Algoritmos y Estructuras de Datos II

Trabajo Práctico N° 2	Unidad X3- Python Tkinter & SQLite
Modalidad: Semi -Presencial	**Estratégica Didáctica:** Trabajo individual
Metodología de Desarrollo: Det. docente	**Metodología de Corrección:** Via Classroom.
Carácter de Trabajo: Obligatorio – Con Nota	**Fecha Entrega:** A confirmar por el Docente.

Python TKinter

MARCO TEÓRICO

1. ¿Qué entiende por GUI?

Una GUI (Interfaz Gráfica de Usuario) es la forma en que interactuamos con un dispositivo sin necesidad de escribir comandos de texto. Es un entorno visual amigable que utiliza un conjunto de imágenes y objetos gráficos, como ventanas, botones y menús, para representar la información y las acciones disponibles. Ejemplos claros son los entornos de Windows, macOS o Android

2. Describir 3 Framework de GUI para Python, Adicionalmente que se usen en C++

Hay tres frameworks para Python:

PyQT5: Es un framework muy popular que consiste en un conjunto de "bindings" (enlaces) de Python para el framework Qt, el cual es usado en C++. Permite crear interfaces sofisticadas y multiplataforma.

Tkinter: Es una de las bibliotecas más populares y se considera la interfaz estándar de Python para el kit de herramientas de Tk. Su gran ventaja es que viene incorporada en Python, por lo que no requiere instalación adicional.

wxPython: Es un kit de herramientas GUI multiplataforma que permite a los programadores de Python crear interfaces robustas. Se implementa como un envoltorio (wrapper) de la popular biblioteca wxWidgets, que está escrita en C++

3. ¿Qué diferencia tienen los módulos TK y TTK?

La principal diferencia es que TK se refiere a los widgets tradicionales y básicos de Tkinter. Son simples, pero su apariencia puede variar mucho entre un sistema operativo y otro. TTK (Themed Tk) es un conjunto de widgets más moderno que

se introdujo para mejorar los widgets TK tradicionales. Los widgets TTK ofrecen una apariencia más moderna y coherente en diferentes plataformas y tienen más capacidad de personalización.
En los ejemplos de la clase, se importan ambos, pero se prefiere usar los widgets ttk como: **ttk.Label** y **ttk.Button**

4. ¿Que es un Widget?, dar 5 ejemplos

Un widget es un elemento gráfico o control que se coloca en la interfaz [220]. Son los bloques de construcción fundamentales de la GUI con los que el usuario interactúa.

- **Label** (Etiqueta): Se usa para mostrar texto estático o imágenes.
- **Entry** (Entrada): Una caja que permite al usuario introducir una línea corta de texto.
- **Button** (Botón): Un botón que ejecuta una función cuando el usuario hace clic en él.
- **Checkbutton** (Casilla de verificación): Permite al usuario elegir varias opciones de una lista.
- **Frame** (Marco): Un contenedor que sirve para agrupar y organizar otros widgets.

5. ¿Explicar el Widget Notebook en Tkinter?

El widget Notebook es un contenedor que permite organizar la interfaz en múltiples páginas o pestañas.
El usuario puede hacer clic en las diferentes pestañas para cambiar entre distintos conjuntos de widgets. Esto es muy útil para aplicaciones complejas, como el ejemplo de "Mantenimiento de artículos", donde se usa un **ttk.Notebook** para separar la aplicación en "Carga de artículos", "Consulta por código" y "Listado completo"

6. ¿Cómo implemento menús en Python con Tkinter.

Para implementar menús, se utiliza la clase Menu de Tkinter. El proceso general, descrito en los documentos, implica crear una barra de menú principal **(menubar = tk.Menu(root))**, y luego crear los menús desplegables (como "Archivo") que se añadirán a esa barra **(file_menu = tk.Menu(menubar))**. A este menú se le añaden las opciones con **.add_command()** y separadores con **.add_separator()**. Finalmente, se "conecta" el menú a la barra con **.add_cascade()** y se asigna la barra de menú a la ventana principal con **root.config(menu=menubar)**.

7. ¿Se puede realizar interfaces graficas en Python con Paradigma estructurado y objetos?

Sí, se pueden usar ambos paradigmas. El material de la clase es un ejemplo perfecto de esto, ya que construye la misma aplicación (un conversor de temperatura) de dos maneras:

1. **Sin Orientación a Objetos** (Estructurado): Define los widgets y las funciones (como **convertir_temp()**) directamente en el script. Es un enfoque válido para aplicaciones pequeñas.
2. **Con Orientación a Objetos** (POO): Crea una clase (ej. **class Aplicacion(ttk.Frame)**) que encapsula y gestiona la interfaz. Los widgets son atributos y las funciones son métodos de esa clase. Este enfoque es más útil para aplicaciones grandes.

8. Describir los 3 métodos para Colocar Widgets en una GUI con TKinter en Python.

Tkinter ofrece tres administradores de geometría para posicionar widgets:

- **Place:** Es el más sencillo de entender. Permite posicionar widgets usando coordenadas absolutas (X e Y), dándote control total sobre la posición exacta. Es el que se usa en el ejemplo del conversor de temperatura (ej. **etiqueta_temp_celsius.place(x=20, y=20)**).

- **Pack:** Este administrador "empaqueta" los widgets uno después del otro, ya sea verticalmente (por defecto) u horizontalmente. Es simple para diseños básicos.

- **Grid:** Organiza los widgets en una matriz de filas y columnas [345, 880]. Simplemente se especifica la fila y columna (ej. widget.grid**(row=0, column=1)**. A menudo es el método preferido por su flexibilidad para crear diseños alineados y complejos.

9. ¿Cómo maneja los eventos en TKinter?, de ser posible ampliar en la Web con “Bind”.

La forma más común de manejar eventos que se ve en la presentación es usando la opción **command** al crear un widget. Por ejemplo, al crear un botón: **boton_convertir = ttk.Button(text="Convertir", command=convertir_temp)**. Esto vincula el clic del botón directamente a la función **convertir_temp**, que se ejecutará automáticamente.

El **método bind** es un manejador de eventos más general. Mientras que command suele ser solo para clics en botones, bind permite asociar una función a una variedad mucho más amplia de eventos (como presionar una tecla, mover el ratón, o hacer clic con un botón específico del ratón como **<Button-1>**) en cualquier widget

Python - SQLite

MARCO TEÓRICO

10. ¿Qué es SQLite, cómo se integra a Python?

SQLite es un sistema de gestión de bases de datos relacional, pero a diferencia de otros sistemas, es ligero, autónomo y no requiere un servidor. Se integra directamente en la aplicación y almacena la base de datos completa en un único archivo. Es ideal para aplicaciones pequeñas, sistemas embebidos o

almacenamiento local, siendo usado por programas como Firefox y sistemas operativos como Android e iOS.

Se integra a Python de forma nativa, ya que viene incluido con Python como parte de su biblioteca estándar. No se necesita instalar ningún módulo adicional; solo basta con **import sqlite3** para empezar a usarlo.

11. ¿Cómo me conecto a una base de datos SQLite desde Python?

La conexión es muy directa. Después de importar el módulo (**import sqlite3**), se utiliza la función connect y se le pasa como argumento el nombre del archivo que contendrá la base de datos.
El ejemplo de la clase es: **conexion=sqlite3.connect("bd1.db")**. Si el archivo **"bd1.db"** no existe, este comando lo creará automáticamente

12. ¿Cómo ejecuto una sentencia SQL en Python + SQLite?

Se utiliza el método **execute()** del objeto de conexión. Para sentencias como CREATE TABLE, INSERT, UPDATE o DELETE, simplemente se pasa el comando SQL como una cadena de texto. Si se ejecuta una consulta SELECT, el método execute() devuelve un objeto Cursor que se puede iterar para obtener los resultados

13. ¿Cómo creo una Tabla en Python – SQLite?

Se crea una tabla ejecutando el comando SQL CREATE TABLE a través del método execute(). La presentación muestra un ejemplo claro donde se define el nombre de la tabla, las columnas (código, descripción, precio) y sus tipos de datos (integer primary key autoincrement, text, real).
Una buena práctica, es usar **CREATE TABLE IF NOT EXISTS** para evitar que el programa falle si la tabla ya había sido creada.

14. Dar ejemplo de Consultas (Select), que devuelva todos los Registros de una tabla.

Para obtener todos los registros, se ejecuta una sentencia **SELECT** sin cláusula **WHERE** . Esto devuelve un objeto Cursor. Luego, se puede usar un **bucle for** para iterar sobre ese cursor, y cada fila será una tupla con los datos de un registro

```
cursor=conexion.execute("select codigo, descripcion, precio from articulos")
      for fila in cursor:
      print(fila)
```

15. Dar ejemplo de Consultas (Select), que devuelva un Registro de una tabla.

Para obtener un solo registro, se usa una consulta **SELECT** con una cláusula **WHERE** para filtrar por un valor único (como el código). En lugar de un bucle for, se utiliza el método **fetchone()** del cursor. Este método devuelve una única tupla con la fila encontrada, o devuelve **None** si no se encontró ningún registro que coincida. Esto es útil para verificar si un artículo existe.

16. Dar ejemplos de insert, delete y Update en Python – SQLite

- **Insert**: Se usa el comando INSERT INTO. Es una buena práctica usar placeholders (?) para pasar los datos de forma segura. Después de ejecutar, es crucial llamar a **conexion.commit()** para guardar los cambios.

```
conexion.execute("insert into articulos (descripcion, precio) values (?,?)",
("naranjas", 23.50))
conexion.commit()
```

- **Update:** Se usa el comando **UPDATE**, especificando qué columna SET (actualizar) y una cláusula WHERE para identificar la fila. También requiere commit.

```
conexion.execute("update articulos set descripcion = 'palta' where codigo=1")
conexion.commit()
```

- **Delete:** Se usa DELETE FROM con una cláusula WHERE para especificar qué fila borrar. También requiere commit.

```
conexion.execute("delete from articulos where codigo=1")
conexion.commit()
```

Marco Práctico Integrado Python - SQLite

Marco Práctico: Realizar en Python

1. Construir una calculadora (suma, resta, multiplicar y dividir)
2. Construir una APP que permita administrar los pedidos de servicio técnico a un “Repair Center”.
    - Ingresar Apellido y Nombre del Cliente.
    - Ingresar la dirección (calle y altura)
    - Ingresar el Inconveniente
    - Asignar un Técnico
    - Agendar la Visita fecha y hora.
3. Para el Ejercicio anterior agrega runa pantalla inicial de Logín (usuario y clave para Ingresar)

```python
import tkinter as tk
from tkinter import messagebox, ttk

class Calculadora:
    def __init__(self, root):
        self.root = root
        self.root.title("Calculadora Básica")
        self.root.geometry("400x500")
        self.root.resizable(False, False)
        self.root.configure(bg="#f0f0f0")

        self.crear_interfaz()

    def crear_interfaz(self):
        # --- Estilos ---
        style = ttk.Style()
        style.configure("TLabel", font=("Arial", 12), background="#f0f0f0")
        style.configure("TButton", font=("Arial", 11, "bold"), padding=10)
        style.configure("TCombobox", font=("Arial", 11))

        # --- Entrada 1 ---
        ttk.Label(self.root, text="Primer número:").pack(pady=(20, 5))
        self.entry_num1 = ttk.Entry(self.root, font=("Arial", 12), justify="center")
        self.entry_num1.pack(pady=5, ipadx=10, ipady=8)

        # --- Entrada 2 ---
        ttk.Label(self.root, text="Segundo número:").pack(pady=(15, 5))
        self.entry_num2 = ttk.Entry(self.root, font=("Arial", 12), justify="center")
        self.entry_num2.pack(pady=5, ipadx=10, ipady=8)

        # --- Operación ---
        ttk.Label(self.root, text="Operación:").pack(pady=(15, 5))
        self.combo_op = ttk.Combobox(
            self.root,
            values=["Sumar", "Restar", "Multiplicar", "Dividir"],
            state="readonly",
            font=("Arial", 11)
        )
        self.combo_op.set("Sumar")
        self.combo_op.pack(pady=5, ipadx=10, ipady=8)

        # --- Botón Calcular ---
        btn_calcular = ttk.Button(
            self.root,
            text="CALCULAR",
            command=self.calcular
        )
        btn_calcular.pack(pady=20, ipadx=20, ipady=10)

        # --- Resultado ---
        self.label_resultado = ttk.Label(
            self.root,
            text="Resultado: -",
            font=("Arial", 14, "bold"),
            foreground="#2c3e50"
        )
        self.label_resultado.pack(pady=20)
```

```
58
59      def calcular(self):
60          try:
61              num1 = float(self.entry_num1.get())
62              num2 = float(self.entry_num2.get())
63              operacion = self.combo_op.get()
64
65              if operacion == "Sumar":
66                  resultado = num1 + num2
67              elif operacion == "Restar":
68                  resultado = num1 - num2
69              elif operacion == "Multiplicar":
70                  resultado = num1 * num2
71              elif operacion == "Dividir":
72                  if num2 == 0:
73                      messagebox.showerror("Error", "No se puede dividir por cero.")
74                      return
75                  resultado = num1 / num2
76
77              self.label_resultado.config(text=f"Resultado: {resultado:.4f}")
78
79          except ValueError:
80              messagebox.showerror("Error", "Por favor ingresa números válidos.")
81          except Exception as e:
82              messagebox.showerror("Error", f"Ocurrió un error: {e}")
83
84
85  # --- EJECUTAR APP ---
86  if __name__ == "__main__":
87      root = tk.Tk()
88      app = Calculadora(root)
89      root.mainloop()
```

repair_center_app.py > ...

```python
import tkinter as tk
from tkinter import ttk, messagebox, simpledialog
import sqlite3
from datetime import datetime
import re

# ----------------------------------------
# BASE DE DATOS SQLITE
# ----------------------------------------
def init_db():
    conn = sqlite3.connect('repair_center.db')
    c = conn.cursor()
    c.execute('''
        CREATE TABLE IF NOT EXISTS pedidos (
            id INTEGER PRIMARY KEY AUTOINCREMENT,
            nombre TEXT NOT NULL,
            apellido TEXT NOT NULL,
            calle TEXT NOT NULL,
            altura TEXT NOT NULL,
            inconveniente TEXT NOT NULL,
            tecnico TEXT NOT NULL,
            fecha TEXT NOT NULL,
            hora TEXT NOT NULL,
            fecha_registro TEXT NOT NULL
        )
    ''')
    # Técnicos predefinidos
    c.execute('CREATE TABLE IF NOT EXISTS tecnicos (nombre TEXT UNIQUE)')
    tecnicos = ["Juan Pérez", "María Gómez", "Carlos López"]
    for t in tecnicos:
        c.execute('INSERT OR IGNORE INTO tecnicos VALUES (?)', (t,))
    conn.commit()
    conn.close()

# ----------------------------------------
# APLICACIÓN PRINCIPAL
# ----------------------------------------
class RepairCenterApp:
    def __init__(self, root):
        self.root = root
        self.root.title("Repair Center - Sistema Integrado")
        self.root.geometry("900x600")
        self.root.configure(bg="#ecf0f1")

        init_db()  # Inicializar DB

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

        self.crear_pestana_calculadora()
        self.crear_pestana_pedidos()
```

```python
    # ----------------------------------
    # PESTAÑA 1: CALCULADORA
    # ----------------------------------
    def crear_pestana_calculadora(self):
        frame = ttk.Frame(self.notebook)
        self.notebook.add(frame, text="  Calculadora  ")

        # Estilos
        style = ttk.Style()
        style.configure("Calc.TLabel", font=("Arial", 12), background="#ecf0f1")
        style.configure("Calc.TButton", font=("Arial", 11, "bold"))

        # Título
        ttk.Label(frame, text="CALCULADORA BÁSICA", font=("Arial", 16, "bold"),
                  foreground="#2c3e50").pack(pady=20)

        # Número 1
        ttk.Label(frame, text="Primer número:", style="Calc.TLabel").pack(pady=(10, 5))
        self.entry_num1 = ttk.Entry(frame, font=("Arial", 12), justify="center")
        self.entry_num1.pack(pady=5, ipadx=10, ipady=8)

        # Número 2
        ttk.Label(frame, text="Segundo número:", style="Calc.TLabel").pack(pady=(15, 5))
        self.entry_num2 = ttk.Entry(frame, font=("Arial", 12), justify="center")
        self.entry_num2.pack(pady=5, ipadx=10, ipady=8)

        # Operación
        ttk.Label(frame, text="Operación:", style="Calc.TLabel").pack(pady=(15, 5))
        self.combo_op = ttk.Combobox(frame, values=["Sumar", "Restar", "Multiplicar", "Dividir"],
                                     state="readonly", font=("Arial", 11))
        self.combo_op.set("Sumar")
        self.combo_op.pack(pady=5, ipadx=10, ipady=8)

        # Botón
        ttk.Button(frame, text="CALCULAR", command=self.calcular, style="Calc.TButton").pack(pady=25, ipadx=20, ipady=10)

        # Resultado
        self.lbl_resultado = ttk.Label(frame, text="Resultado: -", font=("Arial", 14, "bold"), foreground="#27ae60")
        self.lbl_resultado.pack(pady=20)

    def calcular(self):
        try:
            num1 = float(self.entry_num1.get())
            num2 = float(self.entry_num2.get())
            op = self.combo_op.get()

            if op == "Sumar": result = num1 + num2
            elif op == "Restar": result = num1 - num2
            elif op == "Multiplicar": result = num1 * num2
            elif op == "Dividir":
                if num2 == 0:
                    messagebox.showerror("Error", "División por cero no permitida.")
                    return
                result = num1 / num2

            self.lbl_resultado.config(text=f"Resultado: {result:.4f}")
        except ValueError:
            messagebox.showerror("Error", "Ingresa números válidos.")
        except Exception as e:
            messagebox.showerror("Error", str(e))

```

```python
    # ----------------------------------
    # PESTAÑA 2: REPAIR CENTER
    # ----------------------------------
    def crear_pestana_pedidos(self):
        frame = ttk.Frame(self.notebook)
        self.notebook.add(frame, text="  Repair Center  ")

        # Título
        ttk.Label(frame, text="ADMINISTRAR PEDIDOS DE SERVICIO", font=("Arial", 16, "bold"),
                  foreground="#2c3e50").pack(pady=15)

        # Formulario
        form = ttk.Frame(frame)
        form.pack(pady=10, padx=20, fill="x")

        # Cliente
        ttk.Label(form, text="Nombre:").grid(row=0, column=0, sticky="w", pady=5, padx=5)
        self.entry_nombre = ttk.Entry(form, width=30)
        self.entry_nombre.grid(row=0, column=1, pady=5, padx=5)

        ttk.Label(form, text="Apellido:").grid(row=1, column=0, sticky="w", pady=5, padx=5)
        self.entry_apellido = ttk.Entry(form, width=30)
        self.entry_apellido.grid(row=1, column=1, pady=5, padx=5)

        # Dirección
        ttk.Label(form, text="Calle:").grid(row=2, column=0, sticky="w", pady=5, padx=5)
        self.entry_calle = ttk.Entry(form, width=30)
        self.entry_calle.grid(row=2, column=1, pady=5, padx=5)

        ttk.Label(form, text="Altura:").grid(row=3, column=0, sticky="w", pady=5, padx=5)
        self.entry_altura = ttk.Entry(form, width=15)
        self.entry_altura.grid(row=3, column=1, sticky="w", pady=5, padx=5)

        # Inconveniente
        ttk.Label(form, text="Inconveniente:").grid(row=4, column=0, sticky="w", pady=5, padx=5)
        self.entry_inconveniente = tk.Text(form, height=3, width=35)
        self.entry_inconveniente.grid(row=4, column=1, pady=5, padx=5)

        # Técnico
        ttk.Label(form, text="Técnico:").grid(row=5, column=0, sticky="w", pady=5, padx=5)
        self.combo_tecnico = ttk.Combobox(form, state="readonly", width=27)
        self.combo_tecnico.grid(row=5, column=1, pady=5, padx=5)
        self.cargar_tecnicos()

        # Fecha y hora
        ttk.Label(form, text="Fecha (YYYY-MM-DD):").grid(row=6, column=0, sticky="w", pady=5, padx=5)
        self.entry_fecha = ttk.Entry(form, width=15)
        self.entry_fecha.grid(row=6, column=1, sticky="w", pady=5, padx=5)

        ttk.Label(form, text="Hora (HH:MM):").grid(row=7, column=0, sticky="w", pady=5, padx=5)
        self.entry_hora = ttk.Entry(form, width=10)
        self.entry_hora.grid(row=7, column=1, sticky="w", pady=5, padx=5)

```

```
166
167         # Botones
168         btn_frame = ttk.Frame(form)
169         btn_frame.grid(row=8, column=0, columnspan=2, pady=20)
170
171         ttk.Button(btn_frame, text="GUARDAR PEDIDO", command=self.guardar_pedido).pack(side="left", padx=10)
172         ttk.Button(btn_frame, text="VER PEDIDOS", command=self.ver_pedidos).pack(side="left", padx=10)
173         ttk.Button(btn_frame, text="LIMPIAR", command=self.limpiar_form).pack(side="left", padx=10)
174
175         # Tabla de pedidos
176         self.tree = ttk.Treeview(frame, columns=("ID", "Cliente", "Dirección", "Técnico", "Fecha", "Hora"), show="headings", height=10)
177         self.tree.pack(pady=20, padx=20, fill="both", expand=True)
178
179         self.tree.heading("ID", text="ID")
180         self.tree.heading("Cliente", text="Cliente")
181         self.tree.heading("Dirección", text="Dirección")
182         self.tree.heading("Técnico", text="Técnico")
183         self.tree.heading("Fecha", text="Fecha")
184         self.tree.heading("Hora", text="Hora")
185
186         self.tree.column("ID", width=50, anchor="center")
187         self.tree.column("Cliente", width=150)
188         self.tree.column("Dirección", width=150)
189         self.tree.column("Técnico", width=120)
190         self.tree.column("Fecha", width=100, anchor="center")
191         self.tree.column("Hora", width=80, anchor="center")
192
193         self.cargar_pedidos()
194
195     def cargar_tecnicos(self):
196         conn = sqlite3.connect('repair_center.db')
197         c = conn.cursor()
198         c.execute("SELECT nombre FROM tecnicos")
199         tecnicos = [row[0] for row in c.fetchall()]
200         self.combo_tecnico['values'] = tecnicos
201         if tecnicos:
202             self.combo_tecnico.set(tecnicos[0])
203         conn.close()
204
205     def validar_fecha(self, fecha):
206         return re.match(r"^\d{4}-\d{2}-\d{2}$", fecha) is not None
207
208     def validar_hora(self, hora):
209         return re.match(r"^\d{2}:\d{2}$", hora) is not None
210
```

```
    def guardar_pedido(self):
        nombre = self.entry_nombre.get().strip()
        apellido = self.entry_apellido.get().strip()
        calle = self.entry_calle.get().strip()
        altura = self.entry_altura.get().strip()
        inconveniente = self.entry_inconveniente.get("1.0", "end-1c").strip()
        tecnico = self.combo_tecnico.get()
        fecha = self.entry_fecha.get().strip()
        hora = self.entry_hora.get().strip()

        if not all([nombre, apellido, calle, altura, inconveniente, tecnico, fecha, hora]):
            messagebox.showerror("Error", "Todos los campos son obligatorios.")
            return

        if not self.validar_fecha(fecha):
            messagebox.showerror("Error", "Fecha inválida. Usa: AAAA-MM-DD")
            return

        if not self.validar_hora(hora):
            messagebox.showerror("Error", "Hora inválida. Usa: HH:MM")
            return

        try:
            conn = sqlite3.connect('repair_center.db')
            c = conn.cursor()
            c.execute('''
                INSERT INTO pedidos (nombre, apellido, calle, altura, inconveniente, tecnico, fecha, hora, fecha_registro)
                VALUES (?, ?, ?, ?, ?, ?, ?, ?, ?)
            ''', (nombre, apellido, calle, altura, inconveniente, tecnico, fecha, hora, datetime.now().strftime("%Y-%m-%d %H:%M")))
            conn.commit()
            conn.close()

            messagebox.showinfo("Éxito", "Pedido guardado correctamente.")
            self.limpiar_form()
            self.cargar_pedidos()
        except Exception as e:
            messagebox.showerror("Error", f"No se pudo guardar: {e}")

    def cargar_pedidos(self):
        for row in self.tree.get_children():
            self.tree.delete(row)

        conn = sqlite3.connect('repair_center.db')
        c = conn.cursor()
        c.execute('SELECT id, nombre, apellido, calle, altura, tecnico, fecha, hora FROM pedidos ORDER BY fecha_registro DESC')
        for row in c.fetchall():
            cliente = f"{row[1]} {row[2]}"
            direccion = f"{row[3]} {row[4]}"
            self.tree.insert("", "end", values=(row[0], cliente, direccion, row[5], row[6], row[7]))
        conn.close()

    def ver_pedidos(self):
        self.cargar_pedidos()
```

```
    def limpiar_form(self):
        self.entry_nombre.delete(0, "end")
        self.entry_apellido.delete(0, "end")
        self.entry_calle.delete(0, "end")
        self.entry_altura.delete(0, "end")
        self.entry_inconveniente.delete("1.0", "end")
        self.entry_fecha.delete(0, "end")
        self.entry_hora.delete(0, "end")
        self.cargar_tecnicos()

# ----------------------------------------
# EJECUTAR APLICACIÓN
# ----------------------------------------
if __name__ == "__main__":
    root = tk.Tk()
    app = RepairCenterApp(root)
    root.mainloop()
```


Repair Center - Sistema Integrado
Calculadora
Repair Center
CALCULADORA BÁSICA
Primer número:
12
Segundo número:
3
Operación:
Sumar
CALCULAR
Resultado: 15.0000

Lic. Oemig José Luis.