**¿Qué es NumPy?**

NumPy es una librería de Python para computación científica. NumPy significa  Python numérico. Aquí está la descripción oficial de la librería indicada en su página web:

*"NumPy es el paquete fundamental para la computación científica con Python. Contiene entre otras cosas:*

* *un poderoso objeto de*arreglo*N-dimensional*
* *funciones (*radiodifusión*) sofisticadas*
* *herramientas para integrar*código*en C/C++ y Fortran*
* *útiles capacidades de álgebra lineal, transformación de Fourier y números aleatorios*

*Además de sus obvios usos científicos, NumPy puede ser utilizado como un eficiente contenedor multidimensional de datos*genéricos*. Tipos de datos arbitrarios puede ser definidos. Esto permite que NumPy se integre sin problemas y con rapidez con una amplia variedad de bases de datos.*

*NumPy está licenciado bajo el formato BSD, lo que permite la reutilización con pocas restricciones".*

NumPy es una librería de Python tan importante que hay otras librerías (incluyendo pandas) que están construidas enteramente sobre NumPy.

**El principal beneficio de NumPy**

El principal beneficio de NumPy es que permite una generación y manejo de datos extremadamente rápido. NumPy tiene su propia estructura de datos incorporada llamado arreglo que es similar a la lista normal de Python, pero puede almacenar y operar con datos de manera mucho más eficiente.

**Que aprenderemos sobre NumPy**

Los practicantes avanzados de Python pasarán mucho más tiempo trabajando con pandas que trabajando con NumPy. AUn así, dado que pandas se basa en NumPy, es importante comprender los aspectos más importantes de la biblioteca NumPy.

En las siguientes secciones, cubriremos la siguiente información sobre la librería de NumPy:

* Arreglos de NumPy
* Indexación y asignación de NumPy
* Métodos y operaciones de NumPy

**Continuemos**

Pasemos a aprender sobre los arreglos de NumPy, la estructura de datos central con la que todo profesional de NumPy debe estar familiarizado.

**Arreglos de NumPy**

En esta sección, aprenderemos sobre los arreglos de NumPy.

**¿Qué son los Arreglos de NumPy?**

Los arreglos NumPy son la forma principal de almacenar datos utilizando la biblioteca NumPy. Son similares a las listas normales en Python, pero tienen la ventaja de ser más rápidas y tener más métodos integrados.

Los arreglos de NumPy son creados llamando al método array() de la librería de NumPy. Dentro del método, deberías pasar una lista.

A continuación, se muestra un ejemplo de un arreglo de NumPy básico. Ten en cuenta que mientras ejecuto la instrucción import numpy as np al comienzo de este bloque de código, se excluirá de los otros bloques de código en esta sección por razones de brevedad.

import numpy as np

sample\_list = [1, 2, 3]

np.array(sample\_list)

La última línea de ese bloque de código dará como resultado una salida que se ve así.

array([1,2,3])

El contenedor array() indica que esta ya no es una lista normal de Python. En cambio, es un arreglo de NumPy.

**Los dos tipos diferentes de arreglos de NumPy**

Hay dos tipos diferentes de arreglos de NumPy: vectores y matrices.

Los vectores son arreglos de NumPy uni-dimensionales y se ve así:

my\_vector = np.array(['este', 'es', 'un', 'vector'])

Las matrices son arreglo bi-dimensionales y son creadas pasando una lista de lista dentro del método np.array(). Un ejemplo es el siguiente.

my\_matrix = [[1, 2, 3],[4, 5, 6],[7, 8, 9]]

np.array(my\_matrix)

También puedes expandir los arreglos de NumPy para trabajar con matrices de tres, cuatro, cinco, seis o más dimensiones, pero son raras y están en gran parte fuera del alcance de este curso (después de todo, este es un curso sobre programación Python, no álgebra lineal).

**Arreglos de NumPy: Métodos Incorporados**

Los arreglos de NumPy vienen con un número de métodos incorporados útiles. Dedicaremos el resto de esta sección analizando estos métodos en detalle.

**Cómo obtener un rango de números en Python utilizando Numpy**

NumPy tiene un método útil llamado arange que toma dos números y devuelve un arreglo de números enteros que son mayores o iguales a (>=) el primer número y menores que (<) el segundo número.

Un ejemplo del método arange es el siguiente.

np.arange(0,5)

#Devuevle array([0, 1, 2, 3, 4])

También puedes incluir una tercera variable en el método arange que proporciona un tamaño de paso para que la función regrese. Pasar 2 como tercera variable devolverá cada segundo número en el rango, pasar 5 como tercera variable devolverá cada quinto número en el rango, y así sucesivamente.

Un ejemplo de uso de la tercera variable en el método arange se encuentra a continuación.

np.arange(1,11,2)

#Returns array([1, 3, 5, 7, 9])

**Cómo generar Unos y Ceros en Python usando NumPy**

Mientras programas, de vez en cuando necesitará crear arreglos de unos o ceros. NumPy tiene métodos incorporados que te permiten hacer ambas cosas.

Podemos crear arreglos de ceros utilizando el método zeros de NumPy. Le pasas el número de enteros que quisieras crear como el argumento de la función. Un ejemplo es el siguiente.

np.zeros(4)

#Devuelve array([0, 0, 0, 0])

También puedes hacer algo similar utilizando matrices tridimensionales. Por ejemplo, np.zeros(5, 5) crea un arreglo de 5x5 que contiene todos ceros.

Podemos crear arreglos de unos usando un método similar llamado ones. Un ejemplo es el que sigue.

np.ones(5)

#Returns array([1, 1, 1, 1, 1])

**Cómo dividir uniformemente un rango de números en Python usando NumPy**

Hay muchas situaciones en las que tienes un rango de números y te gustaría dividir por igual ese rango de números en intervalos. El método linspace de NumPy está diseñado para resolver este problema. linspace tiene tres argumentos:

1. El inicio del intervalo
2. El fin del intervalo
3. El número de subintervalos en los que deseas que se divida el intervalo

Un ejemplo del método linspace es el siguiente.

np.linspace(0, 1, 10)

#Devuelve array([0, 0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1.0])

**Cómo crear un arreglo Identidad en Python usando NumPy**

Cualquiera que haya estudiado álgebra lineal estará familiarizado con el concepto de un "arregloidentidad", que es un arreglo cuadrada cuyos valores diagonales son todos 1. NumPy tiene una función incorporada que incluye un argumento para construir matrices de identidad. La función es eye.

Algunos ejemplos:

np.eye(1)

#Devuelve un arreglo identidad de 1x1

np.eye(2)

#Devuelve un arreglo identidad de 2x2

np.eye(50)

#Devuelve un arreglo identidad de 50x50

**Cómo crear números aleatorios en Python usando NumPy**

NumPy tiene varios métodos integrados que te permiten crear matrices de números aleatorios. Cada uno de estos métodos comienza con random. A continuación se muestran algunos ejemplos:

np.random.rand(sample\_size)

#Devuelve una muestra de números aleatorios entre 0 y 1.

#El tamaño de la muestra puede ser un número entero (para un arreglo unidimensional) o dos enteros separados por comas (para un arreglo bidimensional).

np.random.randn(sample\_size)

#Devuelve una muestra de números aleatorios entre 0 y 1, siguiendo la distribución normal

#El tamaño de la muestra puede ser un número entero (para un arreglo unidimensional) o dos enteros separados por comas (para un arreglo bidimensional).

np.random.randint(low, high, sample\_size)

#Devuelve una muestra de números enteros que son mayores o iguales que 'low' y menores que 'high'

**Cómo remodelar arreglos de NumPy**

Es muy común tomar un arreglo con ciertas dimensiones y transformar ese arreglo en una forma diferente. Por ejemplo, es posible que tengas un arreglo unidimensional con 10 elementos y desee cambiarla a un arreglo bidimensional de 2x5.

A continuación se muestra un ejemplo:

arr = np.array([0,1,2,3,4,5])

arr.reshape(2,3)

La salida de esta operación es:

array([[0, 1, 2],

[3, 4, 5]])

Ten en cuenta que para usar el método reshape, el arreglo original debe tener la misma cantidad de elementos que el arreglo en el que estás tratando de remodelarlo.

Si tienes curiosidad sobre la forma actual de un arreglo NumPy, puede determinar su forma utilizando el atributo shape de NumPy. Usando nuestra estructura de la variable arr anterior, a continuación se muestra un ejemplo de cómo llamar al atributo shape:

arr = np.array([0,1,2,3,4,5])

arr.shape

#Devuelve (6,)- ten en cuenta que no hay un segundo elemento ya que es un arreglo unidimensional

arr = arr.reshape(2,3)

arr.shape

#Devuelve (2,3)

También puede combinar el método  reshape con el atributo  shape en una línea como esta:

arr.reshape(2,3).shape

#Devuelve (2,3)

**Cómo encontrar el valor máximo y mínimo de un arreglo NumPy**

Para concluir esta sección, aprendamos cuatro métodos útiles para identificar los valores máximo y mínimo dentro de un arreglo NumPy. Trabajaremos con este arreglo:

simple\_array = [1, 2, 3, 4]

Podemos usar el método max para encontrar el máximo valor de un arreglo de NumPy. A continuación se muestra un ejemplo.

simple\_array.max()

#Devuelve 4

Podemos usar también el método argmax para encontrar el índice del máximo valor dentro de un arreglo. Esto es útil cuando deseas encontrar la ubicación del valor máximo pero no necesariametne te interesa su valor en si.

Un ejemplo se observa a continuación.

simple\_array.argmax()

#Devuelve 3

En forma similar, podemos usar los métodos min y argmin para encontrar el valor e índice del mínimo valor dentro de un arreglo de NumPy.

simple\_array.min()

#Devuelve 1

simple\_array.argmin()

#Devuelve 0

**Continuemos**

En esta sección, analizaremos varios atributos y métodos de los arreglos de NumPy.

Seguiremos trabajando en algunos problemas de práctica de arreglos NumPy en la siguiente sección.

**Métodos y Operaciones de NumP**

En esta sección, trabajaremos a través de [varias operaciones incluidas en la biblioteca NumPy.](https://nickmccullum.com/advanced-python/numpy-methods-operations/)

A lo largo de esta sección, asumiremos que el comando import numpy as np ya ha sido ejecutada.

**El arreglo usado en esta sección**

Para esta sección, trabajaré con un arreglo de longitud 4 creada usando np.arange en todos los ejemplos.

Si deseas comparar mi arreglo con las salidas utilizadas en esta sección, así es como creé e imprimí el arreglo:

arr = np.arange(4)

arr

Los valores del arreglo están debajo.

array([0, 1, 2, 3])

**Cómo realizar operaciones aritméticas en Python usando NumPy**

NumPy facilita realizar operaciones aritméticas con arreglos. Puedes realizarlas usando el arreglo y un sólo número, o puedes realizarlas entre dos arreglos NumPy.

Exploramos cada una de las principales operaciones matemáticas a continuación.

**Suma**

Al sumar un sólo número a un arreglo de NumPy, ese número se suma a cada elemento en el arreglo. A continuación se ve un ejemplo:

2 + arr

#Devuelve array([2, 3, 4, 5])

Puedes sumar dos arreglos NumPy usando el operador +. Los arreglos se suman elemento por elemento (lo que significa que los primeros elementos se suman entre si, los segundos elementos se suman se suman entre se, y así sucesivamente).

A continuación se ve un ejemplo.

arr + arr

#Devuelve array([0, 2, 4, 6])

**Resta**

Como la suma, la resta se realiza elemento por elemento para arreglos de NumPy. Puedes encontrar un ejemplo para el caso de un solo número y para el de otro arreglo NumPy a continuación

arr - 10

#Devuelve array([-10, -9, -8, -7])

arr - arr

#Devuelve array([0, 0, 0, 0])

**Multiplicación**

La multiplicación también se realiza elemento por elemento tanto para casos de un sólo número como para casos de opraciones entre arreglos de NumPy.

A continuación se ven dos ejemplo.

6 \* arr

#Devuelve array([ 0, 6, 12, 18])

arr \* arr

#Devuelve array([0, 1, 4, 9])

**División**

En este punto, probablemente no te sorprende saber que la división en arreglos NumPy se realiza elemento por elemento. Un ejemplo de división de arr por un sólo número se ve a continuación:

arr / 2

#Devuelve array([0. , 0.5, 1. , 1.5])

La división tiene una excepción notable en comparación con las otras operaciones matemáticas que hemos visto en esta sección. Dado que no podemos dividir por cero, al hacerlo, el campo correspondiente se completará con un valor nan, que es la abreviatura de Python para "No es un número" (“Not A Number”). Jupyter Notebook también imprimirá una advertencia similar a esta:

RuntimeWarning: invalid value encountered in true\_divide

Un ejemplo de dividir por cero es con un arreglo NumPy que se muestra a continuación.

arr / arr

#Devuelve array([nan, 1., 1., 1.])

Aprenderemos cómo tratar los valores nan con más detalle más adelante en este curso.

**Operaciones complejas en arreglos de NumPy**

Muchas operaciones no se pueden realizar simplemente aplicando la sintaxis normal a un arreglo NumPy. En esta sección, exploraremos varias operaciones matemáticas que tienen métodos incorporados en la biblioteca NumPy.