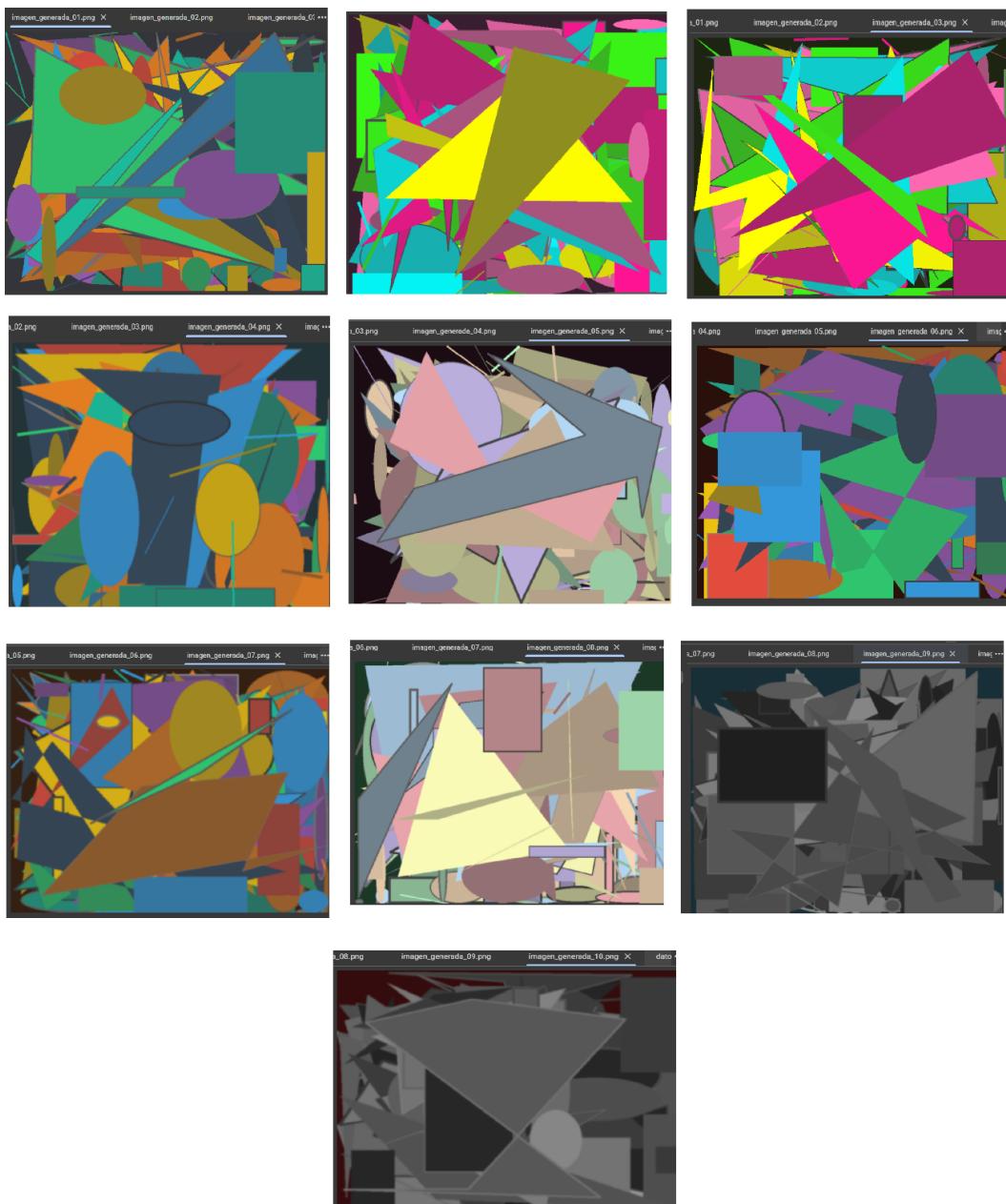


PRACTICA #1**Contents**

1. GENERADOR DE IMÁGENES	2
2. GENERADOR DE ANIMACIÓN	7
3. GENERADOR DE DATOS SINTETICOS.....	11
4. ANEXOS	14

PRACTICA #1**1. GENERADOR DE IMÁGENES****RESULTADOS**

EL generador de imágenes, tiene como base la idea de trabajo con figuras geométricas creando arte abstracto es así que genera imágenes de maneras diferentes la creación es en formato .png y crea 10 de estas



DESCRIPCIÓN

1. Estructura de Configuración (Config)

python

@dataclass

class Config:

 width: int = 1024

 height: int = 1024

 shapes: int = 120

 palette: str = "vivid"

 bgcolor: Tuple[int, int, int] = (245, 246, 250)

 outline_prob: float = 0.35

 blur: float = 0.0

 seed: int = None

 transparent: bool = False

- **Propósito:** Define parámetros configurables para la generación de imágenes.

- **Clave:**

- width/height: Dimensiones de la imagen.
- shapes: Número de figuras geométricas a dibujar.
- palette: Paleta de colores predefinida (ej. "neon").
- bgcolor: Color de fondo en formato RGB.
- outline_prob: Probabilidad de que las figuras tengan borde.
- blur: Radio de desenfoque aplicado a la imagen final.

2. Generación de Colores Aleatorios

python

```
def rand_color(palette: List[Tuple[int,int,int]]) -> Tuple[int,int,int,int]:  
  
    c = random.choice(palette)  
  
    alpha = random.randint(110, 255) # Opacidad variable  
  
    return (c[0], c[1], c[2], alpha)
```

- **Propósito:** Selecciona un color aleatorio de una paleta con transparencia (alpha).
- **Detalle:**
 - random.choice(palette) elige un color RGB de la paleta.
 - alpha entre 110-255 para permitir superposiciones semitransparentes.

3. Dibujo de Figuras Geométricas

python

```
def draw_circle(draw: ImageDraw.ImageDraw, bbox: Tuple[int,int,int,int], fill, outline,  
width: int):
```

```
    draw.ellipse(bbox, fill=fill, outline=outline, width=width)
```

- **Propósito:** Dibuja un círculo (elipse) dentro de un área delimitada (bbox).
- **Parámetros:**
 - bbox: Coordenadas (x1, y1, x2, y2) del rectángulo contenedor.
 - fill: Color de relleno con transparencia (RGBA).
 - outline: Color del borde (opcional).

4. Generación de Coordenadas Aleatorias

python

```
def random_bbox(w: int, h: int, min_size: int = 20, max_size: int = None) ->  
Tuple[int,int,int,int]:
```

```
x1, y1 = random_point(w, h)  
dx = random.randint(min_size, max_size)  
return (x1, y1, min(w, x1 + dx), min(h, y1 + dy))
```

- **Propósito:** Genera coordenadas para figuras evitando que se salgan del lienzo.
- **Nota:** $\min(w, x1 + dx)$ asegura que la figura no exceda el ancho/alto de la imagen.

5. Función Principal generate

python

```
def generate(cfg: Config, out_path: str):  
  
    img = Image.new("RGBA", (cfg.width, cfg.height), (0,0,0,0))  
  
    draw = ImageDraw.Draw(img, "RGBA")  
  
    palette = PALETTES.get(cfg.palette, PALETTES["vivid"])  
  
  
    for _ in range(cfg.shapes):  
  
        shape_type = random.choices(["circle","rect",...], weights=[25,25,...])[0]  
  
        fill = rand_color(palette)  
  
        ...
```

- **Flujo:**

1. Crea una imagen vacía en modo RGBA (con transparencia).
2. Selecciona aleatoriamente un tipo de figura (shape_type).
3. Dibuja cada figura con colores y bordes aleatorios.

6. Guardado Optimizado

python

```
if out_path.lower().endswith((".jpg", ".jpeg")):
```

```
img = img.convert("RGB") # JPEG no soporta transparencia  
img.save(out_path, quality=95, optimize=True)
```

else:

```
    img.save(out_path, optimize=True) # PNG con compresión
```

- **Propósito:** Ajusta el formato de salida según la extensión del archivo.
- **Optimización:**
 - JPEG: Calidad al 95% y compresión.
 - PNG: Compresión sin pérdida.

7. Generación de Múltiples Imágenes (Bloque main)

```
python
```

```
if __name__ == "__main__":  
  
    for i in range(10):  
  
        cfg = Config(  
  
            shapes=random.randint(80, 200),  
  
            palette=random.choice(list(PAlettes.keys())),  
  
            seed=random.randint(0, 99999)  
  
)  
  
        generate(cfg, f"imagen_{i}.png")
```

- **Propósito:** Genera 10 imágenes con configuraciones aleatorias.
- **Personalización:** Cada imagen varía en:
 - Número de figuras (shapes).
 - Paleta de colores (palette).
 - Semilla aleatoria (seed) para reproducibilidad.

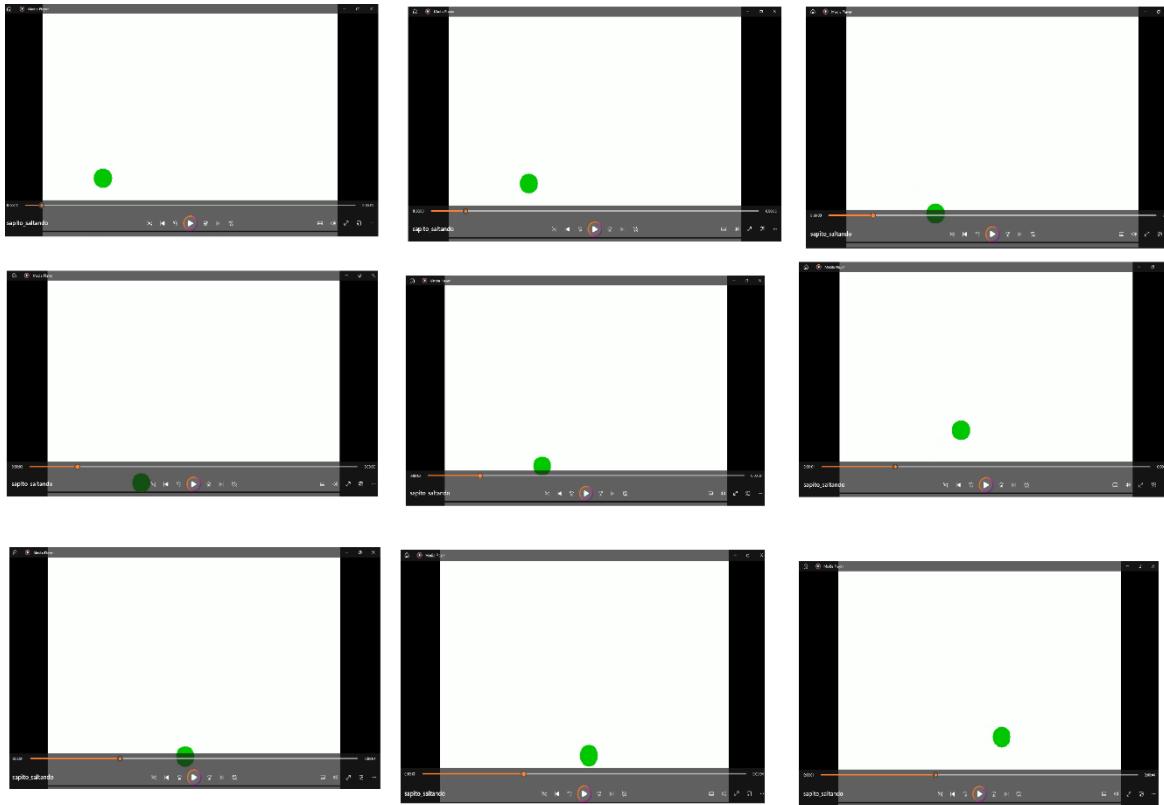
2. GENERADOR DE ANIMACIÓN

RESULTADOS

Este generador de animación fue en base a la creación de un sapito saltando la generación de esta animación es fluida y la creación es realizada en formato .mp4 .

El resultado es el siguiente:

https://drive.google.com/file/d/1QfG-QRf6oR1dZ7nyYqADGru_03WDAhcJ/view?usp=drive_link



DESCRIPCIÓN

1. ¿Qué hace el programa?

Genera imágenes con figuras geométricas aleatorias (círculos, rectángulos, líneas, etc.) usando colores vistosos. Puedes controlar:

- Tamaño de la imagen.
- Número de figuras.
- Colores (usando paletas predefinidas).

- Efectos como bordes o desenfoque.
-

2. Partes Clave Explicadas

A. Paletas de Colores (PAlettes)

python

```
PALETTES = {
```

```
    "vivid": [(231, 76, 60), (46, 204, 113), ...], # Rojos, verdes, azules brillantes  
    "pastel": [(255, 179, 186), ...], # Colores suaves  
    "neon": [(57, 255, 20), ...] # Colores fluorescentes
```

```
}
```

- **Para qué sirve:** Define grupos de colores que se usan para pintar las figuras.
 - **Ejemplo:** Si eliges la paleta "neon", las figuras tendrán colores llamativos como verde fluorescente o rosa brillante.
-

B. Configuración (Config)

python

```
@dataclass
```

```
class Config:
```

```
    width: int = 1024 # Ancho de la imagen  
    height: int = 1024 # Alto de la imagen  
    shapes: int = 120 # Número de figuras a dibujar  
    palette: str = "vivid" # Paleta de colores  
    bgcolor: Tuple = (245, 246, 250) # Color de fondo (gris claro)
```

- **Para qué sirve:** Es como un "panel de control" donde defines cómo quieres que sea la imagen.
 - **Ejemplo:** Si cambias shapes=50, solo se dibujarán 50 figuras.
-

C. Dibujar Figuras

python

```
def draw_circle(draw, bbox, fill, outline, width):  
    draw.ellipse(bbox, fill=fill, outline=outline, width=width)
```

- **Para qué sirve:** Dibuja un círculo (o elipse) en la imagen.

- **Cómo funciona:**

- bbox: Coordenadas donde se dibuja (ejemplo: (10, 20, 100, 150)).
- fill: Color de relleno (ejemplo: (255, 0, 0) para rojo).
- outline: Color del borde (opcional).

D. Generar la Imagen (generate)

python

```
def generate(cfg, out_path):  
    img = Image.new("RGBA", (cfg.width, cfg.height))  
    draw = ImageDraw.Draw(img)  
    for _ in range(cfg.shapes):  
        shape_type = random.choice(["circle", "rect", ...])  
        # ... Dibuja la figura ...  
    img.save(out_path)
```

- **Pasos:**

1. Crea una imagen vacía.
2. Dibuja figuras aleatorias (usando draw_circle, draw_rect, etc.).
3. Guarda la imagen en un archivo (ejemplo: "imagen.png").

E. Programa Principal

python

```
if __name__ == "__main__":  
  
    cfg = Config(width=800, height=600, palette="neon")  
  
    generate(cfg, "mi_imagen.png")
```

- **Qué hace:** Cuando ejecutas el código, genera una imagen de 800x600 con colores neón y la guarda como "mi_imagen.png".

3. GENERADOR DE DATOS SINTETICOS

RESULTADOS

Como se puede observar los datos sintéticos generados son 100 y estos son creados en formato .csv

gen_generada_07.png	imagen_generada_08.png	imagen_generada_09.png	imagen_generada_10.png	1 to 100 of 100 entries	Filter
id	edad	sexo	ingreso_mensual	ciudad	compra
1	56	hombre	5155.17	Santa Cruz	1
2	69	hombre	3748.37	Santa Cruz	1
3	46	hombre	3071.15	La Paz	0
4	32	hombre	3568.27	La Paz	1
5	60	hombre	4868.9	Cochabamba	1
6	25	mujer	3693.08	La Paz	1
7	38	hombre	5019.91	Cochabamba	0
8	56	mujer	2426.41	La Paz	0
9	36	mujer	2955.63	Cochabamba	0
10	40	mujer	3392.03	Santa Cruz	1
11	28	hombre	2428.06	La Paz	1
12	28	hombre	7532.4	La Paz	0
13	41	hombre	2684.48	La Paz	1
14	53	hombre	5097.86	La Paz	1
15	57	mujer	3200.73	Cochabamba	0
16	41	hombre	3383.46	La Paz	0
17	20	hombre	4361.59	Santa Cruz	0
18	39	hombre	2249.78	Santa Cruz	1
19	19	hombre	3204.71	La Paz	1
20	41	hombre	3824.23	Santa Cruz	1
21	61	mujer	2984.78	La Paz	1

77	51	hombre	3020.41	Cochabamba	1
78	51	hombre	1678.52	La Paz	1
79	27	hombre	4180.79	Cochabamba	0
80	53	hombre	3981.14	Cochabamba	1
81	31	hombre	2403.91	Cochabamba	1
82	48	mujer	3969.65	Cochabamba	1
83	65	hombre	5611.8	La Paz	1
84	32	mujer	3118.54	Cochabamba	1
85	25	hombre	2848.13	Cochabamba	1
86	31	hombre	5325.91	Santa Cruz	1
87	40	hombre	4761.09	Cochabamba	0
88	57	mujer	7673.31	Santa Cruz	0
89	38	mujer	2417.72	La Paz	1
90	33	mujer	4864.46	La Paz	1
91	62	mujer	4379.07	La Paz	1
92	35	hombre	4071.14	La Paz	0
93	64	mujer	5092.61	Santa Cruz	1
94	41	hombre	3964.33	Cochabamba	1
95	43	hombre	1676.35	La Paz	0
96	42	mujer	2983.17	Cochabamba	1
97	62	mujer	5637.32	Santa Cruz	1
98	58	mujer	5406.43	Santa Cruz	1
99	46	mujer	3552.26	Santa Cruz	1
100	32	mujer	4604.32	La Paz	0

DESCRIPCIÓN

1. Importar Librerías

python

```
import numpy as np
```

```
import pandas as pd
```

- **Para qué sirve:**

- numpy (np): Genera números aleatorios y operaciones matemáticas.
- pandas (pd): Crea y maneja tablas de datos (DataFrames).

2. Semilla Aleatoria

python

```
np.random.seed(42)
```

- **Para qué sirve:**

Fija la "semilla" para que los datos aleatorios **siempre sean los mismos** cada vez que ejecutes el código (útil para pruebas).

3. Generación de Columnas

python

```
data = {
```

```
    "id": np.arange(1, n+1), # IDs del 1 al 100  
    "edad": np.random.randint(18, 70, size=n), # Edades entre 18 y 70 años  
    "sexo": np.random.choice(["hombre", "mujer"], size=n), # Género aleatorio  
    "ingreso_mensual": np.random.normal(4000, 1200, n).round(2), # Ingresos con distribución normal  
    "ciudad": np.random.choice(["La Paz", "Cochabamba", "Santa Cruz"], size=n), # Ciudad aleatoria  
    "compra": np.random.choice([0, 1], size=n, p=[0.3, 0.7]) # 1=compra (70% probabilidad)
```

```
}
```

- **Explicación de cada columna:**

- id: Números consecutivos (1, 2, 3...).
- edad: Números enteros entre 18 y 70.
- sexo: Elige entre "hombre" o "mujer".
- ingreso_mensual: Ingresos con promedio \$4000 y desviación de \$1200.
- ciudad: Elige entre 3 ciudades.
- compra: Binario donde 1 (compra) tiene 70% de probabilidad.

4. Creación del DataFrame

python

```
df = pd.DataFrame(data)
```

- **Para qué sirve:**

Convierte el diccionario data en una tabla organizada (como Excel) con filas y columnas.

5. Mostrar y Guardar Datos

python

```
print(df.head()) # Muestra las primeras 5 filas  
df.to_csv("datos_sinteticos.csv", index=False) # Guarda en CSV
```

- **Resultados:**

- df.head() muestra:

text

```
id  edad  sexo  ingreso_mensual  ciudad  compra  
0  1  51  mujer    4875.34  Santa Cruz    1  
1  2  33  hombre   2342.15  La Paz    0
```

- Guarda los datos en un archivo CSV (que puedes abrir en Excel o Python).

4. ANEXOS

CODIGO GENERADOR DE IMÁGENES

```
# geometric_generator.py
```

```
# Generador simple de imágenes con figuras geométricas.
```

```
# Requiere: pillow (PIL) -> pip install pillow
```

```
import random
import math
import argparse
from dataclasses import dataclass
from typing import Tuple, List
from PIL import Image, ImageDraw, ImageFilter
```

```
# Paletas de color opcionales
```

```
PALETTES = {
    "vivid": [
        (231, 76, 60), (46, 204, 113), (52, 152, 219),
        (155, 89, 182), (241, 196, 15), (230, 126, 34),
        (26, 188, 156), (52, 73, 94)
    ],
    "pastel": [
        (255, 179, 186), (255, 223, 186), (255, 255, 186),
        (186, 255, 201), (186, 225, 255), (213, 198, 255)
    ]
}
```

```
],
"mono": [(30, 30, 30), (60, 60, 60), (90, 90, 90), (120, 120, 120), (150, 150, 150)],
"neon": [(57,255,20), (255,20,147), (0,255,255), (255,255,0), (255,105,180)]
}
```

@dataclass**class Config:**

```
width: int = 1024
height: int = 1024
shapes: int = 120
palette: str = "vivid"
bgcolor: Tuple[int, int, int] = (245, 246, 250)
outline_prob: float = 0.35
blur: float = 0.0
seed: int = None
transparent: bool = False
```

def rand_color(palette: List[Tuple[int,int,int]]) -> Tuple[int,int,int,int]:

```
c = random.choice(palette)
# Opacidad aleatoria suave para capas interesantes
alpha = random.randint(110, 255)
return (c[0], c[1], c[2], alpha)
```

def random_point(w: int, h: int) -> Tuple[int, int]:

```
return random.randint(0, w), random.randint(0, h)
```

```
def draw_circle(draw: ImageDraw.ImageDraw, bbox: Tuple[int,int,int,int], fill, outline,
width: int):
    draw.ellipse(bbox, fill=fill, outline=outline, width=width)

def draw_rect(draw: ImageDraw.ImageDraw, bbox: Tuple[int,int,int,int], fill, outline,
width: int):
    draw.rectangle(bbox, fill=fill, outline=outline, width=width)

def draw_line(draw: ImageDraw.ImageDraw, p1: Tuple[int,int], p2: Tuple[int,int], fill,
width: int):
    draw.line([p1, p2], fill=fill, width=width)

def draw_triangle(draw: ImageDraw.ImageDraw, pts: List[Tuple[int,int]], fill, outline,
width: int):
    draw.polygon(pts, fill=fill, outline=outline)
    if outline and width > 1:
        draw.line(pts + [pts[0]], fill=outline, width=width)

def draw_polygon(draw: ImageDraw.ImageDraw, pts: List[Tuple[int,int]], fill, outline,
width: int):
    draw.polygon(pts, fill=fill, outline=outline)
    if outline and width > 1:
        draw.line(pts + [pts[0]], fill=outline, width=width)

def random_bbox(w: int, h: int, min_size: int = 20, max_size: int = None) ->
Tuple[int,int,int,int]:
    if max_size is None:
        max_size = min(w, h) // 2
```

```
x1, y1 = random_point(w, h)

dx = random.randint(min_size, max_size)

dy = random.randint(min_size, max_size)

return (x1, y1, min(w, x1 + dx), min(h, y1 + dy))

def random_poly_points(w: int, h: int, n: int) -> List[Tuple[int,int]]:
    return [random_point(w, h) for _ in range(n)]

def generate(cfg: Config, out_path: str):
    if cfg.seed is not None:
        random.seed(cfg.seed)

    mode = "RGBA" if (cfg.transparent or cfg.blur > 0) else "RGB"
    bg = (0,0,0,0) if cfg.transparent else cfg.bgcolor
    img = Image.new(mode, (cfg.width, cfg.height), bg)
    draw = ImageDraw.Draw(img, "RGBA")

    palette = PALETTES.get(cfg.palette, PALETTES["vivid"])

    for _ in range(cfg.shapes):
        shape_type = random.choices(
            ["circle","rect","line","triangle","polygon"],
            weights=[25,25,15,20,15], k=1
        )[0]

        fill = rand_color(palette)
```

```

outline = None

outline_width = 0


if random.random() < cfg.outline_prob:
    # Outline con un gris al azar

    g = random.randint(40, 160)

    outline = (g, g, g, random.randint(120, 220))

    outline_width = random.randint(1, 6)


if shape_type == "circle":

    bbox = random_bbox(cfg.width, cfg.height)

    draw_circle(draw, bbox, fill, outline, outline_width)


elif shape_type == "rect":

    bbox = random_bbox(cfg.width, cfg.height)

    draw_rect(draw, bbox, fill, outline, outline_width)

elif shape_type == "line":

    p1 = random_point(cfg.width, cfg.height)

    # Líneas con longitud mínima

    angle = random.random() * 2 * math.pi

    length = random.randint(30, min(cfg.width, cfg.height) // 2)

    p2 = (min(max(int(p1[0] + length * math.cos(angle)), 0), cfg.width),
          min(max(int(p1[1] + length * math.sin(angle)), 0), cfg.height))

    lw = random.randint(1, 8)

    col = rand_color(palette)

    draw_line(draw, p1, p2, col, lw)

```

```
elif shape_type == "triangle":  
    pts = random_poly_points(cfg.width, cfg.height, 3)  
    draw_triangle(draw, pts, fill, outline, outline_width)  
  
else: # polygon  
    n = random.randint(4, 7)  
    pts = random_poly_points(cfg.width, cfg.height, n)  
    draw_polygon(draw, pts, fill, outline, outline_width)  
  
if cfg.blur > 0:  
    img = img.filter(ImageFilter.GaussianBlur(radius=cfg.blur))  
  
# Optimiza PNG y JPEG  
save_kwargs = {}  
if out_path.lower().endswith((".jpg", ".jpeg")):  
    save_kwargs.update({"quality": 95, "optimize": True})  
    if mode == "RGBA":  
        # Convertir a RGB para JPEG (sin transparencia)  
        img = img.convert("RGB")  
    else:  
        save_kwargs.update({"optimize": True})  
  
img.save(out_path, **save_kwargs)  
  
def parse_rgb(s: str) -> Tuple[int,int,int]:
```

```
r, g, b = [int(v) for v in s.split(",")]
return (r, g, b)
```

Define configuration and generate image directly

```
cfg = Config(
    width=800,
    height=600,
    shapes=150,
    palette="neon",
    bgcolor=(20, 20, 20),
    outline_prob=0.5,
    blur=1.0,
    seed=123,
    transparent=False
)
```

geometric_generator.py

Generador de múltiples imágenes geométricas

... (importaciones y funciones anteriores permanecen igual) ...

Cambia este bloque al final del archivo:

```
if __name__ == "__main__":
    NUM_IMAGENES = 10 # Número de imágenes a generar

    for i in range(NUM_IMAGENES):
```

```

seed = random.randint(0, 99999)

cfg = Config(

    width=800,
    height=600,
    shapes=random.randint(80, 200), # Varía la cantidad de formas
    palette=random.choice(list(PALETTES.keys())), # Cambia paleta cada vez
    bgcolor=(random.randint(10, 60), random.randint(10, 60), random.randint(10,
60)),
    outline_prob=random.uniform(0.2, 0.6),
    blur=random.uniform(0.0, 2.0),
    seed=seed,
    transparent=False

)
output_filename = f"imagen_generada_{i+1:02d}.png"
generate(cfg, output_filename)
print(f"  Imagen {i+1} generada: {output_filename}"')

```

CODIGO DEL GENERADOR DE ANIMACIÓN

```

x += vx
y += vy
vy += g # gravedad

# Rebote en el suelo
if y >= height - radius - 10:
    y = height - radius - 10
    vy = -20 # nuevo salto

```

```
# Rebote en las paredes  
if x >= width - radius or x <= radius:  
    vx *= -1  
  
# Guardar frame  
out.write(img)  
  
out.release()  
print("✅ Video generado: sapito_saltando.mp4")
```

CODIGO DEL GENERADOR DE DATOS SINTÉTICOS

```
# Generador de datos sintéticos en Python  
# -----  
import numpy as np  
import pandas as pd  
  
# 🔘 Semilla para reproducibilidad  
np.random.seed(42)  
  
# 🔘 Cantidad de filas  
n = 100  
  
# 🔘 Generamos columnas sintéticas  
data = {  
    "id": np.arange(1, n+1),  
    "edad": np.random.randint(18, 70, size=n),      # entre 18 y 70 años
```

```
"sexo": np.random.choice(["hombre", "mujer"], size=n),  
"ingreso_mensual": np.random.normal(4000, 1200, n).round(2), # normal distrib  
"ciudad": np.random.choice(["La Paz", "Cochabamba", "Santa Cruz"], size=n),  
"compra": np.random.choice([0, 1], size=n, p=[0.3, 0.7]) # binario  
}  
  
# ◆ Convertimos a DataFrame  
df = pd.DataFrame(data)  
  
# ◆ Mostrar primeras filas  
print(df.head())  
  
# ◆ Guardar en CSV  
df.to_csv("datos_sinteticos.csv", index=False)  
print("\nArchivo guardado como 'datos_sinteticos.csv'")
```