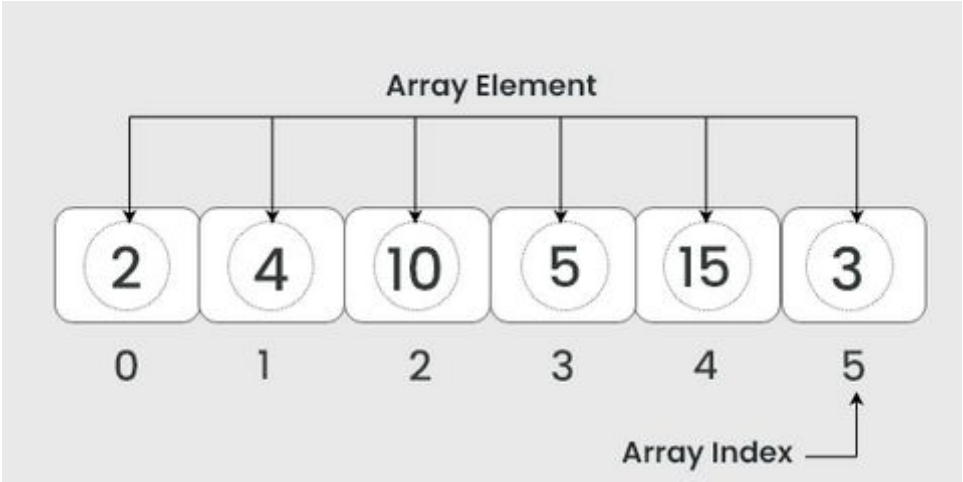
TEMA 1: NUMPY

NumPy es una librería que da soporte para crear vectores y matrices grandes multidimensionales, junto con una gran colección de funciones matemáticas de alto nivel para operar con ellas de forma eficiente.

Arrays. Un array es un conjunto de elementos del mismo tipo. Si definimos un array 6 enteros, podemos almacenar en él 6 números enteros. Cada posición del array está identificada por el índice, que indica su posición dentro del array.



Básicos NumPy

NumPy es una librería externa a Python, por lo que no forma parte de su librería estándar. Para utilizar las funciones de NumPy es necesario importar la librería.

Importamos una librería con la sentencia **import**. A partir de la importación, podemos usar las funciones dentro de la librería anteponiendo el nombre de la librería. Por ejemplo:

```
import numpy

mi_array = numpy.función(parámetros)
```

Es usual renombrar los nombres de la librería para hacerlos más cortos

```
import numpy as np

mi_array = np.función(parámetros)
```

Básicos NumPy

El objeto elemental de funcionamiento en Numpy es el **ndarray** (n dimensional array)

Un **ndarray** es un array multidimensional que contiene datos homogéneos, esto es, todos son del mismo tipo. Un ndarray es más rápido y flexible que estructuras de datos primitivas en Python (como listas) para grandes conjuntos de datos.

Atributos de **ndarray**:

- ndim: Número de dimensiones.
- size: Número de elementos que contiene.
- shape: Tupla de enteros que indica tamaño de cada dimensión.
- dtype: Almacena el tipo de cada elemento en el array.

```python
import numpy as np

# Creación del array a partir de una lista
lista = [1, 2, 3, 4]
mi_array = np.array(lista)

#Consulta de atributos
print("Dimensiones:", mi_array.ndim)
print("Tamaño:", mi_array.size)
print("Shape:", mi_array.shape)
print("Tipo:", mi_array.dtype)
```

```
Dimensiones: 1
Tamaño: 4
Shape: (4,)
Tipo: int32
```

Básicos NumPy

Funciones para crear array 1 dimensión:

- numpy.array(secuencia). Crea el array de la secuencia recibida por parámetro.
- numpy.ones(N). Crea array con todo 1 de tamaño N.
- numpy.zeros(N). Crea array con todo 0 de tamaño N.
- numpy.empty(N). Crea array sin inicializar de tamaño N.
- numpy.arange(N). Crea array con enteros del 0 a N-1

El tipo de los elementos del array los infiere automáticamente NumPy según los datos recibidos (int, float…)

```
import numpy as np

mi_array = np.array([1, 2, 3, 4])
print(mi_array)
print(mi_array.dtype)
```

```
[1 2 3 4]
int32
```

Básicos NumPy

Asignar valores: Para asignar/modificar un valor a una posición del array, especificamos el nombre del array y el índice de la posición donde queremos almacenar el dato.

¡Los índices comienzan por 0!

```
import numpy as np

mi_array = np.array([1, 2, 3, 4])
print(mi_array)

# Actualizamos elemento posición 2
mi_array[2] = 45
print(mi_array)
```

```
[1 2 3 4]
[ 1  2 45  4]
```

Básicos NumPy

Acceder a elementos almacenados: Para leer la posición de un array procedemos de la misma manera, indicando su índice.

```
import numpy as np

mi_array = np.array([1, 2, 3, 4])

# Mostramos el elemento posición 3
elemento = mi_array[3]
print(elemento)
```

```
4
```

Estadística básica

- **Variable estadística:** En un instrumento matemático que representa una característica o comportamiento observable, que puede cambiar con el tiempo y sobre el que deseamos investigar.

- **Media**: La media aritmética es un promedio estándar de una variable estadística. Se define sumando los valores de la variable y dividiéndolos entre el número de observaciones.

$$\bar{x} = \frac{1}{n} \sum_{i=1}^{n} x_i$$

Estadística básica

- **Mediana:** La mediana de una lista finita de números **ordenados de menor a mayor** corresponde a: si el conjunto de datos tiene un número impar de observaciones, se selecciona el número del medio. Ejemplo:

  1, 3, 3, **6**, 7, 8, 9 El valor mediano es el 6

  Si el conjunto de datos tiene un número par de observaciones, no hay un valor medio distinto y la mediana suele definirse como la media de los dos valores medios.

  Ejemplo: 1, 2, 3, **4, 5**, 6, 8, 9 mediana = *4.5*, es decir (4 + 5) / 2

- **Moda:** es el valor que aparece con mayor frecuencia en un conjunto de datos.

- **Desviación estándar (o típica):** es una medida que se utiliza para cuantificar la variación o la dispersión de un conjunto de datos numéricos. Una desviación estándar baja indica que la mayor parte de los datos de una muestra tienden a estar agrupados cerca de su media (también denominada el *valor esperado*), mientras que una desviación estándar alta indica que los datos se extienden sobre un rango de valores más amplio.

$$\sigma = \sqrt{\frac{\sum_i^N (X_i - \bar{X})^2}{N}}$$

Otras funciones crear arrays en NumPy

Funciones para crear array 1 dimensión:

- numpy.ones_like(mi_array). Crea array con todo 1 de igual tamaño y tipo que mi_array.
- numpy.zeros_like(mi_array). Igual que anterior con todo a 0
- numpy.empty_like(mi_array). Igual que anterior sin valores asignados.
- numpy.full(N, valor). Crea array de tamaño N con todo iniciado a “valor”
- numpy.full_like(array, valor).
- numpy.linspace(N, M, X). Crea un array de X valores numéricos desde el valor N hasta el valor M igualmente separados unos de otros.

```
import numpy as np
x=np.linspace(3,7,6)
print(x)
```

```
[3.  3.8 4.6 5.4 6.2 7. ]
```

Aritmética en NumPy

NumPy tiene implementadas operaciones aritméticas entre arrays o entre un array un un escalar de forma mucho más eficiente que las listas de Python y sin utilizar for para realizar estas operaciones.

Operaciones entre elementos de dos arrays, elemento a elemento y devuelve 1 array. Por ejemplo, sumar los elementos del array_1 con los elementos del array_2

```
import numpy as np

x1=np.array([0,1,2,3,4])
x1=np.ones_like(x1)
print(x1)
print(x2)
print(x1 + x2)
```

```
[1 1 1 1 1]
[1 2 3 4 5]
[2 3 4 5 6]
```

Aritmética en NumPy

Operaciones entre array y un escalar: Devuelve un array aplicando la operación aritmética entre cada elemento del array y el escalar de forma individual.

```
x1=np.array([0,1,2,3,4])
print(x1)
print(x1 * 2)
```

```
[0 1 2 3 4]
[0 2 4 6 8]
```

Comparaciones en NumPy

Comparaciones entre arrays. Operadores de comparación que comparan elemento a elemento entre 2 arrays y devuelven un array de True/False, siendo cada uno el resultado de la comparación de los elementos del mismo índice.

También podemos comparar cada elemento de un array con un escalar.

```
x1 = np.array([0, 1, 2, 3, 4])
x2 = np.array([3, 0, 5, 2, 1])
print(x1)
print(x2)
print(x1 > x2)
```

```
[0 1 2 3 4]
[3 0 5 2 1]
[False  True False  True  True]
```

Funciones estadísticas de NumPy de agrupamiento: Retornan 1 valor

Media	numpy.mean(array)
Mediana	numpy.median(array)
Máximo	numpy.max(array)
Mínimo	numpy.min(array)
Desviación estándar	numpy.std(array)
Varianza	numpy.var(array)
Suma de valores del array	numpy.sum(array)
Índice del valor mínimo	numpy.argmin(array)
Índice del valor máximo	numpy.argmax(array)

Otra funciones NumPy

Ordenar valores de menor a mayor. Devuelve una **copia** del array ordenado.	numpy.sort(array)
Obtener ordenados sin repetidos	numpy.unique(array)

```
x = np.array([4, 2, 5, 2, 3, 4])
print("Array ordenado", np.sort(x))
print("Array ordenado sin repetidos", np.unique(x))
print("Array original", x)
```

```
Array ordenado [2 2 3 4 4 5]
Array ordenado sin repetidos [2 3 4 5]
Array original [4 2 5 2 3 4]
```

Otra funciones NumPy

Hasta ahora hemos usados las funciones de NumPy a través del módulo importado o renombrado con “np”. Algunas funciones que afectan directamente a un array, como la operación sort, pueden ser invocadas directamente a través del objeto, como si fuese un método propio al igual que en POO.

En este caso, sort no crea una copia.

```python
import numpy as np

# Creación del array a partir de una lista
lista = [4, 3, 2, 1]
mi_array = np.array(lista)

# Ordenar array sin crear copia
mi_array.sort()
print(mi_array)

# Otras funciones
print("Media:", mi_array.mean())
print("Valor máximo:", mi_array.max())
print("Valor mínimo:", mi_array.min())
print("Desviación estándar:", mi_array.std())
print("Varianza:", mi_array.var())
print("Suma de valores:", mi_array.sum())
print("Índice del valor mínimo:", mi_array.argmin())
print("Índice del valor máximo:", mi_array.argmax())
```

```
[1 2 3 4]
Media: 2.5
Valor máximo: 4
Valor mínimo: 1
Desviación estándar: 1.118033988749895
Varianza: 1.25
Suma de valores: 10
Índice del valor mínimo: 0
Índice del valor máximo: 3
```

Indexación

Indexación: Acceder o seleccionar elementos del array.

La indexación más simple es acceder a un solo elemento a través de su índice:

ndarray[i] = valor

Pero existen diversas formas de indexar un subconjunto de varios elementos del ndarray a la vez, obteniendo un ndarray:

- Slicing
- Boolean indexing
- Fancy indexing

Slicing en Listas

Las "slices" de listas existen de forma primitiva en Python. Crea una **nueva lista** con la porción.

Una slice o porción de una lista se obtiene así -> [inicio : parada : incremento]

El valor de parada no se incluye. Los valores opcionales, si no se especifican, se usan los valores por defecto:

- inicio=0
- parada=final de la lista, incluyendo el último
- incremento=1

Podemos obtener listas inversas con incremento -1.

```
lista = [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
porcion = lista[5:]
print(porcion)
```

```
[5, 6, 7, 8, 9]
```

```
print(lista[2::2])
```

```
[2, 4, 6, 8]
```

```
print(lista[7:1:-1])
```

```
[7, 6, 5, 4, 3, 2]
```

Slicing en NumPy

En **NumPy**, los slice son "**vistas**" del array original y **no copia de datos** a nueva variable, cualquier modificación al slice afectará a los datos originales.

```
# Creamos un array con números del 0 al 9 (10-1)
mi_array = np.arange(10)
print("mi_array original: ", mi_array)
```

```
mi_array original:  [0 1 2 3 4 5 6 7 8 9]
```

```
porcion = mi_array[1:4]
print("porcion", porcion)
# Modifico en array porcion su posición 0
porcion[0] = 99
print("porcion modificada", porcion)
```

```
porcion [1 2 3]
porcion modificada [99  2  3]
```

```
print("mi_array original: ", mi_array)
```

```
mi_array original:  [ 0 99  2  3  4  5  6  7  8  9]
```

Slicing en NumPy

Podemos forzar una COPIA del slice a nueva zona de memoria con el método ndarray.copy()

copia_array = ndarray.copy() Devuelve una copia en memoria del ndarray.

```
# Creamos un array con números del 0 al 9 (10-1)
mi_array = np.arange(10)

# Slice copiando a ndarray diferente
copia_porcion = mi_array[1:4].copy()
copia_porcion[0] = 99
print("copia porcion", porcion)
print("array original", mi_array)
```

```
copia porcion [99  2  3]
array original [0 1 2 3 4 5 6 7 8 9]
```

Slicing en NumPy

El slice sirve para asignar 1 valor de forma unitaria.

Por ejemplo: Poner a 0 las posiciones de la 2 a la 4.

```
# Creamos un array con números del 0 al 9 (10-1)
mi_array = np.arange(10)
print("array   original", mi_array)

# Asignación a 0 de las posiciones 2 a 4
mi_array[2:5] = 0
print("array modificado", mi_array)
```

```
array   original [0 1 2 3 4 5 6 7 8 9]
array modificado [0 1 0 0 0 5 6 7 8 9]
```

O para asignar a una porción otro ndarray o lista. Por ejemplo: Poner las posiciones de la 2 a la 4 los valores de otro array.

```
# Creamos un array con números del 0 al 9 (10-1)
mi_array = np.arange(10)
print("array   original", mi_array)

#Segundo array de 3 posiciones
ar2 = np.array([9, 99, 999])

# Asignación de un array a a la porcion de otro
mi_array[2:5] = ar2
print("array modificado", mi_array)
```

```
array   original [0 1 2 3 4 5 6 7 8 9]
array modificado [  0   1   9  99 999   5   6   7   8   9]
```

Indexación booleana

En NumPy podemos pasarle a un ndarray otro ndarray (o lista) con valores booleanos. La indexación del array original nos devolverá aquellos que coincidan en su índice con un valor a True.

```
# Array original de enteros de 0 a 4
mi_array = np.arange(5)

# Array de 5 valores booleanos
array_bool = np.array([True, False, True, False, True])

# Indexación booleana, me devuelve los valores de mi_array que coinciden en posición con True
array_indexado_bool = mi_array[array_bool]
print("array original", mi_array)
print("array booleano", array_bool)
print("array indexado", array_indexado_bool)
```

```
array original [0 1 2 3 4]
array booleano [ True False  True False  True]
array indexado [0 2 4]
```

Indexación booleana

Recuerda que en NumPy podemos comparar cada elemento de ndarray con otro ndarray o con 1 escalar y nos genera un ndarray booleano (usar operadores lógicos & |).

Podemos usar este ndarray booleano para indexar datos que cumplan la condición. Devuelve una copia de datos.

```
# Tenemos en un ndarray los nombres de 5 trabajadores
nombres = np.array(["Juan", "Inés", "Pablo", "Antonio", "María"])

# Tenemos en otro ndarray los sueldos correspondientes a los trabajadores
sueldos = np.array([2500, 1500, 3000, 1200, 1800])

# Vamos a seleccionar los trabajadores con sueldo mayor a 1500
# Generamos array booleano a través del ndarray sueldos
bool_mayores = sueldos > 1500
print(bool_mayores)
```

```
[ True False  True False  True]
```

```
# Indexación booleana
# Trabajadores con sueldo > a 1500
resultado = nombres[bool_mayores]
print(resultado)
print(type(resultado))
```

```
['Juan' 'Pablo' 'María']
<class 'numpy.ndarray'>
```

```
# Mismo resultado pero en 1 sola línea
nombres = np.array(["Juan", "Inés", "Pablo", "Antonio", "María"])
sueldos = np.array([2500, 1500, 3000, 1200, 1800])
print(nombres[sueldos > 1500])
```

```
['Juan' 'Pablo' 'María']
```

Fancy indexing

Indexar un array mediante una lista o ndarray de enteros. Esta lista de enteros indica los índices de los elementos que deseamos, en cualquier orden.

Crea una copia de datos, a diferencia del slicing.

```
# Tenemos en un ndarray los nombres de 6 personas
nombres = np.array(["Juan", "Inés", "Pablo", "Antonio", "María", "Pedro"])

# Indexación por índices, indicamos que índices queremos mediante una lista de índices
nombres_copia = nombres[ [5, 3, 4] ]
print(nombres_copia)
```

```
['Pedro' 'Antonio' 'María']
```

```
# Devuelve una copia, no una vista
nombres_copia[0] = "-------"
print(nombres_copia)
print(nombres)
```

```
['-------' 'Antonio' 'María']
['Juan' 'Inés' 'Pablo' 'Antonio' 'María' 'Pedro']
```

ufunc de NumPy

Funciones universales, o ufunc, es una función que realiza una operación elemento a elemento de los datos en un ndarray. Existen operaciones unarias aplicadas entre los elementos de un array y operaciones binarias a aplicar entre un array y un escalar o entre dos arrays. Estas funciones devuelven un array. Esto es lo que se llama en NumPy funciones vectorizadas.

ufunc de NumPy

ufunc unarias: Reciben por parámetro 1 ndarray y devuelven un ndarray

- Valor absoluto: numpy.abs(ndarray)
- Raíz cuadrada: numpy.sqrt(ndarray)
- Cuadrado: numpy.square(ndarray)
- Redondear al entero más cercano: numpy.rint(ndarray)
- Signo: Devuelve un array con -1 si el signo es negativo, 1 si es positivo, o 0 si es 0

  numpy.sign(ndarray)

- modf: Separa la parte entera y parte decimal de un array

  decimal, entera = numpy.modf(ndarray)

ufunc de NumPy

ufunc binarias: Los operadores aritméticos (+, -, *, /) implementan las ufuncs binarias. También los operadores de comparación. Reciben por parámetro 2 ndarrays (o escalares) y devuelven 1 ndarray.

- Suma (es equivalente al operador '+': numpy.add(ar1, ar2)
- Valor máximo entre los elementos de dos arrays (fmax ignora nan): numpy.maximum(ar1, ar2))
- # Valor mínimo entre los elementos de dos arrays (fmin ignora nan): numpy.minimum(ar1, ar2))

```
# suma
ar1 = np.array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9])
ar2 = np.ones(10)
print(ar1)
print(ar2)


# suma
print("\nSuma", np.add(ar1, ar2))
print("+    ", ar1 + ar2)

# maximum, fmax ignora nan
print("\n maximum ", np.maximum(ar1, ar2))

# maximum, fmin ignora nan
print("\n minimum ", np.minimum(ar1, ar2))
```

```
[0 1 2 3 4 5 6 7 8 9]
[1. 1. 1. 1. 1. 1. 1. 1. 1. 1.]

Suma [ 1.  2.  3.  4.  5.  6.  7.  8.  9. 10.]
+    [ 1.  2.  3.  4.  5.  6.  7.  8.  9. 10.]

 maximum  [1. 1. 2. 3. 4. 5. 6. 7. 8. 9.]

 minimum  [0. 1. 1. 1. 1. 1. 1. 1. 1. 1.]
```

nan

numpy.nan: Constante de NumPy que equivale a null o valor desconocido.

Métodos numpy.isnan(ndarray) devuelve 1 array booleano indicando si cada elemento del ndarray es nan o no.

```
# Constante de Numpy que representa el valor desconocido o null (not a number)
print(np.nan)

# Comprobar si un valor es nan
ar3 = np.array([1, 2, 3, np.nan, 4, 5])
print(ar3)
print(np.isnan(ar3[3]))
print(np.isnan(ar3))
```

```
nan
[ 1.  2.  3. nan  4.  5.]
True
[False False False  True False False]
```

Métodos comprobar valores booleanos

ndarray.any() -> Devuelve True si el array contiene algún valor a True.

ndarray.all() -> Devuelve True si todos los valores del array son True.

```
# Comprobar valores booleanos

ar4 = np.array([False, False, True, False])

print("Algun True?", ar4.any())
print("Todos True?", ar4.all())
```

```
Algun True? True
Todos True? False
```

Operaciones con conjuntos

Operaciones con conjuntos típicas matemáticas: Intersección, unión, diferencia. Más filtrado de únicos (numpy.unique()) y método in para comprobar pertenencia (numpy.in1d()).

```
ar1 = np.array([0, 1, 3, 5, 5, 2, 1,  7, 10, 12])
ar2 = np.array([0, 2, 4, 6, 1, 2, 0,  8, 10, 12])

# unicos
print("únicos", np.unique(ar1))

# Intersección
print("interseccion", np.intersect1d(ar1, ar2))

# Union
print("union", np.union1d(ar1, ar2))

# in
print("in", np.in1d(ar1, ar2))

# diferencia
print("diferencia", np.setdiff1d(ar1, ar2))
```

```
únicos [ 0  1  2  3  5  7 10 12]
interseccion [ 0  1  2 10 12]
union [ 0  1  2  3  4  5  6  7  8 10 12]
in [ True  True False False False  True  True False  True  True]
diferencia [3 5 7]
```

Función where

La función numpy.where() es una función vectorizada equivalente a una expresión if-else.

Sintáxis: numpy.where(array_condición, valores_Si_es_True, valores_Si_es_False)

Los valores pueden ser arrays o escalares.

```
# Arrays de datos iniciales
impares_arr = np.array([1, 3 , 5, 7, 9])
pares_arr = np.array([2, 4, 6, 8, 10])

# Array booleano
cond_arr = np.array([True, False, True, False, True])

# Where.
result = np.where(cond_arr, impares_arr, pares_arr)
print(result)

# También pueden ser escalares los arrays de datos
result = np.where(cond_arr, impares_arr, 0)
print(result)
```

```
[1 4 5 8 9]
[1 0 5 0 9]
```