

**Instituto Politecnico Nacional**

**Escuela superior de computo**

**Oscar Humberto Mayoleytte Paredes**

**Armando Jimenez Galvan**

**Materia: Algebra Lineal**

**Profesor: David Correa Coyac**

**Grupo: 2BM2**

**Proyecto: Visualizador de Transformaciones Geométricas en 2D**

**Documentación detallada para terminal\_reader.pyPropósito del Código**

**El archivo terminal\_reader.py implementa una interfaz gráfica que permite a los usuarios trabajar con figuras geométricas básicas como cuadrados y triángulos, y realizar transformaciones geométricas como rotación, escala, reflexión y traslación. Los resultados de las transformaciones se visualizan gráficamente.El programa está diseñado para facilitar la comprensión de transformaciones geométricas en 2D y es útil en entornos educativos y de análisis geométrico.Documentación del Código1. Librerías Importadas**pythonimport numpy as np # Biblioteca para cálculos matemáticos y manejo de arreglosimport tkinter as tk # Biblioteca para crear interfaces gráficasfrom tkinter import ttk # Widgets mejorados para interfaces gráficasimport matplotlib.pyplot as plt # Biblioteca para graficar figuras-numpy: Realiza cálculos numéricos, operaciones matriciales y manipulación de arreglos.tkinter y ttk: Proveen herramientas para construir la interfaz gráfica de usuario (GUI).matplotlib.pyplot: Se utiliza para visualizar las figuras geométricas y sus transformaciones.**2. Clase Principal: TransformationAppDefinición de la clase y método de inicialización**pythonclass TransformationApp: Clase principal que gestiona la interfaz gráfica y las operaciones de transformación de figuras geométricas. def \_\_init\_\_(self, root):  **Inicializa la ventana principal y los componentes de la interfaz gráfica. Parámetros: root: Ventana principal de la aplicación (Tkinter).** self.root = root self.root.title("Transformaciones de Figuras") self.figure = None # Figura actual seleccionada self.result\_dict = {} # Diccionario que almacena resultados de transformaciones self.vertices = [] # Lista de vértices personalizados self.figure: Almacena las coordenadas de la figura geométrica actual (cuadrado o triángulo).self.result\_dict: Guarda los resultados de las transformaciones aplicadas a la figura.self.vertices: Lista de vértices personalizados añadidos por el usuario.**3. Configuración de la InterfazBotones para la selección de figuras**

python ttk.Button(root, text="Crear Cuadrado", command=self.create\_square).pack(pady=5) ttk.Label(root, text="Crea un cuadrado predeterminado").pack(anchor="w") ttk.Button(root, text="Crear Triángulo", command=self.create\_triangle).pack(pady=5) ttk.Label(root, text="Crea un triángulo predeterminado").pack(anchor="w") ttk.Button(root, text="Agregar Vértice", command=self.add\_vertex).pack(pady=5) ttk.Label(root, text="Añade vértices personalizados a la figura").pack(anchor="w")**Crear Cuadrado: Genera un cuadrado con coordenadas predefinidas.Crear Triángulo: Genera un triángulo equilátero.Agregar Vértice: Permite añadir vértices personalizados a la figura.Entradas para transformaciones geométricas**python frame = ttk.LabelFrame(root, text="Transformaciones") frame.pack(pady=10, padx=10, fill=tk.X) self.rotation\_entry = self.create\_input(frame, "Rotación (°):") self.scale\_entry = self.create\_input(frame, "Escala (Sx, Sy):") self.translation\_entry = self.create\_input(frame, "Traslación (Tx, Ty):") self.reflection\_entry = self.create\_input(frame, "Reflexión (H/V):")**Se crean entradas específicas para cada transformación: rotación, escala, traslación y reflexión.Cada entrada se asocia con un cuadro de texto donde el usuario ingresa los valores necesarios.Botones para aplicar transformaciones y graficar**python ttk.Button(frame, text="Aplicar Transformaciones", command=self.apply\_transformations).pack(pady=5) ttk.Label(frame, text="Aplica las transformaciones seleccionadas a la figura actual").pack(anchor="w") ttk.Button(frame, text="Graficar", command=self.plot\_results).pack(pady=5) ttk.Label(frame, text="Genera una gráfica con la figura y sus transformaciones").pack(anchor="w")**Aplicar Transformaciones: Ejecuta las transformaciones especificadas en las entradas.Graficar: Visualiza la figura y las transformaciones aplicadas.4. Métodos para Crear FigurasCrear Cuadrado**python def create\_square(self): self.figure = np.array([[0, 0], [5, 0], [5, 5], [0, 5]]) # Coordenadas de un cuadrado self.result\_dict = {"original": {"value": self.figure, "color": "#1A0014"}} print("Cuadrado creado:", self.figure)**Genera un cuadrado con vértices en el plano cartesiano.**Los resultados se almacenan en el diccionario `self.result\_dict`.**Crear Triángulo**python def create\_triangle(self): self.figure = np.array([[0, 0], [5, 0], [2.5, 5]]) # Coordenadas de un triángulo equilátero self.result\_dict = {"original": {"value": self.figure, "color": "#1A0014"}} print("Triángulo creado:", self.figure)**Genera un triángulo equilátero con vértices predefinidos.5. Transformaciones Geométricas**

**Aplicar Transformaciones**python def apply\_transformations(self): if self.figure is None and not self.vertices: print("No hay figura ni vértices personalizados creados.") return**Comprueba si hay una figura o vértices disponibles antes de realizar las transformaciones.Rotación**python angle = self.get\_float(self.rotation\_entry.get(), radians=True) if angle: matrix = [[np.cos(angle), -np.sin(angle)], [np.sin(angle), np.cos(angle)]] self.result\_dict["rotation"] = {"value": np.dot(vertices, matrix), "color": "#FF5733"}Calcula la matriz de rotación y aplica la transformación.**Escala**

python scale\_values = self.get\_float\_list(self.scale\_entry.get()) if scale\_values and len(scale\_values) == 2: matrix = [[scale\_values[0], 0], [0, scale\_values[1]]] self.result\_dict["scale"] = {"value": np.dot(vertices, matrix), "color": "#33FF57"}**Aumenta o disminuye el tamaño de la figura según los factores de escala en X e Y.**Reflexión

python ref\_type = self.reflection\_entry.get().strip().lower() if ref\_type == "h": matrix = [[1, 0], [0, -1]] self.result\_dict["reflection"] = {"value": np.dot(vertices, matrix), "color": "#3357FF"} elif ref\_type == "v": matrix = [[-1, 0], [0, 1]] self.result\_dict["reflection"] = {"value": np.dot(vertices, matrix), "color": "#3357FF"}**Aplica reflexiones horizontales (H) o verticales (V) mediante matrices predefinidas.Traslación** python translation\_values = self.get\_float\_list(self.translation\_entry.get()) if translation\_values and len(translation\_values) == 2: self.result\_dict["translation"] = {"value": vertices + translation\_values, "color": "#FFD700"}**Desplaza la figura a nuevas coordenadas especificadas por el usuario.6. Visualización Gráfica Graficar Figuras**python def plot\_results(self): if not self.result\_dict: print("No hay datos para graficar.") return fig, ax = plt.subplots(figsize=(6, 6)) for key, data in self.result\_dict.items(): points = np.array(data["value"]) points = np.vstack([points, points[0]]) # Cerrar la figura ax.fill(points[:, 0], points[:, 1], label=key.capitalize(), alpha=0.5, color=data["color"]) ax.plot(points[:, 0], points[:, 1], linestyle="--", color="black")**Dibuja las figuras transformadas y originales en un gráfico utilizando matplotlib.Ejecución1.Requisitos previos:** bash pip install numpy matplotlib **2. Ejecutar el programa:** bash python terminal\_reader.py **Este código proporciona una herramienta visual e interactiva para explorar transformaciones geométricas en 2D.**