SESIÓN LA LIBRERÍA NUMPY

CONTENIDOS:

- Reseña de la librería NumPy.
- Creación de arreglos de Numpy: Vectores, Matrices.
- Funciones preconstruidas de creación:
    - arreglo con valores (arange).
    - Matrices de ceros y unos.
    - Vector con distribución de puntos.
    - Matriz identidad.
    - Matriz aleatoria.
    - Redimensionado de un arreglo.
    - Indexación y selección: selección de elementos de un arreglo.
    - Selección condicional de elementos de un arreglo.
    - Referencia y copia de arreglos.
- Operaciones: operaciones entre arreglos, Operaciones con escalares.
- Aplicando funciones a un arreglo

RESEÑA DE LA LIBRERÍA NUMPY

NumPy (Numerical Python) es una librería fundamental para la computación numérica en Python. Proporciona una estructura de datos llamada array (arreglo), que permite trabajar con vectores y matrices de manera eficiente, optimizando el uso de memoria y procesamiento. NumPy es la base de muchas otras librerías de Ciencia de Datos como Pandas, SciPy, Matplotlib y TensorFlow, lo que la convierte en una herramienta esencial para cualquier persona que trabaje en análisis de datos, machine learning o computación científica.

¿Por qué usar NumPy?

1. **Más rápido que listas de Python:**
   - NumPy está implementado en C, lo que lo hace significativamente más rápido que las listas nativas de Python para operaciones numéricas.
   - Los arreglos de NumPy almacenan datos de manera contigua en memoria, lo que reduce la sobrecarga y mejora el rendimiento.
2. **Operaciones vectorizadas:**
   - NumPy permite realizar operaciones en arreglos completos sin necesidad de bucles explícitos (vectorización). Esto no solo simplifica el código, sino que también lo hace más eficiente.
   - Ejemplo: Sumar dos arreglos elemento por elemento con una sola línea de código.
3. **Funciones matemáticas optimizadas:**
   - NumPy incluye una amplia gama de funciones matemáticas predefinidas, como operaciones trigonométricas, logarítmicas, estadísticas y de álgebra lineal.
   - Estas funciones están altamente optimizadas para trabajar con arreglos multidimensionales.
4. **Facilidad para manejar grandes volúmenes de datos:**
   - NumPy es ideal para trabajar con grandes conjuntos de datos, ya que su estructura de arreglos es más eficiente en términos de memoria y procesamiento que las listas de Python.
   - Permite manipular datos en forma de matrices multidimensionales, lo que es especialmente útil en aplicaciones de machine learning y procesamiento de imágenes.
5. **Interoperabilidad con otras librerías:**
   - NumPy es compatible con una gran cantidad de librerías de Python, lo que facilita su integración en flujos de trabajo de ciencia de datos y machine learning.
   - Por ejemplo, Pandas utiliza arreglos de NumPy para almacenar y manipular datos en sus estructuras de DataFrames.
6. **Indexación y slicing avanzado:**
   - NumPy permite realizar operaciones de indexación y slicing de manera más poderosa y flexible que las listas de Python.
   - Esto es especialmente útil para extraer subconjuntos de datos o realizar operaciones en secciones específicas de un arreglo.

- Soporte para arreglos multidimensionales: NumPy permite trabajar con arreglos de una, dos o más dimensiones, lo que lo hace ideal para aplicaciones como procesamiento de imágenes (arreglos 2D) o datos científicos (arreglos 3D o superiores).

Ilustración 1 Logo NumPy

Ejemplos de uso de NumPy:

1) Creación de arreglos:

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

Ilustración 2 Ejemplo creación de arreglos

2) Operaciones matemáticas

```
a = np.array([1, 2, 3])
b = np.array([4, 5, 6])
suma = a + b  # Resultado: [5, 7, 9]
```

Ilustración 3 Ejemplo operaciones matemáticas

3) Funciones estadísticas:

```
datos = np.array([10, 20, 30, 40, 50])
media = np.mean(datos)  # Resultado: 30.0
```

Ilustración 4 Ejemplo funciones estadísticas

4) Álgebra lineal:

```
matriz = np.array([[1, 2], [3, 4]])
determinante = np.linalg.det(matriz)  # Resultado: -2.0
```

Ilustración 5 Ejemplo álgebra lineal

NumPy es una herramienta indispensable para cualquier persona que trabaje en computación científica, análisis de datos o Machine Learning. Su capacidad para manejar operaciones numéricas de manera eficiente, junto con su facilidad de uso y su integración con otras librerías, lo convierten en la opción preferida para trabajar con datos numéricos en Python. Si bien las listas de Python son útiles para tareas generales, NumPy ofrece un rendimiento y funcionalidades superiores cuando se trata de cálculos numéricos intensivos. Por estas razones, aprender y dominar NumPy es un paso crucial para cualquier profesional o entusiasta de la ciencia de datos.

CREACIÓN DE ARREGLOS DE NUMPY: VECTORES, MATRICES.

NumPy permite crear diferentes tipos de arreglos, desde vectores simples hasta matrices y arreglos multidimensionales. Estos arreglos permiten almacenar y manipular datos numéricos de manera eficiente, con una sintaxis clara y optimizada para cálculos matemáticos y científicos.

Vectores

Los vectores son arreglos de una sola dimensión, similares a una lista en Python, pero con la ventaja de ser más eficientes en términos de almacenamiento y velocidad de procesamiento.

Ejemplo de creación de un vector en NumPy:

```
import numpy as np

# Crear un vector con valores específicos
vector = np.array([1, 2, 3, 4, 5])
print("Vector:\n", vector)
```

Ilustración 6 Ejemplo de creación de un vector en NumPy

Características de los vectores en NumPy:

- Se crean con np.array(), pasándole una lista de valores.
- Son la base de operaciones matemáticas optimizadas en NumPy.
- Soportan operaciones matemáticas directas, como suma y multiplicación por escalares.

Ejemplo de operaciones con vectores:

```
vector2 = vector * 2  # Multiplicar cada elemento por 2
print("Vector multiplicado por 2:\n", vector2)
```

Ilustración 7 Ejemplo de operaciones con vectores

Matrices (2D)

Las matrices son arreglos de dos dimensiones (filas y columnas), similares a una tabla o una hoja de cálculo. Se utilizan en múltiples áreas, como álgebra lineal, procesamiento de imágenes y análisis de datos.

Ejemplo de creación de una matriz en NumPy:

```
# Crear una matriz 3x3
matriz = np.array([[1, 2, 3], [4, 5, 6], [7, 8, 9]])
print("Matriz:\n", matriz)
```

Ilustración 8 Ejemplo de creación de una matriz en NumPy

Características de las matrices en NumPy:

- Se crean con np.array() pasando una lista de listas.
- Cada lista interna representa una fila de la matriz.
- Se pueden manipular con operaciones como suma, multiplicación y transposición.

Ejemplo de operaciones con matrices:

```
# Transpuesta de la matriz
print("Matriz Transpuesta:\n", matriz.T)

# Suma de matrices
matriz2 = np.array([[1, 1, 1], [1, 1, 1], [1, 1, 1]])
print("Suma de Matrices:\n", matriz + matriz2)
```

Ilustración 9 Ejemplo de operaciones con matrices

Arreglos Multidimensionales (Más de 2 Dimensiones)

NumPy permite crear arreglos con más de dos dimensiones, ideales para aplicaciones avanzadas como procesamiento de imágenes, redes neuronales y datos científicos complejos.

Ejemplo de creación de un arreglo 3D (tensor) en NumPy:

```
# Crear un arreglo tridimensional (3D)
tensor = np.array([[[1, 2], [3, 4]], [[5, 6], [7, 8]]])
print("Arreglo 3D:\n", tensor)
```

Ilustración 10 Ejemplo de creación de un arreglo 3D (tensor) en NumPy

Características de los arreglos multidimensionales en NumPy:

- Se crean con np.array() anidando más niveles de listas.
- Se pueden visualizar como una pila de matrices.
- Útiles en análisis de datos en múltiples dimensiones.

Ejemplo de acceso a un elemento en un arreglo 3D:

```
print("Elemento en la posición (1,0,1):", tensor[1, 0, 1])
```

Ilustración 11 Ejemplo de acceso a un elemento en un arreglo 3D

NumPy es una herramienta fundamental para manipular arreglos de manera eficiente en Python, permitiendo realizar operaciones matemáticas optimizadas y facilitando el manejo de grandes volúmenes de datos.

FUNCIONES PRECONSTRUIDAS DE CREACIÓN

NumPy proporciona diversas funciones predefinidas que facilitan la creación de arreglos sin necesidad de ingresar manualmente los valores. Estas funciones permiten generar arreglos con patrones específicos, lo que resulta útil en cálculos científicos, simulaciones y procesamiento de datos.

arange(): Arreglos con Valores Definidos

La función **arange()** permite crear arreglos de valores dentro de un rango determinado con un paso específico. Es similar a la función **range()** de Python, pero en lugar de devolver una lista, devuelve un arreglo NumPy optimizado para cálculos numéricos.

Sintaxis:

```
numpy.arange(inicio, fin, paso)
```

Ilustración 12 Sintaxis función arange()

- inicio: Valor inicial del rango (opcional, por defecto es 0).
- fin: Valor hasta donde se generarán los números (sin incluirlo).
- paso: Incremento entre los valores generados (opcional, por defecto es 1).

Ejemplo:

```
import numpy as np
arr = np.arange(1, 10, 2)  # Genera [1, 3, 5, 7, 9]
print(arr)
```

Ilustración 13 Ejemplo función arange()

Matrices de Ceros y Unos

NumPy ofrece funciones para generar matrices llenas de ceros o unos, útiles en inicialización de estructuras o procesamiento de datos.

```
ceros = np.zeros((3,3))  # Matriz 3x3 de ceros
unos = np.ones((2,2))  # Matriz 2x2 de unos
```

Ilustración 14 Ejemplo matrices de 0 y 1

Vector con Distribución de Puntos (linspace())

La función **linspace()** genera un arreglo con valores equidistantes entre un punto inicial y final. Es útil cuando necesitamos un número específico de valores dentro de un intervalo.

Sintaxis:

```
numpy.linspace(inicio, fin, num_puntos)
```

Ilustración 15 Sintaxis función linspace()

- inicio: Primer valor del vector.
- fin: Último valor del vector.
- num_puntos: Cantidad de valores a generar.

Ejemplo

```
puntos = np.linspace(0, 10, 5)  # 5 valores entre 0 y 10
print(puntos)
```

Ilustración 16 Ejemplo función linspace()

Matriz de Identidad

La matriz identidad es una matriz cuadrada con valores 1 en la diagonal principal y 0 en el resto. Es utilizada en álgebra lineal.

Ejemplo:

```
identidad = np.eye(4)  # Matriz identidad 4x4
print(identidad)
```

Ilustración 17 Ejemplo matriz de identidad

Matriz Aleatoria

La función **random.rand()** de NumPy genera una matriz con valores aleatorios entre 0 y 1.

Ejemplo

```
aleatoria = np.random.rand(3,3)  # Matriz 3x3 de valores aleatorios
print(aleatoria)
```

Ilustración 18 Ejemplo matriz aleatoria

Para valores enteros en un rango específico, usamos randint():

```
np.random.randint(1, 100, (2, 4))  # Matriz 2x4 con enteros entre 1 y 100
```

Ilustración 19 Ejemplo randint()

Redimensionado de un Arreglo

Podemos cambiar la forma de un arreglo sin alterar sus datos usando **.reshape().**

Ejemplo:

```
arreglo = np.arange(9)
arreglo_reshaped = arreglo.reshape(3,3)
print(arreglo_reshaped)
```

Ilustración 20 Ejemplo de reshape()

Indexación y Selección en NumPy

- **Selección de elementos**
  Podemos acceder a elementos específicos de un arreglo usando índices, similar a las listas en Python.

  Ejemplo:

```
arreglo = np.array([10, 20, 30, 40, 50])
print(arreglo[2])  # Índice 2 (tercer elemento)
```

Ilustración 21 Ejemplo selección de elementos

Para matrices, se usa la notación [fila, columna]:

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

Ilustración 22 Ejemplo selección de elementos matrices

- **Selección condicional**

  NumPy permite seleccionar elementos que cumplen una condición específica.

  Ejemplo:

```
arr = np.array([10, 20, 30, 40, 50])
print(arr[arr > 25])  # Devuelve [30, 40, 50]
```

Ilustración 23 Ejemplo selección condicional

Referencia y Copia de Arreglos

En NumPy, las asignaciones pueden compartir memoria o generar copias nuevas.

- Referencia: Modifica el arreglo original.

```
original = np.array([1,2,3])
modifica = original # modifica apunta a los mismos datos de original
modifica[0] = 99
print(original) # modifica también se modifica
```

Ilustración 24 Ejemplo modificación de arreglo original

- Copia: Crea un nuevo arreglo independiente.

```
copia = original.copy()
copia[0] = 99
print(original) #[1,2,3]
print(original) #[99,2,3]
```

Ilustración 25 Ejemplo de creación de un nuevo arreglo

OPERACIONES CON NUMPY

Operaciones Entre Arreglos

NumPy permite realizar operaciones matemáticas entre arreglos sin necesidad de bucles.

Ejemplo:

```
arr1 = np.array([1, 2, 3])
arr2 = np.array([4, 5, 6])
print(arr1 + arr2)  # [5 7 9]
```

Ilustración 26 Ejemplo operaciones entre arreglos

Otras operaciones

```
resta = arr1 - arr2
producto = arr1 * arr2
division = arr1 / arr2
```

Ilustración 27 Ejemplo de otras operaciones

Operaciones con Escalares

Podemos realizar operaciones entre arreglos y valores escalares, afectando a todos los elementos del arreglo.

Ejemplo:

```
arreglo = np.array([2, 4, 6])
print(arreglo * 2)  # [4 8 12]
```

Ilustración 28 Ejemplo operaciones con escalares

APLICANDO FUNCIONES A UN ARREGLO

NumPy incluye funciones matemáticas optimizadas para operaciones con arreglos.

Ejemplos:

```
arr = np.array([1, 4, 9, 16])
raiz = np.sqrt(arr)  # Calcula la raíz cuadrada de cada elemento
seno = np.sin(arr)   # Calcula el seno de cada elemento
logaritmo = np.log(arr)  # Calcula el logaritmo natural
```

Ilustración 29 Ejemplo para aplicar funciones a un arreglo

NumPy es una herramienta clave en ciencia de datos y computación. Sus funciones optimizadas hacen más fácil trabajar con datos numéricos, desde crear arreglos hasta hacer operaciones complejas. Aprender a usar NumPy ayuda a procesar y analizar datos en Python.