```python
import os
import cv2
import json
import matplotlib.pyplot as plt
import numpy as np


# Función para leer y parsear el JSON
def parse_json(json_path):
    with open(json_path, 'r') as f:
        data = json.load(f)

    mapping = {}
    for entry in data:
        original_image = os.path.basename(entry["Original_image"])  # Nombre de la imagen original
        colored_image = entry["drawing"]  # Nombre de la imagen coloreada
        mapping[original_image] = colored_image

    return mapping


# Función para generar una máscara desde una imagen coloreada
def generate_mask(image_path, save_path, visualize=False):
    """
    Genera una máscara a partir de una imagen coloreada y la guarda con valores escalados.

    Args:
        image_path (str): Ruta de la imagen coloreada.
        save_path (str): Ruta para guardar la máscara generada.
        visualize (bool): Si True, visualiza las máscaras generadas.
    """
    # Cargar la imagen coloreada
    image = cv2.imread(image_path)
    if image is None:
        print(f"Error: No se pudo cargar la imagen coloreada en {image_path}. Archivo omitido.")
        return

    # Convertir a espacio de color HSV
    hsv_image = cv2.cvtColor(image, cv2.COLOR_BGR2HSV)

    # Definir rangos de colores en HSV
    color_ranges = {
        1: {'lower': (15, 180, 180), 'upper': (45, 255, 255)},  # Amarillo (hueso)
        2: {'lower': (70, 180, 180), 'upper': (110, 255, 255)},  # Azul (tendón)
        3: {'lower': (130, 180, 180), 'upper': (170, 255, 255)},  # Morado (límites del tendón)
    }

    # Crear una máscara vacía
    mask = np.zeros(image.shape[:2], dtype=np.uint8)
```

```
    # Generar una máscara para cada clase
    for class_id, color_range in color_ranges.items():
        lower_bound = np.array(color_range['lower'], dtype=np.uint8)
        upper_bound = np.array(color_range['upper'], dtype=np.uint8)
        class_mask = cv2.inRange(hsv_image, lower_bound, upper_bound)  #
Detectar píxeles en el rango
        mask[class_mask > 0] = class_id  # Asignar la etiqueta
correspondiente

    # Escalar los valores de la máscara para visores estándar (1 → 85, 2
→ 170, 3 → 255)
    mask_scaled = (mask * 85).astype(np.uint8)

    # Guardar la máscara escalada
    cv2.imwrite(save_path, mask_scaled)
    print(f"Máscara escalada guardada en {save_path}")

    # Visualizar las máscaras generadas si visualize=True
    if visualize:
        plt.figure(figsize=(10, 5))
        plt.imshow(mask_scaled, cmap='gray')
        plt.title(f"Máscara Escalada para
{os.path.basename(image_path)}")
        plt.show()


# Directorios base
original_image_dir = r"C:\Users\ismae\Desktop\TFG\Python
Windows\frames"  # Carpeta con imágenes originales
colored_image_dir =
r"C:\Users\ismae\Desktop\22Noviembre\DATA_22_Noviembre\DATA-Tendon-
Long"   # Carpeta con imágenes coloreadas
mask_dir = r"C:\Users\ismae\Desktop\TFG\Mascaras"           # Carpeta
para guardar máscaras generadas
os.makedirs(mask_dir, exist_ok=True)

# Ruta al archivo JSON
json_path = r"C:\Users\ismae\Desktop\22Noviembre\DATA_22_Noviembre\DATA-
Tendon-Long\file_info.json"

# Parsear el JSON para obtener el mapeo
image_mapping = parse_json(json_path)

# Procesar todas las imágenes del JSON
for original_name, colored_name in image_mapping.items():
    # Construir las rutas dinámicamente
    original_image_path = os.path.join(original_image_dir, original_name)
    mask_path = os.path.join(mask_dir,
f"{os.path.splitext(original_name)[0]}_mask.png")
    colored_image_path = os.path.join(colored_image_dir, colored_name)

    # Verificar si las rutas son válidas
    if not os.path.exists(original_image_path):
        print(f"Archivo original no encontrado: {original_image_path}")
```

```
        continue

    if not os.path.exists(colored_image_path):
        print(f"Archivo coloreado no encontrado: {colored_image_path}")
        continue

    # Generar la máscara y visualizarla si es necesario
    generate_mask(colored_image_path, mask_path, visualize=False)
```