

**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 argument.py**

**Propósito del programaEste código implementa una aplicación gráfica interactiva que permite al usuario realizar transformaciones geométricas en 2D. Incluye:**

**Rotaciones: Girar figuras alrededor de su origen.Escalas: Cambiar el tamaño proporcional de figuras.Traslaciones: Mover figuras a nuevas posiciones.Reflexiones: Invertir figuras horizontal o verticalmente.Las figuras geométricas incluyen cuadrados, triángulos y figuras personalizadas, y los resultados se visualizan gráficamente mediante matplotlib.** **1. 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, filedialog # Widgets avanzados y diálogos para seleccionar archivosimport matplotlib.pyplot as plt # Biblioteca para generar gráficos y visualizacionesfrom datetime import datetime # Biblioteca para manejar fechas y horasExplicación:numpy: Realiza cálculos matemáticos y manipulación de vectores.tkinter: Crea una interfaz gráfica de usuario (GUI).ttk y filedialog: Mejora la GUI y permite la carga de archivos.matplotlib.pyplot: Genera visualizaciones gráficas de las transformaciones.datetime: Genera nombres únicos para los archivos de salida.**2. Clase principal: TransformationApp**

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: Almacena las coordenadas de la figura original.self.transformed\_vertices: Guarda las coordenadas de las figuras después de las transformaciones.**Opciones de figuras y botones de acción** python ttk.Label(root, text="Opciones de Figura:").grid(row=0, column=0, pady=5, sticky="w") self.option = tk.StringVar(value="square") # 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") **Opciones disponibles: Cuadrado. Triángulo. Figura personalizada (ingresando vértices manualmente).** **Entradas para transformaciones** python self.transformation\_frame = ttk.Frame(root) self.transformation\_frame.grid(row=5, column=0, pady=10) self.add\_transformation\_inputs()**Contenedor donde se añaden las entradas para rotación, escala, traslación y reflexión.3. Métodos principales de la clase**Método add\_transformation\_inputspython def add\_transformation\_inputs(self): **Crear entradas para las transformaciones geométricas (rotación, escala, traslación, reflexión).** self.transformations = {} fields = [("Rotación (°)", "rotation"), ("Escala (Sx, Sy)", "scale"), ("Traslación (Tx, Ty)", "translation"), ("Reflexión (H/V)", "reflection")] for i, (label, key) in enumerate(fields): ttk.Label(self.transformation\_frame, text=label).grid(row=i, column=0, sticky="w") entry = ttk.Entry(self.transformation\_frame) entry.grid(row=i, column=1) self.transformations[key] = entry **Campos de transformación: Rotación: Ángulo de rotación en grados. Escala: Factores de escala para los ejes x e y. Traslación: Valores de desplazamiento en x e y. Reflexión: H para reflexión horizontal y V para reflexión vertical.** 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) 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() if scale and len(scale) == 2: scale\_matrix = [[scale[0], 0], [0, scale[1]]] self.transformed\_vertices["scale"] = np.dot(vertices, scale\_matrix).tolist() if translation and len(translation) == 2: translated = vertices + translation self.transformed\_vertices["translation"] = translated.tolist() 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}") **Explicación: Reflexión: Horizontal (H): Inversa en el eje Y. Vertical (V): Inversa en el eje X. Las transformaciones se realizan en matrices mediante numpy.**

Método plot\_data python def plot\_data(self):  **5. Ejecución del programa**1.Requisitos previos: bash pip install numpy matplotlib **2. Ejecución:** bash python arguments.py **El programa permite cargar figuras, aplicar transformaciones y visualizar los resultados gráficamente.**