Actividad 1. Cuestionario de conceptos clave Tkinter

1. ¿Qué es Tkinter y por qué es tan importante?
   - a) Una biblioteca para desarrollo web.
   - b) Una interfaz estándar de Python para crear aplicaciones gráficas (GUI).
   - c) Un lenguaje de programación alternativo a Python.
   - d) Una herramienta para análisis de datos.

2. ¿Cómo se instala Tkinter en Python?
   - a) Con el comando pip install tkinter.
   - b) No es necesario instalarlo, ya viene incluido en la instalación estándar de Python.
   - c) Descargándolo desde la página oficial de Tkinter.
   - d) Usando conda install tk.

3. ¿Cuál es la estructura básica de una aplicación Tkinter?
   - a) Importar Tkinter → Crear widgets → Configurar ventana → Iniciar bucle principal.
   - b) Solo necesitamos crear widgets sin una ventana principal.
   - c) Usar tkinter.run() directamente.
   - d) Definir clases sin usar objetos.

4. ¿Qué son los widgets en Tkinter y cómo se usan?
   - a) Son herramientas de depuración.
   - b) Elementos gráficos (botones, etiquetas, cuadros de texto) que permiten interactuar con el usuario.
   - c) Librerías externas para diseño web.
   - d) Funciones matemáticas en Python.

5. ¿Qué comando se usa para configurar la ventana principal en Tkinter?
   - a) window.setup()
   - b) root = tk.Tk()
   - c) Tkinter.create_window()
   - d) main_window.init()

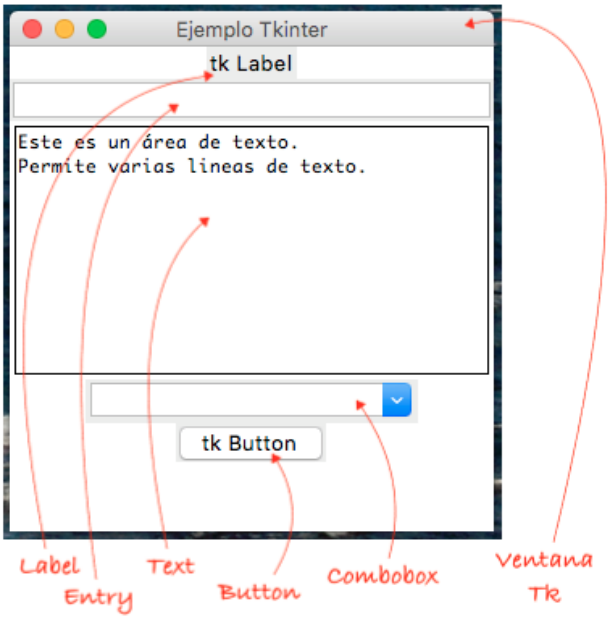
6. ¿Para qué sirve el método mainloop() en Tkinter?
   - a) Para cerrar la aplicación inmediatamente.
   - b) Para ejecutar un bucle que mantiene la ventana abierta y escucha eventos.
   - c) Para importar widgets adicionales.
   - d) Para ocultar la ventana principal.

7. ¿Cuál de estos es un widget de Tkinter?
   - a) tk.Button
   - b) tk.Database
   - c) tk.Cloud
   - d) tk.Algorithm

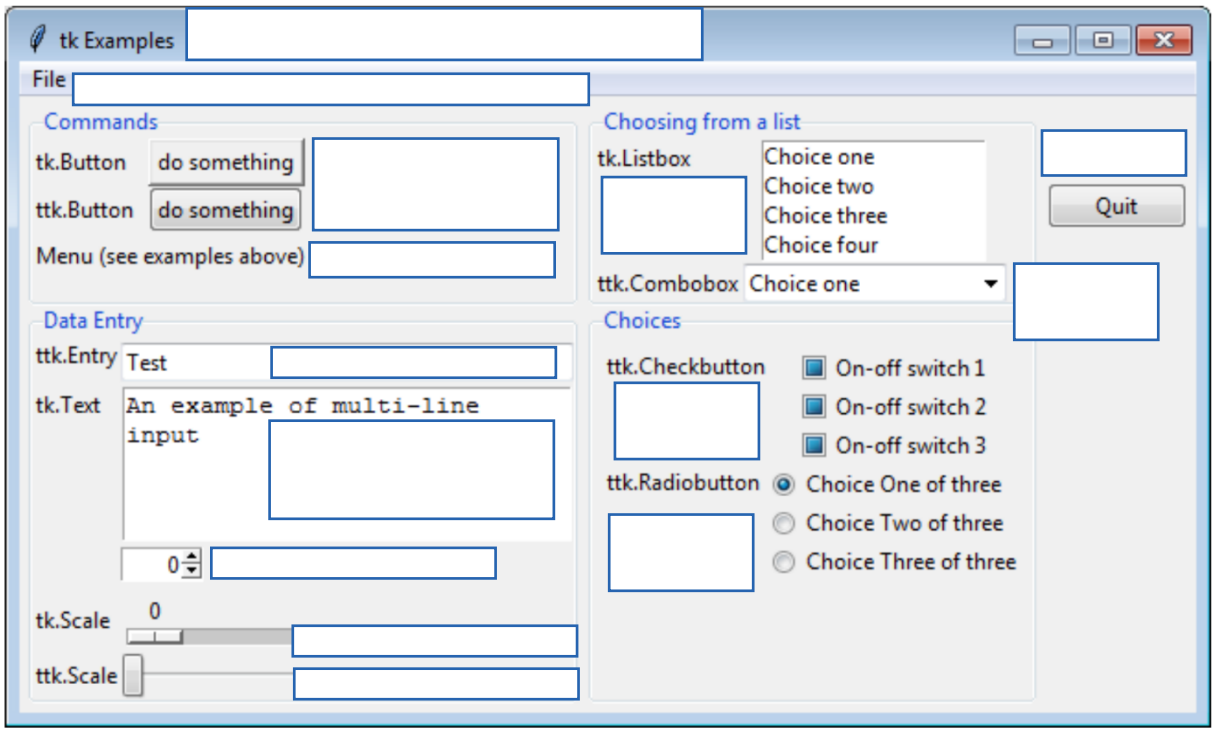
8. 10. Menciona 2 widgets comunes en Tkinter y su función.

Actividad 2.1. Identifica los componentes.

Los componentes son aquellos que se pueden colocar o ubicar en un contenedor como la ventana principal. Todos los componentes deben declararse y ubicarse para ser vistos y utilizados.



Ejemplo de ventana y algunos componentes.



Actividad 2.2. Mapa conceptual .

1. **¿Qué es Tkinter?** Biblioteca estándar de Python para crear interfaces gráficas (GUI).
    - Fácil de usar y preinstalada en Python.
    - Multiplataforma (Windows, macOS, Linux).
    - Ideal para aplicaciones simples y rápidas.
2. **Instalación y Configuración**
    - Python 3.x (Tkinter viene incluido).
    - Importar: `import tkinter as tk  # Convención común`

3. **Estructura Básica de una App Tkinter**

```python
import tkinter as tk

# 1. Ventana principal
root = tk.Tk()
root.title("Mi App")
root.geometry("400x300")

# 2. Widgets (ejemplo: Label)
label = tk.Label(root, text="¡Hola, Tkinter!")
label.pack()

# 3. Bucle principal
root.mainloop()
```

4. **Componentes Principales**
    - Widgets:
        - Elementos visuales/interactivos.
            - Label: Muestra texto.
            - Button: Botón clickeable (command= para acciones).
            - Entry: Campo de texto.
            - Frame: Contenedor de otros widgets.
        - Métodos de Posicionamiento:
            - .pack(): Apila widgets.
            - .grid(): Organiza en filas/columnas.
            - .place(): Posición absoluta (x, y).
        - Ventanas y Diálogos:
            - Tk(): Ventana principal.
            - Toplevel(): Ventanas secundarias.
            - messagebox: Diálogos de alerta.

4. Manejo de eventos (clics)

- Usando el parámetro command en botones o el método .bind().

Ejemplo con Button:

```
def saludar():
    print("Hola")

boton = tk.Button(root, text="Saludar", command=saludar)
boton.pack()
```

Eventos con .bind():

```
def tecla_presionada(event):
    print(f"Tecla: {event.char}")

root.bind("<Key>", tecla_presionada)
```

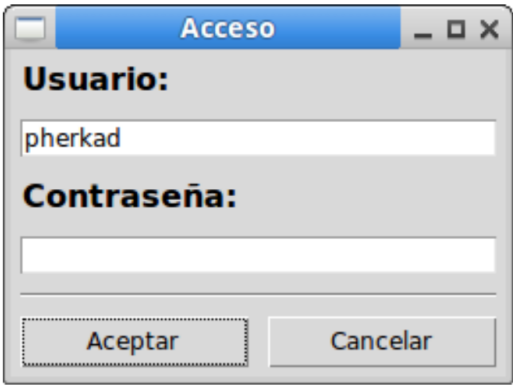
PEGA AQUÍ TU MAPA

Gestores de Distribución (Geometry Managers) en Tkinter

Controlan cómo se **organizan los widgets** dentro de un **contenedor** (ventana, frame, panel, etc.).

Tipos de Gestores

1. pack()



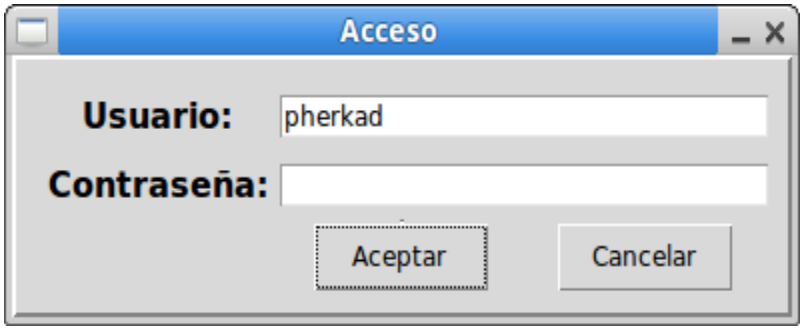
- **Función**: Organiza widgets en bloques (vertical u horizontalmente).
- **Ventajas**: Sencillo para diseños lineales y expande widgets para llenar espacio disponible.
- **Desventajas**: Limitado para diseños complejos.

Ejemplo:

```
label = tk.Label(root, text="¡Hola!")
label.pack(side="top", fill="x", padx=10, pady=5)  # side: top, bottom, left, right
```

El layout pack() divide el contenedor en cuatro zonas **TOP, BOTTOM, LEFT y RIGHT**, zona que se asigna utilizando la propiedad **«side».** Este gestor permite indicar que los componente acompañe los cambios de tamaño del contenedor utilizando la propiedad **«fill»** con tres posibles valores: BOTH (horizontal y vertical), **X (horizontal) y Y (vertical)**, y para que se active esta propiedad, es necesario utilizar «expand» en verdadero. Por último se puede definir el espaciado horizontal y vertical con las propiedades **«padx» y «pady»** indicando de forma numérica el espaciado en pixeles.

2. grid()



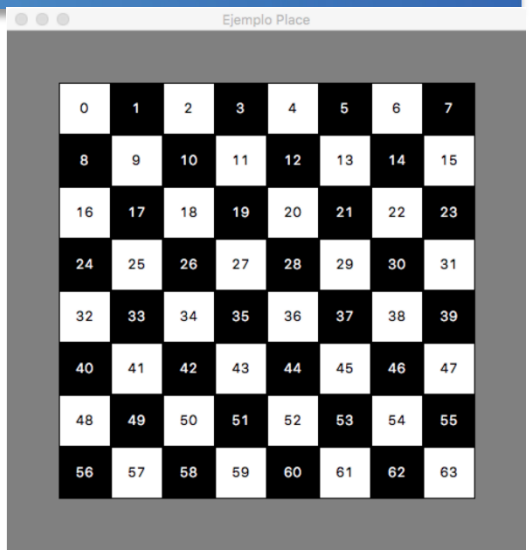
- **Función**: Distribuye widgets en una cuadrícula (filas y columnas).
- **Ventajas**: Ideal para formularios o layouts estructurados y permite combinar celdas (rowspan, columnspan).
- **Desventajas**: No funciona junto con pack() en el mismo contenedor.

Ejemplo:

```
boton = tk.Button(root, text="Aceptar")
boton.grid(row=0, column=0, sticky="ew")  # sticky: n, s, e, w (alineación)
```

El layout grid, divide el contenedor en **filas y columnas**, permitiendo acomodar los componentes indicando la fila y la columna y la cantidad de columnas que va a ocupar dicho componente. Este gestor, también admite adaptarse a las modificaciones de tamaño del contenedor.

3. place()



- **Función**: Posiciona widgets en coordenadas absolutas o relativas.
- **Ventajas**: Máximo control sobre ubicación y tamaño.
- Desventajas: Poco adaptable a cambios de tamaño de ventana.

Ejemplo:

```
entry = tk.Entry(root)
entry.place(x=50, y=20, width=100, height=30)  # (x, y) desde esquina superior izquierda
```

El gestor place(), requiere que se indique pa posición en x y la posición en y en pixeles, y también el width (ancho) y height (alto) del componente.

Comparativa

Gestor	Uso Típico	Flexibilidad	Adaptabilidad	Complejidad
pack()	Diseños simples (menús, barras)	Media	Alta	Baja
grid()	Formularios, tablas	Alta	Media	Media
place()	Posicionamiento exacto (ej: juegos)	Baja	Baja	Alta

Reglas Importantes

1. No mezclar gestores en un mismo contenedor: Si usas pack() en un Frame, no puedes usar grid() en el mismo Frame, puedes usar diferentes gestores en contenedores anidados (ej: grid() en un Frame dentro de una ventana con pack()).
   - Sticky en grid(): Controla cómo se expande el widget en su celda (ej: sticky="nsew" para llenar todo el espacio).
2. Elige **pack()** para diseños simples y rápidos.
3. Prefiere **grid()** para estructuras complejas (formularios, rejillas).
4. Usa **place()** solo cuando necesites control pixel-perfect (ej: gráficos interactivos).

Actividad 2.3 . Realiza tablas comparativas.

	pack()	grid()	place()

Widgets en Tkinter: Elementos Visuales e Interactivos

Los widgets son los componentes básicos de una interfaz gráfica en Tkinter. Permiten mostrar información, recibir entrada del usuario o ejecutar acciones.

📌 Tipos de Widgets Comunes

Widget	Descripción	Ejemplo de Uso	Dibuja un ejemplo
Label	Muestra texto o imágenes.	Label(root, text="Hola")	
Button	Botón que ejecuta una acción al hacer clic.	Button(root, text="Aceptar", command=funcion)	
Entry	Campo de texto de una línea.	Entry(root, width=30)	
Text	Área de texto multilínea (con formato).	Text(root, height=5)	
Frame	Contenedor para agrupar otros widgets.	Frame(root, bg="gray")	
Checkbutton	Casilla de verificación (ON/OFF).	Checkbutton(root, text="Acepto términos")	
Radiobutton	Botón de opción (selección única).	Radiobutton(root, text="Opción 1", value=1)	
Listbox	Lista de elementos seleccionables.	Listbox(root, height=4)	
Combobox	Menú desplegable (requiere ttk).	ttk.Combobox(root, values=["A", "B"])	
Scale	Barra deslizante (ej: volumen).	Scale(root, from_=0, to=100)	
Menu			
tkk.Menubutton			
tk.OptionMenu			
tkk.Separator			
tkk.Progressbar			
ttk.Treeview			

```python
import tkinter as tk

root = tk.Tk()

# Widget Label
etiqueta = tk.Label(root, text="Nombre:")
etiqueta.grid(row=0, column=0)  # Posición en grid

# Widget Entry
entrada = tk.Entry(root)
entrada.grid(row=0, column=1)

# Widget Button
def saludar():
    print(f"¡Hola, {entrada.get()}!")

boton = tk.Button(root, text="Saludar", command=saludar)
boton.grid(row=1, columnspan=2)  # Ocupa 2 columnas

root.mainloop()
```

Ejemplo con Label y Button

Cómo Usar Widgets

1. Creación y Posicionamiento

Todo widget requiere:

1. Contenedor padre (root, Frame, etc.).
2. Método de geometría:
   - pack(),
   - grid(),
   - o place().

Personalización de Widgets

Puedes modificar su apariencia con:

- **Colores**: fg, bg, activebackground.
- **Fuentes**: font=("Arial", 12).
- **Bordes**: relief="sunken" (otros: flat, raised, groove).
- **Imágenes**: PhotoImage (para Button, Label).

Ejemplo:
Botón con Estilo

```python
boton = tk.Button(
    root,
    text="Clic aquí",
    bg="#4CAF50",       # Color de fondo
    fg="white",         # Color del texto
    font=("Arial", 12, "bold"),
    relief="raised",    # Borde 3D
    padx=10, pady=5     # Espaciado interno
)
boton.pack()
```

1. Colores en Tkinter

Los colores se definen en los widgets usando los parámetros:

- **fg o foreground**: Color del texto.
- **bg o background**: Color de fondo.
- **activebackground**: Color de fondo al interactuar (ej: botón presionado).

Formas de Especificar Colores

- Nombres de colores predefinidos: lista de colores válidos: "white", "blue", "yellow", etc.

```python
label = tk.Label(root, text="Hola", fg="red", bg="black")
```

- Códigos hexadecimales:

```python
boton = tk.Button(root, text="Aceptar", bg="#FF5733", fg="#FFFFFF")
```

- RGB (en formato #RRGGBB):

```python
frame = tk.Frame(root, bg="#4A235A")  # Morado oscuro
```

2. Fuentes en Tkinter

Se pueden personalizar las fuentes usando el parámetro font en widgets como Label, Button o Entry.

Sintaxis Básica

```
widget = tk.Widget(root, font=("Familia", Tamaño, "Estilo"))
```

Ejemplo:

```
label = tk.Label(root, text="Texto personalizado", font=("Arial", 12, "bold italic"))
```

Opciones de Fuente

Parámetro	Valores posibles
Familia	"Arial", "Times New Roman", "Courier", etc.
Tamaño	12, 14, 16, ... (en puntos)
Estilo	"bold", "italic", "underline", o combinaciones como "bold italic"

Usar Fuentes Personalizadas (.ttf)

Si necesitas una fuente externa (ej: Roboto.ttf):

1. Asegúrate de que esté instalada en el sistema.
2. Usa el nombre exacto de la familia en Tkinter:

```
label = tk.Label(root, text="Fuente Custom", font=("Roboto", 10))
```

3. Imágenes en Tkinter

Para usar imágenes en widgets como Label o Button, Tkinter requiere que se carguen en un formato compatible (PNG, GIF, PPM).

Soporte de Formatos

- **PhotoImage**: Para imágenes .gif, .png, .ppm.
- **PIL (Pillow)**: Para formatos como .jpg o .bmp (requiere instalación).

Ejemplo con PhotoImage (imagen PNG/GIF)

```
from tkinter import PhotoImage

imagen = PhotoImage(file="ruta/a/la/imagen.png")  # Cargar imagen
label_imagen = tk.Label(root, image=imagen)
label_imagen.pack()
```

Nota: Si la imagen no se muestra, asegúrate de mantener una referencia global (ej: global imagen).

Ejemplo con Pillow (para JPG/otros formatos)

1. Instala Pillow: `pip install pillow`
2. Carga la imagen:

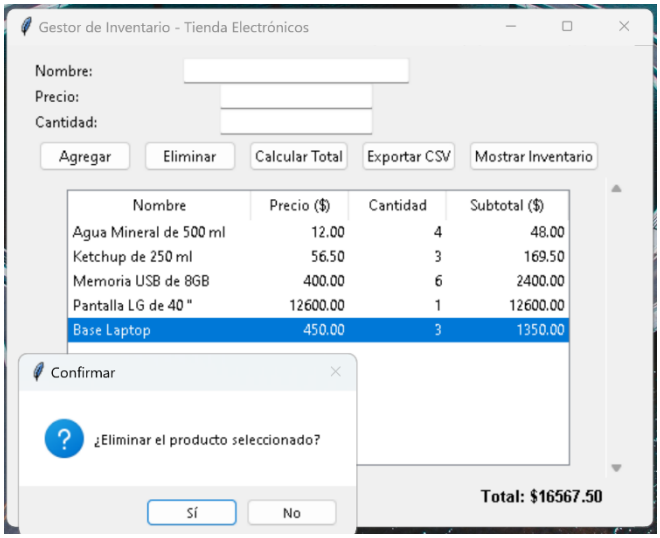
```
from PIL import Image, ImageTk

imagen_pil = Image.open("imagen.jpg")  # Abrir con PIL
imagen_tk = ImageTk.PhotoImage(imagen_pil)  # Convertir a formato Tkinter

label = tk.Label(root, image=imagen_tk)
label.pack()
```

Actividad 3. Exposición:
¿Qué es un (widget seleccionado) y para qué sirve?
¿Propiedades principales?
¿Ejemplo práctico?

Pega aquí tu exposición

Actividad 4. Análisis de un caso de uso real de GUIs.

Sistema de Gestión de Inventario para una Pequeña Tienda



Una tienda de electrónicos necesita un sistema simple para:

1. Registrar productos (nombre, precio, cantidad).
2. Agrega/eliminar/Calcular/Exportar/Mostrar.
3. Exportar el inventario a un archivo CSV.

¿Por qué Tkinter?

- Es ligero, no requiere instalación adicional (viene con Python).
- Perfecto para una app local y de bajo mantenimiento.

casopractico.py > ...

```python
import tkinter as tk
from tkinter import ttk, messagebox, filedialog
import csv
import os

class InventarioApp:
    def __init__(self, root):
        self.root = root
        self.root.title("Gestor de Inventario - Tienda Electrónicos")
        self.productos = []  # Lista para almacenar productos

        # Configuración inicial
        self.archivo_inventario = "inventario.csv"

        # Widgets principales
        self.frame = ttk.Frame(root)
        self.frame.pack(padx=10, pady=10)

        self._crear_formulario()
        self._crear_tabla()
        self._crear_botones()

        # Cargar inventario al iniciar
        self._cargar_inventario()

    def _crear_formulario(self):
        # Etiquetas y campos de entrada
        ttk.Label(self.frame, text="Nombre:").grid(row=0, column=0, sticky="w")
        self.nombre_entry = ttk.Entry(self.frame, width=30)
        self.nombre_entry.grid(row=0, column=1)

        ttk.Label(self.frame, text="Precio:").grid(row=1, column=0, sticky="w")
        self.precio_entry = ttk.Entry(self.frame)
        self.precio_entry.grid(row=1, column=1)
        ttk.Label(self.frame, text="Cantidad:").grid(row=2, column=0, sticky="w")
        self.cantidad_entry = ttk.Entry(self.frame)
        self.cantidad_entry.grid(row=2, column=1)

    def _crear_tabla(self):
        # Treeview (tabla)
        self.tabla = ttk.Treeview(
            self.frame,
            columns=("Nombre", "Precio", "Cantidad", "Subtotal"),
            show="headings",
            selectmode='browse'  # Permite selección de items
        )
```

```
        self.tabla.heading("Nombre", text="Nombre")
        self.tabla.heading("Precio", text="Precio ($)")
        self.tabla.heading("Cantidad", text="Cantidad")
        self.tabla.heading("Subtotal", text="Subtotal ($)")
        self.tabla.grid(row=4, columnspan=5, pady=10)

        # Configurar ancho de columnas
        self.tabla.column("Nombre", width=150)
        self.tabla.column("Precio", width=80, anchor='e')
        self.tabla.column("Cantidad", width=80, anchor='e')
        self.tabla.column("Subtotal", width=100, anchor='e')

        # Scrollbar para la tabla
        scrollbar = ttk.Scrollbar(self.frame, orient="vertical", command=self.tabla.yview)
        scrollbar.grid(row=4, column=5, sticky='ns')
        self.tabla.configure(yscrollcommand=scrollbar.set)

        # Etiqueta para mostrar el total
        self.total_label = ttk.Label(self.frame, text="Total: $0.00", font=('Arial', 10, 'bold'))
        self.total_label.grid(row=5, column=0, columnspan=5, pady=5, sticky='e')

    def _crear_botones(self):
        # Frame para botones
        botones_frame = ttk.Frame(self.frame)
        botones_frame.grid(row=3, column=0, columnspan=5, pady=5)

        # Botones de acciones
        ttk.Button(
            botones_frame,
            text="Agregar",
            command=self._agregar_producto
        ).pack(side='left', padx=5)
        ttk.Button(
            botones_frame,
            text="Eliminar",
            command=self._eliminar_producto
        ).pack(side='left', padx=5)

        ttk.Button(
            botones_frame,
            text="Calcular Total",
            command=self._calcular_total
        ).pack(side='left', padx=5)

        ttk.Button(
            botones_frame,
            text="Exportar CSV",
            command=self._exportar_csv
        ).pack(side='left', padx=5)

        ttk.Button(
            botones_frame,
            text="Mostrar Inventario",
            command=self._mostrar_inventario
        ).pack(side='left', padx=5)

    def _agregar_producto(self):
        # Validación


        if not nombre:
            messagebox.showerror("Error", "El nombre no puede estar vacío")
            return

        try:
            precio = float(precio)
            cantidad = int(cantidad)
            if precio <= 0 or cantidad < 0:
                raise ValueError
        except ValueError:
            messagebox.showerror("Error", "Precio debe ser positivo y cantidad entera no negativa")
            return

        # Calcular subtotal
        subtotal = precio * cantidad

        # Agregar a la lista y tabla
        self.productos.append((nombre, precio, cantidad, subtotal))
        self.tabla.insert("", "end", values=(nombre, f"{precio:.2f}", cantidad, f"{subtotal:.2f}"))

        # Limpiar campos
        self.nombre_entry.delete(0, tk.END)
        self.precio_entry.delete(0, tk.END)
        self.cantidad_entry.delete(0, tk.END)

        # Actualizar total
        self._calcular_total()
        # Guardar automáticamente
        self._guardar_inventario()
```

```
    def _eliminar_producto(self):
        # Obtener item seleccionado
        seleccionado = self.tabla.selection()
        if not seleccionado:
            messagebox.showwarning("Advertencia", "Seleccione un producto para eliminar")
            return

        # Confirmar eliminación
        if not messagebox.askyesno("Confirmar", "¿Eliminar el producto seleccionado?"):
            return

        # Eliminar de la tabla
        item = self.tabla.item(seleccionado)
        valores = item['values']
        self.tabla.delete(seleccionado)

        # Eliminar de la lista de productos
        for i, producto in enumerate(self.productos):
            if (producto[0] == valores[0] and
                float(producto[1]) == float(valores[1]) and
                producto[2] == int(valores[2])):
                del self.productos[i]
                break

        # Actualizar total y guardar cambios
        self._calcular_total()
        self._guardar_inventario()

    def _calcular_total(self):
        total = sum(producto[3] for producto in self.productos)
        self.total_label.config(text=f"Total: ${total:.2f}")
        return total
    def _exportar_csv(self):
        if not self.productos:
            messagebox.showwarning("Advertencia", "No hay datos para exportar")
            return

        # Calcular total antes de exportar
        total = self._calcular_total()

        archivo = filedialog.asksaveasfilename(
            defaultextension=".csv",
            filetypes=[("CSV Files", "*.csv")],
            initialfile="inventario_exportado.csv"
        )
        if archivo:
            try:
                with open(archivo, "w", newline="", encoding='utf-8') as f:
                    writer = csv.writer(f)
                    writer.writerow(["Nombre", "Precio", "Cantidad", "Subtotal"])
                    writer.writerows(self.productos)
                    writer.writerow(["", "", "TOTAL:", f"{total:.2f}"])
                messagebox.showinfo("Éxito", f"Inventario exportado a:\n{archivo}")
            except Exception as e:
                messagebox.showerror("Error", f"No se pudo exportar:\n{str(e)}")

    def _mostrar_inventario(self):
        # Limpiar tabla primero
        for item in self.tabla.get_children():
            self.tabla.delete(item)

        # Mostrar todos los productos
        for producto in self.productos:
            self.tabla.insert("", "end", values=(
                producto[0],
                f"{producto[1]:.2f}",
                producto[2],
                f"{producto[3]:.2f}"
            ))

        # Actualizar total
        self._calcular_total()
        messagebox.showinfo("Inventario", f"Mostrando {len(self.productos)} productos")
```

```python
    def _cargar_inventario(self):
        if not os.path.exists(self.archivo_inventario):
            return

        try:
            with open(self.archivo_inventario, "r", newline="", encoding='utf-8') as f:
                reader = csv.reader(f)
                next(reader)  # Saltar encabezado
                for row in reader:
                    if len(row) >= 3:  # Acepta filas con o sin subtotal
                        nombre, precio, cantidad = row[:3]
                        try:
                            precio = float(precio)
                            cantidad = int(cantidad)
                            subtotal = precio * cantidad
                            self.productos.append((nombre, precio, cantidad, subtotal))
                        except ValueError:
                            continue
            self._mostrar_inventario()
        except Exception as e:
            messagebox.showerror("Error", f"No se pudo cargar el inventario:\n{str(e)}")

    def _guardar_inventario(self):
        try:
            with open(self.archivo_inventario, "w", newline="", encoding='utf-8') as f:
                writer = csv.writer(f)
                writer.writerow(["Nombre", "Precio", "Cantidad", "Subtotal"])
                writer.writerows(self.productos)
        except Exception as e:
            messagebox.showerror("Error", f"No se pudo guardar el inventario:\n{str(e)}")

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

Preguntas sobre Conceptos de Tkinter en el Código del Sistema de Inventario

1. Conceptos Básicos de Tkinter

1. ¿Qué función cumple tk.Tk() en este programa?
2. ¿Para qué sirve el método pack() y en qué parte del código se utiliza?
3. ¿Qué diferencia hay entre tk.Label y ttk.Label?
4. ¿Por qué es importante llamar a mainloop() al final del programa?

2. Estructura de la Interfaz

5. ¿Cómo están organizados los widgets en la ventana principal?
6. ¿Qué ventaja tiene usar ttk.Frame para agrupar elementos?
7. ¿Por qué se usan tanto grid() como pack() en el código?
8. ¿Qué hace el parámetro columnspan en el grid()?

3. Funcionamiento del Treeview

9. ¿Qué representa el ttk.Treeview en esta aplicación?

10. ¿Para qué sirve el parámetro show="headings"?

11. ¿Cómo se añaden elementos a la tabla con insert()?

12. ¿Qué función cumple el ttk.Scrollbar adjunto a la tabla?

4. Manejo de Datos

13. ¿Cómo se almacenan los productos en memoria?

14. ¿Por qué se usa una lista de tuplas para self.productos?

15. ¿Qué contiene cada tupla dentro de self.productos?
16. ¿Cómo se calcula el subtotal de cada producto?

5. Validación de Entradas

17. ¿Qué validaciones se hacen antes de agregar un producto?

18. ¿Por qué se usa isdigit() para validar el precio y cantidad?

19. ¿Qué problema podría ocurrir si no validáramos los datos de entrada?

6. Botones y Funcionalidades

20. ¿Qué hace el botón "Calcular Total"?

21. ¿Cómo funciona el proceso de eliminar un producto?

22. ¿Qué ocurre cuando se presiona "Mostrar Inventario"?

23. ¿Qué información se guarda en el archivo CSV?

7. Persistencia de Datos

24. ¿Cómo se cargan los datos al iniciar la aplicación?

25. ¿Qué hace el método _guardar_inventario()?

26. ¿Por qué se usa newline="" al abrir el archivo CSV?

27. ¿Qué pasa si el archivo CSV no existe al iniciar el programa?

8. Manejo de Errores

28. ¿Dónde se usa try-except en el código y para qué?

29. ¿Qué tipos de errores se están manejando?

30. ¿Cómo se muestran los mensajes de error al usuario?