

The background is a dense, overlapping collage of various Tkinter GUI components. It includes window frames with titles like 'tk', 'Ventana Principal', and 'Ejemplo de paquete'. There are buttons labeled 'Nuevo', 'Abrir', 'Guardar', 'Botón 1', and 'Hacer Clic'. Menus are visible with options 'Opción 1' and 'Opción 2'. Text labels include 'Hola, Mundo!', 'HOLA MUNDO!', '¡Bienvenido a la Ventana Principal!', 'Fila 1, Columna 1 y 2', 'Fila 2, Columna 1', 'Fila 0, Columna 1', 'Etiqueta 1', 'Etiqueta 2', 'Este es una etiqueta', and 'Acepto los términos'. There are also checkboxes and a 'Clic Aquí' button.

Tkinter en Python

Desarrollando Interfaces Gráficas de Usuario

Programación Avanzada - Prof. Felipe Morales

Comisión 1 y 2 - Avalos Maia - Lucca Pérez Veltri - Euler Diego





TEMARIO



01

Tkinter:
Introducción

02

Características
y Ventajas de
Tkinter

03

Características
Avanzadas de
Tkinter

04

Estructura
Básica de una
GUI en
Tkinter

05

Implementación
Librería
Tkinter

06

Interfaz Gráfica:
P. Gestión de
Inventario

07

[EJEMPLO]

08

[EJEMPLO]

09

[EJEMPLO]

10

[EJEMPLO]

11

[EJEMPLO]

1. Tkinter: Introducción

Definición de Tkinter:

- **Tkinter es un marco de interfaz gráfica integrado en la biblioteca estándar de Python.**
- **Actúa como puente entre Python y la librería TCL/TK.**
- **Es multiplataforma: los elementos se representan utilizando componentes nativos de cada sistema operativo.**

Verificación de la Instalación:

- **Podemos verificar la instalación de Tkinter ejecutando la siguiente línea en la consola:**

```
python -m tkinter
```


2. Características y Ventajas de Tkinter:

Características Clave:

- **Multiplataforma:** Funciona en diferentes sistemas operativos utilizando componentes nativos.
- **Facilidad de Uso:** Comparado con otros marcos, es fácil de aprender y usar.
- **Ligero:** Es un marco liviano, ideal para desarrollar aplicaciones GUI rápidas y funcionales.
- **Biblioteca Estándar:** No requiere instalaciones adicionales, ya que está incluida en Python

Ventajas de Usar Tkinter:

- **Documentación y Comunidad:** Amplia documentación y una comunidad activa para soporte.

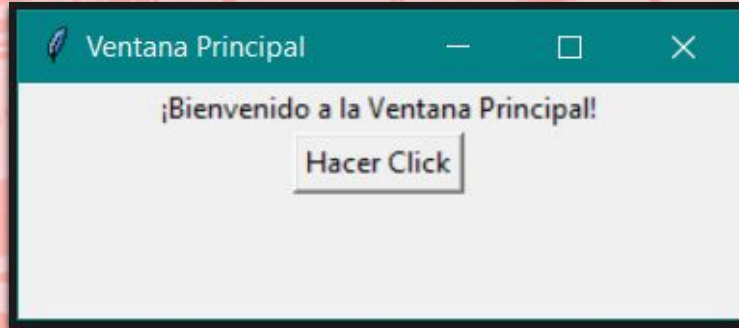
3. Características Avanzadas de Tkinter:

- **Widget Personalizables:** Gran variedad de widgets (botones, etiquetas, cuadros de texto, etc.) personalizables.
- **Soporte para Eventos:** Manejo de eventos como clics de ratón, teclas presionadas, etc.
- **Diseño de Layout Flexible:** Uso de gestores de geometría como pack, grid y place para organizar los widgets.
- **Extensibilidad:** Posibilidad de crear widgets personalizados y extender funcionalidades.
- **Soporte para Temas:** Permite cambiar el aspecto de la interfaz utilizando diferentes temas.

4. Estructura Básica de una GUI en Tkinter

4.1 La Ventana

Es el elemento fundamental de una GUI.



Contenedor Principal: Todas las interacciones y componentes de la GUI se agrupan dentro de la ventana.

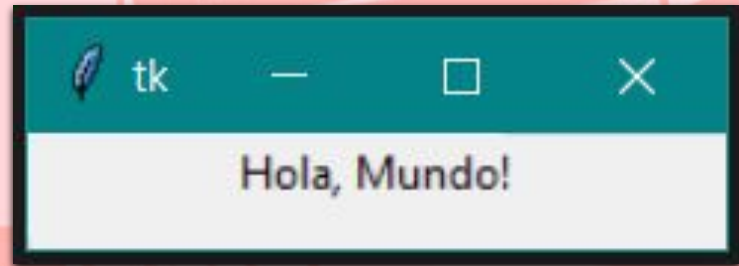
Widgets: Los elementos dentro de la ventana, como cuadros de texto, etiquetas y botones, son denominados widgets.

Gestión de Elementos: Permite organizar y gestionar la disposición de los widgets mediante diferentes gestores de geometría (pack, grid, place).

4.2 Pasos Básicos para Crear una Aplicación

1. Importar Tkinter
2. Crear una Ventana Principal
3. Añadir Widgets a la Ventana
4. Iniciar el Bucle de Eventos

```
import tkinter as tk  
  
root = tk.Tk()  
label = tk.Label(root, text="Hola, Mundo!")  
label.pack()  
root.mainloop()
```



5. Implementación Librería Tkinter

5.1 Proyecto Gestión de Inventario

- ❖ Este proyecto combina la funcionalidad de un sistema de gestión de inventario con una interfaz gráfica intuitiva desarrollada con Tkinter en Python.
- ❖ Permite a los usuarios administrar eficientemente productos, cantidades y notificaciones de inventario bajo una estructura modularizada en tres archivos principales: **Main.gui**, **Inventario.py** y **Gui.py**.
- ❖ Cada archivo contribuye de manera específica al funcionamiento y la interfaz visual del sistema, facilitando así la administración y manipulación de datos de inventario de manera efectiva.

5.2 Archivo Main.gui

```
import tkinter as tk
from gui import InventoryApp

if __name__ == "__main__":
    root = tk.Tk() # Inicia la ventana principal de la aplicación
    app = InventoryApp(root) # Crea una instancia de la aplicación de
    inventario
    app.pack() # Empaqueta la aplicación en la ventana principal
    root.mainloop() # Inicia el bucle principal de la interfaz gráfica
```

- ❖ **Inicialización del Programa:** Este archivo actúa como el punto de entrada de la aplicación, donde se inicia la interfaz gráfica de usuario utilizando Tkinter.
- ❖ **Configuración de la Ventana Principal:** Configura la ventana principal de la aplicación con un título y las dimensiones adecuadas.
- ❖ **Instanciación de InventoryApp:** Crea una instancia de la clase InventoryApp, que contiene la lógica y la interfaz del sistema de inventario.
- ❖ **Ejecución del Mainloop de Tkinter:** Llama a root.mainloop() para iniciar el bucle principal de eventos de Tkinter, permitiendo que la ventana permanezca abierta y responda a las interacciones del usuario.

5.3 Archivo Inventario.py - Bloque 1: Inicialización y Carga del Inventario

Contiene la lógica de negocio para gestionar el inventario, incluyendo métodos para agregar, modificar, eliminar, buscar y mostrar productos.

- ❖ **Carga de Datos:** Gestiona la carga de datos del inventario de un archivo CSV .

```
import csv
```

```
class Inventory:
```

```
    def __init__(self, filename="inventory.csv"):
        self.filename = filename
        self.inventory = {}
        self.load_inventory()
```

```
    # Carga el inventario desde el archivo CSV al inicializar la clase
```

```
    def load_inventory(self):
```

```
        try:
```

```
            with open(self.filename, mode='r') as file:
```

```
                reader = csv.reader(file)
```

```
                for row in reader:
```

```
                    name, quantity = row
```

```
                    self.inventory[name] = int(quantity)
```

```
            except FileNotFoundError:
```

```
                # Maneja el caso donde el archivo no existe aún (por ejemplo, en la primera ejecución)
```

```
                pass
```

5.4 Archivo Inventario.py - Bloque 2: Guardar Inventario

```
# Guarda el inventario en el archivo CSV
def save_inventory(self):
    with open(self.filename, mode='w', newline='') as file:
        writer = csv.writer(file)
        for name, quantity in self.inventory.items():
            writer.writerow([name, quantity])
```

- ❖ **Guardado de Datos:** Gestiona la carga y el guardado de datos del inventario en un archivo CSV, asegurando persistencia de datos entre sesiones.

5.5 Archivo Inventario.py - Bloque 3: Operaciones de Producto

- ❖ **Agregar Productos:** Permite añadir nuevos productos al inventario, aumentando la cantidad si el producto ya existe.
- ❖ **Modificar Cantidades:** Permite modificar la cantidad de productos existentes en el inventario, actualizando el archivo CSV.
- ❖ **Eliminar Productos:** Permite eliminar productos del inventario y guarda los cambios en el archivo CSV.

Permite añadir productos al inventario

```
def add_product(self, name, quantity):  
    if name in self.inventory:  
        self.inventory[name] += quantity  
    else:  
        self.inventory[name] = quantity  
    self.save_inventory()
```

Permite eliminar productos del inventario

```
def remove_product(self, name):  
    if name in self.inventory:  
        del self.inventory[name]  
    self.save_inventory()
```

Permite modificar la cantidad de un producto existente en el inventario

```
def modify_quantity(self, name, new_quantity):  
    if name in self.inventory:  
        self.inventory[name] = new_quantity  
    self.save_inventory()
```


5.6 Archivo Inventario.py - Bloque 4: Consulta y División del Inventario

```
# Devuelve el inventario actual como un diccionario
```

```
def get_inventory(self):  
    return self.inventory
```

```
# Divide el inventario en dos listas: una para productos con 5 o más  
# unidades y otra para productos con menos de 5 unidades
```

```
def split_inventory(self):  
    above_five = []  
    below_five = []  
    for name, quantity in self.inventory.items():  
        if quantity >= 5:  
            above_five.append((name, quantity))  
        else:  
            below_five.append((name, quantity))  
    return above_five, below_five
```

❖ **Consulta de Inventario:**
Proporciona métodos para obtener el inventario actual, facilitando la consulta de productos y cantidades.

❖ **Manejo de Errores:** Gestiona situaciones en las que el archivo CSV no existe, creando uno nuevo si es necesario, y maneja correctamente la conversión de datos.



5.7 Archivo Gui.py - Bloque 1: Inicialización

- ❖ **Define la interfaz de usuario utilizando Tkinter y maneja los eventos de la interfaz.**

```
import tkinter as tk
from tkinter import messagebox, simpledialog
from tkinter import ttk
from inventory import Inventory

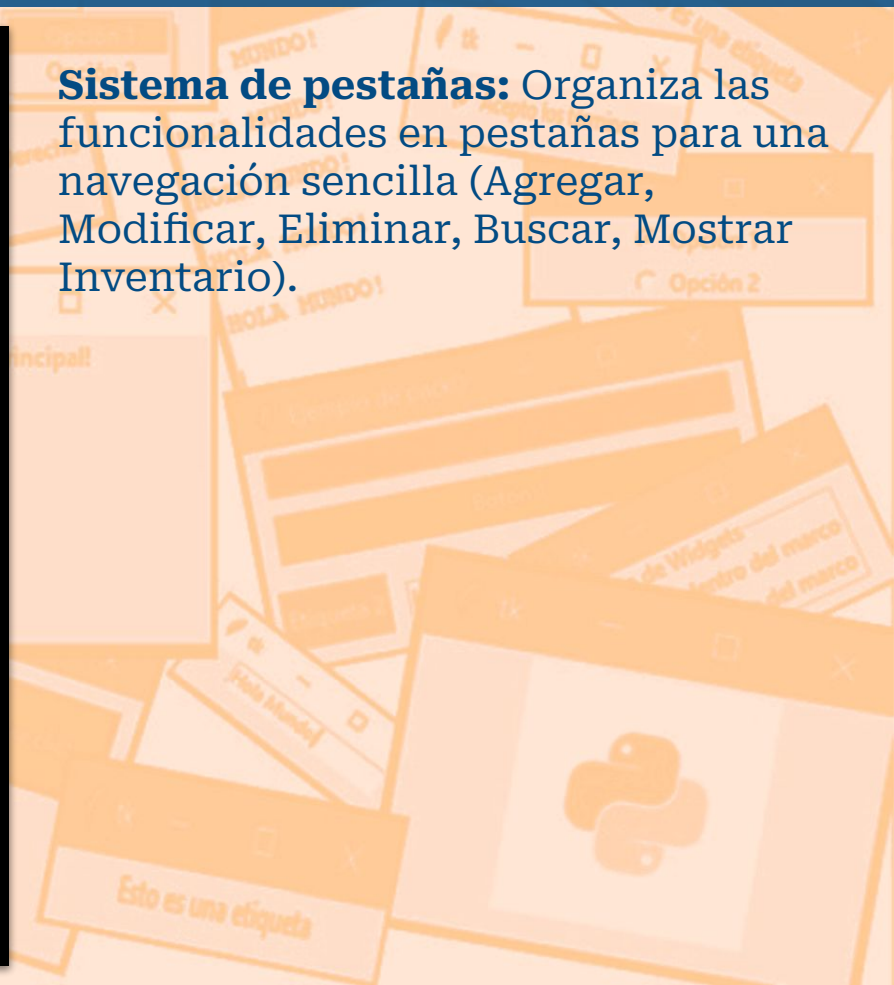
class InventoryApp(tk.Frame):
    def __init__(self, master):
        super().__init__(master)
        self.master = master
        self.master.title("Sistema de Inventario")
        self.inventory = Inventory()
        self.create_widgets()
```

5.8 Archivo Gui.py - Bloque 2: Creación de Pestañas

Crear las pestañas y los widgets

```
def create_widgets(self):  
    self.notebook = ttk.Notebook(self.master)  
    self.notebook.pack(padx=10, pady=10, fill=tk.BOTH, expand=True)  
  
    self.tab_add = tk.Frame(self.notebook)  
    self.notebook.add(self.tab_add, text="Agregar Producto")  
    self.create_add_widgets()  
  
    self.tab_modify = tk.Frame(self.notebook)  
    self.notebook.add(self.tab_modify, text="Modificar Producto")  
    self.create_modify_widgets()  
  
    self.tab_remove = tk.Frame(self.notebook)  
    self.notebook.add(self.tab_remove, text="Eliminar Producto")  
    self.create_remove_widgets()  
  
    self.tab_search = tk.Frame(self.notebook)  
    self.notebook.add(self.tab_search, text="Buscar Producto")  
    self.create_search_widgets()  
  
    self.tab_show = tk.Frame(self.notebook)  
    self.notebook.add(self.tab_show, text="Mostrar Inventario")  
    self.create_show_widgets()
```

Sistema de pestañas: Organiza las funcionalidades en pestañas para una navegación sencilla (Agregar, Modificar, Eliminar, Buscar, Mostrar Inventario).



5.9 Archivo Gui.py - Bloque 3: Widgets para Agregar Productos

Gestión de productos: Permite agregar nuevos productos al inventario con su nombre y cantidad.

```
# Widgets para agregar productos
def create_add_widgets(self):
    self.label_name_add = tk.Label(self.tab_add, text="Nombre del
Producto:")
    self.label_name_add.pack(pady=5)
    self.entry_name_add = tk.Entry(self.tab_add)
    self.entry_name_add.pack(pady=5)

    self.label_quantity_add = tk.Label(self.tab_add,
text="Cantidad:")
    self.label_quantity_add.pack(pady=5)
    self.entry_quantity_add = tk.Entry(self.tab_add)
    self.entry_quantity_add.pack(pady=5)

    self.button_add = tk.Button(self.tab_add, text="Agregar
Producto", command=self.add_product)
    self.button_add.pack(pady=10)
```


5.10 Archivo Gui.py - Bloque 4: Función para Agregar Productos al Inventario

Función para agregar productos al inventario

```
def add_product(self):  
    name = self.entry_name_add.get()  
    quantity = self.entry_quantity_add.get()  
    if name and quantity:  
        try:  
            quantity = int(quantity)  
            self.inventory.add_product(name, quantity)  
            self.check_inventory_alert(name, quantity)  
            messagebox.showinfo("Éxito", f"Producto '{name}' agregado al inventario.")  
            self.entry_name_add.delete(0, tk.END)  
            self.entry_quantity_add.delete(0, tk.END)  
        except ValueError:  
            messagebox.showerror("Error", "La cantidad debe ser un número entero.")  
    else:  
        messagebox.showerror("Error", "Por favor ingrese el nombre y la cantidad del producto.")
```

5.11 Archivo Gui.py - Bloque 5: Widgets para Modificar Productos

Modificación de productos: Facilita la actualización de la cantidad de productos existentes.

```
# Widgets para modificar productos
def create_modify_widgets(self):
    self.label_name_modify = tk.Label(self.tab_modify,
    text="Nombre del Producto:")
    self.label_name_modify.pack(pady=5)
    self.entry_name_modify = tk.Entry(self.tab_modify)
    self.entry_name_modify.pack(pady=5)

    self.label_quantity_modify = tk.Label(self.tab_modify,
    text="Nueva Cantidad:")
    self.label_quantity_modify.pack(pady=5)
    self.entry_quantity_modify = tk.Entry(self.tab_modify)
    self.entry_quantity_modify.pack(pady=5)

    self.button_modify = tk.Button(self.tab_modify, text="Modificar
    Cantidad", command=self.modify_quantity)
    self.button_modify.pack(pady=10)
```

5.12 Archivo Gui.py - Bloque 6: Función para Modificar Cantidad de Productos en el Inventario

```
# Función para modificar la cantidad de productos en el inventario
def modify_quantity(self):
    name = self.entry_name_modify.get()
    quantity = self.entry_quantity_modify.get()
    if name and quantity:
        try:
            quantity = int(quantity)
            self.inventory.modify_quantity(name, quantity)
            self.check_inventory_alert(name, quantity)
            messagebox.showinfo("Éxito", f"Cantidad de '{name}' modificada correctamente.")
            self.entry_name_modify.delete(0, tk.END)
            self.entry_quantity_modify.delete(0, tk.END)
        except ValueError:
            messagebox.showerror("Error", "La cantidad debe ser un número entero.")
    else:
        messagebox.showerror("Error", "Por favor ingrese el nombre y la nueva cantidad del producto.")
```

5.13 Archivo Gui.py - Bloque 7: Widgets y Función para Eliminar Productos

Eliminación de productos: Proporciona una manera de eliminar productos del inventario.

```
# Widgets para eliminar productos
def create_remove_widgets(self):
    self.label_name_remove = tk.Label(self.tab_remove, text="Nombre
del Producto:")
    self.label_name_remove.pack(pady=5)
    self.entry_name_remove = tk.Entry(self.tab_remove)
    self.entry_name_remove.pack(pady=5)

    self.button_remove = tk.Button(self.tab_remove, text="Eliminar
Producto", command=self.remove_product)
    self.button_remove.pack(pady=10)

# Función para eliminar productos del inventario
def remove_product(self):
    name = self.entry_name_remove.get()
    if name:
        if name in self.inventory.get_inventory():
            self.inventory.remove_product(name)
            messagebox.showinfo("Éxito", f"Producto '{name}' eliminado del
inventario.")
        else:
            messagebox.showerror("Error", f"No se encontró el producto
'{name}' en el inventario.")
    else:
        messagebox.showerror("Error", "Por favor ingrese el nombre del
producto a eliminar.")
```


5.14 Archivo Gui.py - Bloque 8: Widgets y Funciones para Buscar Inventario

```
# Widgets para buscar productos
def create_search_widgets(self):
    self.label_search = tk.Label(self.tab_search, text="Buscar Producto:")
    self.label_search.pack(pady=5)
    self.entry_search = tk.Entry(self.tab_search)
    self.entry_search.pack(pady=5)

    self.button_search = tk.Button(self.tab_search, text="Buscar",
    command=self.search_product)
    self.button_search.pack(pady=10)

# Función para buscar productos en el inventario
def search_product(self):
    search_term = self.entry_search.get()
    if search_term:
        if search_term in self.inventory.get_inventory():
            quantity = self.inventory.get_inventory()[search_term]
            messagebox.showinfo("Producto Encontrado", f"{search_term}: {quantity}")
        else:
            messagebox.showinfo("Producto no encontrado", f"No se encontró el producto '{search_term}' en el inventario.")
    else:
        messagebox.showerror("Error", "Por favor ingrese un término de búsqueda.")
```

Búsqueda de productos: Permite buscar productos por nombre y muestra su cantidad si están en el inventario.

5.15 Archivo Gui.py - Bloque 9: Widgets y Funciones para Mostrar Inventario

Visualización del inventario:

Muestra el inventario completo, separando los productos con menos de cinco unidades de los que tienen cinco o más.

Mensajes emergentes: Utiliza messagebox para mostrar mensajes de éxito, error y advertencia al usuario.

Alertas de baja cantidad: Emite alertas cuando la cantidad de un producto es menor que cinco unidades.

```
# Widgets para mostrar el inventario
def create_show_widgets(self):
    self.button_show = tk.Button(self.tab_show, text="Mostrar Inventario",
                                command=self.show_inventory)
    self.button_show.pack(pady=10)

# Función para mostrar el inventario completo
def show_inventory(self):
    above_five, below_five = self.inventory.split_inventory()

    if above_five or below_five:
        message = ""

        if above_five:
            message += "Productos con 5 o más unidades:\n"
            message += "\n".join(f"{name}: {quantity}" for name, quantity in above_five)
            message += "\n\n"

        if below_five:
            message += "Productos con menos de 5 unidades:\n"
            message += "\n".join(f"{name}: {quantity}" for name, quantity in below_five)

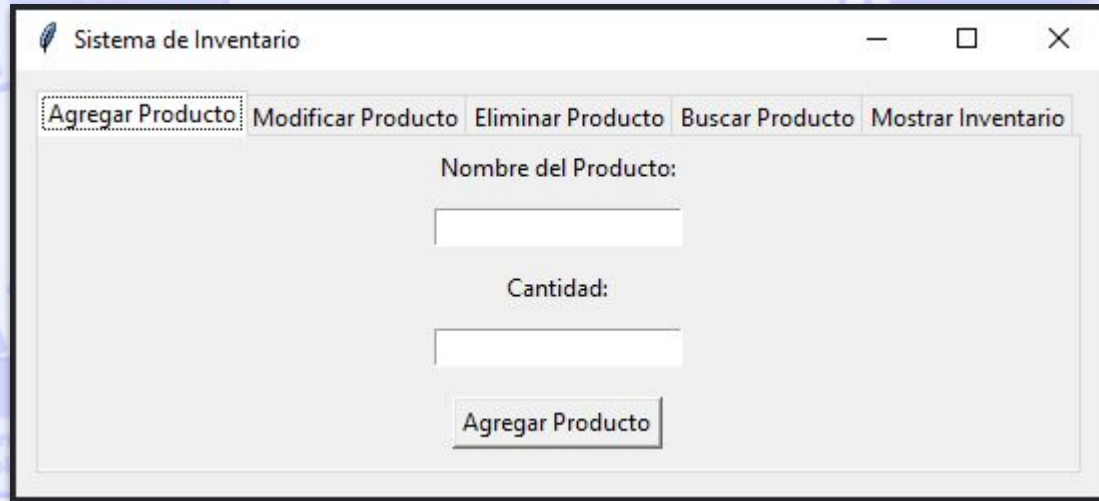
        messagebox.showinfo("Inventario", message)
    else:
        messagebox.showinfo("Inventario", "El inventario está vacío.")

# Verifica si la cantidad del producto es menor de 5 para mostrar una alerta
def check_inventory_alert(self, name, quantity):
    if quantity < 5:
        messagebox.showwarning("Alerta de Inventario", f"¡Alerta! Quedan menos de 5 unidades de '{name}'.")
```

6. Interfaz Gráfica: Funcionalidades P. Gestión de Inventario

6.1 Agregar Producto:

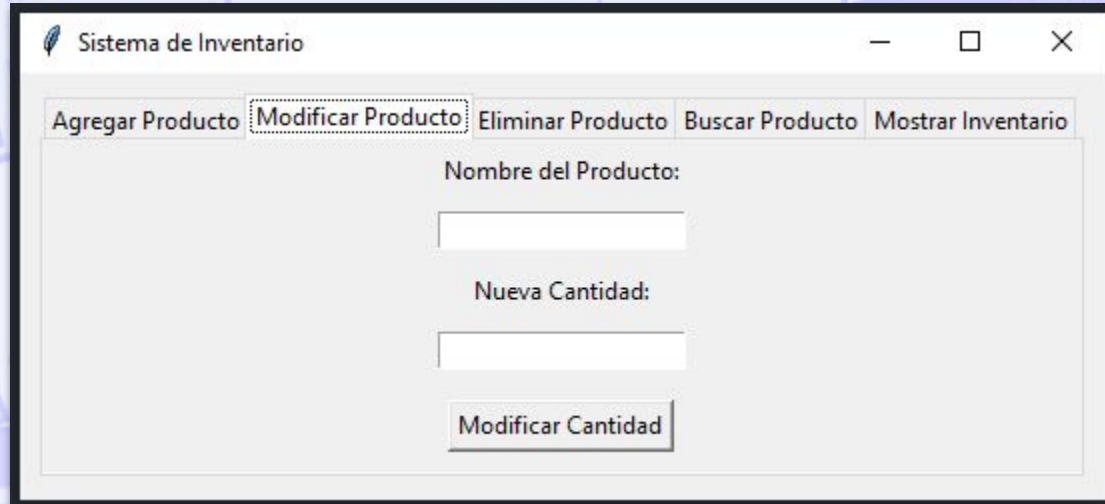
- ❖ Permite añadir un nuevo producto al inventario especificando su nombre y cantidad.
- ❖ Valida que la cantidad ingresada sea un número entero y actualiza el inventario almacenando los datos en un archivo CSV.



The screenshot shows a window titled "Sistema de Inventario" with standard Windows window controls (minimize, maximize, close). Inside the window, there is a horizontal menu bar with five buttons: "Agregar Producto", "Modificar Producto", "Eliminar Producto", "Buscar Producto", and "Mostrar Inventario". The "Agregar Producto" button is currently selected and highlighted with a dashed border. Below the menu bar, the form contains two labels: "Nombre del Producto:" and "Cantidad:". Each label is followed by a text input field. At the bottom of the form, there is a button labeled "Agregar Producto". The background of the slide features a collage of various UI elements, including buttons labeled "Opción 1", "Opción 2", "Opción 3", and "Opción 4", as well as a Python logo.

6.2 Modificar Cantidad:

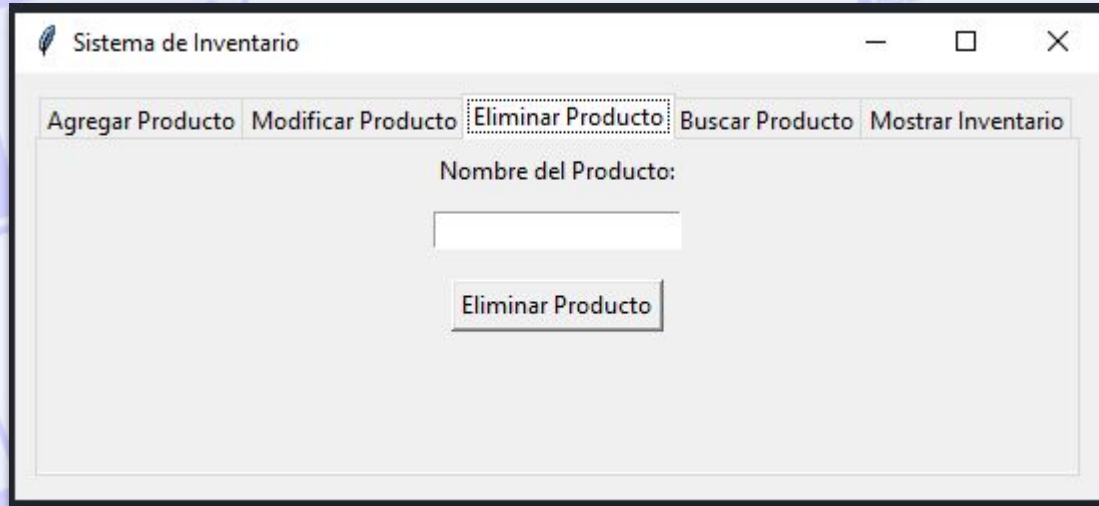
- ❖ Permite cambiar la cantidad de unidades disponibles de un producto existente en el inventario.
- ❖ Verifica que la nueva cantidad sea un número entero y actualiza el inventario con la nueva información.



The screenshot shows a window titled "Sistema de Inventario" with standard Windows window controls. Inside the window, there is a horizontal menu bar with five buttons: "Agregar Producto", "Modificar Producto" (which is currently selected and has a dashed border), "Eliminar Producto", "Buscar Producto", and "Mostrar Inventario". Below the menu bar, the form is divided into two sections. The first section is labeled "Nombre del Producto:" and contains a single-line text input field. The second section is labeled "Nueva Cantidad:" and contains a single-line text input field. At the bottom of the form, there is a button labeled "Modificar Cantidad".

6.3 Eliminar Producto:

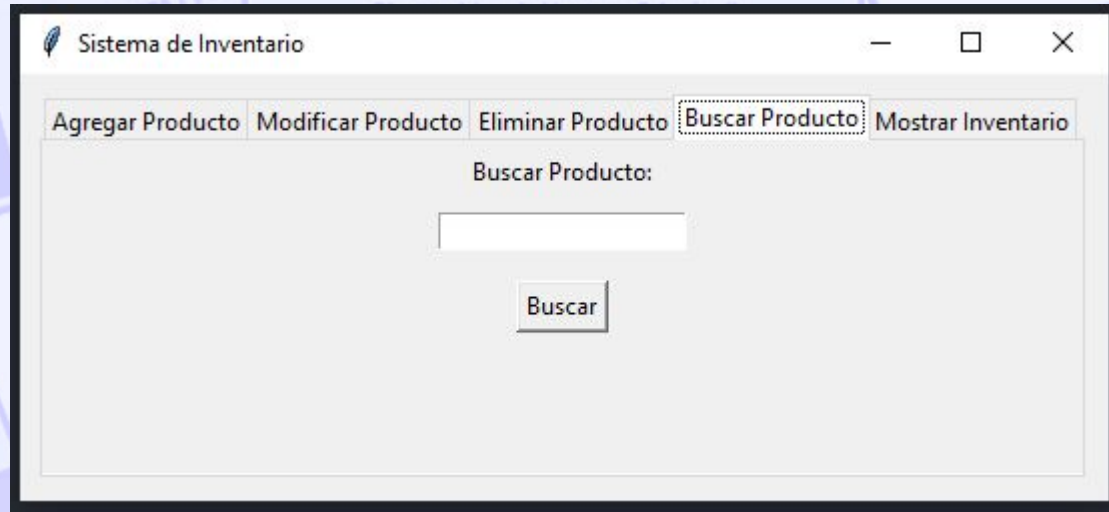
- ❖ Permite eliminar un producto del inventario según su nombre.
- ❖ Verifica si el producto existe en el inventario antes de proceder con la eliminación.



The screenshot shows a window titled "Sistema de Inventario" with a menu bar containing five options: "Agregar Producto", "Modificar Producto", "Eliminar Producto" (which is currently selected), "Buscar Producto", and "Mostrar Inventario". Below the menu bar, there is a label "Nombre del Producto:" followed by a single-line text input field. At the bottom of the window, there is a button labeled "Eliminar Producto".

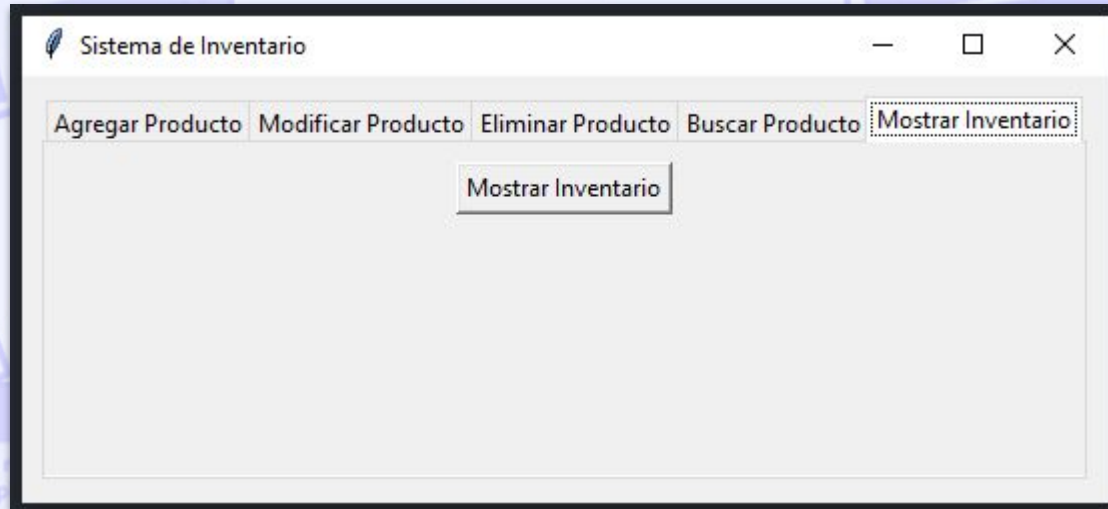
6.4 Buscar Producto:

- ❖ Permite buscar un producto en el inventario según su nombre.
- ❖ Muestra la cantidad disponible del producto si se encuentra en el inventario; de lo contrario, muestra un mensaje indicando que el producto no fue encontrado.



6.5 Mostrar Inventario:

- ❖ Muestra una vista del inventario clasificada en dos categorías:
- ❖ Productos con 5 o más unidades.
- ❖ Productos con menos de 5 unidades.
- ❖ Utiliza ventanas emergentes (messagebox) para mostrar esta información de manera clara al usuario.



Documentación y Recursos Adicionales

Recursos para Aprender Más

- * Documentación oficial: docs.python.org
- * Tutoriales: tkdocs.com
- * Ejemplos y Guías: realpython.com



Conclusión

- Tkinter es una herramienta poderosa y fácil de usar para crear GUIs en Python.
- Permite la creación de interfaces interactivas y personalizables.

¡Gracias por su atención!

