NUMPY

Es una biblioteca para facilitar el manejo de vectores, matrices o tensores.

```
#Importing Numpy with an Commonly used alias np
import numpy as np
```

Crear una matriz

```
import numpy as np
a=np.array([[10,2],[30,4],[60,7]])
```

Operaciones

Multiplicación:

```
np.multiply(a,2)
np.multiply(a,b)
np.matmul(a,b) #operador @   a@b
np.add(a, 5)
np.divide(a, 2)
np.power(a, 3)
```

Aplicar una función

```
a=np.array([10,20,30,40,50])
f=lambda x: x+5
np.array([f(x) for x in a])
```

Añadir elementos

Con `append`

Información:

Máximo	`np.max(a)`
Valores únicos	`np.unique(a)`
Dimensiones	`a.ndim`
Forma	`a.shape`
Tipo de datos	`a.dtype`

Transformaciones:

Tipo de datos:	`a.astype(np.uint8)  #float64 ...` Tipos en Numpy
Unir tensores	`np.append(a, b)`
Concatenar columnas	`np.column_stack((a,b))`
Dimensiones:	`a.reshape(12)  #-1 automáticamente.`
	`a.T  #matriz transpuesta`

Meshgrid: conbinar dos vectores. Se usa para crear gráficos en 3D.

Guardar y cargar.

```
np.genfromtxt
np.savetxt
```

BIBLIOTECA: **https://numpy.org/**

Arrays vacíos, unos y ceros

```
np.empty((2, 3))    # Matriz vacía, con valores residuales de la memoria

np.zeros((3, 1)) # Matriz de ceros

np.ones((3, 2)) # Matriz de unos
```

Con el sufijo `_like`, podemos crear matrices **con la misma dimensión que una dada**. Son `empty_like`, `zeros_like` y `ones_like`

```
a = np.zeros((3, 2))

np.empty_like(a) # Matriz vacía con la forma de a

np.zeros_like(a) # Matriz de ceros con la forma de a

np.ones_like(a) # Matriz de unos con la forma de a
```

Para crear una **matriz identidad**, esto es, una matriz cuadrada con unos en la diagonal, podemos usar la función `identity`. Para un caso un poco más general, sin que sean matrices necesariamente cuadradas, podemos usar `eye`:

```
np.identity(3) # Matriz identidad de tamaño 3

np.identity(5) # Matriz identidad de tamaño 4

np.eye(4, 3) # Matriz de 4x3 con unos en una diagonal y ceros en el resto de elementos

np.eye(4, 3, k=-1) # Con el parámetro k podemos controlar qué diagonal está llena de unos
```

Arrays a partir de listas

Cuando conocemos todos los valores del array antes de crearlo, podemos utilizar la función `array` y pasarle como argumento una lista, tupla o, en general, una secuencia.

```
np.array( [1, 2, 3] )    #Lista

np.array( [ [1, -1], [2, 0] ] )    #Lista de listas

np.array( (0, 1, -1) )   # Tupla

np.array(range(5))
```

Rangos numéricos

NumPy también ofrece funciones para crear rangos numéricos, particiones de intervalos, discretizaciones o como queráis llamarlos. Por ejemplo, la función `arange` está pensada rangos de números enteros, de manera similar a la función `range` de la biblioteca estándar de Python:

```
np.arange(4)

np.arange(2, 5)

np.arange(2, 10, 3)
```

Las funciones `linspace` y `logspace`, aceptan como argumento el número de elementos en lugar del paso: #Fijaros que sí que se obtiene el último valor.

```
     np.linspace(0, 1, 11) # 11 puntos equiespaciados entre 0 y 1

     np.logspace(2, 5, 4, base=10) # 4 puntos equiespaciados según una escala logarítmica entre 10^2 y 10^5


#Resumen de parámetros linspace:

linspace(start,
         stop,
         num=50,
         endpoint=True,
         retstep=False,
         dtype=None,
         axis=0
         )
```

Podemos indicar el tipo de los datos. <-Tipos de datos->

```
a=np.array([10,20,30,40,50],dtype=np.int32)
```