(sin interaccion de consola)Realice un programa que sume, reste, multiplique (producto punto y producto cruz) y divida dos vectores previamente inicializados.

Hola mundo github :::

```
# Definición de funciones para operaciones con vectores
def suma_vectores(vec1, vec2):
    suma = []
    for i in range(len(vec1)):
        suma.append(vec1[i] + vec2[i])
    return suma

def resta_vectores(vec1, vec2):
    resta = []
    for i in range(len(vec1)):
        resta.append(vec1[i] - vec2[i])
    return resta

def producto_punto(vec1, vec2):
    producto = 0
    for i in range(len(vec1)):
        producto += vec1[i] * vec2[i]
    return producto

def producto_cruz(vec1, vec2):
    if len(vec1) != 3 or len(vec2) != 3:
        print("Producto cruz está definido solo para vectores de tamaño 3.")
        return None
    else:
        producto = [vec1[1]*vec2[2] - vec1[2]*vec2[1],
                    vec1[2]*vec2[0] - vec1[0]*vec2[2],
                    vec1[0]*vec2[1] - vec1[1]*vec2[0]]
        return producto

# Vectores inicializados
vector1 = [1, 2, 3]
vector2 = [4, 5, 6]

# Operaciones
suma = suma_vectores(vector1, vector2)
resta = resta_vectores(vector1, vector2)
producto_punto_resultado = producto_punto(vector1, vector2)
```

```
producto_punto_resultado = producto_punto(vector1, vector2)
producto_cruz_resultado = producto_cruz(vector1, vector2)

# Resultados
print("Suma de vectores:", suma)
print("Resta de vectores:", resta)
print("Producto punto de vectores:", producto_punto_resultado)
print("Producto cruz de vectores:", producto_cruz_resultado)

```

```
Suma de vectores: [5, 7, 9]
Resta de vectores: [-3, -3, -3]
Producto punto de vectores: 32
Producto cruz de vectores: [-3, 6, -3]
```

Realice un programa que sume, reste, multiplique (producto punto y producto cruz) y divida dos matrices previamente inicializadas.(sin interaccion de consola)

```
# Definición de funciones para operaciones con matrices
def suma_matrices(matriz1, matriz2):
    suma = []
    for i in range(len(matriz1)):
        fila = []
        for j in range(len(matriz1[0])):
            fila.append(matriz1[i][j] + matriz2[i][j])
        suma.append(fila)
    return suma

def resta_matrices(matriz1, matriz2):
    resta = []
    for i in range(len(matriz1)):
        fila = []
        for j in range(len(matriz1[0])):
            fila.append(matriz1[i][j] - matriz2[i][j])
        resta.append(fila)
    return resta

def producto_punto_matrices(matriz1, matriz2):
    if len(matriz1[0]) != len(matriz2):
        print("No se puede calcular el producto punto. El número de columnas d
        return None
    else:
        producto = []
        for i in range(len(matriz1)):
```

```
            fila = []
            for j in range(len(matriz2[0])):
                elemento = 0
                for k in range(len(matriz2)):
                    elemento += matriz1[i][k] * matriz2[k][j]
                fila.append(elemento)
            producto.append(fila)
        return producto

# Vectores inicializados
matriz1 = [[1, 2, 3],
           [4, 5, 6],
           [7, 8, 9]]

matriz2 = [[9, 8, 7],
           [6, 5, 4],
           [3, 2, 1]]

# Operaciones
suma = suma_matrices(matriz1, matriz2)
resta = resta_matrices(matriz1, matriz2)
producto_punto = producto_punto_matrices(matriz1, matriz2)

# Resultados
print("Suma de matrices:")
for fila in suma:
    print(fila)

print("\nResta de matrices:")
for fila in resta:
    print(fila)

print("\nProducto punto de matrices:")
for fila in producto_punto:
    print(fila)

```

```
Suma de matrices:
[10, 10, 10]
[10, 10, 10]
[10, 10, 10]

Resta de matrices:
[-8, -6, -4]
[-2, 0, 2]
[4, 6, 8]

Producto punto de matrices:
[30, 24, 18]
[84, 69, 54]
[138, 114, 90]
```

Realice un programa que convierta coordenadas rectangulares a cilíndricas y esféricas, para lo cual deben consultar sobre el uso de funciones trigonométricas en Python.

```python
import math

# Función para convertir coordenadas rectangulares a cilíndricas
def rectangulares_a_cilindricas(x, y, z):
    r = math.sqrt(x**2 + y**2)
    theta = math.atan2(y, x)
    return r, theta, z

# Función para convertir coordenadas rectangulares a esféricas
def rectangulares_a_esfericas(x, y, z):
    r = math.sqrt(x**2 + y**2 + z**2)
    theta = math.atan2(y, x)
    phi = math.acos(z / r)
    return r, theta, phi

# Coordenadas rectangulares
x = 3
y = 4
z = 5

# Conversión a coordenadas cilíndricas
r_cilindrica, theta_cilindrica, z_cilindrica = rectangulares_a_cilindricas(x, y

# Conversión a coordenadas esféricas
r_esferica, theta_esferica, phi_esferica = rectangulares_a_esfericas(x, y, z)

# Resultados
print("Coordenadas rectangulares: (x={}, y={}, z={})".format(x, y, z))
print("Coordenadas cilíndricas: (r={}, theta={}, z={})".format(r_cilindrica, t
print("Coordenadas esféricas: (r={}, theta={}, phi={})".format(r_esferica, the

```

```
Coordenadas rectangulares: (x=3, y=4, z=5)
Coordenadas cilíndricas: (r=5.0, theta=0.9272952180016122, z=5)
Coordenadas esféricas: (r=7.0710678118654755, theta=0.9272952180016122, phi=0.
```

Realice un programa para el cálculo de la resistencia de una RTD de platino (PT100) en función de la temperatura.

```python
# Definición de la función para el cálculo de la resistencia de una PT100
def calcular_resistencia_PT100(temperatura):
    # Coeficientes para la ecuación de Callendar-Van Dusen
    A = 3.9083e-3
    B = -5.775e-7
    R0 = 100  # Resistencia nominal a 0°C

    # Ecuación de Callendar-Van Dusen para la resistencia en función de la tem
    resistencia = R0 * (1 + A * temperatura + B * temperatura**2)

    return resistencia

# Temperatura en grados Celsius
temperatura = 25  # Cambiar según la temperatura deseada

# Calcular la resistencia para la temperatura dada
resistencia = calcular_resistencia_PT100(temperatura)

# Mostrar resultado
print("La resistencia de la PT100 a", temperatura, "grados Celsius es:", resis

```

```
La resistencia de la PT100 a 25 grados Celsius es: 109.73465625 Ohms
```

Realice en funciones las rotaciones en X, Y y Z, donde se tenga un parámetro de entrada (ángulo) y un parámetro de salida (matriz).

```python
import math

# Función para rotación en el eje X
def rotacion_x(angulo):
    radianes = math.radians(angulo)
    matriz = [
        [1, 0, 0],
        [0, math.cos(radianes), -math.sin(radianes)],
        [0, math.sin(radianes), math.cos(radianes)]
    ]
    return matriz

# Función para rotación en el eje Y
def rotacion_y(angulo):
    radianes = math.radians(angulo)
```

```
    matriz = [
        [math.cos(radianes), 0, math.sin(radianes)],
        [0, 1, 0],
        [-math.sin(radianes), 0, math.cos(radianes)]
    ]
    return matriz

# Función para rotación en el eje Z
def rotacion_z(angulo):
    radianes = math.radians(angulo)
    matriz = [
        [math.cos(radianes), -math.sin(radianes), 0],
        [math.sin(radianes), math.cos(radianes), 0],
        [0, 0, 1]
    ]
    return matriz

# Ejemplo de uso
angulo = 45  # Ángulo de rotación en grados

# Rotación en el eje X
matriz_rotacion_x = rotacion_x(angulo)
print("Matriz de rotación en el eje X para un ángulo de", angulo, "grados:")
for fila in matriz_rotacion_x:
    print(fila)

# Rotación en el eje Y
matriz_rotacion_y = rotacion_y(angulo)
print("\nMatriz de rotación en el eje Y para un ángulo de", angulo, "grados:")
for fila in matriz_rotacion_y:
    print(fila)

# Rotación en el eje Z
matriz_rotacion_z = rotacion_z(angulo)
print("\nMatriz de rotación en el eje Z para un ángulo de", angulo, "grados:")
for fila in matriz_rotacion_z:
    print(fila)

```

```
Matriz de rotación en el eje X para un ángulo de 45 grados:
[1, 0, 0]
[0, 0.7071067811865476, -0.7071067811865475]
[0, 0.7071067811865475, 0.7071067811865476]

Matriz de rotación en el eje Y para un ángulo de 45 grados:
[0.7071067811865476, 0, 0.7071067811865475]
[0, 1, 0]
[-0.7071067811865475, 0, 0.7071067811865476]

Matriz de rotación en el eje Z para un ángulo de 45 grados:
[0.7071067811865476, -0.7071067811865475, 0]
[0.7071067811865475, 0.7071067811865476, 0]
[0, 0, 1]
```

Realice un programa que calcule la fuerza de avance y retroceso de un cilindro neumático de doble efecto. Debe establecer previamente los valores de presión, así como las dimensiones físicas del cilindro para realizar el cálculo.

```
# Función para calcular la fuerza de avance del cilindro neumático
def fuerza_avance(presion, area_piston):
    fuerza = presion * area_piston
    return fuerza

# Función para calcular la fuerza de retroceso del cilindro neumático
def fuerza_retroceso(presion, area_piston, area_vastago):
    fuerza = (presion * area_piston) - (presion * area_vastago)
    return fuerza

# Valores dados
presion = 100  # Presión en psi
diametro_piston = 2  # Diámetro del pistón en pulgadas
diametro_vastago = 1  # Diámetro del vástago en pulgadas

# Área de los pistones
area_piston = (3.1416 * (diametro_piston / 2)**2)  # Área del pistón en pulgad
area_vastago = (3.1416 * (diametro_vastago / 2)**2)  # Área del vástago en pul

# Calcular fuerzas
fuerza_avance_cilindro = fuerza_avance(presion, area_piston)
fuerza_retroceso_cilindro = fuerza_retroceso(presion, area_piston, area_vastag

# Mostrar resultados
print("Fuerza de avance del cilindro neumático:", fuerza_avance_cilindro, "lib
print("Fuerza de retroceso del cilindro neumático:", fuerza_retroceso_cilindro

```

```
Fuerza de avance del cilindro neumático: 314.15999999999997 libras
Fuerza de retroceso del cilindro neumático: 235.61999999999998 libras
```

Con interacción de consola (fprintf o disp) y teclado (input) Realice un programa que calcule la potencia que consume un circuito ingresando por teclado el valor de corriente y voltaje.

```
# Solicitar al usuario que ingrese el valor de la corriente y el voltaje
corriente = float(input("Ingrese el valor de la corriente en amperios: "))
voltaje = float(input("Ingrese el valor del voltaje en voltios: "))

# Calcular la potencia consumida por el circuito (P = V * I)
potencia = voltaje * corriente

# Mostrar el resultado al usuario
print("La potencia consumida por el circuito es:", potencia, "vatios")

```

```
Ingrese el valor de la corriente en amperios: 4
Ingrese el valor del voltaje en voltios: 2
La potencia consumida por el circuito es: 8.0 vatios
```

Realice un programa que calcule X números aleatorios en un rango determinado por el usuario.

```
import random

# Solicitar al usuario el número de números aleatorios y los límites del rango
cantidad_numeros = int(input("Ingrese la cantidad de números aleatorios que de
limite_inferior = float(input("Ingrese el límite inferior del rango: "))
limite_superior = float(input("Ingrese el límite superior del rango: "))

# Verificar que el límite superior sea mayor que el límite inferior
if limite_superior <= limite_inferior:
    print("Error: El límite superior debe ser mayor que el límite inferior.")
else:
    # Generar los números aleatorios en el rango especificado
    numeros_aleatorios = [random.uniform(limite_inferior, limite_superior) for

    # Mostrar los números aleatorios generados
    print("Números aleatorios generados en el rango [{}, {}]:".format(limite_i
    for numero in numeros_aleatorios:
        print(numero)

```

```
Ingrese la cantidad de números aleatorios que desea generar: 5
Ingrese el límite inferior del rango: 2
Ingrese el límite superior del rango: 10
Números aleatorios generados en el rango [2.0, 10.0]:
2.286024785783531
7.30681293820855
4.9385917371707775
5.899126651175953
9.017195976648487
```

Realice un programa para el cálculo de volúmenes (Prisma, Pirámide, Cono truncado, Cilindro) donde el usuario pueda seleccionar el sólido y los parámetros de cada volumen.

```
import math

# Función para calcular el volumen de un prisma
def volumen_prisma(base, altura):
    return base * altura

# Función para calcular el volumen de una pirámide
def volumen_piramide(base, altura):
    return (1/3) * base * altura

```

```
# Función para calcular el volumen de un cono truncado
def volumen_cono_truncado(radio_mayor, radio_menor, altura):
    return (1/3) * math.pi * altura * (radio_mayor**2 + radio_mayor * radio_me

# Función para calcular el volumen de un cilindro
def volumen_cilindro(radio, altura):
    return math.pi * radio**2 * altura

# Función para solicitar al usuario los parámetros y calcular el volumen del s
def calcular_volumen():
    opcion = int(input("Seleccione el sólido para calcular su volumen:\n1. Pri

    if opcion == 1:  # Prisma
        base = float(input("Ingrese el valor de la base del prisma: "))
        altura = float(input("Ingrese el valor de la altura del prisma: "))
        resultado = volumen_prisma(base, altura)
    elif opcion == 2:  # Pirámide
        base = float(input("Ingrese el valor de la base de la pirámide: "))
        altura = float(input("Ingrese el valor de la altura de la pirámide: ")
        resultado = volumen_piramide(base, altura)
    elif opcion == 3:  # Cono truncado
        radio_mayor = float(input("Ingrese el valor del radio mayor del cono t
        radio_menor = float(input("Ingrese el valor del radio menor del cono t
        altura = float(input("Ingrese el valor de la altura del cono truncado:
        resultado = volumen_cono_truncado(radio_mayor, radio_menor, altura)
    elif opcion == 4:  # Cilindro
        radio = float(input("Ingrese el valor del radio del cilindro: "))
        altura = float(input("Ingrese el valor de la altura del cilindro: "))
        resultado = volumen_cilindro(radio, altura)
    else:
        print("Opción no válida")
        return

    print("El volumen del sólido seleccionado es:", resultado)

# Llamar a la función para calcular el volumen
calcular_volumen()

```

```
Seleccione el sólido para calcular su volumen:
1. Prisma
2. Pirámide
3. Cono truncado
4. Cilindro
3
Ingrese el valor del radio mayor del cono truncado: 10
Ingrese el valor del radio menor del cono truncado: 3
Ingrese el valor de la altura del cono truncado: 20
El volumen del sólido seleccionado es: 2911.209192326541
```

Realice un programa que le permita al usuario escoger entre robot Cilíndrico, Cartesiano y esférico, donde como respuesta a la selección conteste con el tipo y número de articulaciones que posee.

```
# Función para proporcionar información sobre el tipo y número de articulacion
def robot_cilindrico():
    print("El robot cilíndrico tiene 2 articulaciones rotativas.")

# Función para proporcionar información sobre el tipo y número de articulacion
def robot_cartesiano():
    print("El robot cartesiano tiene 3 articulaciones prismáticas.")

# Función para proporcionar información sobre el tipo y número de articulacion
def robot_esferico():
    print("El robot esférico tiene 3 articulaciones rotativas.")

# Función para solicitar al usuario que seleccione el tipo de robot y mostrar
def seleccionar_robot():
    opcion = int(input("Seleccione el tipo de robot:\n1. Cilíndrico\n2. Cartes

    if opcion == 1:
        robot_cilindrico()
    elif opcion == 2:
        robot_cartesiano()
    elif opcion == 3:
        robot_esferico()
    else:
        print("Opción no válida")

# Llamar a la función para seleccionar el tipo de robot
seleccionar_robot()

```

```
Seleccione el tipo de robot:
1. Cilíndrico
2. Cartesiano
3. Esférico
2
El robot cartesiano tiene 3 articulaciones prismáticas.
```

Escribir un programa que realice la pregunta ¿Desea continuar Si/No? y que no deje de hacerla hasta que el usuario teclee No.

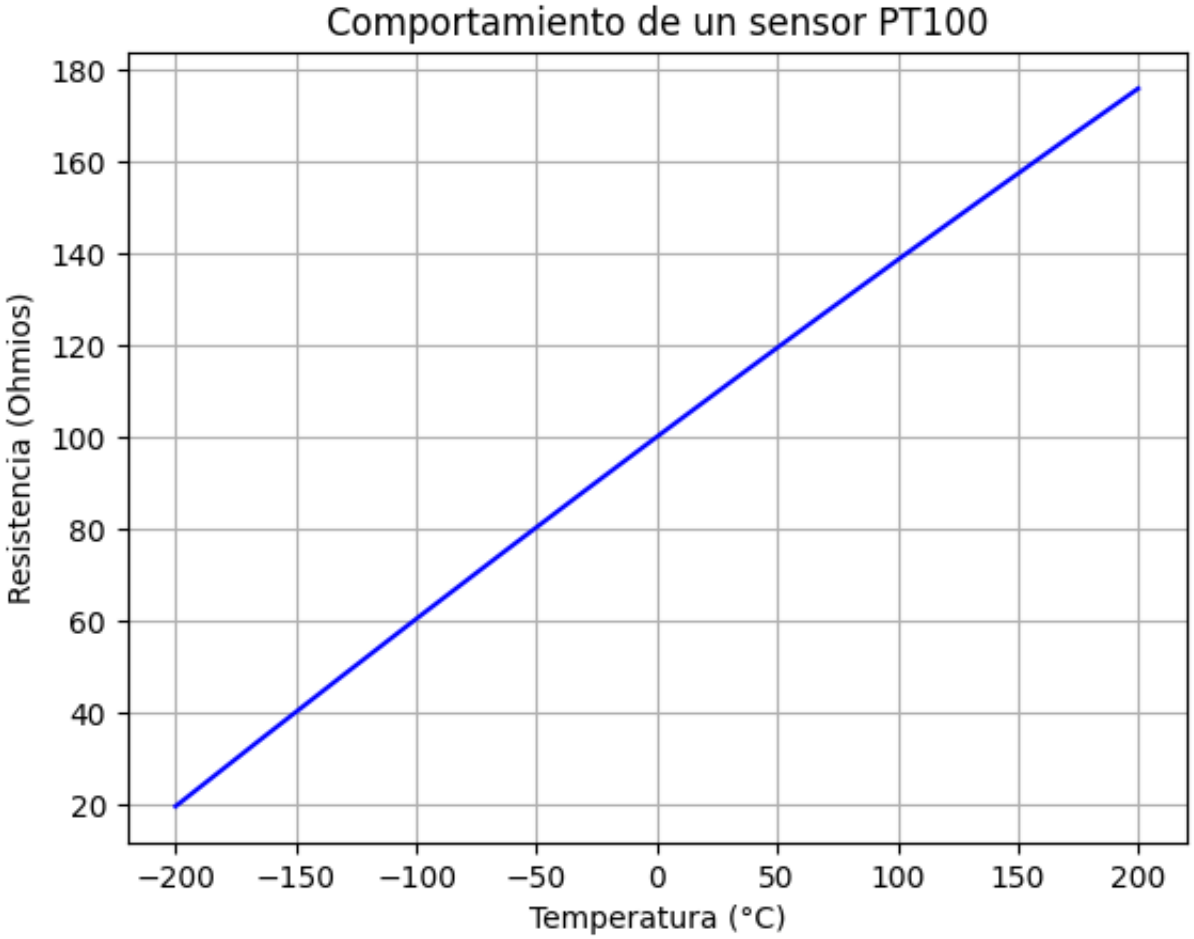
```
# Función para realizar la pregunta y esperar la respuesta del usuario
def preguntar_continuar():
    while True:
        respuesta = input("¿Desea continuar? (Si/No): ")
        if respuesta.lower() == "no":
            print("¡Gracias por usar el programa!")
            break
        elif respuesta.lower() == "si":
            continue
        else:
            print("Por favor, ingrese 'Si' o 'No'.")

# Llamar a la función para iniciar la interacción con el usuario
preguntar_continuar()

```

```
¿Desea continuar? (Si/No): si
¿Desea continuar? (Si/No): si
¿Desea continuar? (Si/No): si
¿Desea continuar? (Si/No): no
¡Gracias por usar el programa!
```

Uso de las funciones para graficar Realice un programa que grafique el comportamiento de un sensor PT100 desde -200°C a 200°C.

```
import matplotlib.pyplot as plt

# Función para calcular la resistencia de un sensor PT100 en función de la tem
def resistencia_PT100(temperatura):
    # Coeficientes de la ecuación de Callendar-Van Dusen
    A = 3.9083e-3
    B = -5.775e-7
    R0 = 100  # Resistencia nominal a 0°C

    # Ecuación de Callendar-Van Dusen para la resistencia en función de la tem
    resistencia = R0 * (1 + A * temperatura + B * temperatura**2)

    return resistencia

# Temperaturas desde -200°C a 200°C
temperaturas = list(range(-200, 201))

# Calcular las resistencias correspondientes
```

```
resistencias = [resistencia_PT100(temp) for temp in temperaturas]

# Graficar
plt.plot(temperaturas, resistencias, color='blue')
plt.title('Comportamiento de un sensor PT100')
plt.xlabel('Temperatura (°C)')
plt.ylabel('Resistencia (Ohmios)')
plt.grid(True)
plt.show()

```

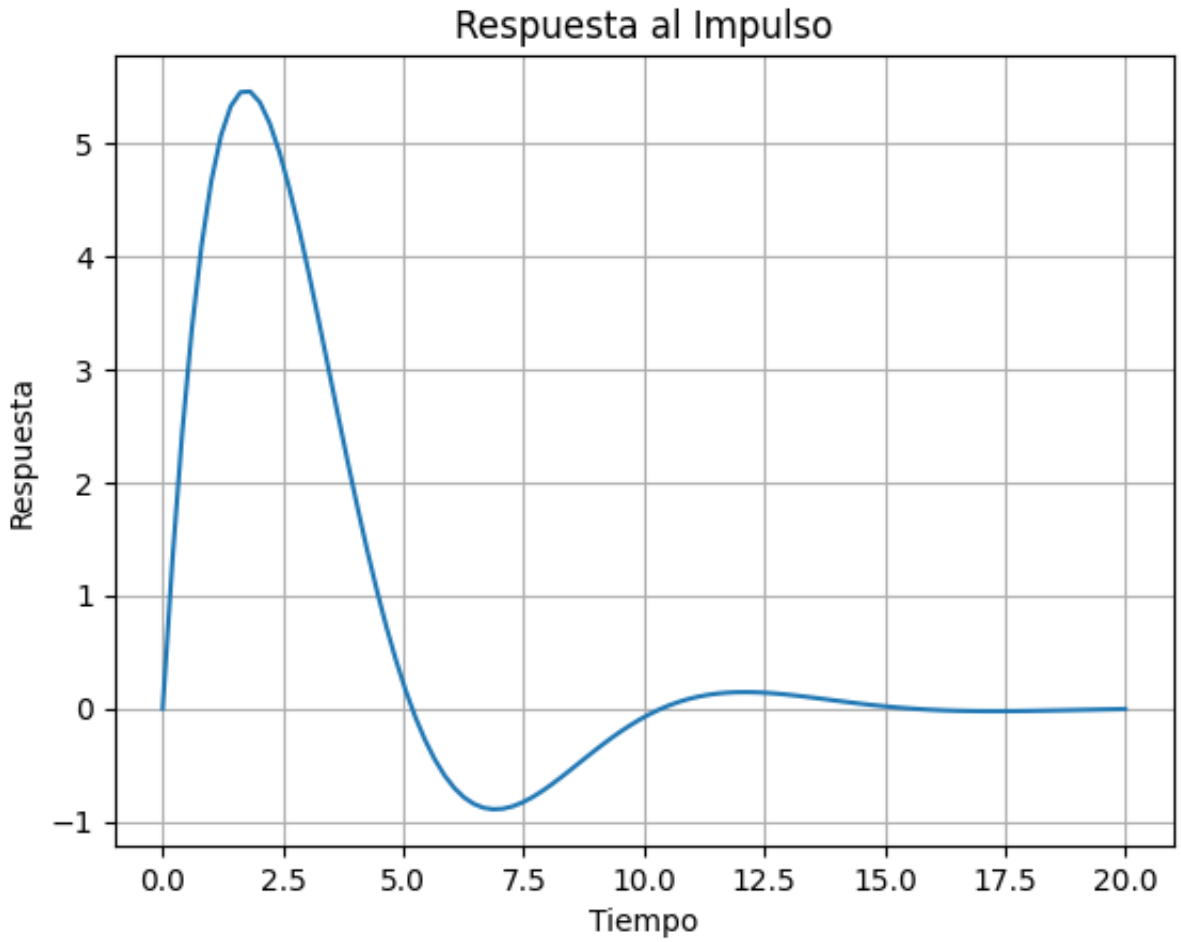


Realice un programa que le permita al usuario ingresar los coeficientes de una función de transferencia de segundo orden y graficar su comportamiento, además se debe mostrar que tipo de sistema es: subamortiguado, criticamente amortiguado y sobreamortiguado.

```
import numpy as np
import matplotlib.pyplot as plt
from scipy import signal
```

```
4
5 def sistema_segundo_orden(wn, zeta):
6     # Sistema de segundo orden
7     num = [wn**2]
8     den = [1, 2 * zeta * wn, wn**2]
9     return num, den
10
11 def tipo_de_sistema(zeta):
12     # Determinar el tipo de sistema
13     if zeta < 1:
14         return "Subamortiguado"
15     elif zeta == 1:
16         return "Críticamente Amortiguado"
17     else:
18         return "Sobreamortiguado"
19
20 def respuesta_impulso(num, den):
21     # Calcular la respuesta al impulso
22     _, y = signal.impulse((num, den))
23     return y
24
25 # Solicitar coeficientes al usuario
26 wn = float(input("Ingrese la frecuencia natural (wn): "))
27 ganancia = float(input("Ingrese la ganancia: "))
28 zeta = float(input("Ingrese el coeficiente de amortiguamiento (zeta): "))
29
30 # Determinar el tipo de sistema
31 tipo = tipo_de_sistema(zeta)
32 print("El sistema es:", tipo)
33
34 # Obtener la función de transferencia
35 num, den = sistema_segundo_orden(wn, zeta)
36
37 # Graficar la respuesta al impulso
38 # Generar tiempo para la gráfica de acuerdo con la longitud de y
39 t = np.linspace(0, 20, len(y))
40
41 y = respuesta_impulso(num, den)
42 plt.plot(t, ganancia * y)
43 plt.title('Respuesta al Impulso')
44 plt.xlabel('Tiempo')
45 plt.ylabel('Respuesta')
46 plt.grid(True)
47 plt.show()
48
```

```
Ingrese la frecuencia natural (wn): 5
Ingrese la ganancia: 2
Ingrese el coeficiente de amortiguamiento (zeta): 0.5
El sistema es: Subamortiguado
```

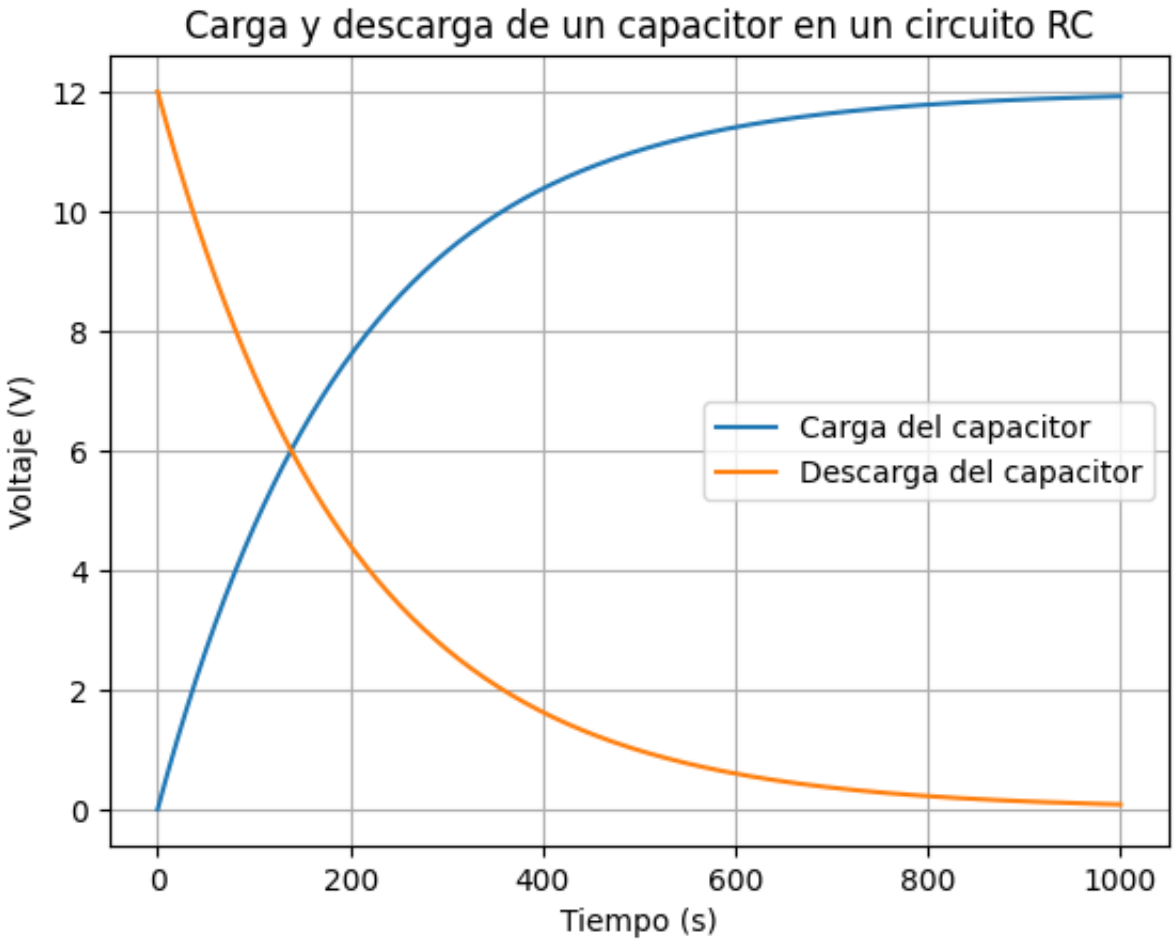


Implemente la ecuación de carga y descarga para un circuito RC. El usuario ingresa por teclado el valor de voltaje (V), capacitancia (μF) y resistencia (Ω). Posteriormente realice en Python la gráfica.

```
import numpy as np
import matplotlib.pyplot as plt

# Función para calcular la carga del capacitor
def carga_capacitor(voltaje, capacitancia, resistencia, tiempo):
    return voltaje * (1 - np.exp(-tiempo / (resistencia * capacitancia)))

# Función para calcular la descarga del capacitor
def descarga_capacitor(voltaje, capacitancia, resistencia, tiempo):
    return voltaje * np.exp(-tiempo / (resistencia * capacitancia))
```

```
    return voltaje * np.exp(-tiempo / (resistencia * capacitancia))

# Solicitar al usuario que ingrese los valores
voltaje = float(input("Ingrese el valor del voltaje (V): "))
capacitancia = float(input("Ingrese el valor de la capacitancia (μF): "))
resistencia = float(input("Ingrese el valor de la resistencia (Ω): "))

# Crear un arreglo de tiempo desde 0 a 5 constantes de tiempo
constante_tiempo = resistencia * capacitancia
tiempo = np.linspace(0, 5 * constante_tiempo, 500)

# Calcular la carga y descarga del capacitor
carga = carga_capacitor(voltaje, capacitancia, resistencia, tiempo)
descarga = descarga_capacitor(voltaje, capacitancia, resistencia, tiempo)

# Graficar
plt.plot(tiempo, carga, label='Carga del capacitor')
plt.plot(tiempo, descarga, label='Descarga del capacitor')
plt.title('Carga y descarga de un capacitor en un circuito RC')
plt.xlabel('Tiempo (s)')
plt.ylabel('Voltaje (V)')
plt.legend()
plt.grid(True)
plt.show()

```

```
Ingrese el valor del voltaje (V): 12
Ingrese el valor de la capacitancia (µF): 100
Ingrese el valor de la resistencia (Ω): 2
```

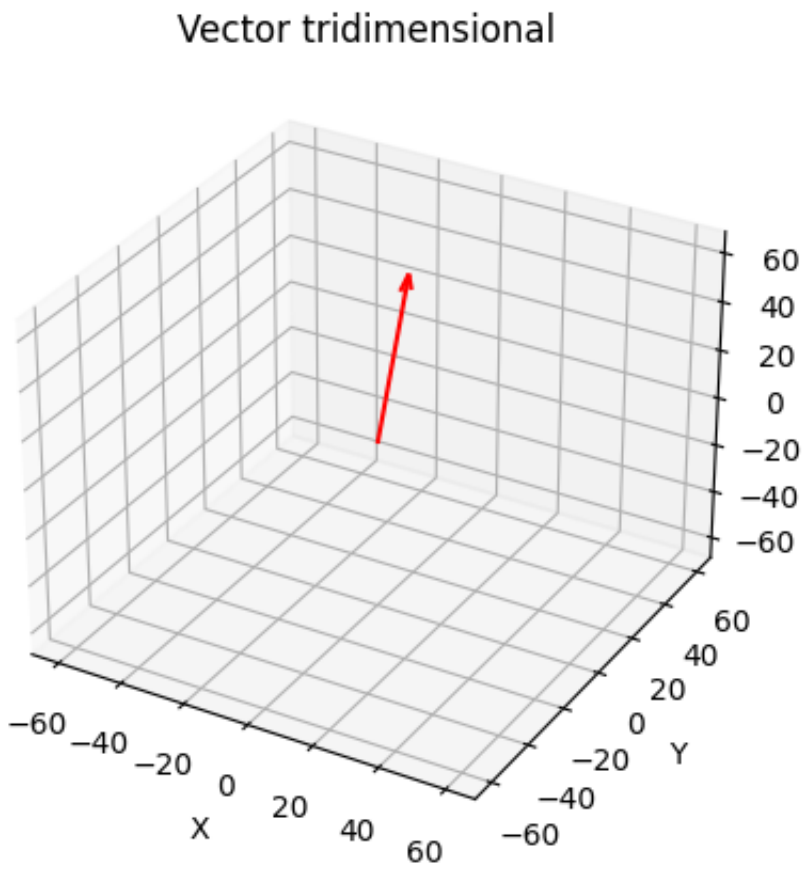


Consulte y elabore un sistema coordenado X, Y y Z donde se dibuje un vector con coordenadas ingresadas por el usuario.

```
import numpy as np
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D

# Solicitar al usuario que ingrese las coordenadas del vector
x = float(input("Ingrese la coordenada X del vector: "))
y = float(input("Ingrese la coordenada Y del vector: "))
z = float(input("Ingrese la coordenada Z del vector: "))

# Crear la figura y los ejes 3D
fig = plt.figure()
ax = fig.add_subplot(111, projection='3d')
```

```
13
14 # Dibujar el vector
15 ax.quiver(0, 0, 0, x, y, z, color='r', arrow_length_ratio=0.1)
16
17 # Establecer límites de los ejes
18 max_coord = max(abs(x), abs(y), abs(z))
19 ax.set_xlim([-max_coord, max_coord])
20 ax.set_ylim([-max_coord, max_coord])
21 ax.set_zlim([-max_coord, max_coord])
22
23 # Etiquetas de los ejes
24 ax.set_xlabel('X')
25 ax.set_ylabel('Y')
26 ax.set_zlabel('Z')
27
28 # Título del gráfico
29 plt.title('Vector tridimensional')
30
31 # Mostrar el gráfico
32 plt.show()
33
```

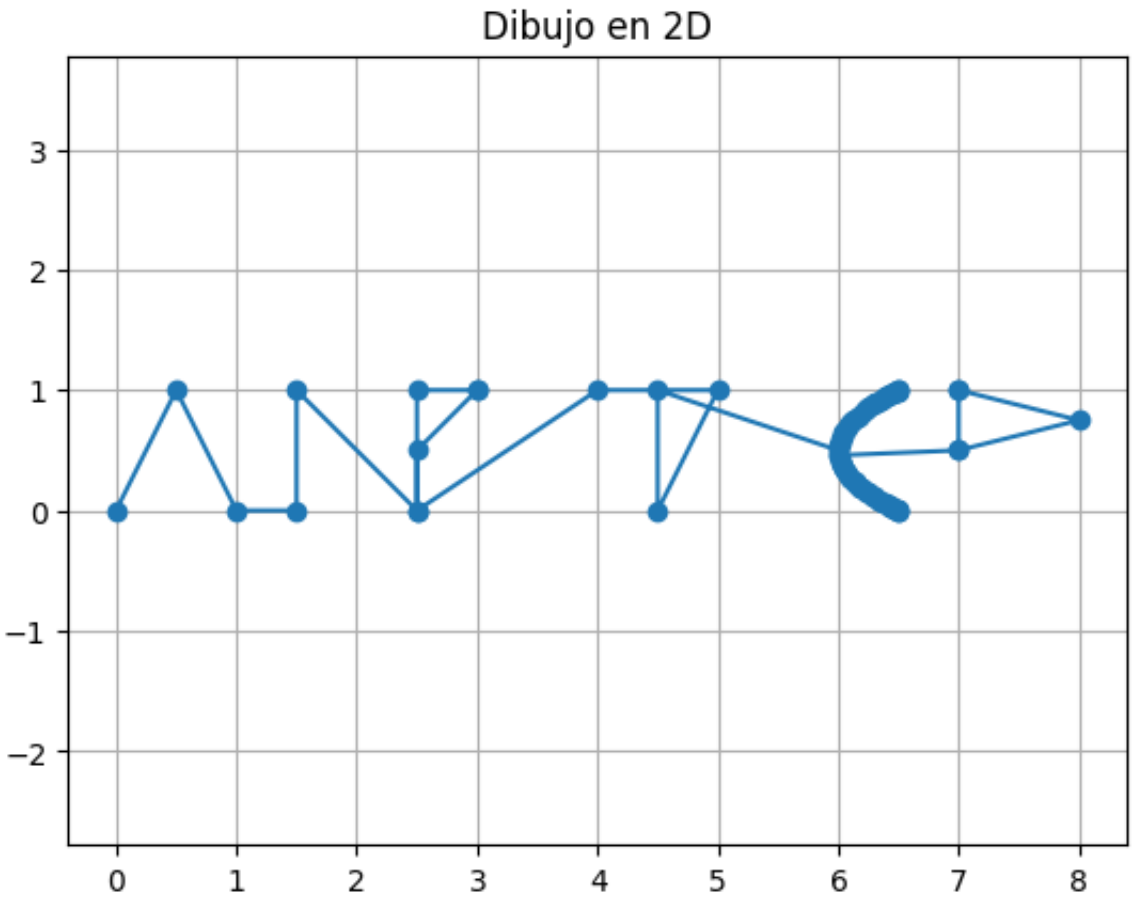
```
Ingrese la coordenada X del vector: 5
Ingrese la coordenada Y del vector: 8
Ingrese la coordenada Z del vector: 66
```



Dibuje el nombre de cada uno de los integrantes del grupo en un plot en 2D, teniendo en cuenta líneas rectas y/o curvas.

```
import matplotlib.pyplot as plt

# Coordenadas para la letra 'V'
V_x = [0, 0.5, 1]
V_y = [0, 1, 0]

# Coordenadas para la letra 'I'
I_x = [1.5, 1.5]
I_y = [0, 1]

# Coordenadas para la letra 'C'
C_x = [2.5, 2.5, 3]
C_y = [0, 1, 1]
C_curve_x = [3, 2.5, 2.5]
```

```
C_curve_y = [1, 0.5, 0]

# Coordenadas para la letra 'T'
T_x = [4, 5]
T_y = [1, 1]
T_vline_x = [4.5, 4.5]
T_vline_y = [0, 1]

# Coordenadas para la letra 'O'
O_theta = [i * 0.1 for i in range(63)]  # Ángulos para trazar el círculo
O_x = [6 + 0.5 * (1 - pow(math.cos(theta), 2)) for theta in O_theta]
O_y = [0.5 + 0.5 * math.sin(theta) for theta in O_theta]

# Coordenadas para la letra 'R'
R_x = [7, 7]
R_y = [0.5, 1]
R_curve_x = [7, 8, 7]
R_curve_y = [1, 0.75, 0.5]

# Unir las coordenadas de todas las letras
x = V_x + I_x + C_x + C_curve_x + T_x + T_vline_x + O_x + R_x + R_curve_x
y = V_y + I_y + C_y + C_curve_y + T_y + T_vline_y + O_y + R_y + R_curve_y

# Dibujar las letras
plt.plot(x, y, marker='o')
plt.title('Dibujo en 2D')
plt.axis('equal')  # Para asegurar que los ejes tengan la misma escala
plt.grid(True)  # Para añadir una cuadrícula al gráfico
plt.show()

```



Obtenga las coordenadas X y Y de los contornos de dos logos de automóviles (Chevrolet, Hyundai, Mazda, etc.), a través de Python.

```
1
```