

Esqueleto del programa – Sesión 3

Proyecto flex

Inregantes, roles y tareas - win_home.py

Gonzalo - integrante Flex - win_table.py

Diego - QA/Analista - win_canvas.py

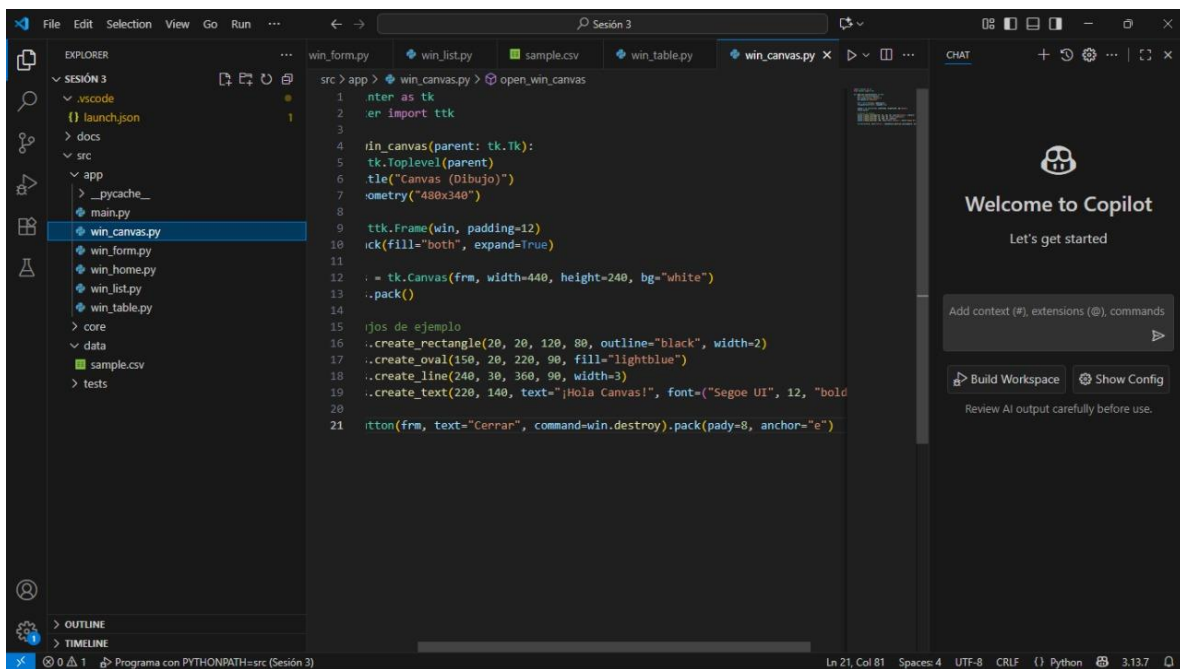
Julián - Desarrollador UI - win_form.py

Alexis - Líder Técnico - win_list.py

Sebastián - Product Owner - Creacion de repositorio y tests

Daniel - Desarrollador de Datos/API - win_home.py

Explicación del código



```
src > app > win_canvas.py > open_win_canvas
1  nter as tk
2  er import ttk
3
4  rin_canvas(parent: tk.Tk):
5      tk.Toplevel(parent)
6      tile("Canvas (Dibujo)")
7      ometry("480x340")
8
9      ttk.Frame(win, padding=12)
10     ck(fill="both", expand=True)
11
12     := tk.Canvas(frm, width=440, height=240, bg="white")
13     .pack()
14
15     rjos de ejemplo
16     ..create_rectangle(20, 20, 120, 80, outline="black", width=2)
17     ..create_oval(150, 20, 220, 90, fill="lightblue")
18     ..create_line(240, 30, 360, 90, width=3)
19     ..create_text(220, 140, text="¡Hola Canvas!", font=("Segoe UI", 12, "bold
20
21     rttion(frm, text="Cerrar", command=win.destroy).pack(pady=8, anchor="e")
```

Importaciones:

El código importa Tkinter (como tk) y el submódulo ttk para usar widgets temáticos modernos.

Definición de función:

La función principal `win_canvas(parent: tk.Tk)` recibe un `parent`, que normalmente sería la ventana principal de la aplicación.

Ventana `Toplevel`:

Se crea una ventana hija (`tk.Toplevel(parent)`) con título "Canvas (Dibujo)" y un tamaño fijo de 480x340 píxeles.

Frame de organización:

Usa un frame (`ttk.Frame`) con `padding`, que se expande para contener el canvas y el botón de cerrar.

Canvas:

El canvas (`tk.Canvas`) mide 440x240 y tiene fondo blanco, donde se dibujan elementos gráficos.

Ejemplos de dibujo:

El código incluye ejemplos para crear figuras básicas:

Un rectángulo negro (con ancho de línea 2).

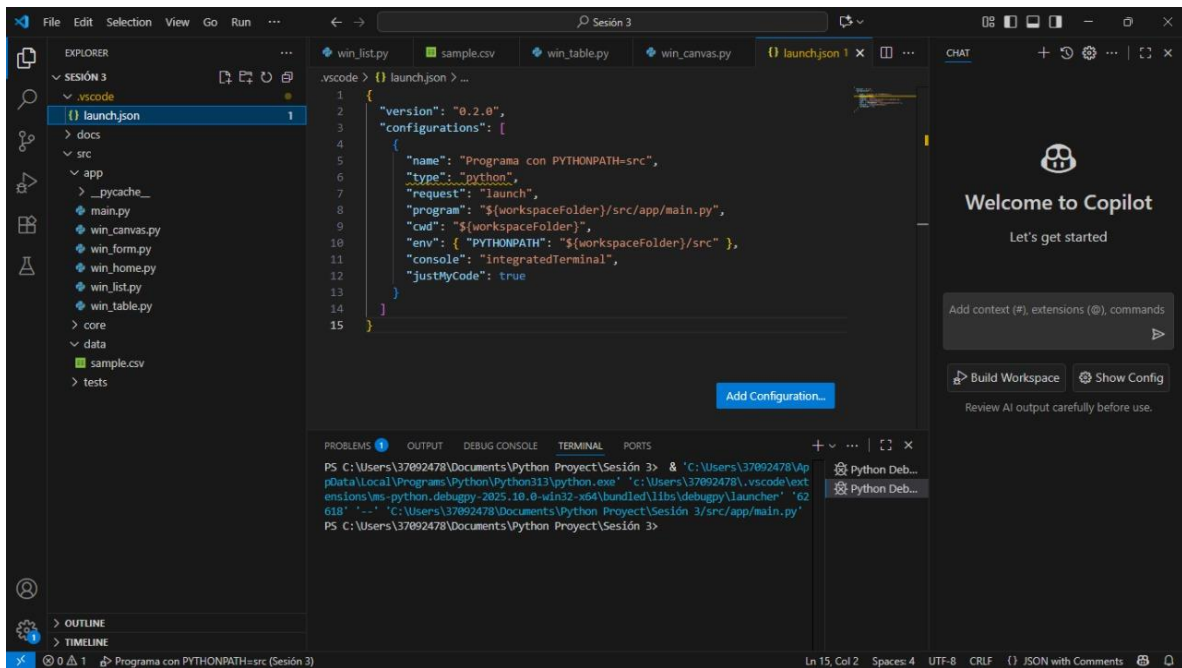
Un óvalo azul claro.

Una línea gruesa.

Un texto centrado: "¡Hola Canvas!" con fuente grande y negrita.

Botón de cerrar:

Finalmente, hay un botón "Cerrar" que destruye la ventana cuando se presiona.



Análisis de las claves principales

version: Indica el formato del archivo; aquí es "0.2.0".

configurations: Es una lista de posibles configuraciones de lanzamiento. En este caso, solo hay una configuración personalizada:

name: "Programa con PYTHONPATH=src"

Es el nombre descriptivo usado en el menú de ejecución.

type: "python"

Indica que es un entorno de depuración Python.

request: "launch"

Especifica que debe lanzar el programa (en vez de adjuntar a uno ya corriendo).

program: "\${workspaceFolder}/src/app/main.py"

Es la ruta al archivo principal que se ejecuta; aquí, el programa empieza en main.py dentro de src/app.

cwd (current working directory): "\${workspaceFolder}"

El directorio raíz del proyecto; útil para referencias relativas.

env:

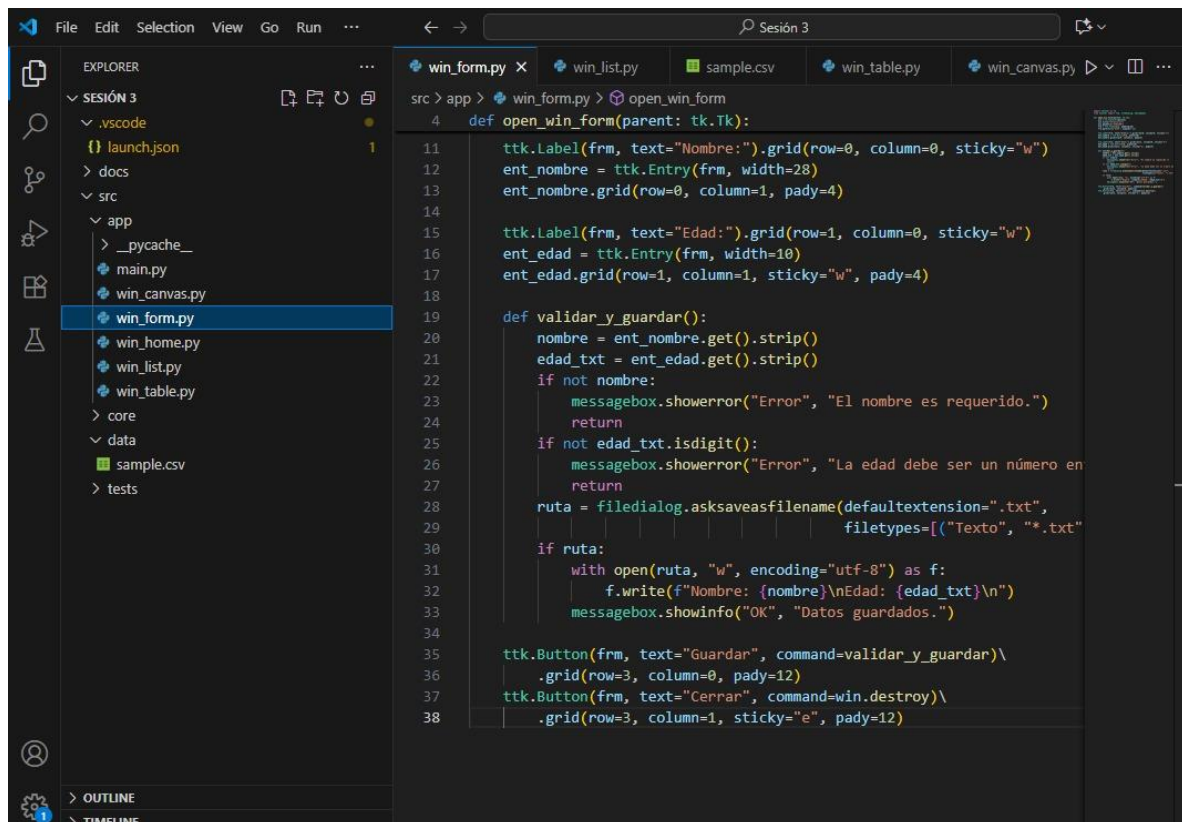
Se define PYTHONPATH con valor `${workspaceFolder}/src`. Esto permite que los módulos dentro de src se importen correctamente, facilitando la organización del proyecto.

console: "integratedTerminal"

Las entradas/salidas del programa se muestran en la terminal integrada de VS Code.

justMyCode: true

Solo depura el código propio del usuario, ignorando librerías externas.



The screenshot shows the Visual Studio Code interface with a Python project. The Explorer sidebar on the left shows a project structure with a 'src' folder containing an 'app' subfolder. The 'app' folder contains several Python files, with 'win_form.py' selected. The main editor window displays the code for 'win_form.py', which defines a Tkinter window with two text input fields for 'Nombre' and 'Edad', and buttons for 'Guardar' and 'Cerrar'. The code includes validation logic for the inputs and a function to save the data to a file.

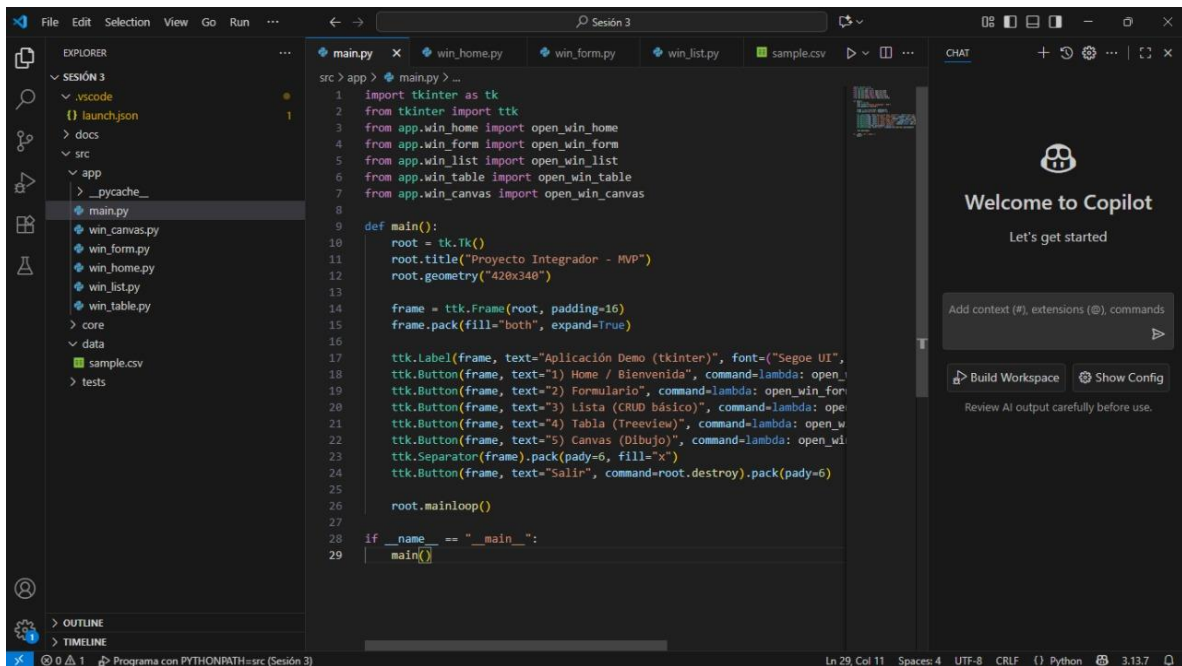
```
src > app > win_form.py > open_win_form
4 def open_win_form(parent: tk.Tk):
11     ttk.Label(frm, text="Nombre:").grid(row=0, column=0, sticky="w")
12     ent_nombre = ttk.Entry(frm, width=28)
13     ent_nombre.grid(row=0, column=1, pady=4)
14
15     ttk.Label(frm, text="Edad:").grid(row=1, column=0, sticky="w")
16     ent_edad = ttk.Entry(frm, width=10)
17     ent_edad.grid(row=1, column=1, sticky="w", pady=4)
18
19     def validar_y_guardar():
20         nombre = ent_nombre.get().strip()
21         edad_txt = ent_edad.get().strip()
22         if not nombre:
23             messagebox.showerror("Error", "El nombre es requerido.")
24             return
25         if not edad_txt.isdigit():
26             messagebox.showerror("Error", "La edad debe ser un número en")
27             return
28         ruta = filedialog.asksaveasfilename(defaultextension=".txt",
29                                             filetypes=[("Texto", "*.txt")])
30         if ruta:
31             with open(ruta, "w", encoding="utf-8") as f:
32                 f.write(f"Nombre: {nombre}\nEdad: {edad_txt}\n")
33             messagebox.showinfo("OK", "Datos guardados.")
34
35     ttk.Button(frm, text="Guardar", command=validar_y_guardar)\
36         .grid(row=3, column=0, pady=12)
37     ttk.Button(frm, text="Cerrar", command=win.destroy)\
38         .grid(row=3, column=1, sticky="e", pady=12)
```

Explicación línea por línea (resumida y profesional)

- `open_win_form(parent: tk.Tk)`: función principal, recibe la ventana padre y crea un nuevo formulario como ventana hija.
- Elementos UI:
 - `ttk.Label` y `ttk.Entry` para capturar Nombre y Edad.
 - Los elementos se colocan usando `.grid()` para posicionar en la interfaz.
- Función interna `validar_y_guardar`:
 - Obtiene el texto de los campos.
 - Valida que el nombre *no esté vacío* y que la edad *sea un número* (usando `isdigit`).
 - Si hay errores, muestra ventana de error con `messagebox.showerror` y detiene la función.
 - Si los datos son válidos, abre un diálogo para guardar el archivo (`filedialog.asksaveasfilename`).
 - Si el usuario escoge ruta válida, escribe los datos en el archivo en formato UTF-8 con salto de línea.
 - Al terminar, indica éxito usando `messagebox.showinfo`.
- Botones:
 - "Guardar": ejecuta `validar_y_guardar` al hacer clic.
 - "Cerrar": cierra la ventana de formulario.

Buenas prácticas y aspectos destacados

- Validación robusta: comprueba ambos campos antes de permitir el guardado.
- Diálogo de guardado estándar: permite al usuario elegir el destino del archivo y generar archivos de texto fácilmente.
- Encapsulamiento: el código mantiene la función de guardar interna, facilitando el mantenimiento y la ampliación.



Este archivo funciona como el punto de entrada principal de una aplicación de escritorio desarrollada en Python con Tkinter. Aquí se inicializa la ventana principal y se conecta el menú central con las diferentes "ventanas hijas" del proyecto módulo por módulo.

¿Cómo está organizado?

- Importaciones: Se importan Tkinter y sus widgets de estilo (ttk). También se importan las funciones para abrir cada ventana específica desde módulos separados (win_home, win_form, etc.) ubicados en src/app/.
- Función main()
 - Ventana raíz:
 - root = tk.Tk() inicia la aplicación.
 - Título: "Proyecto Integrador - MVP".
 - Tamaño fijo: 420x340 píxeles (lo ajustas aquí).
 - Frame principal:
 - Con padding y configurado para llenar la ventana y permitir expansión flexible.
 - Menú de botones:
 - Usa un Label como encabezado.
 - Crea seis botones principales, cada uno asociado a una ventana específica del proyecto:
 - Home/Bienvenida (open_win_home)

- Formulario (`open_win_form`)
- Lista (CRUD básico) (`open_win_list`)
- Tabla (Treeview) (`open_win_table`)
- Canvas (Dibujo) (`open_win_canvas`)
- Separador (visualmente separa el botón de salida)
- Salir: cierra la aplicación (`root.destroy`)
- Bucle principal: `root.mainloop()` mantiene la ventana activa.
- Bloque protector:
 - `if __name__ == "__main__": main()` asegura que solo se ejecute el programa cuando se corre el archivo directamente, no al importarlo como módulo.

Ventajas de este enfoque

- Modularidad: Cada ventana se mantiene en un archivo distinto, lo que simplifica el mantenimiento y la ampliación.
- Claridad: El menú principal centraliza el acceso a todas las funcionalidades.
- Escalabilidad: Fácil de añadir más ventanas o cambiar el menú.

```
src > app > win_list.py > open_win_list
1 import tkinter as tk
2 from tkinter import ttk, messagebox
3
4 def open_win_list(parent: tk.Tk):
5     win = tk.Toplevel(parent)
6     win.title("Lista (CRUD básico)")
7     win.geometry("420x380")
8
9     frm = ttk.Frame(win, padding=12)
10    frm.pack(fill="both", expand=True)
11
12    lb = tk.Listbox(frm, height=10)
13    lb.grid(row=0, column=0, rowspan=4, sticky="nsew", padx=(0, 8))
14    frm.columnconfigure(0, weight=1)
15    frm.rowconfigure(0, weight=1)
16
17    ent_item = ttk.Entry(frm)
18    ent_item.grid(row=0, column=1, sticky="ew")
19
20    def agregar():
21        v = ent_item.get().strip()
22        if v:
23            lb.insert("end", v)
24            ent_item.delete(0, "end")
25        else:
26            messagebox.showwarning("Aviso", "Escribe un texto para agregar")
27
28    def eliminar():
29        sel = lb.curselection()
30        if sel:
31            lb.delete(sel[0])
32
33    def limpiar():
34        lb.delete(0, "end")
```

Este módulo crea una ventana secundaria en una aplicación Tkinter de Python para gestionar una lista simple (CRUD básico), donde se pueden agregar, eliminar y limpiar elementos rápidamente.

Estructura y funcionalidad principal

- `open_win_list(parent: tk.Tk)`: es la función para abrir la ventana de lista.
 - Crea una ventana hija (`Toplevel`) titulada "Lista (CRUD básico)" con tamaño 420x380.
 - Dentro de la ventana, hay un `Frame` principal para ordenar el contenido con padding.

Widgets utilizados:

- `Listbox (lb)`: Muestra los elementos (height=10 filas). Ocupa la columna 0 y varias filas, y puede crecer según el espacio (usa `sticky="nsew"` y configuración de peso para expandirse con la ventana).
- `Entry (ent_item)`: Campo de texto para introducir nuevos elementos. Ocupa la columna 1 en la misma fila que la lista.

Funciones internas:

`agregar()`

- Obtiene texto limpio del Entry. Si no está vacío:
 - Inserta el texto al final de la lista (1b), luego limpia el Entry.
- Si está vacío:
 - Muestra una advertencia usando `messagebox.showwarning` pidiendo al usuario que escriba algo antes de agregar.

`eliminar()`

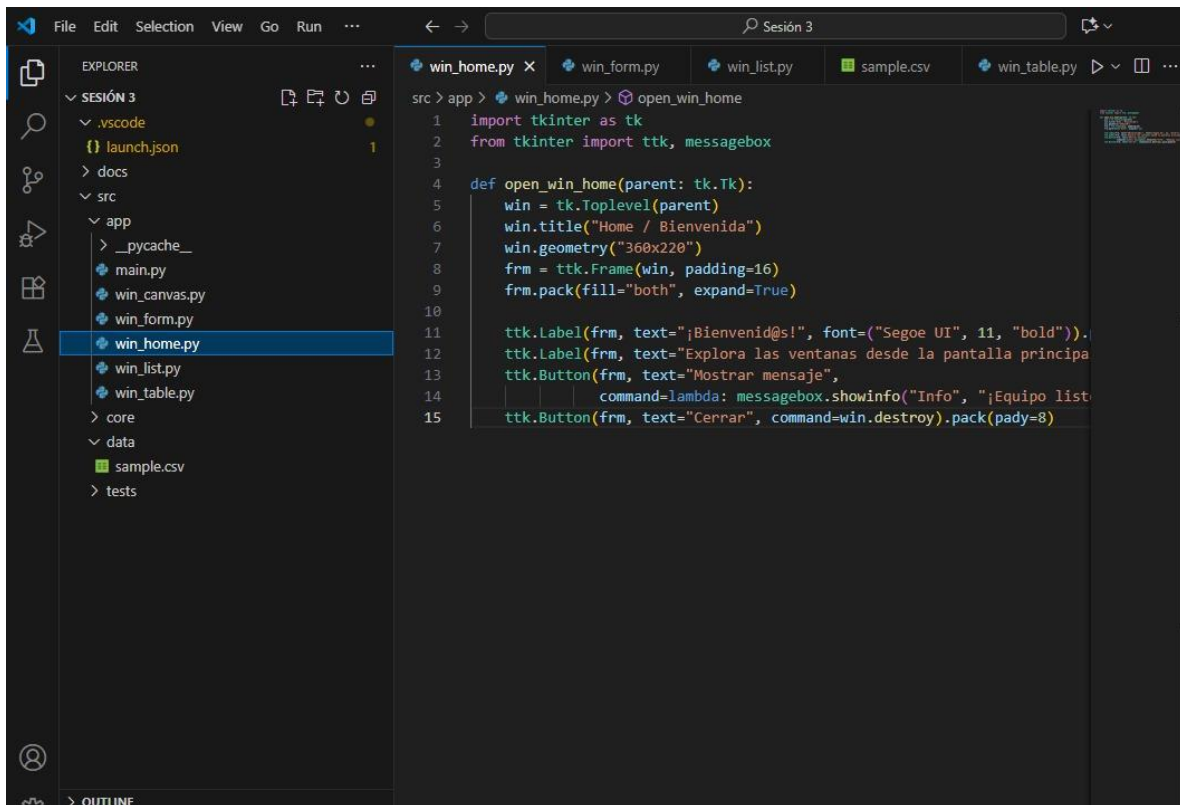
- Borra el elemento actualmente seleccionado en el Listbox, si hay alguno.

`limpiar()`

- Borra todos los elementos del Listbox.

¿Cómo conectan los widgets y las funciones?

El código aprovecha la organización en filas y columnas de Tkinter y encapsula toda la lógica en la ventana secundaria. Aunque en la imagen no se muestran los botones que llaman a `agregar()`, `eliminar()`, y `limpiar()`, normalmente estos se añaden debajo o al lado del Entry con `tk.Button` o `ttk.Button`, cada uno con su comando configurado.



```
src > app > win_home.py > open_win_home
1  import tkinter as tk
2  from tkinter import ttk, messagebox
3
4  def open_win_home(parent: tk.Tk):
5      win = tk.Toplevel(parent)
6      win.title("Home / Bienvenida")
7      win.geometry("360x220")
8      frm = ttk.Frame(win, padding=16)
9      frm.pack(fill="both", expand=True)
10
11     ttk.Label(frm, text="¡Bienvenid@s!", font=("Segoe UI", 11, "bold")).
12     ttk.Label(frm, text="Explora las ventanas desde la pantalla principa
13     ttk.Button(frm, text="Mostrar mensaje",
14               command=lambda: messagebox.showinfo("Info", "¡Equipo list
15     ttk.Button(frm, text="Cerrar", command=win.destroy).pack(pady=8)
```

Este módulo define la ventana de bienvenida de tu aplicación, utilizada para mostrar un mensaje introductorio y opciones sencillas al usuario.

¿Qué hace este módulo?

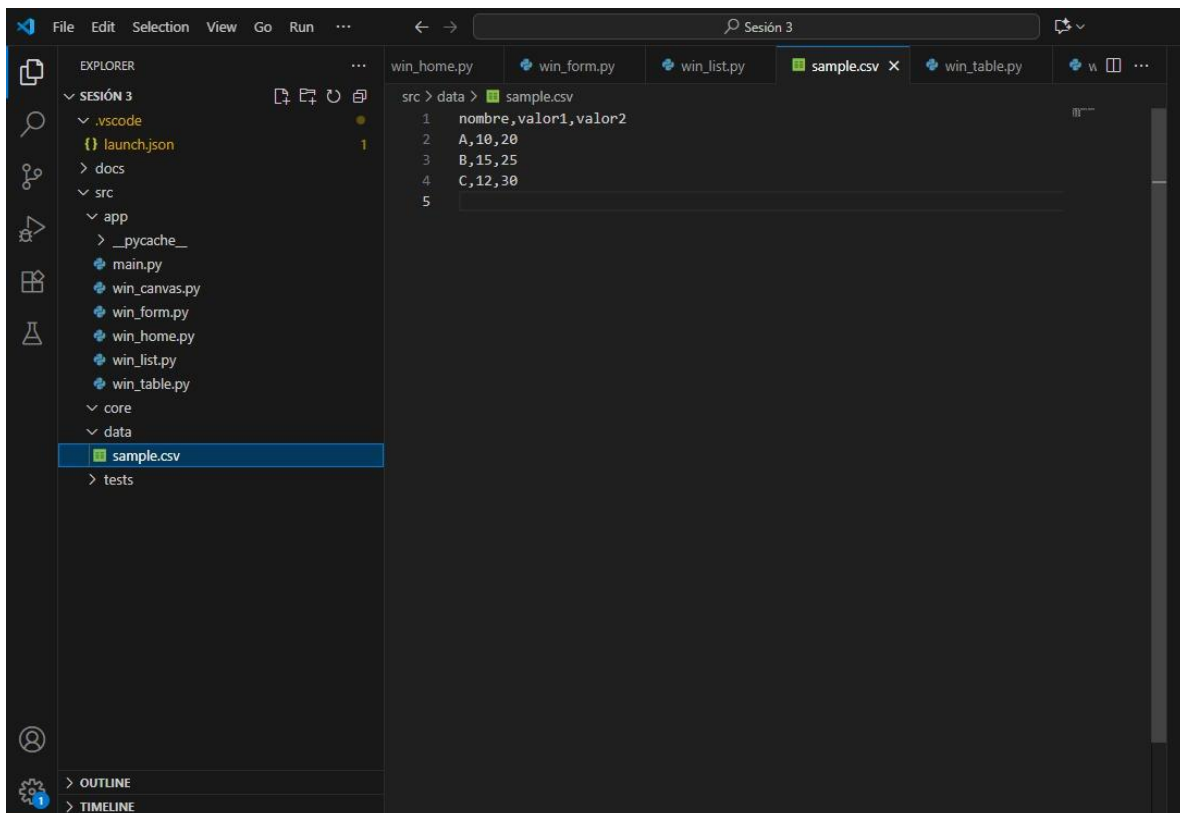
- Abre una ventana secundaria (Toplevel) cuando se llama la función `open_win_home(parent)`.
- La ventana tiene título "Home / Bienvenida" y tamaño fijo de 360x220 píxeles.
- Crea un Frame con relleno para una mejor presentación.

Widgets destacados:

- Label de bienvenida:
 - Muestra el mensaje "¡Bienvenid@s!" con fuente negrita y mayor tamaño.
- Label de instrucciones:
 - Explica que se pueden explorar las ventanas desde la pantalla principal.
- Botón "Mostrar mensaje":
 - Al hacer click, abre una ventana emergente (MessageBox) con información: "¡Equipo listo para comenzar!".
- Botón "Cerrar":
 - Cierra esta ventana de bienvenida al hacer clic.

Buenas prácticas

- Modularidad: Facilita reutilizar y mantener la bienvenida como ventana independiente.
- UI sencilla y clara: Útil para orientar por primera vez al usuario.
- Uso de lambda para comandos inline: Permite pasar argumentos o funciones anónimas para acciones en botones.

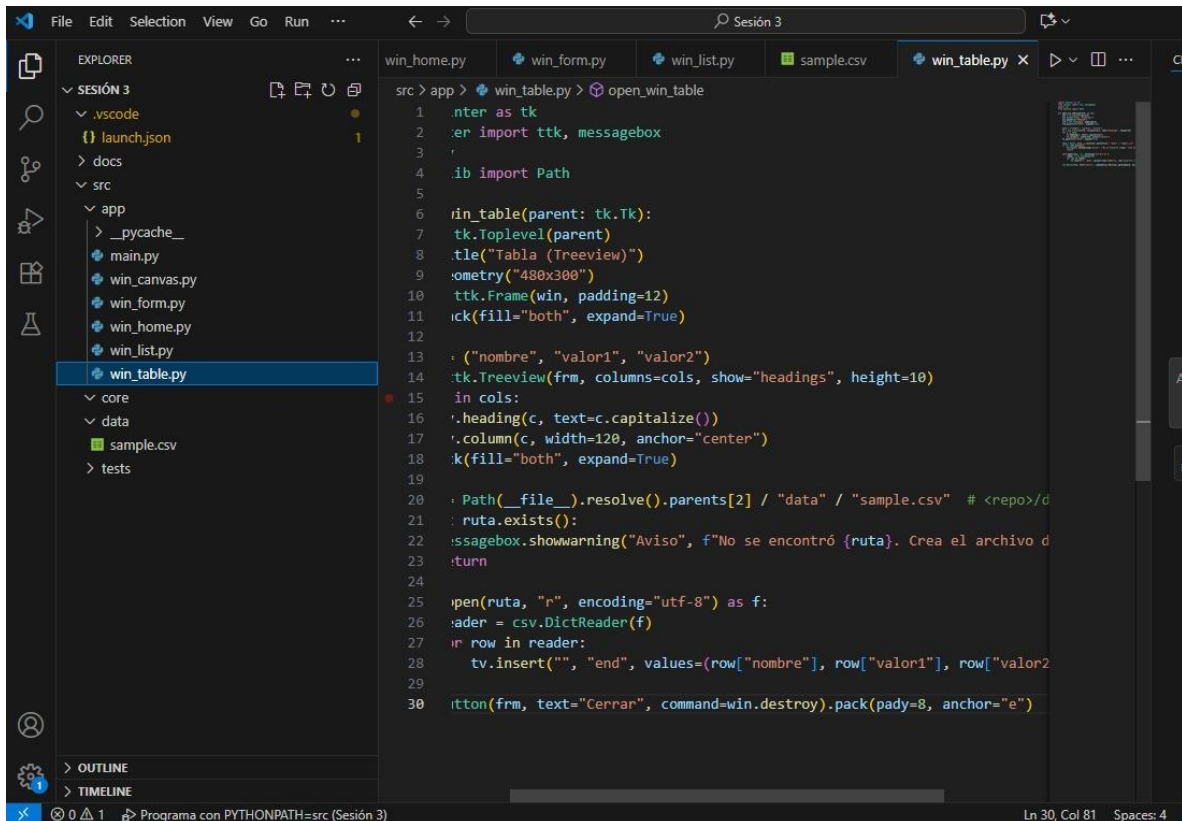


Este archivo es un archivo CSV (valores separados por comas), ubicado en la carpeta `src/data/` de tu proyecto. Es un formato común para manejar y compartir datos tabulares.

Estructura del contenido

- Primera fila: Son los encabezados de las columnas: `nombre, valor1, valor2`.
- Filas siguientes: Cada una representa un registro con tres campos:
 - Ejemplo:
 - `A,10,20` → Nombre: A, Valor1: 10, Valor2: 20
 - `B,15,25` → Nombre: B, Valor1: 15, Valor2: 25

- C,12,30 → Nombre: C, Valor1: 12, Valor2: 30



```
src > app > win_table.py > open_win_table
1  .enter as tk
2  .er import ttk, messagebox
3  .
4  .ib import Path
5
6  rin_table(parent: tk.Tk):
7      tk.Toplevel(parent)
8      .tile("Tabla (Treeview)")
9      .ometry("480x300")
10     ttk.Frame(win, padding=12)
11     ck(fill="both", expand=True)
12
13     . ("nombre", "valor1", "valor2")
14     tk.Treeview(frm, columns=cols, show="headings", height=10)
15     in cols:
16         .heading(c, text=c.capitalize())
17         .column(c, width=120, anchor="center")
18         k(fill="both", expand=True)
19
20     . Path(__file__).resolve().parents[2] / "data" / "sample.csv" # <repo>/d
21     . ruta.exists():
22         .ssagebox.showwarning("Aviso", f"No se encontró {ruta}. Crea el archivo d
23     .return
24
25     .open(ruta, "r", encoding="utf-8") as f:
26         .ader = csv.DictReader(f)
27         in row in reader:
28             .tv.insert("", "end", values=(row["nombre"], row["valor1"], row["valor2
29
30     .ttton(frm, text="Cerrar", command=win.destroy).pack(pady=8, anchor="e")
```

Este módulo implementa una ventana de tabla en tu aplicación Tkinter usando el widget Treeview de ttk, mostrando datos que vienen directamente del archivo sample.csv.

¿Qué hace el código?

- Función principal: open_win_table(parent: tk.Tk)
 - Crea una ventana hija (Toplevel) titulada "Tabla (Treeview)".
 - Tamaño: 480x300 px.
- Estructura del Frame:
 - Se crea un Frame para organizar los componentes usando padding.
- Definición de columnas:
 - Define las columnas nombre, valor1 y valor2.

- Crea un `Treeview` configurado para mostrar esos encabezados y establece propiedades (ancho, alineación al centro).
 - El widget se expande para llenar el frame.
- Carga de datos desde CSV:
 - Usa la clase `Path` de `pathlib` para armar la ruta absoluta al archivo `sample.csv` dentro de la carpeta `data`.
 - Chequeo robusto: Si no existe el archivo, muestra advertencia con `messagebox.showwarning` y sale de la función.
 - Si existe, abre el archivo usando `csv.DictReader` (esto convierte cada fila a un diccionario) y cada registro se inserta como nueva fila en el `Treeview`.
- Botón cerrar:
 - Agrega un botón para cerrar la ventana, alineado a la derecha inferior (`anchor="e"`).

Ventajas y buenas prácticas

- Separación UI/Lógica: Todo lo relacionado con la tabla está bien encapsulado.
 - Uso de `DictReader`: Permite manejar cabecales dinámicamente y escribir código más legible.
 - Reutilizable: Si cambias los datos del CSV, la tabla los muestra automáticamente en cada apertura.
1. Pasos de Ejecución:

Inicio desde `launch.json`:

 - a. VS Code usa el archivo `launch.json` que indica ejecutar el script principal `src/app/main.py`.
 - b. Establece `PYTHONPATH` a `src/` para permitir importaciones modulares.
 2. Ejecuta `main.py`:
 - a. Se importa `Tkinter` y funciones para abrir ventanas específicas (`home`, `form`, `list`, `table`, `canvas`).
 - b. Se crea la ventana principal (`root = tk.Tk()`), con título y tamaño definidos.
 - c. Se crea un frame principal con botones que llaman a funciones para abrir ventanas hijas de la app.
 3. Despliegue del menú principal:
 - a. Se muestran botones para abrir ventanas: bienvenida, formulario, lista CRUD, tabla, dibujo.
 - b. También un botón para salir que cierra la ventana `root`.
 4. Interacción del usuario:

- a. Al pulsar botones, se llaman funciones específicas que abren ventanas secundarias (Toplevel).
 - i. Por ejemplo, `open_win_form` abre la ventana formulario con validaciones y guardado a archivo.
 - ii. `open_win_list` muestra un listado editable simple.
 - iii. `open_win_table` lee el archivo CSV y muestra sus datos en un Treeview.
 - iv. `open_win_canvas` crea un área de dibujo con formas básicas.
 - v. `open_win_home` muestra una ventana de bienvenida con mensajes.
- 5. Cada ventana se ejecuta en segundo plano independientemente:
 - a. Cada ventana Toplevel puede manejar su propio evento y lógica mientras la ventana principal sigue abierta.
 - b. Botones como "Cerrar" destruyen su ventana secundaria.
- 6. Cierre del programa:
 - a. Cuando se cierra la ventana principal (`root.destroy()`), se termina todo el ciclo de eventos y la ejecución del programa.

pruebas

