Nombre de la práctica	Numpy			No.	1
Asignatura:	SIMULACION	Carrera:	INGENIERÍA EN SISTEMAS COMPUTACIONALES	Duración de la práctica (Hrs)	5 horas

NOMBRE DEL ALUMNO: Francisco David Colin Lira
GRUPO: 3502

I. Competencia(s) específica(s):

Analiza, modela, desarrolla y experimenta sistemas productivos y de servicios, reales o hipotéticos, a través de la simulación de eventos discretos, para dar servicio al usuario que necesite tomar decisiones, con el fin de describir con claridad su funcionamiento, aplicando herramientas matemáticas.

Encuadre con CACEI: Registra el (los) atributo(s) de egreso y los criterios de desempeño que se evaluarán en esta práctica.

Encuadre con CACEI

No. atributo	Atributos de egreso del PE que impactan en la asignatura	No.Criterio	Criterios de desempeño	No. Indicador	Indicadores
1	El estudiante identificará los principios de las ciencias básicas para la resolución de problemas prácticos de ingeniería	CD1	Propone alternativas de solucion	I1	diseño algoritmico
				I2	empleo de formulas y funciones
2	el estudiante diseñará esquemas de trabajo y procesos, usando metodologías congruentes en la resolución de problemas de ingeniería en sistemas computacionales	CD1	identifica metodologías y procesos empleados en la resolución de problemas	I1	identificacion y reconocimiento de distintas metodologias para la resolucion de problemas.
				I2	Manejo de procesos especificos en la solucion de problemas y/o deteccion de necesidades.
		CD2	diseña soluciones a problemas, empleando metodologías apropiadas al area	I1	uso de metodologias para el modelado de la solucion de sistemas y aplicaciones
				I2	diseño algoritmico.

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

Aula y equipo de cómputo personal.

III. Material empleado:

- Equipo de cómputo
- Jupyter Notebook
- Anaconda

IV. Desarrollo de la práctica:

Introduccion a Numpy ¶

Numpy es una bliblioteca fundamental para la cumputación cientifica con Python.

- Proporciona arrays N-dimensiones.
- Proporciona funciones matematticas sofesticadas.
- Implementa herramientas para integrar C/C++.
- Proporciona mecanismos para facilitar la realizacion de 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 coleccion de elementos (valores o variables), cada uno identifcado por almenos un indece o clave. Un **array** se almacena de modo que la posicion de cada elemento se pueda calcular a partir de su tupla de indice mediante uan fórmula matemática. El tipo mas simple de array es un lineal, tambien llamado array unidimensional.

En Numpy:

- Cada dimension de denominda **axis**.
- El numero de dimensiones de denomina **rank**.
- La lista de dimensiones con su correspondiente longitud se denomina **shape**.
- El numero total de elementos (multiplicacion de la longitud de las dimensiones) 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**, elprimero 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 indicados
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]]])
```

```
# Creacion de un array con valores de todo 0
np.empty((2, 3, 8))
```

```
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.]]])
```

```
# Inicializacion del 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)
```

```
# Creacion de una funcion basada en rangos
# (min, max, numero de 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 un array con valores alatorios
np.random.rand(2, 3, 4)
```

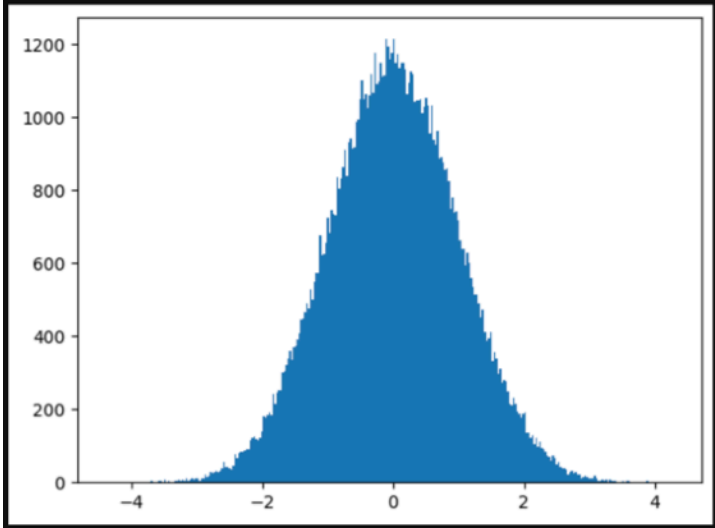
```
array([[[0.70618401, 0.63012869, 0.26600066, 0.81862382],
        [0.56808865, 0.22417943, 0.27786849, 0.40124984],
        [0.44799766, 0.09633236, 0.23069128, 0.75430842]],

       [[0.2781372 , 0.84543085, 0.51604424, 0.69952681],
        [0.72868086, 0.36581381, 0.05694011, 0.96308034],
        [0.12378207, 0.62626852, 0.30993939, 0.95709319]]])
```

```
# Inicializar un array con valores aleatorios conforme a una distrubucion normal
np.random.randn(2, 4)
```

```
array([[ 0.7229525 ,  0.94973779, -0.31422974, -0.31417679],
       [-0.16922654,  2.89466144,  0.73598613, -0.44870157]])
```

```
%matplotlib inline
import matplotlib.pyplot as plt
c = np.random.randn(100000)
plt.hist(c, bins=300)
plt.show()
```




```
# Inicializar un array utilizando una funcion 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.]])
```

Acceso a los elementos de un array

Array unidimensional

```
# Creación de un arrray unidimensional
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 5to elemento del array
array_uni[4]
```

```
np.int64(9)
```

```
# Accediendo al 3ro y 4to elemento del array
array_uni[2:4]
```

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

```
# Acceder a los elementos 0, 3, 5 del array
array_uni[0::5]
```

```
array([ 1, 11])
```

Array multidimensional

```
# Creacion de array multidimensional
array_multi = np.array([[1, 2 ,3 ,4], [5, 6, 7 ,8]])
print("shape: ", array_multi.shape)
print("array_multi:", array_multi)
```

```
shape:  (2, 4)
array_multi: [[1 2 3 4]
 [5 6 7 8]]
```

```
# Accediendo al 4to elemento del array
array_multi[0, 3]
```

```
np.int64(4)
```

```
# Accediendo a la segunda fila del array
# el : sirve para recorrer todos los elementos del array
array_multi[1, :]
```

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

```
# Accediendo al 3ro elemento de las 2 primeras filas del array
array_multi[0: 2, 2]
```

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

Modificacion de un Array

```
# Creacion de un aaray unidimensional inicializando el rango con el rango de elementos que va del 0 al 27
array1 = np.arange(28)
print("shape: ", array1.shape)
print("array 1: \n", array1)
```

```
shape:  (28,)
array 1:
 [ 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 longitudes
array1.shape = (7, 4)
print("shape: ", array1.shape)
print("array 1: \n", array1)
```

```
shape:  (7, 4)
array 1:
 [[ 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: las modificaciones en un array, modifican el otro array
array2= array1.reshape(4, 7)
print("shape: ", array2.shape)
print("array 2: \n", array2)
```

```
shape:  (4, 7)
array 2:
 [[ 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 array devuelto
array2[0,3]= 20
print("Array 2: \n", array2)
```

```
Array 2:
 [[ 0  1  2 20  4  5  6]
 [ 7  8  9 10 11 12 13]
 [14 15 16 17 18 19 20]
 [21 22 23 24 25 26 27]]
```

```
print("Array 1: \n", array1)
```

```
Array 1:
 [[ 0  1  2 20]
 [ 4  5  6  7]
 [ 8  9 10 11]
 [12 13 14 15]
 [16 17 18 19]
 [20 21 22 23]
 [24 25 26 27]]
```

```
# Devuelve el array a una sola dimension
# NOTA: el nuevo array apunta a los mismos datos
print("Array 1: \n", array1.ravel())
```

```
Array 1:
 [ 0  1  2 20  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20 21 22 23
 24 25 26 27]
```

Operaciones aritméticas con arrays

```
# Creacion de 2 arrays unidimensionales.
array1= np.arange(2, 18, 2)
array2= np.arange(8)
print("Array 1: \n", array1)
print("Array 2: \n", array2)
```

```
Array 1:
 [ 2  4  6  8 10 12 14 16]
Array 2:
 [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 aritmeticas sobre arrays que no tienen la misma forma (shape), numpy aplica una propiedad que se denomina Broadcasting

```
# Creacion de 2 arrays unidimensionales
array1= np.arange(5)
array2= np.array([3])
print("shape: ", array1.shape)
print("array 1: \n", array1)
print("shape: ", array2.shape)
print("array 2: \n", array2)
```

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

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

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

```
# Creacion de 2 arrays multidimensionales y unidimensional
array1 = np.arange(6)
array1.shape = (2, 3)
array2 = np.arange(6,18, 4)
print("shape: ", array1.shape)
print("array 1: \n", array1)
print("shape: ", array2.shape)
print("array 2: \n", array2)
```

```
shape:  (2, 3)
array 1:
 [[0 1 2]
 [3 4 5]]
shape:  (3,)
array 2:
 [ 6 10 14]
```

```
array1 + array2
```

```
array([[ 6, 11, 16],
       [ 9, 14, 19]])
```

Funciones estadisticas sobre arrays

```
# Creacion de un arryas unidimensional
array1 = np.arange(1, 20, 2)
print("array 11: \n", array1)
```

```
array 11:
 [ 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 eficientes proporcionadas por numpy: **ufunc**

```
# Cuadrado de los elementos de un 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])
```

```
# Logaritmo natural 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:

La verdad comprendí que el uso de Python con esta herramienta como es Numpy es útil para realizar operaciones en array de una dimensión o multidimensional y es rápido y eficaz, ya que en otros lenguajes de programación replicar esto conlleva bastante trabajo, lo principal por que no existe bibliotecas para realizar las operaciones y uno las tendría que programar pero eso no asegura que es resultado sea exitoso o eficaz como lo es Python con Numpy, ademas comprender como se denominan las partes de un array en Numpy es creo parte fundamental ya que con eso se puede realizar operaciones sobre este mismo