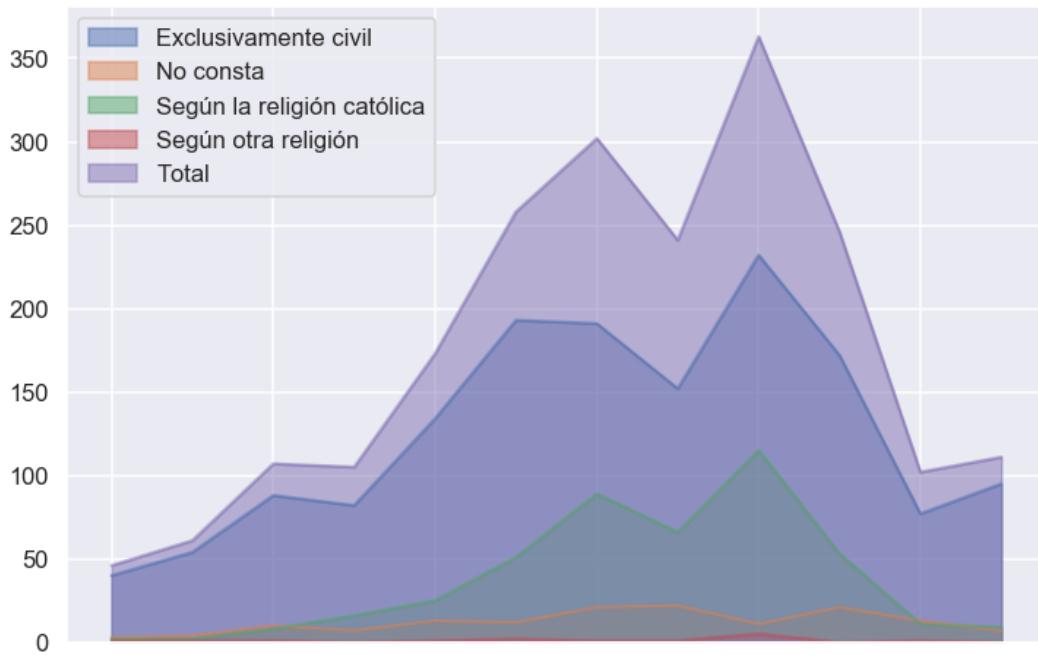
- ▶ str: título de la figura
- ▶ list(str): título de cada subplot



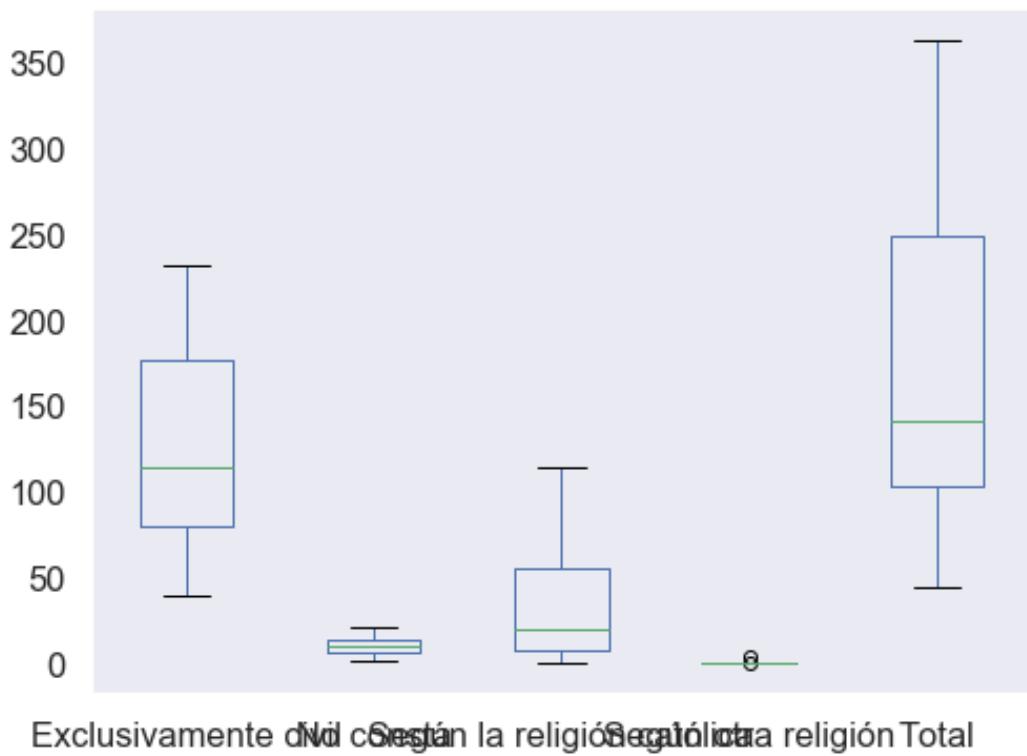
5. figsize: tamaño de la figura



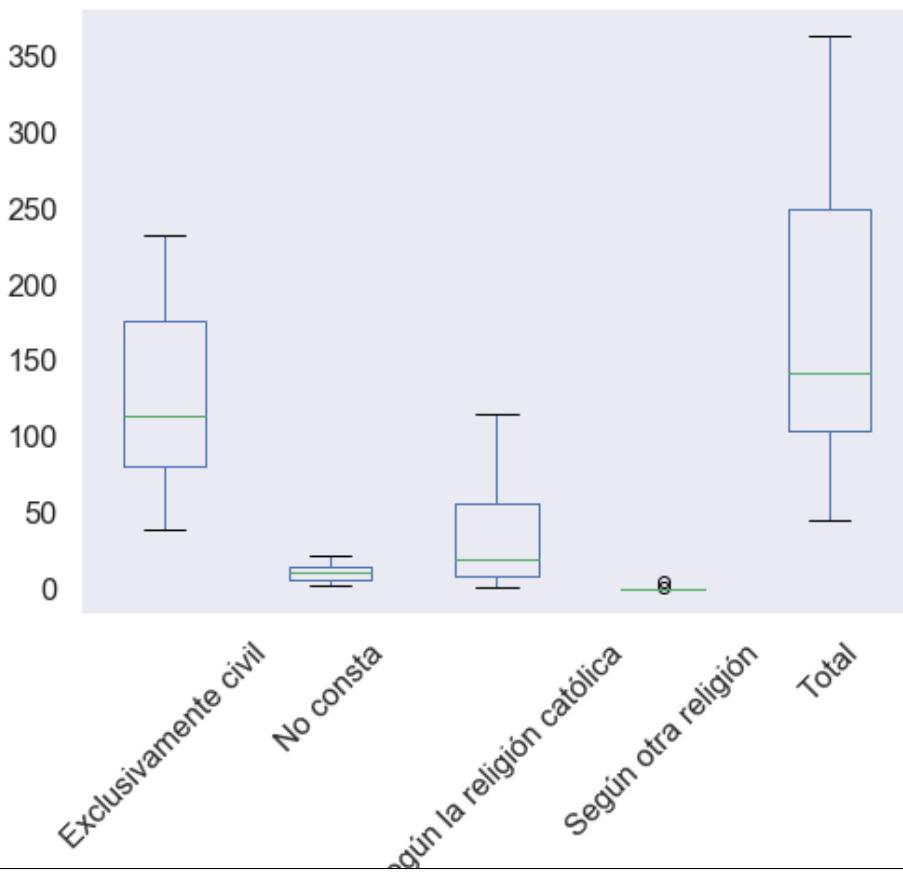
6. stacked [True|False]: apilar datos



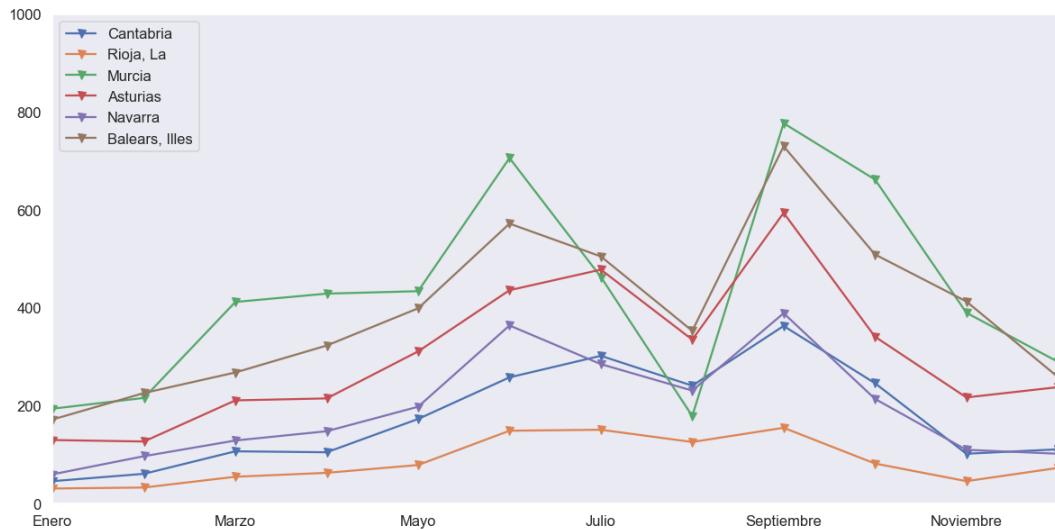
7. grid [True|False]: dibujar cuadrícula



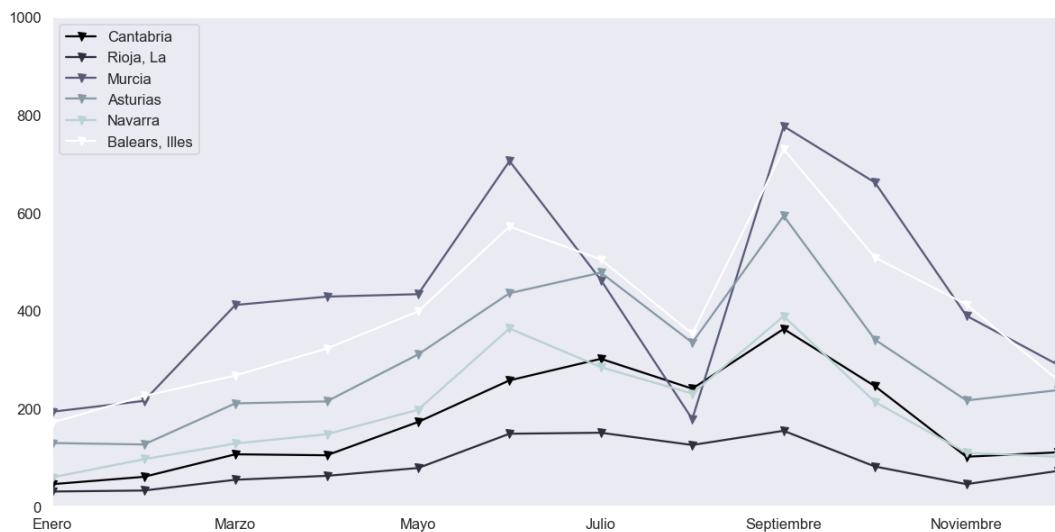
8. rot: 'horizontal', 'vertical', o número (en grados)



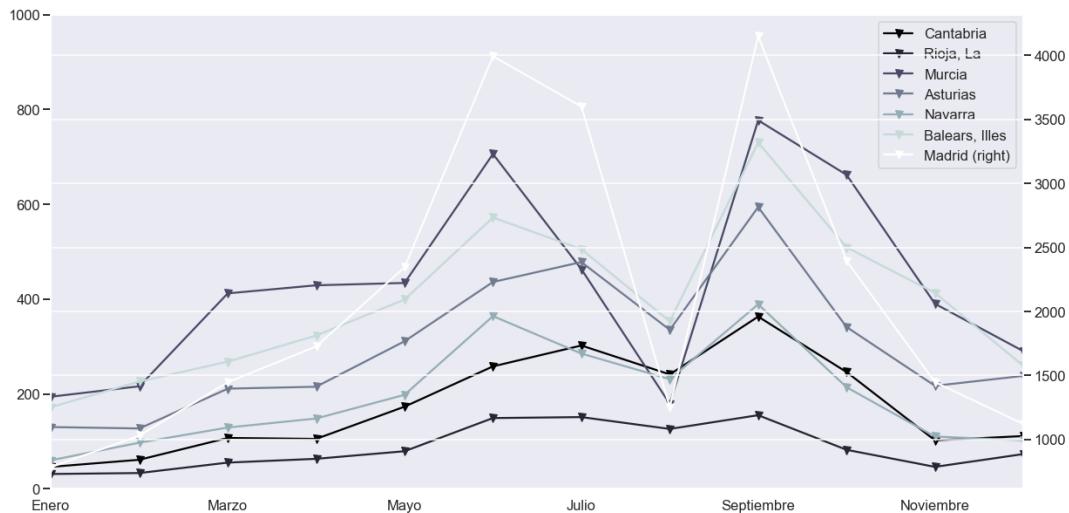
9. xlim, ylim: tuplas (lo, hi) para delimitar visualización



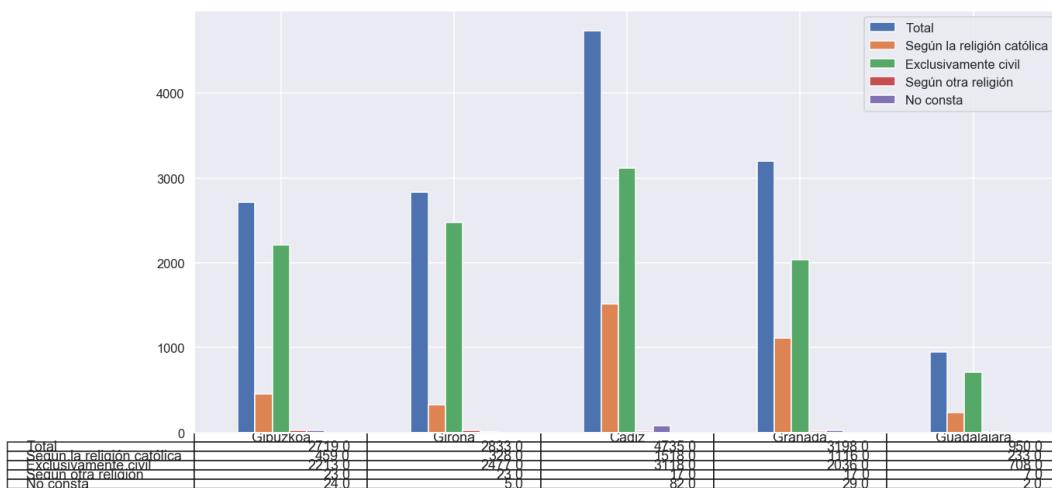
10. colormap: mapa de colores (matplotlib.cm.)



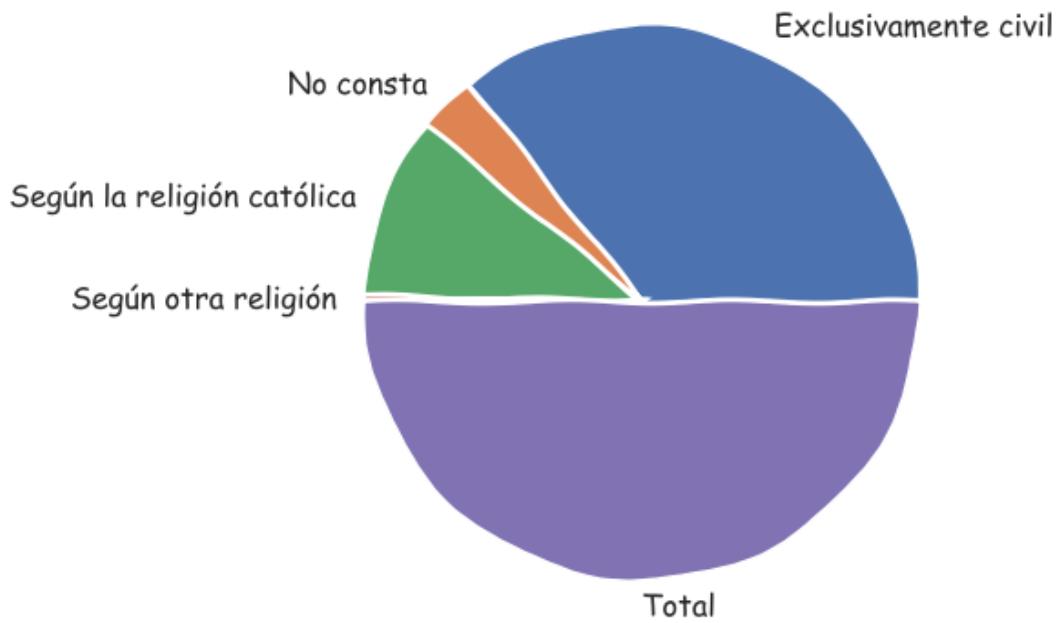
11. secondary\_y: segundo eje de ordenadas



12. table [True|False]: mostrar tabla bajo el gráfico



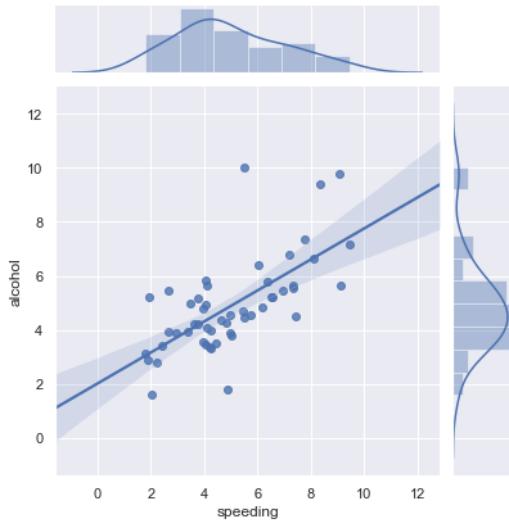
**BONUS** Generar un gráfico circular con estilo xkcd



Alternativas: [seaborn](#)

Simplifica (aún más) ciertas tareas de visualización

```
sns.set() # tema por defecto  
crashes = sns.load_dataset("car_crashes")  
  
with sns.color_palette("husl", 8):  
    sns.jointplot(  
        "speeding",  
        "alcohol",  
        crashes,  
        kind='reg'  
)
```



Con matplotlib: ...

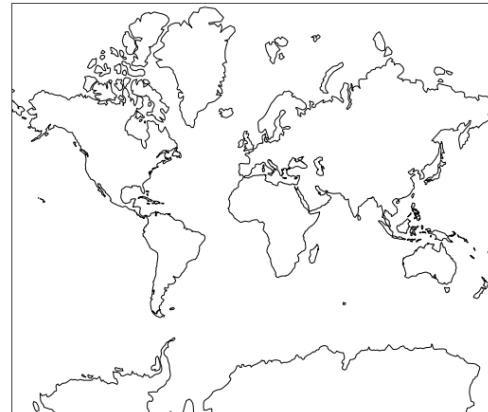
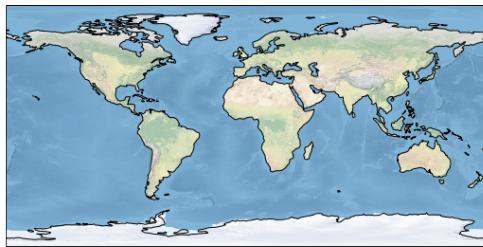
```
from scipy.stats import gaussian_kde  
  
ax1 = plt.subplot2grid((4, 4), (1, 0), colspan=3, rowspan=3)  
ax2 = plt.subplot2grid((4, 4), (0, 0), colspan=3)  
ax3 = plt.subplot2grid((4, 4), (1, 3), rowspan=3)  
  
crashes.plot.kde(y='speeding', ax=ax2, sharex=ax1, legend=None)  
crashes.plot.hist(y='speeding', bins=6, ax=ax2, sharex=ax1, normed=True,  
                  legend=None, alpha=.5, color='red')  
crashes.plot.scatter(x='speeding', y='alcohol', ax=ax1, color='red', s=50)  
  
ax2.set_ylabel('')  
ax2.set_yticks([])  
ax2.set_yticklabels([])  
  
# No está soportado directamente el rotado en kde  
kde_speeding = gaussian_kde(crashes.alcohol)  
y = np.linspace(np.amin(crashes.alcohol), np.amax(crashes.alcohol), 100)  
ax3.plot(kde_speeding(y), y)  
crashes.plot.hist(y='alcohol', ax=ax3, sharey=ax1, normed=True, legend=None,  
                  orientation='horizontal', alpha=.5, color='red')
```

► Hexbin

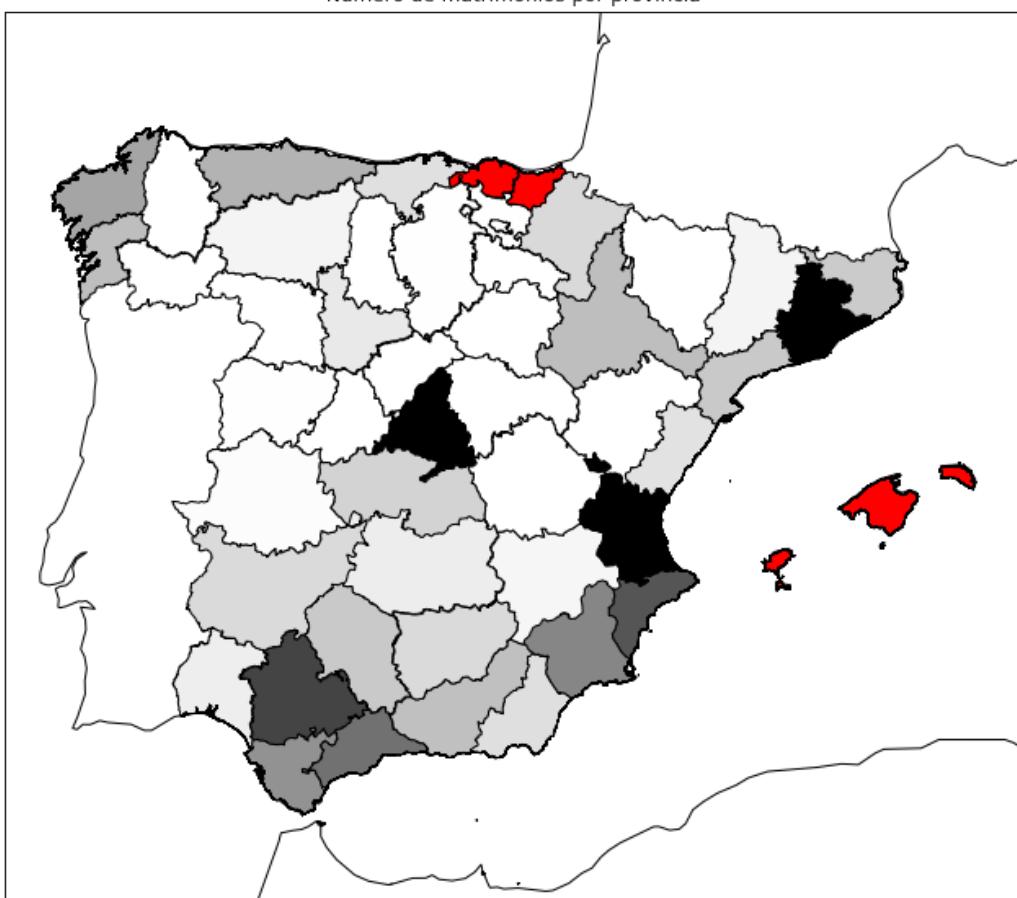
```
x, y = np.random.multivariate_normal(mean, cov, 1000).T
with sns.axes_style("white"):
    sns.jointplot(x=x, y=y, kind="hex", color="k");
+ ejemplos
```

## Geovisualización

- ▶ Librería standard de facto: matplotlib-toolkit (**obsoleta**)
- ▶ Oficialmente reemplazada por cartopy
- ▶ Alternativa: folium



Número de matrimonios por provincia



Folium  
ejemplos online

Cuadernos jupyter

- ▶ matplotlib/pyplot
- ▶ pandas
- ▶ caso práctico
- ▶ **Soluciones** caso práctico