

**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 de app.py**

**Propósito del Código**

El archivo app.py implementa una aplicación gráfica interactiva que permite al usuario realizar transformaciones geométricas en 2D. Estas transformaciones incluyen:1. Rotaciones: Girar figuras respecto al origen.2. Escalas: Modificar el tamaño de las figuras en proporciones específicas.3. Traslaciones: Mover figuras a nuevas ubicaciones.4. Reflexiones: Invertir figuras sobre el eje horizontal o vertical.El programa proporciona una interfaz gráfica para seleccionar figuras geométricas básicas como cuadrados, triángulos o figuras personalizadas. También permite ingresar parámetros para las transformaciones y visualizar los resultados gráficamente utilizando matplotlib.**Código Detallado y Documentado1. Importación de Bibliotecas**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 generar gráficos y visualizaciones**Explicación:**

numpy: Para realizar cálculos matemáticos como multiplicación de matrices.-tkinter y ttk: Herramientas para construir la interfaz gráfica del usuario (GUI).matplotlib.pyplot: Para graficar las figuras geométricas y sus transformaciones.2. Clase Principal: TransformationApp\*Declaración de la Clase

pythonclass TransformationApp:**Clase principal que gestiona la interfaz gráfica y las operaciones de transformación de figuras geométricas. Esta clase contiene toda la lógica para la interfaz, las transformaciones geométricas y la visualización de resultados.**Método \_\_init\_\_: Inicialización python def \_\_init\_\_(self, root): Inicializa la ventana principal y los componentes de la interfaz. self.root = root self.root.title("Transformaciones de Figuras en 2D") self.vertices = [] # Lista de vértices iniciales self.transformed\_vertices = {} # Diccionario para almacenar transformaciones aplicadasself.vertices: Lista que almacena los puntos de la figura seleccionada.self.transformed\_vertices: Diccionario para guardar las figuras transformadas.Opciones de Figura y Transformaciones

python ttk.Label(root, text="Opciones de Figura:").grid(row=0, column=0, pady=5, sticky="w") self.option = tk.StringVar(value="square") # Variable para controlar la opción seleccionada options = [("Cuadrado", "square"), ("Triángulo", "triangle"), ("Agregar Vértices", "vertex")] for i, (label, value) in enumerate(options): ttk.Radiobutton(root, text=label, variable=self.option, value=value, command=self.update\_interface).grid(row=i+1, column=0, sticky="w") **options: Lista con las figuras disponibles:** 1. Cuadrado. 2. Triángulo. 3. Figura personalizada (agregando vértices manualmente). **3. Métodos Principales** Método update\_interface

python def update\_interface(self):

Actualizar la interfaz gráfica según la figura seleccionada. for widget in self.dynamic\_frame.winfo\_children(): widget.destroy() if self.option.get() == "square": self.add\_inputs([("x", "0"), ("y", "0"), ("Tamaño", "5")]) elif self.option.get() == "triangle": self.add\_inputs([("x1, y1", "0,0"), ("x2, y2", "5,0"), ("x3, y3", "2.5,5")]) elif self.option.get() == "vertex": self.add\_inputs([("Vértices (x, y separados por ;)", "")]) **Propósito: Cambiar dinámicamente las entradas que el usuario necesita rellenar según la figura seleccionada.**Método add\_inputspython def add\_inputs(self, fields): Crear entradas dinámicas basadas en los campos proporcionados.

self.inputs = {} # Diccionario para almacenar referencias a las entradas for i, (label, default) in enumerate(fields): ttk.Label(self.dynamic\_frame, text=label).grid(row=i, column=0, sticky="w") entry = ttk.Entry(self.dynamic\_frame) entry.insert(0, default) entry.grid(row=i, column=1) self.inputs[label] = entry **Explicación: Este método genera dinámicamente los cuadros de texto necesarios para cada figura seleccionada.**Método apply\_transformations

python def apply\_transformations(self):

Aplicar las transformaciones seleccionadas a la figura cargada. self.get\_vertices() if not self.vertices: print("No hay vértices cargados.") return angle = self.get\_float(self.transformations["rotation"].get(), radians=True) scale = self.get\_float\_list(self.transformations["scale"].get()) translation = self.get\_float\_list(self.transformations["translation"].get()) reflection = self.transformations["reflection"].get().strip().lower() self.transformed\_vertices = {"original": self.vertices} vertices = np.array(self.vertices) **Rotación** if angle is not None: rotation\_matrix = [[np.cos(angle), -np.sin(angle)], [np.sin(angle), np.cos(angle)]] self.transformed\_vertices["rotation"] = np.dot(vertices, rotation\_matrix).tolist() **Escala** if scale and len(scale) == 2: scale\_matrix = [[scale[0], 0], [0, scale[1]]] self.transformed\_vertices["scale"] = np.dot(vertices, scale\_matrix).tolist() **Traslación** if translation and len(translation) == 2: translated = vertices + translation self.transformed\_vertices["translation"] = translated.tolist() **Reflexión** if reflection in ["h", "v"]: reflection\_matrix = [[1, 0], [0, -1]] if reflection == "h" else [[-1, 0], [0, 1]] self.transformed\_vertices["reflection"] = np.dot(vertices, reflection\_matrix).tolist() print("Transformaciones aplicadas:") for key, value in self.transformed\_vertices.items(): print(f"{key.capitalize()}: {value}")```**Reflexión:** - Horizontal (`H`): Invierte las coordenadas Y. - Vertical (`V`): Invierte las coordenadas X.

Método plot\_data```python def plot\_data(self): **Graficar las figuras y transformaciones aplicadas.** if not self.transformed\_vertices: print("No hay datos para graficar.") return fig, ax = plt.subplots(figsize=(8, 8)) for key, vertices in self.transformed\_vertices.items(): vertices = np.array(vertices) vertices = np.vstack([vertices, vertices[0]]) ax.fill(vertices[:, 0], vertices[:, 1], alpha=0.5, label=key.capitalize()) ax.plot(vertices[:, 0], vertices[:, 1], linestyle="--", color="black") ax.legend() ax.set\_aspect("equal") plt.show()**Este método grafica las figuras originales y transformadas.4. Ejecución del Programa**

**1. Instalar Dependencias:** bash pip install numpy matplotlib **2. Ejecutar el Archivo:** bash python app.py **El programa abrirá una ventana interactiva para seleccionar figuras, ingresar transformaciones y visualizar los resultados.**