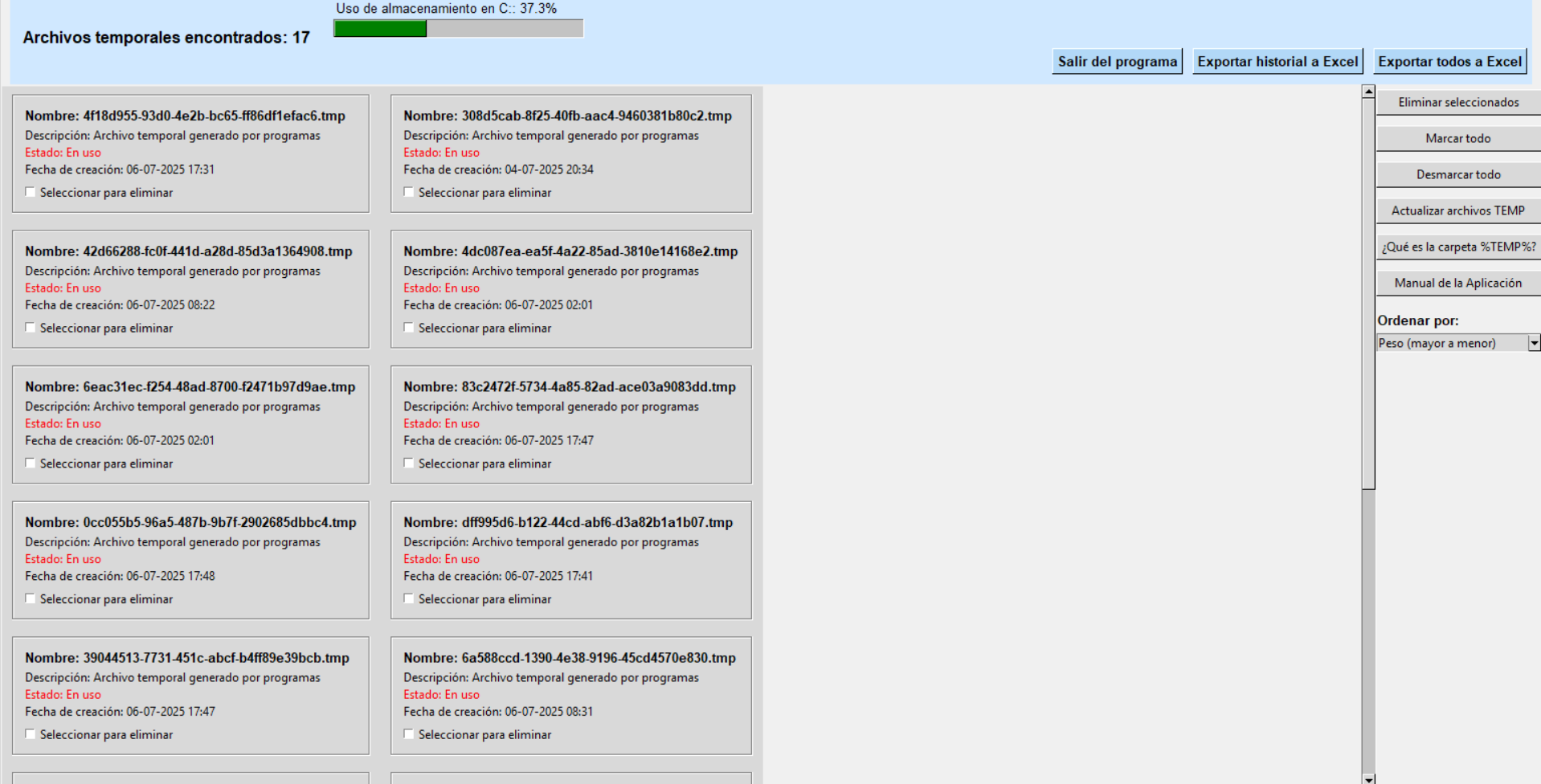
**CÓDIGO FUENTE**

****

|  |
| --- |
| **CÓDIGO FUENTE** |
| # -\*- coding: utf-8 -\*-  import os  import tkinter as tk  from tkinter import ttk, messagebox, filedialog  import psutil  import textwrap  from datetime import datetime  import pandas as pd  import threading  from openpyxl import load\_workbook  from openpyxl.utils import get\_column\_letter  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",  }  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"  }  temp\_path = os.environ.get("TEMP", "/tmp")  historial\_eliminados = []  total\_archivos = 0  checkbox\_vars = {}  no\_accesibles = 0  archivos\_encontrados = []  # <--- NUEVA VARIABLE GLOBAL  def mostrar\_ventana\_proceso():      ventana\_proceso = tk.Toplevel(ventana)      ventana\_proceso.title("Cargando...")      ventana\_proceso.geometry("300x100")      ventana\_proceso.resizable(False, False)      ventana\_proceso.grab\_set()      ventana\_proceso.attributes("-topmost", True)      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.pack(pady=5)      barra.start(10)      return ventana\_proceso  def actualizar\_barra\_almacenamiento():      disco = os.path.splitdrive(temp\_path)[0] or "C:"      uso = psutil.disk\_usage(disco)      porcentaje = uso.percent      barra\_almacenamiento['value'] = porcentaje      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}%")  def actualizar\_archivos():      progress\_bar.pack(fill="x", padx=20, pady=(5, 10))      progress\_var.set(0)      btn\_actualizar.config(state="disabled")      # Mostrar ventana emergente de proceso      ventana\_cargando = mostrar\_ventana\_proceso()      # Ejecutar carga en hilo separado      threading.Thread(target=lambda: cargar\_archivos\_con\_progreso\_con\_ventana(ventana\_cargando)).start()  def cargar\_archivos\_con\_progreso\_con\_ventana(ventana\_cargando):      cargar\_archivos\_con\_progreso()      ventana.after(100, ventana\_cargando.destroy)  def get\_file\_description(nombre):      ext = os.path.splitext(nombre)[1].lower()      return purpose\_keywords.get(ext, "Archivo temporal no clasificado del sistema")  def is\_file\_locked(filepath):      try:          os.rename(filepath, filepath)          return False      except:          return True  def cargar\_archivos\_con\_progreso():      global total\_archivos, no\_accesibles, archivos\_encontrados      for widget in frame\_archivos.winfo\_children():          widget.destroy()      checkbox\_vars.clear()      archivos = []      no\_accesibles = 0      try:          lista = os.listdir(temp\_path)      except Exception as e:          messagebox.showerror("Error", f"No se puede acceder al directorio TEMP:\n{e}")          return      total\_items = len(lista)      procesados = 0      for archivo in lista:          ruta = os.path.join(temp\_path, archivo)          try:              if os.path.isfile(ruta):                  archivos.append({                      "ruta": ruta,                      "nombre": archivo,                      "descripcion": get\_file\_description(archivo),                      "estado": "En uso" if is\_file\_locked(ruta) else "Libre",                      "ram": "-",                      "peso": "-",                      "fecha": datetime.fromtimestamp(os.path.getctime(ruta)).strftime("%d-%m-%Y %H:%M"),                      "size": os.path.getsize(ruta),                      "ctime": os.path.getctime(ruta)                  })          except:              no\_accesibles += 1          procesados += 1          if total\_items:              progress = int(procesados / total\_items \* 100)              ventana.after(1, progress\_var.set, progress)      total\_archivos = len(archivos)      label\_total.config(text=f"Archivos temporales encontrados: {total\_archivos}")      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()  # <--- ACTUALIZA LA VARIABLE GLOBAL      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)          frame.grid(row=row, column=col, padx=10, pady=8, sticky="nsew")          nombre\_formateado = "\n".join(textwrap.wrap(archivo["nombre"], width=100))          color\_estado = "green" if archivo["estado"] == "Libre" else "red"          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")          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)      if no\_accesibles > 0:          messagebox.showwarning("Advertencia", f"{no\_accesibles} archivo(s) no pudieron accederse por permisos o bloqueo.")      ventana.after(1, lambda: progress\_bar.pack\_forget())      btn\_actualizar.config(state="normal")  def cambio\_orden(event=None):      actualizar\_archivos()  def marcar\_todo():      for var, \_ in checkbox\_vars.values():          var.set(True)  def desmarcar\_todo():      for var, \_ in checkbox\_vars.values():          var.set(False)  def eliminar\_archivos():      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      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):                  os.remove(ruta)                  eliminados += 1                  historial\_eliminados.append({                      "Nombre": datos["nombre"],                      "Descripción": datos["descripcion"],                      "Peso": datos["peso"],                      "RAM estimada": datos["ram"],                      "Fecha de creación": datos["fecha"],                      "Fecha Eliminación": datetime.now().strftime("%d-%m-%Y %H:%M")                  })              else:                  errores += 1          except:              errores += 1      messagebox.showinfo("Resultado", f"Eliminados: {eliminados}. Errores: {errores}.")      actualizar\_archivos()    def exportar\_historial():      if not historial\_eliminados:          messagebox.showinfo("Historial vacío", "Aún no has eliminado ningún archivo.")          return      archivo = filedialog.asksaveasfilename(defaultextension=".xlsx", filetypes=[("Excel files", "\*.xlsx")])      if archivo:          df = pd.DataFrame(historial\_eliminados)          df.to\_excel(archivo, index=False)          wb = load\_workbook(archivo)          ws = wb.active          for col in ws.columns:              max\_length = 0              column = 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[get\_column\_letter(column)].width = adjusted\_width          wb.save(archivo)          messagebox.showinfo("Exportado", f"Historial exportado a:\n{archivo}")  def exportar\_todos\_a\_excel():      if not archivos\_encontrados:          messagebox.showinfo("Sin datos", "No hay archivos para exportar.")          return      archivo = filedialog.asksaveasfilename(defaultextension=".xlsx", filetypes=[("Excel files", "\*.xlsx")])      if archivo:          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)          wb = load\_workbook(archivo)          ws = wb.active          for col in ws.columns:              max\_length = 0              column = 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[get\_column\_letter(column)].width = adjusted\_width          wb.save(archivo)          messagebox.showinfo("Exportado", f"Todos los archivos exportados a:\n{archivo}")  def mostrar\_info\_temp():      messagebox.showinfo("¿Qué es %TEMP%?", "¿QUÉ ES LA CARPETA %TEMP%?\nLa 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\nCon el tiempo, la carpeta %TEMP% puede llenarse de archivos innecesarios, lo que puede consumir espacio en disco y afectar el rendimiento del sistema.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.NOTA: Algunos archivos pueden estar en uso y no podrán eliminarse hasta que la aplicación asociada se cierre o el sistema se reinicie.")  def mostrar\_manual():      messagebox.showinfo("Manual", "MANUAL DE USO DEL GESTOR TEMPORAL\n1. Presiona 'Actualizar archivos TEMP' para escanear la carpeta de archivos temporales.\n2. Puedes ordenar los resultados por peso, nombre o fecha usando el menú desplegable.\n3. Marca las casillas de los archivos que deseas eliminar.\n4. Presiona 'Eliminar seleccionados' para limpiarlos.\n5. Usa 'Exportar historial a Excel' si quieres guardar un registro de lo que se eliminó.\n6. Puedes seleccionar o deseleccionar todos los archivos con un solo clic usando los botones correspondientes.\n7. Observa el porcentaje de uso del almacenamiento en el disco que contiene la carpeta TEMP.\n8. Usa el botón '¿Qué es la carpeta %TEMP%?' para más información sobre su propósito.\n9. Cuando termines, presiona 'Salir del programa'.¡Mantén tu sistema más limpio y ordenado con este gestor!")  # INTERFAZ  def salir():      ventana.quit()  # INTERFAZ  ventana = tk.Tk()  ventana.title("Gestor Rápido de Archivos TEMP")  ventana.geometry("1080x720")  style = ttk.Style()  style.theme\_use("default")  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 = 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\_botones = {"bg": "#b2d8f7", "fg": "black", "activebackground": "#91c8f6", "relief": "raised", "bd": 1, "font": ("Arial", 10, "bold")}  tk.Button(header, text="Exportar todos a Excel", command=exportar\_todos\_a\_excel, \*\*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="Salir del programa", command=salir, \*\*estilo\_botones).pack(side="right", padx=5, pady=10)  main\_frame = tk.Frame(ventana)  main\_frame.pack(fill="both", expand=True)  frame\_scroll = tk.Frame(main\_frame)  frame\_scroll.grid(row=0, column=0, sticky="nsew")  contenedor = tk.Canvas(frame\_scroll)  scrollbar = ttk.Scrollbar(frame\_scroll, orient="vertical", command=contenedor.yview)  frame\_archivos = ttk.Frame(contenedor)  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")  main\_frame.grid\_rowconfigure(0, weight=1)  main\_frame.grid\_columnconfigure(0, weight=1)  panel\_botones = tk.Frame(main\_frame, bg="#f0f0f0", width=200)  panel\_botones.grid(row=0, column=1, sticky="ns")  btn\_eliminar = ttk.Button(panel\_botones, text="Eliminar seleccionados", command=eliminar\_archivos)  btn\_eliminar.pack(pady=5, fill="x")  ttk.Button(panel\_botones, text="Marcar todo", command=marcar\_todo).pack(pady=5, fill="x")  ttk.Button(panel\_botones, text="Desmarcar todo", command=desmarcar\_todo).pack(pady=5, fill="x")  btn\_actualizar = ttk.Button(panel\_botones, text="Actualizar archivos TEMP", command=actualizar\_archivos)  btn\_actualizar.pack(pady=5, fill="x")  ttk.Button(panel\_botones, text="¿Qué es la carpeta %TEMP%?", command=mostrar\_info\_temp).pack(pady=5, fill="x")  ttk.Button(panel\_botones, text="Manual de la Aplicación", command=mostrar\_manual).pack(pady=5, fill="x")  ttk.Label(panel\_botones, text="Ordenar por:", background="#f0f0f0", font=("Arial", 10, "bold")).pack(pady=(10, 2), fill="x")  combo\_orden = ttk.Combobox(panel\_botones, values=list(opciones\_orden\_map.keys()), state="readonly")  combo\_orden.set("Peso (mayor a menor)")  combo\_orden.pack(pady=2, fill="x")  combo\_orden.bind("<<ComboboxSelected>>", cambio\_orden)  progress\_var = tk.IntVar()  progress\_bar = ttk.Progressbar(panel\_botones, variable=progress\_var, maximum=100, mode="determinate")  actualizar\_barra\_almacenamiento()  actualizar\_archivos()  ventana.mainloop() |