📚 Tutorial Completo de Tkinter de Python

MANIPULACION Y PRESERVACION DE DATOS. PROFE: AGUSTIN GATICA

🔹 MÓDULO 1 — Fundamentos de Tkinter

1.1 ¿Qué es Tkinter y cómo funciona internamente?

Tkinter es la biblioteca GUI estándar de Python. Funciona mediante:

- **Event Loop (mainloop)**: Bucle infinito que espera eventos del usuario
- **Arquitectura basada en eventos**: Cada interacción dispara callbacks
- **Bindings**: Asociación entre eventos y funciones

```
# estructura_basica.py
import tkinter as tk

# Crear ventana raíz
root = tk.Tk()
root.title("Mi primera ventana")
root.geometry("400x300")

# Event loop - espera eventos
root.mainloop()  # Bloquea hasta cerrar ventana
```

1.2 Crear ventana básica

```
# ventana_basica.py
import tkinter as tk

root = tk.Tk()
root.title("Ventana Básica")
root.geometry("500x400")
root.resizable(False, False)  # No redimensionable

# Configurar icono (opcional)
# root.iconbitmap("ruta/icono.ico")

# Color de fondo
```

```
root.configure(bg="#f0f0f0")

root.mainloop()
```

1.3 Widgets básicos

```
# widgets_basicos.py
import tkinter as tk

root = tk.Tk()
root.title("Widgets Básicos")
root.geometry("400x300")

# 1. LABEL (etiqueta)
label = tk.Label(
    root,
    text="Hola Tkinter",
    font=("Arial", 16, "bold"),
    fg="white",
    bg="#2196F3",
    padx=10,
    pady=10
)
label.pack()

# 2. BUTTON (botón)
def saludar():
    print("¡Botón presionado!")

button = tk.Button(
    root,
    text="Haz clic aquí",
    command=saludar,
    bg="#4CAF50",
    fg="white",
    padx=20,
    pady=10,
    font=("Arial", 12)
)
button.pack(pady=10)

# 3. ENTRY (entrada de texto)
entry = tk.Entry(
    root,
    width=30,
    font=("Arial", 12),
    bd=2
)
entry.pack(pady=10)
```

```
entry.insert(0, "Escribe aquí...")

root.mainloop()
```

1.4 Métodos de layout: pack(), grid(), place()

```
# layout_comparison.py
import tkinter as tk

# ===== PACK =====
# Simpler, apila widgets verticalmente
root1 = tk.Tk()
root1.title("Pack")

tk.Label(root1, text="Widget 1", bg="red", fg="white", padx=20, pady=20).pack()
tk.Label(root1, text="Widget 2", bg="blue", fg="white", padx=20, pady=20).pack()
tk.Label(root1, text="Widget 3", bg="green", fg="white", padx=20, pady=20).pack()

# ===== GRID =====
# Basado en filas/columnas, más flexible
root2 = tk.Tk()
root2.title("Grid")

tk.Label(root2, text="Row 0, Col 0", bg="red", fg="white", padx=20, pady=20).grid(row=0, colum
tk.Label(root2, text="Row 0, Col 1", bg="blue", fg="white", padx=20, pady=20).grid(row=0, colu
tk.Label(root2, text="Row 1, Col 0", bg="green", fg="white", padx=20, pady=20).grid(row=1, col

# ===== PLACE =====
# Posicionamiento absoluto (coordenadas)
root3 = tk.Tk()
root3.title("Place")
root3.geometry("400x300")

tk.Label(root3, text="Pos (50,50)", bg="red", fg="white", padx=20, pady=20).place(x=50, y=50)
tk.Label(root3, text="Pos (200,150)", bg="blue", fg="white", padx=20, pady=20).place(x=200, y=

# RECOMENDACIÓN: Usar grid() para layouts profesionales
```

1.5 Sistema de eventos y funciones callback

```
# eventos_callbacks.py
import tkinter as tk

root = tk.Tk()
root.title("Sistema de Eventos")
```

```
root.geometry("400x300")

# Variable para almacenar datos
resultado = tk.StringVar(value="")

# Callback - función que se ejecuta al evento
def on_button_click():
    resultado.set("¡Botón presionado!")

def on_entry_change(event):
    texto = entrada.get()
    resultado.set(f"Escribiste: {texto}")

def on_key_press(event):
    resultado.set(f"Tecla presionada: {event.char}")

# Widgets
entrada = tk.Entry(root, font=("Arial", 12))
entrada.pack(pady=10)
entrada.bind("<KeyRelease>", on_entry_change)  # Evento cuando se suelta tecla
entrada.bind("<Return>", on_key_press)  # Enter específicamente

boton = tk.Button(root, text="Click", command=on_button_click)
boton.pack(pady=10)

etiqueta = tk.Label(root, textvariable=resultado, font=("Arial", 14), fg="blue")
etiqueta.pack(pady=10)

root.mainloop()
```

1.6 Mini proyecto: Calculadora simple

```
# calculadora_simple.py
import tkinter as tk
from tkinter import messagebox

class Calculadora:
    def __init__(self, root):
        self.root = root
        self.root.title("Calculadora Simple")
        self.root.geometry("300x400")
        self.root.resizable(False, False)

        self.expresion = ""
        self.setup_ui()

    def setup_ui(self):
        # Pantalla de resultado
        self.pantalla = tk.Entry(
```

```
            self.root,
            font=("Arial", 20),
            justify="right",
            bd=2,
            state="readonly"
        )
        self.pantalla.pack(fill="both", padx=10, pady=10)

        # Frame para botones
        frame_botones = tk.Frame(self.root)
        frame_botones.pack(fill="both", expand=True, padx=10, pady=10)

        # Botones
        botones = [
            ["7", "8", "9", "/"],
            ["4", "5", "6", "*"],
            ["1", "2", "3", "-"],
            ["0", ".", "=", "+"]
        ]

        for fila_idx, fila in enumerate(botones):
            for col_idx, btn_text in enumerate(fila):
                btn = tk.Button(
                    frame_botones,
                    text=btn_text,
                    font=("Arial", 18),
                    command=lambda x=btn_text: self.on_button_click(x)
                )
                btn.grid(row=fila_idx, column=col_idx, sticky="nsew", padx=5, pady=5)
                frame_botones.grid_rowconfigure(fila_idx, weight=1)
                frame_botones.grid_columnconfigure(col_idx, weight=1)

        # Botón limpiar
        btn_limpiar = tk.Button(
            self.root,
            text="C",
            font=("Arial", 14),
            command=self.limpiar,
            bg="#ff5252",
            fg="white"
        )
        btn_limpiar.pack(fill="x", padx=10, pady=5)

    def on_button_click(self, char):
        if char == "=":
            try:
                resultado = eval(self.expresion)
                self.expresion = str(resultado)
            except:
                messagebox.showerror("Error", "Expresión inválida")
                self.expresion = ""
        else:
            self.expresion += char
```

```
        self.actualizar_pantalla()

    def limpiar(self):
        self.expresion = ""
        self.actualizar_pantalla()

    def actualizar_pantalla(self):
        self.pantalla.config(state="normal")
        self.pantalla.delete(0, tk.END)
        self.pantalla.insert(0, self.expresion)
        self.pantalla.config(state="readonly")

if __name__ == "__main__":
    root = tk.Tk()
    app = Calculadora(root)
    root.mainloop()
```

🔹 MÓDULO 2 — Manejo avanzado de Widgets

2.1 Frame y organización modular

```
# frames_modular.py
import tkinter as tk

root = tk.Tk()
root.title("Organización con Frames")
root.geometry("500x400")

# Frame superior (header)
frame_header = tk.Frame(root, bg="#2196F3", height=100)
frame_header.pack(fill="x")

tk.Label(frame_header, text="Título Principal", font=("Arial", 20, "bold"),
         fg="white", bg="#2196F3").pack(pady=20)

# Frame principal (contenido)
frame_content = tk.Frame(root, bg="white")
frame_content.pack(fill="both", expand=True, padx=10, pady=10)

tk.Label(frame_content, text="Contenido aquí", font=("Arial", 14)).pack()

# Frame inferior (footer)
frame_footer = tk.Frame(root, bg="#f0f0f0", height=50)
frame_footer.pack(fill="x", side="bottom")

tk.Label(frame_footer, text="© 2026 - Mi Aplicación",
         font=("Arial", 10), bg="#f0f0f0").pack(pady=10)
```

```
root.mainloop()
```

2.2 Widget Text (editor de texto)

```
# widget_text.py
import tkinter as tk

root = tk.Tk()
root.title("Widget Text")
root.geometry("600x400")

# Text widget
text = tk.Text(root, font=("Arial", 11), wrap="word", undo=True, redo=True)
text.pack(fill="both", expand=True, padx=10, pady=10)

# Funciones para manipular texto
def obtener_texto():
    contenido = text.get("1.0", tk.END)
    print("Contenido:", contenido)

def insertar_texto():
    text.insert(tk.END, "\n--- Texto insertado ---\n")

def limpiar():
    text.delete("1.0", tk.END)

# Botones
frame_botones = tk.Frame(root)
frame_botones.pack(fill="x", padx=10, pady=5)

tk.Button(frame_botones, text="Obtener", command=obtener_texto).pack(side="left", padx=5)
tk.Button(frame_botones, text="Insertar", command=insertar_texto).pack(side="left", padx=5)
tk.Button(frame_botones, text="Limpiar", command=limpiar).pack(side="left", padx=5)

root.mainloop()
```

2.3 Listbox (lista de elementos)

```
# widget_listbox.py
import tkinter as tk

root = tk.Tk()
root.title("Widget Listbox")
root.geometry("400x400")
```

```
# Listbox con scrollbar
frame = tk.Frame(root)
frame.pack(fill="both", expand=True, padx=10, pady=10)

scrollbar = tk.Scrollbar(frame)
scrollbar.pack(side="right", fill="y")

listbox = tk.Listbox(frame, yscrollcommand=scrollbar.set, font=("Arial", 11))
listbox.pack(side="left", fill="both", expand=True)
scrollbar.config(command=listbox.yview)

# Agregar elementos
items = ["Python", "JavaScript", "Java", "C++", "Go", "Rust", "Swift"]
for item in items:
    listbox.insert(tk.END, item)

# Funciones
def obtener_seleccion():
    try:
        seleccion = listbox.curselection()
        if seleccion:
            print(f"Seleccionado: {listbox.get(seleccion[0])}")
    except:
        pass

def agregar_item():
    nueva = entrada.get()
    if nueva:
        listbox.insert(tk.END, nueva)
        entrada.delete(0, tk.END)

def eliminar_item():
    try:
        indice = listbox.curselection()[0]
        listbox.delete(indice)
    except:
        pass

# Botones
frame_botones = tk.Frame(root)
frame_botones.pack(fill="x", padx=10, pady=5)

entrada = tk.Entry(frame_botones, font=("Arial", 11))
entrada.pack(side="left", fill="x", expand=True, padx=5)

tk.Button(frame_botones, text="Agregar", command=agregar_item).pack(side="left", padx=5)
tk.Button(frame_botones, text="Eliminar", command=eliminar_item).pack(side="left", padx=5)
tk.Button(frame_botones, text="Ver", command=obtener_seleccion).pack(side="left", padx=5)

root.mainloop()
```

2.4 Checkbutton y Radiobutton

```
# checkbutton_radiobutton.py
import tkinter as tk

root = tk.Tk()
root.title("Check y Radio Buttons")
root.geometry("400x300")

# Variables
var_check1 = tk.BooleanVar()
var_check2 = tk.BooleanVar()
var_radio = tk.StringVar(value="opcion1")

# Checkbuttons
tk.Label(root, text="Selecciona opciones (múltiple):", font=("Arial", 12, "bold")).pack(anchor

check1 = tk.Checkbutton(root, text="Opción 1", variable=var_check1)
check1.pack(anchor="w", padx=40)

check2 = tk.Checkbutton(root, text="Opción 2", variable=var_check2)
check2.pack(anchor="w", padx=40)

# Radiobuttons
tk.Label(root, text="Selecciona una opción (exclusiva):", font=("Arial", 12, "bold")).pack(anc

radio1 = tk.Radiobutton(root, text="Python", variable=var_radio, value="python")
radio1.pack(anchor="w", padx=40)

radio2 = tk.Radiobutton(root, text="JavaScript", variable=var_radio, value="javascript")
radio2.pack(anchor="w", padx=40)

radio3 = tk.Radiobutton(root, text="Java", variable=var_radio, value="java")
radio3.pack(anchor="w", padx=40)

# Botón para ver resultados
def ver_selecciones():
    print(f"Check 1: {var_check1.get()}")
    print(f"Check 2: {var_check2.get()}")
    print(f"Radio: {var_radio.get()}")

tk.Button(root, text="Ver Selecciones", command=ver_selecciones).pack(pady=20)

root.mainloop()
```

2.5 Variables Tkinter (StringVar, IntVar, BooleanVar)

```python
# variables_tkinter.py
import tkinter as tk

root = tk.Tk()
root.title("Variables Tkinter")
root.geometry("400x300")

# Variables con traza (observables)
nombre = tk.StringVar(value="")
edad = tk.IntVar(value=0)
suscrito = tk.BooleanVar(value=False)

# Callbacks cuando cambian variables
def on_nombre_change(*args):
    print(f"Nombre cambió a: {nombre.get()}")

def on_edad_change(*args):
    print(f"Edad cambió a: {edad.get()}")

# Trazar cambios
nombre.trace("w", on_nombre_change)
edad.trace("w", on_edad_change)

# Widgets vinculados a variables
tk.Label(root, text="Nombre:").pack()
tk.Entry(root, textvariable=nombre).pack(padx=10, pady=5)

tk.Label(root, text="Edad:").pack()
tk.Entry(root, textvariable=edad).pack(padx=10, pady=5)

tk.Checkbutton(root, text="Suscrito", variable=suscrito).pack(pady=10)

# Mostrar valores
def mostrar_valores():
    print(f"\nDatos finales:")
    print(f"Nombre: {nombre.get()}")
    print(f"Edad: {edad.get()}")
    print(f"Suscrito: {suscrito.get()}")

tk.Button(root, text="Guardar", command=mostrar_valores).pack(pady=10)

root.mainloop()
```

2.6 Validación de datos

```python
# validacion_datos.py
import tkinter as tk
from tkinter import messagebox
```

```
root = tk.Tk()
root.title("Validación de Datos")
root.geometry("400x300")

# Validar solo números
def validar_solo_numeros(valor):
    return valor.isdigit() or valor == ""

# Validar email
def validar_email(email):
    return "@" in email and "." in email.split("@")[1]

# Registrar función de validación
vcmd_numeros = (root.register(validar_solo_numeros), "%S")

# Entrada con validación
tk.Label(root, text="Edad (solo números):", font=("Arial", 12)).pack(pady=10)
entrada_edad = tk.Entry(root, validate="key", validatecommand=vcmd_numeros)
entrada_edad.pack(padx=20, pady=5)

tk.Label(root, text="Email:", font=("Arial", 12)).pack(pady=10)
entrada_email = tk.Entry(root)
entrada_email.pack(padx=20, pady=5)

def validar_formulario():
    edad = entrada_edad.get()
    email = entrada_email.get()

    if not edad:
        messagebox.showerror("Error", "Ingresa la edad")
        return

    if not email:
        messagebox.showerror("Error", "Ingresa el email")
        return

    if not validar_email(email):
        messagebox.showerror("Error", "Email inválido")
        return

    messagebox.showinfo("Éxito", f"Datos válidos!\nEdad: {edad}\nEmail: {email}")

tk.Button(root, text="Enviar", command=validar_formulario, bg="#4CAF50", fg="white").pack(pady

root.mainloop()
```

2.7 Mini proyecto: Formulario con validación

```python
# formulario_validacion.py
import tkinter as tk
from tkinter import messagebox
import re

class FormularioApp:
    def __init__(self, root):
        self.root = root
        self.root.title("Formulario de Registro")
        self.root.geometry("450x500")
        self.root.resizable(False, False)

        self.variables = {
            "nombre": tk.StringVar(),
            "email": tk.StringVar(),
            "telefono": tk.StringVar(),
            "edad": tk.IntVar(),
            "pais": tk.StringVar(value="Selecciona"),
            "genero": tk.StringVar(value="M"),
            "terminos": tk.BooleanVar()
        }

        self.setup_ui()

    def setup_ui(self):
        # Título
        tk.Label(self.root, text="FORMULARIO DE REGISTRO", font=("Arial", 16, "bold"),
                 bg="#2196F3", fg="white").pack(fill="x", pady=10)

        # Frame principal
        frame = tk.Frame(self.root)
        frame.pack(fill="both", expand=True, padx=20, pady=20)

        # Nombre
        tk.Label(frame, text="Nombre:", font=("Arial", 11)).grid(row=0, column=0, sticky="w",
        tk.Entry(frame, textvariable=self.variables["nombre"], width=30).grid(row=0, column=1,

        # Email
        tk.Label(frame, text="Email:", font=("Arial", 11)).grid(row=1, column=0, sticky="w", p
        tk.Entry(frame, textvariable=self.variables["email"], width=30).grid(row=1, column=1,

        # Teléfono
        tk.Label(frame, text="Teléfono:", font=("Arial", 11)).grid(row=2, column=0, sticky="w"
        tk.Entry(frame, textvariable=self.variables["telefono"], width=30).grid(row=2, column=

        # Edad
        tk.Label(frame, text="Edad:", font=("Arial", 11)).grid(row=3, column=0, sticky="w", pa
        tk.Entry(frame, textvariable=self.variables["edad"], width=30).grid(row=3, column=1, p

        # País
        tk.Label(frame, text="País:", font=("Arial", 11)).grid(row=4, column=0, sticky="w", pa
        paises = ["Selecciona", "México", "España", "Argentina", "Colombia", "Chile"]
```

```
        tk.OptionMenu(frame, self.variables["pais"], *paises).grid(row=4, column=1, sticky="w"

        # Género
        tk.Label(frame, text="Género:", font=("Arial", 11)).grid(row=5, column=0, sticky="w",
        tk.Radiobutton(frame, text="Masculino", variable=self.variables["genero"], value="M").
        tk.Radiobutton(frame, text="Femenino", variable=self.variables["genero"], value="F").g

        # Términos
        tk.Checkbutton(frame, text="Acepto términos y condiciones",
                       variable=self.variables["terminos"]).grid(row=7, column=0, columnspan=2,

        # Botones
        frame_botones = tk.Frame(self.root)
        frame_botones.pack(fill="x", padx=20, pady=10)

        tk.Button(frame_botones, text="Guardar", command=self.guardar,
                  bg="#4CAF50", fg="white", padx=20).pack(side="left", padx=5)
        tk.Button(frame_botones, text="Limpiar", command=self.limpiar,
                  bg="#2196F3", fg="white", padx=20).pack(side="left", padx=5)

    def validar(self):
        nombre = self.variables["nombre"].get().strip()
        email = self.variables["email"].get().strip()
        telefono = self.variables["telefono"].get().strip()
        edad = self.variables["edad"].get()
        pais = self.variables["pais"].get()
        terminos = self.variables["terminos"].get()

        if not nombre or len(nombre) < 3:
            messagebox.showerror("Error", "Nombre inválido (mínimo 3 caracteres)")
            return False

        if not re.match(r'^[\w\.-]+@[\w\.-]+\.\w+$', email):
            messagebox.showerror("Error", "Email inválido")
            return False

        if not telefono.isdigit() or len(telefono) < 10:
            messagebox.showerror("Error", "Teléfono inválido")
            return False

        try:
            edad_num = int(edad)
            if edad_num < 18 or edad_num > 120:
                raise ValueError
        except:
            messagebox.showerror("Error", "Edad debe ser entre 18 y 120")
            return False

        if pais == "Selecciona":
            messagebox.showerror("Error", "Selecciona un país")
            return False

        if not terminos:
```

```
            messagebox.showerror("Error", "Debes aceptar los términos")
            return False

        return True

    def guardar(self):
        if self.validar():
            datos = {k: v.get() for k, v in self.variables.items()}
            messagebox.showinfo("Éxito", f"Datos guardados:\n{datos}")

    def limpiar(self):
        for var in self.variables.values():
            if isinstance(var, tk.BooleanVar):
                var.set(False)
            elif isinstance(var, tk.IntVar):
                var.set(0)
            else:
                var.set("" if var != self.variables["pais"] else "Selecciona")

if __name__ == "__main__":
    root = tk.Tk()
    app = FormularioApp(root)
    root.mainloop()
```

🔹 MÓDULO 3 — Layout profesional

3.1 Grid avanzado con peso de filas y columnas

```
# grid_avanzado.py
import tkinter as tk

root = tk.Tk()
root.title("Grid Avanzado - Responsive")
root.geometry("600x400")

# Configurar pesos para responsividad
root.grid_rowconfigure(0, weight=0)      # Header - no crece
root.grid_rowconfigure(1, weight=1)      # Content - crece
root.grid_rowconfigure(2, weight=0)      # Footer - no crece
root.grid_columnconfigure(0, weight=1)   # Única columna crece

# Header
header = tk.Frame(root, bg="#2196F3", height=80)
header.grid(row=0, column=0, sticky="nsew")
header.grid_propagate(False)

tk.Label(header, text="Dashboard Principal", font=("Arial", 18, "bold"),
         fg="white", bg="#2196F3").pack(pady=20)
```

```
# Content con 3 columnas
content = tk.Frame(root, bg="white")
content.grid(row=1, column=0, sticky="nsew", padx=10, pady=10)

# Configurar pesos en grid del content
content.grid_columnconfigure(0, weight=1)
content.grid_columnconfigure(1, weight=1)
content.grid_columnconfigure(2, weight=1)
content.grid_rowconfigure(0, weight=1)

# 3 paneles
for i in range(3):
    panel = tk.Frame(content, bg=["#4CAF50", "#2196F3", "#FF9800"][i], relief="raised", bd=2)
    panel.grid(row=0, column=i, sticky="nsew", padx=5)

    tk.Label(panel, text=f"Panel {i+1}", font=("Arial", 14, "bold"),
             fg="white", bg=["#4CAF50", "#2196F3", "#FF9800"][i]).pack(expand=True)

# Footer
footer = tk.Frame(root, bg="#f0f0f0", height=50)
footer.grid(row=2, column=0, sticky="nsew")
footer.grid_propagate(False)

tk.Label(footer, text="© 2026 - Aplicación Responsive", bg="#f0f0f0").pack(pady=15)

root.mainloop()
```

3.2 ttk vs widgets clásicos

```
# ttk_vs_clasicos.py
import tkinter as tk
from tkinter import ttk

root = tk.Tk()
root.title("TTK vs Widgets Clásicos")
root.geometry("600x500")

# Widgets clásicos
frame_clasicos = tk.LabelFrame(root, text="Widgets Clásicos (tk)", font=("Arial", 12, "bold"))
frame_clasicos.pack(fill="x", padx=10, pady=10)

tk.Button(frame_clasicos, text="Botón Clásico", bg="#2196F3", fg="white").pack(pady=5)
tk.Entry(frame_clasicos, font=("Arial", 11)).pack(pady=5, padx=10, fill="x")

combo_clasico = tk.Listbox(frame_clasicos, height=3)
combo_clasico.pack(pady=5, padx=10, fill="x")
for item in ["Opción 1", "Opción 2", "Opción 3"]:
    combo_clasico.insert(tk.END, item)
```

```
# Widgets ttk (temáticos)
frame_ttk = tk.LabelFrame(root, text="Widgets TTK (temáticos)", font=("Arial", 12, "bold"))
frame_ttk.pack(fill="x", padx=10, pady=10)

ttk.Button(frame_ttk, text="Botón TTK").pack(pady=5)
ttk.Entry(frame_ttk, font=("Arial", 11)).pack(pady=5, padx=10, fill="x")

combo_ttk = ttk.Combobox(frame_ttk, values=["Opción 1", "Opción 2", "Opción 3"], state="readon
combo_ttk.pack(pady=5, padx=10, fill="x")

# Progreso
ttk.Label(frame_ttk, text="Progreso:").pack(pady=(10, 5), padx=10, anchor="w")
progress = ttk.Progressbar(frame_ttk, length=300, mode="determinate", value=65)
progress.pack(pady=5, padx=10, fill="x")

# Tema
ttk.Label(root, text="✓ TTK se adapta al tema del sistema operativo", font=("Arial", 11, "ital

root.mainloop()
```

3.3 Temas visuales

```
# temas_visuales.py
import tkinter as tk
from tkinter import ttk

root = tk.Tk()
root.title("Temas TTK")
root.geometry("500x400")

# Obtener temas disponibles
temas_disponibles = ttk.Style().theme_names()

tk.Label(root, text=f"Temas disponibles: {', '.join(temas_disponibles)}",
         font=("Arial", 11)).pack(pady=10)

# Frame actual
current_theme = tk.StringVar(value=ttk.Style().theme_use())

def cambiar_tema(tema):
    ttk.Style().theme_use(tema)
    current_theme.set(tema)
    tk.Label(root, text=f"Tema actual: {tema}", font=("Arial", 12, "bold"),
             fg="green").pack(pady=10)

# Botones para cambiar tema
frame_botones = tk.Frame(root)
frame_botones.pack(fill="x", padx=20, pady=10)
```

```
for tema in temas_disponibles:
    ttk.Button(frame_botones, text=tema, command=lambda t=tema: cambiar_tema(t)).pack(pady=5)

# Ejemplo de componentes
ttk.Label(root, text="Ejemplo de componentes con tema:").pack(pady=(20, 10))
ttk.Button(root, text="Botón TTK").pack(pady=5)
ttk.Entry(root).pack(pady=5, padx=20, fill="x")
ttk.Combobox(root, values=["Opción 1", "Opción 2"]).pack(pady=5, padx=20, fill="x")

root.mainloop()
```

3.4 Mini proyecto: Panel administrativo

```
# panel_administrativo.py
import tkinter as tk
from tkinter import ttk
from datetime import datetime

class PanelAdmin:
    def __init__(self, root):
        self.root = root
        self.root.title("Panel Administrativo")
        self.root.geometry("900x600")

        self.setup_ui()

    def setup_ui(self):
        # Header
        header = tk.Frame(self.root, bg="#1a1a2e", height=60)
        header.pack(fill="x")
        header.pack_propagate(False)

        tk.Label(header, text="PANEL ADMINISTRATIVO", font=("Arial", 18, "bold"),
                fg="white", bg="#1a1a2e").pack(side="left", padx=20, pady=15)
        tk.Label(header, text=f"Bienvenido | {datetime.now().strftime('%H:%M')}",
                font=("Arial", 10), fg="#ccc", bg="#1a1a2e").pack(side="right", padx=20, pady=

        # Sidebar + Content
        container = tk.Frame(self.root)
        container.pack(fill="both", expand=True)

        # Sidebar
        sidebar = tk.Frame(container, bg="#16213e", width=200)
        sidebar.pack(side="left", fill="y")
        sidebar.pack_propagate(False)

        tk.Label(sidebar, text="MENÚ", font=("Arial", 12, "bold"),
                fg="white", bg="#16213e").pack(pady=15)
```

```
        opciones = [
            ("📊 Dashboard", "dashboard"),
            ("👥 Usuarios", "usuarios"),
            ("📁 Reportes", "reportes"),
            ("⚙️ Configuración", "config"),
            ("🚪 Salir", "salir")
        ]

        for texto, cmd in opciones:
            btn = tk.Button(sidebar, text=texto, font=("Arial", 11),
                            bg="#16213e", fg="white", bd=0, padx=20, pady=10,
                            activebackground="#0f3460", command=lambda c=cmd: self.cambiar_vist
            btn.pack(fill="x")

        # Content
        self.content = tk.Frame(container, bg="white")
        self.content.pack(side="right", fill="both", expand=True)

        # Mostrar dashboard por defecto
        self.cambiar_vista("dashboard")

    def limpiar_content(self):
        for widget in self.content.winfo_children():
            widget.destroy()

    def cambiar_vista(self, vista):
        self.limpiar_content()

        if vista == "dashboard":
            self.mostrar_dashboard()
        elif vista == "usuarios":
            self.mostrar_usuarios()
        elif vista == "reportes":
            self.mostrar_reportes()
        elif vista == "config":
            self.mostrar_config()
        elif vista == "salir":
            self.root.quit()

    def mostrar_dashboard(self):
        tk.Label(self.content, text="Dashboard", font=("Arial", 16, "bold")).pack(pady=20)

        # Estadísticas
        frame = tk.Frame(self.content, bg="white")
        frame.pack(fill="both", expand=True, padx=20, pady=20)

        stats = [
            ("Usuarios", "1,234", "#4CAF50"),
            ("Ventas", "$45,678", "#2196F3"),
            ("Reportes", "89", "#FF9800"),
            ("Tareas", "12", "#e91e63")
        ]
```

```
        for i, (titulo, valor, color) in enumerate(stats):
            card = tk.Frame(frame, bg=color, relief="raised", bd=1)
            card.grid(row=0, column=i, padx=10, sticky="nsew", ipady=20)
            frame.grid_columnconfigure(i, weight=1)

            tk.Label(card, text=titulo, font=("Arial", 11), fg="white", bg=color).pack()
            tk.Label(card, text=valor, font=("Arial", 18, "bold"), fg="white", bg=color).pack(

    def mostrar_usuarios(self):
        tk.Label(self.content, text="Gestión de Usuarios", font=("Arial", 16, "bold")).pack(pa

        # Tabla simulada
        frame_tabla = tk.Frame(self.content)
        frame_tabla.pack(fill="both", expand=True, padx=20, pady=20)

        tree = ttk.Treeview(frame_tabla, columns=("ID", "Nombre", "Email", "Estado"), height=1
        tree.heading("#0", text="ID")
        tree.heading("ID", text="ID")
        tree.heading("#1", text="Nombre")
        tree.heading("#2", text="Email")
        tree.heading("#3", text="Estado")

        datos = [
            ("1", "Juan Pérez", "juan@email.com", "Activo"),
            ("2", "María García", "maria@email.com", "Activo"),
            ("3", "Carlos López", "carlos@email.com", "Inactivo"),
        ]

        for i, (id_, nombre, email, estado) in enumerate(datos):
            tree.insert("", tk.END, text=id_, values=(id_, nombre, email, estado))

        tree.pack(fill="both", expand=True)

    def mostrar_reportes(self):
        tk.Label(self.content, text="Reportes", font=("Arial", 16, "bold")).pack(pady=20)
        tk.Label(self.content, text="Sección de reportes en desarrollo...", font=("Arial", 12)

    def mostrar_config(self):
        tk.Label(self.content, text="Configuración", font=("Arial", 16, "bold")).pack(pady=20)

        frame = tk.Frame(self.content)
        frame.pack(padx=20, pady=20)

        tk.Label(frame, text="Tema:", font=("Arial", 11)).grid(row=0, column=0, sticky="w", pa
        ttk.Combobox(frame, values=["Claro", "Oscuro"], state="readonly").grid(row=0, column=1

        tk.Label(frame, text="Notificaciones:", font=("Arial", 11)).grid(row=1, column=0, stic
        ttk.Checkbutton(frame).grid(row=1, column=1, sticky="w")

if __name__ == "__main__":
    root = tk.Tk()
```

```
    app = PanelAdmin(root)
    root.mainloop()
```

🔹 MÓDULO 4 — Arquitectura profesional

4.1 Separación lógica / UI con clases

```
# arquitectura_profesional/
# ├── main.py
# ├── models/
# │   ├── __init__.py
# │   └── usuario.py
# ├── views/
# │   ├── __init__.py
# │   └── login_view.py
# └── controllers/
#     ├── __init__.py
#     └── login_controller.py

# models/usuario.py
class Usuario:
    def __init__(self, nombre, email, edad):
        self.nombre = nombre
        self.email = email
        self.edad = edad

    def validar(self):
        if len(self.nombre) < 3:
            return False, "Nombre muy corto"
        if "@" not in self.email:
            return False, "Email inválido"
        if self.edad < 18:
            return False, "Debes ser mayor de edad"
        return True, "OK"

# controllers/login_controller.py
class LoginController:
    def __init__(self, model):
        self.model = model

    def registrar_usuario(self, nombre, email, edad):
        usuario = self.model(nombre, email, edad)
        valido, msg = usuario.validar()
        return valido, msg, usuario if valido else None

# views/login_view.py
import tkinter as tk
from tkinter import messagebox
```

```python
class LoginView:
    def __init__(self, root, controller):
        self.root = root
        self.controller = controller
        self.root.title("Login")
        self.root.geometry("400x300")

        self.setup_ui()

    def setup_ui(self):
        tk.Label(self.root, text="Registro", font=("Arial", 16, "bold")).pack(pady=20)

        frame = tk.Frame(self.root)
        frame.pack(padx=20, pady=10, fill="x")

        tk.Label(frame, text="Nombre:").pack()
        self.nombre_var = tk.StringVar()
        tk.Entry(frame, textvariable=self.nombre_var).pack(pady=5, fill="x")

        tk.Label(frame, text="Email:").pack()
        self.email_var = tk.StringVar()
        tk.Entry(frame, textvariable=self.email_var).pack(pady=5, fill="x")

        tk.Label(frame, text="Edad:").pack()
        self.edad_var = tk.IntVar()
        tk.Entry(frame, textvariable=self.edad_var).pack(pady=5, fill="x")

        tk.Button(self.root, text="Registrar", command=self.registrar,
                  bg="#4CAF50", fg="white").pack(pady=20)

    def registrar(self):
        nombre = self.nombre_var.get()
        email = self.email_var.get()
        edad = self.edad_var.get()

        valido, msg, usuario = self.controller.registrar_usuario(nombre, email, edad)

        if valido:
            messagebox.showinfo("Éxito", f"Usuario {nombre} registrado")
        else:
            messagebox.showerror("Error", msg)

# main.py
import tkinter as tk
from models.usuario import Usuario
from controllers.login_controller import LoginController
from views.login_view import LoginView

if __name__ == "__main__":
    root = tk.Tk()
    controller = LoginController(Usuario)
```

```
    view = LoginView(root, controller)
    root.mainloop()
```

4.2 Patrón MVC simplificado

```
# mvc_app.py
import tkinter as tk
from tkinter import messagebox, ttk
import json
import os

# MODEL
class TodoModel:
    def __init__(self, filename="todos.json"):
        self.filename = filename
        self.todos = self.cargar()

    def cargar(self):
        if os.path.exists(self.filename):
            with open(self.filename, "r") as f:
                return json.load(f)
        return []

    def guardar(self):
        with open(self.filename, "w") as f:
            json.dump(self.todos, f, indent=4)

    def agregar(self, titulo, descripcion):
        self.todos.append({
            "id": len(self.todos) + 1,
            "titulo": titulo,
            "descripcion": descripcion,
            "completado": False
        })
        self.guardar()

    def eliminar(self, id_):
        self.todos = [t for t in self.todos if t["id"] != id_]
        self.guardar()

    def marcar_completado(self, id_):
        for todo in self.todos:
            if todo["id"] == id_:
                todo["completado"] = not todo["completado"]
        self.guardar()

# CONTROLLER
class TodoController:
    def __init__(self, model):
```

```
        self.model = model

    def obtener_todos(self):
        return self.model.todos

    def crear_todo(self, titulo, descripcion):
        if not titulo.strip():
            return False, "El título no puede estar vacío"
        self.model.agregar(titulo, descripcion)
        return True, "Tarea creada"

    def eliminar_todo(self, id_):
        self.model.eliminar(id_)
        return True, "Tarea eliminada"

    def completar_todo(self, id_):
        self.model.marcar_completado(id_)
        return True, "Estado actualizado"

# VIEW
class TodoView:
    def __init__(self, root, controller):
        self.root = root
        self.controller = controller
        self.root.title("Gestor de Tareas (MVC)")
        self.root.geometry("700x500")

        self.setup_ui()
        self.actualizar_lista()

    def setup_ui(self):
        # Header
        header = tk.Frame(self.root, bg="#2196F3", height=60)
        header.pack(fill="x")
        header.pack_propagate(False)

        tk.Label(header, text="📋 Gestor de Tareas", font=("Arial", 18, "bold"),
                 fg="white", bg="#2196F3").pack(pady=10)

        # Input
        frame_input = tk.Frame(self.root)
        frame_input.pack(fill="x", padx=20, pady=15)

        tk.Label(frame_input, text="Título:", font=("Arial", 11)).pack(anchor="w")
        self.titulo_var = tk.StringVar()
        tk.Entry(frame_input, textvariable=self.titulo_var, font=("Arial", 11)).pack(fill="x",

        tk.Label(frame_input, text="Descripción:", font=("Arial", 11)).pack(anchor="w")
        self.descripcion_var = tk.StringVar()
        tk.Entry(frame_input, textvariable=self.descripcion_var, font=("Arial", 11)).pack(fill

        tk.Button(frame_input, text="Agregar Tarea", command=self.agregar_tarea,
                  bg="#4CAF50", fg="white", padx=20).pack(pady=10)
```

```
        # Tabla
        frame_tabla = tk.Frame(self.root)
        frame_tabla.pack(fill="both", expand=True, padx=20, pady=15)

        self.tree = ttk.Treeview(frame_tabla, columns=("ID", "Título", "Descripción", "Estado"
        self.tree.heading("#0", text="ID")
        self.tree.column("#0", width=30)
        self.tree.heading("ID", text="ID")
        self.tree.column("ID", width=30)
        self.tree.heading("#1", text="Título")
        self.tree.column("#1", width=150)
        self.tree.heading("#2", text="Descripción")
        self.tree.column("#2", width=250)
        self.tree.heading("#3", text="Estado")
        self.tree.column("#3", width=100)

        self.tree.pack(fill="both", expand=True)
        self.tree.bind("<Double-1>", self.on_double_click)

    def agregar_tarea(self):
        titulo = self.titulo_var.get()
        descripcion = self.descripcion_var.get()

        exito, msg = self.controller.crear_todo(titulo, descripcion)

        if exito:
            self.titulo_var.set("")
            self.descripcion_var.set("")
            self.actualizar_lista()
            messagebox.showinfo("Éxito", msg)
        else:
            messagebox.showerror("Error", msg)

    def actualizar_lista(self):
        for item in self.tree.get_children():
            self.tree.delete(item)

        for todo in self.controller.obtener_todos():
            estado = "✓ Completado" if todo["completado"] else "⏳ Pendiente"
            self.tree.insert("", tk.END, text=todo["id"],
                             values=(todo["id"], todo["titulo"], todo["descripcion"], estado))

    def on_double_click(self, event):
        selection = self.tree.selection()
        if selection:
            item = self.tree.item(selection[0])
            id_ = int(item["text"])

            # Menú contextual
            self.mostrar_menu_contexto(id_)

    def mostrar_menu_contexto(self, id_):
```

```
        menu = tk.Menu(self.root, tearoff=False)
        menu.add_command(label="Marcar como completado",
                         command=lambda: self.completar(id_))
        menu.add_command(label="Eliminar",
                         command=lambda: self.eliminar(id_))
        menu.post(self.root.winfo_pointerx(), self.root.winfo_pointery())

    def completar(self, id_):
        self.controller.completar_todo(id_)
        self.actualizar_lista()

    def eliminar(self, id_):
        if messagebox.askyesno("Confirmar", "¿Eliminar tarea?"):
            self.controller.eliminar_todo(id_)
            self.actualizar_lista()

if __name__ == "__main__":
    root = tk.Tk()
    model = TodoModel()
    controller = TodoController(model)
    view = TodoView(root, controller)
    root.mainloop()
```

🔹 MÓDULO 5 — Funcionalidades reales

5.1 Manejo de archivos (JSON)

```
# manejo_archivos.py
import tkinter as tk
from tkinter import filedialog, messagebox, ttk
import json
import os

class GestorArchivos:
    def __init__(self, root):
        self.root = root
        self.root.title("Gestor de Archivos JSON")
        self.root.geometry("600x400")

        self.archivo_actual = None
        self.datos = {}

        self.setup_ui()

    def setup_ui(self):
        # Barra de herramientas
        toolbar = tk.Frame(self.root, bg="#f0f0f0", relief="raised", bd=1)
        toolbar.pack(fill="x", padx=5, pady=5)
```

```
        tk.Button(toolbar, text="📂 Abrir", command=self.abrir_archivo).pack(side="left", padx
        tk.Button(toolbar, text="💾 Guardar", command=self.guardar_archivo).pack(side="left",
        tk.Button(toolbar, text="🆕 Nuevo", command=self.nuevo_archivo).pack(side="left", padx

        tk.Label(toolbar, text="Archivo: No abierto", font=("Arial", 10)).pack(side="right", p

        # Área de contenido
        frame = tk.Frame(self.root)
        frame.pack(fill="both", expand=True, padx=10, pady=10)

        tk.Label(frame, text="Contenido JSON:", font=("Arial", 11, "bold")).pack(anchor="w")

        self.text = tk.Text(frame, font=("Arial", 10), wrap="word", relief="solid", bd=1)
        self.text.pack(fill="both", expand=True, pady=5)

        # Botones de acción
        frame_botones = tk.Frame(self.root)
        frame_botones.pack(fill="x", padx=10, pady=10)

        tk.Button(frame_botones, text="Agregar Registro", command=self.agregar_registro,
                  bg="#4CAF50", fg="white").pack(side="left", padx=5)
        tk.Button(frame_botones, text="Validar JSON", command=self.validar_json,
                  bg="#2196F3", fg="white").pack(side="left", padx=5)

    def abrir_archivo(self):
        archivo = filedialog.askopenfilename(
            filetypes=[("JSON files", "*.json"), ("All files", "*.*")]
        )
        if archivo:
            try:
                with open(archivo, "r", encoding="utf-8") as f:
                    self.datos = json.load(f)
                self.archivo_actual = archivo
                self.mostrar_contenido()
                messagebox.showinfo("Éxito", f"Archivo abierto: {os.path.basename(archivo)}")
            except Exception as e:
                messagebox.showerror("Error", f"No se pudo abrir: {e}")

    def guardar_archivo(self):
        if not self.archivo_actual:
            archivo = filedialog.asksaveasfilename(
                defaultextension=".json",
                filetypes=[("JSON files", "*.json")]
            )
            if not archivo:
                return
            self.archivo_actual = archivo

        try:
            contenido = self.text.get("1.0", tk.END).strip()
            self.datos = json.loads(contenido)
```

```
            with open(self.archivo_actual, "w", encoding="utf-8") as f:
                json.dump(self.datos, f, indent=4, ensure_ascii=False)

            messagebox.showinfo("Éxito", "Archivo guardado")
        except json.JSONDecodeError as e:
            messagebox.showerror("Error", f"JSON inválido: {e}")
        except Exception as e:
            messagebox.showerror("Error", f"No se pudo guardar: {e}")

    def nuevo_archivo(self):
        self.datos = {}
        self.archivo_actual = None
        self.text.delete("1.0", tk.END)
        messagebox.showinfo("Nuevo", "Archivo nuevo creado")

    def mostrar_contenido(self):
        self.text.delete("1.0", tk.END)
        self.text.insert(tk.END, json.dumps(self.datos, indent=4, ensure_ascii=False))

    def agregar_registro(self):
        ventana = tk.Toplevel(self.root)
        ventana.title("Agregar Registro")
        ventana.geometry("300x150")

        tk.Label(ventana, text="Clave:", font=("Arial", 11)).pack()
        clave_var = tk.StringVar()
        tk.Entry(ventana, textvariable=clave_var).pack(pady=5, padx=10, fill="x")

        tk.Label(ventana, text="Valor:", font=("Arial", 11)).pack()
        valor_var = tk.StringVar()
        tk.Entry(ventana, textvariable=valor_var).pack(pady=5, padx=10, fill="x")

        def guardar_registro():
            clave = clave_var.get()
            valor = valor_var.get()

            if not clave or not valor:
                messagebox.showerror("Error", "Completa todos los campos")
                return

            self.datos[clave] = valor
            self.mostrar_contenido()
            ventana.destroy()
            messagebox.showinfo("Éxito", "Registro agregado")

        tk.Button(ventana, text="Guardar", command=guardar_registro,
                  bg="#4CAF50", fg="white").pack(pady=10)

    def validar_json(self):
        try:
            contenido = self.text.get("1.0", tk.END).strip()
            json.loads(contenido)
            messagebox.showinfo("Válido", "JSON correcto ✓")
```

```
        except json.JSONDecodeError as e:
            messagebox.showerror("Inválido", f"Error: {e}")

if __name__ == "__main__":
    root = tk.Tk()
    app = GestorArchivos(root)
    root.mainloop()
```

5.2 Manejo de base de datos SQLite

```
# gestion_sqlite.py
import tkinter as tk
from tkinter import messagebox, ttk
import sqlite3
import os

class ContactoApp:
    def __init__(self, root):
        self.root = root
        self.root.title("Gestor de Contactos - SQLite")
        self.root.geometry("700x500")

        self.db_file = "contactos.db"
        self.init_db()
        self.setup_ui()
        self.cargar_contactos()

    def init_db(self):
        conn = sqlite3.connect(self.db_file)
        cursor = conn.cursor()

        cursor.execute("""
        CREATE TABLE IF NOT EXISTS contactos (
            id INTEGER PRIMARY KEY,
            nombre TEXT NOT NULL,
            email TEXT NOT NULL,
            telefono TEXT NOT NULL,
            fecha_creacion TIMESTAMP DEFAULT CURRENT_TIMESTAMP
        )
        """)

        conn.commit()
        conn.close()

    def setup_ui(self):
        # Frame superior
        frame_input = tk.LabelFrame(self.root, text="Agregar/Editar Contacto", padx=10, pady=1
        frame_input.pack(fill="x", padx=10, pady=10)
```

```
        tk.Label(frame_input, text="Nombre:").grid(row=0, column=0, sticky="w", pady=5)
        self.nombre_var = tk.StringVar()
        tk.Entry(frame_input, textvariable=self.nombre_var, width=30).grid(row=0, column=1, pa

        tk.Label(frame_input, text="Email:").grid(row=1, column=0, sticky="w", pady=5)
        self.email_var = tk.StringVar()
        tk.Entry(frame_input, textvariable=self.email_var, width=30).grid(row=1, column=1, pad

        tk.Label(frame_input, text="Teléfono:").grid(row=2, column=0, sticky="w", pady=5)
        self.telefono_var = tk.StringVar()
        tk.Entry(frame_input, textvariable=self.telefono_var, width=30).grid(row=2, column=1,

        frame_botones = tk.Frame(frame_input)
        frame_botones.grid(row=3, column=0, columnspan=2, pady=10)

        tk.Button(frame_botones, text="Agregar", command=self.agregar_contacto,
                  bg="#4CAF50", fg="white").pack(side="left", padx=5)
        tk.Button(frame_botones, text="Actualizar", command=self.actualizar_contacto,
                  bg="#2196F3", fg="white").pack(side="left", padx=5)
        tk.Button(frame_botones, text="Limpiar", command=self.limpiar_campos).pack(side="left"

        # Tabla
        frame_tabla = tk.LabelFrame(self.root, text="Contactos", padx=10, pady=10)
        frame_tabla.pack(fill="both", expand=True, padx=10, pady=10)

        self.tree = ttk.Treeview(frame_tabla, columns=("ID", "Nombre", "Email", "Teléfono"), h
        self.tree.heading("#0", text="ID")
        self.tree.column("#0", width=30)
        self.tree.heading("ID", text="ID")
        self.tree.column("ID", width=30)
        self.tree.heading("#1", text="Nombre")
        self.tree.column("#1", width=150)
        self.tree.heading("#2", text="Email")
        self.tree.column("#2", width=200)
        self.tree.heading("#3", text="Teléfono")
        self.tree.column("#3", width=120)

        self.tree.pack(fill="both", expand=True)
        self.tree.bind("<ButtonRelease-1>", self.on_select)

        # Frame botones inferiores
        frame_inf = tk.Frame(self.root)
        frame_inf.pack(fill="x", padx=10, pady=10)

        tk.Button(frame_inf, text="Eliminar Seleccionado", command=self.eliminar_contacto,
                  bg="#ff5252", fg="white").pack(side="left", padx=5)
        tk.Button(frame_inf, text="Actualizar Lista", command=self.cargar_contactos,
                  bg="#9C27B0", fg="white").pack(side="left", padx=5)

    def agregar_contacto(self):
        nombre = self.nombre_var.get().strip()
        email = self.email_var.get().strip()
        telefono = self.telefono_var.get().strip()
```

```
        if not all([nombre, email, telefono]):
            messagebox.showerror("Error", "Completa todos los campos")
            return

        try:
            conn = sqlite3.connect(self.db_file)
            cursor = conn.cursor()
            cursor.execute(
                "INSERT INTO contactos (nombre, email, telefono) VALUES (?, ?, ?)",
                (nombre, email, telefono)
            )
            conn.commit()
            conn.close()

            self.limpiar_campos()
            self.cargar_contactos()
            messagebox.showinfo("Éxito", "Contacto agregado")
        except Exception as e:
            messagebox.showerror("Error", f"Error al agregar: {e}")

    def cargar_contactos(self):
        for item in self.tree.get_children():
            self.tree.delete(item)

        try:
            conn = sqlite3.connect(self.db_file)
            cursor = conn.cursor()
            cursor.execute("SELECT id, nombre, email, telefono FROM contactos ORDER BY nombre"

            for fila in cursor.fetchall():
                self.tree.insert("", tk.END, text=fila[0],
                                 values=(fila[0], fila[1], fila[2], fila[3]))

            conn.close()
        except Exception as e:
            messagebox.showerror("Error", f"Error al cargar: {e}")

    def on_select(self, event):
        selection = self.tree.selection()
        if selection:
            item = self.tree.item(selection[0])
            valores = item["values"]

            self.nombre_var.set(valores[1])
            self.email_var.set(valores[2])
            self.telefono_var.set(valores[3])
            self.id_seleccionado = valores[0]

    def actualizar_contacto(self):
        if not hasattr(self, "id_seleccionado"):
            messagebox.showerror("Error", "Selecciona un contacto")
            return
```

```
        nombre = self.nombre_var.get().strip()
        email = self.email_var.get().strip()
        telefono = self.telefono_var.get().strip()

        if not all([nombre, email, telefono]):
            messagebox.showerror("Error", "Completa todos los campos")
            return

        try:
            conn = sqlite3.connect(self.db_file)
            cursor = conn.cursor()
            cursor.execute(
                "UPDATE contactos SET nombre=?, email=?, telefono=? WHERE id=?",
                (nombre, email, telefono, self.id_seleccionado)
            )
            conn.commit()
            conn.close()

            self.limpiar_campos()
            self.cargar_contactos()
            messagebox.showinfo("Éxito", "Contacto actualizado")
        except Exception as e:
            messagebox.showerror("Error", f"Error al actualizar: {e}")

    def eliminar_contacto(self):
        if not hasattr(self, "id_seleccionado"):
            messagebox.showerror("Error", "Selecciona un contacto")
            return

        if messagebox.askyesno("Confirmar", "¿Eliminar contacto?"):
            try:
                conn = sqlite3.connect(self.db_file)
                cursor = conn.cursor()
                cursor.execute("DELETE FROM contactos WHERE id=?", (self.id_seleccionado,))
                conn.commit()
                conn.close()

                self.limpiar_campos()
                self.cargar_contactos()
                messagebox.showinfo("Éxito", "Contacto eliminado")
            except Exception as e:
                messagebox.showerror("Error", f"Error al eliminar: {e}")

    def limpiar_campos(self):
        self.nombre_var.set("")
        self.email_var.set("")
        self.telefono_var.set("")
        if hasattr(self, "id_seleccionado"):
            delattr(self, "id_seleccionado")

if __name__ == "__main__":
    root = tk.Tk()
```

```
    app = ContactoApp(root)
    root.mainloop()
```

5.3 Messagebox y FileDialog

```
# dialogs_avanzados.py
import tkinter as tk
from tkinter import messagebox, filedialog, colorchooser
import os

class DialogosDemo:
    def __init__(self, root):
        self.root = root
        self.root.title("Diálogos Avanzados")
        self.root.geometry("500x400")

        self.setup_ui()

    def setup_ui(self):
        frame = tk.Frame(self.root)
        frame.pack(fill="both", expand=True, padx=20, pady=20)

        # MessageBox
        tk.Label(frame, text="MessageBox (Mensajes):", font=("Arial", 12, "bold")).pack(anchor

        tk.Button(frame, text="Información", command=self.show_info).pack(fill="x", pady=3)
        tk.Button(frame, text="Advertencia", command=self.show_warning).pack(fill="x", pady=3)
        tk.Button(frame, text="Error", command=self.show_error).pack(fill="x", pady=3)
        tk.Button(frame, text="Pregunta (Si/No)", command=self.show_question).pack(fill="x", p
        tk.Button(frame, text="Pregunta (Sí/No/Cancelar)", command=self.show_yesnocancel).pack

        # FileDialog
        tk.Label(frame, text="FileDialog (Archivos):", font=("Arial", 12, "bold")).pack(anchor

        tk.Button(frame, text="Abrir Archivo", command=self.open_file).pack(fill="x", pady=3)
        tk.Button(frame, text="Abrir Múltiples", command=self.open_files).pack(fill="x", pady=
        tk.Button(frame, text="Guardar Como", command=self.save_file).pack(fill="x", pady=3)
        tk.Button(frame, text="Elegir Directorio", command=self.choose_directory).pack(fill="x
        tk.Button(frame, text="Elegir Color", command=self.choose_color).pack(fill="x", pady=3

    def show_info(self):
        messagebox.showinfo("Información", "Este es un diálogo de información")

    def show_warning(self):
        messagebox.showwarning("Advertencia", "Esto es una advertencia")

    def show_error(self):
        messagebox.showerror("Error", "Ocurrió un error")
```

```
    def show_question(self):
        respuesta = messagebox.askyesno("Pregunta", "¿Deseas continuar?")
        print(f"Respuesta: {respuesta}")

    def show_yesnocancel(self):
        respuesta = messagebox.askyesnocancel("Pregunta", "¿Guardar cambios?")
        print(f"Respuesta: {respuesta}")  # True=Sí, False=No, None=Cancelar

    def open_file(self):
        archivo = filedialog.askopenfilename(
            title="Abrir archivo",
            filetypes=[("Texto", "*.txt"), ("Python", "*.py"), ("Todos", "*.*")]
        )
        if archivo:
            print(f"Archivo seleccionado: {archivo}")

    def open_files(self):
        archivos = filedialog.askopenfilenames(
            title="Abrir múltiples archivos",
            filetypes=[("Texto", "*.txt"), ("Todos", "*.*")]
        )
        if archivos:
            for archivo in archivos:
                print(f"- {archivo}")

    def save_file(self):
        archivo = filedialog.asksaveasfilename(
            title="Guardar como",
            defaultextension=".txt",
            filetypes=[("Texto", "*.txt"), ("Python", "*.py")]
        )
        if archivo:
            print(f"Guardar en: {archivo}")

    def choose_directory(self):
        directorio = filedialog.askdirectory(title="Elegir carpeta")
        if directorio:
            print(f"Carpeta seleccionada: {directorio}")

    def choose_color(self):
        color = colorchooser.askcolor(title="Elegir color")
        if color[1]:
            print(f"Color seleccionado: {color[1]}")

if __name__ == "__main__":
    root = tk.Tk()
    app = DialogosDemo(root)
    root.mainloop()
```

5.4 Ventanas secundarias (Toplevel)

```python
# ventanas_secundarias.py
import tkinter as tk
from tkinter import messagebox

class MainApp:
    def __init__(self, root):
        self.root = root
        self.root.title("Ventana Principal")
        self.root.geometry("400x300")

        self.ventanas_abiertas = {}

        self.setup_ui()

    def setup_ui(self):
        tk.Label(self.root, text="Ventana Principal", font=("Arial", 14, "bold")).pack(pady=20

        frame = tk.Frame(self.root)
        frame.pack(fill="x", padx=20, pady=10)

        tk.Button(frame, text="Abrir Ventana Simple", command=self.abrir_simple).pack(fill="x"
        tk.Button(frame, text="Abrir Ventana Modal", command=self.abrir_modal).pack(fill="x",
        tk.Button(frame, text="Abrir Múltiple Única", command=self.abrir_unica).pack(fill="x",
        tk.Button(frame, text="Abrir con Parámetros", command=self.abrir_parametros).pack(fill

    def abrir_simple(self):
        ventana = tk.Toplevel(self.root)
        ventana.title("Ventana Simple")
        ventana.geometry("300x200")

        tk.Label(ventana, text="Ventana secundaria", font=("Arial", 12)).pack(pady=20)
        tk.Button(ventana, text="Cerrar", command=ventana.destroy).pack()

    def abrir_modal(self):
        ventana = tk.Toplevel(self.root)
        ventana.title("Ventana Modal")
        ventana.geometry("300x200")
        ventana.resizable(False, False)

        # Hacer modal (bloquea interacción con ventana padre)
        ventana.grab_set()

        tk.Label(ventana, text="Ventana MODAL", font=("Arial", 12, "bold")).pack(pady=20)
        tk.Label(ventana, text="(No puedes interactuar con la ventana principal)").pack()
        tk.Button(ventana, text="Aceptar", command=ventana.destroy, bg="#4CAF50", fg="white").

    def abrir_unica(self):
        # Solo permite una ventana "info" abierta
        if "info" in self.ventanas_abiertas and self.ventanas_abiertas["info"].winfo_exists():
            self.ventanas_abiertas["info"].lift()
            return
```

```
        ventana = tk.Toplevel(self.root)
        ventana.title("Ventana Única")
        ventana.geometry("300x200")

        def on_close():
            del self.ventanas_abiertas["info"]
            ventana.destroy()

        ventana.protocol("WM_DELETE_WINDOW", on_close)
        self.ventanas_abiertas["info"] = ventana

        tk.Label(ventana, text="Solo puede haber una de estas", font=("Arial", 11)).pack(pady=
        tk.Button(ventana, text="Cerrar", command=on_close).pack()

    def abrir_parametros(self):
        ventana = tk.Toplevel(self.root)
        ventana.title("Ventana con Datos")
        ventana.geometry("300x250")

        tk.Label(ventana, text="Ingresa datos:", font=("Arial", 12, "bold")).pack(pady=10)

        tk.Label(ventana, text="Nombre:").pack(anchor="w", padx=20, pady=(10, 0))
        entrada_nombre = tk.Entry(ventana)
        entrada_nombre.pack(padx=20, pady=5, fill="x")

        tk.Label(ventana, text="Email:").pack(anchor="w", padx=20, pady=(10, 0))
        entrada_email = tk.Entry(ventana)
        entrada_email.pack(padx=20, pady=5, fill="x")

        def guardar():
            nombre = entrada_nombre.get()
            email = entrada_email.get()

            if nombre and email:
                # Devolver datos a ventana principal
                messagebox.showinfo("Datos", f"Nombre: {nombre}\nEmail: {email}")
                ventana.destroy()
            else:
                messagebox.showerror("Error", "Completa todos los campos")

        tk.Button(ventana, text="Guardar", command=guardar, bg="#4CAF50", fg="white").pack(pad
        tk.Button(ventana, text="Cancelar", command=ventana.destroy).pack()

if __name__ == "__main__":
    root = tk.Tk()
    app = MainApp(root)
    root.mainloop()
```

5.5 Menús

```python
# menus_completo.py
import tkinter as tk
from tkinter import messagebox

class AppConMenus:
    def __init__(self, root):
        self.root = root
        self.root.title("Aplicación con Menús")
        self.root.geometry("600x400")

        self.crear_menus()
        self.setup_ui()

    def crear_menus(self):
        menubar = tk.Menu(self.root)
        self.root.config(menu=menubar)

        # Menú Archivo
        menu_archivo = tk.Menu(menubar, tearoff=0)
        menubar.add_cascade(label="Archivo", menu=menu_archivo)

        menu_archivo.add_command(label="Nuevo", command=self.nuevo, accelerator="Ctrl+N")
        menu_archivo.add_command(label="Abrir", command=self.abrir, accelerator="Ctrl+O")
        menu_archivo.add_command(label="Guardar", command=self.guardar, accelerator="Ctrl+S")
        menu_archivo.add_separator()
        menu_archivo.add_command(label="Salir", command=self.root.quit, accelerator="Ctrl+Q")

        # Menú Edición
        menu_edicion = tk.Menu(menubar, tearoff=0)
        menubar.add_cascade(label="Edición", menu=menu_edicion)

        menu_edicion.add_command(label="Cortar", accelerator="Ctrl+X")
        menu_edicion.add_command(label="Copiar", accelerator="Ctrl+C")
        menu_edicion.add_command(label="Pegar", accelerator="Ctrl+V")
        menu_edicion.add_separator()
        menu_edicion.add_command(label="Preferencias", command=self.preferencias)

        # Menú Ver
        menu_ver = tk.Menu(menubar, tearoff=0)
        menubar.add_cascade(label="Ver", menu=menu_ver)

        self.fullscreen_var = tk.BooleanVar()
        menu_ver.add_checkbutton(label="Pantalla Completa", variable=self.fullscreen_var,
                                 command=self.toggle_fullscreen)

        # Menú Ayuda
        menu_ayuda = tk.Menu(menubar, tearoff=0)
        menubar.add_cascade(label="Ayuda", menu=menu_ayuda)

        menu_ayuda.add_command(label="Documentación", command=self.documentacion)
        menu_ayuda.add_separator()
        menu_ayuda.add_command(label="Acerca de", command=self.acerca_de)
```

```
        # Bind de atajos de teclado
        self.root.bind("<Control-n>", lambda e: self.nuevo())
        self.root.bind("<Control-o>", lambda e: self.abrir())
        self.root.bind("<Control-s>", lambda e: self.guardar())
        self.root.bind("<Control-q>", lambda e: self.root.quit())

    def setup_ui(self):
        tk.Label(self.root, text="Aplicación con Menús", font=("Arial", 16, "bold")).pack(pady
        tk.Label(self.root, text="Usa los menús arriba para ver las opciones", font=("Arial",

    def nuevo(self):
        messagebox.showinfo("Nuevo", "Crear nuevo documento")

    def abrir(self):
        messagebox.showinfo("Abrir", "Abrir documento")

    def guardar(self):
        messagebox.showinfo("Guardar", "Guardar documento")

    def preferencias(self):
        messagebox.showinfo("Preferencias", "Abrir preferencias")

    def toggle_fullscreen(self):
        estado = self.fullscreen_var.get()
        self.root.attributes('-zoomed', estado)

    def documentacion(self):
        messagebox.showinfo("Documentación", "Abrir documentación")

    def acerca_de(self):
        messagebox.showinfo("Acerca de", "Mi Aplicación v1.0\n© 2026")

if __name__ == "__main__":
    root = tk.Tk()
    app = AppConMenus(root)
    root.mainloop()
```

5.6 Mini proyecto: CRUD completo

```
# crud_completo.py
import tkinter as tk
from tkinter import messagebox, ttk, filedialog
import sqlite3
import json
from datetime import datetime

class CRUDApp:
    def __init__(self, root):
```

```
        self.root = root
        self.root.title("Sistema CRUD Completo")
        self.root.geometry("1000x700")

        self.db_file = "datos.db"
        self.init_db()
        self.id_seleccionado = None

        self.crear_menus()
        self.setup_ui()
        self.cargar_datos()

    def init_db(self):
        conn = sqlite3.connect(self.db_file)
        cursor = conn.cursor()

        cursor.execute("""
        CREATE TABLE IF NOT EXISTS productos (
            id INTEGER PRIMARY KEY,
            nombre TEXT NOT NULL,
            descripcion TEXT,
            precio REAL NOT NULL,
            cantidad INTEGER NOT NULL,
            categoria TEXT,
            fecha_creacion TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
            fecha_actualizacion TIMESTAMP DEFAULT CURRENT_TIMESTAMP
        )
        """)

        conn.commit()
        conn.close()

    def crear_menus(self):
        menubar = tk.Menu(self.root)
        self.root.config(menu=menubar)

        # Menú Archivo
        archivo = tk.Menu(menubar, tearoff=0)
        menubar.add_cascade(label="Archivo", menu=archivo)
        archivo.add_command(label="Exportar a JSON", command=self.exportar_json)
        archivo.add_command(label="Importar JSON", command=self.importar_json)
        archivo.add_separator()
        archivo.add_command(label="Salir", command=self.root.quit)

        # Menú Edición
        edicion = tk.Menu(menubar, tearoff=0)
        menubar.add_cascade(label="Edición", menu=edicion)
        edicion.add_command(label="Limpiar formulario", command=self.limpiar_formulario)

        # Menú Ayuda
        ayuda = tk.Menu(menubar, tearoff=0)
        menubar.add_cascade(label="Ayuda", menu=ayuda)
        ayuda.add_command(label="Acerca de", command=self.acerca_de)
```

```python
def setup_ui(self):
    # Header
    header = tk.Frame(self.root, bg="#2196F3", height=60)
    header.pack(fill="x")
    header.pack_propagate(False)

    tk.Label(header, text="📦 Sistema CRUD de Productos", font=("Arial", 16, "bold"),
            fg="white", bg="#2196F3").pack(side="left", padx=20, pady=15)

    # Contenedor principal
    container = tk.Frame(self.root)
    container.pack(fill="both", expand=True)

    # Lado izquierdo - Formulario
    left = tk.LabelFrame(container, text="Formulario", padx=10, pady=10)
    left.pack(side="left", fill="both", padx=10, pady=10, ipady=10)

    tk.Label(left, text="Nombre:", font=("Arial", 10)).pack(anchor="w", pady=(5, 0))
    self.nombre_var = tk.StringVar()
    tk.Entry(left, textvariable=self.nombre_var, width=30).pack(pady=5)

    tk.Label(left, text="Descripción:", font=("Arial", 10)).pack(anchor="w", pady=(5, 0))
    self.desc_var = tk.StringVar()
    tk.Entry(left, textvariable=self.desc_var, width=30).pack(pady=5)

    tk.Label(left, text="Precio:", font=("Arial", 10)).pack(anchor="w", pady=(5, 0))
    self.precio_var = tk.StringVar()
    tk.Entry(left, textvariable=self.precio_var, width=30).pack(pady=5)

    tk.Label(left, text="Cantidad:", font=("Arial", 10)).pack(anchor="w", pady=(5, 0))
    self.cantidad_var = tk.StringVar()
    tk.Entry(left, textvariable=self.cantidad_var, width=30).pack(pady=5)

    tk.Label(left, text="Categoría:", font=("Arial", 10)).pack(anchor="w", pady=(5, 0))
    self.categoria_var = tk.StringVar(value="Selecciona")
    categorias = ["Selecciona", "Electrónica", "Ropa", "Alimentos", "Otros"]
    tk.OptionMenu(left, self.categoria_var, *categorias).pack(fill="x", pady=5)

    # Botones
    frame_botones = tk.Frame(left)
    frame_botones.pack(fill="x", pady=20)

    tk.Button(frame_botones, text="Agregar", command=self.agregar,
             bg="#4CAF50", fg="white", width=12).pack(pady=5)
    tk.Button(frame_botones, text="Actualizar", command=self.actualizar,
             bg="#2196F3", fg="white", width=12).pack(pady=5)
    tk.Button(frame_botones, text="Eliminar", command=self.eliminar,
             bg="#ff5252", fg="white", width=12).pack(pady=5)
    tk.Button(frame_botones, text="Limpiar", command=self.limpiar_formulario, width=12).pa

    # Lado derecho - Tabla
    right = tk.LabelFrame(container, text="Productos", padx=10, pady=10)
```

```
        right.pack(side="right", fill="both", expand=True, padx=10, pady=10)

        # Barra búsqueda
        frame_buscar = tk.Frame(right)
        frame_buscar.pack(fill="x", pady=10)

        tk.Label(frame_buscar, text="Buscar:", font=("Arial", 10)).pack(side="left", padx=5)
        self.buscar_var = tk.StringVar()
        self.buscar_var.trace("w", lambda *args: self.filtrar_tabla())
        tk.Entry(frame_buscar, textvariable=self.buscar_var, width=30).pack(side="left", padx=

        # Tabla
        self.tree = ttk.Treeview(right, columns=("ID", "Nombre", "Descripción", "Precio", "Can
        self.tree.heading("#0", text="ID")
        self.tree.column("#0", width=40)
        for col in ("ID", "Nombre", "Descripción", "Precio", "Cantidad", "Categoría"):
            self.tree.heading(col, text=col)
            self.tree.column(col, width=120)

        self.tree.pack(fill="both", expand=True)
        self.tree.bind("<ButtonRelease-1>", self.on_select)

    def agregar(self):
        nombre = self.nombre_var.get().strip()
        descripcion = self.desc_var.get().strip()
        precio = self.precio_var.get().strip()
        cantidad = self.cantidad_var.get().strip()
        categoria = self.categoria_var.get()

        if not all([nombre, precio, cantidad]) or categoria == "Selecciona":
            messagebox.showerror("Error", "Completa todos los campos")
            return

        try:
            precio_float = float(precio)
            cantidad_int = int(cantidad)

            conn = sqlite3.connect(self.db_file)
            cursor = conn.cursor()
            cursor.execute(
                """INSERT INTO productos (nombre, descripcion, precio, cantidad, categoria)
                VALUES (?, ?, ?, ?, ?)""",
                (nombre, descripcion, precio_float, cantidad_int, categoria)
            )
            conn.commit()
            conn.close()

            self.limpiar_formulario()
            self.cargar_datos()
            messagebox.showinfo("Éxito", "Producto agregado")
        except ValueError:
            messagebox.showerror("Error", "Precio y cantidad deben ser números")
        except Exception as e:
```

```
            messagebox.showerror("Error", f"Error: {e}")

    def actualizar(self):
        if self.id_seleccionado is None:
            messagebox.showerror("Error", "Selecciona un producto")
            return

        nombre = self.nombre_var.get().strip()
        descripcion = self.desc_var.get().strip()
        precio = self.precio_var.get().strip()
        cantidad = self.cantidad_var.get().strip()
        categoria = self.categoria_var.get()

        if not all([nombre, precio, cantidad]) or categoria == "Selecciona":
            messagebox.showerror("Error", "Completa todos los campos")
            return

        try:
            precio_float = float(precio)
            cantidad_int = int(cantidad)

            conn = sqlite3.connect(self.db_file)
            cursor = conn.cursor()
            cursor.execute(
                """UPDATE productos SET nombre=?, descripcion=?, precio=?, cantidad=?, categor
                fecha_actualizacion=CURRENT_TIMESTAMP WHERE id=?""",
                (nombre, descripcion, precio_float, cantidad_int, categoria, self.id_seleccion
            )
            conn.commit()
            conn.close()

            self.limpiar_formulario()
            self.cargar_datos()
            messagebox.showinfo("Éxito", "Producto actualizado")
        except ValueError:
            messagebox.showerror("Error", "Precio y cantidad deben ser números")
        except Exception as e:
            messagebox.showerror("Error", f"Error: {e}")

    def eliminar(self):
        if self.id_seleccionado is None:
            messagebox.showerror("Error", "Selecciona un producto")
            return

        if messagebox.askyesno("Confirmar", "¿Eliminar producto?"):
            try:
                conn = sqlite3.connect(self.db_file)
                cursor = conn.cursor()
                cursor.execute("DELETE FROM productos WHERE id=?", (self.id_seleccionado,))
                conn.commit()
                conn.close()

                self.limpiar_formulario()
```

```
            self.cargar_datos()
            messagebox.showinfo("Éxito", "Producto eliminado")
        except Exception as e:
            messagebox.showerror("Error", f"Error: {e}")

def cargar_datos(self):
    for item in self.tree.get_children():
        self.tree.delete(item)

    try:
        conn = sqlite3.connect(self.db_file)
        cursor = conn.cursor()
        cursor.execute("SELECT id, nombre, descripcion, precio, cantidad, categoria FROM p

        for fila in cursor.fetchall():
            self.tree.insert("", tk.END, text=fila[0],
                             values=(fila[0], fila[1], fila[2], f"${fila[3]:.2f}", fila[4],

        conn.close()
    except Exception as e:
        messagebox.showerror("Error", f"Error al cargar: {e}")

def filtrar_tabla(self):
    busqueda = self.buscar_var.get().lower()

    for item in self.tree.get_children():
        valores = self.tree.item(item)["values"]
        nombre = str(valores[1]).lower()

        if busqueda in nombre:
            self.tree.item(item, tags=("",))
        else:
            self.tree.item(item, tags=("oculto",))

    self.tree.tag_configure("oculto", foreground="#ccc")

def on_select(self, event):
    selection = self.tree.selection()
    if selection:
        item = self.tree.item(selection[0])
        valores = item["values"]

        self.id_seleccionado = valores[0]
        self.nombre_var.set(valores[1])
        self.desc_var.set(valores[2])
        precio_str = str(valores[3]).replace("$", "")
        self.precio_var.set(precio_str)
        self.cantidad_var.set(valores[4])
        self.categoria_var.set(valores[5])

def limpiar_formulario(self):
    self.nombre_var.set("")
    self.desc_var.set("")
```

```
        self.precio_var.set("")
        self.cantidad_var.set("")
        self.categoria_var.set("Selecciona")
        self.id_seleccionado = None

    def exportar_json(self):
        try:
            conn = sqlite3.connect(self.db_file)
            cursor = conn.cursor()
            cursor.execute("SELECT id, nombre, descripcion, precio, cantidad, categoria, fecha

            productos = []
            for fila in cursor.fetchall():
                productos.append({
                    "id": fila[0],
                    "nombre": fila[1],
                    "descripcion": fila[2],
                    "precio": fila[3],
                    "cantidad": fila[4],
                    "categoria": fila[5],
                    "fecha_creacion": fila[6]
                })

            archivo = filedialog.asksaveasfilename(
                defaultextension=".json",
                filetypes=[("JSON", "*.json")]
            )

            if archivo:
                with open(archivo, "w", encoding="utf-8") as f:
                    json.dump(productos, f, indent=4, ensure_ascii=False)
                messagebox.showinfo("Éxito", f"Datos exportados a {archivo}")

            conn.close()
        except Exception as e:
            messagebox.showerror("Error", f"Error al exportar: {e}")

    def importar_json(self):
        archivo = filedialog.askopenfilename(
            filetypes=[("JSON", "*.json")]
        )

        if archivo:
            try:
                with open(archivo, "r", encoding="utf-8") as f:
                    productos = json.load(f)

                conn = sqlite3.connect(self.db_file)
                cursor = conn.cursor()

                for prod in productos:
                    cursor.execute(
                        """INSERT INTO productos (nombre, descripcion, precio, cantidad, categ
```

```
                        VALUES (?, ?, ?, ?, ?)""",
                        (prod["nombre"], prod["descripcion"], prod["precio"], prod["cantidad"]
                    )

                conn.commit()
                conn.close()

                self.cargar_datos()
                messagebox.showinfo("Éxito", f"Se importaron {len(productos)} productos")
            except Exception as e:
                messagebox.showerror("Error", f"Error al importar: {e}")

    def acerca_de(self):
        messagebox.showinfo("Acerca de", "Sistema CRUD v1.0\n© 2026\nTodos los derechos reserv

if __name__ == "__main__":
    root = tk.Tk()
    app = CRUDApp(root)
    root.mainloop()
```

🔷 MÓDULO 6 — Nivel avanzado

6.1 Threading (manejo de tareas largas)

```
# threading_avanzado.py
import tkinter as tk
from tkinter import messagebox, ttk
import threading
import time

class AppConThreads:
    def __init__(self, root):
        self.root = root
        self.root.title("Aplicación Multitarea")
        self.root.geometry("600x400")

        self.thread_activo = False
        self.setup_ui()

    def setup_ui(self):
        frame = tk.Frame(self.root)
        frame.pack(fill="both", expand=True, padx=20, pady=20)

        tk.Label(frame, text="Operaciones Largas con Threading", font=("Arial", 14, "bold")).p

        # Progreso
        tk.Label(frame, text="Progreso:").pack(anchor="w")
        self.progress = ttk.Progressbar(frame, length=400, mode="determinate")
```

```
        self.progress.pack(fill="x", pady=10)

        self.status_label = tk.Label(frame, text="Estado: Listo", font=("Arial", 11))
        self.status_label.pack(pady=10)

        # Botones
        frame_botones = tk.Frame(frame)
        frame_botones.pack(fill="x", pady=20)

        tk.Button(frame_botones, text="Tarea 1 (10 seg)", command=self.tarea_1,
                  bg="#4CAF50", fg="white").pack(side="left", padx=5)
        tk.Button(frame_botones, text="Tarea 2 (Descarga)", command=self.tarea_2,
                  bg="#2196F3", fg="white").pack(side="left", padx=5)
        tk.Button(frame_botones, text="Detener", command=self.detener,
                  bg="#ff5252", fg="white").pack(side="left", padx=5)

        # Salida
        tk.Label(frame, text="Log:").pack(anchor="w", pady=(20, 0))
        self.log_text = tk.Text(frame, height=8, width=60, font=("Courier", 9))
        self.log_text.pack(fill="both", expand=True, pady=5)

    def log(self, mensaje):
        self.log_text.insert(tk.END, f"{mensaje}\n")
        self.log_text.see(tk.END)
        self.root.update()

    def tarea_1(self):
        if self.thread_activo:
            messagebox.showwarning("Advertencia", "Ya hay una tarea en curso")
            return

        thread = threading.Thread(target=self._tarea_1_worker, daemon=True)
        thread.start()

    def _tarea_1_worker(self):
        self.thread_activo = True
        self.status_label.config(text="Estado: Procesando...")

        try:
            for i in range(11):
                if not self.thread_activo:
                    break

                self.progress["value"] = i * 10
                self.log(f"Tarea 1: Paso {i+1}/10")
                time.sleep(1)

            self.log("✓ Tarea 1 completada")
            self.status_label.config(text="Estado: Listo")
            messagebox.showinfo("Éxito", "Tarea 1 completada")

        except Exception as e:
            self.log(f"✗ Error: {e}")
```

```
        finally:
            self.thread_activo = False
            self.progress["value"] = 0

    def tarea_2(self):
        if self.thread_activo:
            messagebox.showwarning("Advertencia", "Ya hay una tarea en curso")
            return

        thread = threading.Thread(target=self._tarea_2_worker, daemon=True)
        thread.start()

    def _tarea_2_worker(self):
        self.thread_activo = True
        self.status_label.config(text="Estado: Descargando...")

        try:
            archivos = ["archivo1.zip", "archivo2.zip", "archivo3.zip"]

            for idx, archivo in enumerate(archivos):
                if not self.thread_activo:
                    break

                for i in range(11):
                    if not self.thread_activo:
                        break

                    progreso_total = (idx * 33) + (i * 3)
                    self.progress["value"] = min(progreso_total, 100)
                    self.log(f"Descargando {archivo}: {progreso_total}%")
                    time.sleep(0.3)

            self.log("✓ Todas las descargas completadas")
            self.status_label.config(text="Estado: Listo")
            messagebox.showinfo("Éxito", "Descargas completadas")

        except Exception as e:
            self.log(f"✗ Error: {e}")
        finally:
            self.thread_activo = False
            self.progress["value"] = 0

    def detener(self):
        self.thread_activo = False
        self.status_label.config(text="Estado: Cancelado")
        self.log("⚠ Operación cancelada")

if __name__ == "__main__":
    root = tk.Tk()
    app = AppConThreads(root)
    root.mainloop()
```

6.2 After() y tareas programadas

```
# after_y_timers.py
import tkinter as tk
from datetime import datetime

class AppConTimers:
    def __init__(self, root):
        self.root = root
        self.root.title("Timers y After()")
        self.root.geometry("500x400")

        self.contador = 0
        self.reloj_activo = False
        self.timer_activo = False

        self.setup_ui()

    def setup_ui(self):
        frame = tk.Frame(self.root)
        frame.pack(fill="both", expand=True, padx=20, pady=20)

        # Reloj digital
        tk.Label(frame, text="Reloj Digital:", font=("Arial", 12, "bold")).pack(anchor="w")
        self.reloj_label = tk.Label(frame, text="00:00:00", font=("Arial", 32, "monospace"))
        self.reloj_label.pack(pady=10)

        frame_reloj = tk.Frame(frame)
        frame_reloj.pack(fill="x", pady=5)

        tk.Button(frame_reloj, text="Iniciar Reloj", command=self.iniciar_reloj,
                  bg="#4CAF50", fg="white").pack(side="left", padx=5)
        tk.Button(frame_reloj, text="Detener Reloj", command=self.detener_reloj,
                  bg="#ff5252", fg="white").pack(side="left", padx=5)

        # Countdown
        tk.Label(frame, text="Cronómetro (10 segundos):", font=("Arial", 12, "bold")).pack(anc
        self.countdown_label = tk.Label(frame, text="10", font=("Arial", 32))
        self.countdown_label.pack(pady=10)

        frame_countdown = tk.Frame(frame)
        frame_countdown.pack(fill="x", pady=5)

        tk.Button(frame_countdown, text="Iniciar Countdown", command=self.iniciar_countdown,
                  bg="#2196F3", fg="white").pack(side="left", padx=5)

        # Tareas programadas
        tk.Label(frame, text="Tarea Programada:", font=("Arial", 12, "bold")).pack(anchor="w",
        self.tarea_label = tk.Label(frame, text="Estado: Inactiva", font=("Arial", 11))
        self.tarea_label.pack(pady=10)
```

```
        tk.Button(frame, text="Ejecutar Tarea cada 2 seg", command=self.programar_tarea,
                  bg="#FF9800", fg="white").pack(pady=5)
        tk.Button(frame, text="Cancelar Tarea", command=self.cancelar_tarea,
                  bg="#ff5252", fg="white").pack(pady=5)

    def actualizar_reloj(self):
        if self.reloj_activo:
            ahora = datetime.now().strftime("%H:%M:%S")
            self.reloj_label.config(text=ahora)
            self.root.after(1000, self.actualizar_reloj)  # Actualizar cada 1 segundo

    def iniciar_reloj(self):
        if not self.reloj_activo:
            self.reloj_activo = True
            self.actualizar_reloj()

    def detener_reloj(self):
        self.reloj_activo = False

    def iniciar_countdown(self):
        if not self.timer_activo:
            self.contador = 10
            self.timer_activo = True
            self.actualizar_countdown()

    def actualizar_countdown(self):
        if self.timer_activo and self.contador > 0:
            self.countdown_label.config(text=str(self.contador))
            self.contador -= 1
            self.root.after(1000, self.actualizar_countdown)
        elif self.timer_activo and self.contador == 0:
            self.countdown_label.config(text="¡Tiempo!")
            self.tarea_label.config(fg="green")
            self.timer_activo = False
            self.root.after(2000, lambda: self.countdown_label.config(text="10"))
            self.root.after(2000, lambda: self.tarea_label.config(fg="black"))

    def programar_tarea(self):
        if not hasattr(self, "tarea_id"):
            self.ejecutar_tarea_programada()

    def ejecutar_tarea_programada(self):
        self.tarea_label.config(text=f"Estado: Ejecutada a {datetime.now().strftime('%H:%M:%S'
        self.tarea_id = self.root.after(2000, self.ejecutar_tarea_programada)

    def cancelar_tarea(self):
        if hasattr(self, "tarea_id"):
            self.root.after_cancel(self.tarea_id)
            delattr(self, "tarea_id")
            self.tarea_label.config(text="Estado: Cancelada")

if __name__ == "__main__":
    root = tk.Tk()
```

```
    app = AppConTimers(root)
    root.mainloop()
```

6.3 Manejo de excepciones en GUI

```
# manejo_excepciones.py
import tkinter as tk
from tkinter import messagebox, ttk
import logging
import traceback

# Configurar logging
logging.basicConfig(
    filename="app.log",
    level=logging.DEBUG,
    format="%(asctime)s - %(levelname)s - %(message)s"
)

class AppRobusta:
    def __init__(self, root):
        self.root = root
        self.root.title("Manejo Robusto de Excepciones")
        self.root.geometry("600x500")

        # Global exception handler
        self.root.report_callback_exception = self.handle_exception

        self.setup_ui()

    def setup_ui(self):
        frame = tk.Frame(self.root)
        frame.pack(fill="both", expand=True, padx=20, pady=20)

        tk.Label(frame, text="Ejemplos de Manejo de Errores", font=("Arial", 14, "bold")).pack

        # Entrada de número
        tk.Label(frame, text="Ingresa un número:").pack(anchor="w")
        self.numero_var = tk.StringVar()
        tk.Entry(frame, textvariable=self.numero_var).pack(fill="x", pady=5)

        # Botones de prueba
        frame_botones = tk.Frame(frame)
        frame_botones.pack(fill="x", pady=10)

        tk.Button(frame_botones, text="Convertir a Int", command=self.prueba_int).pack(side="l
        tk.Button(frame_botones, text="División por cero", command=self.prueba_division).pack(
        tk.Button(frame_botones, text="Acceso lista", command=self.prueba_lista).pack(side="le

        # Log
```

```
        tk.Label(frame, text="Log de Eventos:", font=("Arial", 11, "bold")).pack(anchor="w", p

        self.log_text = tk.Text(frame, height=15, width=60, font=("Courier", 9))
        self.log_text.pack(fill="both", expand=True)

        # Botón para ver archivo log
        tk.Button(frame, text="Ver archivo log completo", command=self.ver_log_archivo).pack(p

    def log_mensaje(self, nivel, mensaje):
        timestamp = __import__('datetime').datetime.now().strftime("%Y-%m-%d %H:%M:%S")
        msg_formateado = f"[{timestamp}] {nivel}: {mensaje}"
        self.log_text.insert(tk.END, msg_formateado + "\n")
        self.log_text.see(tk.END)

        # También guardar en archivo
        if nivel == "ERROR":
            logging.error(mensaje)
        elif nivel == "WARNING":
            logging.warning(mensaje)
        else:
            logging.info(mensaje)

    def prueba_int(self):
        try:
            valor = self.numero_var.get()
            numero = int(valor)
            self.log_mensaje("INFO", f"Conversión exitosa: {numero}")
            messagebox.showinfo("Éxito", f"Número: {numero}")
        except ValueError:
            self.log_mensaje("ERROR", f"No se puede convertir '{valor}' a entero")
            messagebox.showerror("Error", f"'{valor}' no es un número válido")
        except Exception as e:
            self.handle_exception(Exception, e, e.__traceback__)

    def prueba_division(self):
        try:
            numero = float(self.numero_var.get())
            resultado = 10 / numero
            self.log_mensaje("INFO", f"División: 10 / {numero} = {resultado}")
        except ValueError:
            self.log_mensaje("ERROR", "Entrada no numérica")
            messagebox.showerror("Error", "Debes ingresar un número")
        except ZeroDivisionError:
            self.log_mensaje("ERROR", "División por cero")
            messagebox.showerror("Error", "No se puede dividir por cero")
        except Exception as e:
            self.handle_exception(Exception, e, e.__traceback__)

    def prueba_lista(self):
        try:
            indice = int(self.numero_var.get())
            lista = [10, 20, 30]
            valor = lista[indice]
```

```
            self.log_mensaje("INFO", f"Acceso a lista[{indice}] = {valor}")
        except ValueError:
            self.log_mensaje("ERROR", "El índice debe ser un número")
            messagebox.showerror("Error", "Índice inválido")
        except IndexError:
            self.log_mensaje("ERROR", f"Índice fuera de rango")
            messagebox.showerror("Error", f"Índice {indice} fuera de rango (0-2)")
        except Exception as e:
            self.handle_exception(Exception, e, e.__traceback__)

    def handle_exception(self, exc_type, exc_value, exc_traceback):
        """Manejador global de excepciones"""
        if issubclass(exc_type, KeyboardInterrupt):
            sys.__excepthook__(exc_type, exc_value, exc_traceback)
            return

        # Log completo
        tb_str = "".join(traceback.format_exception(exc_type, exc_value, exc_traceback))
        logging.error(f"Excepción no capturada:\n{tb_str}")

        # Mostrar en GUI
        self.log_mensaje("ERROR", f"{exc_type.__name__}: {exc_value}")

        # Mostrar diálogo
        messagebox.showerror(
            "Error Inesperado",
            f"{exc_type.__name__}: {exc_value}\n\nRevisa el log para más detalles"
        )

    def ver_log_archivo(self):
        try:
            with open("app.log", "r") as f:
                contenido = f.read()

            ventana_log = tk.Toplevel(self.root)
            ventana_log.title("Archivo Log Completo")
            ventana_log.geometry("700x500")

            text = tk.Text(ventana_log, font=("Courier", 9))
            text.pack(fill="both", expand=True, padx=10, pady=10)
            text.insert(tk.END, contenido)
            text.config(state="disabled")
        except Exception as e:
            messagebox.showerror("Error", f"No se pudo abrir el log: {e}")

if __name__ == "__main__":
    import sys
    root = tk.Tk()
    app = AppRobusta(root)
    root.mainloop()
```

6.4 Optimización de rendimiento

```
# optimizacion_rendimiento.py
import tkinter as tk
from tkinter import ttk, messagebox
import time
import threading

class OptimizationDemo:
    def __init__(self, root):
        self.root = root
        self.root.title("Optimización de Rendimiento")
        self.root.geometry("700x600")

        self.setup_ui()

    def setup_ui(self):
        notebook = ttk.Notebook(self.root)
        notebook.pack(fill="both", expand=True, padx=10, pady=10)

        # Tab 1: Renderizado eficiente
        frame1 = ttk.Frame(notebook)
        notebook.add(frame1, text="Renderizado Eficiente")
        self.tab_renderizado(frame1)

        # Tab 2: Gestión de memoria
        frame2 = ttk.Frame(notebook)
        notebook.add(frame2, text="Gestión de Memoria")
        self.tab_memoria(frame2)

        # Tab 3: Cache
        frame3 = ttk.Frame(notebook)
        notebook.add(frame3, text="Cache")
        self.tab_cache(frame3)

    def tab_renderizado(self, parent):
        frame = tk.Frame(parent)
        frame.pack(fill="both", expand=True, padx=20, pady=20)

        tk.Label(frame, text="Renderizado Eficiente", font=("Arial", 12, "bold")).pack()
        tk.Label(frame, text="❌ MAL: Actualizar 1000 labels en bucle").pack(anchor="w", pady=

        def renderizado_malo():
            ventana = tk.Toplevel(self.root)
            ventana.title("Renderizado Ineficiente")
            ventana.geometry("300x300")

            frame_labels = tk.Frame(ventana)
            frame_labels.pack(fill="both", expand=True)

            inicio = time.time()
```

```
            for i in range(100):
                tk.Label(frame_labels, text=f"Label {i}").pack()
                ventana.update()  # ❌ Mala práctica

            tiempo = time.time() - inicio
            messagebox.showinfo("Tiempo", f"Tiempo: {tiempo:.2f}s (ineficiente)")

        tk.Button(frame, text="Renderizado Ineficiente", command=renderizado_malo,
                  bg="#ff5252", fg="white").pack(fill="x", pady=5)

        tk.Label(frame, text="✓ BIEN: Usar grid_remove() y actualizar al final").pack(anchor='

        def renderizado_bueno():
            ventana = tk.Toplevel(self.root)
            ventana.title("Renderizado Eficiente")
            ventana.geometry("300x300")

            frame_labels = tk.Frame(ventana)
            frame_labels.pack(fill="both", expand=True)

            inicio = time.time()

            # Crear todos sin actualizar
            labels = []
            for i in range(100):
                lbl = tk.Label(frame_labels, text=f"Label {i}")
                lbl.pack()
                labels.append(lbl)

            # Una sola actualización
            ventana.update()

            tiempo = time.time() - inicio
            messagebox.showinfo("Tiempo", f"Tiempo: {tiempo:.4f}s (eficiente)")

        tk.Button(frame, text="Renderizado Eficiente", command=renderizado_bueno,
                  bg="#4CAF50", fg="white").pack(fill="x", pady=5)

    def tab_memoria(self, parent):
        frame = tk.Frame(parent)
        frame.pack(fill="both", expand=True, padx=20, pady=20)

        tk.Label(frame, text="Gestión de Memoria", font=("Arial", 12, "bold")).pack()

        self.datos_grande = None

        def crear_lista_grande():
            self.datos_grande = list(range(1000000))
            messagebox.showinfo("Info", "Lista de 1M elementos creada")

        def limpiar_memoria():
            self.datos_grande = None
            import gc
```

```
        gc.collect()
        messagebox.showinfo("Info", "Memoria liberada")

    tk.Button(frame, text="Crear lista grande", command=crear_lista_grande,
              bg="#2196F3", fg="white").pack(fill="x", pady=5)
    tk.Button(frame, text="Limpiar memoria", command=limpiar_memoria,
              bg="#4CAF50", fg="white").pack(fill="x", pady=5)

    tk.Label(frame, text="Tip: Libera variables grandes que no necesites").pack(anchor="w"

def tab_cache(self, parent):
    frame = tk.Frame(parent)
    frame.pack(fill="both", expand=True, padx=20, pady=20)

    tk.Label(frame, text="Uso de Cache", font=("Arial", 12, "bold")).pack()

    self.cache = {}

    def fibonacci_sin_cache(n):
        if n <= 1:
            return n
        return fibonacci_sin_cache(n-1) + fibonacci_sin_cache(n-2)

    def fibonacci_con_cache(n):
        if n in self.cache:
            return self.cache[n]

        if n <= 1:
            resultado = n
        else:
            resultado = fibonacci_con_cache(n-1) + fibonacci_con_cache(n-2)

        self.cache[n] = resultado
        return resultado

    def test_sin_cache():
        inicio = time.time()
        resultado = fibonacci_sin_cache(30)
        tiempo = time.time() - inicio
        messagebox.showinfo("Sin Cache", f"Resultado: {resultado}\nTiempo: {tiempo:.2f}s")

    def test_con_cache():
        self.cache.clear()
        inicio = time.time()
        resultado = fibonacci_con_cache(30)
        tiempo = time.time() - inicio
        messagebox.showinfo("Con Cache", f"Resultado: {resultado}\nTiempo: {tiempo:.4f}s")

    tk.Button(frame, text="Fibonacci sin Cache", command=test_sin_cache,
              bg="#ff5252", fg="white").pack(fill="x", pady=5)
    tk.Button(frame, text="Fibonacci con Cache", command=test_con_cache,
              bg="#4CAF50", fg="white").pack(fill="x", pady=5)
```

```
if __name__ == "__main__":
    root = tk.Tk()
    app = OptimizationDemo(root)
    root.mainloop()
```

🔷 MÓDULO 7 — Proyecto Final Integrador

Mini ERP Simplificado

```
# mini_erp.py
"""
Mini ERP: Sistema integrado de gestión empresarial básico
- Gestión de productos
- Gestión de ventas
- Reportes
- Base de datos
"""

import tkinter as tk
from tkinter import messagebox, ttk, filedialog
import sqlite3
import json
from datetime import datetime
import threading

class MiniERP:
    def __init__(self, root):
        self.root = root
        self.root.title("Mini ERP - Sistema de Gestión")
        self.root.geometry("1200x700")
        self.root.state('zoomed')  # Maximizar

        self.db_file = "mini_erp.db"
        self.init_db()

        self.crear_ui_principal()

    def init_db(self):
        conn = sqlite3.connect(self.db_file)
        cursor = conn.cursor()

        # Tabla de productos
        cursor.execute("""
        CREATE TABLE IF NOT EXISTS productos (
            id INTEGER PRIMARY KEY,
            codigo TEXT UNIQUE NOT NULL,
            nombre TEXT NOT NULL,
            descripcion TEXT,
```

```
            precio_costo REAL NOT NULL,
            precio_venta REAL NOT NULL,
            cantidad_stock INTEGER NOT NULL,
            categoria TEXT,
            fecha_creacion TIMESTAMP DEFAULT CURRENT_TIMESTAMP
        )
        """)

        # Tabla de ventas
        cursor.execute("""
        CREATE TABLE IF NOT EXISTS ventas (
            id INTEGER PRIMARY KEY,
            numero_venta TEXT UNIQUE NOT NULL,
            fecha TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
            cliente TEXT NOT NULL,
            total REAL NOT NULL,
            estado TEXT DEFAULT 'Completada'
        )
        """)

        # Tabla de detalles venta
        cursor.execute("""
        CREATE TABLE IF NOT EXISTS detalles_venta (
            id INTEGER PRIMARY KEY,
            venta_id INTEGER,
            producto_id INTEGER,
            cantidad INTEGER NOT NULL,
            precio_unitario REAL NOT NULL,
            subtotal REAL NOT NULL,
            FOREIGN KEY(venta_id) REFERENCES ventas(id),
            FOREIGN KEY(producto_id) REFERENCES productos(id)
        )
        """)

        conn.commit()
        conn.close()

    def crear_ui_principal(self):
        # Header
        header = tk.Frame(self.root, bg="#1a1a2e", height=80)
        header.pack(fill="x")
        header.pack_propagate(False)

        tk.Label(header, text="🏢 MINI ERP - Sistema Integrado de Gestión",
                 font=("Arial", 18, "bold"), fg="white", bg="#1a1a2e").pack(side="left", padx=2
        tk.Label(header, text=f"Fecha: {datetime.now().strftime('%d/%m/%Y %H:%M')}",
                 font=("Arial", 10), fg="#ccc", bg="#1a1a2e").pack(side="right", padx=20, pady=

        # Contenedor principal
        container = tk.Frame(self.root)
        container.pack(fill="both", expand=True)

        # Sidebar
```

```
        sidebar = tk.Frame(container, bg="#16213e", width=250)
        sidebar.pack(side="left", fill="y")
        sidebar.pack_propagate(False)

        tk.Label(sidebar, text="MÓDULOS", font=("Arial", 12, "bold"),
                 fg="white", bg="#16213e", pady=15).pack(fill="x")

        modulos = [
            ("📊 Dashboard", self.mostrar_dashboard),
            ("📦 Productos", self.mostrar_productos),
            ("🛒 Ventas", self.mostrar_ventas),
            ("📈 Reportes", self.mostrar_reportes),
            ("⚙️ Configuración", self.mostrar_config),
            ("🚪 Salir", self.root.quit)
        ]

        for texto, cmd in modulos:
            btn = tk.Button(sidebar, text=texto, font=("Arial", 11),
                            bg="#16213e", fg="white", bd=0, padx=20, pady=15,
                            activebackground="#0f3460", command=cmd, anchor="w")
            btn.pack(fill="x")

        # Área de contenido
        self.content = tk.Frame(container, bg="white")
        self.content.pack(side="right", fill="both", expand=True)

        # Footer
        footer = tk.Frame(self.root, bg="#f0f0f0", height=30)
        footer.pack(fill="x")
        footer.pack_propagate(False)

        tk.Label(footer, text="Mini ERP v1.0 © 2026 - Sistema Educativo",
                 font=("Arial", 9), bg="#f0f0f0").pack(pady=5)

        # Mostrar dashboard por defecto
        self.mostrar_dashboard()

    def limpiar_content(self):
        for widget in self.content.winfo_children():
            widget.destroy()

    def mostrar_dashboard(self):
        self.limpiar_content()

        tk.Label(self.content, text="Dashboard", font=("Arial", 16, "bold")).pack(pady=20)

        frame_stats = tk.Frame(self.content)
        frame_stats.pack(fill="x", padx=20, pady=10)

        # Obtener estadísticas
        conn = sqlite3.connect(self.db_file)
        cursor = conn.cursor()
```

```
        cursor.execute("SELECT COUNT(*) FROM productos")
        total_productos = cursor.fetchone()[0]

        cursor.execute("SELECT COUNT(*) FROM ventas")
        total_ventas = cursor.fetchone()[0]

        cursor.execute("SELECT SUM(total) FROM ventas")
        total_vendido = cursor.fetchone()[0] or 0

        cursor.execute("SELECT SUM(cantidad_stock) FROM productos")
        stock_total = cursor.fetchone()[0] or 0

        conn.close()

        # Tarjetas de estadísticas
        stats = [
            ("Productos", total_productos, "#4CAF50"),
            ("Ventas", total_ventas, "#2196F3"),
            ("Total Vendido", f"${total_vendido:.2f}", "#FF9800"),
            ("Stock Total", stock_total, "#e91e63")
        ]

        for titulo, valor, color in stats:
            card = tk.Frame(frame_stats, bg=color, relief="raised", bd=2)
            card.pack(side="left", fill="both", expand=True, padx=10, ipady=20)

            tk.Label(card, text=titulo, font=("Arial", 11), fg="white", bg=color).pack()
            tk.Label(card, text=str(valor), font=("Arial", 18, "bold"), fg="white", bg=color).

    def mostrar_productos(self):
        self.limpiar_content()

        tk.Label(self.content, text="Gestión de Productos", font=("Arial", 16, "bold")).pack(p

        # Formulario
        frame_form = tk.LabelFrame(self.content, text="Nuevo Producto", padx=10, pady=10)
        frame_form.pack(fill="x", padx=20, pady=10)

        form_data = {}

        campos = [
            ("Código:", "codigo"),
            ("Nombre:", "nombre"),
            ("Descripción:", "descripcion"),
            ("Precio Costo:", "precio_costo"),
            ("Precio Venta:", "precio_venta"),
            ("Cantidad Stock:", "cantidad"),
            ("Categoría:", "categoria")
        ]

        for label, key in campos:
            tk.Label(frame_form, text=label).pack(anchor="w")
            var = tk.StringVar()
```

```
            tk.Entry(frame_form, textvariable=var, width=40).pack(pady=3, fill="x")
            form_data[key] = var

        def agregar_producto():
            try:
                datos = {k: v.get() for k, v in form_data.items()}

                if not all(datos.values()):
                    messagebox.showerror("Error", "Completa todos los campos")
                    return

                conn = sqlite3.connect(self.db_file)
                cursor = conn.cursor()
                cursor.execute("""
                INSERT INTO productos (codigo, nombre, descripcion, precio_costo, precio_venta
                VALUES (?, ?, ?, ?, ?, ?, ?)
                """, (datos["codigo"], datos["nombre"], datos["descripcion"],
                      float(datos["precio_costo"]), float(datos["precio_venta"]),
                      int(datos["cantidad"]), datos["categoria"]))
                conn.commit()
                conn.close()

                messagebox.showinfo("Éxito", "Producto agregado")
                for v in form_data.values():
                    v.set("")
                self.cargar_tabla_productos()
            except Exception as e:
                messagebox.showerror("Error", f"Error: {e}")

        tk.Button(frame_form, text="Agregar Producto", command=agregar_producto,
                  bg="#4CAF50", fg="white").pack(pady=10)

        # Tabla de productos
        frame_tabla = tk.LabelFrame(self.content, text="Productos Registrados", padx=10, pady=
        frame_tabla.pack(fill="both", expand=True, padx=20, pady=10)

        self.tree_productos = ttk.Treeview(frame_tabla,
                                           columns=("Código", "Nombre", "Precio Costo", "Preci
                                           height=15)
        self.tree_productos.heading("#0", text="ID")
        self.tree_productos.column("#0", width=30)

        for col in ("Código", "Nombre", "Precio Costo", "Precio Venta", "Stock", "Categoría"):
            self.tree_productos.heading(col, text=col)
            self.tree_productos.column(col, width=120)

        self.tree_productos.pack(fill="both", expand=True)

        self.cargar_tabla_productos()

    def cargar_tabla_productos(self):
        for item in self.tree_productos.get_children():
            self.tree_productos.delete(item)
```

```
        conn = sqlite3.connect(self.db_file)
        cursor = conn.cursor()
        cursor.execute("SELECT id, codigo, nombre, precio_costo, precio_venta, cantidad_stock,

        for fila in cursor.fetchall():
            self.tree_productos.insert("", tk.END, text=fila[0],
                                       values=(fila[1], fila[2], f"${fila[3]:.2f}", f"${fila[4]

        conn.close()

    def mostrar_ventas(self):
        self.limpiar_content()

        tk.Label(self.content, text="Gestión de Ventas", font=("Arial", 16, "bold")).pack(pady
        tk.Label(self.content, text="(Módulo en desarrollo)").pack()

    def mostrar_reportes(self):
        self.limpiar_content()

        tk.Label(self.content, text="Reportes", font=("Arial", 16, "bold")).pack(pady=10)

        conn = sqlite3.connect(self.db_file)
        cursor = conn.cursor()

        cursor.execute("SELECT COUNT(*) as total FROM productos")
        total_prod = cursor.fetchone()[0]

        cursor.execute("SELECT AVG(precio_venta) as promedio FROM productos")
        promedio_precio = cursor.fetchone()[0] or 0

        cursor.execute("SELECT SUM(cantidad_stock * precio_venta) as valor_inventario FROM pro
        valor_inv = cursor.fetchone()[0] or 0

        conn.close()

        frame_reportes = tk.Frame(self.content)
        frame_reportes.pack(fill="x", padx=20, pady=20)

        tk.Label(frame_reportes, text=f"Total de Productos: {total_prod}", font=("Arial", 12))
        tk.Label(frame_reportes, text=f"Precio Promedio: ${promedio_precio:.2f}", font=("Arial
        tk.Label(frame_reportes, text=f"Valor Total de Inventario: ${valor_inv:.2f}", font=("A

    def mostrar_config(self):
        self.limpiar_content()

        tk.Label(self.content, text="Configuración", font=("Arial", 16, "bold")).pack(pady=20)

        frame = tk.Frame(self.content)
        frame.pack(padx=20, pady=20)

        tk.Button(frame, text="Exportar Datos a JSON", command=self.exportar_datos,
                  bg="#2196F3", fg="white", padx=20, pady=10).pack(pady=5)
```

```
        tk.Button(frame, text="Limpiar Base de Datos", command=self.limpiar_db,
                  bg="#ff5252", fg="white", padx=20, pady=10).pack(pady=5)

    def exportar_datos(self):
        try:
            archivo = filedialog.asksaveasfilename(defaultextension=".json")
            if archivo:
                conn = sqlite3.connect(self.db_file)
                cursor = conn.cursor()

                cursor.execute("SELECT * FROM productos")
                productos = [dict(zip([d[0] for d in cursor.description], fila)) for fila in c

                with open(archivo, "w", encoding="utf-8") as f:
                    json.dump(productos, f, indent=4, ensure_ascii=False)

                messagebox.showinfo("Éxito", f"Datos exportados a {archivo}")
                conn.close()
        except Exception as e:
            messagebox.showerror("Error", f"Error: {e}")

    def limpiar_db(self):
        if messagebox.askyesno("Confirmar", "¿Eliminar todos los datos?"):
            try:
                import os
                os.remove(self.db_file)
                self.init_db()
                messagebox.showinfo("Éxito", "Base de datos limpiada")
            except Exception as e:
                messagebox.showerror("Error", f"Error: {e}")

if __name__ == "__main__":
    root = tk.Tk()
    app = MiniERP(root)
    root.mainloop()
```

🔹 EXTRA — Empaquetado con PyInstaller

```
# instalación
pip install pyinstaller

# Crear ejecutable simple
pyinstaller --onefile mini_erp.py

# Con icono
pyinstaller --onefile --icon=icon.ico mini_erp.py

# Sin consola (GUI)
pyinstaller --onefile --windowed mini_erp.py
```

```
# Archivo .spec avanzado
pyinstaller --onefile --windowed --icon=icon.ico --name="Mini ERP" --add-data="datos:datos" mi
```