

# Interfaces gráficas de usuario

Ricardo Pérez López

IES Doñana, curso 2025/2026

Generado el 2025/12/17 a las 16:46:00

## Índice

<b>1. Introducción a Tkinter</b>	<b>2</b>
1.1. ¿Qué es Tkinter?	2
1.2. Instalación y primeras pruebas	2
1.3. Recursos y documentación	2
1.4. Un primer ejemplo básico	3
1.5. La ventana principal ( <code>Tk</code> )	3
1.6. El bucle principal ( <code>mainloop</code> )	3
<b>2. Widgets básicos</b>	<b>4</b>
2.1. Introducción	4
2.2. <code>Label</code>	5
2.3. <code>Button</code>	5
2.4. <code>Entry</code>	6
2.5. <code>Text</code>	6
2.6. <code>Checkbutton</code>	7
2.7. <code>Radiobutton</code>	7
2.7.1. Atributos comunes	8
2.7.2. Métodos útiles	9
<b>3. Layout y organización de la interfaz</b>	<b>10</b>
3.1. Introducción	10
3.2. Geometría con <code>pack</code> , <code>grid</code> y <code>place</code>	11
3.3. Uso de <code>Frame</code> para dividir la ventana	11
3.4. Diseño <i>responsive</i> básico	12
<b>4. Eventos y funciones asociadas</b>	<b>12</b>
4.1. Introducción	12
4.2. Asociar funciones a eventos ( <i>callbacks</i> )	12
4.2.1. Uso de <code>command=</code>	12
4.2.2. Eventos con <code>bind</code>	12
4.3. Variables de control	13

## 1. Introducción a Tkinter

### 1.1. ¿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

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

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](https://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 *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 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*)

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

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
- `Checkbutton`: casilla de verificación
- `Radiobutton`: botón de opción (selección exclusiva)

## 2.2. Label

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()
```

## 2.3. Button

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

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")
```

## 2.4. Entry

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

- Usos comunes:
  - \* Campos de formulario.
  - \* 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")
```

## 2.5. Text

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)
```

## 2.6. Checkbutton

`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
```

## 2.7. Radiobutton

`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(root, text="Opción B", variable=opcion, value="B")

rb1.pack()
rb2.pack()
```

Obtener la opción seleccionada:

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

### 2.7.1. Atributos comunes

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()`.

#### 2.7.1.1. 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")
```

#### 2.7.1.2. Color (`fg` y `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`").

### 2.7.1.3. 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"

### 2.7.1.4. Tamaño (width y 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).

Ejemplos:

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

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

### 2.7.2. Métodos útiles

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

### 2.7.2.1. `get`

Obtiene el contenido actual del widget.

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

### 2.7.2.2. `insert`

Inserta texto en una posición determinada, con la siguiente sintaxis:

```
insert(posición, texto)
```

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

### 2.7.2.3. `delete`

Elimina contenido entre dos posiciones, con la siguiente sintaxis:

```
delete(inicio, fin)
```

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

## Resumen práctico

Widget	<code>get</code>	<code>insert</code>	<code>delete</code>
<code>Entry</code>	Sí	Sí	Sí
<code>Text</code>	Sí	Sí	Sí

## 3. Layout y organización de la interfaz

### 3.1. Introducción

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

En Tkinter, `pack`, `grid` y `place` son gestores de geometría (*geometry managers*).

Su función es decidir dónde y cómo se colocan los widgets dentro de un contenedor (una ventana `Tk` o un `Frame`).

Un widget no aparece en pantalla hasta que se le aplica alguno de estos métodos.

Los gestores de geometría son el mecanismo que usa Tkinter para:

- Asignar posición.
- Asignar tamaño.
- Reorganizar los widgets cuando la ventana cambia de tamaño.

Cada contenedor (`Tk`, `Frame`, etc.) puede usar sólo un gestor de geometría a la vez.

### 3.2. Geometría con `pack`, `grid` y `place`

Los tres gestores de geometría en Tkinter son:

- `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

`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

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

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

### 4.2. 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")
```

#### 4.2.1. 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**.

#### 4.2.2. 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>`: pulsación del botón izquierdo del ratón.
- `<Key>`: pulsación de cualquier tecla.

- `<Return>`: pulsación de la tecla Enter.

### 4.3. Variables de control

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`.

## Bibliografía

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