Una Ambiciosa Introducción a Python
Parte II

Introducción a Tkinter

Tkinter es la biblioteca estándar de Python para crear interfaces gráficas de usuario (GUIs). El script ui.py que pertenece al proyecto de Biblioteca visto en clase nos presenta un ejemplo completo de cómo estructurar una aplicación Tkinter para gestionar propiedades.

Estructura básica

1. Ventana principal (Tk): Es el contenedor principal de la aplicación.

```
self.__root = Tk()
self.root.geometry("900x900")  # Define tamaño
self.root.title("AIPython II - Inmobiliaria")  # Título de la ventana
```

2. Frames: Los contenedores que nos permiten organizar los widgets.

```
form_frame = Frame(self.root)  # Crea un frame dentro de la ventana principal
form_frame.pack(expand=True)  # Lo posiciona usando pack()
```

Widgets utilizados

- Label: Muestra un texto estático.

```
Label(master=header_frame, text="Registro de propiedades", font=(APP_FONT, 20))
```

- Entry: Permite al usuario ingresar datos.

```
Entry(data_frame, textvariable=field_var, width=entry_width)
```

- Combobox: Listas desplegables para selección

```
ttk.Combobox(frame, textvariable=property_type, values=property_types, state="readonly")
```

- Checkbutton: Casillas de verificación para opciones booleanes.

```
Checkbutton(data_frame, text="Tiene balcón", variable=balcony_var)
```

- Treeview: Muestra datos en forma de tabla

```
self.properties_tree = ttk.Treeview(self.table_frame, columns=columns, show='headings')
```

Gestión de Variables

Tkinter usa variables especiales para manejar los datos de los widgets:

```
location_input = StringVar()  # Para texto
rooms_input = IntVar()       # Para números enteros
surface_input = DoubleVar()  # Para números decimales
balcony_var = BooleanVar()   # Para valores True/False
```

Lic. Franco Herrera

Una Ambiciosa Introducción a Python
Parte II

Organización de widgets

Métodos de posicionamiento

- pack: Organiza widgets en bloques antes de colocarlos.

```
header.pack(side=TOP, fill=BOTH, ipady=10)
```

- grid: Organiza widgets en una cuadricula, ver clase.
- place: Posicionamiento absoluto por coordenadas, ver clase.

Manejo de eventos

Los widgets pueden responder a acciones del usuario:

```
# Evento de doble clic en la tabla
self.properties_tree.bind("<Double-1>", self.on_property_double_click)

# Evento al cambiar selección en combobox
property_type.trace('w', update_additional_fields)
```

Ventanas secundarias (Toplevel)

Para crear ventanas emergentes:

```
edit_window = Toplevel(self.root)
edit_window.title("Editar Propiedad")
edit_window.geometry("300x400")
```

Buenas prácticas aplicadas en el ejemplo:

1. **Separación en métodos**: Cada componente tiene su propio método (__add_title(), __add_presentation(), etc.).
2. **Uso de frames**: Se agrupan widgets relacionados lógicamente.
3. **Manejo de errores**: Validación de entradas y mensajes de error claros.
4. **Reutilización de código**: El método create_label() simplifica la creación de etiquetas.
5. **Actualización dinámica**: El método refersh_properties_tree() actualiza la vista cuando cambian los datos

Tips

- Planifica primero: Dibuja un esquema de tu interfaz antes de codificar.
- Empieza simple: Construye la estructura básica primero y luego añade complejidad.
- Usa nombres descriptivos: Como se ve en el código, nombres como form_frame o add_property_button ayudan a entender el código.
- Mantén la lógica separada: La clase GraphicalUserInterface se encarga solo de la presentación, no de la lógica de negocio.
- Prueba frecuentemente: Verifica cada nuevo widget o funcionalidad que añadas.

Lic. Franco Herrera