Introducción a NumPy

J.M. Marín Ramírez

Métodos computacionales

Contenido

¿Qué es una librería y cómo funciona?

- Una **librería** es un conjunto de funciones y herramientas predefinidas que permiten realizar tareas específicas sin tener que escribir el código desde cero.
- En Python, las librerías se agrupan en módulos que pueden ser importados usando `import`.
- Las librerías ayudan a mejorar la **eficiencia** y **reusabilidad** del código.
- **Funcionamiento:**
  - Las librerías se instalan y se importan en el código.
  - Los desarrolladores pueden usar las funciones y clases que proporciona la librería.
  - Esto evita la necesidad de implementar cada funcionalidad desde cero.
- Ejemplos: NumPy, pandas, Matplotlib, entre otras.

Cómo llamar métodos o funciones en una librería

- Para utilizar una librería en Python, primero debes importarla con `import`.
- Luego, puedes acceder a las funciones o métodos que ofrece usando la sintaxis:

  `nombre_de_librería.función()`

- Ejemplo con NumPy:
  - Importar la librería:

```
import numpy as np
```

  - Llamar a una función como `np.array()`:

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

- También puedes usar un alias para simplificar el nombre de la librería, como en `numpy` que suele abreviarse como `np`.

¿Qué es NumPy?

- NumPy es una librería de Python utilizada para el cálculo numérico.
- Proporciona soporte para arrays y matrices multidimensionales.
- NumPy es la base para muchas otras librerías científicas en Python, como SciPy y pandas.
- Para más información: Documentación oficial de NumPy.

Importancia de NumPy

- Es extremadamente eficiente para trabajar con grandes conjuntos de datos numéricos.
- Ofrece operaciones vectorizadas, lo que acelera el cálculo y reduce la necesidad de bucles explícitos.
- Compatible con funciones matemáticas avanzadas, como álgebra lineal, transformadas de Fourier y generación de números aleatorios.
- NumPy es fundamental en áreas como análisis de datos, machine learning y simulaciones científicas.

Creación de Arrays con NumPy

Ejemplos:

```
import numpy as np

# Crear un array a partir de una lista
array1 = np.array([1, 2, 3, 4, 5])

# Crear un array lleno de ceros
array_zeros = np.zeros(5)

# Crear un array lleno de unos
array_ones = np.ones(5)

# Crear un array con un rango de valores
array_range = np.arange(0, 10, 2)

# Crear un array con números equidistantes
array_linspace = np.linspace(0, 1, 5)  # [0. , 0.25, 0.5, 0.75, 1.]
```

Operaciones básicas con Arrays 1D

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

# Operaciones matemáticas
suma = array + 2          # [3, 4, 5, 6]
resta = array - 1         # [0, 1, 2, 3]
multiplicacion = array * 3  # [3, 6, 9, 12]
division = array / 2      # [0.5, 1, 1.5, 2]

# Operaciones estadísticas
suma_total = np.sum(array)        # 10
promedio = np.mean(array)         # 2.5
max_valor = np.max(array)         # 4
min_valor = np.min(array)         # 1
```

¿Qué es un `ndarray` y cómo acceder a sus elementos?

Definición

`ndarray` es el tipo de array principal en NumPy y significa **N-dimensional array**. Es una estructura de datos que permite almacenar y operar con matrices de cualquier número de dimensiones (1D, 2D, 3D, etc.).

- Puede ser un array de una o más dimensiones.
- Soporta operaciones matemáticas eficientes.
- Todos los elementos deben ser del mismo tipo (tipado homogéneo).

Acceso a los elementos:

```
# Crear un ndarray 2D (matriz)
array_2d = np.array([[1, 2, 3], [4, 5, 6]])
# Acceder al elemento en la fila 0, columna 1
elemento = array_2d[0, 1]  # Resultado: 2
# Acceder a una fila completa
fila = array_2d[1, :]  # Resultado: [4, 5, 6]
# Acceder a una columna completa
columna = array_2d[:, 2]  # Resultado: [3, 6]
```

Comparación: Lista de Python vs NumPy Array (Parte 1)

- **Estructura:**
  - **Listas**: Pueden almacenar diferentes tipos de datos (enteros, flotantes, cadenas).
  - **Arrays de NumPy**: Almacenan solo un tipo de dato (homogéneos).
- **Operaciones:**
  - **Listas**: Las operaciones aritméticas no están vectorizadas. Necesitas bucles para operaciones sobre todos los elementos.
  - **Arrays de NumPy**: Soportan operaciones vectorizadas, como suma, resta o multiplicación, aplicadas a todos los elementos sin necesidad de bucles.

Comparación: Lista de Python vs NumPy Array (Parte 2)

- **Velocidad:**
  - **Listas**: Son más lentas cuando se manejan grandes volúmenes de datos debido a la naturaleza dinámica y heterogénea de las listas.
  - **Arrays de NumPy**: Son más rápidos debido a su implementación en C y la optimización de operaciones matemáticas.
- **Memoria:**
  - **Listas**: Ocupan más memoria porque almacenan tipos de datos diferentes y sus referencias.
  - **Arrays de NumPy**: Son más eficientes en el uso de memoria, ya que todos los elementos son del mismo tipo.