Numpy

Introducción al paquete Numpy

CertiDevs

Índice de contenidos

1. ¿Qué es NumPy?

NumPy (Numerical Python) es una biblioteca de Python que proporciona soporte para trabajar con arreglos multidimensionales, matrices y funciones matemáticas de alto nivel.

NumPy es ampliamente utilizado en la comunidad científica y de análisis de datos para realizar **operaciones matemáticas** y **estadísticas**, así como para manipular datos en forma de arreglos y matrices.

Sitio web oficial de NumPy

2. ¿Para qué sirve NumPy?

NumPy es esencial para realizar **cálculos numéricos** en Python.

Proporciona **estructuras de datos** eficientes y optimizadas para **cálculos matemáticos** y ofrece una amplia gama de **funciones matemáticas** que permiten realizar operaciones avanzadas con facilidad.

Algunas de las aplicaciones de NumPy incluyen:

- Álgebra lineal
- Análisis estadístico
- Generación de números aleatorios
- Operaciones con matrices
- Interfaz de bajo nivel para operaciones de alto rendimiento y compatibilidad con bibliotecas de C/C++

3. ¿Por qué aprender NumPy?

Aprender NumPy es fundamental para cualquier persona que trabaje en el campo del **análisis de datos**, aprendizaje automático (machine learning), inteligencia artificial o ciencias.

NumPy es una herramienta versátil y eficiente que facilita el trabajo con **datos numéricos** y simplifica la realización de **cálculos matemáticos complejos**.

Además, muchas otras bibliotecas de Python, como **Pandas**, **Matplotlib** y **Scikit-learn**, se basan en **NumPy**, por lo que comprender su funcionamiento es esencial para utilizar estas herramientas de manera efectiva.

4. Popularidad de NumPy

NumPy es una de las bibliotecas más populares en Python, especialmente en la comunidad científica y de análisis de datos.

Su popularidad se debe a su simplicidad, eficiencia y versatilidad.

NumPy ha sido ampliamente adoptado en una variedad de campos, desde la física y la biología hasta las finanzas y la ingeniería, lo que ha llevado a una amplia comunidad de usuarios y una base de código en constante evolución y mejora.

5. Historia de NumPy

NumPy fue creado en **2005** por Travis Oliphant, un científico e ingeniero de software que buscaba una solución más eficiente y flexible para realizar cálculos numéricos en Python.

Antes de NumPy, existían otras bibliotecas numéricas en Python, como Numeric y Numarray, pero ninguna ofrecía el rendimiento y la facilidad de uso que NumPy proporciona.

Desde su creación, NumPy ha sido desarrollado y mantenido por una comunidad de desarrolladores y usuarios apasionados que han contribuido a su éxito y crecimiento.

6. Diferencia entre arrays de numpy y listas de python

Los **arrays de NumPy** y las **listas de Python** son estructuras de datos para almacenar colecciones de elementos, pero presentan diferencias fundamentales que los hacen adecuados para diferentes propósitos.

Aquí hay algunas diferencias clave entre los arrays de NumPy y las listas de Python:

- **Homogeneidad de tipos de datos**: Los arrays de NumPy son homogéneos, lo que significa que todos los elementos del array deben ser del mismo tipo de datos, podrían crearse de distinto tipo pero afectaría al rendimiento. Por otro lado, las listas de Python pueden almacenar elementos de diferentes tipos de datos en la misma lista.
- **Rendimiento**: Los arrays de NumPy están diseñados para un rendimiento óptimo en operaciones matemáticas y de manipulación de datos. NumPy almacena los datos de manera contigua en la memoria y ofrece funciones altamente optimizadas para operaciones numéricas. Las listas de Python, por otro lado, están diseñadas para ser más generales y no están optimizadas para cálculos numéricos.
- **Operaciones matemáticas y de álgebra lineal**: NumPy proporciona una amplia gama de funciones matemáticas, de álgebra lineal y de manipulación de arrays que no están disponibles con las listas de Python. Los arrays de NumPy permiten operaciones elemento por elemento, broadcasting y otras características que facilitan las operaciones matemáticas.
- **Tamaño**: Los arrays de NumPy tienen un tamaño fijo en el momento de la creación, lo que significa que no puedes agregar ni eliminar elementos del array sin crear un nuevo array. Por otro lado, las listas de Python tienen un tamaño dinámico y puedes agregar o eliminar elementos fácilmente.
- **Memoria**: Los arrays de NumPy son más eficientes en términos de memoria que las listas de Python. Como los arrays de NumPy son homogéneos y tienen un tamaño fijo, su estructura de

datos es más compacta y requiere menos memoria para almacenar la misma cantidad de datos que una lista de Python.

Deberías utilizar arrays de NumPy en lugar de listas de Python cuando:

- Trabajas con datos numéricos y necesitas realizar operaciones matemáticas y de álgebra lineal.
- Necesitas un rendimiento óptimo y una eficiencia de memoria en tus cálculos.
- Deseas utilizar características específicas de los arrays de NumPy, como el broadcasting y las funciones universales (ufuncs).

Las listas de Python son más adecuadas para casos en los que necesitas una estructura de datos flexible y general para almacenar elementos de diferentes tipos o cuando deseas agregar o eliminar elementos de la colección con frecuencia.

7. Instalación de NumPy

Para instalar NumPy, es recomendable utilizar un gestor de paquetes como **pip** o **conda**.

A continuación, se muestran las instrucciones para instalar NumPy usando ambos gestores de paquetes.

7.1. Instalación con pip

Para **instalar NumPy** utilizando **pip**, abra una terminal o una ventana de comandos y ejecute el siguiente comando:

```
pip install numpy
```

7.2. Instalación con conda

Si está utilizando la distribución Anaconda de Python, puede instalar NumPy utilizando el gestor de paquetes **conda**.

Para hacerlo, abra una terminal o una ventana de comandos y ejecute el siguiente comando:

```
conda install numpy
```

8. Comprobación de la versión de NumPy

Una vez instalado NumPy, puede comprobar la versión de la biblioteca ejecutando el siguiente código en un script de Python o en un intérprete interactivo:

```
import numpy as np
```

```
print("Versión de NumPy:", np.__version__)
```

Este código importa NumPy con el alias `np` (un alias comúnmente utilizado en la comunidad de Python) y luego imprime la versión de la biblioteca.

9. Instalar Jupyter Notebook

En caso de querer crear un entorno de desarrollo para trabajar con Python y NumPy, se recomienda instalar Jupyter Notebook.

Para crear y ejecutar archivos Jupyter notebook (.ipynb) desde Visual Studio Code (VSCode), sigue los siguientes pasos:

1. Instalar Visual Studio Code: Si aún no lo has hecho, descarga e instala Visual Studio Code desde el sitio oficial (https://code.visualstudio.com/). Sigue las instrucciones de instalación según tu sistema operativo (Windows, macOS o Linux).
2. Instalar Python: Asegúrate de tener Python instalado en tu computadora. Si aún no lo tienes, descarga e instala la última versión desde el sitio oficial (https://www.python.org/downloads/).
3. Instalar la extensión de Python para VSCode: Abre Visual Studio Code, ve al panel de Extensiones haciendo clic en el ícono de bloques en la barra lateral izquierda (o presiona Ctrl+Shift+X en Windows/Linux o Cmd+Shift+X en macOS). Busca "Python" en la barra de búsqueda y selecciona la extensión oficial de Python desarrollada por Microsoft. Haz clic en el botón "Install" para instalarla.
4. Instalar Jupyter: Abre una terminal o ventana de comandos en tu computadora y ejecuta el siguiente comando para instalar Jupyter:

```
pip install jupyter
```

1. Crear un nuevo archivo Jupyter notebook: Para crear un nuevo archivo Jupyter notebook en VSCode, ve al menú "File" (Archivo) y selecciona "New File" (Nuevo archivo) o presiona Ctrl+N en Windows/Linux o Cmd+N en macOS. Luego, guarda el archivo con la extensión ".ipynb" (por ejemplo, "mi_notebook.ipynb").
2. Abrir un archivo Jupyter notebook existente: Si ya tienes un archivo Jupyter notebook, simplemente ábrelo en VSCode haciendo clic en "File" (Archivo) > "Open File..." (Abrir archivo...) y seleccionando el archivo .ipynb en tu sistema de archivos.
3. Ejecutar celdas de código: Para ejecutar una celda de código en un Jupyter notebook, selecciona la celda y haz clic en el botón "Run" (Ejecutar) en la parte superior de la celda o presiona Shift+Enter. La salida de la celda se mostrará debajo de la misma.
4. Agregar nuevas celdas: Para agregar nuevas celdas, haz clic en el botón "+" en la esquina superior izquierda del notebook, o usa la barra de herramientas en la parte superior del archivo. Puedes cambiar el tipo de celda entre código y texto (Markdown) utilizando la lista desplegable en la barra de herramientas.
5. Guardar y compartir: Guarda tu archivo Jupyter notebook regularmente usando "File" (Archivo)

> "Save" (Guardar) o Ctrl+S en Windows/Linux o Cmd+S en macOS. Puedes compartir tus notebooks con otras personas simplemente compartiendo el archivo .ipynb.