NUMPY - Vectores y Matrices

Por Jose R. Zapata

NumPy es una libreria de Algebra Lineal para Python, la razón por la cual es tan importante para Data Science con Python es que casi todas las librerias del ecosistema de PyData confían en NumPy como uno de sus principales componentes.

Numpy también es increíblemente rápido, ya que tiene enlaces a librerias en C. Para obtener más información sobre por qué usar Arrays en lugar de listas, mira esto StackOverflow post.

Instrucciones de instalacion

Es altamente recomendable que instale Python utilizando la distribución Anaconda para asegurarse de que todas las dependencias subyacentes (como las bibliotecas de Álgebra lineal) se sincronicen con el uso de una instalación de conda. **Si esta trabajando en Google Colab o Si tiene Anaconda Normalmente ya tiene instalado NumPy**, de ser necesario puede instalar NumPy en el terminal escribiendo:

```
pip install numpy

conda install numpy
```

Si no tiene Anaconda y no lo puede instalar, puede ver como instalar NumPy en la documentacion oficial Numpy's official documentation on various installation instructions.

Importando NumPy

Luego de instalar Numpy la libreria se importa de la siguiente manera:

```
import numpy as np # esta es la forma estandar de importar numpy
```

Numpy tiene muchas funciones integradas. No los cubriremos todos, sino que nos centraremos en algunos de los aspectos más importantes de Numpy: vectores, arreglos, matrices y generación de números.

Arreglos en Numpy (Arrays)

Los arreglos en NumPy son la principal forma de usar Numpy. Los arreglos de NumPy esencialmente vienen en dos tipos: vectores y matrices. Los vectores son estrictamente matrices de 1-d y las matrices son 2-d (pero debe tener en cuenta que una matriz aún puede tener solo una fila o una columna).

Crear Arreglos desde Listas de Python

Podemos crear un arreglo mediante la conversión directa de una lista o lista de listas:

```
my_list = [1,2,3] #lista en python
my_list
```

```
[1, 2, 3]
```

```
np.array(my_list) #conversion de una lista a arreglo mediante el metodo 'array'
```

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

```
a = np.array(my_list) #se peude asignar a una variable
type(a) #tipo de dato
```

```
numpy.ndarray
```

```
a.dtype  #tipo de datos que estan en el arreglo
```

```
dtype('int64')
```

```
my_matrix = [[1,2,3],[4,5,6],[7,8,9]] # Una lista de listas
my_matrix
```

```
[[1, 2, 3], [4, 5, 6], [7, 8, 9]]
```

```
np.array(my_matrix) # conversion de una lista de listas a Matriz
```

```
array([[1, 2, 3],
       [4, 5, 6],
       [7, 8, 9]])
```

```
b = np.array(my_matrix) # aasginar la conversion a una variable
type(b) # tipo de dato
```

```
numpy.ndarray
```

Metodos integrados (Built-in Methods)

Existen muchas funciones para de generar arreglos

arange

Devuelve valores espaciados uniformemente dentro de un intervalo dado. Es muy parecida a la funcion range de las basicas de python.

```
# generar un arreglo de enteros desde 0 hasta 9, recordar que el valor final no se in
np.arange(0,10)  # arange(valor_inicial,valor_final-1)
```

```
array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9])
```

```
np.arange(0,11,2) ## arange(valor_inicial,valor_final-1,paso)
```

```
array([ 0,  2,  4,  6,  8, 10])
```

zeros y ones

Generar arreglos de Ceros o Unos

```
np.zeros(3) # Generar un arreglo de ceros que contenga 3 elementos
```

```
array([0., 0., 0.])
```

```
np.zeros((5,5)) # Generar una matriz de ceros de 5 x 5 elementos
```

```
array([[0., 0., 0., 0., 0.],
       [0., 0., 0., 0., 0.],
       [0., 0., 0., 0., 0.],
       [0., 0., 0., 0., 0.],
       [0., 0., 0., 0., 0.]])
```

```
np.ones(3) # Generar un arreglo de unos que contenga 3 elementos
```

```
array([1., 1., 1.])
```

```
np.ones((3,3)) # Generar una matriz de unos de 5 x 5 elementos
```

```
array([[1., 1., 1.],
       [1., 1., 1.],
       [1., 1., 1.]])
```

linspace

Devuelve números espaciados uniformemente durante un intervalo especificado.

```
# Generar 3 valores iniciando en 0 y terminando en 10 (incluyendolo)
np.linspace(0,10,3) # linspace(valor_inicial, valor_final, numero_de_elementos)
```

```
array([ 0.,  5., 10.])
```

```
# Generar 50 valores iniciando en 0 y terminando en 10 (incluyendolo)
np.linspace(0,10,50)
```

```
array([ 0.        ,  0.20408163,  0.40816327,  0.6122449 ,  0.81632653,
        1.02040816,  1.2244898 ,  1.42857143,  1.63265306,  1.83673469,
        2.04081633,  2.24489796,  2.44897959,  2.65306122,  2.85714286,
        3.06122449,  3.26530612,  3.46938776,  3.67346939,  3.87755102,
        4.08163265,  4.28571429,  4.48979592,  4.69387755,  4.89795918,
        5.10204082,  5.30612245,  5.51020408,  5.71428571,  5.91836735,
        6.12244898,  6.32653061,  6.53061224,  6.73469388,  6.93877551,
        7.14285714,  7.34693878,  7.55102041,  7.75510204,  7.95918367,
        8.16326531,  8.36734694,  8.57142857,  8.7755102 ,  8.97959184,
        9.18367347,  9.3877551 ,  9.59183673,  9.79591837, 10.        ])
```

eye

Crea la matriz identidad

```
np.eye(4) # crea la matriz identidad de 4x4 elementos
```

```
array([[1., 0., 0., 0.],
       [0., 1., 0., 0.],
       [0., 0., 1., 0.],
       [0., 0., 0., 1.]])
```

Numeros Aleatorios (Random)

Numpy tiene diferentes formas de crear arrelgos de numeros aleatorios, el modulo para realizar esto se llama **Random**:

rand

Crea un arreglo de la forma dada y rellenela con muestras aleatorias de una distribución uniforme sobre `[0, 1)`.

```
# creacion de un arreglo de 2 elementos 1 una dimension
np.random.rand(2) #Los numeros aleatorios seran de una distribucion uniforme
```

```
array([0.55464672, 0.60410863])
```

```
# creacion de un arreglo de 5x5
np.random.rand(5,5) #Los numeros aleatorios seran de una distribucion uniforme
```

```
array([[0.62152507, 0.23262413, 0.04667912, 0.141403  , 0.9316874 ],
       [0.21404498, 0.28747701, 0.79732099, 0.24004423, 0.20145937],
       [0.34281437, 0.62585515, 0.66643919, 0.59694899, 0.47552382],
       [0.66862586, 0.66065437, 0.13807543, 0.68819131, 0.30860298],
       [0.80021553, 0.67175583, 0.35132375, 0.85041896, 0.83744445]])
```

randn

Devuelve una muestra (o muestras) de la distribución "estándar normal". A diferencia del rand que es uniforme:

```
# creacion de un arreglo de 2 elementos 1 una dimension
np.random.randn(2) #Los numeros aleatorios seran de una distribucion "estándar normal"
```

```
array([-0.34307289, -0.619604  ])
```

```
# creacion de un arreglo de 5x5
np.random.randn(5,5) #Los numeros aleatorios seran de una distribucion "estándar norm
```

```
array([[ 0.15919913, -0.49995497,  0.03898612, -1.75276706, -1.35012055],
       [ 0.95082787, -2.19552647,  1.36339451,  0.80626886,  0.0098413 ],
       [-1.18150005,  0.46867232,  0.60158901,  0.73681089,  0.4030547 ],
       [-0.5162413 ,  0.88873899, -0.07352698,  0.35424756,  0.53750849],
       [ 1.56882644, -0.88221093, -0.05125666, -0.10572003, -0.58459871]])
```

randint

Entrega numeros enteros aleatorios desde `inicio` (inclusivo) hasta `final` (exclusivo).

```
np.random.randint(1,100) # Genera un numero aleatorio entre 1 y 99
```

```
19
```

```
np.random.randint(1,100,(10,2)) # Genera un arreglo de 10 elementos entre 1 y 99
```

```
array([[47, 84],
       [16, 64],
       [82, 87],
       [33, 58],
       [48,  4],
       [69, 11],
       [82, 53],
       [76, 32],
       [97, 53],
       [ 4, 29]])
```

Atributos y Metodos de los arreglos

```
import numpy as np
arr = np.arange(25) #Genera un arreglo de numeros enteros del 0 al 24
ranarr = np.random.randint(0,50,10) #Genera un arreglo de 10 elementosdel 0 al 49
```

```
arr
```

```
array([ 0,  1,  2,  3,  4,  5,  6,  7,  8,  9, 10, 11, 12, 13, 14, 15, 16,
       17, 18, 19, 20, 21, 22, 23, 24])
```

```
ranarr
```

```
array([41, 20,  5, 13, 40,  1,  7, 43, 33, 34])
```

Reshape

Devuelve una matriz que contiene los mismos datos con una nueva distribucion

```
# arr es un vector de 25 elementos y se convertira en una matriz de 5x5
arr.reshape(5,5)
```

```
array([[ 0,  1,  2,  3,  4],
       [ 5,  6,  7,  8,  9],
       [10, 11, 12, 13, 14],
       [15, 16, 17, 18, 19],
       [20, 21, 22, 23, 24]])
```

max,min,argmax,argmin

Estos son métodos útiles para encontrar valores máximos o mínimos. O para encontrar el indice de su ubicacione usando argmin o argmax

```
ranarr
```

```
array([41, 20,  5, 13, 40,  1,  7, 43, 33, 34])
```

```
ranarr.max() # Valor maximo del arreglo
```

```
43
```

```
ranarr.argmax() # Posicion del valor maximo del arreglo (recordar que empieza en cero
```

```
7
```

```
ranarr.min() # Valor minimo del arreglo
```

```
1
```

```
ranarr.argmin() # Posicion del valor maximo del arreglo (recordar que empieza en cero
```

```
5
```

Shape

Shape es un attribute que los arreglos tienen para definir sus dimensiones (No es metodo):

```
# Vector
arr.shape
```

```
(25,)
```

```
# Cambiando las dimensiones del arreglo para que sea una matriz
# de una sola dimension horizontal
arr.reshape(1,25)
```

```
array([[ 0,  1,  2,  3,  4,  5,  6,  7,  8,  9, 10, 11, 12, 13, 14, 15,
        16, 17, 18, 19, 20, 21, 22, 23, 24]])
```

```
# Cambiando las dimensiones del arreglo para que sea una matriz
# de una sola dimension vertical
arr.reshape(25,1)
```

```
array([[ 0],
       [ 1],
       [ 2],
       [ 3],
       [ 4],
       [ 5],
       [ 6],
       [ 7],
       [ 8],
       [ 9],
       [10],
       [11],
       [12],
       [13],
       [14],
       [15],
       [16],
       [17],
       [18],
       [19],
       [20],
       [21],
       [22],
       [23],
       [24]])
```

```
arr.reshape(25,1).shape
```

```
(25, 1)
```

dtype

Para obtener los tipos de datos dentro del arreglo:

```
arr.dtype
```

```
dtype('int64')
```

size

Numero de elementos en un arreglo

```
arr_2d = np.array(([5,10,15],[20,25,30],[35,40,45]))
arr_2d.size
```

```
9
```

ndim

Numero de dimensiones del arreglo o matriz

```
arr_2d.ndim
```

```
2
```

Indexacion y Seleccion en NumPy

Como indexar y seleccionar elementos o grupos de elementos de un arreglo (array)

```
#Creando un arreglo de ejemplo
arr = np.arange(0,11) # Generar un arreglo de enteros del 0 hasta el 10
arr
```

```
array([ 0,  1,  2,  3,  4,  5,  6,  7,  8,  9, 10])
```

```
arr.dtype # tipos de datos dentro del arreglo
```

```
dtype('int64')
```

Indexación y Selección con corchetes

La forma más sencilla de elegir uno o algunos elementos de una matriz es similar a las listas de Python:

```
# Obtener un valor conociendo su indice (index)
arr[8]
```

```
8
```

```
#Obtener los valores en un rango [valor_inicial, valor_final -1]
arr[1:5]
```

```
array([1, 2, 3, 4])
```

```
#Obtener los valores en un rango [valor_inicial, valor_final -1]
arr[0:5]
```

```
array([0, 1, 2, 3, 4])
```

Broadcasting (Difusion)

Los arreglos de Numpy difieren de una lista normal de Python en su capacidad de Broadcasting, qeu es asignar un valor a un rango de posiciones:

```
# definiendo un valor para todo un rango de posiciones (Broadcasting)
arr[0:5]=100 # Asignar el numero 100 a las posiciones desde el 0 hasta el 4
arr
```

```
array([100, 100, 100, 100, 100,   5,   6,   7,   8,   9,  10])
```

```
#crear nuevamente el arreglo con el que estabamos trabajando
arr = np.arange(0,11)
arr
```

```
array([ 0,  1,  2,  3,  4,  5,  6,  7,  8,  9, 10])
```

```
#NOTA importante en la seleccion de rangos (sclicing)
# los arreglos son mutables
slice_of_arr = arr[0:6]
slice_of_arr
```

```
array([0, 1, 2, 3, 4, 5])
```

```
#Cambiar todos los valores a 99
slice_of_arr[:]=99
slice_of_arr
```

```
array([99, 99, 99, 99, 99, 99])
```

Observe que los cambios tambien ocurren en el arreglo original

```
arr
```

```
array([99, 99, 99, 99, 99, 99,  6,  7,  8,  9, 10])
```

Los datos no se copian, ¡es un puntero a el arreglo original! ¡Esto evita problemas de memoria!

```
# Para obtener una copia, se debe hacer explicito
arr = np.arange(0,11)
arr_copy = arr.copy()
arr_copy[:] = 99
arr_copy
```

```
array([99, 99, 99, 99, 99, 99, 99, 99, 99, 99, 99])
```

```
# Observe que el arreglo original no se modifico
arr
```

```
array([ 0,  1,  2,  3,  4,  5,  6,  7,  8,  9, 10])
```

Indexacion de arreglos 2D (matrices)

El formato general es **arr_2d[fila][col]** o **arr_2d[fila,col]**. Se recomienda usar la notacion con la coma por claridad.

```
# Creacion de una matriz de 3x3
arr_2d = np.array(([5,10,15],[20,25,30],[35,40,45]))

arr_2d
```

```
array([[ 5, 10, 15],
       [20, 25, 30],
       [35, 40, 45]])
```

```
#Indexando por filas
arr_2d[1] # Obtener la fila 1 (recordar que el indice de python empieza en 0)
```

```
array([20, 25, 30])
```

```
# El formato es **arr_2d[fila][col]** o **arr_2d[fila,col]**
# Obteniendo un elemento en especifico
arr_2d[1][0] # elemento de la fila 1 columna 0
```

```
20
```

```
# Obteniendo un elemento en especifico
arr_2d[1,0] # elemento de la fila 1 columna 0
```

```
20
```

```
# seleccion de rangos en arreglos 2D (slicing)

#dimensiones (2,2) desde la esquina superior derecha
arr_2d[:2,1:] # filas [0,1] y columnas 1 hasta el final
```

```
array([[10, 15],
       [25, 30]])
```

```
# fila de indice 2
arr_2d[2]
```

```
array([35, 40, 45])
```

```
# todos los elementos de las columnas que esten en la fila de posicion 2
arr_2d[2,:]
```

```
array([35, 40, 45])
```

Indexacion especial

La indexación especial permite seleccionar filas o columnas enteras desordenadas

```
#Creando una matriz de zeros
arr2d = np.zeros((10,10))
arr2d
```

```
array([[0., 0., 0., 0., 0., 0., 0., 0., 0., 0.],
       [0., 0., 0., 0., 0., 0., 0., 0., 0., 0.],
       [0., 0., 0., 0., 0., 0., 0., 0., 0., 0.],
       [0., 0., 0., 0., 0., 0., 0., 0., 0., 0.],
       [0., 0., 0., 0., 0., 0., 0., 0., 0., 0.],
       [0., 0., 0., 0., 0., 0., 0., 0., 0., 0.],
       [0., 0., 0., 0., 0., 0., 0., 0., 0., 0.],
       [0., 0., 0., 0., 0., 0., 0., 0., 0., 0.],
       [0., 0., 0., 0., 0., 0., 0., 0., 0., 0.],
       [0., 0., 0., 0., 0., 0., 0., 0., 0., 0.]])
```

```
#Tamaño del arreglo
arr_length = arr2d.shape[1]
arr_length
```

```
10
```

```
# Creando una matriz con elementos que contienen el valor correspondiente a la posici
for i in range(arr_length):
    arr2d[i] = i

arr2d
```

```
array([[0., 0., 0., 0., 0., 0., 0., 0., 0., 0.],
       [1., 1., 1., 1., 1., 1., 1., 1., 1., 1.],
       [2., 2., 2., 2., 2., 2., 2., 2., 2., 2.],
       [3., 3., 3., 3., 3., 3., 3., 3., 3., 3.],
       [4., 4., 4., 4., 4., 4., 4., 4., 4., 4.],
       [5., 5., 5., 5., 5., 5., 5., 5., 5., 5.],
       [6., 6., 6., 6., 6., 6., 6., 6., 6., 6.],
       [7., 7., 7., 7., 7., 7., 7., 7., 7., 7.],
       [8., 8., 8., 8., 8., 8., 8., 8., 8., 8.],
       [9., 9., 9., 9., 9., 9., 9., 9., 9., 9.]])
```

La indexacion especial permite:

```
# sacar las filas 2, 4 , 6, 8
arr2d[[2,4,6,8]] # observe el uso de los dobles corchetes
```

```
array([[2., 2., 2., 2., 2., 2., 2., 2., 2., 2.],
       [4., 4., 4., 4., 4., 4., 4., 4., 4., 4.],
       [6., 6., 6., 6., 6., 6., 6., 6., 6., 6.],
       [8., 8., 8., 8., 8., 8., 8., 8., 8., 8.]])
```

```
#Permite cualquier orden
arr2d[[6,4,2,7]]
```

```
array([[6., 6., 6., 6., 6., 6., 6., 6., 6., 6.],
       [4., 4., 4., 4., 4., 4., 4., 4., 4., 4.],
       [2., 2., 2., 2., 2., 2., 2., 2., 2., 2.],
       [7., 7., 7., 7., 7., 7., 7., 7., 7., 7.]])
```

Ayudas para indexacion

La indexación de una matriz 2d puede ser un poco confusa al principio, especialmente cuando comienza a agregar pasos en la seleccion. Pruebe buscar imagenes en google con la parala NumPy indexing y encontrara ejemplos utiles como:

Seleccion basados en operadores de comparacion

```
# creacion de un arreglo de enteros desde 0 hasta 10
arr = np.arange(1,11) # recordar que el ultimo valor no se incluye
arr
```

```
array([ 1,  2,  3,  4,  5,  6,  7,  8,  9, 10])
```

```
arr > 4 # determinar cuales valores del arreglo son mayores que 4
# el resultado es un arreglo de Booleans
```

```
array([False, False, False, False,  True,  True,  True,  True,  True,
        True])
```

```
bool_arr = arr>4 # creacion de arreglo de booleans
```

```
bool_arr
```

```
array([False, False, False, False,  True,  True,  True,  True,  True,
        True])
```

```
# Seleccion de elementos usando un arreglo de Booleans
arr[bool_arr] # Solamente se retornan los elementos en los cuales las posiciones de b
```

```
array([ 5,  6,  7,  8,  9, 10])
```

```
# Se puede hacer esta seleccion mucho mas rapida realizando la comparacion dentro de
arr[arr>2] # Obtener los valores del arreglo mayores que 2
```

```
array([ 3,  4,  5,  6,  7,  8,  9, 10])
```

```
x = 2 # se puede hacer usando una variable
arr[arr>x]
```

```
array([ 3,  4,  5,  6,  7,  8,  9, 10])
```

Concatenacion

```
# En una dimension
x = np.array([1, 2, 3]) # Vector de valores
y = np.array([3, 2, 1]) # Vector de valores
np.concatenate([x, y]) # Concatenacion
```

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

```
# En dos dimensiones de forma vertical
grid = np.array([[1, 2, 3],[4, 5, 6]])
np.concatenate([grid, grid])
```

```
array([[1, 2, 3],
       [4, 5, 6],
       [1, 2, 3],
       [4, 5, 6]])
```

```
# En dos dimensiones de forma horizontal
grid = np.array([[1, 2, 3],[4, 5, 6]])
np.concatenate([grid, grid],axis=1)
```

```
array([[1, 2, 3, 1, 2, 3],
       [4, 5, 6, 4, 5, 6]])
```

Operaciones con NumPy

Con las listas de python no se pueden realizar operaciones elemento a elemento, pero con NumPy si se puede realizar.

Como es el comportamiento de las listas ante las siguientres operaciones:

```
# observar el comportamiento de las listas
lista = list(range(10))
lista
```

```
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
```

```
lista + lista # este procedimiento lo que hace es concatenar las listas
```

```
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
```

```
x = 2
lista*x # este procedimiento hace que se repita la lista x numero de veces
```

```
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
```

Si quiero realizar operaciones vectoriales o con matrices se realizan con arreglos NumPy

Aritmetica

Puede realizar fácilmente aritmética de matriz a matriz o aritmética de escalar con matriz.

```
import numpy as np # importar la libreria de NumPy
arr = np.arange(0,10) # crear un arreglo de 10 elementos
arr
```

```
array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9])
```

```
arr + arr # Suma elemento a elemento de los arreglos
```

```
array([ 0,  2,  4,  6,  8, 10, 12, 14, 16, 18])
```

```
# Suma de valores uno a uno con arreglos
aa = np.arange(5)
bb = np.arange(10) # este arreglo es mas grande que el anterios
aa+bb[:5] # deben tener el mismo tamaño para realizar la suma
```

```
array([0, 2, 4, 6, 8])
```

```
arr * arr # Multiplicacion elemento a elemento
```

```
array([ 0,  1,  4,  9, 16, 25, 36, 49, 64, 81])
```

```
arr - arr # Resta elemento a elemento
```

```
array([0, 0, 0, 0, 0, 0, 0, 0, 0, 0])
```

```
# Mensaje de advertencia si se presenta una division por cero, pero no hay error!
# solo se reemplazo por el valor nan
arr/arr # division elemento a elemento
```

```
array([nan,  1.,  1.,  1.,  1.,  1.,  1.,  1.,  1.,  1.])
```

```
# Tambien genera una advertencia, pero no hay error lo que se genera es un infinity
# Observar el primer elemento del arreglo
1/arr
```

```
array([       inf, 1.        , 0.5       , 0.33333333, 0.25      ,
       0.2       , 0.16666667, 0.14285714, 0.125     , 0.11111111])
```

```
arr**3 # eleva al cubo cada elemento del arreglo
```

```
array([  0,   1,   8,  27,  64, 125, 216, 343, 512, 729])
```

Funciones Universales de los arreglos

NumPy tiene integrado muchas funciones universales, que son esencialmente solo operaciones matemáticas que se pueden usar para realizar la operación en todo el arreglo. Las mas importantes son:

- Matematicas
- Trigonometria
- Funciones de comparacion
- Funciones de punto flotante

Algunos ejemplos son:

```
#Calcular la raiz cuadrada de cada elemento
np.sqrt(arr)
```

```
array([0.        , 1.        , 1.41421356, 1.73205081, 2.        ,
       2.23606798, 2.44948974, 2.64575131, 2.82842712, 3.        ])
```

```
#Calcular el exponencial (e^) de cada elemento
np.exp(arr)
```

```
array([1.00000000e+00, 2.71828183e+00, 7.38905610e+00, 2.00855369e+01,
       5.45981500e+01, 1.48413159e+02, 4.03428793e+02, 1.09663316e+03,
       2.98095799e+03, 8.10308393e+03])
```

```
# Obtener el valor maximo como una funcion
np.max(arr) #lo mismo arr.max()
```

```
9
```

```
# Calcular el Sin de cada elemento
np.sin(arr)
```

```
array([ 0.        ,  0.84147098,  0.90929743,  0.14112001, -0.7568025 ,
       -0.95892427, -0.2794155 ,  0.6569866 ,  0.98935825,  0.41211849])
```

```
# Calcular el Logaritmo de cada elemento
np.log(arr)
```

```
array([      -inf, 0.        , 0.69314718, 1.09861229, 1.38629436,
       1.60943791, 1.79175947, 1.94591015, 2.07944154, 2.19722458])
```

```
# Calcular el valor absoluto
np.abs(arr)
```

```
array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9])
```

Estadistica

Mas funciones en:

https://docs.scipy.org/doc/numpy/reference/routines.statistics.html

```
# Desviacion estandar
np.std(arr)
```

```
2.8722813232690143
```

```
# Promedio de los valores
np.mean(arr)
```

```
4.5
```

```
# Media
np.median(arr)
```

```
4.5
```

```
# Varianza
np.var(arr)
```

```
8.25
```

Operaciones de Matrices

```
#creacion de una matriz de ejemplo
arr_2d = np.array(([5,10,15],[20,25,30],[35,40,45]))
arr_2d
```

```
array([[ 5, 10, 15],
       [20, 25, 30],
       [35, 40, 45]])
```

```
# transpuesta de una matriz
arr_2d.T # tambien puede ser arr_2d.transpose()
```

```
array([[ 5, 20, 35],
       [10, 25, 40],
       [15, 30, 45]])
```

```
# Multiplicacion de matrices
a = np.array([1,4,3]) # vector = arreglo de 1 dimension
b = np.array([2,-1,5]) # vector = arreglo de 1 dimension
a@b
```

```
13
```

```
# Multiplicacion de matrices
# Producto Punto
a = np.array(([2,0,1],[3,0,0],[5,1,1]))
b = np.array(([1,0,1],[1,2,1],[1,1,0]))
a@b
```

```
array([[3, 1, 2],
       [3, 0, 3],
       [7, 3, 6]])
```

```
#Tambien puede ser de esta forma
np.dot(a,b)
```

```
array([[3, 1, 2],
       [3, 0, 3],
       [7, 3, 6]])
```

```
# Producto Cruz
a = np.array([1,4,3]) # vector = arreglo de 1 dimension
b = np.array([2,-1,5]) # vector = arreglo de 1 dimensi
np.cross(a,b)
```

```
array([23,  1, -9])
```

Referencias

- Tutorial oficial de Numpy
- NumPy Cheatsheet
- Python - Linear Algebra - cheatsheet
- Numpy for Matlab Users
- Funciones universales

Phd. Jose R. Zapata

- https://joserzapata.github.io/
- https://twitter.com/joserzapata

Last updated on Sep 9, 2020