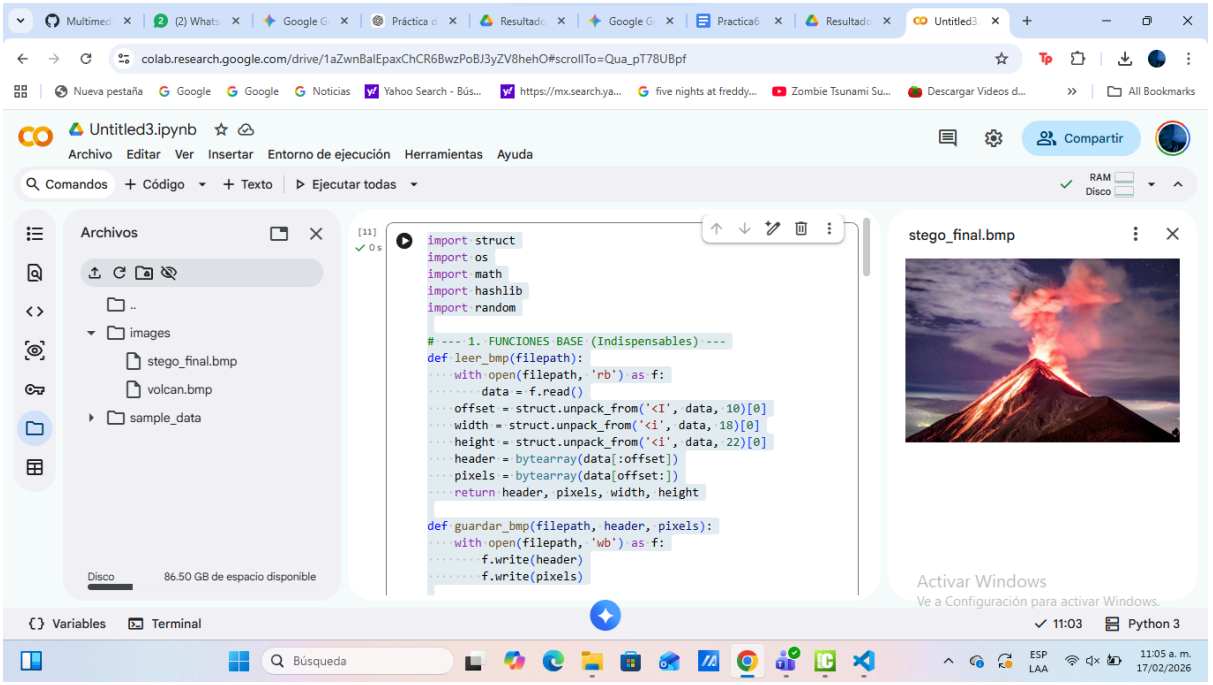
El programa no elige las posiciones al azar de forma caótica, sino de forma **determinista** mediante una clave.

- **Hashing del Password:** El password que elegiste se somete a una función **SHA-256**. Esto genera una huella digital única.
- **Generador Pseudoaleatorio:** Esa huella se usa como **semilla (seed)** para el generador de números aleatorios.
- **Sincronización:** Esto garantiza que la *misma* clave siempre generará la *misma* secuencia de posiciones. Sin la clave exacta, es matemáticamente imposible saber en qué píxeles se escondieron los bits, incluso si el atacante sabe que el programa usa este algoritmo.

```
import struct
import os
import math
import hashlib
import random

# --- 1. FUNCIONES BASE (Indispensables) ---
def leer_bmp(filepath):
    with open(filepath, 'rb') as f:
        data = f.read()
    offset = struct.unpack_from('<I', data, 10)[0]
    width = struct.unpack_from('<i', data, 18)[0]
    height = struct.unpack_from('<i', data, 22)[0]
    header = bytearray(data[:offset])
    pixels = bytearray(data[offset:])
    return header, pixels, width, height

def guardar_bmp(filepath, header, pixels):
    with open(filepath, 'wb') as f:
        f.write(header)
        f.write(pixels)

def calcular_psnr(original_path, stego_path):
    h1, p_orig, w, h = leer_bmp(original_path)
    h2, p_steg, w, h = leer_bmp(stego_path)
    mse = sum((a - b)**2 for a, b in zip(p_orig, p_steg)) /
(len(p_orig))
    if mse == 0: return float('inf')
    psnr = 10 * math.log10(255**2 / mse)
    return psnr

# --- 2. HERRAMIENTAS DE SEGURIDAD XOR Y ALEATORIEDAD ---
```

```
def cifrar_descifrar_xor(datos, password):
    hash_base = hashlib.sha256(password.encode()).digest()
    clave_extendida = (hash_base * (len(datos) // len(hash_base) +
1))[:len(datos)]
    return bytes(a ^ b for a, b in zip(datos, clave_extendida))

def obtener_posiciones_aleatorias(total_pixeles, cantidad_bits,
password):
    semilla = int(hashlib.sha256(password.encode()).hexdigest(), 16) %
(10**8)
    random.seed(semilla)
    return random.sample(range(total_pixeles), cantidad_bits)

# --- 3. ALGORITMO PRÁCTICA 2 (Secure Steganography) ---
def embed_secure(src_path, dst_path, mensaje, password):
    header, pixels, width, height = leer_bmp(src_path)
    msg_bytes = mensaje.encode('utf-8')
    msg_cifrado = cifrar_descifrar_xor(msg_bytes, password)

    datos_a_esconder = struct.pack('>I', len(msg_bytes)) + msg_cifrado
    bits = []
    for byte in datos_a_esconder:
        for i in range(7, -1, -1):
            bits.append((byte >> i) & 1)

    if len(bits) > len(pixels):
        raise ValueError("Imagen demasiado pequeña")

    posiciones = obtener_posiciones_aleatorias(len(pixels), len(bits),
password)
    pixels_mod = bytearray(pixels)
    for i, pos in enumerate(posiciones):
        pixels_mod[pos] = (pixels_mod[pos] & 0xFE) | bits[i]

    guardar_bmp(dst_path, header, pixels_mod)

def extract_secure(stego_path, password):
    header, pixels, width, height = leer_bmp(stego_path)

    pos_longitud = obtener_posiciones_aleatorias(len(pixels), 32,
password)
    len_bits = [pixels[pos] & 1 for pos in pos_longitud]
```

```
    msg_len = 0
    for b in len_bits:
        msg_len = (msg_len << 1) | b

    total_bits = 32 + (msg_len * 8)
    if total_bits > len(pixels):
        return "Error: Password incorrecto o imagen corrupta."

    todas_las_pos = obtener_posiciones_aleatorias(len(pixels),
total_bits, password)
    pos_mensaje = todas_las_pos[32:]

    bits_mensaje = [pixels[pos] & 1 for pos in pos_mensaje]
    msg_cifrado = bytearray()
    for i in range(0, len(bits_mensaje), 8):
        byte = 0
        for bit in bits_mensaje[i:i+8]:
            byte = (byte << 1) | bit
        msg_cifrado.append(byte)

    return cifrar_descifrar_xor(msg_cifrado, password).decode('utf-8')

# --- 4. EJECUCIÓN FINAL ---
try:
    if not os.path.exists('images'): os.makedirs('images')

    ORIGINAL = './images/volcan.bmp'
    STEGO_ADV = './images/stego_final.bmp'
    TEXTO = "Mensaje Cifrado UPIITA 2026"
    CLAVE = "mi_llave_maestra"

    # Ejecutar proceso
    embed_secure(ORIGINAL, STEGO_ADV, TEXTO, CLAVE)
    recuperado = extract_secure(STEGO_ADV, CLAVE)

    print(f"--- RESULTADO PRÁCTICA 2 ---")
    print(f"Mensaje recuperado: {recuperado}")
    print(f"PSNR: {calcular_psnr(ORIGINAL, STEGO_ADV):.2f} dB")

except Exception as e:
    print(f"Error: {e}. Revisa que 'volcan.bmp' esté en la carpeta
'images'.")
```


Untitled3.ipynb
Archivo Editar Ver Insertar Entorno de ejecución Herramientas Ayuda
Compartir
Comandos + Código + Texto Ejecutar todas
RAM
Disco
Archivos
..
images
stego_final.bmp
volcan.bmp
sample_data
Disco
86.50 GB de espacio disponible
[11]
0 s
import struct
import os
import math
import hashlib
import random
# --- 1. FUNCIONES BASE (Indispensables) ---
def leer_bmp(filepath):
with open(filepath, 'rb') as f:
data = f.read()
offset = struct.unpack_from('<I', data, 10)[0]
width = struct.unpack_from('<i', data, 18)[0]
height = struct.unpack_from('<i', data, 22)[0]
header = bytearray(data[:offset])
pixels = bytearray(data[offset:])
return header, pixels, width, height
def guardar_bmp(filepath, header, pixels):
with open(filepath, 'wb') as f:
f.write(header)
f.write(pixels)
stego_final.bmp
Activar Windows
Ve a Configuración para activar Windows.
Variables
Terminal
11:03
Python 3
Búsqueda
ESP
LAA
11:05 a. m.
17/02/2026