Introducción a NumPy

Este Práctico proporciona una introducción básica a NumPy, destacando algunas de sus características clave y mostrando ejemplos de su uso. A continuación tendrán un resúmen de algunos comando y su implementación. Esto servirá de guia para desarrollar las actividades propuestas a continuación.

Comparación entre uso de vectores de Python y NumPy

A continuación se presentan varios aspectos para poder comparar la implementación de vectores a través del tipo de datos LISTA , comparado con la implementación de vectores con NumPy:

1- Creación de vectores

En esta celda, se importa la biblioteca NumPy y se **CREA UN VECTOR** utilizando Python puro y NumPy. Luego, se imprime cada uno de los vectores.

```
import numpy as np
vector_py = [1, 2, 3, 4, 5]
vector_np = np.array([1, 2, 3.5, 4, 5])
print("Vector en Python puro:", vector_py)
print("Vector en NumPy:", vector_np)
```

```
Vector en Python puro: [1, 2, 3, 4, 5]
Vector en NumPy: [1.  2.  3.5 4.  5. ]
```

2- Operaciones matemáticas

En esta celda, se realizan operaciones matemáticas en un vector utilizando Python puro y NumPy. Luego, se imprime el resultado de cada operación.

```
vector_py = [1, 2, 3, 4, 5]
vector_np = np.array([1, 2, 3, 4, 5])
suma_py = [x + 2 for x in vector_py]
suma_np = vector_np *2
#print("Suma en Python puro:", suma_py)
#print("Suma en NumPy:", suma_np)
vector_py*2
```

```
[1, 2, 3, 4, 5, 1, 2, 3, 4, 5]
```

3- Indexado y segmentación

En esta celda, se realiza el indexado y segmentación en un vector utilizando Python puro y NumPy. Luego, se imprime el resultado de cada operación.

```
vector_py = [1, 2, 3, 4, 5]
vector_np = np.array([1, 2, 3, 4, 5])
segmento_py = vector_py[1:4]
segmento_np = vector_np[1:4]
print("Segmento en Python puro:", segmento_py)
print("Segmento en NumPy:", segmento_np)
```

```
Segmento en Python puro: [2, 3, 4]
Segmento en NumPy: [2 3 4]
```

Operaciones entre vectores y funciones sobre vectores

A continuación se presentan algunos ejemplos de operaciones básicas entre vectores utilizando NumPy:

Suma de vectores

En esta celda, se crea un vector 'vector_a' y un vector 'vector_b', y se realiza la suma de ambos vectores utilizando NumPy. Luego, se imprime el resultado.

```
vector_a = np.array([1, 2, 3])
vector_b = np.array([4, 5, 6])
```

```
suma_vector = vector_a + vector_b
print("Suma de vectores:", suma_vector)
```

```
Suma de vectores: [5 7 9]
```

EJERCICIOS PARTE 1

EJERCICIO 1:

Dada la siguiente tabla (matriz) de datos, donde cada fila representa la cantidad que se vendio en referencia a un solo producto durante toda la semana; mientras que cada columna representa la venta total en un dia de la semana.Obtener la siguiente información detallada:

- Importe total de la venta por cada dia de la semana
- Importe total de la venta por cada producto en la semana
- Importe total de la venta en toda la semana

```
import numpy as np


# Datos de ventas diarias de 4 productos durante una semana

ventas_diarias = np.array(
    #Lun,Mar,Mie,Jue,Vie,Sab,Dom
    [[20, 15, 25, 30, 18, 22, 24],  #Producto A
    [12, 20, 14, 8, 15, 18, 16],    #Producto B
    [35, 28, 32, 30, 26, 24, 30],   #Producto C
    [40, 38, 45, 42, 39, 41, 37]    #Producto D
    ]
)

print("Matriz de ventas diarias:",ventas_diarias)

#==== FORMAS DE RECORRER UNA MATRIZ FIJADA LA COLUMNA J=1
#    Calcula la suma total del dia lunes
suma=0
for i in range(4):
  print(ventas_diarias[i,1])
  suma=suma+ventas_diarias[i,1]
print('la suma total del lunes es:', suma)

#===== IMPLEMENTACIÓN DE UN MÉTODO DE NUMPY
# Sumar las ventas por día (sumar las columnas)
ventas_por_dia = np.sum(ventas_diarias, axis=0)
print("Total de ventas por día:",ventas_por_dia)
#desarrollo
ventas_por_dia = np.sum(ventas_diarias, axis=0)
ventas_por_producto = np.sum(ventas_diarias, axis=1)
venta_total_semana = np.sum(ventas_diarias)
print("Importe total de la venta por día:", ventas_por_dia)
print("Importe total de la venta por producto:", ventas_por_producto)
print("Importe total de la venta en toda la semana:", venta_total_semana)
```

```
Matriz de ventas diarias: [[20 15 25 30 18 22 24]
 [12 20 14  8 15 18 16]
 [35 28 32 30 26 24 30]
 [40 38 45 42 39 41 37]]
15
20
28
38
la suma total del lunes es: 101
Total de ventas por día: [107 101 116 110  98 105 107]
Importe total de la venta por día: [107 101 116 110  98 105 107]
Importe total de la venta por producto: [154 103 205 282]
Importe total de la venta en toda la semana: 744
```

EJERCICIO 2:

Crear un programa donde se le pida al usuario que ingrese la cantidad de elementos de una lista de números reales positivos. Luego Convertir esa lista en un vector de Numpy.

```
cantidad_elementos = int(input("Ingrese la cantidad de elementos: "))
lista_numeros = [int(input(f"Ingrese el elemento {i+1}: ")) for i in range(cantidad_elementos)]
vector_numeros = np.array(lista_numeros)
print("Vector de Numpy:", vector_numeros)
```

```
Ingrese la cantidad de elementos: 6
Ingrese el elemento 1: 33
Ingrese el elemento 2: 22
Ingrese el elemento 3: 11
Ingrese el elemento 4: 66
Ingrese el elemento 5: 44
Ingrese el elemento 6: 9
Vector de Numpy: [33 22 11 66 44  9]
```

Ejercicio 3:

Crear un programa donde el usuario ingrese la cantidad de filas y columnas que tendra una tabla de datos. Luego el programa pedira ingresar los datos de la tabla fila por fila. Todos los datos serán numéricos.

Mostrar la tabla ingresada en formato LISTA de Python, y mostrar la misma tabla en formato array de Numpy.

Solicitar al usuario que ingrese las posiciones de dos filas y realice la suma de las mismas. Mostrar este vector resultado.

```
filas = int(input("Ingrese la cantidad de filas: "))
columnas = int(input("Ingrese la cantidad de columnas: "))

tabla_lista = []
for i in range(filas):
    fila = [int(input(f"Ingrese el dato en la fila {i+1}, columna {j+1}: ")) for j in range(columnas)]
    tabla_lista.append(fila)

tabla_np = np.array(tabla_lista)

print("Tabla en formato de lista:")
for fila in tabla_lista:
    print(fila)

print("Tabla en formato de array de Numpy:")
print(tabla_np)

fila1 = int(input("Ingrese la primera fila a sumar: "))
fila2 = int(input("Ingrese la segunda fila a sumar: "))
suma_filas = tabla_np[fila1] + tabla_np[fila2]
print("Resultado de la suma de filas:", suma_filas)
```

```
Ingrese la cantidad de filas: 5
Ingrese la cantidad de columnas: 4
Ingrese el dato en la fila 1, columna 1: 55
Ingrese el dato en la fila 1, columna 2: 11
Ingrese el dato en la fila 1, columna 3: 22
Ingrese el dato en la fila 1, columna 4: 33
Ingrese el dato en la fila 2, columna 1: 4
Ingrese el dato en la fila 2, columna 2: 9
Ingrese el dato en la fila 2, columna 3: 1
Ingrese el dato en la fila 2, columna 4: 66
Ingrese el dato en la fila 3, columna 1: 33
Ingrese el dato en la fila 3, columna 2: 77
Ingrese el dato en la fila 3, columna 3: 6
Ingrese el dato en la fila 3, columna 4: 2
Ingrese el dato en la fila 4, columna 1: 8
Ingrese el dato en la fila 4, columna 2: 14
Ingrese el dato en la fila 4, columna 3: 23
Ingrese el dato en la fila 4, columna 4: 45
Ingrese el dato en la fila 5, columna 1: 15
Ingrese el dato en la fila 5, columna 2: 78
Ingrese el dato en la fila 5, columna 3: 98
Ingrese el dato en la fila 5, columna 4: 65
Tabla en formato de lista:
[55, 11, 22, 33]
[4, 9, 1, 66]
[33, 77, 6, 2]
[8, 14, 23, 45]
[15, 78, 98, 65]
Tabla en formato de array de Numpy:
[[55 11 22 33]
 [ 4  9  1 66]
 [33 77  6  2]
 [ 8 14 23 45]
 [15 78 98 65]]
Ingrese la primera fila a sumar: 2
Ingrese la segunda fila a sumar: 3
Resultado de la suma de filas: [41 91 29 47]
```

Ejercicio 4:

A continuación se muestran los valores de los siguientes productos:

['arroz', 'harina','fideo','yerba','azucar']=[145.6 , 100 , 89.90 , 700 , 95]

Los valores de estos productos son aproximados de hace dos meses, debido a la inflación y alza de los precios, se vieron afectados de la siguiente manera:

- Producto arroz , harina, azucar duplicaron su precio
- Productos restantes incrementaron en un 75% su precio

Mostrar los datos en forma de vector y actualizar sus precios, de manera que se pueda comparar ambos vectores.

```
productos = ['arroz', 'harina', 'fideo', 'yerba', 'azucar']
valores_iniciales = np.array([145.6, 100, 89.90, 700, 95])
incrementos = np.array([1.75])

valores_actualizados = valores_iniciales * incrementos
print("Valores iniciales:", valores_iniciales)
print("Valores actualizados:", valores_actualizados)
```

```
Valores iniciales: [145.6 100.   89.9 700.   95. ]
Valores actualizados: [ 254.8    175.     157.325 1225.     166.25 ]
```

EJERCICIO 5

Completar la siguiente tabla de comandos y funciones que se utilizarán sobre vectores definidos a través de Numpy

	Comando	operación y funcionalidad	resultado	ejemplo
1	np.array([lista])	crea un vector o table con Numpy	matriz	np.array([1.6, 2, 0, 6.75])
2	np.sqrt(vector)	***	***	np.sqrt(vector_np)
3	np.random.rand(n)	***	***	np.random.rand(5)
4	np.ones((n))	***	***	np.ones((3))
5	np.zeros((n))	***	***	np.zeros((3))
6	np.min(array)	***	valor mínimo	np.min(vector_np)
7	np.max(array)	***	valor máximo	np.max(vector_np)
8	np.where(CONDICIÓN SOBRE EL VECTOR)	***	***	np.where(vector_np>1)
9	np.random.shuffle(MATRIZ)	***	***	VER EJERCICIO PARTE 2
10	array.shape[n], n=0,1	****	***	VER EJERCICIO PARTE 2
11	np.sum(array, axis=n), n=0,1	***	***	VER EJERCICIO PARTE 1
12	np.arange(a, b, p)	***	***	np.arange(0, 10, 0.1) VER EJERCICIOS PARTE 3

Funciones matemáticas sobre vectores

En esta sección, se aplican funciones matemáticas a un vector utilizando Python puro y NumPy. Luego, se imprime el resultado de cada función.

ACLARACIÓN: Estas funciones utilizan funciones y operaciones elementales matemáticas, sobre cada una de las posiciones del vector. Pero en general, se pueden definir funciones matemáticas que relacionan diferentes posiciones de un vector.

```
import math
vector_py = [1, 2, 3, 4, 5]
vector_np = np.array([1, 2, 3, 4, 5])
raiz_cuadrada_py = [math.sqrt(x) for x in vector_py]
vector_cuad=[(x**2) for x in vector_np]
vector_log=[math.log(x) for x in vector_np]
raiz_cuadrada_np = np.sqrt(vector_np)
vector_npLog=np.log(vector_np)
print("Raíz cuadrada en Python puro:", raiz_cuadrada_py)
print("Raíz cuadrada en NumPy:", raiz_cuadrada_np)
print('Vector al cuadrado en Python: ',vector_cuad)
print('logaritmo de un Vector en Python: ',vector_log)
print('logaritmo de un Vector en Numpy: ',vector_npLog)
```

```
Raíz cuadrada en Python puro: [1.0, 1.4142135623730951, 1.7320508075688772, 2.0, 2.23606797749979]
Raíz cuadrada en NumPy: [1.         1.41421356 1.73205081 2.         2.23606798]
Vector al cuadrado en Python:  [1, 4, 9, 16, 25]
logaritmo de un Vector en Python:  [0.0, 0.6931471805599453, 1.0986122886681098, 1.3862943611198906, 1.6094379124341003]
logaritmo de un Vector en Numpy:  [0.         0.69314718 1.09861229 1.38629436 1.60943791]
```

Rendimiento

En esta celda, se mide el rendimiento de operaciones en un vector utilizando Python puro y NumPy. Se imprime el tiempo de ejecución de cada operación.

```
import time
vector_py = [i for i in range(1000000)]
vector_np = np.arange(1000000)
```

```
start_time = time.time()
[x * 2 for x in vector_py]
end_time = time.time()
print("Tiempo en Python puro:", end_time - start_time, "segundos")

start_time = time.time()
end_time = time.time()
print("Tiempo en NumPy:", end_time - start_time, "segundos")
```

```
Tiempo en Python puro: 0.14838314056396484 segundos
Tiempo en NumPy: 5.1975250244140625e-05 segundos
```

Redimensionar un NumPy

A continuación se presentan dos ejemplos de redimensionamiento de un NumPy:

Redimensionamiento 1

En esta celda, se crea un NumPy 'array' y se utiliza la función reshape() de NumPy para redimensionarlo a una forma diferente. Luego, se imprime el nuevo array.

```
array = np.array([1, 2, 3, 4, 5, 6])
nuevo_array = array.reshape((2, 3))
print("Nuevo array redimensionado:", nuevo_array)
```

```
Nuevo array redimensionado: [[1 2 3]
 [4 5 6]]
```

Redimensionamiento 2

En esta celda, se crea un NumPy 'array' y se utiliza la función np.resize() de NumPy para redimensionarlo a una forma diferente. Luego, se imprime el nuevo array.

```
array = np.array([1, 2, 3, 4, 5, 6])
nuevo_array = np.resize(array, (3, 3))
print("Nuevo array redimensionado:", nuevo_array)
```

```
Nuevo array redimensionado: [[1 2 3]
 [4 5 6]
 [1 2 3]]
```

EJERCICIOS PARTE 2

Ejercicio 1:

Dado una matriz de datos, dividir el 70% de filas en un array_entrenamiento y el otro 30% en otro array_testeo. Esta distribución de filas de la matriz inicial, debe ser aleatoria. Mostrar las matrices al ser modificadas por el comando np.random.shuffle('matriz'). Finalmente mostrar los array_entrenamiento y array_testeo.

```
dataset = np.array([[25, 1, 7, 100, 1],
                    [30, 2, 5, 120, 0],
                    [22, 1, 6, 80, 1],
                    [28, 1, 6, 90, 0],
                    [35, 2, 4, 130, 1],
                    [32, 2, 6, 110, 1],
                    [26, 1, 8, 95, 1],
                    [24, 1, 5, 85, 0],
                    [29, 2, 7, 115, 1],
                    [31, 2, 6, 105, 0]])

# Mezclar las filas de la matriz para obtener una distribución aleatoria de los datos
np.random.shuffle(dataset)
print("Matriz despues de mezclar filas:\n", dataset)

filas_de_entrenamiento = int(0.7*dataset.shape[0])# aqui se me presenta un problema que no toma el asterisco para operar
array_entrenamiento = dataset[:filas_de_entrenamiento]
array_de_testeo = dataset[filas_de_entrenamiento]

print("Array de entrenamiento:\n ", array_entrenamiento)
print("Array de testeo:\n "), array_de_testeo
```

```
Matriz despues de mezclar filas:
 [[ 28   1   6  90   0]
```

```
       [ 35   2   4 130   1]
       [ 30   2   5 120   0]
       [ 25   1   7 100   1]
       [ 22   1   6  80   1]
       [ 29   2   7 115   1]
       [ 26   1   8  95   1]
       [ 32   2   6 110   1]
       [ 24   1   5  85   0]
       [ 31   2   6 105   0]]
     Array de entrenamiento:
       [[ 28   1   6  90   0]
       [ 35   2   4 130   1]
       [ 30   2   5 120   0]
       [ 25   1   7 100   1]
       [ 22   1   6  80   1]
       [ 29   2   7 115   1]
       [ 26   1   8  95   1]]
     Array de testeo:

     (None, array([ 32,   2,   6, 110,   1]))
```

Ejercicio 2:

Dado la siguiente tabla de datos poblaciones de las Provincias de Argentina (Ejercicio 10 del Práctico 1), Realizar el siguiente analisis.

- indicar la cantidad de filas y columnas que posee la tabla de datos.
- Mostrar toda la información de la provincia con Mayor Cantidad de habitantes. AYUDA: usar la función np.max(array)
- Agregar a la tabla de datos una fila al final , indicando los totales de cada columna. Mostrar el resultado de la nueva tabla.

```
poblacionArgentina1=[
    ['PROVINCIA','CANTIDAD DE HABITANTES','CONSUMO EN MWH','SUPERFICIE EN KM^2'],
    ['Buenos Aires','17.569.053',' 16543722',' 305907'],
    ['Córdoba','3.978.984',' 10606601','164708'],
    ['Santa Fe','3.556.522',' 13078203',' 133249'],
    ['Ciudad Autónoma de Buenos Aires','3.120.612','51712507',' 201'],
    ['Mendoza','2.014.533',' 5652519',' 149069'],
    ['Tucumán','1.703.186','3208711','22.524'],
    ['Salta','1.440.672',' 2214796',' 155341'],
    ['Entre Ríos','1.426.426','3906353','78384'],
    ['Misiones','1.280.960','2845762',' 29911'],
    ['Corrientes','1.197.553','2997612',' 89123'],
    ['Chaco','1.142.963','3045380',' 99763'],
    ['Santiago del Estero','1.054.028',' 1811277',' 136934'],
    ['San Juan','818.234',' 2381940',' 88296'],
   ['Jujuy','797.955',' 1136336',' 53244'],
   ['Río Negro','762.067',' 1984782','202169'],
   ['Neuquén','726.590','1834879',' 94422'],
   ['Formosa','606.041',' 1388311','75488'],
   ['Chubut','603.120','1646029',' 224302'],
   ['San Luis','540.905',' 1780881','75347'],
   ['Catamarca','429.556',' 1337032','101486'],
   ['La Rioja','384.607','1572290',' 91494'],
   ['La Pampa','366.022','915781',' 143493'],
   ['Santa Cruz','333.473',' 1025648',' 244458'],
   ['Tierra del Fuego, Antártida e Islas del Atlántico Sur','190.641',' s/d ',' 37131']]

poblacionArgentina=np.array(poblacionArgentina1)

print(poblacionArgentina)
# Cantidad de filas y columnas
filas, columnas = poblacionArgentina.shape
print("Cantidad de filas:", filas)
print("Cantidad de columnas:", columnas)
# Encontrar la provincia con mayor cantidad de habitantes usando np.max
mayor_habitantes = 0
provincia_mayor_habitantes = ''

for i in range(1, filas):
    habitantes = int(poblacionArgentina[i, 1].replace('.', ''))
    if habitantes > mayor_habitantes:
        mayor_habitantes = habitantes
        provincia_mayor_habitantes = poblacionArgentina[i, 0]

print("Provincia con mayor cantidad de habitantes:", provincia_mayor_habitantes)

# Mostrar la nueva tabla de datos
print(poblacionArgentina)
```

```
     [['PROVINCIA' 'CANTIDAD DE HABITANTES' 'CONSUMO EN MWH'
       'SUPERFICIE EN KM^2']
      ['Buenos Aires' '17.569.053' ' 16543722' ' 305907']
      ['Córdoba' '3.978.984' ' 10606601' '164708']
```

```
 ['Santa Fe' '3.556.522' ' 13078203' ' 133249']
 ['Ciudad Autónoma de Buenos Aires' '3.120.612' '51712507' ' 201']
 ['Mendoza' '2.014.533' ' 5652519' ' 149069']
 ['Tucumán' '1.703.186' '3208711' '22.524']
 ['Salta' '1.440.672' ' 2214796' ' 155341']
 ['Entre Ríos' '1.426.426' '3906353' '78384']
 ['Misiones' '1.280.960' '2845762' ' 29911']
 ['Corrientes' '1.197.553' '2997612' ' 89123']
 ['Chaco' '1.142.963' '3045380' ' 99763']
 ['Santiago del Estero' '1.054.028' ' 1811277' ' 136934']
 ['San Juan' '818.234' ' 2381940' ' 88296']
 ['Jujuy' '797.955' ' 1136336' ' 53244']
 ['Río Negro' '762.067' ' 1984782' '202169']
 ['Neuquén' '726.590' '1834879' ' 94422']
 ['Formosa' '606.041' ' 1388311' '75488']
 ['Chubut' '603.120' '1646029' ' 224302']
 ['San Luis' '540.905' ' 1780881' '75347']
 ['Catamarca' '429.556' ' 1337032' '101486']
 ['La Rioja' '384.607' '1572290' ' 91494']
 ['La Pampa' '366.022' '915781' ' 143493']
 ['Santa Cruz' '333.473' ' 1025648' ' 244458']
 ['Tierra del Fuego, Antártida e Islas del Atlántico Sur' '190.641'
  ' s/d ' ' 37131']]
Cantidad de filas: 25
Cantidad de columnas: 4
Provincia con mayor cantidad de habitantes: Buenos Aires
[['PROVINCIA' 'CANTIDAD DE HABITANTES' 'CONSUMO EN MWH'
  'SUPERFICIE EN KM^2']
 ['Buenos Aires' '17.569.053' ' 16543722' ' 305907']
 ['Córdoba' '3.978.984' ' 10606601' '164708']
 ['Santa Fe' '3.556.522' ' 13078203' ' 133249']
 ['Ciudad Autónoma de Buenos Aires' '3.120.612' '51712507' ' 201']
 ['Mendoza' '2.014.533' ' 5652519' ' 149069']
 ['Tucumán' '1.703.186' '3208711' '22.524']
 ['Salta' '1.440.672' ' 2214796' ' 155341']
 ['Entre Ríos' '1.426.426' '3906353' '78384']
 ['Misiones' '1.280.960' '2845762' ' 29911']
 ['Corrientes' '1.197.553' '2997612' ' 89123']
 ['Chaco' '1.142.963' '3045380' ' 99763']
 ['Santiago del Estero' '1.054.028' ' 1811277' ' 136934']
 ['San Juan' '818.234' ' 2381940' ' 88296']
 ['Jujuy' '797.955' ' 1136336' ' 53244']
 ['Río Negro' '762.067' ' 1984782' '202169']
 ['Neuquén' '726.590' '1834879' ' 94422']
 ['Formosa' '606.041' ' 1388311' '75488']
 ['Chubut' '603.120' '1646029' ' 224302']
 ['San Luis' '540.905' ' 1780881' '75347']
 ['Catamarca' '429.556' ' 1337032' '101486']
 ['La Rioja' '384.607' '1572290' ' 91494']
 ['La Pampa' '366.022' '915781' ' 143493']
 ['Santa Cruz' '333.473' ' 1025648' ' 244458']
 ['Tierra del Fuego, Antártida e Islas del Atlántico Sur' '190.641'
  ' s/d ' ' 37131']]
```

Ploteo de datos con Matplotlib

A continuación se presentan ejemplos de diferentes tipos de gráficos utilizando Matplotlib:

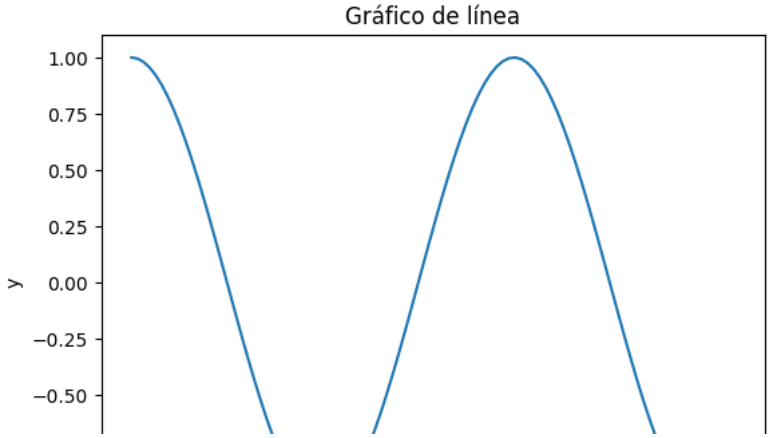
Gráfica de funciones matemáticas elementales

En esta celda, se crea un array 'x' con valores en el rango de 0 a 10 y se utiliza la función np.sin() y np.cos() de NumPy para calcular el seno y coseno de cada valor en 'x'. Luego, se utiliza la biblioteca Matplotlib para trazar un gráfico de línea con 'x' en el eje x y 'y' en el eje y. También se agrega etiquetas y un título al gráfico.

```
import matplotlib.pyplot as plt

x = np.arange(0, 10, 0.1)
y = np.cos(x)

plt.plot(x, y)
plt.xlabel('x')
plt.ylabel('y')
plt.title('Gráfico de línea')
plt.show()
```



EJERCICIOS PARTE 3

Realizar las gráficas de als siguientes expresiones matemáticas. Determinar el dominio del eje x adecuado para plotear las funciones de manera que se visualice su comportamiento.

- $y = 3x - 2$
- $y = 2x^2 + 4x + 2$
- $y = |x| = \begin{cases} -x & x < 0 \\ x & x \geq 0 \end{cases}$
- $y = 1/x$
- $y = \sqrt{x}$

```
#definir los valores de x

x = np.linspace(-10, 10, 400)  # Amplio rango para capturar todas las características

# y = 3x - 2
y1 = 3*x - 2

# y = 2x^2 + 4x + 2
y2 = 2*x**2 + 4*x + 2

# y = |x|
y3_positive = x[x >= 0]
y3_negative = -x[x < 0]

# y = 1/x (excluyendo el punto en x = 0)
y4 = 1/x[x != 0]

# y = √x (solo valores positivos de x)
y5 = np.sqrt(x[x >= 0])

# Crear subplots para mostrar múltiples gráficos en una sola figura
fig, axes = plt.subplots(2, 3, figsize=(15, 10))

# Graficar cada función en su respectivo subplot
axes[0, 0].plot(x, y1)
axes[0, 0].set_title('y = 3x - 2')

axes[0, 1].plot(x, y2)
axes[0, 1].set_title('y = 2x^2 + 4x + 2')

axes[0, 2].plot(y3_positive, y3_positive, label='y = |x| (x >= 0)')
axes[0, 2].plot(y3_negative, -y3_negative, label='y = |x| (x < 0)')
axes[0, 2].set_title('y = |x|')
axes[0, 2].legend()

axes[1, 0].plot(x[x != 0], y4)
axes[1, 0].set_title('y = 1/x')
axes[1, 0].set_ylim(-10, 10)  # Ajustar límites para evitar valores extremos

axes[1, 1].plot(x[x >= 0], y5)
axes[1, 1].set_title('y = √x')

# Eliminar el subplot no utilizado
fig.delaxes(axes[1, 2])

# Ajustar espaciado entre subplots
plt.tight_layout()

# Mostrar los gráficos
```

```
# Mostrar los gráficos
plt.show()
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
<ipython-input-13-65379618a796> in <cell line: 21>()
     19
     20 # Crear subplots para mostrar múltiples gráficos en una sola figura
---> 21 fig, axes = plt.subplots(2, 3, figsize=(15, 10))
     22
     23 # Graficar cada función en su respectivo subplot

NameError: name 'plt' is not defined
```

BUSCAR EN STACK OVERFLOW

0 s se ejecutó 09:40