**Numpy**

**¿Qué es NumPy?**

Es una biblioteca de Python diseñada específicamente para el cálculo numérico. NumPy, que significa "Numerical Python" (Python Numérico), proporciona una estructura de datos llamada "arreglo" Los arreglos de NumPy son multidimensionales, lo que significa que pueden tener una o más dimensiones, como filas y columnas en una matriz.

Ofrece muchas funciones y operaciones matemáticas que son extremadamente rápidas y optimizadas para trabajar con arreglos. Esto se debe a que NumPy está escrito en un lenguaje de programación llamado C, que es muy rápido en comparación con Python. Cuando realizas cálculos con NumPy, estos se ejecutan en el bajo nivel de C, lo que brinda un rendimiento mucho mejor que si lo hicieras solo con Python.

Además de la eficiencia, NumPy también proporciona una amplia gama de funciones para realizar operaciones matemáticas en los arreglos.

Dimensiones se refiere a cuantas filas y cuantas columnas

**¿Cuál es la función principal de NumPy?**

**¿Cómo se crea un arreglo unidimensional en NumPy?**

Se utilizan [corchetes], puede ser uni o multidimensional, comienzan las dimensiones y los elementos en el indice en 0

arr = np.array([1, 2, 3, 4, 5]) # unidimensional

print("Elemento en la posición 2:", arr[2])

**¿Cuál es la diferencia entre una lista de Python y un arreglo de NumPy?**

Una lista es un tipo de datos abstractos que implica una secuencia ordenada de valores. Un array es un “tipo de lista”, pero una lista no necesariamente es un array.

**¿Cuál es la ventaja de utilizar operaciones vectorizadas en NumPy en lugar de bucles de Python?**

Debido a que se ejecuta la operación en código compilado, la versión de NumPy de la operación se calcula mucho más rápido.

Ambas pueden ser indexadas e iteradas, pero una lista no te permite hacer operaciones artiméticas, como la division.

Para usar arrays se debe tomar un paso mas, que es declararlos, mientras que para usar una lista no, ya que ellas forman parte de la sintaxis de Python.

**¿Qué es el broadcasting en NumPy?**

El broadcasting permite que este tipo de operaciones binarias se realicen en arreglos de diferentes tamaños; por ejemplo, podemos sumar un escalar (pensado como un arreglo de dimensiones cero) a un arreglo:

**¿Cuál es la función de np.sum() en NumPy?**

sum(L)

np.sum(L) sumar

**¿Cuál es la forma recomendada de importar NumPy en un programa de Python?**

import numpy as np

**Atributos de un array**

**Tamaño de los arreglos**: Puedes obtener el tamaño de un arreglo utilizando el atributo size. Este atributo devuelve el número total de elementos en el arreglo.

**Forma de los arreglos**: El atributo shape te permite conocer la estructura del arreglo, es decir, el número de elementos en cada dimensión. Si tienes un arreglo unidimensional, el atributo shape devolverá una tupla con un solo valor que representa el tamaño del arreglo. Si tienes un arreglo multidimensional, shape devolverá una tupla con el tamaño en cada dimensión.

**Consumo de memoria de los arreglos:** El atributo nbytes te permite saber cuánta memoria está ocupando el arreglo en bytes. Es útil para evaluar la eficiencia del uso de memoria de tu código.

**Tipos de datos de los arreglos**: Puedes utilizar el atributo dtype para obtener el tipo de datos de los elementos en el arreglo. NumPy ofrece varios tipos de datos, como int, float, bool, string, entre otros.

**Tipo de datos del arreglo: int64**

Indexado

El indexado en NumPy se realiza utilizando corchetes [] y puede ser de una o varias dimensiones, dependiendo de la forma del arreglo.

**Indexado unidimensional:** Si tienes un arreglo unidimensional, puedes acceder a los elementos individuales utilizando un índice entre corchetes. El índice comienza en cero para el primer elemento y va incrementando de uno en uno.

**Indexado multidimensional**: Si tienes un arreglo multidimensional, el indexado se realiza especificando el índice en cada dimensión separado por comas. Puedes acceder a elementos individuales o a subarreglos utilizando esta notación.

Establecer el valor de un elemento: Además de obtener valores, también puedes establecer nuevos valores en un arreglo.

**Indexado con rangos**: Puedes utilizar rangos para acceder a subarreglos dentro de un arreglo. Los rangos se especifican utilizando la notación start:end:step, donde start es el índice de inicio (incluido), end es el índice final (excluido) y step es el incremento entre los elementos.

El "slicing" o rebanado es una técnica muy útil para acceder a subarreglos dentro de un arreglo multidimensional. Permite seleccionar porciones específicas del arreglo en una o varias dimensiones, lo que facilita la manipulación y extracción de datos.

El rebanado en NumPy se realiza especificando rangos en cada dimensión, separados por comas dentro de los corchetes []. Puedes utilizar los rangos para especificar el inicio (start), fin (stop) y el paso (step) del rebanado.

Veamos algunos ejemplos de cómo se utiliza el rebanado en arreglos multidimensionales:

En este ejemplo, hemos seleccionado una porción del arreglo original arr que incluye las filas de índice 0 y 1, y las columnas de índice 1 y 2. El resultado es un nuevo subarreglo [2 3; 6 7].

Además de especificar rangos, también puedes omitir valores para seleccionar todas las filas o columnas en una dimensión determinada. Por ejemplo:

Aquí hemos seleccionado todas las filas (:) y la tercera columna (2) del arreglo arr. El resultado es un nuevo arreglo unidimensional [3 7 11] que contiene los elementos de la tercera columna.

También puedes utilizar valores negativos en el rebanado para contar desde el final del arreglo. Por ejemplo:

En este caso, hemos seleccionado las últimas dos filas (-2:) y las últimas dos columnas (-2:) del arreglo arr, obteniendo un nuevo subarreglo [[7 8]; [11 12]].

**Reshape**

La modificación de la forma de los arreglos en NumPy se refiere a cambiar la estructura o dimensión de un arreglo existente sin modificar sus elementos. Esto se logra utilizando el método reshape() de NumPy.

En este ejemplo, hemos creado un arreglo unidimensional arr con 6 elementos. Luego utilizamos el método reshape() para cambiar su forma a una matriz de 2 filas y 3 columnas. El resultado es el arreglo reshaped\_arr, que tiene la forma [[1 2 3]; [4 5 6]].

La modificación de la forma de un arreglo no altera los elementos en sí, simplemente los reorganiza en una nueva estructura. Es importante tener en cuenta que la cantidad total de elementos en el arreglo original debe ser igual a la cantidad total de elementos en la forma deseada.

En este caso, hemos creado un arreglo bidimensional arr de 3 filas y 4 columnas. Luego, utilizamos reshape() para cambiar su forma a una matriz de 2 filas y 6 columnas. El resultado es el arreglo reshaped\_arr, que tiene la forma [[1 2 3 4 5 6]; [7 8 9 10 11 12]].

También es posible utilizar el valor -1 en una dimensión al utilizar reshape(), lo que indica que NumPy debe inferir automáticamente el tamaño adecuado en esa dimensión. Veamos un ejemplo:

En este ejemplo, hemos utilizado reshape() para cambiar la forma del arreglo arr a una matriz de 3 filas y un tamaño de columna automático. NumPy infiere que el tamaño de columna adecuado es 4, lo que resulta en el arreglo reshaped\_arr mostrado arriba.

**Combinaciones de Arrays**

NumPy ofrece varias funciones para combinar múltiples arreglos en uno solo. Algunas de las funciones comunes son concatenate(), vstack() y hstack().

En este ejemplo, utilizamos la función concatenate() para unir los arreglos arr1 y arr2 en uno solo. El resultado es un nuevo arreglo [1 2 3 4 5 6].

También puedes utilizar vstack() para unir arreglos verticalmente (a lo largo del eje 0) o hstack() para unir arreglos horizontalmente (a lo largo del eje 1). Veamos un ejemplo de cada uno:

**Operaciones vectorizadas**

**La lentitud de los bucles**

La implementación predeterminada de Python (conocida como CPython) realiza algunas operaciones de manera muy lenta. Esto se debe en parte a la naturaleza dinámica e interpretada del lenguaje: el hecho de que los tipos sean flexibles, lo que impide que las secuencias de operaciones se compilen en un código de máquina eficiente como en lenguajes como C y Fortran. Recientemente ha habido varios intentos de abordar esta debilidad: ejemplos conocidos son el proyecto PyPy, una implementación de Python compilada en tiempo real (JIT); el proyecto Cython, que convierte el código Python en código C compilable; y el proyecto Numba, que convierte fragmentos de código Python en un bytecode rápido de LLVM. Cada uno de estos enfoques tiene sus fortalezas y debilidades, pero es seguro decir que ninguno de los tres ha superado hasta ahora el alcance y la popularidad del motor CPython estándar.

La relativa lentitud de Python generalmente se manifiesta en situaciones donde se repiten muchas operaciones pequeñas, por ejemplo, recorrer arreglos para operar en cada elemento.

**Vectorización de operaciones**

Para muchos tipos de operaciones, NumPy proporciona una interfaz conveniente para este tipo de rutinas compiladas de tipo estático. Esto se conoce como una operación vectorizada. Esto se puede lograr simplemente realizando una operación en el arreglo, la cual se aplicará a cada elemento. Este enfoque vectorizado está diseñado para trasladar el bucle a la capa compilada que subyace en NumPy, lo que resulta en una ejecución mucho más rápida.

**Operaciones aritméticas**

NumPy sobrecarga los operadores aritméticos en Python para permitir realizar operaciones aritméticas vectorialmente

**Agregaciones**

NumPy tiene varias funciones de agregación incorporadas para trabajar con arreglos

**Suma de los valores en un arreglo**

Como ejemplo rápido, consideremos calcular la suma de todos los valores en un arreglo. Python en sí mismo puede hacer esto utilizando la función sum incorporada

la función sum y la función np.sum no son idénticas, ¡lo que a veces puede generar confusión! En particular, sus argumentos opcionales tienen diferentes significados y np.sum puede manejar arreglos de múltiples dimensiones.

**Mínimo y máximo**

De manera similar, Python tiene las funciones incorporadas min y max, que se utilizan para encontrar el valor mínimo y el valor máximo de cualquier arreglo dado:

Python: min(big\_array), max(big\_array)

Numpy: np.min(big\_array), np.max(big\_array)

Para min, max, sum y varios otros agregados de NumPy, hay una sintaxis más corta para usar los métodos del propio objeto de arreglo:

print(big\_array.min(), big\_array.max(), big\_array.sum())

**Agregaciones multidimensionales**

Un tipo común de operación de agregación es un agregado a lo largo de una fila o columna.

Las funciones de agregación toman un argumento adicional que especifica el eje a lo largo del cual se calcula el agregado. Por ejemplo, podemos encontrar el valor mínimo dentro de cada columna especificando axis=0: M.min(axis=0)

De manera similar, podemos encontrar el valor máximo dentro de cada fila: M.max(axis=1)

La palabra clave axis especifica la dimensión del arreglo que se colapsará, en lugar de la dimensión que se devolverá. Por lo tanto, especificar axis=0 significa que se colapsará el primer eje: para arreglos bidimensionales, esto significa que se agregarán los valores dentro de cada columna.

**Broadcasting**

Recordemos que, para arreglos del mismo tamaño, las operaciones binarias se realizan elemento por elemento:

El broadcasting permite que este tipo de operaciones binarias se realicen en arreglos de diferentes tamaños; por ejemplo, podemos sumar un escalar (pensado como un arreglo de dimensiones cero) a un arreglo

Buenos días Angélica, espero te encuentres bien.

Quisiera pedirte un favor, no he podido participar en las tutorías de los sábados y estoy atrasadísima en el curso. Te puedo enviar mis soluciones para ver porque algunas no me corren.

Te agradecería.

Saludos.