

# Librerías para ciencia de datos – Numpy

Las herramientas del científico de datos

Juan Manuel Moreno — [jmmoreno@profesorescol.imf.com](mailto:jmmoreno@profesorescol.imf.com)



# ÍNDICE

1. Objetivos unidad 2
2. Numpy



# 01

# Objetivos unidad 2

# 1.– Objetivos Unidad 2

- Conocer los principales tipos de archivos con los que podemos trabajar en Python.
- Saber realizar operaciones de lectura y escritura de archivos.
- Aprender a utilizar arrays uni-dimensionales y multi-dimensionales.
- Diferenciar las diferentes dimensiones de un array.
- Conocer las principales funciones de cálculo numérico, vectorial y matricial sobre arrays con Numpy.
- Comprender qué es un objeto Series y cuál es su relación dentro de los Dataframes.
- Conocer las funcionalidades y transformaciones fundamentales que podemos realizar en dataframes.



# 02

# Numpy

### 2.1.- ¿Qué es Numpy?

- Se trata de una librería dedicada al cálculo numérico en Python (**NUM**eric **PY**thon).
- Proporciona estructuras de datos tales como vectores y, matrices unidimensionales o multidimensionales.
- Los objetos (arrays) que genera son más ligeros en memoria que las listas.
- Los arrays son más rápidos en procesamiento que las listas.
- Una operación numérica se aplica sobre todo un array.
- Puede ser necesaria su instalación `pip install numpy`
- Se utiliza comúnmente con el alias `np`
- Para importar la librería se emplea `import numpy as np`





### 2.2.- Numpy Básicos

- Para crear un array se utiliza la función `numpy.array()`
- Todos los elementos de la función array se pasan en forma de lista multi-dimensional
- A través de `ndim()` podemos conocer el número de dimensiones de un array.
- Con la función `shape()` podemos conocer cuántos elementos tiene cada dimensión del array
- Podemos convertir otras estructuras de datos como arrays (listas, tuplas, diccionarios)
- Cuando el array sea de más de una dimensión tendremos disponibles atributos para sus ejes, `axes = 1` (filas), `axes = 0` (columnas)

```
import numpy as np

# Creamos de nuevo un array
arr = np.array([23, 42, 58])

arr

array([23, 42, 58])
```

```
# Creamos nuestro primer array

arr = numpy.array([10, 20, 98, 34])

arr

array([10, 20, 98, 34])
```

## 2.- Numpy

### 2.2.- Numpy Básicos

```
>>> a[(0,1,2,3,4), (1,2,3,4,5)]  
array([1, 12, 23, 34, 45])
```

```
>>> a[3:, [0,2,5]]  
array([[30, 32, 35],  
       [40, 42, 45],  
       [50, 52, 55]])
```

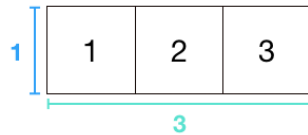
```
>>> mask = np.array([1,0,1,0,0,1], dtype=bool)  
>>> a[mask, 2]  
array([2, 22, 52])
```

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

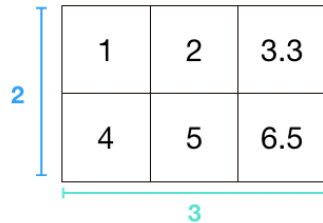


# Anatomy of a NumPy array

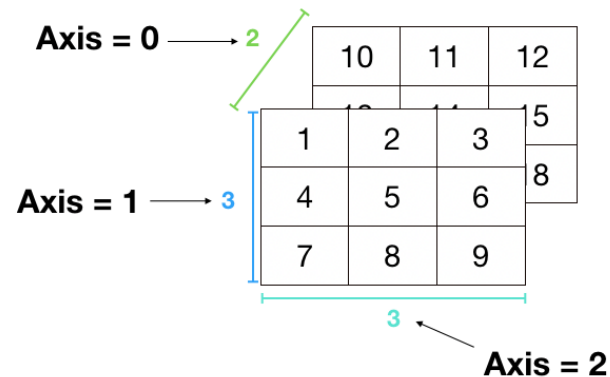
## Data



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



1	2	3.3
4	5	6.5



10	11	12
1	2	3
4	5	6
7	8	9

## NumPy

```
array([1, 2, 3])
```

```
array([[1. , 2. , 3.3],
       [4. , 5. , 6.5]])
```

```
array([[[ 1, 2, 3],
        [ 4, 5, 6],
        [ 7, 8, 9]],
       [[10, 11, 12],
        [13, 14, 15],
        [16, 17, 18]]])
```

## Details

- Names: Array, vector
- 1-dimensional
- Shape = (1, 3)

- Names: Array, matrix
- More than 1-dimension
- Shape = (2, 3)

- Names: Array, matrix
- More than 1-dimension
- Shape = (2, 3, 3)

## 2.- Numpy

### 2.3.- Numpy Funciones

- **reshape**: Nos sirve reestructurar las dimensiones y elementos de un array.
- **ravel**: Devuelve un array plano (devuelve una lista).
- **flatten**: Devuelve una copia.
- **hstack**: Apila (stack) los array de forma horizontal (equivalente a concatenate axis = 1).
- **vstack**: Apila (stack) los array de forma vertical (equivalente a concatenate, axis = 0).
- **hsplit**: Divide (Split) los array de forma horizontal (equivalente a split axis = 1).
- **vsplit**: Divide (Split) los array de forma vertical (equivalente a split axis = 0).

```
big = np.arange(27).reshape(3, 3, 3)
```

```
big
```

```
array([[[ 0,  1,  2],  
        [ 3,  4,  5],  
        [ 6,  7,  8]],  
       [[ 9, 10, 11],  
        [12, 13, 14],  
        [15, 16, 17]],  
       [[18, 19, 20],  
        [21, 22, 23],  
        [24, 25, 26]]])
```

### 2.4.- Numpy Atributos

- **ndim**: Dimensiones de un array.
- **size**: Número de elementos (array plano).
- **itemsize**: Espacio que ocupa en bytes cada elemento del array
- **nbytes**: Tamaño de todo el array en bytes.
- **T**: Transpuesta del array

```
big.T
```

```
array([[ [ 0,  9, 18],  
        [ 3, 12, 21],  
        [ 6, 15, 24]],  
  
       [[ 1, 10, 19],  
        [ 4, 13, 22],  
        [ 7, 16, 25]],  
  
       [[ 2, 11, 20],  
        [ 5, 14, 23],  
        [ 8, 17, 26]]])
```

### 2.5.- Numpy cambiando el tipo del array

- **tolist**: Convierte a lista el array

```
lista = big.tolist()
```

```
print(lista)  
print(type(lista))
```

```
[[[0, 1, 2], [3, 4, 5], [6, 7, 8]], [[9, 10, 11], [12, 13, 14], [15, 16, 17]],  
[[18, 19, 20], [21, 22, 23], [24, 25, 26]]]  
<class 'list'>
```

- **astype(tipo)**: Convierte a otro tipo de dato los elementos del array

```
big.astype(float)
```

```
array([[ 0.,  1.,  2.,  
        3.,  4.,  5.,  
        6.,  7.,  8.]
```

```
      [ 9., 10., 11.,  
       12., 13., 14.,  
       15., 16., 17.]
```

```
      [18., 19., 20.,  
       21., 22., 23.,  
       24., 25., 26.]])
```



### 2.6.- Copias y vistas

- Al igual que las listas los arrays comparten posiciones en nuestra memoria física, por lo tanto, si no utilizamos los comandos necesarios para realizar una copia de un array correctamente, los cambios que hagamos en el array\_a, se verán reflejados en el array\_b. Para realizar una copia correctamente, emplearemos la función copy()

```
array_dos = np.arange(0, 31)
```

```
c = array_dos[0:5]
```

```
c[:] = 100  
c
```

```
array([100, 100, 100, 100, 100])
```

```
array_dos
```

```
array([100, 100, 100, 100, 100,  5,  6,  7,  8,  9, 10, 11, 12,  
       13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25,  
       26, 27, 28, 29, 30])
```

```
array_copiado = array_1.copy()
```

Realizamos una modificación en el array original

```
array_1[1] = [2200, 34]
```

```
array_1
```

```
array([[10000,  3],  
       [ 2200, 34],  
       [   4,  5]])
```

Vemos si los cambios aparecen en nuestra copia

```
array_copiado
```

```
array([[10000,  3],  
       [   2,  3],  
       [   4,  5]])
```

### 2.7.- Operaciones

- **sum**: Suma de los elementos de un array.
- **empty**: Genera un array con valores arbitrarios sin inicializar.
- **ones**: Un array con unos.
- **zeros**: Un array con ceros.
- **round**: Redondear a n decimales un array
- **fill**: Permite rellenar un array con un valor pasado como parámetro.
- **mean**: Realiza la media del array
- **dot**: Producto de matrices
- **min**: Valor mínimo del array.
- **max**: Valor máximo del array.
- **argmin**: Posición del índice con el valor mínimo
- **argmax**: Posición del índice con el valor máximo
- **sort**: Ordenar elementos de un array.

```
e = np.array([[[1, 0], [0, 0]],  
              [[1, 1], [1, 0]],  
              [[1, 0], [0, 1]]])  
  
e.sum(axis = 0)  
  
array([[3, 1],  
       [1, 1]])
```

```
zero_array = np.zeros(20)  
zero_array[5:15].fill(5)  
zero_array  
  
array([0., 0., 0., 0., 0., 5., 5., 5., 5., 5., 5., 5., 5., 5., 5., 0., 0.,  
       0., 0., 0.])
```

### 2.8.- Aleatorios y distribuciones

- `random.rand`: Valores aleatorios
- `random.randint`: Valores aleatorios enteros
- `random.random`: Valores aleatorios en un intervalo de [0.0 a 1.0]
- `random.randn`: Valores aleatorios siguiendo una distribución normal.
- `random.beta`: Valores aleatorios siguiendo una distribución beta.
- etc...

```
rand = np.random.rand(10)
rand
```

```
array([0.00679214, 0.87719023, 0.96415461, 0.15073363, 0.53432306,  
       0.46579972, 0.65413353, 0.17148798, 0.26835114, 0.53212125])
```

```
randn = np.random.randn(10)
randn
```

```
array([ 0.67349417,  0.44322206,  1.14267717,  0.09392948,  0.67466901,  
       -1.00494561,  1.29410641, -0.9576192 ,  1.59516205, -0.68454205])
```

```
randint = np.random.randint(20, 100, 10)
randint
```

```
array([54, 46, 64, 94, 46, 74, 24, 56, 35, 93])
```

## Seguimiento práctico del contenido

A partir de aquí, vamos a ver las principales funciones, operaciones y curiosidades que podemos realizar con Numpy.

### 2\_1\_Numpy.ipynb



IMF

Smart Education