# Reporte – Cálculo de Centroide y Momento de Inercia

## Proyecto Grupo N.1

# Índice

# Introducción

Este proyecto se centra en el cálculo y la visualización de centroides y momentos de inercia, conceptos esenciales en diversas áreas de la ingeniería, especialmente en el diseño y análisis estructural. A través del desarrollo de una herramienta computacional, buscamos entender mejor estos conceptos y aplicarlos a figuras bidimensionales.

El propósito principal de nuestro proyecto es crear un software innovador que no solo calcule, sino que también visualice centroides y momentos de inercia para una variedad de figuras geométricas 2D, tanto simples como compuestas. Este proyecto tiene un enfoque educativo y práctico, donde se reforzara nuestros conocimientos teóricos sobre estos conceptos fundamentales, mejorar nuestras habilidades en programación y visualización de datos.

Al desarrollar este software, aspiramos a proporcionar una herramienta práctica y educativa que no solo facilite el cálculo de estos parámetros, sino que también permita su visualización gráfica, ayudando así a una mejor comprensión y aplicación en contextos reales.

# Objetivo del Proyecto

El objetivo principal de nuestro proyecto es crear un software que calcule y visualice centroides y momentos de inercia para diferentes figuras geométricas 2D, tanto simples como compuestas. Queremos usar este proyecto para reforzar nuestros conocimientos teóricos y mejorar nuestras habilidades en programación y visualización de datos.

- Desarrollar un algoritmo eficiente para el cálculo de centroides de figuras geométricas 2D simples, como triángulos, cuadrados y círculos. También calcular centroides de figuras geométricas 2D compuestas (combinaciones de figuras simples).
- Diseñar una interfaz de usuario intuitiva que permita a los usuarios introducir los parámetros de las figuras geométricas y visualizar los resultados de manera clara y comprensible.
- Programar la visualización gráfica de los centroides y momentos de inercia calculados, utilizando gráficos 2D para representar las figuras y sus propiedades.
- Implementar una sección de ayuda y documentación que explique los conceptos teóricos de centroides y momentos de inercia, así como instrucciones detalladas sobre el uso del software.

***Importancia de los Centroides y Momentos de Inercia***

- **Centroides:** El centroide representa el centro geométrico de una figura. Este punto es crucial para determinar las propiedades estáticas y para diseñar estructuras equilibradas y estables.
- **Momento de Inercia:** Este concepto mide la resistencia de un objeto a girar alrededor de un eje, lo cual es fundamental para analizar la rigidez y estabilidad de las estructuras, influenciando directamente su capacidad para soportar cargas y resistir deformaciones.

# Marco Teórico

***Centroide***

**Definición:**

El centroide de una figura geométrica es el punto que representa el centro de masa de la figura si esta fuera de un material homogéneo. En términos simples, es el "centro geométrico" de una figura.

**Propiedades:**

1. Ubicación: El centroide se encuentra en el punto de equilibrio de la figura, donde se puede considerar que toda la masa de la figura está concentrada.

2. Simetría: Para figuras simétricas, el centroide coincide con el centro de simetría.

3. Divisibilidad: El centroide de una figura compuesta se puede encontrar dividiendo la figura en partes más simples y utilizando el principio de superposición.

**Métodos de Cálculo:**

1. Cuadrado: El centroide de un cuadrado está en la intersección de sus diagonales, es decir, en el centro del cuadrado.

2. Rectángulo: El centroide de un rectángulo también está en la intersección de sus diagonales, en el centro geométrico del rectángulo.

3. Triángulo Rectángulo: El centroide se encuentra a un tercio de la distancia desde cada vértice hasta el punto medio del lado opuesto.

4. Triángulo Isósceles: El centroide está ubicado a un tercio de la altura desde la base hasta el vértice opuesto.

5. Círculo: El centroide de un círculo está en su centro geométrico.

6. Elipse: El centroide de una elipse está en su centro geométrico, en la intersección de sus ejes mayor y menor.

7. Cuarto de Círculo: El centroide se encuentra a una distancia de $\frac{4R}{3\pi}$ desde el vértice del ángulo recto hacia el centro del círculo.

8. Semicírculo: El centroide está ubicado a una distancia de $\frac{4R}{3\pi}$ desde el diámetro hacia el arco.

***Momento de Inercia***

**Definición:**

El momento de inercia es una medida de la resistencia de una figura a rotar alrededor de un eje. Es una propiedad fundamental en el análisis estructural y dinámico, ya que influye en la estabilidad y el comportamiento de las estructuras bajo cargas.

**Propiedades:**

1. Dependencia del Eje: El momento de inercia varía según el eje de rotación. Existen dos tipos principales: momento de inercia respecto a un eje perpendicular al plano de la figura y momento de inercia respecto a un eje en el plano de la figura.

2. Principio de Steiner (Teorema del Eje Paralelo): Permite calcular el momento de inercia respecto a cualquier eje paralelo a un eje que pasa por el centroide.

3. Aditividad: Para figuras compuestas, el momento de inercia total se puede calcular como la suma de los momentos de inercia de las partes componentes.

**Métodos de Cálculo:**

1. Cuadrado: El momento de inercia respecto a su eje central horizontal o vertical es $\bar{I} = \frac{1}{12} a^4$ , donde a es el lado del cuadrado.

2. Rectángulo: El momento de inercia respecto a su eje central horizontal es $\bar{I}_x = \frac{1}{12} bh^3$ , y respecto al eje central vertical es $\bar{I}_y = \frac{1}{12} hb^3$ , donde b es la base y h es la altura.

3. Triángulo Rectángulo: El momento de inercia respecto a la base es $\bar{I}_x = \frac{1}{36} bh^3$, y respecto al eje central vertical es $\bar{I}_y = \frac{1}{36} hb^3$ donde b es la base y h es la altura.

4. Triángulo Isósceles: El momento de inercia respecto a la base es $\bar{I}_x = \frac{1}{36} bh^3$ , y respecto al eje central vertical es $\bar{I}_y = \frac{1}{48} hb^3$ donde b es la base y h es la altura.

5. Círculo: El momento de inercia respecto a su eje central, $\bar{I}_x = \bar{I}_y = \frac{\pi}{4} R^4$ donde R es el radio.

6. Elipse: El momento de inercia respecto al eje mayor es $I_x = {}^1\!/_4\,\pi a b^3$, y respecto al eje menor es $I_y = {}^1\!/_4\,\pi b a^3$, donde a la mitad del ancho y b es la mitad de la altura.
7. Cuarto de Círculo: El momento de inercia respecto a los ejes perpendiculares que pasan por el vértice del ángulo recto es $I_x = I_y = 0.05488R^4$, donde R es el radio.
8. Semicírculo: El momento de inercia respecto a su eje central horizontal (que pasa por el diámetro) es $I_x = 0.1098R^4$, y respecto al eje vertical que pasa por el centroide es $I_y = {}^\pi\!/_8\,R^4$, donde R es el radio.

# Implementación del software

## Cálculo del Centroide:

Para los cálculos de centroide aplicamos las siguientes líneas de código, el programa es simple de entender por los nombres definidos de cada variable.

1. Centroide del triángulo: este calcula cualquier triangulo ya que usa el teorema del baricentro.

```
#Creación de funciones con respecto al calculo de los centroides por cada figura
def centro_triangulo(base,altura,puntos,hueco,direccion,nombre):
    x1, y1 = puntos[0]
    x2, y2 = puntos[1]
    x3, y3 = puntos[2]
    area = base * altura / 2
    area = ((-1)*area) if hueco == 1 else area
    x_centro = (x1 + x2 + x3) / 3
    y_centro = (y1 + y2 + y3) / 3
```

2. Centroide de cuadrado o rectángulo: ambos usan los mismos cálculos para obtener el centroide, pero la función centro cuadrado solo necesita un lado en este caso anchura.

```
def centro_cuadrado(anchura,x_inicial,y_inicial,hueco):
    area = anchura * anchura
    area = ((-1)*area) if hueco == 1 else area
    x_centro = x_inicial + anchura / 2
```

```
def centro_rectangulo(base,altura,x_inicial, y_inicial,hueco):
    area = base * altura
    area = ((-1)*area) if hueco == 1 else area
    x_centro = x_inicial + base / 2
    y_centro = y_inicial + altura / 2
```

3. Centroide del círculo:

```
def centro_circulo(radio,x_inicial, y_inicial,hueco):
    x_centro = x_inicial; y_centro = y_inicial
    area = pi * radio ** 2
    area = ((-1)*area) if hueco == 1 else area
    nombre = 'Circulo '+str(len(lista))
    control figuras.append([nombre,radio])
```

4. Centroide del semicírculo:

```
def centro_semicirculo(radio,x_inicial, y_inicial, direccion,hueco):
    area = (pi * (radio ** 2)) / 2
    area = ((-1)*area) if hueco == 1 else area
    control = (4 * radio) / (3 * pi)
    x_centro = x_inicial - control if direccion == 2 else x_inicial + cont
    y_centro = y_inicial + control if direccion == 0 else y_inicial - cont
    nombre = 'Semicirculo '+str(len(lista))
```

5. Centroide de un cuarto de circulo:

```
def centro_cuartocirculo(radio,x_inicial, y_inicial, cuadrante,hueco):
    area = (pi * radio ** 2) / 4
    area = (-1)*area if hueco == 1 else area
    control = (4 * radio) / (3 * pi)
    x_centro = x_inicial + control if cuadrante == 0 or cuadrante == 3 els
    y_centro = y_inicial + control if cuadrante == 0 or cuadrante == 1 els
    nombre = 'CuartoCirc'+str(len(lista))
```

6. Centroide de una elipse:

```
def centro_cuartocirculo(radio,x_inicial, y_inicial, cuadrante,hueco):
    area = (pi * radio ** 2) / 4
    area = (-1)*area if hueco == 1 else area
    control = (4 * radio) / (3 * pi)
    x_centro = x_inicial + control if cuadrante == 0 or cuadrante == 3 els
    y_centro = y_inicial + control if cuadrante == 0 or cuadrante == 1 els
    nombre = 'CuartoCirc'+str(len(lista))
```

Para calcular los datos de una figura compuesta, sumamos cada una de las áreas, x, y, área*x, área*y de cada una de las figuras, para ello utilizamos la función centroide general:

```
def centroide_general():
    sumatoria = ["Sumatoria", 0, "", "", 0, 0]
    for row in centroide:
        sumatoria[1] += row[1]
        sumatoria[4] += row[4]
        sumatoria[5] += row[5]
        """row.append(row[1] * row[2])
        row.append(row[1] * row[3])"""
    x_general = sumatoria[4]/sumatoria[1]
```

El método utilizado para encontrar el centroide de la figura compuesta es la función evento centroide

```
def event_centroide(canvas,text,tab):
    if len(centroide) < 1:
        mb.showinfo("Centroide","No existe figura para calcular el ce
        return None
    tab.set("Centroide")
    text.delete("0.0", "end")
    texto = ""
    #Limpieza de centroides en la grafica que ya estuvieron calculado
    try:
        for a in puntos_centroide:
            a.remove()
        puntos_centroide.clear()
```

**Cálculo del Momento de Inercia:**

Para los cálculos del momento de inercia aplicamos las siguientes líneas código (funciones) para las figuras simples.

1. Distancia área: utilizada para el calculo del momento de inercia de figuras compuestas:

```
#Funciones para el calculo de la inercia por figura
def a_Distancia(area, x_centro, y_centro, x_general, y_general):
    area_distanciaX = area * ((y_general - y_centro)**2)
    area_distanciaY = area * ((x_general - x_centro)**2)
    return area_distanciaX, area_distanciaY
```

2. Inercia Rectángulo: como cuadrado y rectángulo utilizan la misma inercia se usa una sola función.

```
def inercia_rectangulo( area, base, altura):
    inercia_centroX = (base * (altura**3)) / 12
    inercia_centroY = ((base**3) * altura) / 12
    if area < 0:
        inercia_centroX *= -1; inercia_centroY *= -1
    return inercia_centroX, inercia_centroY
```

3. Inercia Triangulo Rectángulo:

```
def inercia_Rectriangulo(area, base, altura):
    inercia_centroX = (base * (altura**3)) / 36
    inercia_centroY = ((base**3) * altura) / 36
    if area < 0:
        inercia_centroX *= -1; inercia_centroY *= -1
    return inercia_centroX, inercia_centroY
```

4. Inercia Triangulo Isósceles:

```
def inercia_Isotriangulo(area, base, altura, direccion):
    print(area,base,altura,direccion)
    if direccion == 0 or direccion == 1:
        inercia_centroX = (base * (altura**3)) / 36
        inercia_centroY = ((base**3) * altura) / 48
    else:
        inercia_centroX = ((base**3) * altura) / 48
```

5. Inercia del Círculo:

```
def inercia_circulo(area, radio):
    inercia_centroX = (pi * (radio**4)) / 4
    inercia_centroY = inercia_centroX
    if area < 0:
        inercia_centroX *= -1; inercia_centroY *= -1
    return inercia_centroX, inercia_centroY
```

6. Inercia del Semicírculo:

```
def inercia_semicirculo(area, radio, direccion):
    if direccion == 0 or direccion == 1:
        inercia_centroX = 0.1098 * (radio**4)
        inercia_centroY = (pi * (radio**4)) / 8
    else:
        inercia_centroX = (pi * (radio**4)) / 8
        inercia_centroY = 0.1098 * (radio**4)
    if area < 0:
```

7. Inercia del cuarto de circulo

```
def inercia_cuartocirculo(area, radio):
    inercia_centroX = 0.05488 * (radio**4)
    inercia_centroY = inercia_centroX
    if area < 0:
        inercia_centroX *= -1; inercia_centroY *= -1
    return inercia_centroX, inercia_centroY
```

8. Inercia de la Elipse:

```
def inercia_elipse(area, base, altura):
    inercia_centroX = (pi * (base / 2) * ((altura / 2)**3)) / 4
    inercia_centroY = (pi * ((base / 2)**3) * (altura / 2)) / 4
    if area < 0:
        inercia_centroX *= -1; inercia_centroY *= -1
    return inercia_centroX, inercia_centroY
```

Para calcular los datos de una figura compuesta, sumamos cada una de las inercias en X, área + distancia en X, inercia en Y, área + distancia en Y, de cada una de las figuras, para ello utilizamos la función centroide general:

```
def calcular_inercia(x_general,y_general): #Si compuesta es 1 verdadero,
    global inercias
    inercias = []
    sumatoria = ["Sumatoria", 0, "", "", 0, 0]
    sumatoria.extend([0, 0])
    if basic_compuesta == 1:
        sumatoria.extend([0, 0])
    for row, row2 in zip(centroide, control_figuras):
        if row[0][:3]=="RTr":
```

El método utilizado para encontrar el momento de inercia de la figura compuesta es la función evento inercia:

```
def event_inercia(canvas,text,eje,tab):
    print(centroide)
    if len(centroide) < 1:
        mb.showinfo("Inercia","No existe figura para calcular el mom
        return None
    tab.set("Momento de Inercia")
    text.delete("0.0", "end")
```

## Diseño de la Interfaz de Usuario:

Para la interfaz integramos varias funciones y utilizamos la librería customTkinter y Tkinter para diseñar la interfaz de botones:

```python
#Importar librerias
import customtkinter as ct
from tkinter import *
from tkinter import messagebox as mb
from PIL import Image
import matplotlib.pyplot as plt
from matplotlib.backends.backend_tkagg import FigureCanvasTkAgg
import matplotlib.patches as mpatches
from math import pi
import sys
```

1. Este segmento de código es el programa principal para la interfaz de usuario, donde se encuentran las funciones que muestran otros frame y widget como los botones de selección, botones de acción, espacios de entrada, impresiones de los gráficos de la figura, impresiones de texto y comandos tipo lambda para aplicar las acciones que se requieren realizar para utilizar la interfaz adecuadamente.

```python
root = ct.CTk()
root.geometry("800x350+400+200")
root.title("Proyecto N1 - Mecánica")
# Creación del grafico de matplotlib para despligue de las formas desarrollada
fig, ax = plt.subplots()
xi = 0; xf = 10; yi=0; yf=10
lista = []
centroide = []
lineas_momento = []
puntos_centroide = []
control_figuras = []
# Etiquetas de la ventana principal
ct.CTkLabel(root,image=logo,text="").place(x=100,y=130)
ct.CTkLabel(root,image=marca,text="").place(x=500,y=130)
ct.CTkLabel(root, text="¡Bienvenido \°O°/!", font=(font_texto, 20)).pack()
ct.CTkLabel(root, text="CÁLCULO DE CENTROIDE \n Y MOMENTO DE INERCIA",font=(font_titulo, 40)).pack()
ct.CTkLabel(root, text="Seleccione el tipo de figura", font=(font_texto, 20)).place(x=300, y=150)
#Creación del combobox para seleccion de la figura y lo que se desea calcular
opcion_figura = ct.CTkComboBox(root, values=["Figuras Básicas", "Figuras Compuestas"], width = 200, hei
                               button_color = 'green')
opcion_figura.place(x=300, y=200)
#Ultimos botones para distintas opciones de continuación en el programa
ct.CTkButton(root, text="Aceptar", command=lambda:event_aceptar(opcion_figura),width = 150, height=40),
```

2. La Función principal para la ventana: se integran los botones y acciones principales, de la pagina del programa.

```
def ventana_interfaz(app):
    #Frame para el desplegue del grafico de matplotlib
    grafico_frame = ct.CTkFrame(app,width=940,height=533,
    grafico_frame.place(x=0,y=0)
    ct.CTkLabel(grafico_frame,text='Gráfico Matplotlib',fc
    #Inserción del gráfico dentro del frame
```

3. Obtener impresión: es una función que trae a pantalla los valores de la tabla para identificar los datos de cada “problema” que se inserte, ya sea de figuras simples o compuestas.

```
def obtener_impresion(num):
    titulo_centroide_compuesto= ["Figura", "Area", "Posición X", "Posición Y", "Area_x", "Ar
    titulo_centroide_simple= ["Figura", "Area", "Posición X", "Posición Y", "Area_x", "Area_
    titulo = []
    if basic_compuesta == 0:
        titulo = titulo_centroide_simple
    else:
        titulo = titulo_centroide_compuesto
    # Seleccionar los encabezados solicitados
    encabezados = titulo[:num]
    # Seleccionar la información correspondiente de la lista centroide y aplicar formato a pa
    datos = []
    for fila in centroide:
```

4. Crear gráficos de figura: realiza la creación de la figura de la en la ventana gráfica de matplob.

```
def crear_graficos(canvas):
    ax.clear()
    ax.set_xlim(xi, xf)
    ax.set_ylim(yi, yf)
    for patch in lista:
        ax.add_patch(patch)
    # Agregar las anotaciones desde nombre_figuras
    for a in centroide:
        nombre = a[0]
        x = a[2]
        y = a[3]
        ax.annotate(nombre, (x, y), color='black', fontsize=7)
    plt.grid(True)
    canvas.draw()
```

5. Seleccionar figura: muestra un frame que ayuda al usuario a identificar que figura esta ingresando y controla los datos que se tienen que ingresar para la figura.

```
def event_figura_select(valor_seleccionado, insercion_frame,canvas,app):
    #Crear cuadros de texto para la inserción de puntos y otros parametros
    punto_xy = ct.CTkEntry(insercion_frame, placeholder_text="",width=80,height=30,
    punto_xy.place(x=360,y=220)
    ct.CTkLabel(insercion_frame,text="Punto x,y",font=(font_texto,25)).place(x=360,y
    parametro_1 = ct.CTkEntry(insercion_frame, placeholder_text="",width=80,height=3
    parametro_1.place(x=360,y=300)
    text_parametro1 = ct.CTkLabel(insercion_frame,text="",font=(font_texto,25))
```

## Metodología de Pruebas

- **Pruebas Unitarias:**

Verifican la correcta implementación de funciones individuales, como el cálculo del centroide y el momento de inercia. Por ejemplo, se prueba la función que calcula el centroide de un triángulo con datos conocidos y se verifica que el resultado sea correcto.

- **Pruebas de Integración:**

Aseguran que los diferentes módulos del software funcionen correctamente en conjunto. Se combinan pruebas de cálculo con pruebas de visualización gráfica para validar la coherencia de los resultados. Un ejemplo es calcular el momento de inercia de una forma compuesta y verificar que se visualice correctamente en la interfaz gráfica.

- **Pruebas de Regresión:**

Garantizan que nuevas modificaciones o adiciones de código no introduzcan errores en funcionalidades existentes. Se re-ejecutan pruebas unitarias y de integración tras cada actualización del software. Por ejemplo, después de modificar el algoritmo de cálculo, se vuelven a ejecutar todas las pruebas previas para asegurar que los resultados siguen siendo correctos.

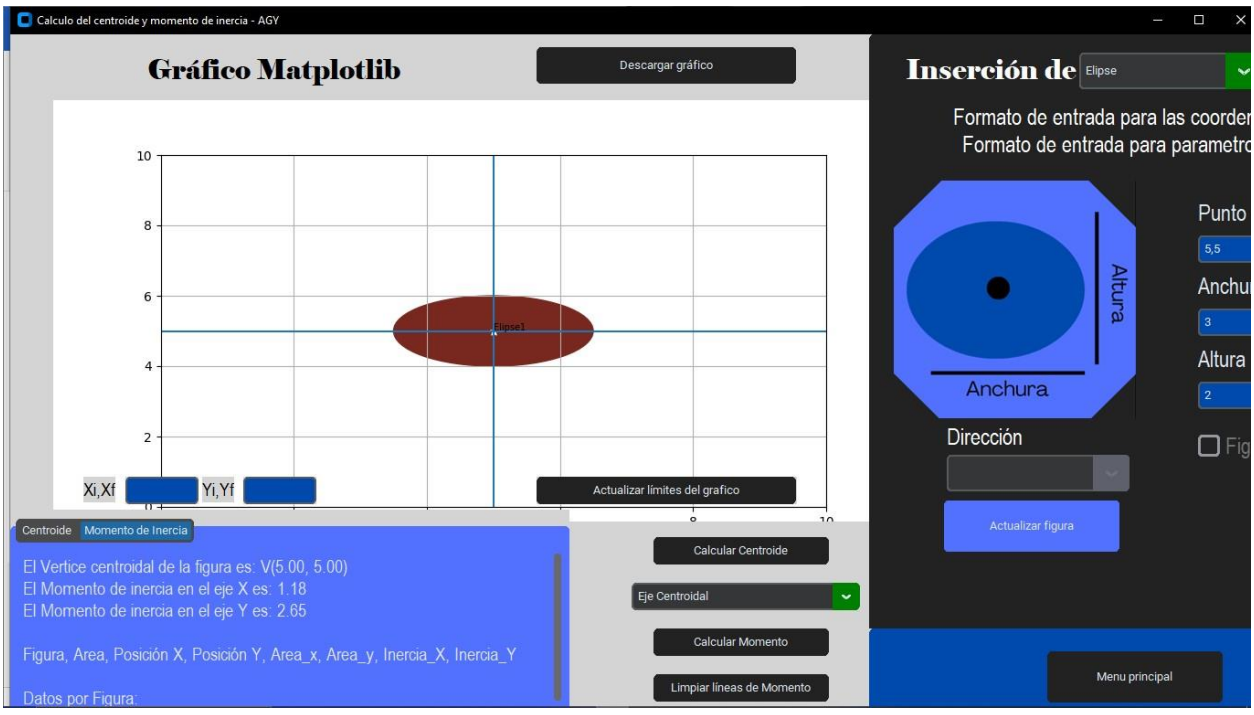
- **Pruebas de Interfaz de Usuario:**

Verifican la funcionalidad y usabilidad de la interfaz interactiva. Se simula la interacción del usuario con la interfaz para modificar figuras y observar cambios en tiempo real. Por ejemplo, se prueba que, al modificar una figura geométrica, la actualización de los cálculos y visualizaciones se realice sin errores.

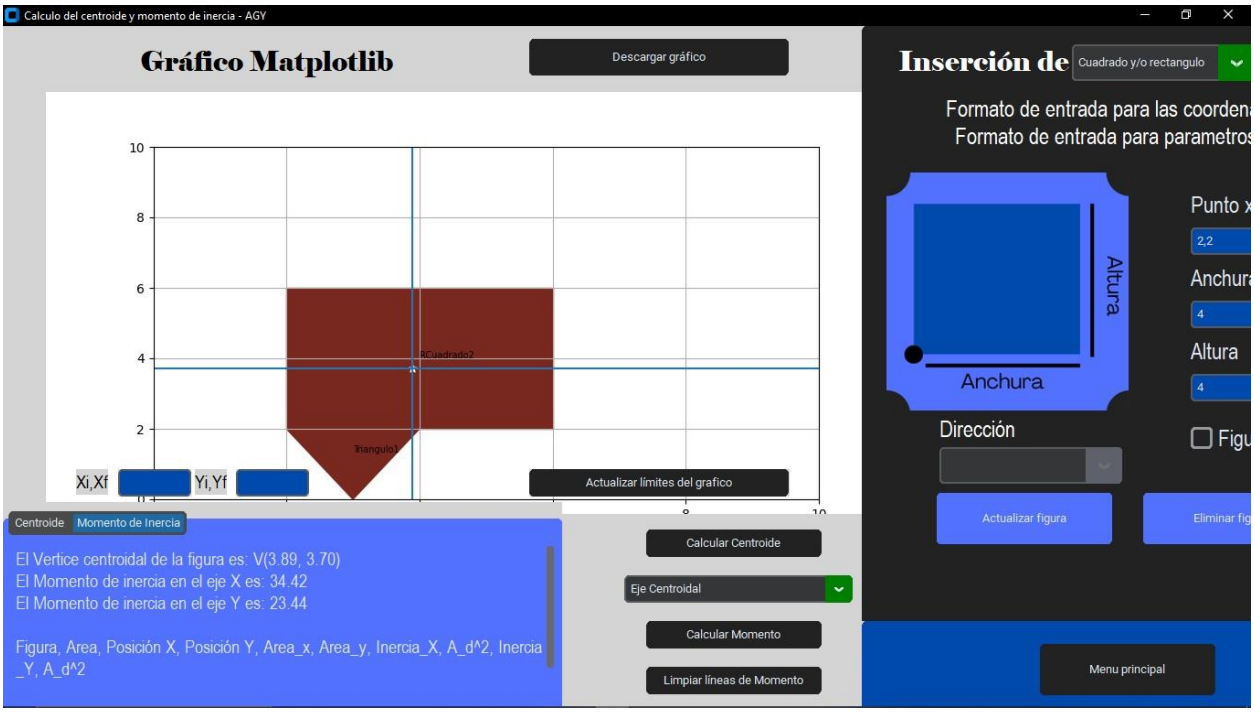
- **Pruebas de Rendimiento:**

Evalúan el desempeño del software bajo diferentes cargas de trabajo. Se realizan pruebas con figuras geométricas de distintas complejidades y tamaños. Por ejemplo, se mide el tiempo necesario para calcular y visualizar el momento de inercia de una figura compuesta grande.

**Pruebas en figuras simples (básicas) - muestra de ejemplos:**

**Pruebas de figuras compuestas - muestra de ejemplos:**


## Lenguaje de Programación

Para el desarrollo de nuestro programa destinado a calcular el centroide y el momento de inercia de figuras simples y compuestas en 2D, hemos seleccionado el lenguaje de programación Python. Esta elección se justifica por los siguientes beneficios y características que Python ofrece:

1. **Simplicidad y Facilidad de Uso:**

Python es conocido por su sintaxis clara y legible, lo cual facilita el desarrollo y mantenimiento del software. Esta característica es especialmente importante para garantizar que los usuarios puedan interactuar con la herramienta de manera intuitiva y eficiente.

2. **Bibliotecas y Herramientas Avanzadas:**

Python cuenta con una amplia gama de bibliotecas científicas y de visualización como NumPy, customTkinter, Tkinter, Pillow y Matplotlib, que son esenciales para realizar cálculos precisos del centroide y el momento de inercia, así como para generar visualizaciones gráficas detalladas de las figuras y sus propiedades.

3. **Desarrollo Rápido y Prototipado:**

La velocidad de desarrollo que permite Python es crucial para iterar rápidamente sobre el diseño y las funcionalidades de la herramienta. Esto nos permite implementar y probar nuevas características de manera ágil, respondiendo a las necesidades de los usuarios en tiempo real.

## Conclusión

El desarrollo de este proyecto nos ha permitido profundizar en los conceptos teóricos de centroides y momentos de inercia, así como aplicarlos de manera práctica a través de la creación de un software interactivo. A lo largo del proyecto, hemos conseguido varios objetivos clave:

1. **Comprensión Teórica:** Hemos adquirido un conocimiento sólido sobre cómo se definen y calculan los centroides y momentos de inercia para diversas figuras geométricas, tanto simples como compuestas. Entender estas propiedades es esencial para diversas aplicaciones en ingeniería, especialmente en el diseño y análisis estructural.
2. **Desarrollo de Algoritmos:** Hemos diseñado y implementado algoritmos eficientes para calcular el centroide y el momento de inercia. Estos algoritmos no solo manejan figuras básicas como triángulos, rectángulos y círculos, sino que también se extienden a formas compuestas, integrando múltiples sub-figuras.
3. **Implementación de Software:** Usando nuestras habilidades en programación, desarrollamos un software robusto que permite a los usuarios ingresar datos, realizar cálculos y visualizar los resultados de manera gráfica. Este software facilita el aprendizaje y la aplicación de los conceptos estudiados, proporcionando una herramienta útil tanto para estudiantes como para profesionales.
4. **Desarrollo de GUI:** Creamos una interfaz gráfica de usuario intuitiva y fácil de usar. La GUI permite la interacción dinámica, permitiendo a los usuarios modificar

las figuras y ver inmediatamente los resultados actualizados de los cálculos de centroides y momentos de inercia.

5. **Pruebas y Validación:** Realizamos pruebas exhaustivas para asegurar la precisión de nuestros cálculos y la funcionalidad del software. Los resultados fueron comparados con cálculos teóricos y valores de referencia, confirmando la exactitud y fiabilidad de nuestra herramienta.

Enfrentamos y superamos varios desafíos durante el desarrollo del proyecto, desde la correcta implementación de algoritmos hasta el diseño de una interfaz gráfica efectiva. Este proceso nos ha ayudado a mejorar nuestras habilidades en programación, resolución de problemas y trabajo en equipo.

## Referencias Bibliográficas


*Documentation Introduction | CustomTkinter*. (s. f.).

https://customtkinter.tomschimansky.com/documentation/ *matplotlib.patches*

*— Matplotlib 3.9.0 documentation*. (s. f.).

https://matplotlib.org/stable/api/patches_api.html

Pereira, R. (2023a). *Centroides – clases de mecánica*. clasesdemecanica.net.

https://clasesdemecanica.net/index.php/centroides/

Pereira, R. (2023b). *Momentos de inercia – clases de mecánica*. clasesdemecanica.net.

https://clasesdemecanica.net/index.php/momentos-de-inercia/

Sousa, A. L. (2020, 9 diciembre). *momentos de inercia círculoe elipse - Resistência dos Materiais I*. Passei Direto. https://www.passeidireto.com/arquivo/86423982/momentosde-inercia-circuloe-elipse

*Stack Overflow - where developers learn, share, & build careers*. (s. f.). Stack Overflow.

https://stackoverflow.com/ *tkinter.messagebox — Tkinter message prompts*. (s. f.). Python Documentation.

https://docs.python.org/3/library/tkinter.messagebox.html#messagebox-types

## Código Fuente

```
#Importar librerias import customtkinter as ct from tkinter
import * from tkinter import messagebox as mb from PIL import
Image import matplotlib.pyplot as plt from
matplotlib.backends.backend_tkagg import FigureCanvasTkAgg
import matplotlib.patches as mpatches from math import pi
import sys

#Seteo de la apariencia de la interfaz ct.set_appearance_mode("dark")

#Cambio de ruta ruta = "C://Users//HP//Desktop//python//ProyectoMecanica -
Alvarado,Guevara,Yee"
#Importación de imagenes y fuentes para la interfaz de usuario font_titulo =
'Elephant' font_texto = 'Arial Narrow' logo =
ct.CTkImage(dark_image=Image.open(ruta+"//logo.png"),size=(200, 150)) marca =
ct.CTkImage(dark_image=Image.open(ruta+"//marca.png"),size=(200, 150)) itriangulo
= ct.CTkImage(dark_image=Image.open(ruta+"//ITriangulo.png"),size=(265,
260)) rtriangulo =
ct.CTkImage(dark_image=Image.open(ruta+"//RTriangulo.png"),size=(265,
260)) rectangulo =
ct.CTkImage(dark_image=Image.open(ruta+"//Rectangulo.png"),size=(265,
260)) circulo = ct.CTkImage(dark_image=Image.open(ruta+"//Circulo.png"),size=(265,
260)) semicirculo =
ct.CTkImage(dark_image=Image.open(ruta+"//Semicirculo.png"),size=(265, 260))
cuarto_circulo = ct.CTkImage(dark_image=Image.open(ruta+"//Cuarto de
circulo.png"),size=(265, 260))
elipse = ct.CTkImage(dark_image=Image.open(ruta+"//Elipse.png"),size=(265, 260))
#Variables necesarias para ejecución del programa  figuras = ['Triangulo',
'RTriangulo', 'Cuadrado y/o rectangulo', 'Circulo',
'Semicirculo', 'Cuarto de area circulo', 'Elipse'] direcciones =
['Arriba', 'Abajo', 'Izquierda', 'Derecha'] cuadrantes = ['Cuadrante
1','Cuadrante 2','Cuadrante 3','Cuadrante 4'] colores =
["#78281F","#CD6155"]

#Creación de funciones con respecto a la validación de cadenas y/u otros valores,
impresiones también def validar_xy(entrada,puntos):
    entrada.replace(' ','') #Primero borramos todos los espacios en blanco de la
cadena
```

```
    try:
        partes = entrada.split(',') # Separar la entrada por la coma
if len(partes) != 2: # Verificar que haya exactamente dos partes
if puntos == True:
                mb.showerror("Error",("Ha ingresado las coordenadas del punto
x,y erroneamente, intentelo nuevamente"))             else:
                mb.showerror("Error",("Ha ingresado los parametros de la figura
erroneamente, intentelo nuevamente"))             return None,None
        # Intentar convertir ambas partes a
flotantes         x = float(partes[0])         y =
float(partes[1])         return x, y     except
ValueError:
        mb.showerror("Error",("Los valores ingresado no cumplen con la condición
de ser numericos intentelo nuevamente"))# Si ocurre una excepción al convertir a
flotante, la entrada no es válida         return None,None
 def
obtener_impresion(num,iner):
    titulo_centroide_compuesto = ["Figura", "Area", "Posición X", "Posición Y",
"Area_x", "Area_y", "Inercia_X", "A_d^2", "Inercia_Y", "A_d^2"]
titulo_centroide_simple = ["Figura", "Area", "Posición X", "Posición Y",
"Area_x", "Area_y", "Inercia_X", "Inercia_Y"]
titulo = []     if basic_compuesta == 0:
        titulo = titulo_centroide_simple
else:
        titulo = titulo_centroide_compuesto
# Seleccionar los encabezados solicitados
encabezados = titulo[:num]
    # Seleccionar la información correspondiente de la lista centroide y
aplicar formato a partir de la posición 1     datos = []     for i, fila in
enumerate(centroide):
        fila_formateada = [fila[0]]  # Agregar el primer elemento (Figura)
sin formato         for valor in fila[1:num]:
            if isinstance(valor, float):
                fila_formateada.append(f"{valor:.2f}")  # Formato de dos decimales
para floats             else:
                fila_formateada.append(valor)  # Mantener otros tipos de datos sin
cambios
```

```
        # Agregar los valores correspondientes de la lista inercias si no
está vacía         if iner and i < len(iner):
            inercia_valores = iner[i][:num]  # Tomar los primeros 'num' valores
de la lista de inercias             for valor in inercia_valores:
if isinstance(valor, float):
                    fila_formateada.append(f"{valor:.2f}")  # Formato de
dos decimales para floats                 else:
                    fila_formateada.append(valor)  # Mantener otros tipos de datos
sin cambios         datos.append(fila_formateada)
    # Formatear la salida como una cadena de texto     resultado_str =
"\n"     resultado_str += ", ".join(encabezados) + "\n\nDatos por
Figura:\n"     for fila in datos:         resultado_str += ",
".join(map(str, fila)) + "\n"     return resultado_str

#Creación de funciones con respecto al calculo de los centroides por cada figura
def centro_triangulo(base,altura,puntos,hueco,direccion,nombre):
    x1, y1 = puntos[0]     x2, y2 = puntos[1]     x3, y3 = puntos[2]     area =
base * altura / 2     area = ((-1)*area) if hueco == 1 else area     x_centro =
(x1 + x2 + x3) / 3     y_centro = (y1 + y2 + y3) / 3     nombre +=
str(len(lista))     control_figuras.append([nombre,base, altura, direccion])
centroide.append([nombre,area,x_centro,y_centro,area*x_centro,area*y_centro])
 def
centro_cuadrado(anchura,x_inicial,y_inicial,hueco):
    area = anchura * anchura     area = ((-
1)*area) if hueco == 1 else area     x_centro
= x_inicial + anchura / 2     y_centro =
y_inicial + anchura / 2     nombre =
'RCuadrado'+str(len(lista))
    control_figuras.append([nombre,anchura, anchura])
    centroide.append([nombre,area,x_centro,y_centro,area*x_centro,area*y_centro])
     def centro_rectangulo(base,altura,x_inicial,
y_inicial,hueco):
    area = base * altura
    area = ((-1)*area) if hueco == 1 else area
x_centro = x_inicial + base / 2     y_centro =
y_inicial + altura / 2
```

```
    nombre = 'Rectangulo '+str(len(lista))
control_figuras.append([nombre,base, altura])
centroide.append([nombre,area,x_centro,y_centro,area*x_centro,area*y_centro])
 def centro_circulo(radio,x_inicial,
y_inicial,hueco):
    x_centro = x_inicial; y_centro = y_inicial     area = pi * radio ** 2
area = ((-1)*area) if hueco == 1 else area     nombre = 'Circulo
'+str(len(lista))      control_figuras.append([nombre,radio])
centroide.append([nombre,area,x_centro,y_centro,area*x_centro,area*y_centro])
 def centro_semicirculo(radio,x_inicial, y_inicial,
direccion,hueco):
    area = (pi * (radio ** 2)) / 2     area = ((-1)*area) if hueco == 1 else
area     control = (4 * radio) / (3 * pi)     x_centro = x_inicial - control if
direccion == 2 else x_inicial + control if direccion == 3 else x_inicial
y_centro = y_inicial + control if direccion == 0 else y_inicial - control if
direccion == 1 else y_inicial     nombre = 'Semicirculo '+str(len(lista))
control_figuras.append([nombre,radio,direccion])
centroide.append([nombre,area,x_centro,y_centro,area*x_centro,area*y_centro])
 def centro_cuartocirculo(radio,x_inicial, y_inicial,
cuadrante,hueco):
    area = (pi * radio ** 2) / 4     area = (-1)*area if hueco == 1 else area
control = (4 * radio) / (3 * pi)     x_centro = x_inicial + control if
cuadrante == 0 or cuadrante == 3 else x_inicial - control     y_centro =
y_inicial + control if cuadrante == 0 or cuadrante == 1 else y_inicial -
control     nombre = 'CuartoCirc'+str(len(lista))
control_figuras.append([nombre,radio])
centroide.append([nombre,area,x_centro,y_centro,area*x_centro,area*y_centro])
 def centro_elipse(base, altura ,x_inicial,
y_inicial,hueco):
    x_centro = x_inicial; y_centro = y_inicial
area = pi * (base/2) * (altura/2)     area =
((-1)*area) if hueco == 1 else area     nombre
= 'Elipse'+str(len(lista))
control_figuras.append([nombre,base,altura])
    centroide.append([nombre,area,x_centro,y_centro,area*x_centro,area*y_centro])
 def
centroide_general():
    sumatoria = ["Sumatoria", 0, "", "", 0, 0]
for row in centroide:
```

```
sumatoria[1] +=
row[1]
sumatoria[4] +=
row[4]
sumatoria[5] +=
row[5]
x_general =
sumatoria[4]/su
matoria[1]
y_general =
sumatoria[5]/su
matoria[1]
            #Sumatoria       de       areas,
xcentroidegeneral,         ycentroidegeneral
return sumatoria,x_general, y_general
#Funciones para el calculo de la
inercia por figura def
a_Distancia(area, x_centro, y_centro,
x_general, y_general):
    area_distanciaX = area *
((y_general - y_centro)**2)
area_distanciaY = area * ((x_general
- x_centro)**2)     return
area_distanciaX, area_distanciaY
 def
inercia_Rectriangulo(
area, base, altura):
inercia_centroX =
(base * (altura**3))
/ 36
inercia_centroY =
((base**3) * altura)
/ 36     if area < 0:
        inercia_centroX *= -1;
inercia_centroY *= -1     return
inercia_centroX, inercia_centroY
 def
inercia_Isotriangulo(area,
base, altura, direccion):
if direccion == 0 or direccion
== 1:
        inercia_centroX = (base
* (altura**3)) / 36
inercia_centroY = ((base**3) *
altura) / 48      else:
```

```
        inercia_centroX =
((base**3) * altura) / 48
inercia_centroY = (base *
(altura**3)) / 36     if
area < 0:
        inercia_centroX *= -1;
inercia_centroY *= -1     return
inercia_centroX, inercia_centroY
 def
inercia_rectangulo(
area, base, altura):
inercia_centroX =
(base * (altura**3))
/ 12
inercia_centroY =
((base**3) * altura)
/ 12     if area < 0:
        inercia_centroX *= -1;
inercia_centroY *= -1     return
inercia_centroX, inercia_centroY
 def
inercia
_circul
o(area,
radio):

inercia_centroX =
(pi * (radio**4))
/ 4
inercia_centroY =
inercia_centroX
if area < 0:
        inercia_centroX *= -1;
inercia_centroY *= -1     return
inercia_centroX, inercia_centroY
 def
inercia_semicirculo(ar
ea, radio, direccion):
```

```
    if direccion == 0 or direccion == 1:
        inercia_centroX = 0.1098 * (radio**4)
inercia_centroY = (pi * (radio**4)) / 8
else:
        inercia_centroX = (pi *
(radio**4)) / 8         inercia_centroY
= 0.1098 * (radio**4)     if area < 0:
        inercia_centroX *= -1; inercia_centroY *= -
1     return inercia_centroX, inercia_centroY
 def inercia_cuartocirculo(area,
radio):     inercia_centroX =
0.05488 * (radio**4)
inercia_centroY = inercia_centroX
if area < 0:
        inercia_centroX *= -1; inercia_centroY *= -
1     return inercia_centroX, inercia_centroY
 def inercia_elipse(area, base,
altura):
    inercia_centroX = (pi * (base / 2) * ((altura /
2)**3)) / 4     inercia_centroY = (pi * ((base / 2)**3) *
(altura / 2)) / 4     if area < 0:
        inercia_centroX *= -1; inercia_centroY *= -
1     return inercia_centroX, inercia_centroY
 def calcular_inercia(x_general,y_general): #Si compuesta es 1
verdadero, si compuesta 0 es falso     global inercias     inercias
= []     sumatoria = ["Sumatoria", 0, "", "", 0, 0]
sumatoria.extend([0, 0])     if basic_compuesta == 1:
sumatoria.extend([0, 0])     for row, row2 in zip(centroide,
control_figuras):         if row[0][:3]=="RTr":
            inercia_ejex,  inercia_ejey  =  inercia_Rectriangulo(row[1],
row2[1], row2[2])         elif row[0][:3]=="Tri":
            inercia_ejex, inercia_ejey = inercia_Isotriangulo(row[1],
row2[1], row2[2], row2[3])         elif row[0][:3]=="Rec" or
row[0][:3] == "RCu":
            inercia_ejex,  inercia_ejey  =  inercia_rectangulo(row[1],
row2[1], row2[2])         elif row[0][:3]=="Cir":
            inercia_ejex, inercia_ejey = inercia_circulo(row[1],
row2[1])         elif row[0][:3]=="Sem":
            inercia_ejex,  inercia_ejey  =  inercia_semicirculo(row[1],
row2[1], row2[2])
```

```
        elif row[0][:3]=="Cua":
            inercia_ejex, inercia_ejey = inercia_cuartocirculo(row[1], row2[1])
elif row[0][:3]=="Eli":
            inercia_ejex, inercia_ejey = inercia_elipse(row[1], row2[1], row2[2])
                if
basic_compuesta == 1:
            #Obtiene el area del eje inercia x o y             a_distanciax ,
a_distanciay = a_Distancia(row[1], row[2], row[3], x_general, y_general)
            #Se crean las nuevas columnas para los momentos de
inercia             inercias.append([inercia_ejex, a_distanciax,
inercia_ejey, a_distanciay])         else:
            inercias.append([inercia_ejex, inercia_ejey])
for row in inercias:
        sumatoria[6] += row[0]
sumatoria[7] += row[1]         if
basic_compuesta == 1:
sumatoria[8] += row[2]
sumatoria[9] += row[3]
     return
sumatoria

#Creación de funciones con respecto a la manipulación de los graficos que muestran
las figuras def descargar_plt():
    dialog = ct.CTkInputDialog(text="Ingrese el nombre del grafico:",
title="Descargar grafico")     text = dialog.get_input()  # waits for
input     try:
        fig.savefig(ruta+'//'+text+'.png')         mb.showinfo("Descargar
grafico","Se descargo correctamente el grafico")     except:
        mb.showerror("Error","No se pudo descargar el grafico")
 def setearlimites(limitex, limitey,
canvas):
    global xi, xf, yi, yf     cadenax = limitex.get()
cadenay = limitey.get()     xi,xf = validar_xy(cadenax,True)
yi,yf = validar_xy(cadenay,True)     if xi is None or xf is
None or yi is None or yf is None:
        return None
else:
        ax.set_xlim(xi, xf)
ax.set_ylim(yi, yf)
canvas.draw()
```

```
        limitex.delete(0,10); limitey.delete(0,10) #Limpia los valores que ya
fueron colocados en el Entry
 def
crear_graficos(canvas):
    ax.clear()
ax.set_xlim(xi, xf)
ax.set_ylim(yi, yf)
for patch in lista:
        ax.add_patch(patch)
    # Agregar las anotaciones desde nombre_figuras     for a in centroide:
nombre = a[0]         x = a[2]         y = a[3]
ax.annotate(nombre, (x, y), color='black', fontsize=7)     plt.grid(True)
canvas.draw()     if basic_compuesta == 0: #Limpia los datos, si la opcion
colocada fue de figuras basiccas         lista.clear()         if
len(centroide) > 1:             control = centroide[-1]
centroide.clear()             centroide.append(control)
control = control_figuras[-1]             control_figuras.clear()
            control_figuras.append(control)
 def forma_itriangulo(punto, base, altura, direccion, value_hueco,
canvas):
    x,y = validar_xy(punto.get(),True)
cadena_parametro = str(base.get())+','+str(altura.get())
base,altura = validar_xy(cadena_parametro,False)
id_hueco = int(value_hueco.get())     try:
        direc = direcciones.index(direccion.get())
except:
        direc = 0     if x is None or y is None or base is
None or altura is None:
        return None
else:
        puntos = []
puntos.append((x,y))
        if direc == 0: #direcciones = ['Arriba', 'Abajo', 'Izquierda',
'Derecha']             x2 = x + base; x3 = x + (base/2); y3 = y + altura;
puntos.append((x2,y)); puntos.append((x3,y3))         elif direc == 1:
```

```
            x2 = x + base; x3 = x + (base/2); y3 = y -
altura; puntos.append((x2,y)); puntos.append((x3,y3))
elif direc == 2:
            y2 = y + base; x3 = x - altura; y3 = y +
(base/2); puntos.append((x,y2)); puntos.append((x3,y3))
elif direc == 3:
            y2 = y + base; x3 = x + altura; y3 = y + (base/2);
puntos.append((x,y2)); puntos.append((x3,y3))         triangulo_forma
= plt.Polygon(puntos,color=colores[id_hueco])
lista.append(triangulo_forma)
centro_triangulo(base,altura,puntos,id_hueco,direc,'Triangulo')
crear_graficos(canvas)     return triangulo_forma
 def forma_rtriangulo (punto, base, altura, direc, value_hueco,
canvas):
    x,y = validar_xy(punto.get(),True)
cadena_parametro = str(base.get())+','+str(altura.get())
base,altura = validar_xy(cadena_parametro,False)
id_hueco = int(value_hueco.get())     try:
        direccion = cuadrantes.index(direc.get())
except:
        direccion = 0     if x is None or y is None or base is
None or altura is None:
        return None
else:
        puntos = []; puntos.append((x,y))
if direccion == 0:
            vertice2 = (x, y + altura) ; vertice3 = (x + base, y)
elif direccion == 1:
            vertice2 = (x, y + altura) ; vertice3 = (x - base, y)
elif direccion == 2:
            vertice2 = (x, y - altura) ; vertice3 = (x - base, y)
elif direccion == 3:
            vertice2 = (x, y - altura) ; vertice3 = (x + base, y)
puntos.append(vertice2); puntos.append(vertice3)
        rtriangulo_forma = plt.Polygon(puntos,color=colores[id_hueco])
lista.append(rtriangulo_forma)
        centro_triangulo(base,altura,puntos,id_hueco,direccion,'RTriangulo')
crear_graficos(canvas)


def forma_cuadro_rec(punto,anchura,altura,value_hueco,canvas):
x,y = validar_xy(punto.get(),True)
    cadena_parametro = str(anchura.get())+','+str(altura.get())
anchura,altura = validar_xy(cadena_parametro,False)     id_hueco
= int(value_hueco.get())     if x is None or y is None or anchura
is None or altura is None:
```

```
        return None
else:
        cuadro_rec =
plt.Rectangle((x,y),width=anchura,height=altura,color=colores[id_hueco]) #Lo
que realmente forma el cuadrado o rectangulo         lista.append(cuadro_rec)
if anchura == altura:
            centro_cuadrado(anchura,x,y,id_hueco)
else:
            centro_rectangulo(anchura,altura,x,y,id_hueco)
crear_graficos(canvas)
 def
forma_circulo(punto,radio,value_hueco,canvas):
    x,y = validar_xy(punto.get(),True)
cadena_parametro = str(radio.get())+',0'
radio,not_use = validar_xy(cadena_parametro,False)
id_hueco = int(value_hueco.get())     if x is None or
y is None or radio is None:
        return None
else:
        circulo =
plt.Circle((x,y),radio,color=colores[id_hueco])
lista.append(circulo)         centro_circulo(radio,x,
y,id_hueco)         crear_graficos(canvas)
 def
forma_cuarto_circulo(punto,radio,direc,value_hueco,canvas):
    x,y = validar_xy(punto.get(),True)
cadena_parametro = str(radio.get())+',0'
radio,not_use = validar_xy(cadena_parametro,False)
id_hueco = int(value_hueco.get())
    #cuadrantes = ['Cuadrante 1','Cuadrante 2','Cuadrante 3','Cuadrante 4']
try:
        direccion = cuadrantes.index(direc.get())
except:
        direccion = 0     if x is None or y is
None or radio is None:
        return None
else:
        if direccion == 1:
            inicio = 90; fin = 180
elif direccion == 2:
            inicio = 180; fin = 270
elif direccion == 3:
            inicio = 270; fin = 360
        else: #Si el usuario no elige ninguna direccion se pone automaticamente
en la primera dado el arreglo              inicio = 0; fin = 90
```

```
        cuarto_circulo = mpatches.Wedge((x,y), radio, inicio,
fin, color=colores[id_hueco])
lista.append(cuarto_circulo)
centro_cuartocirculo(radio,x, y, direccion,id_hueco)
crear_graficos(canvas)
 def
forma_semicirculo(punto,radio,direc,value_hueco,canvas):
    x,y = validar_xy(punto.get(),True)
cadena_parametro = str(radio.get())+',0'
radio,not_use = validar_xy(cadena_parametro,False)
id_hueco = int(value_hueco.get())     try:
        direccion = direcciones.index(direc.get())
except:
        direccion = 0     if x is None or y is
None or radio is None:
        return None
else:
        #direcciones = ['Arriba', 'Abajo', 'Izquierda', 'Derecha']
if direccion == 1:
            inicio = 180; fin = 0
elif direccion == 2:
            inicio = 90; fin = 270
elif direccion == 3:
            inicio = 270; fin = 90          else: #Si el usuario no elige ninguna
direccion se pone automaticamente en la primera dado el arreglo
inicio = 0; fin = 180          semicirculo = mpatches.Wedge((x,y), radio, inicio,
fin, color=colores[id_hueco])         lista.append(semicirculo)
centro_semicirculo(radio,x, y, direccion,id_hueco)         crear_graficos(canvas)
 def
forma_elipse(punto,anchura,altura,value_hueco,canvas):
    x,y = validar_xy(punto.get(),True)
    cadena_parametro = str(anchura.get())+','+str(altura.get())
anchura,altura = validar_xy(cadena_parametro,False)     id_hueco =
int(value_hueco.get())     if x is None or y is None or anchura is
None or altura is None:
        return None
else:
        elipse =
mpatches.Ellipse((x,y),width=anchura,height=altura,color=colores[id_hueco])
lista.append(elipse)
        centro_elipse(anchura,altura,x,y,id_hueco)
crear_graficos(canvas)
```

```
#Creación de funciones para los eventos realizados por botones y/u otros
componentes de la interfaz de usuario def event_aceptar(opcion):
    #Variable que determina que figura fue escogida por el usuario uno vez de
click en le boton aceptar     global basic_compuesta     try:         if
opcion.get() == 'Figuras Básicas':
            basic_compuesta  =  0              elif
opcion.get() == 'Figuras Compuestas':
            basic_compuesta = 1
except:
        basic_compuesta = 0     root.withdraw() #oculta la ventana
principal     app = ct.CTkToplevel(fg_color='#D4D4D4') #Creación de la
segunda ventana     app.title("Calculo del centroide y momento de inercia -
AGY"); app.geometry('1500x750+0+0') #Tamaño     ventana_interfaz(app)
 def event_presentacion():     vn = ct.CTkToplevel()
vn.geometry("700x500")     vn.title("Presentación Formal")
vn.attributes("-topmost", True); vn.lift(); vn.focus_force()
    ct.CTkLabel(vn, text="UNIVERSIDAD TECNOLÓGICA DE PANAMÁ\nFACULTAD DE INGENIERÍA
DE SISTEMAS COMPUTACIONALES"+
            "\nDEPARTAMENTO DE COMPUTACIÓN Y SIMULACIÓN DE SISTEMAS\nASIGNATURA:
MECÁNICA"+
            "\n\nTEMA:\nPROYECTO N1\nCÁLCULO DEL CENTROIDE Y MOMENTO DE
INERCIA\nFIGURAS BASICAS Y COMPUESTAS"+
            "\n\nINTEGRANTES: \nALVARADO ALEX, 8-998-934\nGUEVARA GABRIELA, 8-
1005-662\nYEE ERNESTO, 8-963-608"+
            "\n\nFACILITADOR:\nDR. OSCAR WONG\n\nFECHA: \n19/JUNIO/2024",
font=(font_texto,20)).pack()
 def
event_salir():
    mb.showinfo("Salida","Ha salido del programa")
sys.exit()

def event_regresar(app,canvas): #Limpia todos los datos recopilados hasta
el momento de apretar el boton de regresar     lista.clear()
centroide.clear()     control_figuras.clear()     ax.clear()
canvas.draw()
```

```
    app.destroy()
root.deiconify()
 def
event_centroide(canvas,text,tab):
if len(centroide) < 1:
        mb.showinfo("Centroide","No existe figura para calcular el centroide")
return None     tab.set("Centroide")     text.delete("0.0", "end")     texto =
""
    #Limpieza de centroides en la grafica que ya estuvieron
calculados     try:         for a in puntos_centroide:
            a.remove()
puntos_centroide.clear()
canvas.draw()     except:
        print("")     sumatoria, x_general, y_general = centroide_general()
#Esto es la sumatoria     if basic_compuesta == 1:         confirmar =
mb.askquestion("Centroide","¿Desea agregar los centroides individuales de la
figura?")         if confirmar == 'yes':
            x = [a[2] for a in centroide]             y = [a[3] for a in centroide]
puntos = ax.scatter(x, y, marker = "*", c = '#D4D4D4');
puntos_centroide.append(puntos)     centro = ax.scatter(x_general, y_general,
marker = "*", c = 'white'); puntos_centroide.append(centro)     canvas.draw()
#Titulo de la tabla     texto += "El centroide de la figura es: ({:.2f},
{:.2f})\n\n".format(x_general, y_general)     texto += "Desglose de datos
sumatoria:\n"+"Área:
{:.2f}".format(sumatoria[1])+"\tX*A: {:.2f}".format(sumatoria[4])
texto += "\tY*A: {:.2f}".format(sumatoria[5])     texto +=
obtener_impresion(6,[])     text.insert("insert",texto)
 def
event_inercia(canvas,text,eje,tab):
    if len(centroide) < 1:
        mb.showinfo("Inercia","No existe figura para calcular el momento
de inercia")         return None
    tab.set("Momento de Inercia")     text.delete("0.0",
"end")      area_suma, x_general, y_general =
centroide_general()
```

```
    sumatoria = calcular_inercia(x_general,y_general)
texto = ""     texto += "El Vertice centroidal de la figura
es: V({:.2f},
{:.2f})\n".format(x_general,    y_general)
if eje == 'Eje Centroidal':
        linea = ax.axhline(y_general);
lineas_momento.append(linea)         linea =
ax.axvline(x_general); lineas_momento.append(linea)         if
basic_compuesta == 0: #Inercia si es simple             texto +=
"El Momento de inercia en el eje X es:
{:.2f}\n".format(sumatoria[6])             texto += "El
Momento de inercia en el eje Y es:
{:.2f}\n".format(sumatoria[7])             texto
+= obtener_impresion(8,inercias)         else:
            texto += "El Momento de inercia en el eje X es:
{:.2f}\n".format(sumatoria[6]+sumatoria[7])
texto += "El Momento de inercia en el eje Y es:
{:.2f}\n".format(sumatoria[8]+sumatoria[9])
texto += obtener_impresion(10,inercias)     elif
eje == 'Eje X Arbitrario':
        dialog = ct.CTkInputDialog(text="Ingrese el momento de inercia arbitario
de x:", title="Eje X Arbitrario")         llamar_eje = dialog.get_input()
eje_y,not_use = validar_xy(llamar_eje+",0",False)         if eje_y is None or
not_use is None:
            mb.showerror("Error", "No se ha podido validar el eje
correspondiente, intentelo nuevamente")         else:
            inercia_ArbitrarioX = (sumatoria[6]+sumatoria[7]) + (area_suma[1] *
((eje_y - y_general) ** 2))             linea = ax.axhline(eje_y);
lineas_momento.append(linea)             texto += "El Momento de
inercia Arbitrario en X es:
{:.2f}\n".format(inercia_ArbitrarioX)
texto += obtener_impresion(10,inercias)     elif
eje == 'Eje Y Arbitrario':
        dialog = ct.CTkInputDialog(text="Ingrese el momento de inercia arbitario
de y:", title="Eje Y Arbitrario")         llamar_eje = dialog.get_input()
        eje_x,not_use = validar_xy(llamar_eje+",0",False)
if eje_x is None or not_use is None:
            mb.showerror("Error", "No se ha podido validar el eje
correspondiente, intentelo nuevamente")         else:
            inercia_ArbitrarioY = (sumatoria[8]+sumatoria[9]) +
(area_suma[1]*((eje_x - x_general)**2))             linea =
ax.axvline(eje_x); lineas_momento.append(linea)
```

```
            texto += "El Momento de inercia Arbitrario en Y es:
{:.2f}\n".format(inercia_ArbitrarioY)             texto +=
obtener_impresion(10,inercias)     else:         mb.showerror("Error", "No
ha seleccionado ningun eje para calcular su inercia")
text.insert("insert", texto)     canvas.draw()
 def event_limpiar(canvas):
try:         for a in
lineas_momento:
            a.remove()
lineas_momento.clear()
canvas.draw()     except:
        mb.showwarning("Lineas de momento","No existen lineas para borrar")
 def
event_modifier(app,canvas):
    valores = [a[0] for a in centroide]# Obteniendo solo los nombres de las figuras
if not valores:
        mb.showinfo("Eliminar","No hay figuras para eliminar, se regresara al menu
principal para iniciar una nueva seleccion")         event_regresar(app,canvas)
return None     menu_selec = ct.CTkToplevel()     menu_selec.geometry('344x70')
menu_selec.title("Eliminar figura")     ct.CTkLabel(menu_selec, text="Seleccione
la figura que desee eliminar").pack()     opcion_select =
ct.CTkComboBox(menu_selec, values=valores, command=lambda _:
event_select_fig_delete(menu_selec, opcion_select.get(), canvas))
opcion_select.place(x=100, y=25)     menu_selec.attributes("-
topmost", True)     menu_selec.lift()
menu_selec.focus_force()
 def event_select_fig_delete(toplevel, seleccion,
canvas):
    confirmar = mb.askquestion("Eliminar Figura", "¿Seguro que desea eliminar la
figura " + seleccion + "?", type='yesnocancel')     if confirmar == 'yes':
        try:
            # Encontrar el índice de la figura seleccionada
            indice = next((i for i, fig in enumerate(centroide) if fig[0] ==
seleccion), None)             if indice is not None:
centroide.pop(indice)                 lista.pop(indice)
```

```
                try:
                    control_figuras.pop(indice)
inercias.pop(indice)                 except:
                    print("")
crear_graficos(canvas)
toplevel.destroy()             else:
                mb.showerror("Error",  "No  se  encontró  la  figura")
except Exception as e:
            mb.showerror("Error", f"Ocurrió un error: {e}")
elif confirmar == 'cancel':
        toplevel.destroy()
 def event_figura_select(valor_seleccionado, insercion_frame,canvas,app):
#Crear cuadros de texto para la inserción de puntos y otros parametros     punto_xy
= ct.CTkEntry(insercion_frame, placeholder_text="",width=80,height=30,
fg_color='#004AAD')     punto_xy.place(x=360,y=220)
ct.CTkLabel(insercion_frame,text="Punto
x,y",font=(font_texto,25)).place(x=360,y=180)     parametro_1 =
ct.CTkEntry(insercion_frame, placeholder_text="",width=80,height=30,
fg_color='#004AAD')     parametro_1.place(x=360,y=300)     text_parametro1 =
ct.CTkLabel(insercion_frame,text="",font=(font_texto,25))
text_parametro1.place(x=360,y=260)     parametro_2 = ct.CTkEntry(insercion_frame,
placeholder_text="",width=80,height=30, fg_color='#5E606C', state=DISABLED)
parametro_2.place(x=360,y=380)     text_parametro2 =
ct.CTkLabel(insercion_frame,text="",font=(font_texto,25))
text_parametro2.place(x=360,y=340)     hueco = ct.CTkCheckBox(insercion_frame,
text="Figura Hueca", font=(font_texto,25), onvalue=1, offvalue=0, state=DISABLED)
hueco.place(x=360,y=435)
ct.CTkLabel(insercion_frame,text="Dirección",font=(font_texto,25),text_color='w
hite').place(x=85,y=425)     direccion =
ct.CTkComboBox(insercion_frame,width=200,height=40,button_color='#5E606C',
values=direcciones,state=DISABLED)     direccion.place(x=85,y=460)
    #Si la opcion escogida es de la figuras compuestas se habilita la opcion
de escoger si esta figura es hueco o no     if basic_compuesta == 1:
        hueco.configure(state=NORMAL)
        #Condicionar para que solo sea en la compuesta
ct.CTkButton(insercion_frame,text='Eliminar
figura',width=192,height=56,fg_color='#5271FF',
```

```
                                hover_color='#D4D4D4', command=
lambda:event_modifier(app,canvas)).place(x=308,y=510)     op =
figuras.index(valor_seleccionado) #Pasa a obtener cual fue la figura
seleccionada
    #figuras = ['Triangulo', 'Cuadraro y/o rectangulo', 'Circulo', 'Semicirculo',
'Cuarto de area circulo', 'Elipse']     if op == 0: #Triangulo
ct.CTkLabel(insercion_frame,image=itriangulo,text="").place(x=27,y=160)
parametro_1.configure(state=NORMAL)
text_parametro1.configure(text="Base          ")
parametro_2.configure(state=NORMAL,fg_color='#004AAD')
text_parametro2.configure(text="Altura          ")
direccion.configure(state=NORMAL,button_color='green')
ct.CTkButton(insercion_frame,text='Actualizar
figura',width=192,height=56,fg_color='#5271FF',hover_color='#D4D4D4',
command= lambda:
forma_itriangulo(punto_xy,parametro_1,parametro_2,direccion,hueco,canvas)).place(x=
82,y=510)     elif op == 1: #Triangulo Rectangulo
ct.CTkLabel(insercion_frame,image=rtriangulo,text="").place(x=27,y=160)
parametro_1.configure(state=NORMAL)         text_parametro1.configure(text="Base
")         parametro_2.configure(state=NORMAL,fg_color='#004AAD')
text_parametro2.configure(text="Altura          ")
direccion.configure(state=NORMAL, values=cuadrantes,button_color='green')
ct.CTkButton(insercion_frame,text='Actualizar
figura',width=192,height=56,fg_color='#5271FF',hover_color='#D4D4D4',
command= lambda:
forma_rtriangulo(punto_xy,parametro_1,parametro_2,direccion,hueco,canvas)).place(x=
82,y=510)                 elif    op    ==    2:    #Cuadrado    y/o    rectangulo
ct.CTkLabel(insercion_frame,image=rectangulo,text="").place(x=27,y=160)
parametro_1.configure(state=NORMAL)
text_parametro1.configure(text="Anchura                                        ")
parametro_2.configure(state=NORMAL,fg_color='#004AAD')
text_parametro2.configure(text="Altura                                         ")
ct.CTkButton(insercion_frame,text='Actualizar
figura',width=192,height=56,fg_color='#5271FF',hover_color='#D4D4D4',
command= lambda:
forma_cuadro_rec(punto_xy,parametro_1,parametro_2,hueco,canvas)).place(x=82,y=510)
elif op == 3: #Circulo
        ct.CTkLabel(insercion_frame,image=circulo,text="").place(x=27,y=160)
parametro_1.configure(state=NORMAL)
        text_parametro1.configure(text="Radio       ")
text_parametro2.configure(text="                ")
ct.CTkButton(insercion_frame,text='Actualizar
figura',width=192,height=56,fg_color='#5271FF',hover_color='#D4D4D4',
```

```
                            command= lambda:
forma_circulo(punto_xy,parametro_1,hueco,canvas)).place(x=82,y=510)     elif op
== 4: #Semicirculo
ct.CTkLabel(insercion_frame,image=semicirculo,text="").place(x=27,y=160)
parametro_1.configure(state=NORMAL)
text_parametro1.configure(text="Radio       ")
text_parametro2.configure(text="                  ")
direccion.configure(state=NORMAL,button_color='green')
ct.CTkButton(insercion_frame,text='Actualizar
figura',width=192,height=56,fg_color='#5271FF',hover_color='#D4D4D4',
command= lambda:
forma_semicirculo(punto_xy,parametro_1,direccion,hueco,canvas)).place(x=82,y=510)
elif op == 5: #Cuarto de area circulo
ct.CTkLabel(insercion_frame,image=cuarto_circulo,text="").place(x=27,y=160)
parametro_1.configure(state=NORMAL)         text_parametro1.configure(text="Radio
")         text_parametro2.configure(text="                        ")
direccion.configure(state=NORMAL, values=cuadrantes,button_color='green')
ct.CTkButton(insercion_frame,text='Actualizar
figura',width=192,height=56,fg_color='#5271FF',hover_color='#D4D4D4',
command= lambda:
forma_cuarto_circulo(punto_xy,parametro_1,direccion,hueco,canvas)).place(x=82,y=510
)     elif op == 6: #Elipse
ct.CTkLabel(insercion_frame,image=elipse,text="").place(x=27,y=160)
parametro_1.configure(state=NORMAL)
text_parametro1.configure(text="Anchura      ")
parametro_2.configure(state=NORMAL,fg_color='#004AAD')
text_parametro2.configure(text="Altura          ")
ct.CTkButton(insercion_frame,text='Actualizar
figura',width=192,height=56,fg_color='#5271FF',hover_color='#D4D4D4',
command= lambda:
forma_elipse(punto_xy,parametro_1,parametro_2,hueco,canvas)).place(x=82,y=510)
 def
ventana_interfaz(app):
    #Frame para el desplegue del grafico de matplotlib
    grafico_frame = ct.CTkFrame(app,width=940,height=533,fg_color='#D4D4D4')
grafico_frame.place(x=0,y=0)
    ct.CTkLabel(grafico_frame,text='Gráfico Matplotlib',font=(font_titulo,30),
text_color='black').place(x=150,y=20)     #Inserción del gráfico dentro del
frame
    canvas = FigureCanvasTkAgg(fig, master=grafico_frame)
canvas.draw()
    canvas.get_tk_widget().place(x=47,y=72,width=940,height=500)
    #Boton para la descarga en png del grafico actual
ct.CTkButton(grafico_frame,text='Descargar
gráfico',width=285,height=40,fg_color='#222222',
```

```
                                    hover_color='#5E17EB',
command=lambda:descargar_plt()).place(x=576,y=14)
ct.CTkLabel(grafico_frame,text='Xi,Xf',font=(font_texto,20),
text_color='black').place(x=80,y=484)     limite_x = ct.CTkEntry(grafico_frame,
placeholder_text="",width=80,height=30, fg_color='#004AAD')
limite_x.place(x=126,y=484)
ct.CTkLabel(grafico_frame,text='Yi,Yf',font=(font_texto,20),
text_color='black').place(x=210,y=484)     limite_y = ct.CTkEntry(grafico_frame,
placeholder_text="",width=80,height=30, fg_color='#004AAD')
limite_y.place(x=255,y=484)
    #Boton para la actualización de los limites del gráfico
ct.CTkButton(grafico_frame,text='Actualizar límites del
grafico',width=285,height=30,fg_color='#222222',
hover_color='#5E17EB',
command=lambda:setearlimites(limite_x,limite_y,canvas)).place(x=576,y=484
)
    #Frame de insercion de figura     insercion_frame =
ct.CTkFrame(app,width=560,height=720,fg_color='#222222')
insercion_frame.place(x=940,y=0)
ct.CTkLabel(insercion_frame,text='Inserción de',font=(font_titulo,30),
text_color='white').place(x=40,y=20)     ct.CTkLabel(insercion_frame,text="
Formato de entrada para las coordenadas: x,y\nFormato de entrada para
parametros: num",font=(font_texto,25),text_color='white').place(x=80,y=75)
    # ComboBox para seleccionar la figura     figura_selec =
ct.CTkComboBox(insercion_frame, width=200, height=40, values=figuras,
button_color='green')     figura_selec.place(x=230, y=20)
    # Configuración del comando del ComboBox para cambiar la figura
figura_selec.configure(command=lambda value: event_figura_select(value,
insercion_frame,canvas,app))


    #Frame para los distintos modificadores del programa     mod_frame
= ct.CTkFrame(app,width=560,height=120,fg_color='#004AAD')
mod_frame.place(x=940,y=650)
    #Boton para regresar al menu principal
ct.CTkButton(mod_frame,text='Menu principal',command= lambda:
event_regresar(app,canvas),width=192,height=56,fg_color='#222222'
,                 hover_color='#5271FF').place(x=195,y=25)
    #Frame para las opciones de visualización de resultados
    # Crear el TabView dentro de la ventana
op_calculo = ct.CTkTabview(app,anchor='nw',
width=600,height=190,fg_color='#5271FF')
op_calculo.place(x=0, y=520)
```

```
    # Añadir pestañas al TabView     op_calculo.add("Centroide")  #
Añadir pestaña al final     op_calculo.add("Momento de Inercia")  #
Añadir otra pestaña al final
    # Establecer la pestaña actualmente visible
op_calculo.set("Centroide")     texto_centroide =
ct.CTkTextbox(master=op_calculo.tab("Centroide"),
fg_color='#5271FF',activate_scrollbars=True,width=600,
height=190, font=(font_texto,20))     texto_centroide.pack()     texto_inercia
= ct.CTkTextbox(master=op_calculo.tab("Momento de Inercia"),
fg_color='#5271FF',activate_scrollbars=True,width=600,
height=200, font=(font_texto,20))     texto_inercia.pack()
    #Botones para la impresión de resultados     # ComboBox para
seleccionar la figura     compuesta = ['Eje Centroidal', 'Eje X
Arbitrario', 'Eje Y Arbitrario']     eje_select = ct.CTkComboBox(app,
width=250, height=30, values=['Eje Centroidal'], button_color='green')
eje_select.set("Seleccione el eje del momento")
eje_select.place(x=680, y=600)     if basic_compuesta == 1:
        eje_select.configure(values=compuesta)
ct.CTkButton(app,text='Calcular
Centroide',width=192,height=30,fg_color='#222222',
hover_color='#5E17EB', command=
lambda:event_centroide(canvas,texto_centroide,op_calculo)).place(x=704,y=550)
#command= lambda: impresion_centroide(texto_centroide,canvas)
ct.CTkButton(app,text='Calcular
Momento',width=192,height=30,fg_color='#222222',
hover_color='#5E17EB', command= lambda:
event_inercia(canvas,texto_inercia,eje_select.get(),op_calculo)).place(x=704,y=650)
ct.CTkButton(app,text='Limpiar líneas de
Momento',width=192,height=30,fg_color='#222222',
hover_color='#5E17EB', command= lambda:
event_limpiar(canvas)).place(x=704,y=700)
# Creacion de la ventana principal root
= ct.CTk()
root.geometry("800x350+400+200")
root.title("Proyecto N1 - Mecánica")
# Creación del grafico de matplotlib para despligue de las formas desarrollada
fig, ax = plt.subplots() xi = 0; xf = 10; yi=0; yf=10 lista = [] centroide =
[] lineas_momento = [] puntos_centroide = []
```

```
control_figuras = []
# Etiquetas de la ventana principal
ct.CTkLabel(root,image=logo,text="").place(x=100,y=130)
ct.CTkLabel(root,image=marca,text="").place(x=500,y=130) ct.CTkLabel(root,
text="¡Bienvenido \°O°/!", font=(font_texto, 20)).pack() ct.CTkLabel(root,
text="CÁLCULO DE CENTROIDE \n Y MOMENTO DE INERCIA",font=(font_titulo,
40)).pack() ct.CTkLabel(root, text="Seleccione el tipo de figura",
font=(font_texto,
20)).place(x=300, y=150)
#Creación del combobox para seleccion de la figura y lo que se desea calcular
opcion_figura = ct.CTkComboBox(root, values=["Figuras Básicas", "Figuras
Compuestas"], width = 200, height=40,
button_color = 'green') opcion_figura.place(x=300, y=200)
#Ultimos botones para distintas opciones de continuación en el programa
ct.CTkButton(root, text="Aceptar",
command=lambda:event_aceptar(opcion_figura),width = 150,
height=40).place(x=130,y=280) ct.CTkButton(root, text="Presentacion",width =
150, height=40, command=lambda:event_presentacion()).place(x=330,y=280)
ct.CTkButton(root, text="Salir", command=lambda:event_salir(), width = 150,
height=40).place(x=530,y=280)
#Bucle para la ejecución del programa root.mainloop()
```