.a) Introducción

NumPy (**Num**erical **Py**thon) es una librería de Python utilizada para trabajar con datos numéricos. Incluye funciones y estructuras de datos que permiten realizar una gran variedad de **operaciones matemáticas**.

Para empezar a utilizar NumPy, debemos instalarla desde la línea de comandos con `pip install numpy` e importarla a nuestro código:

```
import numpy as np
```

np es el alias más común que se utiliza para importar numpy.

Debemos recordar trabajar con entornos virtuales cuando instalamos nuevos paquetes.

.b) Array: El vector NumPy

En Python, se utilizan listas para almacenar datos.

Numpy nos provee del **array**, una estructura tipo vector para operar con los datos. Estos son arreglos numéricos parecidos a listas en Python. De hecho tienen propiedas muy parecidas como ser mutables y poder hacer slicing.

Los arrays de numpy son más rápidos y compactos que las listas.

Podemos crear un array de numpy por medio de la función `np.array()`, en la que enviamos como argumento una lista:

```
x = np.array([1, 2, 3, 4]
```

Acabamos de crear el array de numpy llamado x, que contiene 4 valores.

Podemos acceder a sus elementos utilizando sus índices, comenzando desde el 0.

```
print(x[0])
```

Los arrays de numpy son homogéneos, lo que significa que pueden contener datos de un solo tipo, cosa que no ocurría con una lista, donde podíamos insertar datos de todos tipos.

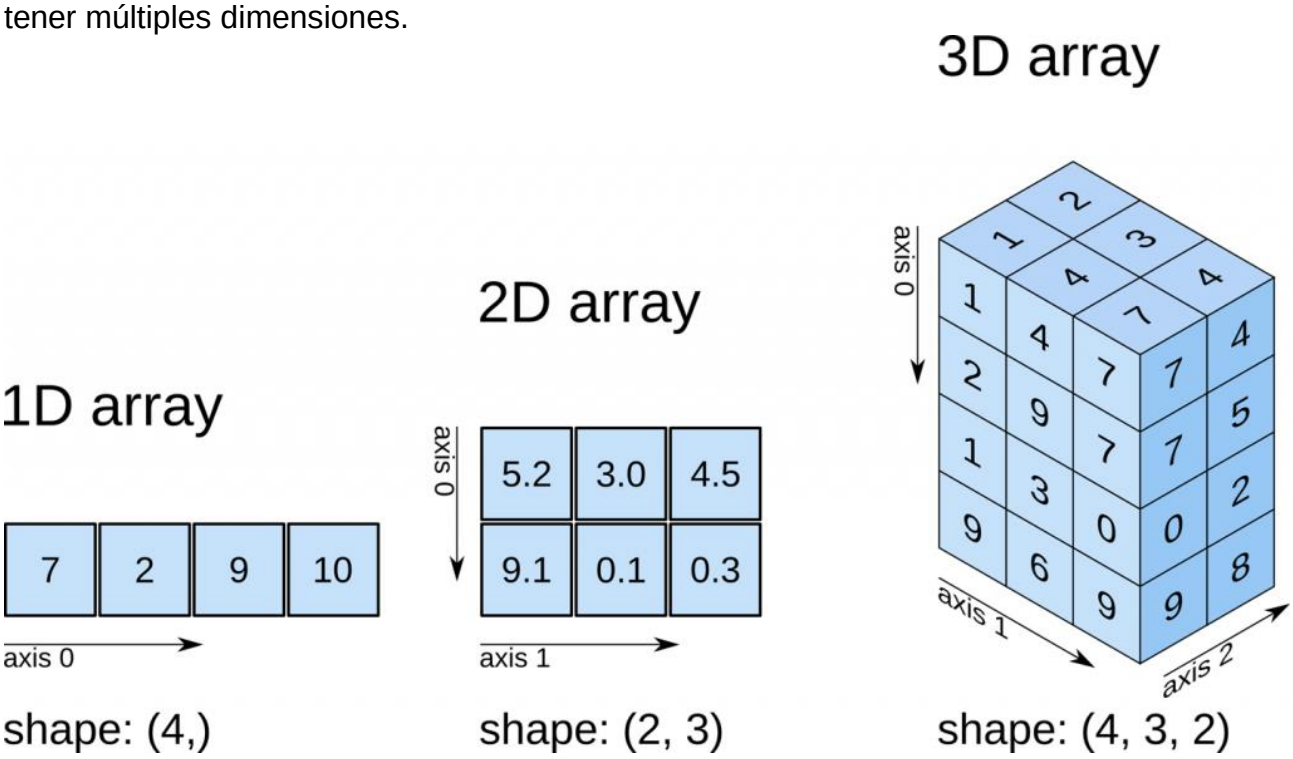
Los arrays son n-dimensionales. Esto quiere decir que pueden ser de muchas dimensiones dependiendo si tienen varias filas y columnas.

- Un array unidimensional es una sola columna o fila de alguna tabla, tal y como una lista de Python
- Un array bidimensional es lo que conocemos como matriz y es la forma en la que podemos representar tablas de datos en NumPy.
- Un array de tres dimensiones o más es una matriz de matrices, mejor conocida como tensor.

.c) Trabajando con arrays

.c.i. Explorando el array

Los arrays de numpy suelen llamarse ndarrays (N-dimensional array), porque pueden tener múltiples dimensiones.



Por ejemplo:

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

Estas líneas crearán un array bidimensional, que tiene 3 filas y 3 columnas. El print va a mostrar el valor de la 2ª fila, 3ª columna.

Podemos acceder a las propiedades de un array:

```
x = np.array([[ 1,  2,  3 ],[ 4,  5,  6 ],[ 7,  8,  9 ]])
print(x.ndim) # 2 -> la dimensión del array
print(x.size) # 9 -> la cantidad de elementos del array
print(x.shape) # (3,3) -> tupla de enteros que muestra la
                    cantidad de elementos almacenados en
                              cada dimensión del array
```

- Para vectores, el único valor enviado como argumento en shape() es la cantidad de elementos del vector.
- Para matrices, el primer valor son las filas y el segundo las columnas.

- Para un tensor, el primer valor es la “profundidad”, el segundo las filas y el tercero las columnas.

Ejemplos:

```
x = np.array([[ 1, 2, 3, 12 ],[ 4, 5, 6, 9 ],[ 7, 8, 9, 7 ]])
print(x.ndim) # 2
print(x.size) # 12
print(x.shape) # (3, 4)
y=np.array([[[ 1, 2, 3, 0 ],[ 4, 5, 6, 0 ],[ 7, 8, 9, 0 ]],[[ 10, 11, 12, 0 ],[ 13, 14, 15, 0 ],[ 16, 17, 18, 0 ]]])
print(y.ndim) # 3
print(y.size) # 24
print(y.shape) # (2,3,4)
```

.c.ii. Modificando el array

Es posible modificar el array por medio de sus funciones:

```
x = np.array([14, 1, 3])
x = np.append(x,7) # agrega un elemento
x = np.delete(x, 0) # elimina un elemento del array
x = np.sort(x) # ordena el array
x = np.arange(2,10,3)  #  crea un array con un rango de
                    elementos a intervalos espaciados
       (similar a range de Python).
                              El resultado es el array [2, 5, 8]
```

.c.iii. Cambiando la forma del array

La forma es el número de filas y columnas del array.

Por ejemplo:

```
x = np.arange(1,13)
```

Es un array unidimensional que contiene 12 elementos.

Podemos cambiar la forma de nuestr0 array, por ejemplo a 3 filas y 4 columnas

```
y = x.reshape(3,4)
```

¡Atención! Al utilizar el método reshape, el array final debe tener el mismo número de elementos que el array original.

Probemos convertir el siguiente array:

```
x = np.array([[1,2],[3,4],[5,6]]
y = x.reshape(6)
```

y resulta un array unidimensional de 6 elementos. Este mismo resultado se puede obtener por medio de la funcion flatten()

.c.iv. Indexado y rebanado

Los arrays de numpy pueden indexarse y rebenarse de la misma forma que las listas de Python.

Por ejemplo:

```
x = np.arange(1,10)
print(x[0:2])
print(x[5:])
print(x[:2])
print(x[-3:])
```

Los índices negativos comienzana contar desde el último elemento, comenzando por -1.

.d) Condiciones

Podemos poner una condición al indexar un array para ir seleccionando los elementos que la cumplan.

```
x=np.arange(1,10)
print(x[x<4])
```

Las condiciones pueden combinarse con los operadores & (and) y | (or).

```
print(x[(x>5) & (x%2==0)]) # números pares mayores a 5
y = (x>5)&(x%2==0)
```

Crea un array y con valores booleanos que indican si cada elemento cumple o no con las condiciones

.e) Operaciones

Numpy nos proporciona gran cantidad de funciones para operar con los arrays.

```
x = np.arange(1,10)
print(x.sum()) # suma
print(x.min()) # valor mínimo
print(x.max()) # máximo
y = x*2 # crea un array con los elementos del array x
      multiplicados por 2.
```

Numpy interpreta que la operación debe ser ejecutada en cada elemento. Esto se llama **broadcasting.**

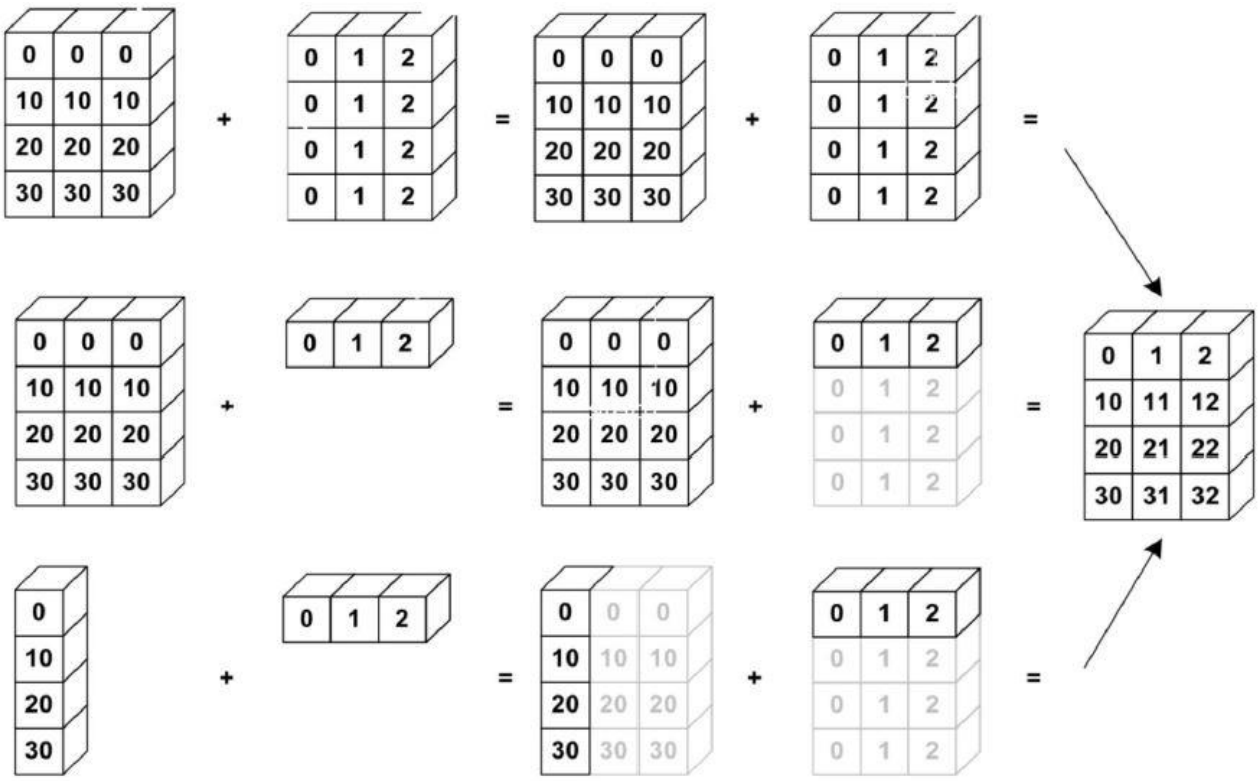
.f) Estadística

```
x = np.array([14,18,19,24,26,33,42,55,67])
print(np.mean(x)) # media
print(np.median(x)) # mediana
print(np.std(x)) # desviación standard
```

.g) Broadcasting

NumPy me permite operar entre arreglos de diferente estructura, transformándolos para que tengan el mismo tamaño para que no se generen errores.

Así es el proceso:


0 0 0
10 10 10
20 20 20
30 30 30
+
0 1 2
0 1 2
0 1 2
0 1 2
=
0 0 0
10 10 10
20 20 20
30 30 30
+
0 1 2
0 1 2
0 1 2
0 1 2
=
0 0 0
10 10 10
20 20 20
30 30 30
+
0 1 2
=
0 0 0
10 10 10
20 20 20
30 30 30
+
0 1 2
0 1 2
0 1 2
0 1 2
=
0 1 2
10 11 12
20 21 22
30 31 32
0
10
20
30
+
0 1 2
=
0 0 0
10 10 10
20 20 20
30 30 30
+
0 1 2
0 1 2
0 1 2
0 1 2
=