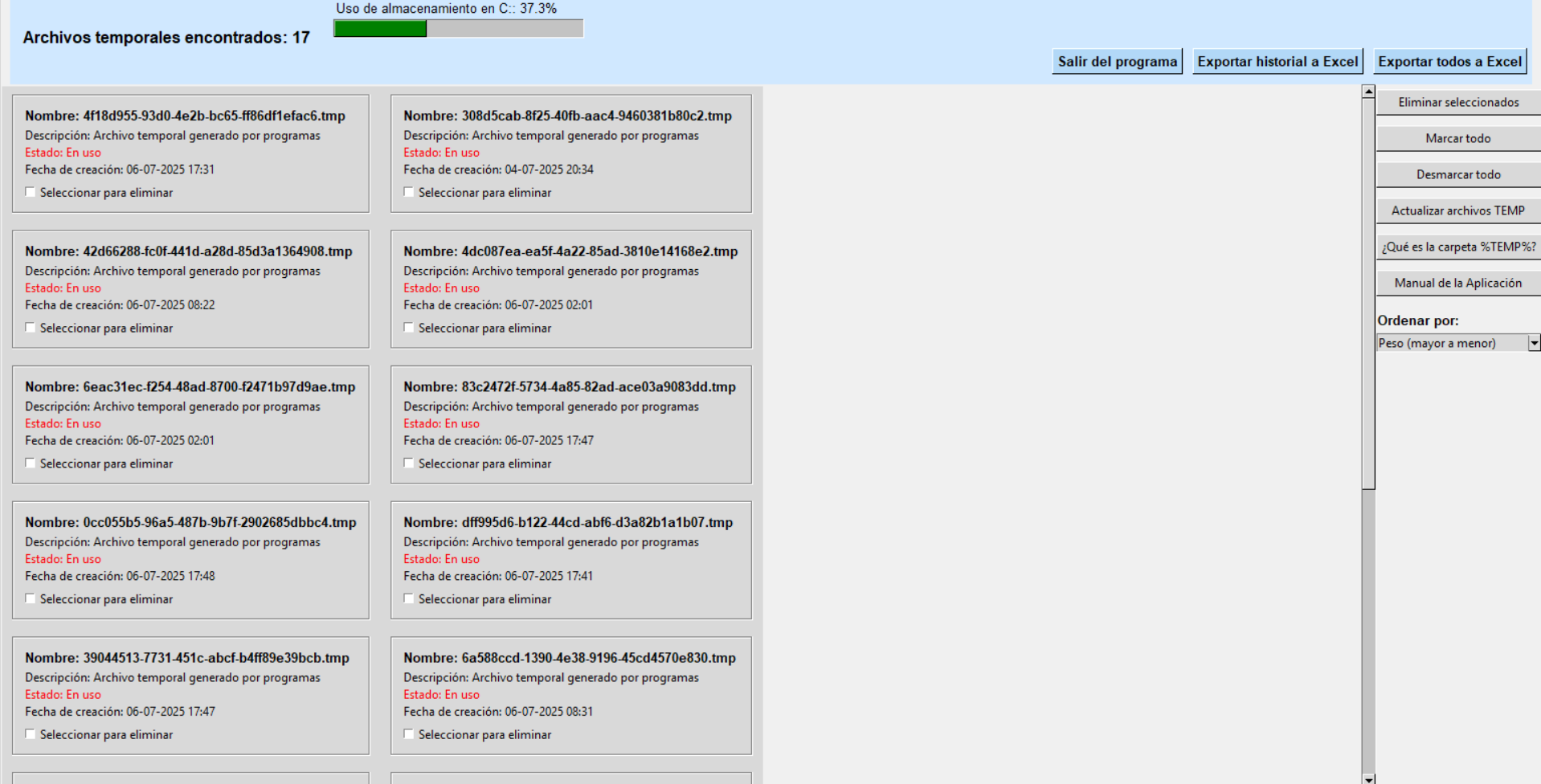
**CÓDIGO FUENTE**

****

|  |
| --- |
| **CÓDIGO FUENTE** |
| # -\*- coding: utf-8 -\*-  # --- IMPORTACIONES DE MÓDULOS NECESARIOS ---  # Se importan las bibliotecas que se usarán en el programa.  import os  # Para interactuar con el sistema operativo (acceder a archivos, rutas, etc.).  import tkinter as tk  # La biblioteca principal para crear la interfaz gráfica de usuario (GUI).  from tkinter import ttk, messagebox, filedialog  # Módulos específicos de tkinter para widgets mejorados, cuadros de diálogo y selección de archivos.  import psutil  # Para obtener información del sistema, como el uso del disco duro.  import textwrap  # Para formatear texto largo en múltiples líneas, útil para nombres de archivo largos.  from datetime import datetime  # Para trabajar con fechas y horas (ej. fecha de creación/eliminación de archivos).  import pandas as pd  # Para crear y manejar DataFrames, que facilitan la exportación a Excel.  import threading  # Para ejecutar tareas pesadas (como escanear archivos) en un hilo separado y no congelar la interfaz.  from openpyxl import load\_workbook  # Para cargar un archivo Excel existente y modificarlo.  from openpyxl.utils import get\_column\_letter  # Para convertir un número de columna (ej. 1) a su letra correspondiente en Excel (ej. 'A').  # --- VARIABLES GLOBALES Y DICCIONARIOS DE CONFIGURACIÓN ---  # Diccionario que asocia extensiones de archivo con una descripción de su propósito.  # Esto ayuda al usuario a entender qué tipo de archivo es.  purpose\_keywords = {      ".log": "Archivo de registro del sistema o aplicaciones",      ".tmp": "Archivo temporal generado por programas",      ".temp": "Archivo temporal generado automáticamente",      ".bak": "Copia de seguridad de un archivo antiguo",      ".dmp": "Volcado de memoria para depurar errores",      ".cache": "Archivo de caché usado para acelerar procesos",      ".old": "Versión anterior de un archivo reemplazado",      ".msi": "Instalador de software temporal",      ".config": "Archivo de configuración o parámetros",      ".json": "Datos temporales estructurados",      ".txt": "Archivo de texto auxiliar o log de errores",      ".zip": "Archivo comprimido generado temporalmente",      ".csv": "Datos temporales en tabla (Excel)",      ".etl": "Archivo de seguimiento de eventos del sistema",      ".wer": "Informe de errores de Windows",  }  # Diccionario que mapea el texto de las opciones del menú desplegable a un valor interno.  # Facilita el manejo de la lógica de ordenamiento de archivos.  opciones\_orden\_map = {      "Peso (mayor a menor)": "peso\_mayor",      "Peso (menor a mayor)": "peso\_menor",      "Fecha (más reciente)": "recientes",      "Fecha (más antiguo)": "antiguos",      "Nombre (A-Z)": "nombre\_az",      "Nombre (Z-A)": "nombre\_za"  }  # Obtiene la ruta de la carpeta de archivos temporales del sistema.  # Usa la variable de entorno "TEMP". Si no existe, usa "/tmp" como alternativa (común en Linux).  temp\_path = os.environ.get("TEMP", "/tmp")  # Lista para guardar un registro de los archivos que han sido eliminados durante la sesión.  historial\_eliminados = []  # Variables globales para llevar la cuenta de los archivos.  total\_archivos = 0  # Contador total de archivos encontrados.  checkbox\_vars = {}  # Diccionario para almacenar las variables de los checkboxes y los datos de cada archivo.  no\_accesibles = 0  # Contador de archivos que no se pudieron leer (por permisos, etc.).  archivos\_encontrados = []  # Lista que almacena la información de todos los archivos encontrados para poder exportarla.  # --- FUNCIONES DE LA APLICACIÓN ---  def mostrar\_ventana\_proceso():      """      Crea y muestra una pequeña ventana emergente con una barra de progreso.      Esta función se usa para indicar al usuario que una tarea larga (como escanear archivos) está en curso.      """      ventana\_proceso = tk.Toplevel(ventana)  # Crea una ventana secundaria.      ventana\_proceso.title("Cargando...")      ventana\_proceso.geometry("300x100")      ventana\_proceso.resizable(False, False)      ventana\_proceso.grab\_set()  # Bloquea la interacción con la ventana principal.      ventana\_proceso.attributes("-topmost", True)  # Mantiene la ventana siempre visible.      label = tk.Label(ventana\_proceso, text="En proceso, espera por favor...", font=("Arial", 10, "bold"))      label.pack(pady=(15, 5))      barra = ttk.Progressbar(ventana\_proceso, mode="indeterminate", length=250) # Barra de progreso indeterminada (se mueve constantemente).      barra.pack(pady=5)      barra.start(10)  # Inicia la animación de la barra.      return ventana\_proceso  # Devuelve la ventana para poder cerrarla después.  def actualizar\_barra\_almacenamiento():      """      Calcula el porcentaje de uso del disco donde se encuentra la carpeta TEMP y actualiza la barra de progreso de almacenamiento.      Cambia de color según el porcentaje de uso (verde, naranja, rojo).      """      disco = os.path.splitdrive(temp\_path)[0] or "C:"  # Obtiene la letra de la unidad (ej. "C:").      uso = psutil.disk\_usage(disco)  # Obtiene las estadísticas de uso del disco.      porcentaje = uso.percent  # Extrae el porcentaje de uso.      barra\_almacenamiento['value'] = porcentaje  # Actualiza el valor de la barra de progreso.      # Cambia el estilo (color) de la barra según el nivel de uso.      if porcentaje <= 50:          barra\_almacenamiento.configure(style="Verde.Horizontal.TProgressbar")      elif porcentaje <= 80:          barra\_almacenamiento.configure(style="Naranja.Horizontal.TProgressbar")      else:          barra\_almacenamiento.configure(style="Rojo.Horizontal.TProgressbar")      label\_almacenamiento.config(text=f"Uso de almacenamiento en {disco}: {porcentaje}%") # Actualiza el texto informativo.  def actualizar\_archivos():      """      Función principal que inicia el proceso de escaneo y listado de archivos.      Muestra la ventana de "Cargando" y ejecuta la tarea de escaneo en un hilo separado para no bloquear la GUI.      """      progress\_bar.pack(fill="x", padx=20, pady=(5, 10))  # Muestra la barra de progreso.      progress\_var.set(0) # Resetea la barra.      btn\_actualizar.config(state="disabled")  # Deshabilita el botón de actualizar para evitar múltiples clics.      ventana\_cargando = mostrar\_ventana\_proceso()  # Muestra la ventana de carga.      # Crea y ejecuta un nuevo hilo que llamará a la función de carga.      # Esto es crucial para mantener la interfaz receptiva.      threading.Thread(target=lambda: cargar\_archivos\_con\_progreso\_con\_ventana(ventana\_cargando)).start()  def cargar\_archivos\_con\_progreso\_con\_ventana(ventana\_cargando):      """      Función intermediaria que es llamada por el hilo.      Ejecuta la carga de archivos y, una vez terminada, cierra la ventana de "Cargando".      """      cargar\_archivos\_con\_progreso()  # Llama a la función que hace el trabajo pesado.      ventana.after(100, ventana\_cargando.destroy)  # Programa el cierre de la ventana de carga en el hilo principal de Tkinter.  def get\_file\_description(nombre):      """      Obtiene la descripción de un archivo basada en su extensión, usando el diccionario 'purpose\_keywords'.      """      ext = os.path.splitext(nombre)[1].lower()  # Obtiene la extensión del archivo en minúsculas (ej. ".log").      return purpose\_keywords.get(ext, "Archivo temporal no clasificado del sistema") # Devuelve la descripción o un texto por defecto.  def is\_file\_locked(filepath):      """      Verifica si un archivo está bloqueado o en uso por otro proceso.      Intenta renombrar el archivo a su mismo nombre. Si falla, es porque está bloqueado.      """      try:          os.rename(filepath, filepath)  # Intento de operación inofensiva.          return False  # Si tiene éxito, no está bloqueado.      except:          return True  # Si falla, está bloqueado.  def cargar\_archivos\_con\_progreso():      """      El corazón del programa. Escanea la carpeta TEMP, recopila información de cada archivo,      los ordena según la selección del usuario y los muestra en la interfaz.      """      global total\_archivos, no\_accesibles, archivos\_encontrados        # Limpia la lista de archivos de la interfaz y los datos anteriores.      for widget in frame\_archivos.winfo\_children():          widget.destroy()      checkbox\_vars.clear()      archivos = []      no\_accesibles = 0      try:          lista = os.listdir(temp\_path) # Obtiene una lista con todos los nombres de archivos y carpetas en TEMP.      except Exception as e:          messagebox.showerror("Error", f"No se puede acceder al directorio TEMP:\n{e}")          return      total\_items = len(lista)      procesados = 0      # Itera sobre cada elemento encontrado en la carpeta TEMP.      for archivo in lista:          ruta = os.path.join(temp\_path, archivo)  # Construye la ruta completa del archivo.          try:              if os.path.isfile(ruta): # Asegura que es un archivo y no una carpeta.                  # Recopila toda la información relevante del archivo en un diccionario.                  archivos.append({                      "ruta": ruta,                      "nombre": archivo,                      "descripcion": get\_file\_description(archivo),                      "estado": "En uso" if is\_file\_locked(ruta) else "Libre",                      "ram": "-",  # Placeholder, no se usa actualmente.                      "peso": "-", # Placeholder, se usa 'size' para el cálculo real.                      "fecha": datetime.fromtimestamp(os.path.getctime(ruta)).strftime("%d-%m-%Y %H:%M"),                      "size": os.path.getsize(ruta),  # Peso en bytes.                      "ctime": os.path.getctime(ruta) # Fecha de creación en formato timestamp para ordenar.                  })          except:              no\_accesibles += 1 # Incrementa el contador si no se puede acceder al archivo.          # Actualiza la barra de progreso en la interfaz.          procesados += 1          if total\_items:              progress = int(procesados / total\_items \* 100)              ventana.after(1, progress\_var.set, progress) # Se usa 'after' para actualizar desde el hilo principal de la GUI.      total\_archivos = len(archivos)      label\_total.config(text=f"Archivos temporales encontrados: {total\_archivos}") # Actualiza la etiqueta del contador.      # Ordena la lista de archivos según la opción seleccionada en el ComboBox.      orden = opciones\_orden\_map.get(combo\_orden.get(), "original")      if orden == "peso\_mayor":          archivos.sort(key=lambda x: x["size"], reverse=True)      elif orden == "peso\_menor":          archivos.sort(key=lambda x: x["size"])      elif orden == "recientes":          archivos.sort(key=lambda x: x["ctime"], reverse=True)      elif orden == "antiguos":          archivos.sort(key=lambda x: x["ctime"])      elif orden == "nombre\_az":          archivos.sort(key=lambda x: x["nombre"].lower())      elif orden == "nombre\_za":          archivos.sort(key=lambda x: x["nombre"].lower(), reverse=True)      archivos\_encontrados = archivos.copy() # Guarda una copia de los archivos para la función de exportar.      # Crea y muestra los widgets para cada archivo en una cuadrícula de 2 columnas.      columnas = 2      for idx, archivo in enumerate(archivos):          row = idx // columnas          col = idx % columnas          frame = ttk.Frame(frame\_archivos, borderwidth=2, relief="groove", padding=10) # Contenedor para cada archivo.          frame.grid(row=row, column=col, padx=10, pady=8, sticky="nsew")          nombre\_formateado = "\n".join(textwrap.wrap(archivo["nombre"], width=100)) # Ajusta nombres largos.          color\_estado = "green" if archivo["estado"] == "Libre" else "red"          # Crea las etiquetas con la información del archivo.          ttk.Label(frame, text=f"Nombre: {nombre\_formateado}", font=("Arial", 10, "bold")).pack(anchor="w")          ttk.Label(frame, text=f"Descripción: {archivo['descripcion']}", wraplength=400).pack(anchor="w", pady=(2, 0))          ttk.Label(frame, text=f"Estado: {archivo['estado']}", foreground=color\_estado).pack(anchor="w")          ttk.Label(frame, text=f"Fecha de creación: {archivo['fecha']}").pack(anchor="w")          # Crea el checkbox para seleccionar el archivo.          var = tk.BooleanVar()          chk = ttk.Checkbutton(frame, text="Seleccionar para eliminar", variable=var)          chk.pack(anchor="w", pady=(5, 0))          checkbox\_vars[archivo["ruta"]] = (var, archivo) # Almacena la variable y los datos del archivo.      # Muestra una advertencia si algunos archivos no se pudieron leer.      if no\_accesibles > 0:          messagebox.showwarning("Advertencia", f"{no\_accesibles} archivo(s) no pudieron accederse por permisos o bloqueo.")      # Oculta la barra de progreso y reactiva el botón de actualizar.      ventana.after(1, lambda: progress\_bar.pack\_forget())      btn\_actualizar.config(state="normal")  def cambio\_orden(event=None):      """      Se ejecuta cuando el usuario cambia la opción en el menú desplegable de orden.      Llama a 'actualizar\_archivos' para volver a cargar y mostrar la lista ordenada.      """      actualizar\_archivos()  def marcar\_todo():      """      Marca todos los checkboxes de la lista de archivos.      """      for var, \_ in checkbox\_vars.values():          var.set(True)  def desmarcar\_todo():      """      Desmarca todos los checkboxes de la lista de archivos.      """      for var, \_ in checkbox\_vars.values():          var.set(False)  def eliminar\_archivos():      """      Elimina los archivos que han sido seleccionados por el usuario.      Verifica si están bloqueados antes de intentar borrarlos.      """      # Filtra para obtener solo los archivos cuyo checkbox está marcado.      seleccionados = [(ruta, datos) for ruta, (var, datos) in checkbox\_vars.items() if var.get()]      total = len(seleccionados)      if total == 0:          messagebox.showinfo("Aviso", "No seleccionaste ningún archivo para eliminar.")          return      # Pide confirmación al usuario.      confirmar = messagebox.askyesno("Confirmar eliminación", f"¿Eliminar {total} archivo(s) seleccionados?")      if not confirmar:          return      eliminados = 0      errores = 0      for ruta, datos in seleccionados:          try:              if not is\_file\_locked(ruta): # Solo intenta eliminar si no está en uso.                  os.remove(ruta)                  eliminados += 1                  # Añade el archivo eliminado al historial.                  historial\_eliminados.append({                      "Nombre": datos["nombre"],                      "Descripción": datos["descripcion"],                      "Peso": datos["size"], # Se podría convertir a KB/MB                      "RAM estimada": datos["ram"],                      "Fecha de creación": datos["fecha"],                      "Fecha Eliminación": datetime.now().strftime("%d-%m-%Y %H:%M")                  })              else:                  errores += 1 # Cuenta como error si el archivo está en uso.          except:              errores += 1 # Cuenta como error si falla la eliminación por cualquier otra razón.        # Muestra un resumen de la operación y actualiza la lista de archivos.      messagebox.showinfo("Resultado", f"Eliminados: {eliminados}. Errores (en uso o protegidos): {errores}.")      actualizar\_archivos()      actualizar\_barra\_almacenamiento() # Actualiza también la barra de uso de disco.    def exportar\_historial():      """      Exporta el historial de archivos eliminados a un archivo Excel (.xlsx).      Ajusta automáticamente el ancho de las columnas.      """      if not historial\_eliminados:          messagebox.showinfo("Historial vacío", "Aún no has eliminado ningún archivo.")          return      # Pide al usuario que elija dónde guardar el archivo.      archivo = filedialog.asksaveasfilename(defaultextension=".xlsx", filetypes=[("Excel files", "\*.xlsx")])      if archivo:          df = pd.DataFrame(historial\_eliminados) # Convierte la lista de diccionarios a un DataFrame de pandas.          df.to\_excel(archivo, index=False) # Exporta el DataFrame a Excel.          # --- Código para autoajustar el ancho de las columnas en el Excel ---          wb = load\_workbook(archivo)          ws = wb.active          for col in ws.columns:              max\_length = 0              column\_letter = get\_column\_letter(col[0].column) # Obtiene la letra de la columna              for cell in col:                  try:                      if cell.value:                          # Encuentra la longitud máxima del contenido en la columna.                          max\_length = max(max\_length, len(str(cell.value)))                  except:                      pass              adjusted\_width = (max\_length + 2) # Añade un poco de espacio extra.              ws.column\_dimensions[column\_letter].width = adjusted\_width          wb.save(archivo) # Guarda los cambios en el archivo Excel.          messagebox.showinfo("Exportado", f"Historial exportado a:\n{archivo}")  def exportar\_todos\_a\_excel():      """      Exporta la lista completa de archivos temporales encontrados (no solo los eliminados) a un archivo Excel.      Similar a 'exportar\_historial' pero usa la lista 'archivos\_encontrados'.      """      if not archivos\_encontrados:          messagebox.showinfo("Sin datos", "No hay archivos para exportar. Primero actualiza la lista.")          return      archivo = filedialog.asksaveasfilename(defaultextension=".xlsx", filetypes=[("Excel files", "\*.xlsx")])      if archivo:          # Prepara los datos para el DataFrame.          datos = []          for a in archivos\_encontrados:              datos.append({                  "Nombre": a["nombre"],                  "Descripción": a["descripcion"],                  "Estado": a["estado"],                  "Peso (bytes)": a["size"],                  "Fecha de creación": a["fecha"],                  "Ruta": a["ruta"]              })          df = pd.DataFrame(datos)          df.to\_excel(archivo, index=False)          # Autoajusta el ancho de las columnas.          wb = load\_workbook(archivo)          ws = wb.active          for col in ws.columns:              max\_length = 0              column\_letter = get\_column\_letter(col[0].column)              for cell in col:                  try:                      if cell.value:                          max\_length = max(max\_length, len(str(cell.value)))                  except:                      pass              adjusted\_width = (max\_length + 2)              ws.column\_dimensions[column\_letter].width = adjusted\_width          wb.save(archivo)          messagebox.showinfo("Exportado", f"Todos los archivos exportados a:\n{archivo}")  def mostrar\_info\_temp():      """      Muestra una ventana informativa explicando qué es la carpeta TEMP.      """      info\_text = (          "¿QUÉ ES LA CARPETA %TEMP%?\n\n"          "La carpeta %TEMP% es un directorio del sistema operativo donde se almacenan archivos temporales creados por aplicaciones, instaladores, procesos del sistema, y otras actividades. Estos archivos sirven como soporte durante la ejecución de programas o tareas.\n\n"          "Con el tiempo, la carpeta %TEMP% puede llenarse de archivos innecesarios, lo que puede consumir espacio en disco y afectar el rendimiento del sistema.\n\n"          "Eliminar archivos temporales de forma segura puede ayudar a:\n"          "- Liberar espacio en disco.\n"          "- Mejorar el rendimiento del sistema.\n"          "- Eliminar restos de instalaciones incompletas o fallidas.\n\n"          "NOTA: Algunos archivos pueden estar en uso y no podrán eliminarse hasta que la aplicación asociada se cierre o el sistema se reinicie."      )      messagebox.showinfo("¿Qué es %TEMP%?", info\_text)  def mostrar\_manual():      """      Muestra una ventana con un breve manual de uso de la aplicación.      """      manual\_text = (          "MANUAL DE USO DEL GESTOR TEMPORAL\n\n"          "1. Presiona 'Actualizar archivos TEMP' para escanear la carpeta de archivos temporales.\n"          "2. Puedes ordenar los resultados por peso, nombre o fecha usando el menú desplegable.\n"          "3. Marca las casillas de los archivos que deseas eliminar.\n"          "4. Presiona 'Eliminar seleccionados' para limpiarlos.\n"          "5. Usa 'Exportar historial a Excel' si quieres guardar un registro de lo que se eliminó.\n"          "6. Puedes seleccionar o deseleccionar todos los archivos con un solo clic usando los botones correspondientes.\n"          "7. Observa el porcentaje de uso del almacenamiento en el disco que contiene la carpeta TEMP.\n"          "8. Usa el botón '¿Qué es la carpeta %TEMP%?' para más información sobre su propósito.\n"          "9. Cuando termines, presiona 'Salir del programa'.\n\n"          "¡Mantén tu sistema más limpio y ordenado con este gestor!"      )      messagebox.showinfo("Manual de Usuario", manual\_text)  def salir():      """      Cierra la aplicación.      """      ventana.quit()  # --- CONFIGURACIÓN DE LA INTERFAZ GRÁFICA (GUI) ---  # Creación de la ventana principal.  ventana = tk.Tk()  ventana.title("VoidCleanTempo")  ventana.geometry("1080x720")  # --- Estilos ---  # Creación y configuración de estilos para los widgets, como la barra de progreso.  style = ttk.Style()  style.theme\_use("default") # Se puede cambiar por 'clam', 'alt', etc.  style.configure("Verde.Horizontal.TProgressbar", foreground="green", background="green")  style.configure("Naranja.Horizontal.TProgressbar", foreground="orange", background="orange")  style.configure("Rojo.Horizontal.TProgressbar", foreground="red", background="red")  # --- Header (Cabecera) ---  # Frame superior que contiene información general y botones de acción.  header = tk.Frame(ventana, bg="#d0e8ff", height=60)  header.pack(fill="x", padx=10, pady=(0, 0))  label\_total = tk.Label(header, text="Archivos temporales encontrados: 0", bg="#d0e8ff", font=("Arial", 12, "bold"))  label\_total.pack(side="left", padx=10, pady=5)  label\_almacenamiento = tk.Label(header, text="Uso de almacenamiento: --%", bg="#d0e8ff", font=("Arial", 10))  label\_almacenamiento.pack(side="top", padx=10, pady=(5, 0), anchor="w")  barra\_almacenamiento = ttk.Progressbar(header, orient="horizontal", length=250, mode="determinate")  barra\_almacenamiento.pack(side="top", padx=10, anchor="w")  # Estilo para los botones principales para un aspecto uniforme.  estilo\_botones = {"bg": "#b2d8f7", "fg": "black", "activebackground": "#91c8f6", "relief": "raised", "bd": 1, "font": ("Arial", 10, "bold")}  # Botones en la cabecera, alineados a la derecha.  tk.Button(header, text="Salir del programa", command=salir, \*\*estilo\_botones).pack(side="right", padx=5, pady=10)  tk.Button(header, text="Exportar historial a Excel", command=exportar\_historial, \*\*estilo\_botones).pack(side="right", padx=5, pady=10)  tk.Button(header, text="Exportar todos a Excel", command=exportar\_todos\_a\_excel, \*\*estilo\_botones).pack(side="right", padx=5, pady=10)  # --- Contenedor Principal ---  # Frame que aloja la lista de archivos y el panel de botones lateral.  main\_frame = tk.Frame(ventana)  main\_frame.pack(fill="both", expand=True)  # --- Zona de la Lista de Archivos (con Scroll) ---  # Se usa un Canvas con una Scrollbar para poder desplazar la lista de archivos si es muy larga.  frame\_scroll = tk.Frame(main\_frame)  frame\_scroll.grid(row=0, column=0, sticky="nsew") # Se usa grid para el layout principal.  contenedor = tk.Canvas(frame\_scroll)  scrollbar = ttk.Scrollbar(frame\_scroll, orient="vertical", command=contenedor.yview)  frame\_archivos = ttk.Frame(contenedor) # Este es el frame que realmente contendrá los widgets de los archivos.  # Configuración para que la scrollbar funcione correctamente con el contenido dinámico.  frame\_archivos.bind("<Configure>", lambda e: contenedor.configure(scrollregion=contenedor.bbox("all")))  contenedor.create\_window((0, 0), window=frame\_archivos, anchor="nw")  contenedor.configure(yscrollcommand=scrollbar.set)  contenedor.pack(side="left", fill="both", expand=True)  scrollbar.pack(side="right", fill="y")  # Configuración del grid para que la zona de la lista se expanda.  main\_frame.grid\_rowconfigure(0, weight=1)  main\_frame.grid\_columnconfigure(0, weight=1)  # --- Panel de Botones Lateral ---  # Panel a la derecha con los controles principales.  panel\_botones = tk.Frame(main\_frame, bg="#f0f0f0", width=200)  panel\_botones.grid(row=0, column=1, sticky="ns") # Ocupa todo el alto de la fila.  btn\_eliminar = ttk.Button(panel\_botones, text="Eliminar seleccionados", command=eliminar\_archivos)  btn\_eliminar.pack(pady=5, padx=5, fill="x")  ttk.Button(panel\_botones, text="Marcar todo", command=marcar\_todo).pack(pady=5, padx=5, fill="x")  ttk.Button(panel\_botones, text="Desmarcar todo", command=desmarcar\_todo).pack(pady=5, padx=5, fill="x")  btn\_actualizar = ttk.Button(panel\_botones, text="Actualizar archivos TEMP", command=actualizar\_archivos)  btn\_actualizar.pack(pady=5, padx=5, fill="x")  ttk.Button(panel\_botones, text="¿Qué es la carpeta %TEMP%?", command=mostrar\_info\_temp).pack(pady=5, padx=5, fill="x")  ttk.Button(panel\_botones, text="Manual de la Aplicación", command=mostrar\_manual).pack(pady=5, padx=5, fill="x")  # --- Controles de Ordenamiento ---  ttk.Label(panel\_botones, text="Ordenar por:", background="#f0f0f0", font=("Arial", 10, "bold")).pack(pady=(10, 2), padx=5, fill="x")  combo\_orden = ttk.Combobox(panel\_botones, values=list(opciones\_orden\_map.keys()), state="readonly")  combo\_orden.set("Peso (mayor a menor)") # Opción por defecto.  combo\_orden.pack(pady=2, padx=5, fill="x")  combo\_orden.bind("<<ComboboxSelected>>", cambio\_orden) # Asocia la función 'cambio\_orden' al evento de selección.  # --- Barra de Progreso (para el escaneo) ---  progress\_var = tk.IntVar()  progress\_bar = ttk.Progressbar(panel\_botones, variable=progress\_var, maximum=100, mode="determinate")  # Esta barra no se muestra inicialmente, la función 'actualizar\_archivos' la hace visible.  # --- INICIO DE LA APLICACIÓN ---  # Llama a las funciones una vez al inicio para cargar la información inicial.  actualizar\_barra\_almacenamiento()  actualizar\_archivos()  # Inicia el bucle principal de la aplicación. La ventana permanecerá abierta y receptiva a eventos.  ventana.mainloop() |