# Módulo 6.1: Numpy

Prof. Carlos Cedeño

## Objetivos de la Clase:

- Comprender qué es NumPy y por qué es fundamental para la computación científica en Python.
- Crear y manipular arreglos ndarray de manera eficiente.
- Identificar y trabajar con los tipos de datos de NumPy.
- Realizar operaciones matemáticas y lógicas con arreglos.
- Utilizar las funciones más comunes de NumPy.
- Generar datos aleatorios de manera eficiente con numpy.random.

# 1. ¿Qué es NumPy y por qué usarlo?

- ¿Qué es NumPy?
  - o La biblioteca fundamental para la computación numérica en Python.
  - Proporciona un objeto de arreglo N-dimensional de alto rendimiento (ndarray).
  - Es la base para otras bibliotecas clave como SciPy, Pandas y scikit-learn.

## • ¿Por qué usar NumPy en lugar de listas de Python?

- Velocidad: Las operaciones con arreglos de NumPy son mucho más rápidas porque están implementadas en C y operan a nivel de bloque de memoria.
- Eficiencia de memoria: Almacenan los datos de manera contigua, lo que es más eficiente.
- Funcionalidad: Ofrece una amplia gama de funciones matemáticas y estadísticas optimizadas que no existen en las listas.
- Sintaxis: El código para operaciones numéricas es más limpio, legible y "vectorizado".

#### • Instalación:

pip install numpy

## • Importación:

import numpy as np

(Explicar por qué la convención es usar np ).

# 2. Creación y Tipos de Datos de Arreglos

# 2.1. Creación de Arreglos

• A partir de listas: La forma más común.

```
lista = [1, 2, 3, 4, 5]
arreglo = np.array(lista)
print(arreglo)
```

# • Arreglos 2D (Matrices):

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

#### • Funciones para crear arreglos comunes:

- o np.zeros(shape): Crea un arreglo lleno de ceros.
- onp.ones(shape): Crea un arreglo lleno de unos.
- onp.arange(start, stop, step): Similar a range(), pero devuelve un arreglo.
- onp.linspace(start, stop, num): Crea un arreglo con números espaciados uniformemente.

```
print(np.zeros(5))
print(np.ones((2, 3)))
print(np.arange(0, 10, 2))
print(np.linspace(0, 1, 5))
```

## 2.2. Atributos de los Arreglos

- arreglo.ndim: Número de dimensiones.
- arreglo.size : Número total de elementos.
- arreglo.dtype: Tipo de dato de los elementos.

## 2.3. Tipos de Datos (dtype)

• dtype comunes:

```
int8, int16, int32, int64float32, float64bool
```

• Especificar el tipo de dato al crear:

```
arreglo_float = np.array([1, 2, 3], dtype=np.float64)
print(arreglo_float)
print(arreglo_float.dtype)
```

• Cambiar el tipo de dato:

```
arreglo_int = arreglo_float.astype(np.int32)
print(arreglo_int)
print(arreglo_int.dtype)
```

# 3. Operadores y Funciones Principales

### 3.1. Operaciones Element-wise (Elemento por Elemento)

- Los operadores estándar ( + , , \* , / ) se aplican a cada elemento del arreglo.
- Operaciones con un escalar:

```
arr = np.array([1, 2, 3, 4])
print(arr + 5)
print(arr * 2)
```

• Operaciones entre dos arreglos (de la misma forma):

```
arr2 = np.array([10, 20, 30, 40])
print(arr + arr2)
print(arr * arr2)
```

• ¡Advertencia! Esto NO es una multiplicación de matrices. Es una multiplicación elemento a elemento. Para la multiplicación de matrices, se usa np.dot() o el operador @.

## 3.2. Operaciones Lógicas y de Comparación

• Las comparaciones devuelven arreglos de booleanos.

```
arr = np.array([1, 5, 2, 8])
print(arr > 3)
```

• Indexación Booleana: Usar arreglos de booleanos para filtrar elementos.

```
arr_filtrado = arr[arr > 3]
print(arr_filtrado)
```

#### 3.3. Funciones Matemáticas Universales (ufuncs)

• Funciones que aplican una operación matemática a cada elemento del arreglo.

```
• Ejemplos: np.sqrt(), np.log(), np.sin(), np.exp().

arr = np.array([1, 4, 9, 16])
    print(np.sqrt(arr))
```

## 3.4. Funciones de Agregación

- Funciones para resumir un arreglo.
- np.sum(): Suma todos los elementos.
- np.mean(): Calcula el promedio.
- np.max(): Encuentra el valor máximo.
- np.min(): Encuentra el valor mínimo.

```
matriz = np.array([[1, 2, 3], [4, 5, 6]])
print(np.sum(matriz))
print(np.sum(matriz, axis=0)) # Suma por columnas
print(np.sum(matriz, axis=1)) # Suma por filas
```

# 4. Generación de Números Aleatorios (numpy.random)

• np.random.rand(d0, d1, ...): Genera números aleatorios de una distribución uniforme entre 0 y 1.

```
print(np.random.rand(3))
print(np.random.rand(2, 2))
```

• np.random.randn(d0, d1, ...): Genera números aleatorios de una distribución normal (gaussiana) estándar (media 0, desviación estándar 1).

```
print(np.random.randn(4))
```

• np.random.randint(low, high, size): Genera enteros aleatorios dentro de un rango específico.

```
print(np.random.randint(1, 10, 5))
print(np.random.randint(1, 10, (2, 3)))
```

• np.random.choice(a, size, replace, p): Elige elementos aleatorios de un arreglo.

```
opciones = ['A', 'B', 'C', 'D']
print(np.random.choice(opciones, size=3))
```