Interfaces Gráficas (GUI) con Tkinter

Clase 4: Múltiples Ventanas y Eventos Avanzados

Prof. D.Sc. BARSEKH-ONJI Aboud

Facultad de Ingeniería
Universidad Anáhuac México

18 de noviembre de 2025

Agenda

Aplicaciones Reales

El Problema

Casi ninguna aplicación real es una sola ventana. Pensemos en cualquier software profesional:

- Una ventana principal (el editor de texto, el navegador, etc.).
- Ventanas secundarias para "Configuración" o "Preferencias".
- Diálogos emergentes para "Guardar Archivo" o "Acerca de...".
- Una ventana de "Login" que da paso a la aplicación principal.

Aplicaciones Reales

La Solución: `tk.Toplevel()`

Para crear una nueva ventana que "flota" encima de nuestra ventana principal, usamos el widget `tk.Toplevel()`.

Crear y Manejar Múltiples Ventanas

Crear una Ventana Toplevel

Se crea de forma muy similar a otros widgets. Es una ventana "hija" de la ventana raíz. Si cierras la raíz, esta también se cierra.

```
# Dentro de un metodo de nuestra clase App
def abrir_ventana_secundaria(self):
    # 'self' (el Frame) es el padre de la nueva ventana
    ventana_nueva = tk.Toplevel(self)

    ventana_nueva.title('Ventana Secundaria')
    tk.Label(ventana_nueva, text='¡Soy una nueva ventana!').pack()

```

Crear y Manejar Múltiples Ventanas

Manejo de Formularios

Podemos controlar la visibilidad de las ventanas:

- `ventana.destroy()`: **Cierra y destruye** la ventana. Se usa para cerrar un pop-up o la ventana de login.
- `ventana.withdraw()`: **Oculta** la ventana sin destruirla.
- `ventana.deiconify()`: **Muestra** una ventana que estaba oculta.

Agenda

Más Allá del `command`

El Problema

La propiedad `command` es genial para botones, pero ¿qué pasa si queremos reaccionar a otras acciones?

- ¿Hacer clic derecho en una etiqueta?
- ¿Presionar la tecla "Enter" en un campo de texto?
- ¿Detectar cuándo el mouse pasa por encima de una imagen?

La Solución: El Método `.bind()`

Todos los widgets en Tkinter tienen un método universal llamado `.bind()` que nos permite "atar" (bind) una función de Python a un evento específico en ese widget.

Sintaxis de .bind()

Paso 1: La Función (Callback)

La función que es llamada por .bind() **debe** aceptar un argumento, que por convención se llama event. Python pasa automáticamente información sobre el evento (como la posición X/Y del mouse).

```
1 # La funcion DEBE aceptar el argumento 'event'
2 def mi_callback(event):
3     print('¡Evento detectado!')
4     # event tiene info util: event.x, event.y
5     print(f'Clic en la posicion: {event.x}, {event.y}')
6
```

Sintaxis de .bind()

Paso 2: El "Atado"

Usamos .bind() con dos argumentos: el string del evento y el nombre de la función (sin paréntesis).

```
# Sintaxis:
# mi_widget.bind(''<DescriptorDelEvento>'', funcion_callback)

# Ejemplo:
etiqueta = tk.Label(self, text=''Haz clic derecho en mi'')
etiqueta.pack()

# Atamos el evento <Button-3> (clic derecho) a la funcion
etiqueta.bind(''<Button-3>'', mi_callback)

```

Principales Eventos de Mouse

Tipos de Eventos de Clic

Podemos capturar diferentes tipos de clics:

- `<Button-1>`: Clic izquierdo del mouse. (También se le llama **mouse down**).
- `<Button-3>`: Clic derecho del mouse.
- `<Double-Button-1>`: Doble clic izquierdo (**dbclick**).

Principales Eventos de Mouse

Drag y Drop (Arrastrar y Soltar)

El "arrastrar y soltar" es una combinación de dos eventos:

- `<B1-Motion>`: **(Drag)** Se dispara continuamente *mientras* el botón 1 está presionado y el mouse se mueve.
- `<ButtonRelease-1>`: **(Drop)** Se dispara cuando el usuario *suelta* el botón 1.

Principales Eventos de Mouse

Ejemplo: Rastrear el Mouse

```
1  def al_arrastrar(event):
2      print(f''Mouse arrastrado a: {event.x}, {event.y}'')
3
4  # Creamos un area de texto
5  area_texto = tk.Text(self, height=5, width=30)
6  area_texto.pack()
7
8  # Atamos el evento de arrastrar
9  area_texto.bind(''<B1-Motion>'', al_arrastrar)
10
```

Agenda

Tarea 4: Aplicación de Login

Objetivo

Combinar el conocimiento de **Clases** (Clase 3) con el manejo de **múltiples ventanas** (Clase 4) para crear una aplicación de login funcional.

Tarea 4: Aplicación de Login

Instrucciones

- **Estructura:** Crea tu aplicación de Login usando la estructura de **Clase** que aprendimos.
- **Ventana de Login:** La ventana principal (‘tk.Tk‘) será la ventana de Login. Debe contener:
  - Un `Label` y un `Entry` para la contraseña.
  - Un `Button` para "Ingresar".
  - Un `Label` para mensajes de error (inicialmente vacío).

Tarea 4: Aplicación de Login

Instrucciones

- **Lógica de Login:** Escribe un **método** (función) que se ejecute con el `command` del botón:
  - Si la contraseña es `''1234''`, el programa debe **cerrar** la ventana de Login (‘self.master.destroy()‘) y **abrir** una nueva ventana principal (mira la pista).
  - Si la contraseña es incorrecta, debe actualizar el `Label` de error con el texto "Contraseña incorrecta".

Tarea 4: Aplicación de Login

Pista Clave: Dos Clases

La mejor forma de hacerlo es con **dos clases**: una para la App de Login y otra para la App Principal.

```
class AppPrincipal(tk.Frame):
    def __init__(self, master):
        super().__init__(master)
        # ... widgets de la app principal ...
class AppLogin(tk.Frame):
    def __init__(self, master):
        super().__init__(master)
        # ... widgets de login (entry, boton) ...
        self.boton_login.config(command=self.verificar_login)
    def verificar_login(self):
        if self.entrada_pass.get() == ''1234'':
            self.master.destroy() # Cierra la ventana de login

```

Tarea 4: Aplicación de Login

Pista Clave: Dos Clases

La mejor forma de hacerlo es con **dos clases**: una para la App de Login y otra para la App Principal.

```
1
2               # Crea la nueva ventana principal
3               nueva_ventana = tk.Tk()
4               app_main = AppPrincipal(master=nueva_ventana)
5               app_main.mainloop()
6           else:
7               # ... mostrar error ...
8   # --- Codigo Principal ---
9   if __name__ == ''__main__'':
10      ventana_login = tk.Tk()
11      app = AppLogin(master=ventana_login)
12      app.mainloop()
13
```