Taller 1 – Python (código)

Gilbert Alexander Cantor

Kevin Dueñas

Daniel Ricardo Quiroga

Leidy Barragan

Universidad ECCI

Electiva Robótica

Fabian Barrera

A. Sin interacción de consola

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

```python
#Realice un programa que sume, reste, multiplique (producto punto y producto cruz) y divida dos
#vectores previamente inicializados.
import numpy as np

v1 = np.array([4,5,7])
v2 = np.array([3,9,2])
print("el vector 1 es: ",v1)
print("el vector 2 es: ",v2)

suma = v1+v2
resta = v1-v2
multi = v1*v2
pp = np.dot(v1,v2)
pc = np.cross(v1,v2)
div = v1/v2

print(f'''la suma de los vectores es: {suma}
la resta de los vectores es: {resta}
la multiplicacion de los vectores es: {multi}
el producto punto es: {pp}
el producto cruz es: {pc}
la division es: {div}
''')
```

```
PS C:\Users\ACER\Desktop\Universidad Ecci\Octavo Semestre\RoboticaIA\Primer corte\Taller1> & C:/Users/ACER/AppData/Local/Programs/Python/Python312/python.exe "c:/Users/ACER/Desktop/Universidad Ecci/Octavo Semestre/RoboticaIA/Primer corte/Taller1/Punto1A.py"
el vector 1 es:  [4 5 7]
el vector 2 es:  [3 9 2]
la suma de los vectores es: [ 7 14  9]
la resta de los vectores es: [ 1 -4  5]
la multiplicacion de los vectores es: [12 45 14]
el producto punto es: 71
el producto cruz es: [-53  13  21]
la division es: [1.33333333 0.55555556 3.5       ]
```

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

```python
#Realice un programa que sume, reste, multiplique (producto punto y producto cruz)
# y divida dos matrices previamente inicializadas.
import numpy as np

a = np.array([[3,5,6],
              [5,8,3],
              [6,7,9]])

b = np.array([[9,8,7],
              [6,5,4],
              [3,5,7]])


suma = a+b
resta = a-b
pp = np.dot(a,b)
pc = np.cross(a,b)
div= np.divide(a,b)

print("Matriz A:\n", a)
print("Matriz B:\n", b)

print(f'''
suma de las matrices a y b\n {suma}\n
resta de las matrices a y b\n {resta}\n
producto punto de las matrices a y b\n {pp}\n
producto cruz de las matrices a y b\n {pc}\n
division de las matrices a y b\n {div}\n
''')
```

```
producto cruz de las matrices a y b
 [[-13  33 -21]
 [ 17  -2 -23]
 [  4 -15   9]]

division de las matrices a y b
 [[0.33333333 0.625      0.85714286]
 [0.83333333 1.6        0.75       ]
 [2.         1.4        1.28571429]]
```

3. 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
#Realice un programa que convierta coordenadas rectangulares a cilíndricas y esféricas,
#para lo cual deben consultar sobre el uso de funciones trigonométricas en Python.
import numpy as np

rectangular = np.array([4,7,3])
xy = rectangular[0]**2+ rectangular[1]**2
radio =np.sqrt (xy)
theta = np.arctan(rectangular[1]/rectangular[0])        #resultado en radianes
zeta = rectangular[2]

xyz= rectangular[0]**2 + rectangular[1]**2 + rectangular[2]**2
r2 = np.sqrt(xyz)
theta2 = np.arctan(rectangular[1]/rectangular[0])       #np.arctan calcula la tangente inversa
phi = np.arccos(rectangular[2]/r2)                      #np.arcos calcula el coseno inverso


print(f'''las coordenadas rectangulares son: {rectangular}
las coordenadas cilindricas son: [{radio:.2f} {theta:.2f} {zeta}] este resultado esta en radianes
las coordenadas esfericas son: [{r2:.2f} {theta2:.2f} {phi:.2f}] este resultado esta en radianes
''')
```

```
PS C:\Users\ACER\Desktop\Universidad Ecci\Octavo Semestre\RoboticaIA\Primer corte\Taller1> & C:/U
s/Python/Python312/python.exe "c:/Users/ACER/Desktop/Universidad Ecci/Octavo Semestre/RoboticaIA/
"
las coordenadas rectangulares son: [4 7 3]
las coordenadas cilindricas son: [8.06 1.05 3] este resultado esta en radianes
las coordenadas esfericas son: [8.60 1.05 1.21] este resultado esta en radianes

PS C:\Users\ACER\Desktop\Universidad Ecci\Octavo Semestre\RoboticaIA\Primer corte\Taller1>
```

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

```python
#Realice un programa para el cálculo de la resistencia de una RTD de platino (PT100) en función de la temperatur
temp=80         #ingreso de temperatura

ro=100          # 100 ohms a 0 grados
coe_pt=0.385    # coeficiente platino
ar=coe_pt*temp
##{
##calculo de resistencia de RTD
##RT=ro+Δr
##Δr=(α+ro)*(Δt)
##%}
resistencia=ro+ar
print()
print(f'''El calculo de una resistencia de valor {temp} grados en funcion de la temperatura es {resistencia} apr
print()
```

```
las coordenadas rectangulares son: [4 7 3]
las coordenadas cilindricas son: [8.06 1.05 3] este resultado esta en radianes
las coordenadas esfericas son: [8.60 1.05 1.21] este resultado esta en radianes
```

5. 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
#Realice en funciones las rotaciones en X, Y y Z,
#donde se tenga un parámetro de entrada (ángulo) y un parámetro de salida (matriz).
import numpy as np
np.set_printoptions(precision=4) #para cambiar el numero de decimales sin cambiar tipo de variable

def Rx(angulo):
    grados = int(angulo)
    matriz_rotacion_x = np.array([
        [1, 0, 0],
        [0, np.cos(grados), -np.sin(grados)],
        [0, np.sin(grados), np.cos(grados)]
    ])
    return matriz_rotacion_x

def Ry(angulo):
    grados = int(angulo)
    matriz_rotacion_y = np.array([
        [np.cos(grados), 0, np.sin(grados)],
        [0, 1, 0],
        [-np.sin(grados), 0, np.cos(grados)]
    ])
    return matriz_rotacion_y

def Rz(angulo):
    radianes = int(angulo)
    matriz_rotacion_z = np.array([
        [np.cos(radianes), -np.sin(radianes), 0],
        [np.sin(radianes), np.cos(radianes), 0],
        [0, 0, 1]
    ])
    return matriz_rotacion_z

```

```python


angulo_x = 98
angulo_y = 65
angulo_z = 30


matriz_rotacion_x = Rx(angulo_x)
print()
print(f''''Matriz de rotacion en x:\n {matriz_rotacion_x}''')


matriz_rotacion_y = Ry(angulo_y)
print()
print(f'''La matriz de rotacion en y:\n {matriz_rotacion_y}
      ''')

matriz_rotacion_z = Rz(angulo_z)
print()
print(f'''La matriz de rotacion en z:\n {matriz_rotacion_z}''')
```

```
'Matriz de rotacion en x:
 [[ 1.      0.      0.    ]
 [ 0.     -0.8193  0.5734]
 [ 0.     -0.5734 -0.8193]]

La matriz de rotacion en y:
 [[-0.5625  0.      0.8268]
 [ 0.      1.      0.    ]
 [-0.8268  0.     -0.5625]]


La matriz de rotacion en z:
 [[ 0.1543  0.988   0.    ]
 [-0.988   0.1543  0.    ]
 [ 0.      0.      1.    ]]
PS C:\Users\ACER\Desktop\Universidad Ecci\Octavo Semestre\RoboticaIA\Prim
```

6. 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.

```python
#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
import numpy as np
embolo = 50                             # diametro embolo unidad en milimetros
vastago = 20                            # diametro del vastago debe ser menor a la del embolo mm
presiontrabajo = 6                      # unidad presion en bar

D=embolo*0.001                          #pasa de mm a metros el embolo
superficie= np.pi*(D**2/4)              #calcula superficie del embolo
presion= presiontrabajo*(10**5/1)       #pasa de bar a pascal
fuerza_avance=presion*superficie        #la fuerza de avance en newton

D1=vastago*0.001                        #pasa de mm a metros el vastago
superficie2= np.pi*(D1**2/4)            #calcula superficie del vastago
s=superficie-superficie2                #calcula area util
fuerza_retroceso=presion*s              #la fuerza de retroceso en newton
print()
print(f'''caracteristica del cilindro neumatico
      Embolo {embolo} mm.
      Vastago {vastago} mm.
      Presion de trabajo {presiontrabajo} bar.
      la fuerza de avance para el cilindro con estas caracteriticas es {fuerza_avance:.2f} Newtons.
      la fuerza de retroceso para el cilindro es {fuerza_retroceso:.2f} Newtons.''')
```

```
caracteristica del cilindro neumatico
      Embolo 50 mm.
      Vastago 20 mm.
      Presion de trabajo 6 bar.
      la fuerza de avance para el cilindro con estas caracteriticas es 1178.10 Newtons.
      la fuerza de retroceso para el cilindro es 989.60 Newtons.
PS C:\Users\ACER\Desktop\Universidad Ecci\Octavo Semestre\RoboticaIA\Primer corte\Taller1>
```

B. Con interacción de consola (fprintf o disp) y teclado (input)

1. Realice un programa que calcule la potencia que consume un circuito ingresando por teclado el valor de corriente y voltaje.

```python
voltaje = float(input("ingrese el voltaje: "))
corriente = float(input("ingrese la corriente: "))

potencia = voltaje ** corriente
print("la potencia es: ",potencia)

```

```
ingrese el voltaje: 5
ingrese la corriente: 9
la potencia es:  1953125.0
PS C:\Users\ACER\Desktop\Universidad Ecci\Octavo Semestre\RoboticaIA\Primer corte\Taller1>
```

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

```python
import random



inicial = int(input("numero inicial: "))
final = int(input("numero final: "))

resultado = random.randint(inicial,final)
print("el numero es: ",resultado)

```

```
numero inicial: 30
numero final: 65
el numero es:  55
PS C:\Users\ACER\Desktop\Universidad Ecci\Octavo Semestre\RoboticaIA\Primer corte\Taller1>
```

3. 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.

```python
import math
def pri(a, b, c):
    return a * b * c
def pir(a, b, c):
    return a * b * c / 3
def con(a, b, c):
    return 1/3 * math.pi * a *  b**2 + c**2 + b*c
def cil(a, b):
    return math.pi * a**2 * b

while True:
    print("escojer figura")
    print("1. prisma")
    print("2. piramide")
    print("3. cono truncado")
    print("4. cilindro")

    eleccion = input("escojer numero de la figura: ")
    n1 = float(input("numero 1: "))
    n2 = float(input("numero 2: "))
    n3 = float(input("numero 3: "))

    if eleccion=="1":
        print("el volumen del prisma es: ",pri(n1,n2,n3))
    elif eleccion=="2":
        print("el volumen de la piramide es: ",pir(n1,n2,n3))
    elif eleccion=="3":
        print("el volumen del cono truncado es: ",con(n1,n2,n3))
    elif eleccion=="4":
        print("el volumen del cilindro es: ", cil(n1,n2))
    else:
        print("opcion invalida")

```

```
escojer figura
1. prisma
2. piramide
3. cono truncado
4. cilindro
escojer numero de la figura: 3
numero 1: 5
numero 2: 6
numero 3: 2
el volumen del cono truncado es:  204.49555921538757
```

4. 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.

```python
while True:
    print("***** robots *****")
    print("1. cilindrico")
    print("2. carteciano")
    print("3. esferico")

    eleccion = input("escoje un robot: ")
    if eleccion == "1":
        print(" el robot es cilindrico")
        print(" tiene 1 articulacion de revolucion y 2 articulaciones prismáticas")
    elif eleccion == "2":
        print(" el robot es carteciano ")
        print( "tine 3 articulaciones prismáticas")
    elif eleccion == "3":
        print(" el robot es esferico ")
        print( "tine 3 articulaciones rotacionales")
    else:
        print("no valido el digito")
```

```
***** robots *****
1. cilindrico
2. carteciano
3. esferico
escoje un robot: 2
 el robot es carteciano
tine 3 articulaciones prismáticas
```

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

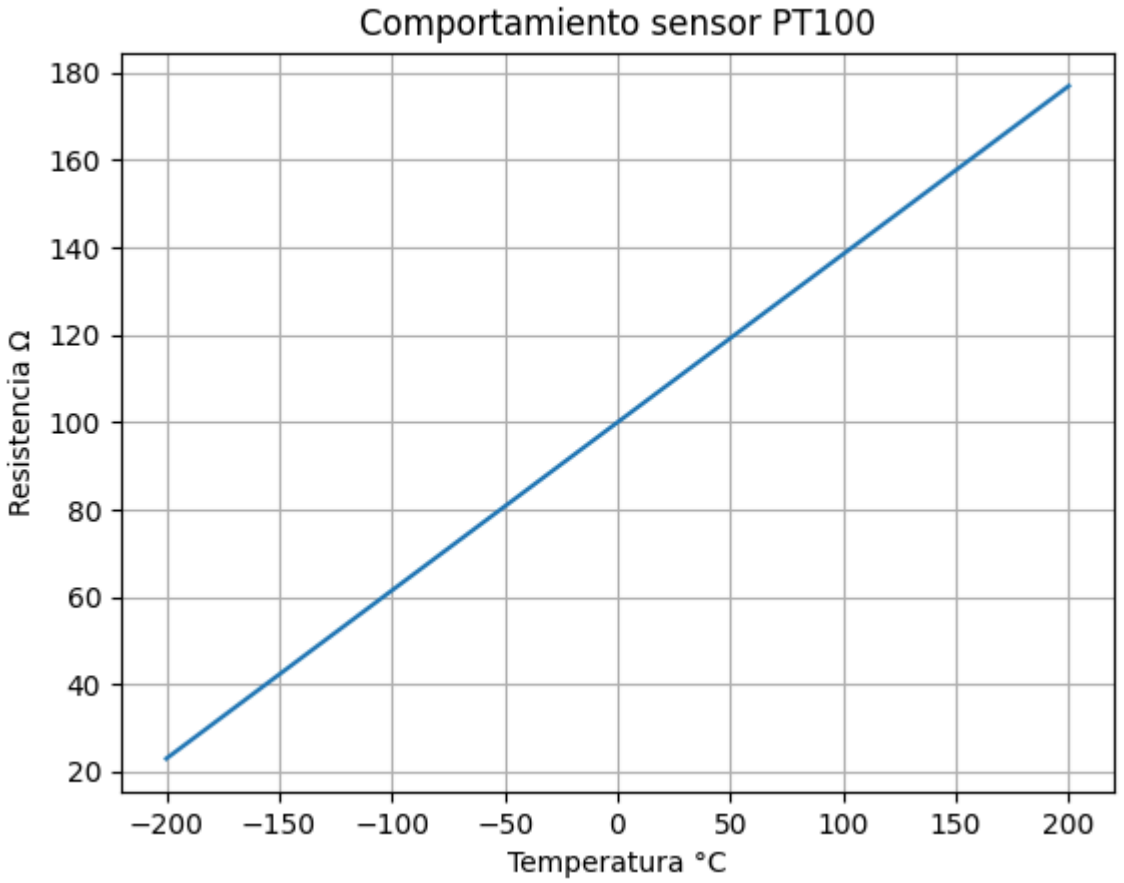
```python
palabra="si"

while palabra=="si":
    palabra=input("escribe si/no: ")
```

```
escribe si/no: si
escribe si/no: si
escribe si/no: si
escribe si/no: no
PS C:\Users\ACER\Desktop\Universidad Ecci\Octavo Semestre\RoboticaIA\Primer corte\Taller1>
```

C. Uso de las funciones para graficar

1. Realice un programa que grafique el comportamiento de un sensor PT100 desde -200°C a 200°C.

```python
#Realice un programa que grafique el comportamiento de un sensor PT100 desde -200°C a 200°C.
import numpy as np
from matplotlib import pyplot as plt

temp = np.arange (-200, 200.5, 0.5)       #ingreso de temperatura numpy.arange genera un conjunto de números en
ro=100                                    #100 ohms a 0 grados
coe_pt=0.385                              #coeficiente platino
ar=coe_pt*temp
resistencia=ro+ar
plt.title ('Comportamiento sensor PT100')
plt.ylabel('Resistencia Ω')
plt.xlabel('Temperatura °C')
plt.plot(temp,resistencia)
plt.grid(True)
plt.show()
```

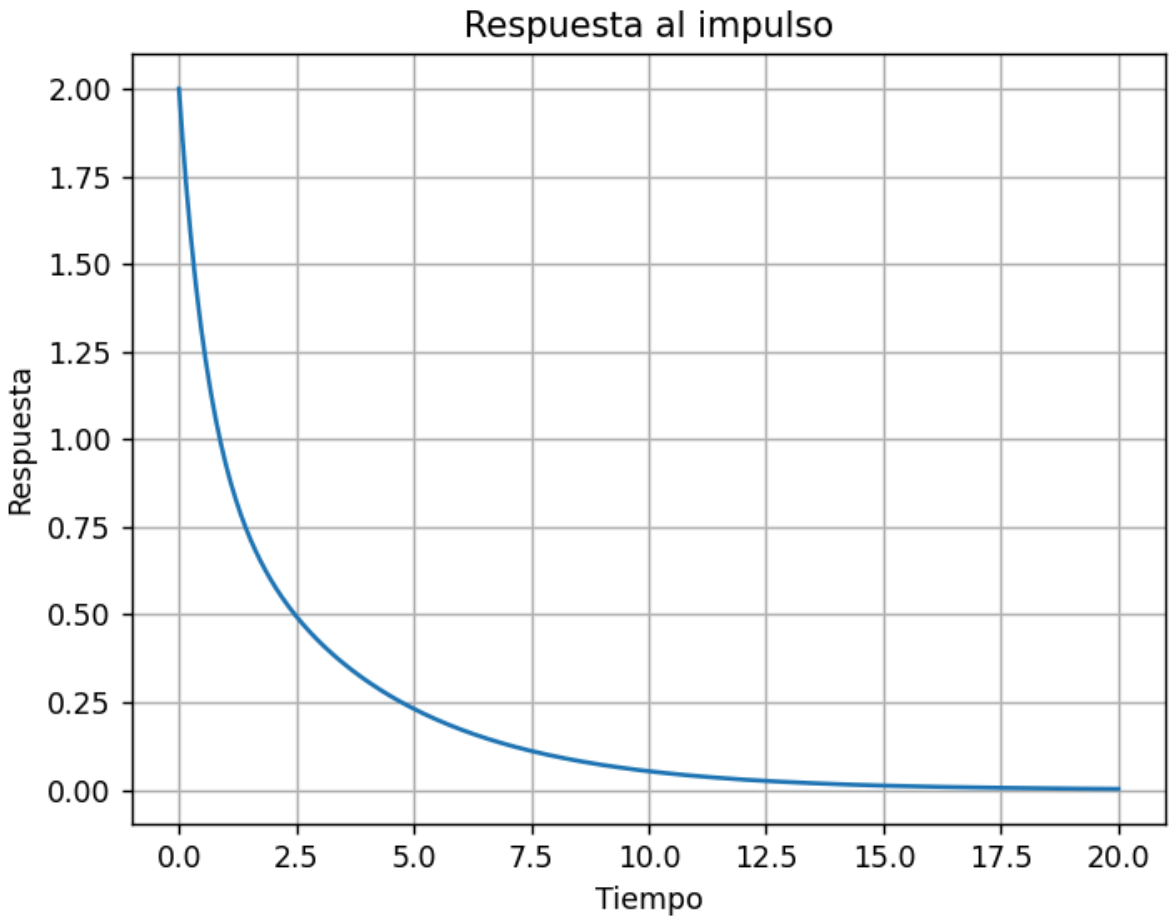


2. Realice un programa que le permita al usuario ingresar los coeficientes de una función de transferencia de segundo orden y graficar su comportamiento, además se debe mostrar que tipo de sistema es: subamortiguado, criticamente amortiguado y sobreamortiguado.

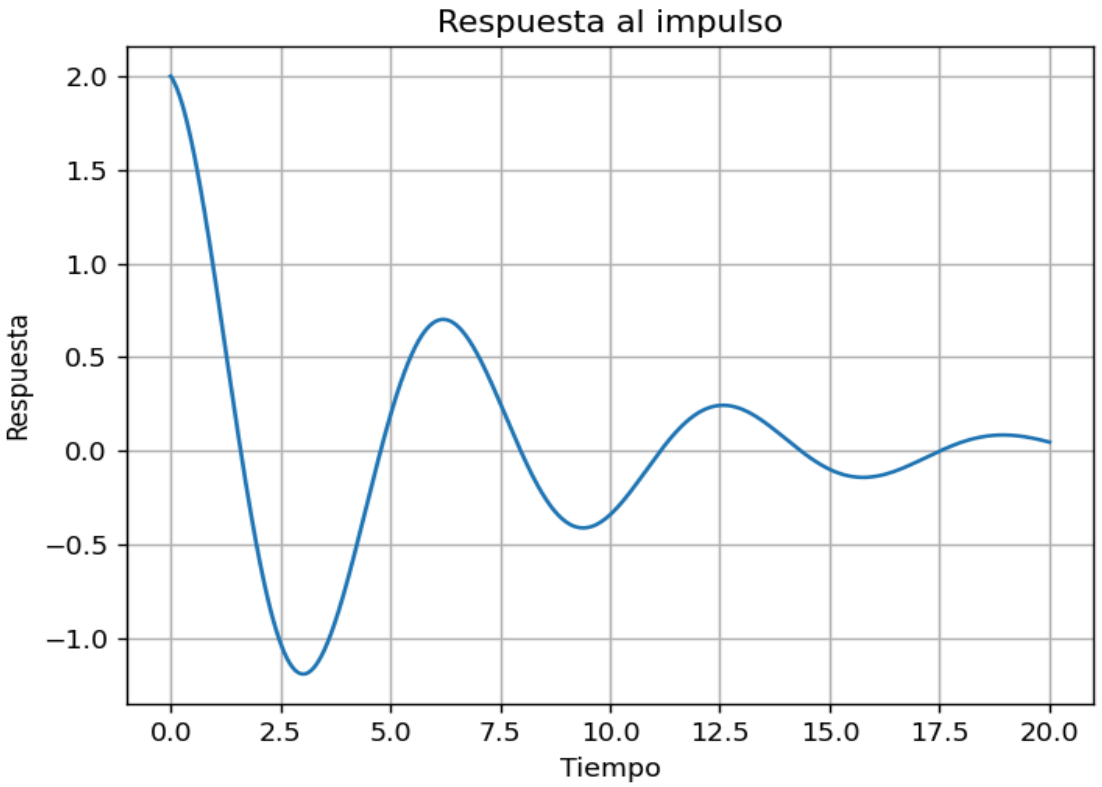
```python
import numpy as np
import matplotlib.pyplot as plt

def segundo_orden(coeficientes):
    num = coeficientes[0]
    den = coeficientes[1]
    return np.poly1d(num), np.poly1d(den)

def determinar_tipo_sistema(coeficientes):
    a1, a2 = coeficientes[1][1], coeficientes[1][2]
    discriminante = a1**2 - 4*a2

    if discriminante > 0:
        return "Sobreamortiguado"
    elif discriminante == 0:
        return "Críticamente amortiguado"
    else:
        return "Subamortiguado"

def graficar_respuesta_impulso(coeficientes):
    num_poly, den_poly = segundo_orden(coeficientes)
    t = np.linspace(0, 20, 1000)
    response = np.zeros_like(t)
    for i, time in enumerate(t):
        response[i] = np.sum(num_poly.coeffs * np.exp(den_poly.roots * time))

    plt.plot(t, response)
    plt.title("Respuesta al impulso")
    plt.xlabel("Tiempo")
    plt.ylabel("Respuesta")
    plt.grid(True)
    plt.show()
```

```python
# imprimir valores
def main():
    print("Ingrese los coeficientes de la función de transferencia de segundo orden:")
    num_coef = list(map(float, input("Ingrese los coeficientes del frecuencia natural: ").split()))
    den_coef = list(map(float, input("Ingrese los coeficientes del ganancia, factor de amortiguamiento separados por espacios: ").split()))

    coeficientes = [num_coef, den_coef]
    tipo_sistema = determinar_tipo_sistema(coeficientes)
    print("El sistema es:", tipo_sistema)

    graficar_respuesta_impulso(coeficientes)

if __name__ == "__main__":
    main()
```

```
PS D:\robotica_fabian\primer_corte> & D:/python/python.exe d:/robotica_fabian/primer_corte/tallerc3/tallerc2.py
Ingrese los coeficientes de la función de transferencia de segundo orden:
Ingrese los coeficientes del frecuencia natural: 1
Ingrese los coeficientes del ganancia, factor de amortiguamiento separados por espacios: 4 8 2
El sistema es: Sobreamortiguado
```



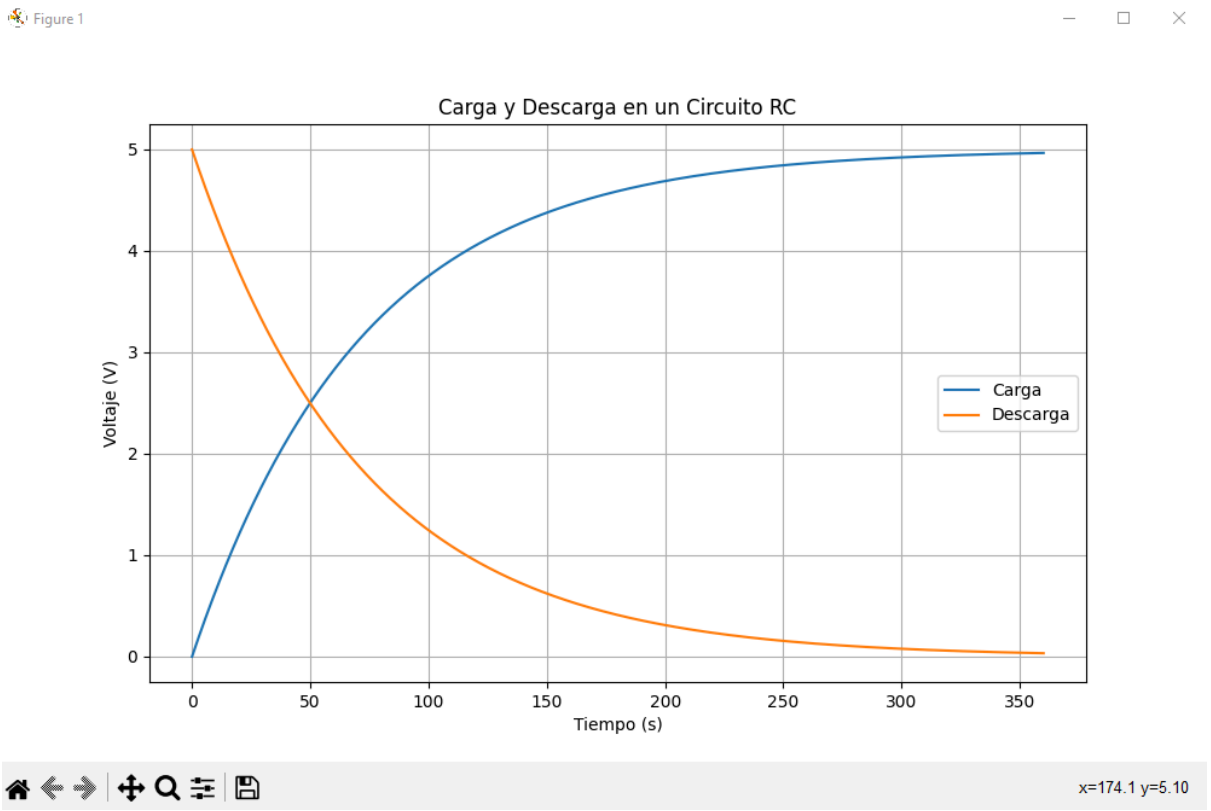

```
PS D:\robotica_fabian\primer_corte> & D:/python/python.exe d:/robotica_fabian/primer_corte/tallerc3/tallerc2.py
Ingrese los coeficientes de la función de transferencia de segundo orden:
Ingrese los coeficientes del frecuencia natural: 1
Ingrese los coeficientes del ganancia, factor de amortiguamiento separados por espacios: 3 1 3
El sistema es: Subamortiguado
d:\robotica_fabian\primer_corte\tallerc3\tallerc2.py:25: ComplexWarning: Casting complex values to real discards the imaginary part
  response[i] = np.sum(num_poly.coeffs * np.exp(den_poly.roots * time))
```



3. 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.

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

def carga_descarga(V, R, C, t):
    tau = R * C
    carga = V * (1 - np.exp(-t / tau))
    descarga = V * np.exp(-t / tau)
    return carga, descarga

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

# Definir el intervalo de tiempo
t = np.linspace(0, 5 * R * C, 1000)

# Calcular la carga y descarga
carga, descarga = carga_descarga(V, R, C, t)

# Graficar
plt.figure(figsize=(10, 6))
plt.plot(t, carga, label="Carga")
plt.plot(t, descarga, label="Descarga")
plt.title("Carga y Descarga en un Circuito RC")
plt.xlabel("Tiempo (s)")
plt.ylabel("Voltaje (V)")
plt.legend()
plt.grid(True)
plt.show()
```

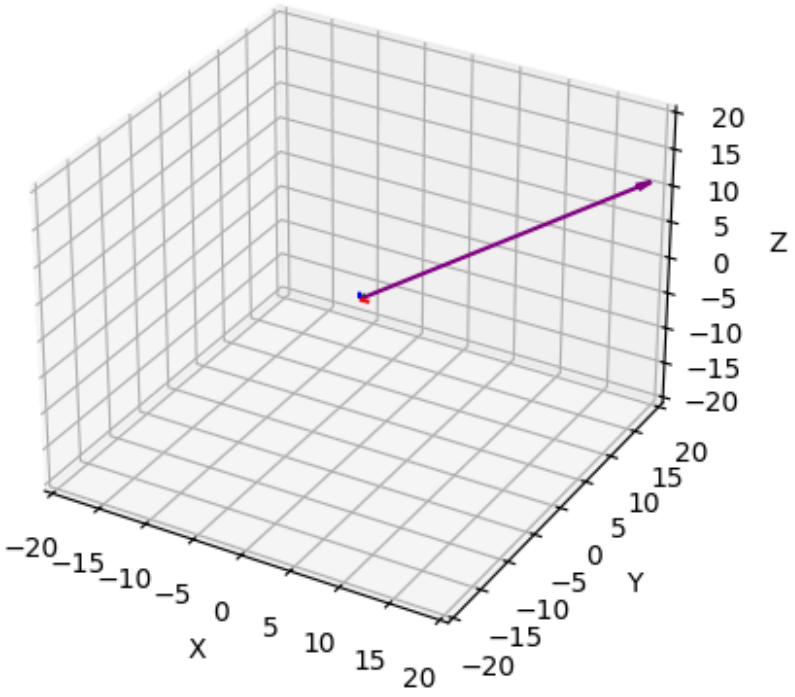
```
Ingrese el valor de voltaje (V): 5
Ingrese el valor de capacitancia (µF): 8
Ingrese el valor de resistencia (Ω): 9
```



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

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

# Dibujar el sistema de coordenadas
ax.quiver(0, 0, 0, 1, 0, 0, color='r', arrow_length_ratio=0.05)
ax.quiver(0, 0, 0, 0, 1, 0, color='g', arrow_length_ratio=0.05)
ax.quiver(0, 0, 0, 0, 0, 1, color='b', arrow_length_ratio=0.05)

# Dibujar el vector ingresado por el usuario
ax.quiver(0, 0, 0, x, y, z, color='purple', arrow_length_ratio=0.05)

# Configurar etiquetas
ax.set_xlabel('X')
ax.set_ylabel('Y')
ax.set_zlabel('Z')

# Configurar límites de los ejes
max_range = np.array([x, y, z]).max()
ax.set_xlim([-max_range, max_range])
ax.set_ylim([-max_range, max_range])
ax.set_zlim([-max_range, max_range])

plt.show()
```

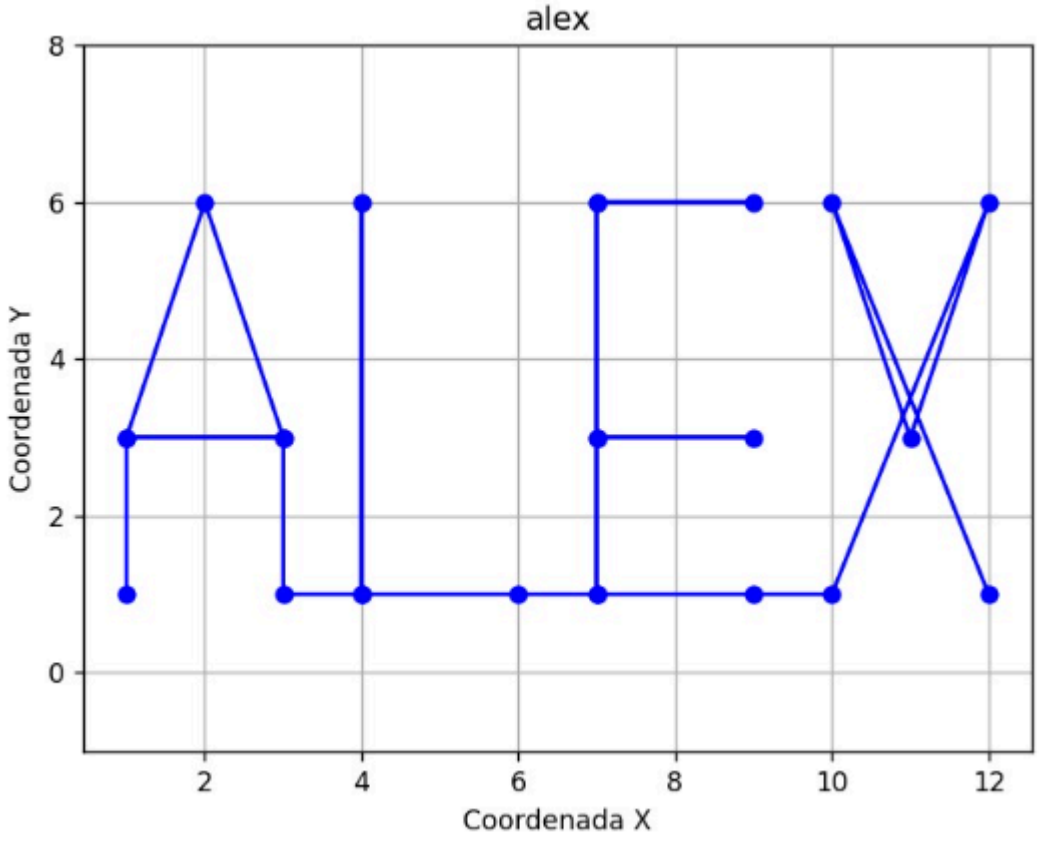
```
Ingrese la coordenada x del vector: 20
Ingrese la coordenada y del vector: 18
Ingrese la coordenada z del vector: 12
```

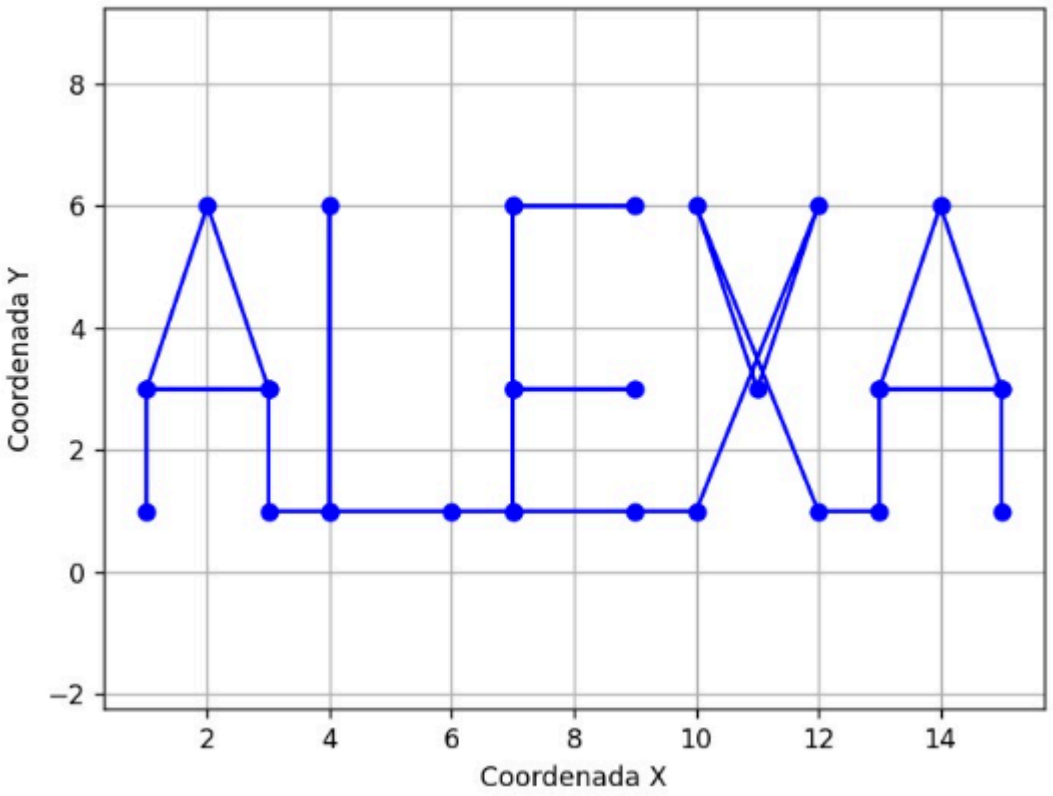


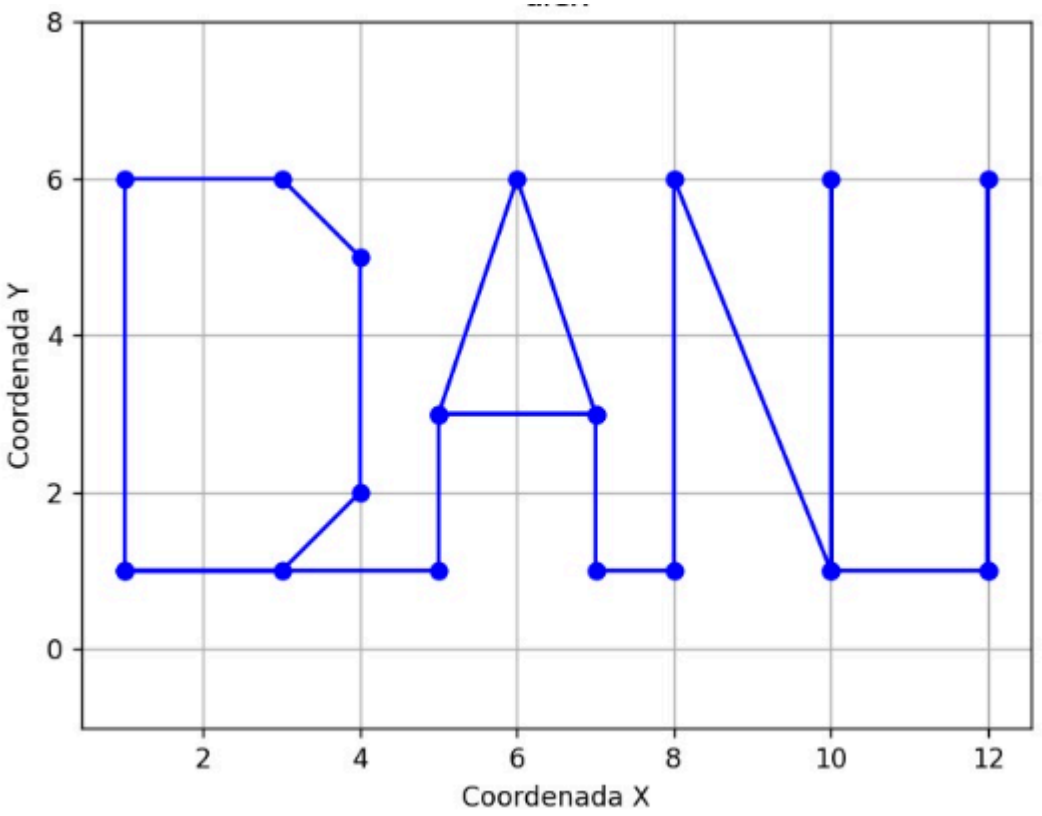
5. Dibuje el nombre de cada uno de los integrantes del grupo en un plot en 2D, teniendo en cuenta líneas rectas y/o curvas.

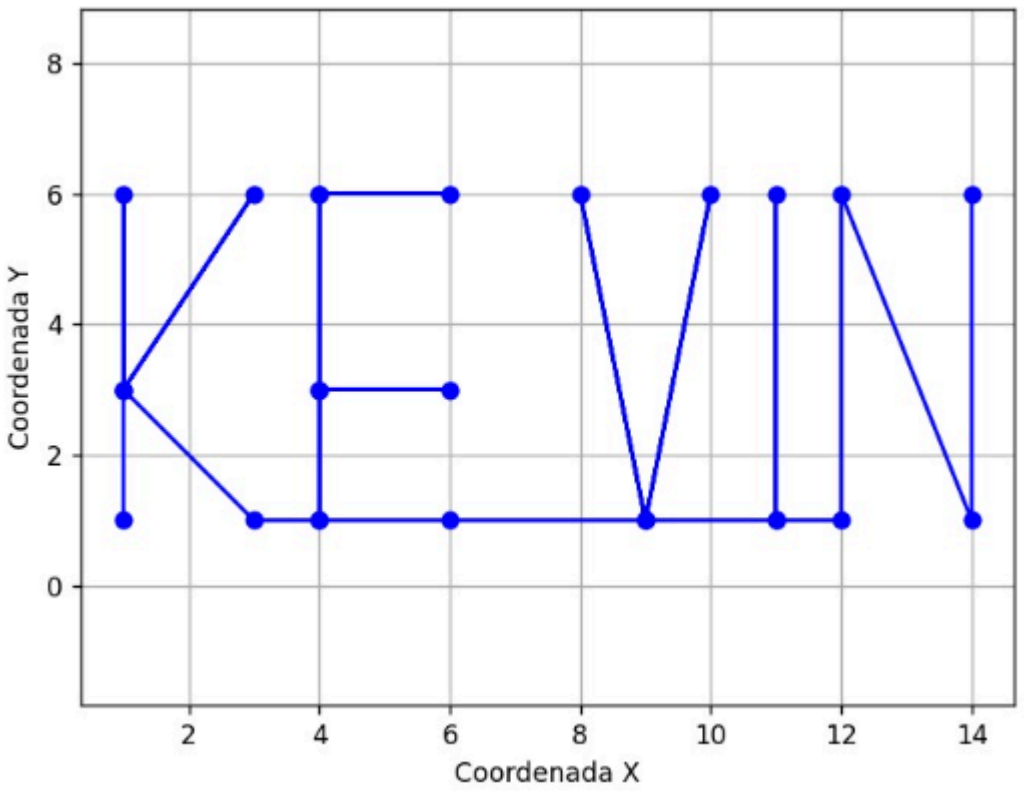
```python
import matplotlib.pyplot as plt

# cordenadas nombre alex
x = [1, 1, 2, 3, 1, 3, 3, 4, 4, 4, 6, 7, 7, 9, 7, 7, 9, 7, 7, 9, 10, 12, 11, 10, 12]
y = [1, 3, 6, 3, 3, 3, 1, 1, 6, 1, 1, 1, 3, 3, 3, 6, 6, 6, 1, 1, 1, 6, 3, 6, 1]

# cordenadas nombre alexa
x2 = [1, 1, 2, 3, 1, 3, 3, 4, 4, 4, 6, 7, 7, 9, 7, 7, 9, 7, 7, 9, 10, 12, 11, 10, 12, 13, 13, 14, 15, 13, 15, 15]
y2 = [1, 3, 6, 3, 3, 3, 1, 1, 6, 1, 1, 1, 3, 3, 3, 6, 6, 6, 1, 1, 1, 6, 3, 6, 1, 1, 3, 6, 3, 3, 3, 1]

## cordenadas nombre daniel
x3=[1,1,3,4,4,3,1,5,5,6,7,5,7,7,8,8,10,10,10,12,12,12]
y3=[1,6,6,5,2,1,1,1,3,6,3,3,3,1,1,6,1,6,1,1,6,1]

##cordenadas nombre kevin

x4=[1,1,1,3,1,3,4,4,6,4,4,6,4,4,6,9,8,9,10,9,11,11,11,12,12,14,14]
y4=[1,6,3,6,3,1,1,3,3,3,6,6,6,1,1,1,6,1,6,1,1,6,1,1,6,1,6]

# Grafica nombre alex
plt.plot(x, y, 'bo-')
plt.axis('equal')  # Establecer los ejes en la misma escala
plt.title('alex')
plt.xlabel('Coordenada X')
plt.ylabel('Coordenada Y')
plt.grid(True)
plt.show()

#grafica nombre alexa
plt.plot(x2, y2, 'bo-')
plt.axis('equal')  # Establecer los ejes en la misma escala
plt.title('alex')
```

```
plt.title('alex')
plt.xlabel('Coordenada X')
plt.ylabel('Coordenada Y')
plt.grid(True)
plt.show()

# Grafica nombre daniel
plt.plot(x3, y3, 'bo-')
plt.axis('equal')  # Establecer los ejes en la misma escala
plt.title('alex')
plt.xlabel('Coordenada X')
plt.ylabel('Coordenada Y')
plt.grid(True)
plt.show()

# Grafica nombre kevin
plt.plot(x4, y4, 'bo-')
plt.axis('equal')  # Establecer los ejes en la misma escala
plt.title('alex')
plt.xlabel('Coordenada X')
plt.ylabel('Coordenada Y')
plt.grid(True)
plt.show()
```




Coordenada Y
Coordenada X
8
6
4
2
0
−2
2
4
6
8
10
12
14


Coordenada Y
Coordenada X
8
6
4
2
0
2
4
6
8
10
12



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

```python
import cv2

img=cv2.imread('logo1.jpg')
imgbinary = cv2.Canny(img,10,50)
contornos, jerarquia =cv2.findContours(imgbinary, cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_NONE)
print(contornos)
cv2.drawContours(img,contornos,-1,(0,255,0),3)

cv2.imshow("imagen1",img)
cv2.imshow("escala gris",imgbinary)

cv2.waitKey(0)
cv2.destroyAllWindows()

```


```
[[494, 579]],
[[494, 580]],
[[494, 581]],
[[494, 582]],
[[494, 583]],
[[494, 584]],
[[494, 585]],
[[494, 586]],
[[494, 587]],
[[494, 588]],
[[494, 589]],
[[494, 590]],
[[494, 591]],
```

Código Github
https://github.com/DanielRquiroga