

Interfaces gráficas de usuario

Ricardo Pérez López

IES Doñana, curso 2025/2026



Generado el 2025/12/17 a las 16:09:00

1. Introducción a Tkinter
2. Widgets básicos
3. *Layout* y organización de la interfaz
4. Eventos y funciones asociadas

1. Introducción a Tkinter

1.1. ¿Qué es Tkinter?

¿Qué es Tkinter?

- ▶ Tkinter es la biblioteca estándar de Python para crear interfaces gráficas de usuario (GUI).
- ▶ Proporciona una forma sencilla de construir aplicaciones con ventanas, botones, etiquetas, campos de texto y otros elementos visuales.
- ▶ Es multiplataforma, lo que significa que las aplicaciones creadas con Tkinter pueden ejecutarse en diferentes sistemas operativos como GNU/Linux, Windows y macOS sin modificaciones importantes.
- ▶ Es relativamente fácil de aprender y usar, lo que la hace adecuada para principiantes en el desarrollo de interfaces gráficas.

1.2. Instalación y primeras pruebas

Instalación y primeras pruebas

- ▶ Tkinter forma parte de la biblioteca estándar de Python, por lo que debería encontrarse en cualquier instalación normal de Python.

Esa es una de las principales ventajas que tiene respecto a otras bibliotecas de GUI como PyQt.

- ▶ Al ejecutar el siguiente comando:

```
$ python -m tkinter
```

desde la línea de comandos del sistema operativo, se debería abrir una ventana que demuestre una interfaz Tk simple para saber si tkinter está instalado correctamente en su sistema.

- ▶ También muestra qué versión de Tcl/Tk está instalada para que pueda leer la documentación de Tcl/Tk específica de esa versión.

1.3. Recursos y documentación

Recursos y documentación

- ▶ Para saber más sobre Tkinter o resolver dudas técnicas puntuales, se pueden consultar los siguientes enlaces:
 - Página de documentación oficial de Tkinter en python.org
Información técnica y oficial de Tkinter.
 - TkDocs
Es un extenso tutorial sobre como crear interfaces de usuario con Tkinter. Explica conceptos clave y muestra enfoques recomendados para usar la API moderna.
 - Tkinter 8.5 reference: a GUI for Python
Documentación de referencia sobre Tkinter 8.5 donde se detallan clases disponibles, métodos y opciones.

1.4. Un primer ejemplo básico

Un primer ejemplo básico

- Un *Hola, mundo* muy elemental en Tkinter podría ser el siguiente:

```
import tkinter as tk          # Importa el módulo tkinter con el nombre tk
raiz = tk.Tk()                # Crea la ventana principal
tk.Button(raiz, text="Hola, mundo").grid() # Crea un botón dentro de la ventana principal
raiz.mainloop()              # Activa el bucle principal
```

- Este programa simplemente abre una ventana en la que se muestra un botón con el texto «**Hola, mundo**».
- El botón tiene el ancho justo para visualizar el texto que contiene, y la ventana tiene prácticamente el tamaño justo para contener el botón.
- Al pulsar ese botón no ocurre nada, y para salir del programa hay que finalizarlo cerrando la ventana o directamente terminando el proceso del intérprete.

1.5. La ventana principal (Tk)

La ventana principal (Tk)

- ▶ La clase `Tk` representa la ventana principal de la aplicación, y esta es importante por varios motivos:
 - Toda aplicación Tkinter debe tener una ventana principal que sea instancia de `Tk` (y lo normal es que sólo sea exactamente una).
 - Las instancias de `Tk` son contenedores de elementos gráficos (también llamados *widgets*). Salvo casos excepcionales (como las ventanas de diálogo), los *widgets* se deben visualizar siempre dentro de un contenedor y la instancia de `Tk` que creamos para nuestro programa nos sirve como contenedor principal de nuestra aplicación.

Sin un contenedor, no podríamos visualizar *widgets*, así que la ventana principal es imprescindible en cualquier aplicación Tkinter.
- ▶ El funcionamiento de la interfaz gráfica realmente comienza cuando activamos el bucle principal de la ventana principal invocando su método `mainloop`.

1.6. El bucle principal (*mainloop*)

El bucle principal (*mainloop*)

- ▶ En interfaces gráficas, la ejecución del programa está dirigida por eventos (por ejemplo, pulsar un botón o elegir una opción en un menú) y no por un flujo lineal de instrucciones como en los programas de consola.
- ▶ El bucle principal (*mainloop*) de Tkinter es la función que pone en marcha la aplicación gráfica y la mantiene funcionando hasta que el usuario la cierra.
- ▶ El bucle principal se activa invocando el método `mainloop` sobre la instancia de `Tk` que representa la ventana principal de la aplicación.
- ▶ A partir de ese momento, el programa irá atendiendo los eventos que se vayan produciendo ejecutando el código encargado de *manejar* o *gestionar* dicho evento.

► Sus funciones principales son:

■ Inicia la gestión de eventos:

Tkinter entra en un bucle infinito en el que espera la aparición de eventos (pulsaciones de los botones del ratón, pulsaciones de teclas, redimensionado de ventanas, etc.) y los gestiona según los manejadores de eventos que se hayan definido en el código.

■ Mantiene visible la ventana:

Mientras `mainloop` está activo, la ventana de la aplicación se sigue mostrando y respondiendo a interacciones. Si no se llama a `mainloop`, la ventana puede crearse y cerrarse instantáneamente, porque el programa termina sin esperar eventos.

■ Es un bucle de eventos:

Internamente, comprueba si hay nuevos eventos en la cola de eventos del sistema operativo y los procesa uno a uno, actualizando la interfaz cuando sea necesario.

- ▶ El bucle principal termina:
 - Cuando el usuario cierra la ventana principal, o
 - Si el programa llama explícitamente al método `quit` de la ventana principal para finalizarlo.

- ▶ Aspectos importantes a tener en cuenta:

- El bucle principal es *bloqueante*:

Cuando se llama a `mainloop`, no se ejecuta nada después de esa línea hasta que el bucle termina.

- No se debe llamar varias veces:

Normalmente, se llama una sola vez. Si se necesita reiniciar la ventana, se debe crear un nuevo `Tk` o usar `mainloop` tras finalizar la ejecución anterior, pero no se debe intentar mantener dos ejecuciones simultáneas de `mainloop`.

2. Widgets básicos

2.1. Introducción

Introducción

- ▶ Los **widgets** son los componentes visuales de la interfaz.
- ▶ Los más comunes son:
 - **Label**: muestra texto o imágenes
 - **Button**: botón pulsable
 - **Entry**: campo de texto de una sola línea
 - **Text**: área de texto multilínea
 - **Checkbox**: casilla de verificación
 - **Radiobutton**: botón de opción (selección exclusiva)

2.2. Label, Button, Entry, Text, Checkbutton, Radiobutton

Label, Button, Entry, Text, Checkbutton, Radiobutton

El widget **Label** se utiliza para **mostrar texto o imágenes**. No permite interacción directa por parte del usuario.

Usos comunes: - Títulos y subtítulos - Etiquetas descriptivas de otros widgets - Mensajes informativos

Ejemplo:

```
label = tk.Label(root, text="Nombre:")  
label.pack()
```

Con estilos:

```
label = tk.Label(  
    root,  
    text="Bienvenido",  
    font=("Arial", 16),  
    fg="white",  
    bg="black"  
)  
label.pack()
```

El widget `Button` representa un **botón pulsable** que ejecuta una función cuando se hace clic.

Usos comunes: - Confirmar acciones - Lanzar cálculos - Cerrar ventanas

Ejemplo:

```
def saludar():  
    print("Hola")  
  
button = tk.Button(root, text="Saludar", command=saludar)  
button.pack()
```

Cambiar el texto dinámicamente:

```
button.config(text="Nuevo texto")
```

Entry es un **campo de texto de una sola línea**, pensado para introducir datos cortos.

Usos comunes: - Nombre de usuario - Edad - Búsquedas

Ejemplo:

```
entry = tk.Entry(root)
entry.pack()
```

Obtener y modificar su contenido:

```
texto = entry.get()
entry.delete(0, tk.END)
entry.insert(0, "Texto inicial")
```


El widget `Text` permite **introducir y mostrar texto multilínea**.

Usos comunes: - Editores de texto simples - Cuadros de comentarios - Visualización de logs

Ejemplo:

```
text = tk.Text(root, width=40, height=10)
text.pack()
```

Insertar y borrar contenido:

```
text.insert("1.0", "Hola
")
text.delete("1.0", tk.END)
```

`Checkbutton` representa una **casilla de verificación** que puede estar activada o desactivada.

Usos comunes: - Opciones independientes - Preferencias del usuario

Suele utilizarse junto con `BooleanVar` o `IntVar`.

Ejemplo:

```
var = tk.BooleanVar()
check = tk.Checkbutton(root, text="Aceptar condiciones", variable=var)
check.pack()
```

Consultar el estado:

```
print(var.get()) # True o False
```

Radiobutton permite **seleccionar una opción entre varias**, siendo excluyentes entre sí.

Usos comunes: - Selección de una sola alternativa - Formularios

Todos los botones del grupo comparten la misma variable de control.

Ejemplo:

```
opcion = tk.StringVar(value="A")  
rb1 = tk.Radiobutton(root, text="Opción A", variable=opcion, value="A")  
rb2 = tk.Radiobutton(r  
rb1.pack()  
rb2.pack()
```

Obtener la opción seleccionada:

```
print(opcion.get())
```

2.3. Atributos comunes: texto, color, fuente, tamaño

Atributos comunes: texto, color, fuente, tamaño

Muchos widgets de Tkinter comparten un conjunto de **atributos (u opciones)** que controlan su apariencia y comportamiento. Estos atributos pueden indicarse al crear el widget o modificarse posteriormente mediante el método `config()`.

Texto (text)

El atributo `text` define el **contenido textual** que muestra el widget.

Widgets habituales: - `Label` - `Button` - `Checkbutton` - `Radiobutton`

Ejemplo:

```
label = tk.Label(root, text="Hola mundo")  
label.pack()
```

Modificar el texto en tiempo de ejecución:

```
label.config(text="Nuevo texto")
```

Color (fg, bg)

Los colores se controlan principalmente con:

- ▶ **fg** o **foreground**: color del texto
- ▶ **bg** o **background**: color de fondo

Ejemplo:

```
button = tk.Button(  
    root,  
    text="Aceptar",  
    fg="white",  
    bg="green"  
)  
button.pack()
```

Los colores pueden indicarse por nombre ("red", "blue") o en formato hexadecimal ("#ff0000").

Fuente (font)

El atributo `font` controla el **tipo de letra, tamaño y estilo**.

Formato habitual:

```
font=(familia, tamaño, estilo)
```

Ejemplo:

```
label = tk.Label(  
    root,  
    text="Título",  
    font=("Arial", 18, "bold")  
)  
label.pack()
```

Estilos comunes: - "`bold`" - "`italic`" - "`underline`"

Tamaño (width, height)

Los atributos `width` y `height` definen el **tamaño del widget**, aunque su significado depende del tipo de widget:

- ▶ En `Label`, `Button` y `Entry`: número de caracteres
- ▶ En `Text`: caracteres (ancho) y líneas (alto)

Ejemplo:

```
entry = tk.Entry(root, width=30)
entry.pack()
```

```
text = tk.Text(root, width=40, height=5)
text.pack()
```

2.4. Métodos útiles: get, insert, delete

Métodos útiles: get, insert, delete

Muchos widgets proporcionan métodos para **acceder y modificar su contenido**, especialmente los widgets de entrada de texto.

get()

Obtiene el contenido actual del widget.

```
texto = entry.get()  
contenido = text.get("1.0", tk.END)
```


insert(posición, texto)

Inserta texto en una posición determinada.

```
entry.insert(0, "Texto inicial")  
text.insert("1.0", "Primera línea  
")
```


delete(inicio, fin)

Elimina contenido entre dos posiciones.

```
entry.delete(0, tk.END)  
text.delete("1.0", tk.END)
```


Resumen práctico

Widget	get	insert	delete
Entry	Sí	Sí	Sí
Text	Sí	Sí	Sí

3. *Layout* y organización de la interfaz

3.1. Introducción

Introducción

- ▶ Tkinter no posiciona los widgets automáticamente. Para ello se usan **gestores de geometría**.

3.2. Geometría con pack, grid y place

Geometría con pack, grid y place

- ▶ **pack**: organiza los widgets en bloques (arriba, abajo, izquierda, derecha)
- ▶ **grid**: organiza los widgets en filas y columnas
- ▶ **place**: posicionamiento absoluto (coordenadas)

Ejemplo con **pack**:

```
label.pack(side="top", fill="x")
```

Ejemplo con **grid**:

```
label.grid(row=0, column=0)  
button.grid(row=1, column=0)
```

❗ No se deben mezclar distintos gestores de geometría **en el mismo contenedor**.

3.3. Uso de Frame para dividir la ventana

Uso de Frame para dividir la ventana

Frame es un contenedor que permite agrupar widgets y estructurar la interfaz.

```
frame = tk.Frame(root)
frame.pack()

label = tk.Label(frame, text="Dentro del frame")
label.pack()
```

Usar **Frame** facilita:

- ▶ Interfaces más claras
- ▶ Reutilización de componentes
- ▶ Uso combinado de distintos gestores de geometría

3.4. Diseño *responsive* básico

Diseño *responsive* básico

Para que la interfaz se adapte al tamaño de la ventana:

- ▶ Usar `fill` y `expand` con `pack`
- ▶ Configurar pesos con `grid_rowconfigure` y `grid_columnconfigure`

Ejemplo:

```
root.columnconfigure(0, weight=1)
root.rowconfigure(0, weight=1)
```

4. Eventos y funciones asociadas

4.1. Introducción

Introducción

Tkinter funciona mediante **eventos** (acciones del usuario) y **callbacks** (funciones que se ejecutan como respuesta).

4.2. Asociar funciones a eventos (*callbacks*)

Asociar funciones a eventos (*callbacks*)

Un *callback* es una función que se pasa como argumento y se ejecuta cuando ocurre un evento.

```
def saludar():  
    print("Hola")
```

4.3. Uso de `command=`

Uso de command=

Muchos widgets (como `Button`) aceptan el parámetro `command`:

```
button = tk.Button(root, text="Saludar", command=saludar)
button.pack()
```

? Se pasa la función **sin paréntesis**.

4.4. Eventos con `bind`

Eventos con bind

`bind` permite asociar funciones a eventos más generales:

```
def al_pulsar_tecla(event):  
    print(event.keysym)  
  
root.bind("<Key>", al_pulsar_tecla)
```

Algunos eventos comunes:

- ▶ `<Button-1>`: clic izquierdo
- ▶ `<Key>`: pulsación de tecla
- ▶ `<Return>`: tecla Enter

4.5. Variables de control (StringVar, IntVar, etc.)

Variables de control (StringVar, IntVar, etc.)

Las **variables de control** enlazan el estado de un widget con una variable Python.

Tipos habituales:

- ▶ StringVar
- ▶ IntVar
- ▶ DoubleVar
- ▶ BooleanVar

Ejemplo:

```
var = tk.StringVar()
entry = tk.Entry(root, textvariable=var)
entry.pack()

print(var.get())
var.set("Nuevo valor")
```

Son especialmente útiles con **Entry**, **Checkbutton** y **Radiobutton**.

5. Bibliografía

Bibliografía

Roseman, Mark. n.d. *TkDocs.com*. <https://tkdocs.com/>.