Nombre de la práctica	Introducción a Numpy			No.	1
Asignatura:	Simulación	Carrera:	Ingeniería en sistemas	Duración de la práctica (Hrs)	

Nombre del alumno: Jesús Navarrete Martínez
Grupo: 3501

I. Competencia(s) específica(s):

II. Lugar de realización de la práctica (laboratorio, taller, aula u otro): Aula

III. Material empleado:

- Equipo de computo

IV. Desarrollo de la práctica:

Introducción a Numpy

Numpy es una librería para la computación con python.

- Proporciona Arrays N-Dimensionales.
- Implementa funciones matemáticas sofisticadas
- proporciona herramientas para integrar C/C++ y Fortran.
- Proporciona mecanismos para facilitar la realización de las tareas relacionadas con algebra lineal o números aleatorios

Imports

```
import numpy as np
```

Arrays

Un array es una estructura de datos que consiste en una colección de elementos (valores o variables), cada uno identificado por al menos un índice o clave. Un array se almacena de modo que la posición de cada elemento se pueda calcular a partir de su tupla de índice, mediante una formula matemática. El tipo más simple de array es un array lineal también llamado array unidimensional

En Numpy:

- Cada dimensión se denomina axis
- El número de dimensiones se denomina rank
- La lista de dimensiones con su correspondiente longitud se denomina shape

- El número total de elementos (multiplicación de la longitud de las dimensiones) a esto se denomina size

```
# Array cuyos valores son todos 0.
a= np.zeros((2, 4))
a
```

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

a es un array:

- con dos axis, el primero de longitud 2 y el segundo de longitud 4.
- Con un rank igual a 2
- Con un shape igual a (2,4)
- Con un size igual a 8

```
a.shape
```

```
(2, 4)
```

```
a.ndim
```

```
2
```

```
a.size
```

```
8
```

Creación de Arrays

```
# Array cuyos valores son todos 0
np.zeros((2,3,4))
```

```
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.]]])
```

```
# Array cuyos valores son todos 1
np.ones((2,3,4))
```

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

       [[1., 1., 1., 1.],
        [1., 1., 1., 1.],
        [1., 1., 1., 1.]]])
```

```
# Array cuyos valores son todos el valor indicado como segundo
parametro de la funcion
np.full((2,3,4),8)
```

```
array([[[8, 8, 8, 8],
        [8, 8, 8, 8],
        [8, 8, 8, 8]],

       [[8, 8, 8, 8],
        [8, 8, 8, 8],
        [8, 8, 8, 8]]])
```

```
# El resultado de np.empty no es predecible
# Se inicializa con los valores del array con lo que haya en memoria
en ese momento
np.empty((2,3,9))
```

```
array([[[ 8.62464756e-317,  0.00000000e+000,  3.77172086e-317,
          4.03182258e-246,  6.92342677e-310,  3.77170505e-317,
```

```
         -2.52548782e+018,  6.92342811e-310,  6.92342677e-310],
        [ 4.36125427e+043,  6.92342677e-310,  6.92342676e-310,
         -7.87637039e+214,  6.92342677e-310,  3.77170505e-317,
          2.89027741e+011,  6.92342677e-310,  6.92342915e-310],
        [-3.28186256e-094,  6.92342677e-310,  3.77172086e-317,
         -4.35581174e+261,  6.92342679e-310,  6.92342677e-310,
         -3.25367713e-011,  6.92342677e-310,  3.77170505e-317]],

       [[-1.63573452e+218,  6.92342677e-310,  6.92342677e-310,
          3.31505661e-302,  6.92342677e-310,  8.89468358e-317,
          3.61250624e+122,  6.92342677e-310,  6.92342677e-310],
        [-9.13951861e-256,  6.92342677e-310,  6.92342677e-310,
          1.09903699e+297,  6.92342677e-310,  3.77170505e-317,
          5.87942096e+243,  6.92342677e-310,  3.77170505e-317],
        [ 2.96420853e-198,  6.92342677e-310,  6.92342677e-310,
          2.18959932e-280,  6.92342679e-310,  9.14751081e-317,
          1.28653140e-252,  6.92342674e-310,  9.23312250e-317]]])
```

```
#Inicializar el array utilizando un array de Python
b=np.array([[1,2,3], [4,5,6]])
b
```

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

```
b.shape
```

```
(2, 3)
```

```
# Crear un array utilizando una función basada en rangos
#(minimo,maximo,numero elementos del array)
print(np.linspace(0,6,10))
```

```
[0.         0.66666667 1.33333333 2.         2.66666667 3.33333333
 4.         4.66666667 5.33333333 6.        ]
```

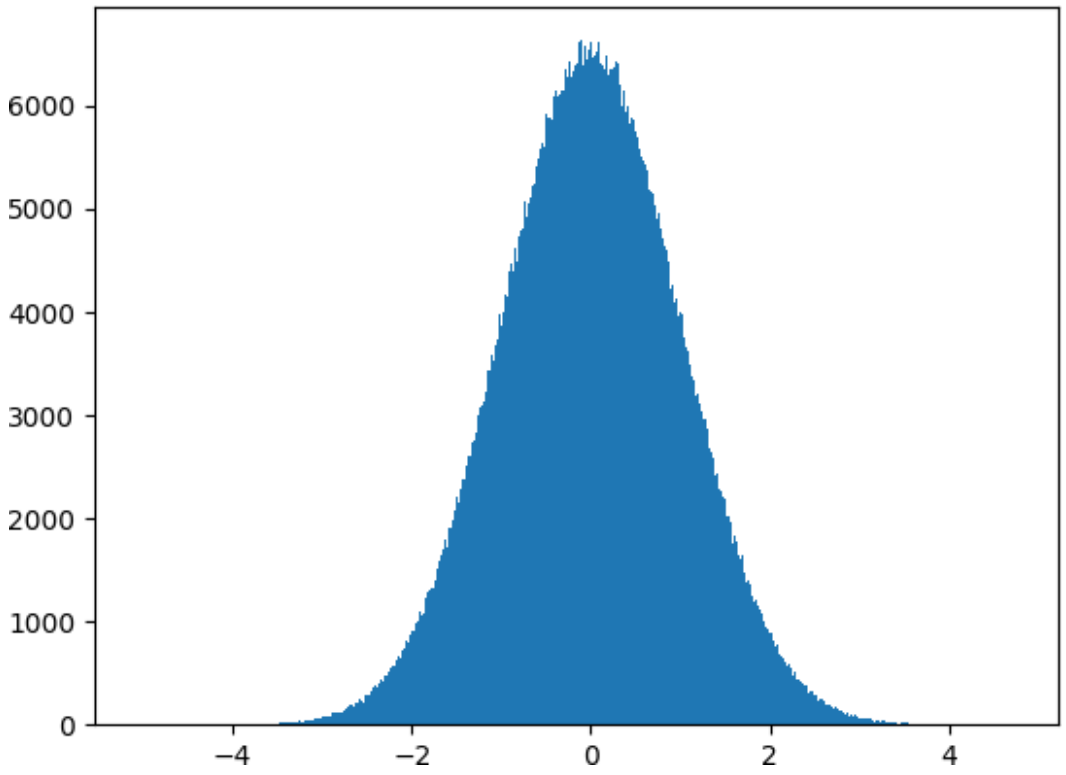
```
# Inicializar el array con valores aleatorios.
np.random.rand(2,3,4)
```

```
array([[[0.17668215, 0.8620062 , 0.3465681 , 0.41524337],
        [0.932917  , 0.77654134, 0.47804248, 0.06130126],
        [0.64987832, 0.76048597, 0.30931948, 0.84286252]],

       [[0.75708896, 0.67669384, 0.50897835, 0.92030351],
        [0.82101019, 0.29045264, 0.18512181, 0.22269108],
        [0.27419697, 0.65941558, 0.34387209, 0.53353637]]])
```

```
# Iniciar arreglo con valores aleatorios con forme a una distribución
normal.
np.random.randn(2,4)
```

```
array([[ 0.48785911, -1.02151676,  1.614854  , -0.17230471],
       [-0.355344  ,  0.41953501,  0.91568087, -0.94567872]])
```

```
%matplotlib inline
import matplotlib.pyplot as plt

c=np.random.randn(1000000)
plt.hist(c,bins=600)
plt.show()
```




```
# Inicializar un array . utilizando una función personalizada

def func(x,y):
    return x + 2 * y

np.fromfunction(func, (3,5))
```

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

```
# Acceder a los elementos de un array.
array_uni  = np.array([1,3,5,7,9,11])
print ("Shape:", array_uni.shape)
print ("Array_uni", array_uni)
```

```
Shape: (6,)
Array_uni [ 1  3  5  7  9 11]

# Accediendo al quinto elemento del array
array_uni[4]

np.int64(9)

#Acceder al tercer y cuarto elemento del array
array_uni[2:4]

array([5, 7])
```

Array Multidimensional

```
# crear un array multidimensional
array_multi=np.array([[1,2,3,4],[5,6,7,8]])
print("Shape:", array_multi.shape)
print("Array_multi:\n", array_multi)

Shape: (2, 4)
Array_multi:
 [[1 2 3 4]
 [5 6 7 8]]

# Acceder al cuarto elemento del array
array_multi[0,3]

np.int64(4)

# Acceder a la segunda fila del array
array_multi[1, :]

array([5, 6, 7, 8])

# Acceder al primer elemento de las dos primeras filas del array
array_multi[0:2, 2]

array([3, 7])
```

Modificación de un array

```
# Crear un arreglo unidimensional e inicializarlo con un rango de
elememento del 0 al 27
array1= np.arange(28)
print("Shape:", array1.shape)
print("Array:\n", array1)

Shape: (28,)
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 25 26 27]
```

```
# Cambiar las dimensiones del array y sus longitudes.
array1.shape=(7,4)
print("Shape:", array1.shape)
print("Array:\n", array1)
```

```
Shape: (7, 4)
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 25 26 27]]
```

```
# El ejemplo anterior devuelve un nuevo array que apunta a los mismos
datos
#Nota: modificaciones en el array, modificaran el otro array.
array2= array1.reshape(4,7)
print("Shape:", array2.shape)
print("Array:\n", array2)
```

```
Shape: (4, 7)
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 25 26 27]]
```

```
# Modificación del nuevo array devuelto
array2[1,3]=30
print("Array:\n", array2)
```

```
Array:
 [[ 0  1  2  3  4  5  6]
 [ 7  8  9 30 11 12 13]
 [14 15 16 17 18 19 20]
 [21 22 23 24 25 26 27]]
```

```
print("Array1:\n", array1)
```

```
Array1:
 [[ 0  1  2  3]
 [ 4  5  6  7]
 [ 8  9 30 11]
 [12 13 14 15]
 [16 17 18 19]
```

```
 [20 21 22 23]
 [24 25 26 27]]

# Devolver el array a su estado original
print("Array1:", array1.ravel())

Array1: [ 0  1  2  3  4  5  6  7  8  9 30 11 12 13 14 15 16 17 18 19
20 21 22 23
 24 25 26 27]
```

Operaciones aritméticas con Arrays

```
array1=np.arange(2,18,2)
array2=np.arange(8)
print("Array1:", array1)
print("Array2:", array2)

Array1: [ 2  4  6  8 10 12 14 16]
Array2: [0 1 2 3 4 5 6 7]

# suma
print(array1  +  array2)

[ 2  5  8 11 14 17 20 23]

# Resta
print(array1-array2)

[2 3 4 5 6 7 8 9]

# Multiplicacion
# Nota: no es una multiplicacion de matrices
print(array1*array2)

[  0   4  12  24  40  60  84 112]
```

Broadcasting

Si se aplican operaciones aritméticas sobre arrays que no tienen la misma forma (shape), Numpy aplica una propiedad que se llama Broadcasting

```
array1=np.arange(5)
array2= np.array([3])
print("Shape:", array1.shape)
print("Array:\n", array1)
print("\n")
print("Shape:", array2.shape)
print("Array:\n", array2)
```

```
Shape: (5,)
Array:
 [0 1 2 3 4]


Shape: (1,)
Array:
 [3]
```

```
# Suma de ambos arrays
array1 + array2
```

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

```
#Multiplicacion
array1 * array2
```

```
array([ 0,  3,  6,  9, 12])
```

Funciones estadísticas sobre arrays

```
# Creación de un array unidimensional
array1=np.arange(1,20,2)
print("Array:\n", array1)
```

```
Array:
 [ 1  3  5  7  9 11 13 15 17 19]
```

```
# Media de los elementos del array
array1.mean()
```

```
np.float64(10.0)
```

```
# Suma de los elementos del array
array1.sum()
```

```
np.int64(100)
```

Funciones universales proporcionadas por numpy: **ufunc**

```
# Cuadrado de los elementos del array
np.square(array1)
```

```
array([  1,   9,  25,  49,  81, 121, 169, 225, 289, 361])
```

```
# Raiz cuadrada de los elementos del array
np.sqrt(array1)
```

```
array([1.        , 1.73205081, 2.23606798, 2.64575131, 3.        ,
       3.31662479, 3.60555128, 3.87298335, 4.12310563, 4.35889894])

# Exponencial de los elementos del array
np.exp(array1)

array([2.71828183e+00, 2.00855369e+01, 1.48413159e+02, 1.09663316e+03,
       8.10308393e+03, 5.98741417e+04, 4.42413392e+05, 3.26901737e+06,
       2.41549528e+07, 1.78482301e+08])

# log de los elementos del array
np.log(array1)

array([0.        , 1.09861229, 1.60943791, 1.94591015, 2.19722458,
       2.39789527, 2.56494936, 2.7080502 , 2.83321334, 2.94443898])
```

V. Conclusiones:

En esta práctica, hemos explorado los fundamentos de NumPy, una de las bibliotecas más importantes para el manejo de datos numéricos en Python. Hemos aprendido a crear y manipular arrays, que son estructuras clave para almacenar y operar con grandes cantidades de datos de manera eficiente. Además, realizamos operaciones matemáticas básicas que ilustran cómo NumPy simplifica y acelera los cálculos en comparación con listas estándar de Python.

Esta introducción nos permite sentar las bases para aplicar NumPy en análisis de datos, donde el manejo eficiente de matrices y datos multidimensionales es crucial.