

Nombre de la práctica	NUMPY			No.	
Asignatura:	Simulación	Carrera :	Inf. Sistemas Computacionales	Duración de la práctica (Hrs)	

Nombre: Ana Edith Hernández Hernández

## I. Competencia(s) específica(s):

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

## III. Material empleado:

Laptop

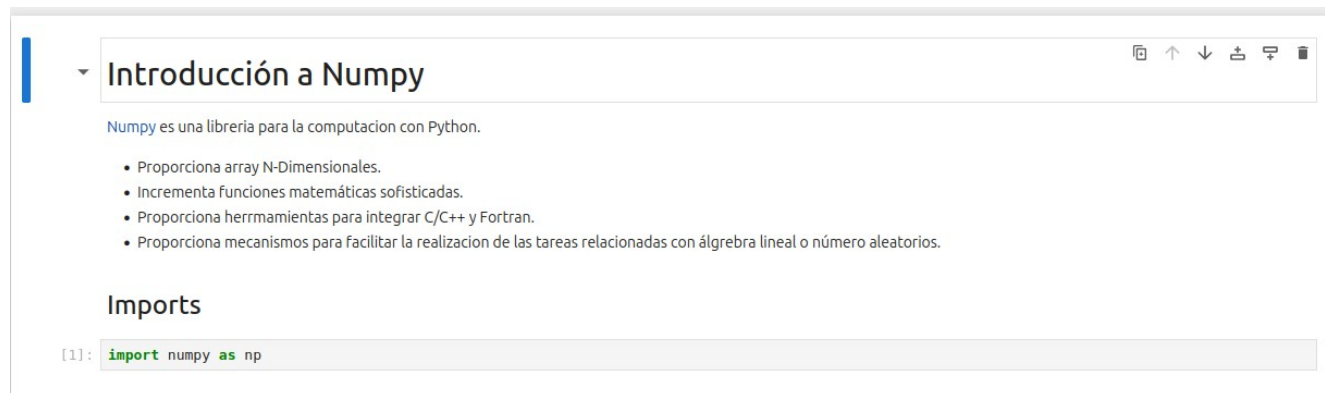
Anaconda

## IV. Desarrollo de la práctica:

### Numpy

Primero comenzamos con una pequeña introducción a Numpy.

Se pone un acceso directo a la documentación de Numpy y se hace la importación de este para poder usarlo en el documento.



Después se da una definición de que es un array y los tipos de array en Numpy.

## ▼ Array

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 índices mediante una fórmula 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**.

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

```
[2]: 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** = a 2.
- Con un **shape** = a (2,4).
- Con un **size** = a 8.

```
[3]: a.shape
```

```
[3]: (2, 4)
```

```
[4]: a.ndim
```

```
[4]: 2
```

```
[5]: a.size
```

```
[5]: 8
```

Comenzamos a crearlos y hacer ejemplos, todo esta documentado:

## Creación de Arrays

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

```
[6]: 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.]])
```

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

```
[7]: 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.]])
```

```
[8]: # Array cuyos valores son todos el valor indicado como segundo parámetro de la función.  
np.full((2, 3, 4), 8)
```

```
[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.]])
```



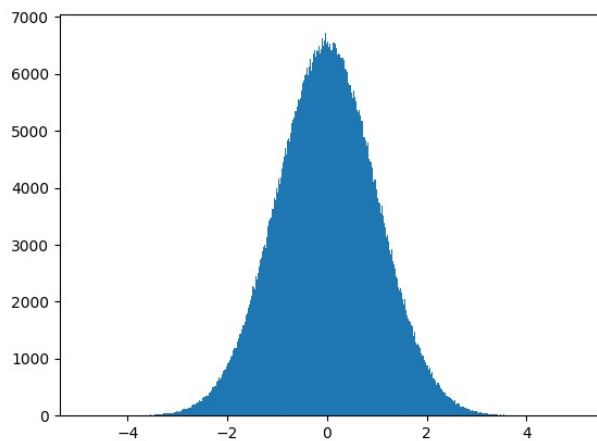
```
[9]: # El resultado de np.empty no es predecible  
# se inicializa con los valores del array con lo que haya de memoria en ese momento.  
np.empty((2, 3, 9))
```

```
[9]: array([[ 2.02778044e-316,  0.00000000e+000,  2.32439981e-316,  
           7.64619881e+787,  7.37478740e-316,  4.04668883e+173
```

```
[14]: # Iniciar array con valores aleatorios conforme a una distribución normal  
np.random.randn(2, 4)
```

```
[14]: array([[ -0.46030319,  0.94457054, -1.1496546,  0.02739483],  
         [ 0.59154036, -0.48067379,  1.0597746, -0.06444422]])
```

```
[20]: %matplotlib inline  
import matplotlib.pyplot as plt  
  
c = np.random.randn(1000000)  
plt.hist(c, bins=600)  
plt.show()
```



```
[13]: array([[1.76546231e-01, 2.77393941e-01, 7.66843746e-01, 2.84033725e-01],  
          [1.31665545e-01, 6.94822027e-01, 5.35616925e-01, 9.63463357e-01],  
          [7.18756993e-01, 9.40941428e-01, 8.51330681e-01, 5.15631858e-01]])
```

Utilizamos matplotlib para hacer gráficos, también se hace una importación de esta.

## ACCEDER A ARRAYS

### Acceso a los elementos de un array

#### Array unidimensional

```
[24]: # 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]
```

```
[33]: # Accediendo a la quinto elemeto del array.  
array_uni[4]
```

```
[33]: 9
```

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

```
[26]: array([5, 7])
```

#### Array multidimensional

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

```
[30]: # Acceder al cuarto elemnto del array  
array_multi[0, 3]
```

```
[30]: 4
```

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

```
[31]: array([5, 6, 7, 8])
```

```
[32]: # Accede al primer elemnto de las dos filas primera filas de array.  
array_multi[0:2, 2]
```

```
[32]: array([3, 7])
```

## MODIFICACIÓN DE ARRAYS

### Modificación de un array

```
[36]: # 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]]

[ ]: # Modificacion del nuevo array devuelto
array2[1, 3] = 30
print("Shape:", array2.shape)
print("Array:\n", array2)

[38]: 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]]

[39]: # 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]

[ 7  8  9 10 11 12 13]
[14 15 16 17 18 19 20]
[21 22 23 24 25 26 27]]

[ ]: # Modificacion del nuevo array devuelto
array2[1, 3] = 30
print("Shape:", array2.shape)
print("Array:\n", array2)

[38]: print("Array1:\n", array1)

Array1:
```

## OPERACIONES ARITMÉTICAS

### Operaciones Aritméticas con Arrays

```
[40]: 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]

[41]: # Suma
print(array1 + array2)

[ 2  5  8 11 14 17 20 23]

[42]: # Resta
print(array1 - array2)

[2  3  4  5  6  7  8  9]

[43]: # Multiplicacion.
# No es una multiplicacin de matrices.
print(array1 * array2)

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



## BROADCASTING

### Broadcastingb

Si se aplican operaciones aritmeticas sobre array que no tienen la misma forma (shape), numpy aplica una propiedad que se llama Broadcasting

```
[45]: array1 = np.arange(5)
      array2 = np.array([3])
      print("Shape:", array1.shape)
      print("Array1:\n", array1)
      print("\n")
      print("Shape:", array2.shape)
      print("Array2:\n", array2)
```

```
Shape: (5,)
Array1:
[0 1 2 3 4]
```

```
Shape: (1,)
Array2:
[3]
```

```
[46]: # Suma de ambos arrays
      array1 + array2
```

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

```
[47]: # Multiplicacion
      array1 * array2
```

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

## FUNCIONES ESTADÍSTICAS SOBRE ARRAYS

### Funciones estadísticas sobre arrays

```
[49]: # Creacion de un array unidimensional
array1 = np.arange(1, 20, 2)
print("Array1:\n", array1)

Array1:
[ 1  3  5  7  9 11 13 15 17 19]
```

```
[50]: #Media de los elemntos del array
array1.mean()
```

```
[50]: 10.0
```

```
[51]: # Suma de los elemntos del array
array1.sum()
```

### Funciones universales proporcionada por numpy: ufunc.

```
[52]: # Cuadrado de los elementos
np.square(array1)
```

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

```
[53]: # Raiz cuadrada de los elementos del array.
np.sqrt(array1)
```

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

```
[54]: # Exponencial de los elemntos del array
np.exp(array1)
```

```
[54]: 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])
```

```
[55]: # Logaritmo natural de los elementos de array
np.log(array1)
```

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

```
[ ]:
```

## UFUNC

### Funciones universales proporcionada por numpy: ufunc.

```
[52]: # Cuadrado de los elementos
np.square(array1)
```

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

```
[53]: # Raiz cuadrada de los elementos del array.
np.sqrt(array1)
```

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

```
[54]: # Exponencial de los elemntos del array
np.exp(array1)
```

```
[54]: 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])
```

```
[55]: # Logaritmo natural de los elementos de array
np.log(array1)
```

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

```
[ ]:
```



## V. Conclusiones:

NumPy es una biblioteca fundamental en el ecosistema de Python para la ciencia de datos y la computación numérica. Su capacidad para manejar grandes volúmenes de datos y realizar operaciones matemáticas complejas de manera eficiente hace que sea una herramienta indispensable para desarrolladores, investigadores y científicos. La implementación de arreglos multidimensionales y funciones matemáticas optimizadas permite a los usuarios realizar cálculos precisos y rápidos, facilitando el desarrollo de algoritmos y la manipulación de datos.