**PRACTICA #1**

Contents

[**1.** **GENERADOR DE IMÁGENES** 2](#_Toc206429268)

[**2.** **GENERADOR DE ANIMACIÓN** 7](#_Toc206429269)

[**3.** **GENERADOR DE DATOS SINTETICOS** 11](#_Toc206429270)

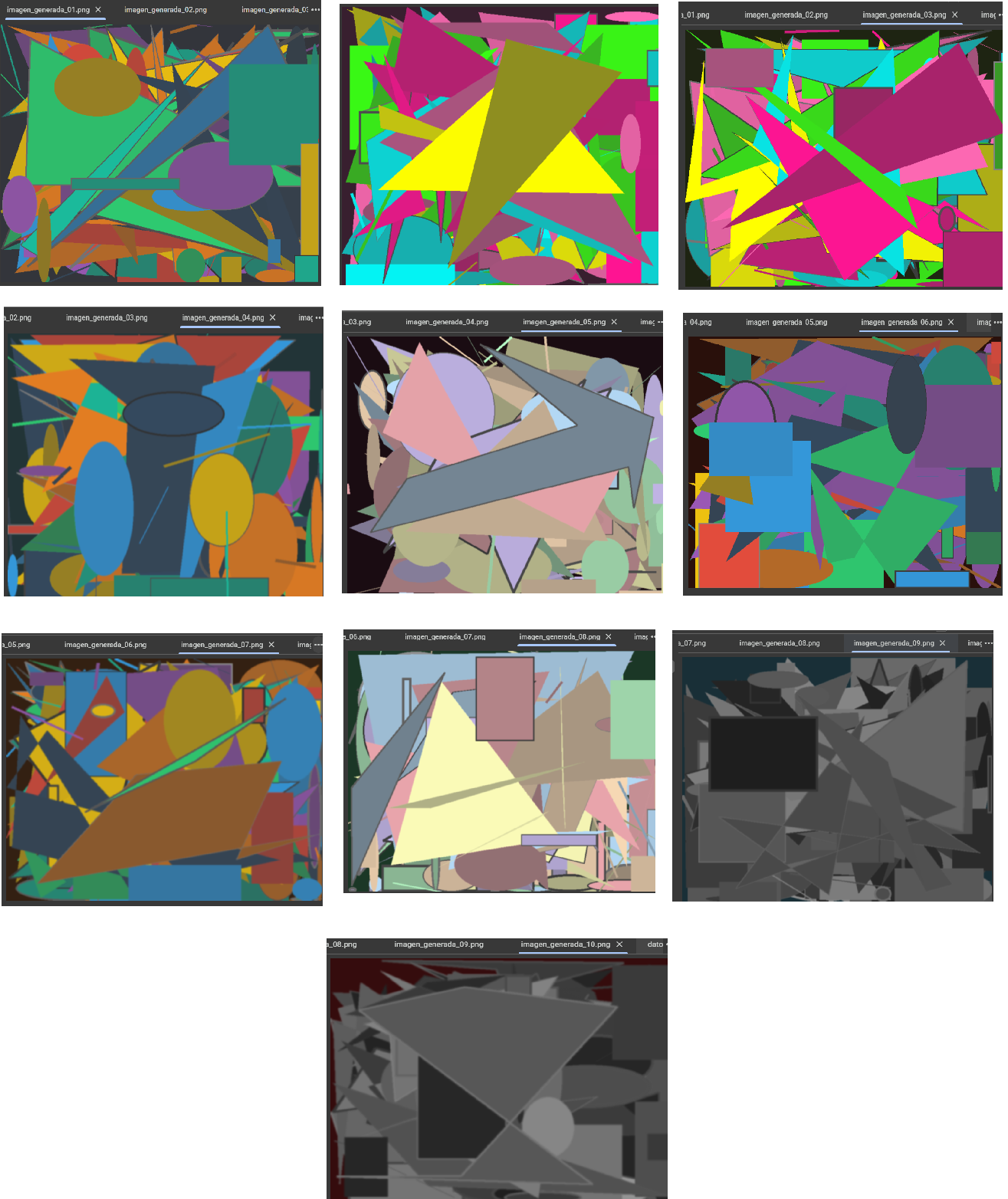
[**4.** **ANEXOS** 14](#_Toc206429271)

**PRACTICA #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.

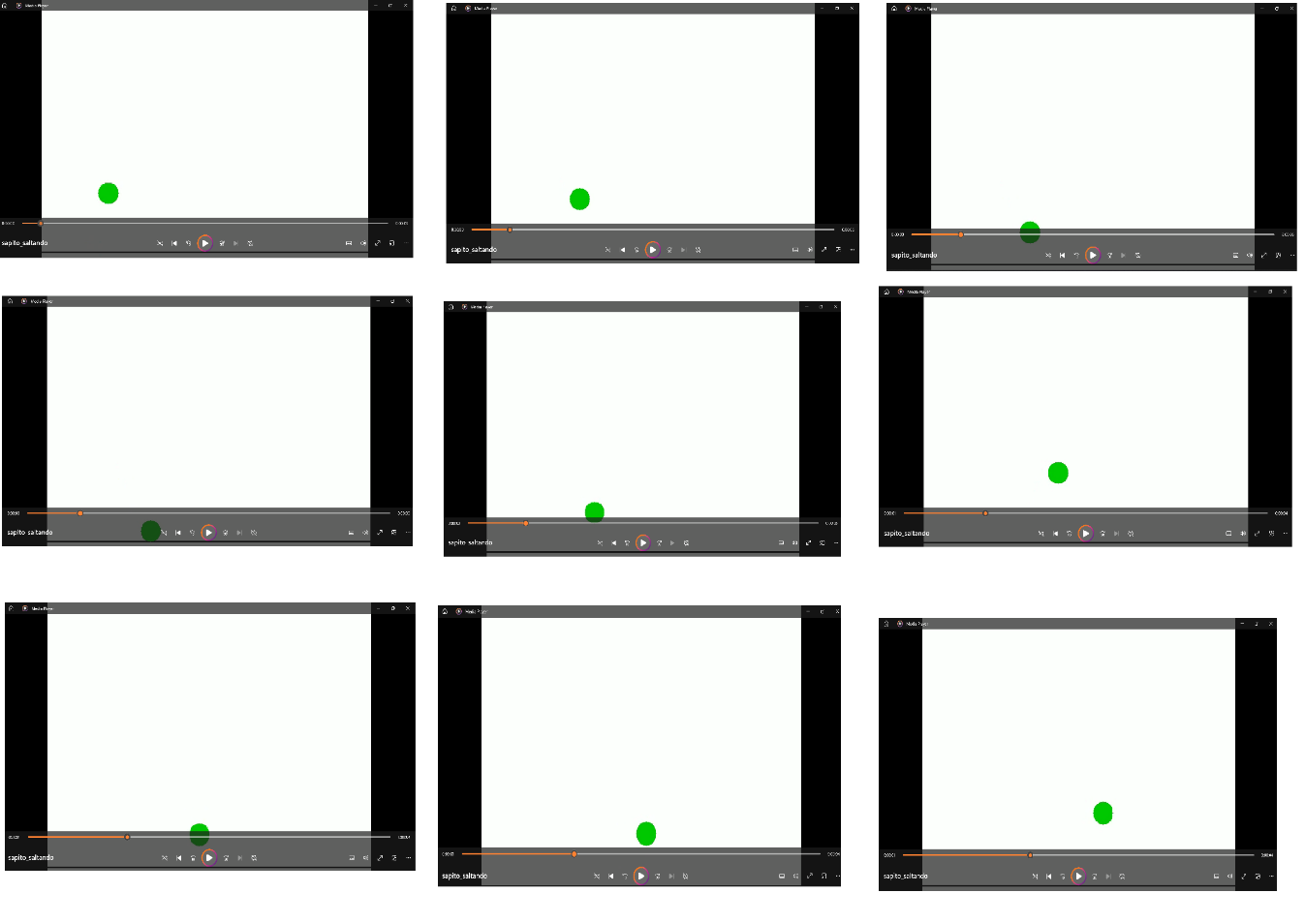
# **GENERADOR DE ANIMACIÓN**

**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")

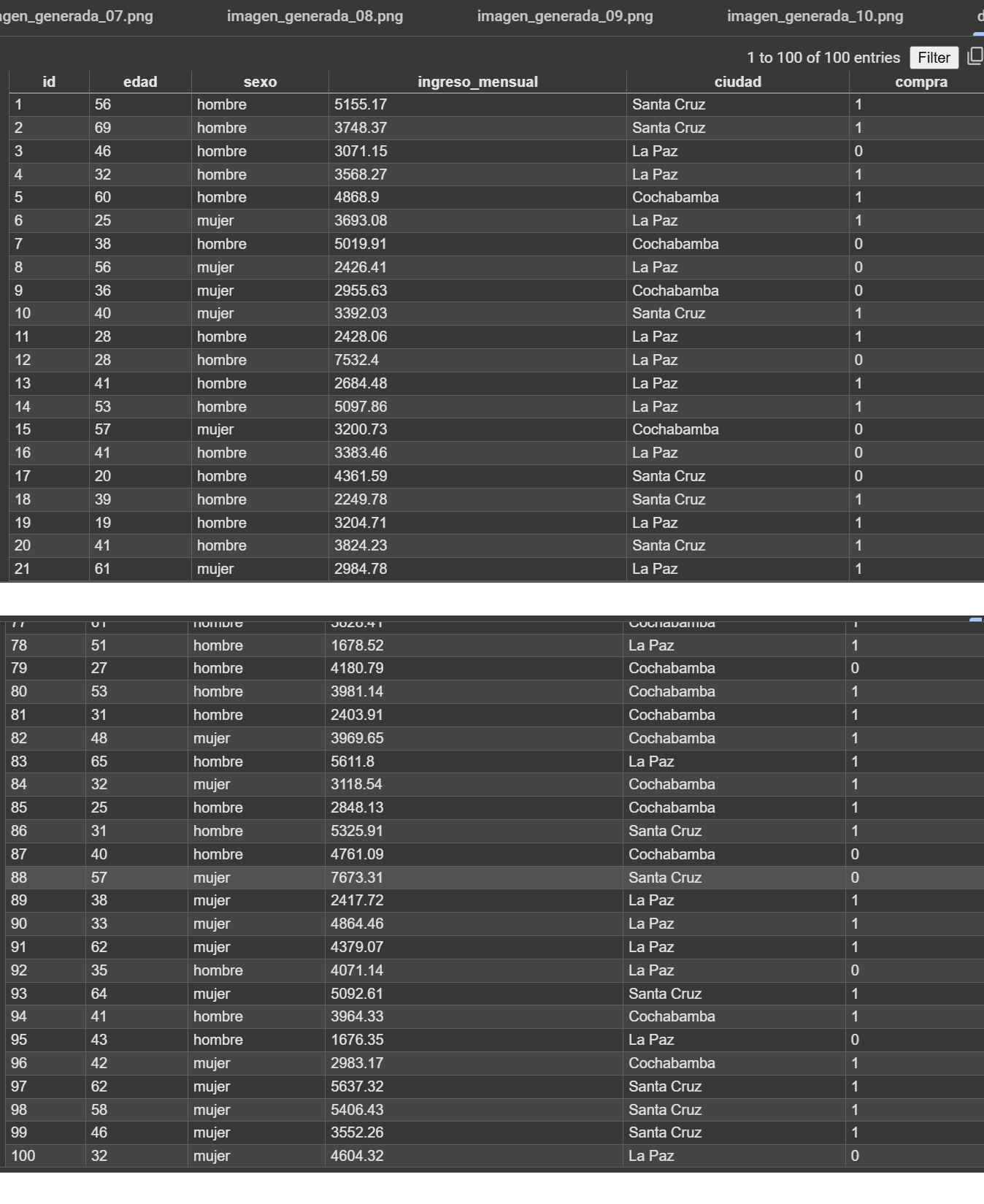
generate(cfg, "mi\_imagen.png")

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

# **GENERADOR DE DATOS SINTETICOS**

**RESULTADOS**

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



**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).

# **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'")