

**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 transformations.pyPropósito del Código**

**El archivo `transformations.py` implementa una interfaz gráfica en Python que permite crear figuras geométricas (cuadrados, triángulos) y aplicar transformaciones geométricas en 2D, como \*\*rotación\*\*, \*\*escala\*\*, \*\*traslación\*\* y \*\*reflexión\*\*. Los resultados de estas transformaciones se visualizan en tiempo real utilizando gráficos generados con Matplotlib.Este programa está diseñado para ser una herramienta interactiva, educativa y demostrativa para comprender y analizar transformaciones geométricas.1. Importación de Librerías**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 figurasnumpy: Maneja arreglos numéricos y realiza operaciones matemáticas como multiplicación de matrices y cálculo de transformaciones.tkinter y ttk: Generan la interfaz gráfica que interactúa con el usuario.matplotlib.pyplot: Se utiliza para visualizar las figuras geométricas y sus transformaciones.**2. Clase Principal: TransformationAppAtributos principales**pythonself.root = rootself.root.title("Transformaciones de Figuras")self.figure = None # Figura actual seleccionadaself.result\_dict = {} # Almacena resultados de transformacionesself.vertices = [] # Vértices personalizadosself.figure: Contiene las coordenadas de la figura geométrica actual (cuadrado o triángulo).self.result\_dict: Diccionario que almacena las transformaciones aplicadas.self.vertices: Lista para almacenar vértices personalizados ingresados por el usuario.**3. Interfaz GráficaBotones principales**pythonttk.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 de tamaño predefinido.Crear Triángulo: Genera un triángulo equilátero.Agregar Vértice: Abre una ventana para que el usuario ingrese coordenadas de nuevos vértices.Entradas para transformaciones**pythonframe = 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):")``` **Crea cuadros de texto para ingresar parámetros de transformaciones geométricas: Rotación\*\*: Ángulo de rotación en grados. Escala\*\*: Factores de escala en los ejes X e Y. Traslación\*\*: Valores de desplazamiento en X e Y. Reflexión\*\*: Reflexión horizontal (H) o vertical (V).4. Métodos para Crear FigurasCuadrado**pythondef create\_square(self): self.figure = np.array([[0, 0], [5, 0], [5, 5], [0, 5]]) self.result\_dict = {"original": {"value": self.figure, "color": "#1A0014"}} print("Cuadrado creado:", self.figure)**Genera un cuadrado de 5x5 unidades en el plano cartesiano.Triángulo**pythondef create\_triangle(self): self.figure = np.array([[0, 0], [5, 0], [2.5, 5]]) self.result\_dict = {"original": {"value": self.figure, "color": "#1A0014"}} print("Triángulo creado:", self.figure)**Genera un triángulo equilátero con coordenadas predefinidas.5. Transformaciones GeométricasMétodo principal: apply\_transformations**pythondef apply\_transformations(self): if self.figure is None and not self.vertices: print("No hay figura ni vértices personalizados creados.") return**Verifica que exista una figura o vértices antes de aplicar transformaciones.Transformaciones específicasRotación**pythonangle = 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"}**Rota la figura según un ángulo especificado.Escala**pythonscale\_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"}**Ajusta el tamaño de la figura en los ejes X e Y.Reflexión\***pythonref\_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 o verticales.Traslación**pythontranslation\_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 según los valores ingresados.6. Visualización**Método plot\_resultspythondef plot\_results(self): if not self.result\_dict and not self.vertices: 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]]) ax.fill(points[:, 0], points[:, 1], label=key.capitalize(), alpha=0.5, color=data["color"]) ax.plot(points[:, 0], points[:, 1], linestyle="--", color="black") plt.show()**Grafica las figuras transformadas y originales en un gráfico interactivo.7. Ejecución**

**1. Instalar Dependencias** bash pip install numpy matplotlib **2. Ejecutar el Archivo** bash python transformations.py

**Este programa es ideal para visualizar y comprender transformaciones geométricas en 2D. La combinación de una GUI intuitiva y gráficos interactivos hace que sea una herramienta útil en entornos educativos o de demostración.**