PythonML / TPN2 / Trabajo_Practico2.ipynb

cristian 13 minutes ago

1259 lines (1259 loc) · 213 KB

Preview Code Blame

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.

In [2]:
```
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.

In [3]:
```
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
```

Out[3]: `[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.

In [4]:
```
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.

In [5]:
```
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

In [6]:
```
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)
```

```
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]
```

Total de ventas por día: [107 101 118 110 98 109 107]

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.

In [7]:
```
# Decidimos crear una funcion para verificar si una entrada es un entero positivo ara pe
def pedir_cantidad():
    cant = -1
    while cant < 0:
        try:
            cant = int(input('Ingrese la cantidad'))
            if cant < 0:
                print('Ingrese un numero positivo')
            else:
                return cant
        except:
            print('Ingrese un numero')
```

In [8]:
```
# Ahora nos aseguramos que ingresara una cantidad positiva y entera
cant = pedir_cantidad()

# Una vez ingresada la cantidad, comenzamos a pedir elementos
lista_numeros = []
print(f'Ingrese {cant} numeros reales positivos')
i = 0
while i < cant:
    try:
        numero = float(input('Ingrese su numero'))
        lista_numeros.append(numero)
        i += 1
    except:
        print('Ingrese numero valido')

# Convertimos la lista generada en np.array y mostramos el array
np_array_numeros = np.array(lista_numeros)
np_array_numeros
```

```
Ingrese 2 numeros reales positivos
```

Out[8]:
```
array([2., 3.])
```

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.

In [9]:
```
# Pedimos filas y columnas
cant_filas = pedir_cantidad()
cant_columnas = pedir_cantidad()

# Definimos una matriz vacia
matriz_py = []

# Ingresamos elemento fila a fila
for i in range(cant_filas):
    fila_nueva = []
    for j in range(cant_columnas):
        valor = float(input(f'Ingrese elemento [{i}][{j}]: '))
        fila_nueva.append(valor)
    matriz_py.append(fila_nueva)
```

```
    matriz_py.append(fila_nueva)

# Convertimos la matriz a numpy
matriz_np = np.array(matriz_py)

# Mostramos primero en formato lista de Python y en array de Numpy
print('Tabla en formato de lista de Python:')
print(f'{matriz_py}')
print('Tabla en formato de array de Numpy:')
print(f'{matriz_np}')

# Pedimos dos filas
fila_1 = int(input('Ingrese numero de una fila: '))
fila_2 = int(input('Ingrese numero de otra fila: '))

# Controlamos que sean cantidades dentro del rango
if fila_1 >= 0 and fila_2 >= 0 and fila_1 < cant_filas and fila_2 < cant_filas:
    resultado = matriz_np[fila_1] + matriz_np[fila_2]
    print(f'vector resultante de sumar las filas {fila_1} y {fila_2} es: {resultado}')
else:
    print('Error al buscar elemento, indice fuera de rango')
```

```
Tabla en formato de lista de Python:
[[2.0, 2.0], [2.0, 2.0]]
Tabla en formato de array de Numpy:
[[2. 2.]
 [2. 2.]]
vector resultante de sumar las filas 0 y 1 es: [4. 4.]
```

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.

In [10]:

```
# Creamos los arrays
productos_py = ['arroz', 'harina', 'fideo', 'yerba', 'azucar']
precios_py = [145.6, 100, 89.90, 700, 95]

# Creamos un diccionario, ya que parece mas conveniente
diccionario_productos = {}

# Los convertimos en array de Numpy
productos_np = np.array(productos_py)
precios_np = np.array(precios_py)

# Calculamos los nuevos precios
precios_np_suba = np.array([])
for i in range(len(precios_np)):
    if i == 0 or i == 1 or i == 4:
        precios_np_suba = np.append(precios_np_suba, precios_np[i] * 1.5)
    else:
        precios_np_suba = np.append(precios_np_suba, precios_np[i] * 1.75)



# Mostramos los datos en forma de vector
print(f'Productos           {productos_py}')
print(f'Precios sin suba    {precios_np}')
print(f'Precios con suba    {precios_np_suba}')
```

```
Productos           ['arroz', 'harina', 'fideo', 'yerba', 'azucar']
```

```
Precios sin suba    [145.6 100.   89.9 700.   95. ]
Precios con suba    [ 218.4    150.     157.325 1225.     142.5  ]
```

__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)| Calcula la raiz cuadrada de cada elemento del vector | array | np.sqrt(vector_np) 3 |np.random.rand(n)| Genera un array de tamaño n con elementos aleatorios | array | np.random.rand(5) 4 |np.ones((n))| Genera un array de tamaño n con elementos igual a 1 | array | np.ones((3)) 5 |np.zeros((n))| Genera un array de tamaño n con elementos igual a 0 | array | np.zeros((3)) 6 |np.min(array)| Calcula el valor minimo del array |valor mínimo| np.min(vector_np) 7 |np.max(array)| Calcula el valor maximo del array |valor máximo| np.max(vector_np) 8 |np.where(CONDICIÓN SOBRE EL VECTOR)| Genera un vector que contien los elementos que cumplen la condicion |array|np.where(vector_np>1) 9 |np.random.shuffle(MATRIZ)| Mezcla las filas de la matriz de manera aleatoria | None | VER EJERCICIO PARTE 2 10 |array.shape[n], n=0,1 | Devuelve la cantidad de filas con n=0 y la cantidad de columnas con n=1 | int | VER EJERCICIO PARTE 2 11 |np.sum(array, axis=n), n=0,1| Calcula la suma del array, segun el valor de n suma las filas o las columnas| array |VER EJERCICIO PARTE 1 12 |np.arange(a, b, p)| Genera un array con valores del intervalo [a , b], estos valores se distancian entre si dado el valor de c | array |np.arange(0, 10, 0.1) VER EJERCICIOS PARTE 3

In [19]:
```
matriz = np.array([[1,2,3],[4,5,6],[7,8,9]])

np.random.shuffle(matriz)
print(matriz)
```

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

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.

In [11]:
```
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, 2.2360679
7749979]
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, 1.3862943
611198906, 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.

In [12]:
```
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()
vector_np * 2
end_time = time.time()
print("Tiempo en NumPy:", end_time - start_time, "segundos")
```

```
Tiempo en Python puro: 0.13469243049621582 segundos
Tiempo en NumPy: 0.009993791580200195 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.

In [13]:
```
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.

In [14]:
```
array = np.array([1, 2, 3, 4, 5, 6])
nuevo_array = np.resize(array, (5, 2))
print("Nuevo array redimensionado:", nuevo_array)
```

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

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.

In [15]:
```
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( dataset)

# Al estar mezclada podemos obtener el 70% de la cantidad de filas y redondear el numero
cant_filas_entrenamiento = int(dataset.shape[0] * .7)
cant_filas_testeo = dataset.shape[0] - cant_filas_entrenamiento

array_entrenamiento = dataset[:cant_filas_entrenamiento]
array_testeo = dataset[cant_filas_entrenamiento:]

print('Entrenamiento: ')
print(array_entrenamiento)

print('Testeo: ')
print(array_testeo)
```

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

In [16]:
```
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']]

poblacion_argentina=np.array(poblacionArgentina1)

print(poblacion_argentina)
```

```
[['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']]
```

In [ ]:

```
# Empezamos el ejercicio 2

# Cantidad de filas y columnas
print(f'Filas:      {poblacion_argentina.shape[0]}')
print(f'Columnas:   {poblacion_argentina.shape[1]}')
```

In [103…

```
# Datos del Mas poblado
# Debido a que los valores numericos estan ingresados como cadenas, procesamos los datos

# Definimos una funcion que borrará los puntos de los elementos, ya que esto generará er
def reemplazar(x):
    return np.char.replace(x, '.', '')

# Para aplicar la solucion necesito la columna de cantidad de habitantes, sin el titulo
cant_habitantes = poblacion_argentina[1:,1]
```

```
# Aplicamos la funcion definida anteriormente
cant_habitantes = np.apply_along_axis(reemplazar, 0, cant_habitantes).astype(int)

# Determinamos la maxima cantidad de habitantes
maximo = np.max(cant_habitantes)

# colocamos en la matriz numpy la columna ya convertida
poblacion_argentina[1:,1] = cant_habitantes

# Mostraremos la info de la provincia con mas habitantes
# la columna no puede ser del tipo int dado que hay una fila que contiene la cadena 'CAN
# La mask nos servira para indexacion booleana, y [:,1] nos permite corroborar que la co
mask = poblacion_argentina[:,1] == str(maximo)
print(poblacion_argentina[mask])
```

```
[['Buenos Aires' '17569053' ' 16543722' ' 305907']]
```

Ploteo de datos con Matplotlib

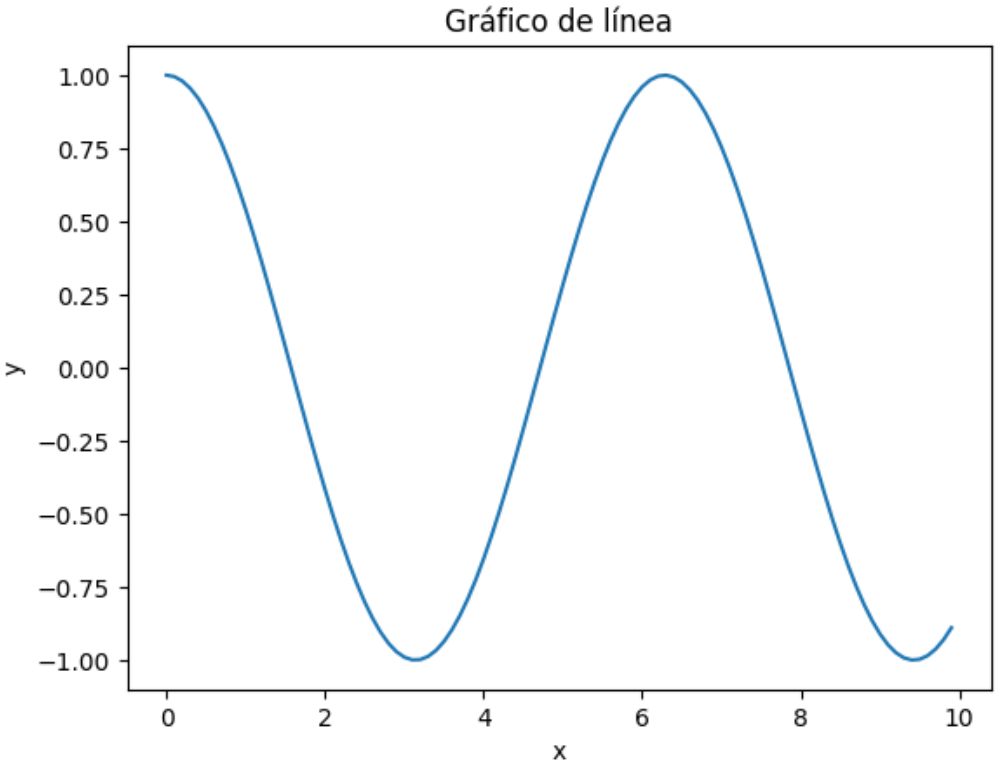
A continuación se presentan ejemplos de diferentes tipos de gráficos utilizando Matplotlib:

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.

In [ ]:

```
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()
```



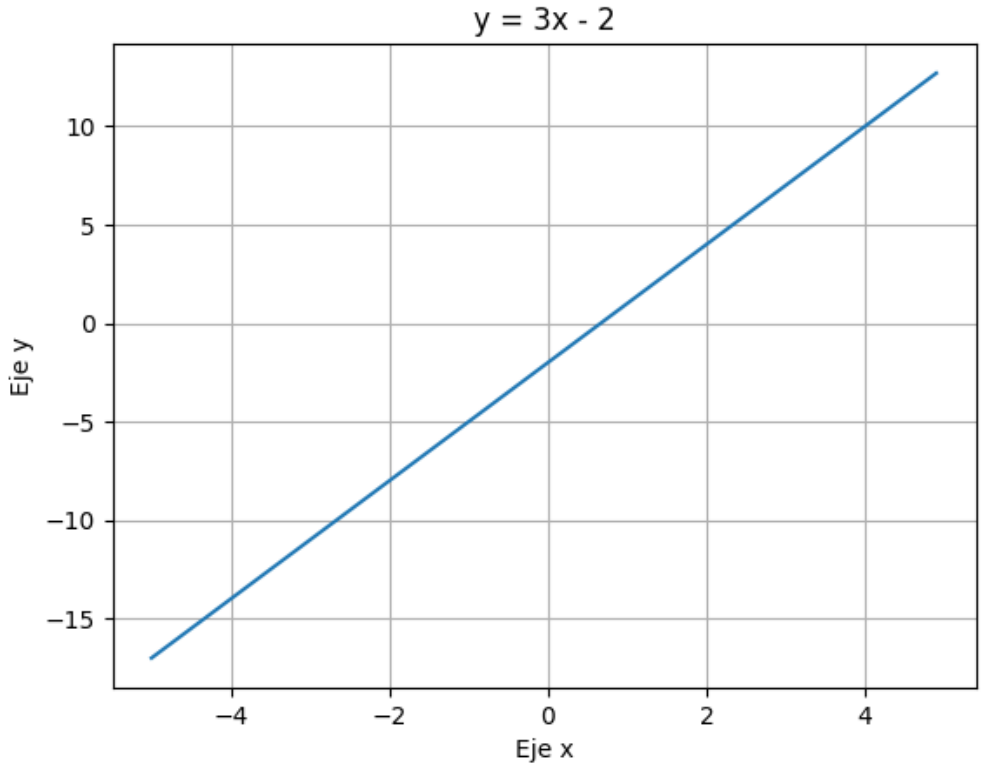
EJERCICIOS PARTE 3

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.

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

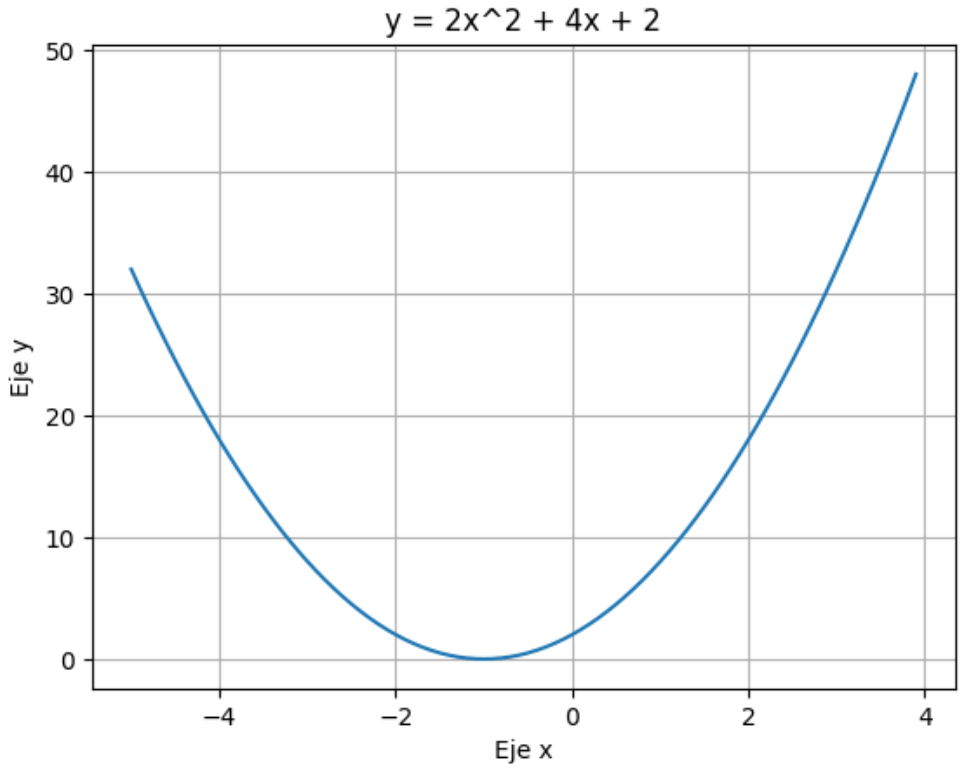
In [2]:
```
# importamos las librerias necesarias
import matplotlib.pyplot as plt
import numpy as np
```

In [110…
```
# y = 3x - 2
# Como su dominio son todos los reales, y es una funcion lineal, tomamos algunos valores
x = np.arange(-5, 5, 0.1)
y = 3 * x - 2

plt.plot(x,y)
plt.grid(True)
plt.xlabel('Eje x')
plt.ylabel('Eje y')
plt.title('y = 3x - 2')
plt.show()
```



In [112…
```
# y = 2x^2 + 4x + 2
# Como su dominio son todos los reales y ademas su imagen son los reales positivos, toma
# Como el determinante es cero, tenemos dos raices iguales, x1 = x2 = -1
x = np.arange(-5, 4, 0.1)
y = 2 * (x ** 2) + 4 * x + 2

plt.plot(x,y)
plt.grid(True)
```

```
plt.xlabel('Eje x')
plt.ylabel('Eje y')
plt.title('y = 2x^2 + 4x + 2')
plt.show()
```



In [117…

```
# Como el valor absoluto se comporta como una funcion lineal cambiando su pendiente con
x = np.arange(-5, 5, 0.1)
y = abs(x)

plt.plot(x,y)
plt.grid(True)
plt.xlabel('Eje x')
plt.ylabel('Eje y')
plt.title('y = |x|')
plt.show()
```