**Clase 3: Numpy**

Paquete para función científica en Python. Usa **arrays** (colecciones de ítems de datos que se almacenan en una sola variable) que pueden ser desde 1 dimensión hasta N-Dimensiones. Hace cálculos muy eficientes con arrays (**Vectorización**: En vez de ir término por término hace los cálculos en bloque). Hace operaciones de **algebra lineal, generación de números aleatorios**, entre otros. Los **elementos de un array**, a diferencia de una lista, deben ser todos del **mismo tipo**.

Ventajas:

* **Buena performance en los cálculos.**
* **Permite almacenar y manipular grandes volúmenes de datos.**
* **Se pueden nombrar y referenciar ítems individuales en una colección de datos.**
* **Y modificarlos masivamente con una sola sentencia.**

Vector: 1D; Tabla/Matriz: 2D; ND Array.

**Para importar información de un txt:**

data = np.genfromtxt(data\_location, skip\_header=1, delimiter=',')

**Construcción de un Array:** numpy.array(object)

IE: Se puede crear un array a partir de una lista: Nuevo\_array = np.array(lista)

**Creación de Arrays Vacíos:** np.empty(shape, dtype=float)

Siendo shape: (#filas,#columnas,#dimensión3, etc.)

Dtype: Por default es float, pero puede ser también int, bool o str).

**Creación de Arrays con Valores ceros:** np.zeros(shape, dtype=float)

**Creación de Arrays con Valores unos:** np.ones(shape, dtype=float)

**Creación de Arrays con Valores espaciados uniformemente:** np.arange(#) devuelve un Array con elementos enteros en orden creciente desde el #0 hasta #-1.

**Creación de Arrays con valores mínimos y máximos, de un tamaño determinado:** np.linspace(#min, #max, #cantidaddeelementos). data\_x = np.linspace(0,10,100) genera un array data\_x con 100 elementos entre 0 y 100.

**Métodos con Arrays:**

Nombre\_de\_array.dim: Devuelve el número de dimensiones de un Array.

Nombre\_de\_array.shape: Devuelve la forma de un array (#dim1, #dim2, #dim3,…,#dimn)

Nombre\_de\_array.size: Devuelve la cantidad de elementos que tiene un Array.

Nombre\_de\_array.dtype: Devuelve el tipo de elementos que tiene un Array.

Nuevo\_array = np.concatenate((Array1,Array2)): Sirve para armar un nuevo array a partir de dos arrays. Este nuevo array contiene los valores de Array1 y Array2.

**Conversión de un array en otro cambiando sus tipos de datos**:

new\_array = old\_array.astype(float)

Los arrays **se indexan** desde 0 hasta len(array)-1.

**Consulta de un valor de un Array:** Array[#] devuelve el valor de ese array en la posición #, siendo # un número entre 0 y len(array)-1.

**Shape** de arrays: (#filas,#columnas).

**Indexing de arrays con Slicing:** nombre\_array[#inicial\_filas:#final\_filas,step\_filas, #inicial\_columnas:#final\_columnas,step\_columnas]. Recordar que el #final no se incluye; si quiero tomar el último valor de una dimensión, no hay que definir valor final.

**Indexing de arrays con Fancy Indexing:** Secrea una lista de índices y se usa para acceder a determinados elementos del array:

nombre\_array[[#1,#2,#3,#4,…,#n]] Siendo cada #i la posición de elemento de array que queremos ver; podemos cambiar el orden en que se presenta también. IE: array[[5,9,3]] nos va a traer los elementos de las posiciones 6, 10 y 4 de un array de una dimensión. Si queremos que en vez de filas nos devuelva columnas, simplemente se cambia la sentencia a: nombre\_array[:**,**[#1,#2,#3,#4,…,#n]]. El output va a ser un array con las columnas que le vayamos indicando. Si aplicamos Fancy indexing indicando filas y columnas al mismo tiempo, nos va a devolver un array de una dimensión con los elementos que contengan la combinación de dichas filas y columnas.

**Indexing de arrays con Boolean Indexing:** Secrea una máscara booleana (array o lista de True False) que luego se aplica sobre un array y nos devuelve sólo los elementos del array donde la máscara booleana da True.

IE: nombre\_array[nombre\_array > 4] traerá sólo aquellos elementos del array cuyos valores sean mayores que 4.

Se pueden combinar máscaras booleanas con los operadores &, |, ˆ, etc.

Si aplico una máscara booleana sobre un array de más de una dimensión sin pedirle que mire sólo una fila o columna determinada, va a devolver un array de 1 dimensión con todos los valores donde se dé esa condición; pero no tengo noción de dónde están dichos valores en el array original.

**Generador de Números Aleatorios:** random\_generator\_seed = np.random.default\_rng(###).

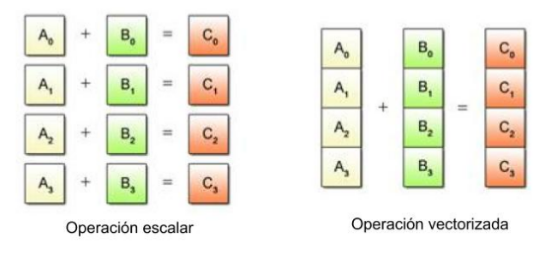
Si se define algún valor en ###, lo que hacemos es fijar una semilla que hace que el número aleatorio generado sea siempre el mismo.

**Generador de Distribución Uniforme:** Una vez generado el random\_generator\_seed, random\_generator\_seed.uniform(low, high, size); siendo low el valor mínimo que quiero tener en dicha distribución; High el valor máximo y size la forma del array que quiero como output. IE, con size (3,4) me devolverá una distribución uniforme en un array de 3 filas x 4 columnas.

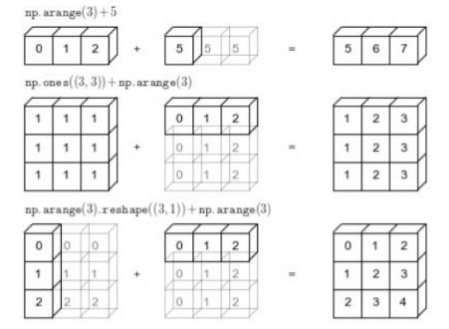
**Generador de Distribución Normal:** Una vez generado el random\_generator\_seed, random\_generator\_seed.normal(mean, sd, size); siendo mean el valor de la media; sd la desviación estándar y size la forma del array que quiero como output. IE, con size (3,4) me devolverá una distribución normal en un array de 3 filas x 4 columnas.

**Operaciónes Vectorizadas:**

Se trabaja con la información en bloque y es mucho más eficiente que hacer el cálculo elemento a elemento (operación escalar).



En caso de aplicar operaciones sobre arrays de distintos tamaños, Numpy proyecta los arrays para que coincidan en filas y columnas:



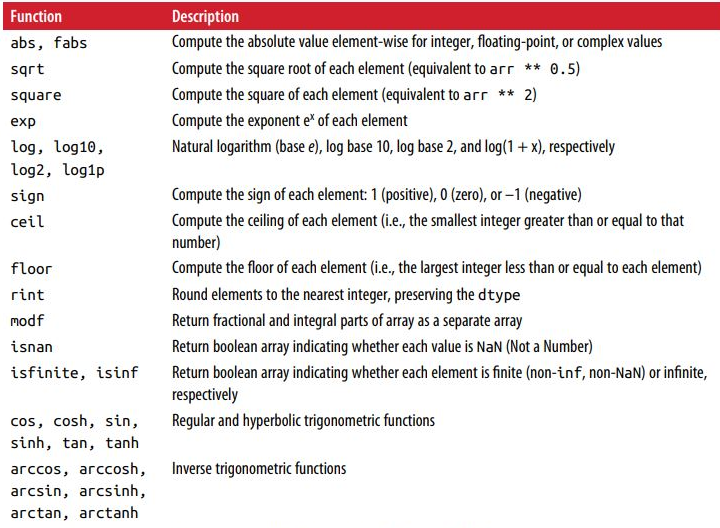
Pero hay 3 reglas a cumplir:

**Regla 1:** Si los dos arrays difieren en su número de dimensiones, la forma de la que tiene menos dimensiones se rellena con unos en su lado delantero (izquierdo).

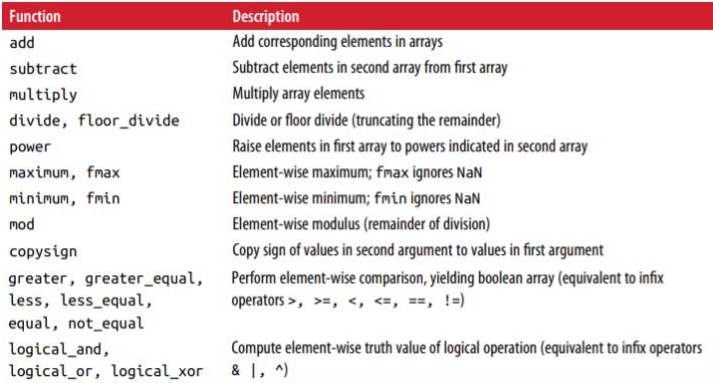
**Regla 2:** Si la forma de los dos arrays no coincide en alguna dimensión, el array con forma igual a 1 en esa dimensión se estira para que coincida con la otra.

**Regla 3:** Si en alguna dimensión los tamaños son diferentes y ninguno es igual a 1, se genera un error.

Listado de funciones:

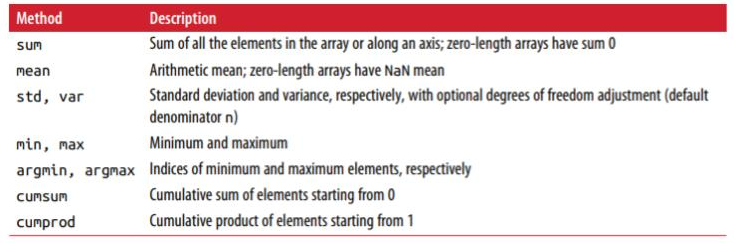






La gran ventaja que tienen estas funciones sobre las funciones escalares son los tiempos de ejecución de las mismas.

**Métodos Matemáticos y estadísticos**:



En caso de tener un array de 2 dimensiones (filas y columnas) y se quiere trabajar sobre un grupo de filas en una columna, o sobre un grupo de columnas en una fila, se usa **axis** para reducir el eje sobre el cual no queremos trabajar (axis = 0 reduce filas; axis = 1 reduce columnas).

minimos\_por\_columnas = nombre\_array.min(axis=0)

Devuelve un array con tantos elementos como columnas, con el valor mínimo de cada fila del array original.

máximos\_por\_filas = nombre\_array.min(axis=1)

Devuelve un array con tantos elementos como filas, con el valor máximo de cada fila del array original.

suma\_de\_cada\_fila= nombre\_array.sum(axis=1)

Devuelve un array con tantos elementos como filas, con la suma de los valores de cada fila del array original.