Tema 12: Numpy y los arrays

Lectura fácil

Numpy y los Arrays en Python

Python, en su forma estándar, no tiene un tipo de dato denominado 'array' como ocurre en otros lenguajes de programación como C o Java. Sin embargo, cuando trabajamos con grandes volúmenes de datos numéricos o necesitamos realizar cálculos matemáticos complejos de manera eficiente, podemos utilizar la biblioteca **NumPy** (Numerical Python). NumPy proporciona una estructura de datos llamada **array** que resulta muy útil para estas tareas, y que tiene algunas características similares a las listas, pero es mucho más eficiente.

1. Introducción a NumPy

NumPy es una de las bibliotecas fundamentales para el cálculo numérico en Python. Proporciona herramientas para trabajar con arreglos multidimensionales y funciones matemáticas eficientes. NumPy es muy utilizado en el ámbito de la ciencia de datos, la inteligencia artificial, el aprendizaje automático y otras disciplinas que requieren procesamiento de datos a gran escala.

Para utilizar NumPy, primero debes instalarlo y luego importarlo en tu código. Puedes instalarlo mediante **pip**:

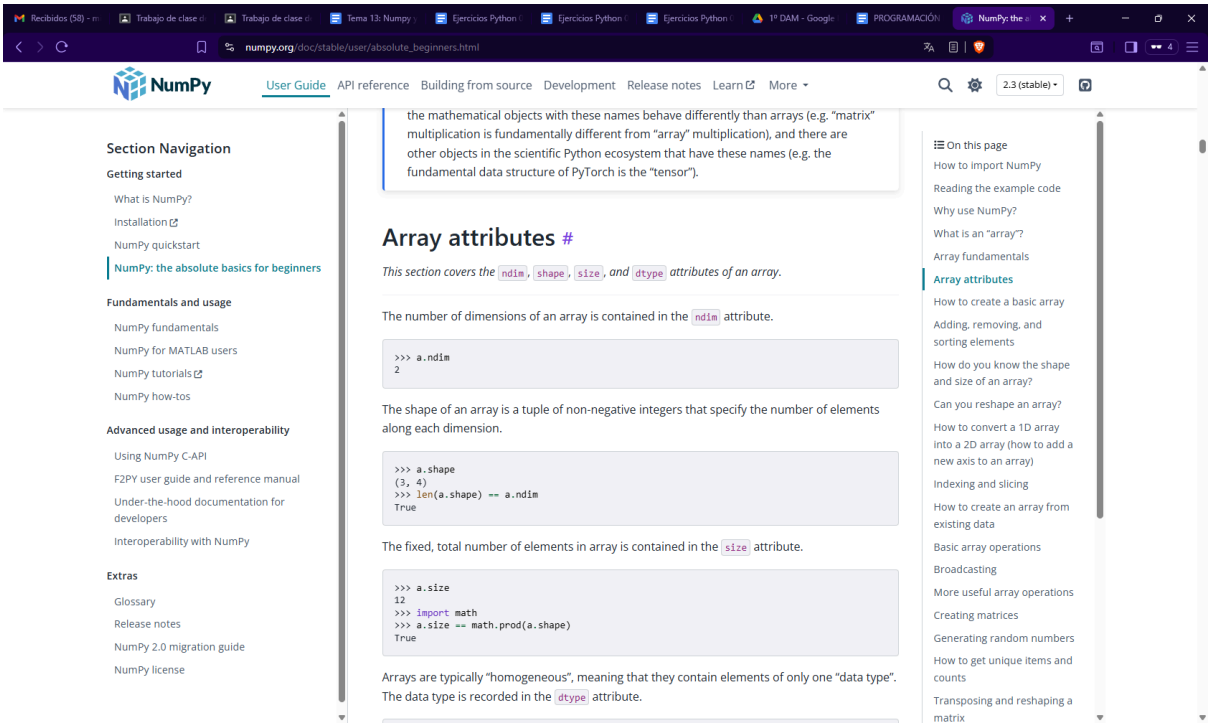
```
pip install numpy
```

Y luego, en tu script de Python, lo puedes importar de la siguiente manera:

```
import numpy as np
```

El uso de np es una convención que permite acceder a las funciones de NumPy de una manera más cómoda.

Podemos encontrar toda la documentación de la librería Numpy, en la página web del proyecto: https://numpy.org/es/



2. Creación de Arrays en NumPy

La estructura de datos principal en NumPy es el **array** (también llamado *ndarray*, que significa arreglo n-dimensional). Los arrays de NumPy se parecen a las listas de Python, pero tienen algunas diferencias importantes: son más rápidos y ocupan menos espacio. Vamos a explorar cómo crearlos.

2.1 Crear un Array desde una Lista

Puedes crear un **array** de NumPy a partir de una lista de Python usando la función `np.array()`:

```
import numpy as np

mi_lista = [1, 2, 3, 4]

mi_array = np.array(mi_lista)

print(mi_array)  # Salida: [1 2 3 4]
```

Este `mi_array` tiene características diferentes a las listas de Python. Por ejemplo, permite operaciones matemáticas vectorizadas, que son mucho más eficientes.

2.2 Arrays Multidimensionales

NumPy también permite crear **arrays multidimensionales** (por ejemplo, matrices). Para crear una matriz de 2x3, puedes pasar una lista de listas:

```
mi_matriz = np.array( [ [ 1, 2, 3 ], [ 4, 5, 6 ] ] )

print(mi_matriz)

# Salida:

# [[1 2 3]    #El valor 2 sería el elemento (0,1) de mi array - fila 0 columna 1

#  [4 5 6]]   #El valor 6 sería el elemento (1,2) de mi array - fila 1 columna2
```

Los arrays de múltiples dimensiones se utilizan mucho para representar datos tabulares, imágenes, y otras estructuras complejas.

2.3 Arrays Especiales

NumPy incluye funciones para crear **arrays especiales** de manera sencilla. Algunas de las más comunes son:

`np.zeros()`: Crea un array de ceros.

```
ceros = np.zeros((3, 4))

print(ceros)

# Salida: una matriz de 3x4 con todos los valores en 0
```

`np.ones(): Crea un array de unos.`

```
unos = np.ones((2, 2))

print(unos)

# Salida: una matriz de 2x2 con todos los valores en 1
```

`np.arange()`: Similar a `range()` en Python, crea un array con una secuencia de valores.

```
secuencia = np.arange(0, 10, 2)

print(secuencia)  # Salida: [0 2 4 6 8]
```

`np.linspace()`: Crea un array de valores espaciados uniformemente entre dos números.

```
valores = np.linspace(0, 1, 5)

print(valores)  # Salida: [0.0   0.25 0.5  0.75   1.0  ]
```

3. Operaciones con Arrays

Una de las ventajas de usar **arrays de NumPy** es la capacidad de realizar operaciones matemáticas de forma más directa y eficiente. A diferencia de las listas de Python, puedes realizar operaciones aritméticas con arrays sin necesidad de utilizar bucles.

3.1 Operaciones Elemento a Elemento

Puedes sumar, restar, multiplicar y dividir arrays de forma directa.

```
array1 = np.array([1, 2, 3, 4])

array2 = np.array([10, 20, 30, 40])



suma = array1 + array2

print(suma)  # Salida: [11 22 33 44]



producto = array1 * array2

print(producto)  # Salida: [10 40 90 160]
```

NumPy realiza estas operaciones **elemento a elemento**, lo cual es muy eficiente.

3.2 Operaciones Matemáticas

NumPy incluye una amplia gama de **funciones matemáticas** que se pueden aplicar a los arrays:

```
valores = np.array([1, 4, 9, 16])

raices = np.sqrt(valores)

print(raices)  # Salida: [1.0  2.0  3.0  4.0]
```

También puedes calcular el **promedio** de un array usando `np.mean()`:

```
promedio = np.mean(valores)

print(promedio)  # Salida: 7.5
```

4. Acceso y Modificación de Elementos

Acceder y modificar elementos de un **array de NumPy** es similar a trabajar con listas en Python.

4.1 Acceder a Elementos

Puedes acceder a los elementos de un array utilizando **índices**:

```
mi_array = np.array([10, 20, 30, 40])

print(mi_array[2])  # Salida: 30
```

Para **arrays multidimensionales**, puedes acceder a los elementos especificando los índices de cada dimensión:

```
mi_matriz = np.array([[1, 2, 3], [4, 5, 6]])

print(mi_matriz[1, 2])  # Salida: 6
```

4.2 Modificar Elementos

También puedes modificar los valores de un array de forma similar a como lo harías con listas:

```
mi_array[0] = 100
```

```
print(mi_array)  # Salida: [100  20  30  40]
```

5. Ventajas de Usar NumPy sobre las Listas de Python

- **Velocidad**: NumPy es mucho más rápido que las listas de Python para realizar cálculos matemáticos.
- **Menor uso de memoria**: Los arrays de NumPy ocupan menos memoria comparados con las listas.
- **Operaciones avanzadas**: NumPy tiene muchas funciones avanzadas para álgebra lineal, estadísticas, y más, que hacen que trabajar con grandes cantidades de datos sea más eficiente y sencillo.

6. Ejemplos Prácticos

Ejemplo 1: Sumar Dos Arrays

Supongamos que tenemos dos listas de calificaciones y queremos sumarlas para obtener el total:

```
import numpy as np


calificaciones_1 = [80, 90, 70, 85]

calificaciones_2 = [75, 85, 95, 80]


# Convertir las listas a arrays de NumPy

array1 = np.array(calificaciones_1)

array2 = np.array(calificaciones_2)


total = array1 + array2

print(total)  # Salida: [155 175 165 165]
```

Ejemplo 2: Crear una Matriz de Identidad

La **matriz identidad** es una matriz cuadrada con unos en la diagonal principal y ceros en el resto de las posiciones. Es posible crear una matriz identidad con NumPy usando `np.eye()`:

```
matriz_identidad = np.eye(3)

print(matriz_identidad)

# Salida:

# [[1. 0. 0.]

#  [0. 1. 0.]

#  [0. 0. 1.]]
```

7. Filtros en Arrays de Numpy

Filtrar datos en Numpy es una de las operaciones más prácticas, ya que permite seleccionar elementos de un array que cumplen con una condición específica sin necesidad de bucles adicionales.

¿Cómo Filtrar Datos?

La forma más sencilla de aplicar un filtro en Numpy es mediante **condiciones** que actúan sobre el array completo, devolviendo solo los elementos que cumplen con dicha condición.

Ejemplo: Filtrar Números Mayores a un Valor

Imaginemos que tenemos un array de valores y queremos quedarnos solo con aquellos que son mayores a 10:

```
import numpy as np


# Array de ejemplo

array = np.array([5, 12, 3, 18, 7, 20, 9])
```

```
# Aplicar el filtro

array_filtrado = array[array > 10]

print(array_filtrado)
```

Salida:

```
[12 18  20]
```

En este ejemplo, `array > 10` crea una condición que se aplica a cada elemento del array. Los elementos que cumplen la condición (`True`) se mantienen en `array_filtrado`.

Más Ejemplos de Filtros
Filtrar valores pares de un array:

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

pares = array[array % 2 == 0]

print(pares)  # Salida: [2 4 6 8]
```

Filtrar valores dentro de un rango específico:

```
array = np.array([10, 15, 20, 25, 30, 35])

rango = array[(array >= 15) & (array <= 30)]

print(rango)  # Salida: [15 20 25 30]
```

Los filtros en Numpy permiten realizar operaciones de selección complejas de forma eficiente, aprovechando la capacidad de Numpy para aplicar operaciones a nivel de array sin necesidad de bucles.

8. Funciones Estadísticas Comunes en Numpy

Numpy incluye una variedad de funciones estadísticas que facilitan el análisis de datos en arrays, proporcionando métodos rápidos y eficientes para calcular medidas comunes. A continuación, se muestra una tabla con las funciones estadísticas más utilizadas:

Función	Descripción	Ejemplo de Uso
`np.mean(array)`	Calcula el promedio de todos los elementos	`np.mean(array)`
`np.sum(array)`	Suma todos los elementos del array	`np.sum(array)`
`np.min(array)`	Encuentra el valor mínimo en el array	`np.min(array)`
`np.max(array)`	Encuentra el valor máximo en el array	`np.max(array)`
`np.std(array)`	Calcula la desviación estándar de los elementos	`np.std(array)`
`np.var(array)`	Calcula la varianza de los elementos	`np.var(array)`
`np.median(array)`	Calcula la mediana de los elementos	`np.median(array)`
`np.percentile(array, q)`	Calcula el percentil q de los elementos	`np.percentile(array, 50)`
`np.cumsum(array)`	Calcula la suma acumulativa de los elementos	`np.cumsum(array)`
`np.cumprod(array)`	Calcula el producto acumulativo de los elementos	`np.cumprod(array)`

Estas funciones son útiles para analizar rápidamente los datos almacenados en arrays de Numpy, proporcionando herramientas para calcular medidas descriptivas, acumulativas y de dispersión.

9. Recapitulando

NumPy es una herramienta extremadamente poderosa para trabajar con datos numéricos en Python. Aunque las listas de Python son útiles para muchos casos, los **arrays de NumPy** son mucho más eficientes para cálculos matemáticos y procesamiento de datos a gran escala. Si estás trabajando en proyectos que involucran manipulación de datos complejos o matemáticas

avanzadas, NumPy es la opción recomendada para hacer que tu código sea más rápido y más fácil de escribir.

El uso de **arrays** en NumPy permite trabajar de manera más eficiente, utilizando menos memoria y ganando velocidad de ejecución. Practicar con NumPy te ayudará a sacarle el máximo provecho y a escribir programas que manejen datos de una forma más efectiva y poderosa.
