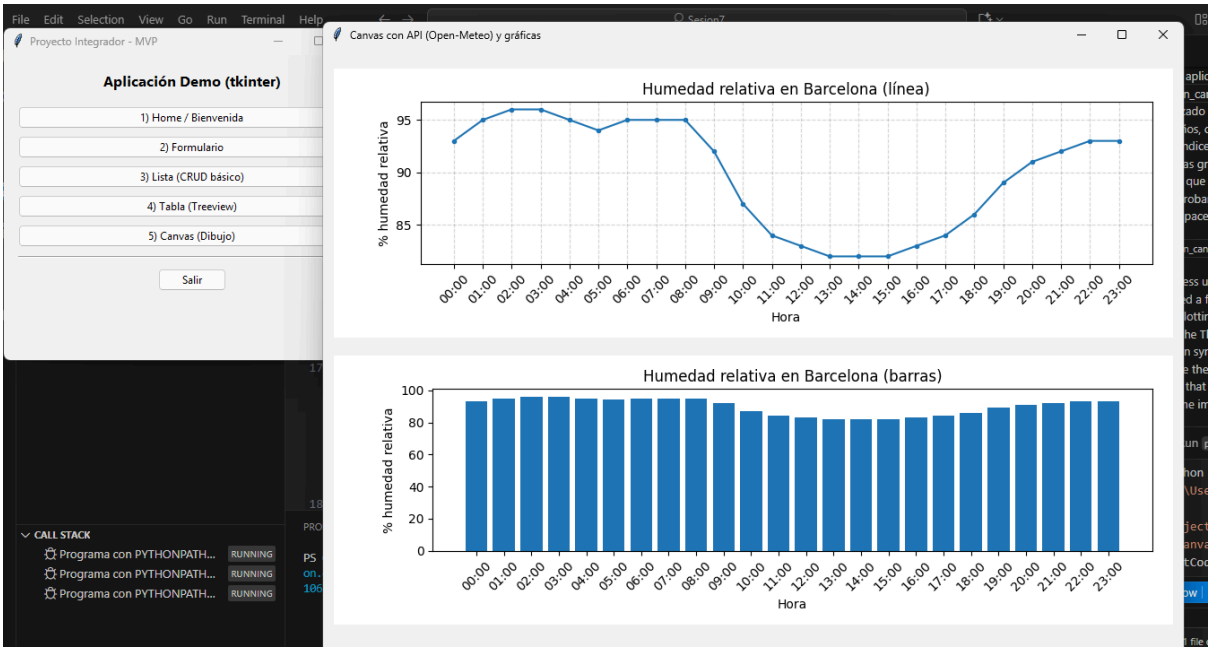
```
import tkinter as tk
from tkinter import ttk, messagebox
import requests
import matplotlib
# Ensure TkAgg backend for embedding in tkinter (local GUIs usually
support this).
matplotlib.use("TkAgg")
import matplotlib.pyplot as plt
from matplotlib.backends.backend_tkagg import FigureCanvasTkAgg


def fetch_data():
    """
    Conecta con la API de Open-Meteo y obtiene temperaturas horarias
    de León, Gto (últimas 24 horas).
    Devuelve dos listas: horas y temperaturas.
    """
    try:
        url = "https://api.open-meteo.com/v1/forecast"
        params = {
            "latitude": 41.404151903488994,
            "longitude": 2.145659784405041,
            "hourly": "relativehumidity_2m",
            "timezone": "auto",
        }
        response = requests.get(url, params=params, timeout=15)
        response.raise_for_status()
        data = response.json()

        # Extraer de forma segura para evitar KeyError si la API cambia
        hourly = data.get("hourly") or {}
        horas = list(hourly.get("time", []) or [])
        temperaturas = list(hourly.get("relativehumidity_2m", []) or
[])

        # Normalizar longitudes y quedarnos con las últimas 24 horas si
hay más
        min_len = min(len(horas), len(temperaturas))
        if min_len == 0:
            raise ValueError("La respuesta de la API no contiene datos
horarios esperados.")
```

```
        # Tomar sólo las últimas 24 mediciones para evitar ejes x muy
cargados
        start = max(0, min_len - 24)
        horas = horas[start:min_len]
        temperaturas = temperaturas[start:min_len]

        # Convertir temperaturas a float y filtrar entradas inválidas
manteniendo la correspondencia
        horas_clean = []
        temps_clean = []
        for h, t in zip(horas, temperaturas):
            try:
                tf = float(t)
            except Exception:
                continue
            horas_clean.append(h)
            temps_clean.append(tf)

        if not horas_clean or not temps_clean:
            raise ValueError("Los datos horarios de la API no contienen
valores numéricos válidos.")

        return horas_clean, temps_clean
    except Exception as e:
        messagebox.showerror("Error", f"No se pudieron obtener los
datos:\n{e}")
        return [], []


def create_line_chart(horas, temps):
    """Gráfica de línea."""
    fig, ax = plt.subplots(figsize=(6, 3))
    x = list(range(len(horas)))
    ax.plot(x, temps, linestyle="-", marker="o", markersize=3)
    ax.grid(True, linestyle="--", alpha=.5)
    ax.set_title("Humedad relativa en Barcelona (línea)")
    ax.set_xlabel("Hora")
    ax.set_ylabel("% humedad relativa")
    # Mostrar sólo la parte de hora para las etiquetas X
    labels = [h.split('T')[-1] if 'T' in h else h for h in horas]
    ax.set_xticks(x)
    ax.set_xticklabels(labels, rotation=45)
    fig.tight_layout()
```

```
    return fig


def create_bar_chart(horas, temps):
    """Gráfica de barras."""
    fig, ax = plt.subplots(figsize=(6, 3))
    x = list(range(len(horas)))
    ax.bar(x, temps)
    ax.set_title("Humedad relativa en Barcelona (barras)")
    ax.set_xlabel("Hora")
    ax.set_ylabel("% humedad relativa")
    labels = [h.split('T')[-1] if 'T' in h else h for h in horas]
    ax.set_xticks(x)
    ax.set_xticklabels(labels, rotation=45)
    fig.tight_layout()
    return fig


def mostrar_graficas(frm, horas, temps):
    """Inserta las gráficas en el frame de la ventana tkinter.

    Este método limpia el contenedor antes de dibujar para evitar
apilar
    widgets en llamadas sucesivas y captura excepciones para que la GUI
    no se rompa en caso de error.
    """
    try:
        # Limpiar contenedor de gráficas (no debería contener el botón)
        for w in frm.winfo_children():
            w.destroy()

        # Línea
        fig1 = create_line_chart(horas, temps)
        canvas1 = FigureCanvasTkAgg(fig1, master=frm)
        canvas1.draw()
        canvas1.get_tk_widget().pack(pady=10, fill="x")

        # Barras
        fig2 = create_bar_chart(horas, temps)
        canvas2 = FigureCanvasTkAgg(fig2, master=frm)
        canvas2.draw()
        canvas2.get_tk_widget().pack(pady=10, fill="x")
    except Exception as e:
```

```
        # No dejar que un error de dibujo cierre la ventana; informar
al usuario
        messagebox.showerror("Error al mostrar gráficas", f"Ocurrió un
error al dibujar las gráficas:\n{e}")


def open_win_canvas(parent: tk.Tk):
    """
    Crea la ventana secundaria con gráficas de la API.
    """
    win = tk.Toplevel(parent)
    win.title("Canvas con API (Open-Meteo) y gráficas")
    win.geometry("960x1000")

    frm = ttk.Frame(win, padding=12)
    frm.pack(fill="both", expand=True)

    # Botón para cargar datos y graficar
    def cargar():
        horas, temps = fetch_data()
        if horas and temps:
            mostrar_graficas(frm, horas, temps)

    ttk.Button(frm, text="Cargar y mostrar gráficas",
command=cargar).pack(pady=10)


# Para pruebas independientes (opcional)
if __name__ == "__main__":
    root = tk.Tk()
    root.title("Prueba win_canvas")
    ttk.Button(root, text="Abrir ventana Canvas", command=lambda:
open_win_canvas(root)).pack(pady=20)
    root.mainloop()
```



En el código hice cambios en la latitud, longitud y titulo de la ciudad, cambiando todos estos por los de la ciudad “Barcelona” en España.

Cambie un parametro para que fuera `hourly=relativehumidity_2m en vez de temperature_2m`

`Agregue rejillas usando el comando ax.grid(True, linestyle="--", alpha=.5)`