Guía Completa del Sistema de Menú - Dulce Control

Documentación Técnica y Tutorial Completo

Tabla de Contenidos

---

1. INTRODUCCIÓN

Este documento explica en detalle el funcionamiento del sistema de menú visual para el proyecto Dulce Control. El menú está construido con la librería **CustomTkinter**, que proporciona una interfaz gráfica moderna y personalizable.

Requisitos Previos

- Python 3.7 o superior
- Instalación de customtkinter: `pip install customtkinter`

Estructura del Proyecto

```
dulce_control/
├── menu.py          (archivo principal)
├── productos.py     (módulo a crear)
├── inventario.py    (módulo a crear)
├── ventas.py        (módulo a crear)
└── reportes.py      (módulo a crear)
```

---

2. IMPORTACIONES Y CONFIGURACIÓN INICIAL

Código:

```python
import customtkinter as ctk
from tkinter import messagebox
import sys
```

Explicación Detallada:

`import customtkinter as ctk`

- Importa la librería de interfaz gráfica moderna
- El `as ctk` es un alias para escribir menos código
- CustomTkinter es una versión mejorada de tkinter estándar

`from tkinter import messagebox`

- Importa los cuadros de diálogo del tkinter estándar
- Permite mostrar alertas, confirmaciones y errores
- Es compatible con customtkinter

`import sys`

- Librería del sistema de Python
- Se usa para cerrar el programa completamente con `sys.exit(0)`

Configuración Global:

```python
ctk.set_appearance_mode("dark")
ctk.set_default_color_theme("blue")
```

`set_appearance_mode("dark")`

- Establece el tema visual de la aplicación
- Opciones disponibles:
  - `"dark"` - Tema oscuro (recomendado para menos cansancio visual)
  - `"light"` - Tema claro
  - `"system"` - Sigue el tema del sistema operativo

`set_default_color_theme("blue")`

- Define el color principal de botones y elementos interactivos
- Opciones disponibles:

- `"blue"` - Azul estándar
- `"green"` - Verde
- `"dark-blue"` - Azul oscuro

---

3. ESTRUCTURA DE CLASES

¿Por Qué Usar Clases en Python?

Las clases ofrecen múltiples ventajas para organizar el código:

Ventajas:

1. **Organización**: Todo el código relacionado está en un solo lugar
2. **Reutilización**: Puedes crear múltiples instancias si lo necesitas
3. **Mantenimiento**: Es más fácil encontrar y modificar funciones específicas
4. **Encapsulación**: Los datos y funciones están contenidos en un objeto
5. **Herencia**: Puedes heredar funcionalidades de otras clases

Definición de la Clase:

```python
class MenuPrincipal(ctk.CTk):
    def __init__(self):
        super().__init__()
```

`class MenuPrincipal(ctk.CTk):`

- Define una nueva clase llamada `MenuPrincipal`
- Hereda de `ctk.CTk` (la ventana principal de customtkinter)
- Al heredar, obtiene todas las funcionalidades de una ventana

`def __init__(self):`

- Constructor de la clase
- Se ejecuta automáticamente al crear un objeto
- Es como decir: "cuando crees el menú, haz esto..."
- El parámetro `self` hace referencia a la instancia actual

`super().__init__()`

- Llama al constructor de la clase padre (`ctk.CTk`)

- Inicializa la ventana base con todas sus funcionalidades
- Siempre debe ir al principio del `__init__`

---

4. CONFIGURACIÓN DE LA VENTANA

Código:

```python
self.title("🍓 Dulce Control - Sistema de Gestión")
self.geometry("600x650")
self.resizable(False, False)
self.center_window()
```

Explicación de Cada Método:

`self.title("texto")`

- Establece el título que aparece en la barra superior de la ventana
- Acepta texto con emojis Unicode
- Ejemplo: `self.title("Mi Aplicación")`

`self.geometry("ancho x alto")`

- Define el tamaño inicial de la ventana en píxeles
- Formato: `"ancho x alto"` o `"ancho x alto + posX + posY"`
- Ejemplos:
  - `self.geometry("800x700")` - Ventana más grande
  - `self.geometry("400x500")` - Ventana más pequeña
  - `self.geometry("600x650+100+50")` - Con posición específica

`self.resizable(ancho, alto)`

- Controla si el usuario puede redimensionar la ventana
- Parámetros: `(True/False, True/False)`
- Ejemplos:
  - `self.resizable(False, False)` - No se puede redimensionar
  - `self.resizable(True, True)` - Se puede redimensionar libremente
  - `self.resizable(True, False)` - Solo ancho variable

Función para Centrar la Ventana:

```python
def center_window(self):
    self.update_idletasks()
    width = self.winfo_width()
    height = self.winfo_height()
    x = (self.winfo_screenwidth() // 2) - (width // 2)
    y = (self.winfo_screenheight() // 2) - (height // 2)
    self.geometry(f'{width}x{height}+{x}+{y}')
```

Explicación paso a paso:

1. **`self.update_idletasks()`**: Actualiza la ventana para obtener dimensiones reales
2. **`winfo_width()`** y **`winfo_height()`**: Obtienen el ancho y alto de la ventana
3. **`winfo_screenwidth()`** y **`winfo_screenheight()`**: Obtienen dimensiones de la pantalla
4. **Cálculo de x**: `(ancho_pantalla / 2) - (ancho_ventana / 2)` = posición horizontal centrada
5. **Cálculo de y**: `(alto_pantalla / 2) - (alto_ventana / 2)` = posición vertical centrada
6. **`geometry()`**: Aplica el nuevo tamaño y posición

---

5. WIDGETS Y ELEMENTOS VISUALES

5.1 Frame Principal (Contenedor)

```python
main_frame = ctk.CTkFrame(self, corner_radius=10)
main_frame.pack(fill="both", expand=True, padx=20, pady=20)
```

`CTkFrame(padre, opciones)`

- Crea un contenedor rectangular para organizar otros widgets
- Es como una "caja" donde pones elementos

Parámetros importantes:

- `corner_radius=10` - Esquinas redondeadas (0 = cuadradas, 20 = muy redondeadas)
- `fg_color="color"` - Color de fondo
- `border_width=2` - Grosor del borde
- `border_color="color"` - Color del borde

Método `.pack()` - Sistema de Posicionamiento:

- `fill="both"` - Se expande horizontal y verticalmente
  - `"x"` - Solo horizontal
  - `"y"` - Solo vertical
  - `"both"` - Ambas direcciones
- `expand=True` - Ocupa todo el espacio disponible
- `padx=20` - Margen horizontal (izquierda y derecha)
- `pady=20` - Margen vertical (arriba y abajo)
- `padx=(10, 20)` - Diferentes márgenes: 10px izquierda, 20px derecha
- `pady=(30, 10)` - 30px arriba, 10px abajo

5.2 Etiquetas de Texto (Labels)

```python
titulo = ctk.CTkLabel(
    main_frame,
    text="🍓 DULCE CONTROL",
    font=ctk.CTkFont(size=32, weight="bold")
)
titulo.pack(pady=(20, 10))
```

`CTkLabel(padre, opciones)`

- Muestra texto estático en la interfaz
- No es interactivo (solo muestra información)

Parámetros importantes:

- `text="texto"` - El texto a mostrar
- `font=CTkFont(size, weight)` - Configuración de fuente
  - `size=32` - Tamaño en puntos (prueba: 10, 16, 24, 36, 48)
  - `weight="bold"` - Negrita ("normal" para texto regular)
  - `family="Arial"` - Tipo de fuente
- `text_color="color"` - Color del texto
  - Nombres: `"red"`, `"blue"`, `"green"`, `"gray"`
  - Código hex: `"#FF5733"`, `"#3498db"`
- `anchor="center"` - Alineación del texto ("w", "e", "n", "s", "center")

5.3 Botones Interactivos

```python
btn_productos = ctk.CTkButton(
    botones_frame,
    text="📦 Gestión de Productos",
    command=self.abrir_productos,
    height=50,
    font=ctk.CTkFont(size=16),
    corner_radius=10
)
btn_productos.pack(pady=10, fill="x")
```

`CTkButton(padre, opciones)`

- Crea un botón clickeable que ejecuta una acción

Parámetros esenciales:

- `text="texto"` - Texto del botón (puede incluir emojis)
- `command=funcion` - Función a ejecutar al hacer clic
  - **IMPORTANTE**: Sin paréntesis `()`
  - Correcto: `command=self.funcion`
  - Incorrecto: `command=self.funcion()`
- `height=50` - Altura en píxeles
- `width=200` - Ancho en píxeles (opcional si usas `fill="x"`)

Parámetros de estilo:

- `fg_color="color"` - Color de fondo del botón
- `hover_color="color"` - Color al pasar el mouse
- `text_color="color"` - Color del texto
- `corner_radius=10` - Redondeo de esquinas
- `border_width=2` - Grosor del borde
- `border_color="color"` - Color del borde
- `state="normal"` - Estado del botón
  - `"normal"` - Activo y clickeable
  - `"disabled"` - Desactivado (gris, no clickeable)

Ejemplo de botón personalizado:

```python
btn_especial = ctk.CTkButton(
    frame,
    text="⭐ Botón Especial",
    command=mi_funcion,
    height=60,
    width=250,
    font=ctk.CTkFont(size=18, weight="bold"),
    fg_color="#e74c3c",        # Rojo
    hover_color="#c0392b",     # Rojo oscuro
    text_color="white",
    corner_radius=15,
    border_width=3,
    border_color="yellow"
)
```

5.4 Separadores Visuales

```python
separador = ctk.CTkFrame(botones_frame, height=2, fg_color="gray30")
separador.pack(pady=15, fill="x")
```

- Crea una línea horizontal para dividir secciones
- `height=2` - Grosor de la línea
- `fg_color="gray30"` - Color de la línea
- `fill="x"` - Se estira horizontalmente

---

6. FUNCIONES Y COMANDOS

6.1 Funciones Básicas con MessageBox

```python
def abrir_productos(self):
    messagebox.showinfo("Productos", "Módulo de Gestión de Productos")
```

Tipos de MessageBox:

`messagebox.showinfo(titulo, mensaje)`

- Muestra información general

- Ícono azul con "i"
- Un solo botón "OK"

`messagebox.showwarning(titulo, mensaje)`

- Muestra una advertencia
- Ícono amarillo con "!"
- Un solo botón "OK"

`messagebox.showerror(titulo, mensaje)`

- Muestra un error
- Ícono rojo con "X"
- Un solo botón "OK"

`messagebox.askyesno(titulo, mensaje)`

- Pregunta Sí/No
- Devuelve `True` si presiona "Sí", `False` si presiona "No"
- Útil para confirmaciones

Ejemplo de confirmación:

```python
def eliminar_producto(self):
    respuesta = messagebox.askyesno(
        "Confirmar",
        "¿Desea eliminar este producto?"
    )
    if respuesta:
        # Código para eliminar
        messagebox.showinfo("Éxito", "Producto eliminado")
    else:
        messagebox.showinfo("Cancelado", "Operación cancelada")
```

6.2 Función para Abrir Ventanas Secundarias

```python
```

```python
def abrir_ventas(self):
    # Crear nueva ventana
    ventana = ctk.CTkToplevel(self)
    ventana.title("Ventas")
    ventana.geometry("500x400")

    # Agregar contenido
    label = ctk.CTkLabel(
        ventana,
        text="Módulo de Ventas",
        font=ctk.CTkFont(size=20, weight="bold")
    )
    label.pack(pady=20)

    btn_cerrar = ctk.CTkButton(
        ventana,
        text="Cerrar",
        command=ventana.destroy
    )
    btn_cerrar.pack(pady=10)

    # Opcional: Bloquear ventana principal
    ventana.grab_set()
```

CTkToplevel(padre)

- Crea una ventana secundaria
- Se mantiene por encima de la ventana principal
- Se cierra independientemente

grab_set()

- Bloquea la interacción con otras ventanas
- El usuario debe cerrar esta ventana primero
- Útil para ventanas modales (diálogos)

6.3 Función de Salir

```python
```

```python
def salir(self):
    respuesta = messagebox.askyesno(
        "Salir",
        "¿Estás seguro que deseas salir?"
    )
    if respuesta:
        self.quit()
        self.destroy()
        sys.exit(0)
```

Métodos para cerrar:

- `self.quit()` - Detiene el mainloop
- `self.destroy()` - Destruye la ventana y libera memoria
- `sys.exit(0)` - Cierra el programa completamente (código 0 = sin errores)

Versión sin confirmación:

```python
def salir(self):
    self.quit()
    self.destroy()
    sys.exit(0)
```

6.4 Importar Módulos Externos

```python
def abrir_productos(self):
    from productos import VentanaProductos
    ventana = VentanaProductos(self)
    ventana.grab_set()
```

Pasos para conectar módulos:

1. Crea el archivo `productos.py`:

```python
```

```
import customtkinter as ctk

class VentanaProductos(ctk.CTkToplevel):
    def __init__(self, parent):
        super().__init__(parent)
        self.title("Productos")
        self.geometry("600x500")

        # Agregar contenido aquí
        label = ctk.CTkLabel(self, text="Gestión de Productos")
        label.pack(pady=20)
```

2. Importa y usa en `menu.py`:

```
python

def abrir_productos(self):
    from productos import VentanaProductos
    ventana = VentanaProductos(self)
```

---

7. EJEMPLOS PRÁCTICOS DE MODIFICACIÓN

7.1 Cambiar el Tamaño de la Ventana

```
python

# Ventana más grande
self.geometry("800x700")

# Ventana más pequeña
self.geometry("500x450")

# Ventana con posición específica
self.geometry("600x650+100+50")  # +100 pixels desde izquierda, +50 desde arriba
```

7.2 Agregar un Nuevo Botón

Paso 1: Agregar el botón en `crear_widgets()`

```
python
```

```
btn_clientes = ctk.CTkButton(
    botones_frame,
    text="👥 Gestión de Clientes",
    command=self.abrir_clientes,
    height=50,
    font=ctk.CTkFont(size=16),
    corner_radius=10,
    fg_color="#2ecc71",      # Verde
    hover_color="#27ae60"    # Verde oscuro
)
btn_clientes.pack(pady=10, fill="x")
```

Paso 2: Crear la función correspondiente

python

```
def abrir_clientes(self):
    messagebox.showinfo("Clientes", "Módulo de Gestión de Clientes")
    # O abrir una ventana:
    # from clientes import VentanaClientes
    # ventana = VentanaClientes(self)
```

7.3 Cambiar Colores del Tema

python

```
# Al inicio del archivo, después de las importaciones

# Tema claro
ctk.set_appearance_mode("light")

# Tema oscuro
ctk.set_appearance_mode("dark")

# Seguir tema del sistema
ctk.set_appearance_mode("system")

# Cambiar color principal
ctk.set_default_color_theme("green")  # Verde
ctk.set_default_color_theme("blue")   # Azul
ctk.set_default_color_theme("dark-blue")  # Azul oscuro
```

7.4 Personalizar Botones Individualmente

python

```python
# Botón rojo para acciones peligrosas
btn_eliminar = ctk.CTkButton(
    frame,
    text="🗑️ Eliminar Todo",
    command=self.eliminar_todo,
    height=50,
    fg_color="#e74c3c",
    hover_color="#c0392b"
)

# Botón verde para acciones positivas
btn_guardar = ctk.CTkButton(
    frame,
    text="💾 Guardar",
    command=self.guardar,
    height=50,
    fg_color="#27ae60",
    hover_color="#229954"
)

# Botón naranja para advertencias
btn_advertencia = ctk.CTkButton(
    frame,
    text="⚠️ Precaución",
    command=self.advertir,
    height=50,
    fg_color="#f39c12",
    hover_color="#d68910"
)

# Botón gris para opciones secundarias
btn_secundario = ctk.CTkButton(
    frame,
    text="📄 Documentación",
    command=self.abrir_docs,
    height=40,
    fg_color="gray40",
    hover_color="gray30"
)
```

7.5 Hacer Botones Más Pequeños o Grandes

python

```python
# Botón pequeño
btn_pequeno = ctk.CTkButton(
    frame,
    text="Mini",
    height=30,
    width=100,
    font=ctk.CTkFont(size=12)
)

# Botón mediano (predeterminado)
btn_mediano = ctk.CTkButton(
    frame,
    text="Mediano",
    height=50,
    font=ctk.CTkFont(size=16)
)

# Botón grande
btn_grande = ctk.CTkButton(
    frame,
    text="GRANDE",
    height=70,
    font=ctk.CTkFont(size=20, weight="bold")
)
```

7.6 Agregar un Logo o Imagen

python

```python
from PIL import Image

# En la función crear_widgets(), después del título:
try:
    logo_image = ctk.CTkImage(
        light_image=Image.open("logo.png"),
        dark_image=Image.open("logo.png"),
        size=(100, 100)  # Ancho x Alto
    )
    logo_label = ctk.CTkLabel(
        main_frame,
        image=logo_image,
        text=""  # Sin texto
    )
    logo_label.pack(pady=10)
except:
    print("No se pudo cargar el logo")
```

Requisito: Instalar Pillow con `pip install Pillow`

7.7 Desactivar/Activar Botones Dinámicamente

```python
# Desactivar un botón
btn_productos.configure(state="disabled")

# Activar un botón
btn_productos.configure(state="normal")

# Ejemplo de uso:
def procesar_datos(self):
    # Desactivar botón durante el proceso
    self.btn_procesar.configure(state="disabled")

    # Hacer algo...
    time.sleep(2)

    # Reactivar botón
    self.btn_procesar.configure(state="normal")
```

7.8 Cambiar Texto de un Botón Dinámicamente

```python
# Guardar referencia al botón como atributo de clase
self.btn_estado = ctk.CTkButton(
    frame,
    text="▶️ Iniciar",
    command=self.cambiar_estado
)

def cambiar_estado(self):
    texto_actual = self.btn_estado.cget("text")
    if "Iniciar" in texto_actual:
        self.btn_estado.configure(text="⏸️ Pausar")
    else:
        self.btn_estado.configure(text="▶️ Iniciar")
```

7.9 Crear un Menú con Pestañas (Tabview)

```python
```

```python
def crear_widgets(self):
    # Crear tabview
    tabview = ctk.CTkTabview(self)
    tabview.pack(fill="both", expand=True, padx=20, pady=20)

    # Agregar pestañas
    tab1 = tabview.add("Productos")
    tab2 = tabview.add("Ventas")
    tab3 = tabview.add("Reportes")

    # Agregar contenido a cada pestaña
    label1 = ctk.CTkLabel(tab1, text="Contenido de Productos")
    label1.pack(pady=20)

    label2 = ctk.CTkLabel(tab2, text="Contenido de Ventas")
    label2.pack(pady=20)

    label3 = ctk.CTkLabel(tab3, text="Contenido de Reportes")
    label3.pack(pady=20)
```

7.10 Agregar Campos de Entrada (Entry)

python

```python
def crear_formulario(self):
    # Label
    label_nombre = ctk.CTkLabel(self, text="Nombre del Producto:")
    label_nombre.pack(pady=5)

    # Entry (campo de texto)
    self.entry_nombre = ctk.CTkEntry(
        self,
        placeholder_text="Ingrese el nombre...",
        width=300
    )
    self.entry_nombre.pack(pady=5)

    # Botón para obtener el valor
    btn_guardar = ctk.CTkButton(
        self,
        text="Guardar",
        command=self.obtener_valor
    )
    btn_guardar.pack(pady=10)

def obtener_valor(self):
    valor = self.entry_nombre.get()
    messagebox.showinfo("Valor ingresado", f"Nombre: {valor}")
```

---

8. REFERENCIA RÁPIDA

Colores Comunes (Hex)

Color	Código Hex	Uso Recomendado
Rojo	#e74c3c	Eliminar, Cancelar
Verde	#27ae60	Guardar, Confirmar
Azul	#3498db	Información, Principal
Naranja	#f39c12	Advertencias
Morado	#9b59b6	Especial, Premium
Gris	#95a5a6	Secundario, Desactivado
Negro	#2c3e50	Texto, Fondos oscuros

Tamaños de Fuente Recomendados

- **Título principal**: 28-36 puntos
- **Subtítulo**: 16-20 puntos

- **Botones**: 14-16 puntos
- **Texto normal**: 12-14 puntos
- **Texto pequeño (footer)**: 9-11 puntos

Alturas de Botones Recomendadas

- **Botones principales**: 50px
- **Botones secundarios**: 40px
- **Botones pequeños**: 30px
- **Botones grandes/destacados**: 60-70px

Métodos Comunes de Widgets

Todos los widgets:

- `.pack()` - Posicionar
- `.configure()` - Cambiar propiedades
- `.cget("propiedad")` - Obtener valor de propiedad
- `.destroy()` - Eliminar widget

Botones:

- `.invoke()` - Simular clic
- `.configure(state="disabled")` - Desactivar
- `.configure(state="normal")` - Activar

Entry (campos de texto):

- `.get()` - Obtener texto
- `.insert(0, "texto")` - Insertar texto
- `.delete(0, "end")` - Borrar todo

Atajos de Teclado Útiles

```python
```

```python
# Vincular tecla Enter a una función
self.bind("<Return>", lambda e: self.funcion())

# Vincular Escape para cerrar
self.bind("<Escape>", lambda e: self.salir())

# Vincular Ctrl+S para guardar
self.bind("<Control-s>", lambda e: self.guardar())
```

Estructura Básica de un Módulo Nuevo

```python
import customtkinter as ctk
from tkinter import messagebox

class VentanaNueva(ctk.CTkToplevel):
    def __init__(self, parent):
        super().__init__(parent)
        self.title("Título de la Ventana")
        self.geometry("600x500")
        self.crear_widgets()

    def crear_widgets(self):
        # Agregar elementos aquí
        label = ctk.CTkLabel(self, text="Contenido")
        label.pack(pady=20)

        btn_cerrar = ctk.CTkButton(
            self,
            text="Cerrar",
            command=self.destroy
        )
        btn_cerrar.pack(pady=10)
```

CONCLUSIÓN

Este documento cubre todos los conceptos fundamentales para trabajar con el sistema de menú de Dulce Control. Con esta información, puedes:

✅ Entender cada línea de código del menú
✅ Modificar el diseño y apariencia
✅ Agregar nuevos botones y funcionalidades
✅ Crear ventanas secundarias

✅ Personalizar colores, tamaños y estilos
✅ Conectar módulos externos

Recursos Adicionales

- **Documentación oficial de CustomTkinter**: https://github.com/TomSchimansky/CustomTkinter
- **Documentación de Tkinter**: https://docs.python.org/3/library/tkinter.html
- **Galería de colores**: https://htmlcolorcodes.com/

Soporte y Actualizaciones

Para dudas o sugerencias sobre el sistema Dulce Control, contactar al desarrollador del proyecto.

Versión del Documento: 1.0
Fecha: Diciembre 2024
Proyecto: Dulce Control - Sistema de Gestión