

# Programacion Python

## Interfaces: Tkinter

1. Introducción
2. Widgets
- 3.



# Introducción: Widgets

<https://recursospython.com/guias-y-manuales/introduccion-a-tkinter/>

Las interfaces gráficas son medios visuales, mucho más cómodos que una terminal de texto, a través de las cuales nuestros usuarios pueden interactuar y realizar tareas gráficamente.

El módulo Tkinter cuenta con una serie de componentes gráficos llamados Widgets, gracias a los cuales podemos diseñar nuestras interfaces.

Los widgets deben seguir una jerarquía a la hora de añadirse a la interfaz. Por ejemplo, un Marco (frame) forma parte del objeto raíz Tk. Y a su vez, un botón (button) puede formar parte de un contenedor como la raíz o un marco.

[Documentacion Tkinter](#)

Los que veremos en esta introducción a Tkinter son:



- **Tk:** Contenedor base o raíz de todos los widgets que forman la interfaz. No tiene tamaño propio sino que se adapta a los widgets que contiene.
- **Frame:** Marco contenedor de otros widgets. Puede tener tamaño propio y posicionarse en distintos lugares de otro contenedor (ya sea la raíz u otro marco).
- **Label:** Etiqueta dónde podemos mostrar algún texto estático.
- **Entry:** Campo de texto sencillo para escribir texto corto. Nombres, apellidos, números..
- **Text:** Campo de texto multilínea para escribir texto largo. Descripciones, comentarios...
- **Button:** Botón con un texto sobre el cual el usuario puede hacer clic.
- **Radiobutton:** Botón radial que se usa en conjunto donde es posible marcar una opción.



- **Checkbutton:** Botón cuadrado que se puede marcar con un tic.
- **Menu:** Estructura de botones centrados en la composición de menús superiores.
- **Dialogs:** Ventanas emergentes que permiten desde mostrar información al usuario (típico mensaje de alerta o de confirmación) hasta ofrecer una forma gráfica de interactuar con el sistema operativo (seleccionar un fichero de un directorio para abrirlo).



## Widget Tk (Raíz)

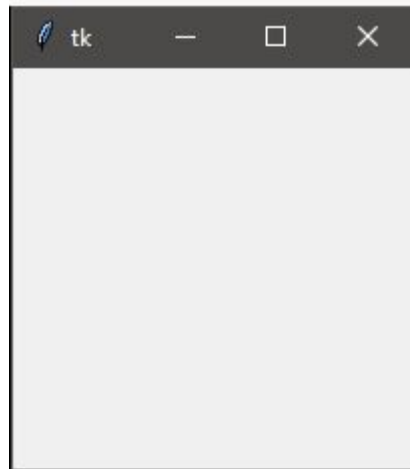
Recordad que la raíz es el contenedor base de todos los widgets que forman la interfaz, no tiene tamaño propio sino que se adapta a los widgets que contiene:

```
from tkinter import *

# Creamos la raíz
root = Tk()

# Comenzamos el bucle de
# aplicación, es como un while True
root.mainloop()

root.title("Hola mundo")      # Título de
# la ventana
root.iconbitmap('hola.ico')   # Icono de
# la ventana, en ico o xbm en Linux
root.resizable(0, 0)          # Desactivar
# redimensión de ventana
```





## Widget Frame (Marco)

Los Frames **son marcos** contenedores de otros widgets. Pueden tener tamaño propio y posicionarse en distintos lugares de otro contenedor (ya sea la raíz u otro marco):

```
from tkinter import *
root = Tk()
frame = Frame(root) # Hijo de root, no ocurre nada
frame.pack()        # Empaqueta el frame en la raíz
# Como no tenemos ningún elemento dentro del frame,
# no tiene tamaño y aparece ocupando lo mínimo posible, 0*0 px
# Color de fondo, background
frame.config(bg="lightblue")
frame.config(width=480,height=320) # Podemos establecer un tamaño,
                                   # la raíz se adapta al frame que contiene
root.mainloop()
```



Algo interesante de los frames es que permiten parámetros visuales utilizando atributos estándar:

```
frame.config(cursor="")           # Tipo de cursor
frame.config(relief="sunken")     # relieve del frame hundido
frame.config(bd=25)               # tamaño del borde en píxeles
```

Pero esto no es algo único de los Frames, todos los widgets aceptan estos parámetros visuales, incluso la raíz:

```
root.config(bg="blue")           # color de fondo, background
root.config(cursor="pirate")     # tipo de cursor (arrow defecto)
root.config(relief="sunken")     # relieve del root
root.config(bd=25)               # tamaño del borde en píxeles
```



Sin embargo, fijaros que curiosamente si hacemos la ventana grande, el frame se encuentra centrado arriba al medio, eso es porque el método pack alinea el widget arriba al medio. Esta posición se conoce como la distribución del Widget y podemos cambiarla de dos formas posibles justo al empacar el frame. Con alineación [arriba, abajo, izquierda, derecha] o con anclaje [N,S,E,W,NE...]:

```
frame.pack(side=RIGHT)    # a la derecha al medio  
frame.pack(anchor=SE)     # sudeste, abajo a la derecha
```





```
from tkinter import *

# Configuración de la raíz
root = Tk()
root.title("Hola mundo")
root.resizable(1,1)
root.iconbitmap('hola.ico')

frame = Frame(root, width=480, height=320)
frame.pack(fill='both', expand=1)
frame.config(cursor="pirate")
frame.config(bg="lightblue")
frame.config(bd=25)
frame.config(relief="sunken")

root.config(cursor="arrow")
root.config(bg="blue")
root.config(bd=15)
root.config(relief="ridge")

# Finalmente bucle de la aplicación
root.mainloop()
```



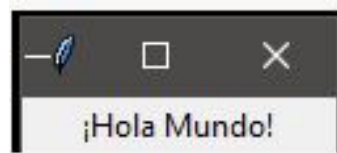
## Widget Label (Etiqueta de texto)

En esta lección vamos a trabajar el widget label utilizado para mostrar textos. Suele ser texto estático, de ahí que se llame label o **etiqueta** de texto.

```
from tkinter import *  
root = Tk()  
  
label = Label(frame, text="¡Hola Mundo!")  
label.pack()  
  
root.mainloop()
```

También se puede añadir directamente a la raíz y empaquetarla:

```
Label(root, text="¡Hola Mundo!").pack()
```





Evidentemente también tienen distintos parámetros visuales, para cambiar el color en primer plano, el del fondo, la fuente y su tamaño...:

```
label = Label(root, text="¡Otra  
etiqueta!")  
label.pack(anchor=CENTER)  
label.config(fg="blue",      # Foreground  
             bg="green",    # Background  
             font=("Verdana",24))
```

Pero una de las cosas más interesantes que nos permiten hacer es cambiar el texto sobre la marcha utilizando un objeto llamado `StringVar()` y su propiedad `textvariable`:

```
texto = StringVar()  
texto.set("Un nuevo texto")  
label.config(textvariable=texto)  # añadimos una variable de  
texto
```



Quizá ahora no parece muy útil, pero más adelante veremos cómo capturar el momento en que apretamos un botón y entonces cambiaremos el texto de una etiqueta.

Por cierto, algo que os va a gustar es que las etiquetas pueden contener imágenes, lo malo es que tkinter sólo acepta dos formatos de imagen a través de la clase `PhotoImage`: `pgm` y `gif`.

```
imagen =  
PhotoImage(file="imagen.gif")  
Label(root, image=imagen,  
      bd=0).pack()
```





## Widget Entry (Texto corto)

Los **campos de texto** sirven generalmente para que el usuario escriba un valor. Sería un puente que equivaldría a la función `input()` pero gráficamente. Lo bueno es que integra muchos métodos que le permiten desde borrar el texto a desactivar el campo.

```
from tkinter import *  
root = Tk()
```

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

```
root.mainloop()
```

```
entry = Entry(root)  
entry.pack(side=RIGHT)
```

```
label = Label(root, text="Nombre")  
label .pack(side=LEFT)
```



Lo malo es que si intentamos crear más etiquetas y campos, esto se posicionan mal:

```
entry2 = Entry(root)
entry2.pack(side=RIGHT)

label2 = Label(root, text="Apellidos")
label2 .pack(side=LEFT)
```

Como sabemos Pack() intenta posicionar automáticamente los elementos y alinearlos con los parámetros side y anchor, pero en este caso no hay una distribución más allá de la alineación, por éso se ve mal.

Para solucionarlo se puede hacer de distintas formas. Por ejemplo creando dos marcos, y añadir una etiqueta y un campo de texto en cada una. De esta forma el marco hará de separador:



```
frame1 = Frame(root)
frame1.pack()
```

```
entry = Entry(frame1)
entry.pack(side=RIGHT)
```

```
label = Label(frame1, text="Nombre")
label.pack(side=LEFT)
```

```
frame2 = Frame(root)
frame2.pack()
```

```
entry2 = Entry(frame2)
entry2.pack(side=RIGHT)
```

```
label2 = Label(frame2, text="Apellidos")
label2.pack(side=LEFT)
```

Pero aún no acaba de salir bien. ¿Entonces cómo podemos posicionar bien estos widgets? La respuesta es utilizando una disposición cuadrícula, otra de las formas de tkinter para distribuir automáticamente los widgets.

Para hacerlo, en lugar del pack() utilizaremos grid() e indicaremos una fila y una columna para cada widget (borramos los frames):



```
label = Label(root, text="Nombre")
label.grid(row=0,column=0)

entry = Entry(root)
entry.grid(row=0,column=1)

label2 = Label(root, text="Apellidos")
label2.grid(row=1,column=0)

entry2 = Entry(root)
entry2.grid(row=1,column=1)
```

De esta forma no importa cuán larga sea una label, todo se posicionará siguiente la cuadrícula y cada widget quedará perfectamente separado de los otros. Quizá el problema que tenemos es que no tenemos las etiquetas justificadas al mismo lado y queda un poco raro. Aquí es donde entra el parámetro sticky (pegado) de grid():





Incluso podríamos acabar de afinar la separación indicando un padding en la grid, aunque este parámetro está en todos los widgets:

```
label = Label(root, text="Nombre")  
label.grid(row=0, column=0, sticky=W)
```

```
entry = Entry(root)  
entry.grid(row=0, column=1)
```

```
label2 = Label(root, text="Apellidos")  
label2.grid(row=1, column=0, sticky=W)
```

```
entry2 = Entry(root)  
entry2.grid(row=1, column=1)
```

```
label = Label(root, text="Nombre")  
label.grid(row=0, column=0, sticky=W, padx=5, pady=5)
```

```
entry = Entry(root)  
entry.grid(row=0, column=1, padx=5, pady=5)
```

```
label2 = Label(root, text="Apellidos")  
label2.grid(row=1, column=0, sticky=W, padx=5, pady=5)
```

```
entry2 = Entry(root)  
entry2.grid(row=1, column=1, padx=5, pady=5)
```

justify=LEFT, CENTER, RIGHT	# justificar el texto
state=DISABLED, NORMAL	# desactivar el campo
show="*"	# para contraseñas mostrar * o lo que sea



```
from tkinter import *
```

```
# Configuración de la raíz  
root = Tk()
```

```
label = Label(root, text="Nombre muy largo")  
label.grid(row=0, column=0, sticky="w", padx=5, pady=5)
```

```
entry = Entry(root)  
entry.grid(row=0, column=1, padx=5, pady=5)  
entry.config(justify="right", state="normal")
```

```
label2 = Label(root, text="Contraseña")  
label2.grid(row=1, column=0, sticky="w", padx=5, pady=5)
```

```
entry2 = Entry(root)  
entry2.grid(row=1, column=1, padx=5, pady=5)  
entry2.config(justify="center", show="?")
```

```
# Finalmente bucle de la aplicación  
root.mainloop()
```



## Widget Text (Texto largo)

Por otro lado, si lo que necesitamos es trabajar con un campo de texto multilínea, podemos utilizar un Widget llamado Text:

```
from tkinter import *  
root = Tk()  
  
texto = Text(root)  
texto.pack()  
  
root.mainloop()
```

Como véis por defecto ocupa un espacio predefinido, pero podríamos establecer una altura y anchura en caracteres, no píxeles:

```
texto.config(width=30, height=10)
```

También acepta propiedades visuales para los colores o la fuente:

```
texto.config(font=("Consolas", 12), selectbackground="red", padx=5, pady=5)
```



```
from tkinter import *
```

```
# Configuración de la raíz
```

```
root = Tk()
```

```
texto = Text(root)
```

```
texto.pack()
```

```
texto.config(width=30, height=10, font=("Consolas",12),  
              padx=15, pady=15, selectbackground="red")
```

```
# Finalmente bucle de la aplicación
```

```
root.mainloop()
```



## Widget Button (Botón)

A partir de esta unidad todo se pone más interesante, porque gracias a los botones vamos a añadir comportamientos dinámicos a nuestras interfaces.

Comenzando por lo esencial, vamos a crear un botón:

```
# Podemos crearlos y empacarlos en una línea
Button(root, text="Clícame").pack()
```

Vamos a darle funcionalidad:

```
from tkinter import *

# Definimos una función a ejecutar al clic el botón
def hola():
    print("Hola mundo!")

root = Tk()

# Enlazamos la función a la acción del botón
Button(root, text="Clícame", command=hola).pack()

root.mainloop()
```

La verdad es que podemos realizar mil cosas jugando con los botones y otros widgets, vamos a crear una interfaz para realizar una suma a partir de dos campos y un tercero no editable para el resultado:

```
from tkinter import *

# Funciones backend
def borrar():
    n1.set('')
    n2.set('')

def sumar():
    r.set( float( n1.get() ) + float(n2.get() ) )
    borrar()
```



```
# Estructura del formulario
root = Tk()
root.config(bd=15) # borde exterior de 15 píxeles, queda mejor

# Tres StringVar para manejar los números y el resultado
n1 = StringVar()
n2 = StringVar()
r = StringVar()

Label(root, text="Numero 1").pack()
Entry(root, justify=CENTER, textvariable=n1).pack()

Label(root, text="\nNumero 2").pack()
Entry(root, justify=CENTER, textvariable=n2).pack()

Label(root, text="\nResultado").pack()
Entry(root, justify=CENTER, state=DISABLED, textvariable=r).pack()

Label(root).pack() # Separador

Button(root, text="Sumar", command=sumar).pack()

root.mainloop()
```

## Widget Radiobutton

Otro componente básico de los formularios son los **botones radiales**. Se utilizan cuando quieres ofrecerle al usuario la posibilidad de elegir una opción entre varias:

```
from tkinter import *

def selec():
    monitor.config(text = "Opción {}".format(opcion.get() ) )

root = Tk()
root.config(bd=15)


opcion = IntVar() # Como StrinVar pero en entero

Radiobutton(root, text="Opción 1", variable=opcion,
             value=1, command=selec).pack()
Radiobutton(root, text="Opción 2", variable=opcion,
             value=2, command=selec).pack()
Radiobutton(root, text="Opción 3", variable=opcion,
             value=3, command=selec).pack()

monitor = Label(root)
monitor.pack()

root.mainloop()
```





Si quisiéramos reiniciar el formulario podríamos añadir un botón y establecer los valores iniciales:

```
def reset():  
    opcion.set(None)           # Reiniciamos el seleccionable  
    monitor.config(text='')    # Reiniciamos la etiqueta
```

```
Button(root, text="Reiniciar", command=reset).pack()
```



## Widget Checkbutton

Con los radiobutton vimos que el usuario puede marcar una opción de entre varias, pero si queremos simplemente proponer una única opción es mejor utilizar un botón de selección. Son bastante parecidos.

```
from tkinter import *

root = Tk()
root.config(bd=15)

leche = IntVar()      # 1 si, 0 no
azucar = IntVar()     # 1 si, 0 no

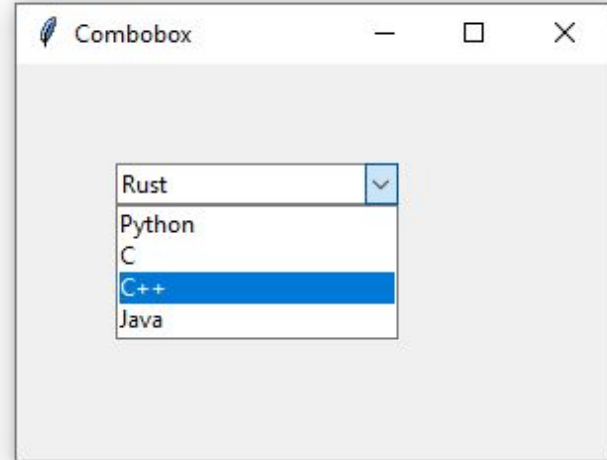
Label(root, text="¿Cómo quieres el café?").pack()
Checkbutton(root, text="Con leche", variable=leche,
             onvalue=1, offvalue=0).pack()
Checkbutton(root, text="Con azúcar", variable=azucar,
             onvalue=1, offvalue=0).pack()

root.mainloop()
```

## Lista desplegable

En la jerga del desarrollo de interfaces de usuario se lo conoce como *Combobox*, pues se trata de una **combinación** entre una **lista** y una caja de texto. Así, puede actuar como una lista desplegable con determinadas opciones y eventualmente permitir al usuario escribir un valor que no se encuentre en la lista.

El control fue introducido en Tk 8.5 y es provisto a través de la clase `ttk.Combobox`, la cual hereda de `ttk.Entry`, por lo que todas las operaciones aplicables a una **caja de texto** también aplican para un *combobox*.





```
from tkinter import ttk
import tkinter as tk
```

```
main_window = tk.Tk()
main_window.config(width=300, height=200)
main_window.title("Combobox")
combo = ttk.Combobox()
combo.place(x=50, y=50)
main_window.mainloop()
```

```
from tkinter import ttk
import tkinter as tk
```

```
class Application(ttk.Frame):
```

```
    def __init__(self, main_window):
        super().__init__(main_window)
        main_window.title("Combobox")
        self.combo = ttk.Combobox(self)
        self.combo.place(x=50, y=50)
        main_window.config(width=300, height=200)
        self.place(width=300, height=200)
```

```
main_window = tk.Tk()
app = Application(main_window)
app.mainloop()
```



Para evitar que el usuario ingrese sus propios valores, es decir, para que el control se comporte únicamente como una lista desplegable y no como una caja de texto, al momento de la creación utilizamos:

```
combo = ttk.Combobox(  
    state="readonly",  
    values=["Python", "C", "C++", "Java"]  
)
```

Un *combobox* en Tk trabaja únicamente con cadenas. Cualquier otro tipo de objeto será convertido antes de ser añadido a la lista

El método `Combobox.get()` retorna la opción seleccionada: ya haya sido esta seleccionada a partir de una de las opciones, ya haya sido escrita manualmente por el usuario (si no se deshabilitó esta opción vía `state="readonly"`).

```

from tkinter import messagebox, ttk
import tkinter as tk

def show_selection():
    # Obtener la opción seleccionada.
    selection = combo.get()
    messagebox.showinfo(
        message=f"La opción seleccionada es:
{selection}",
        title="Selección"
    )

main_window = tk.Tk()
main_window.config(width=300, height=200)
main_window.title("Combobox")
combo = ttk.Combobox(
    state="readonly",
    values=["Python", "C", "C++", "Java"]
)
combo.place(x=50, y=50)
button = ttk.Button(text="Mostrar selección",
command=show_selection)
button.place(x=50, y=100)
main_window.mainloop()

```

```

from tkinter import messagebox, ttk
import tkinter as tk

class Application(ttk.Frame):

    def __init__(self, main_window):
        super().__init__(main_window)
        main_window.title("Combobox")
        self.combo = ttk.Combobox(
            self,
            state="readonly",
            values=["Python", "C", "C++", "Java"]
        )
        self.combo.place(x=50, y=50)
        self.button = ttk.Button(
            text="Mostrar selección",
            command=self.show_selection
        )
        self.button.place(x=50, y=100)
        main_window.config(width=300, height=200)
        self.place(width=300, height=200)

    def show_selection(self):
        # Obtener la opción seleccionada.
        selection = self.combo.get()
        messagebox.showinfo(
            message=f"La opción seleccionada es: {selection}",
            title="Selección"
        )

main_window = tk.Tk()
app = Application(main_window)
app.mainloop()

```



Si no hay ningún elemento seleccionado o si el elemento no se encuentra entre las opciones provistas (esto es, ha sido escrito por el usuario) el valor de retorno es -1.

Análogamente, el método `set()` establece el contenido de la lista.

```
combo.set("Python")
```

Para seleccionar un elemento según su posición o índice se utiliza asimismo el método `current()`, pero pasándole como argumento el elemento que debe ser seleccionado:

```
combo.current(2)
```

Para obtener todas las opciones de la lista:

```
values = combo["values"]
```



El control `ttk.Combobox` introduce un nuevo evento llamado `<<ComboboxSelected>>` que es enviado cuando cambia la opción seleccionada. De esta manera, es posible asociar una función a dicho evento para que sea ejecutada cada vez que el usuario cambia la opción de la lista. Por ejemplo:

```
from tkinter import messagebox, ttk
import tkinter as tk

def selection_changed(event):
    selection = combo.get()
    messagebox.showinfo(
        title="Nuevo elemento seleccionado",
        message=selection
    )

main_window = tk.Tk()
main_window.config(width=300, height=200)
main_window.title("Combobox")
combo = ttk.Combobox(values=["Python",
"C", "C++", "Java"])
combo.bind("<<ComboboxSelected>>",
selection_changed)
combo.place(x=50, y=50)
main_window.mainloop()
```

```
from tkinter import messagebox, ttk
import tkinter as tk

class Application(ttk.Frame):
    def __init__(self, main_window):
        super().__init__(main_window)
        main_window.title("Combobox")
        self.combo = ttk.Combobox(
            self,
            values=["Python", "C", "C++", "Java"]
        )
        self.combo.bind("<<ComboboxSelected>>", self.selection_changed)
        self.combo.place(x=50, y=50)
        main_window.config(width=300, height=200)
        self.place(width=300, height=200)

    def selection_changed(self, event):
        selection = self.combo.get()
        messagebox.showinfo(
            title="Nuevo elemento seleccionado",
            message=selection
        )

main_window = tk.Tk()
app = Application(main_window)
app.mainloop()
```



## Widget Menu

El control `tk.Menu` permite añadir una barra de menús a la ventana principal (`tk.Tk`) o a una ventana secundaria (`tk.Toplevel`) en una aplicación de escritorio de Tk. Los menús de una ventana contienen un texto y/o una imagen y pueden ser asociados a funciones para responder ante la presión del usuario.

Para implementar una funcionalidad como esta, lo primordial es crear una barra de menús vía la clase `tk.Menu` y configurarla en la ventana principal.

Una barra de menús vacía no es de mucha utilidad: ni siquiera se ve en la ventana.

```
import tkinter as tk
```

```
ventana = tk.Tk()
ventana.title("Barra de menús en Tk")
ventana.config(width=400, height=300)
# Crear una barra de menús.
barra_menus = tk.Menu()
# Insertarla en la ventana principal.
ventana.config(menu=barra_menus)
ventana.mainloop()
```




Agreguémosle el primer menú de este artículo, con el título «Archivo», como suele aparecer en muchas aplicaciones de escritorio.

Cada menú que queramos ubicar en la barra de menús también se crea usando la clase `tk.Menu`. De modo que ahora tenemos dos instancias: `barra_menus`, el contenedor de todos los menús de la ventana, y `menu_archivo`, menú al cual en seguida agregaremos algunos botones.

Para agregar menús a una barra de menús se utiliza el método `add_cascade()`, que recibe como argumento el menú (`menu`) para ser insertado y el texto (`label`) con que se quiere mostrar.

```
import tkinter as tk

ventana = tk.Tk()
ventana.title("Barra de menús en Tk")
ventana.config(width=400, height=300)
barra_menus = tk.Menu()
# Crear el primer menú.
menu_archivo = tk.Menu(barra_menus, tearoff=False)
# Agregarlo a la barra.
barra_menus.add_cascade(menu=menu_archivo,
                        label="Archivo")
ventana.config(menu=barra_menus)
ventana.mainloop()
```




Nótese que en la creación de `menu_archivo` (línea 8) pasamos como primer argumento la barra de menús dentro de la cual queremos ubicarlo. El segundo argumento `tearoff=False` evita que Tk agregue una funcionalidad para desacoplar el menú de la ventana.

Ahora ya podemos visualizar nuestra barra de menús con el único menú *Archivo*, pero sin botón alguno. Para añadir un botón o una opción a un menú utilizamos el método `add_command()`.

```
import tkinter as tk

def archivo_nuevo_presionado():
    print("¡Has presionado para crear un nuevo archivo!")

ventana = tk.Tk()
ventana.title("Barra de menús en Tk")
ventana.config(width=400, height=300)
barra_menus = tk.Menu()
menu_archivo = tk.Menu(barra_menus, tearoff=False)
menu_archivo.add_command(
    label="Nuevo",
    accelerator="Ctrl+N",
    command=archivo_nuevo_presionado
)
barra_menus.add_cascade(menu=menu_archivo,
    label="Archivo")
ventana.config(menu=barra_menus)
ventana.mainloop()
```



Aquí insertamos en nuestro menú un botón con texto «Nuevo» (argumento `label`) y un indicador de atajo del teclado «Ctrl+N» (`accelerator`). El argumento `command` funciona de la misma manera que el parámetro homónimo en los `botones`: recibe el nombre de una función que será invocada cuando el usuario presione sobre esa opción del menú.

```
import tkinter as tk

def archivo_nuevo_presionado(event=None):
    print("¡Has presionado para crear un nuevo archivo!")

ventana = tk.Tk()
ventana.title("Barra de menús en Tk")
ventana.config(width=400, height=300)
barra_menus = tk.Menu()
menu_archivo = tk.Menu(barra_menus, tearoff=False)
menu_archivo.add_command(
    label="Nuevo",
    accelerator="Ctrl+N",
    command=archivo_nuevo_presionado
)
# Asociar el atajo del teclado del menú "Nuevo".
ventana.bind_all("<Control-n>", archivo_nuevo_presionado)
barra_menus.add_cascade(menu=menu_archivo, label="Archivo")
ventana.config(menu=barra_menus)
ventana.mainloop()
```



Además de un texto, el botón de un menú puede tener una imagen. La configuración opera de forma similar a la de **una imagen en un botón**. Se crea una instancia de `tk.PhotoImage` con el nombre del archivo y luego se pasa como argumento al método `add_command()`. El argumento `compound` especifica en qué lugar debe aparecer la imagen respecto del texto.

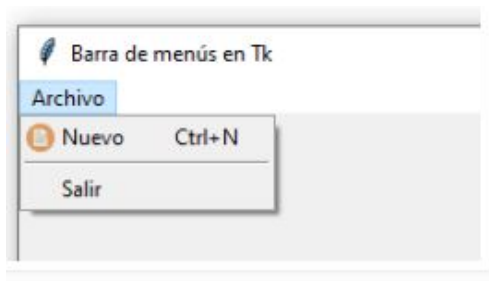
```
import tkinter as tk

def archivo_nuevo_presionado(event=None):
    print("¡Has presionado para crear un nuevo archivo!")

ventana = tk.Tk()
ventana.title("Barra de menús en Tk")
ventana.config(width=400, height=300)
barra_menus = tk.Menu()
menu_archivo = tk.Menu(barra_menus, tearoff=False)
img_menu_nuevo = tk.PhotoImage(file="nuevo_archivo.png")
menu_archivo.add_command(
    label="Nuevo",
    accelerator="Ctrl+N",
    command=archivo_nuevo_presionado,
    image=img_menu_nuevo,
    # Indicar que la imagen debe aparecer a la izquierda del texto.
    compound=tk.LEFT
)
ventana.bind_all("<Control-n>", archivo_nuevo_presionado)
barra_menus.add_cascade(menu=menu_archivo,
    label="Archivo")
ventana.config(menu=barra_menus)
ventana.mainloop()
```

Ahora bien, de seguro un menú tendrá más de un botón, así que podemos invocar `add_command()` tantas veces como sea necesario. Las opciones de un menú aparecen en la interfaz en el mismo orden en que fueron añadidas en el código. Agreguemos un nuevo botón al `menu_archivo` para cerrar el programa.

El método `add_separator()` introduce una línea horizontal que es útil para separar grupos de botones de menús relacionados. El resultado es el siguiente.



```
import tkinter as tk

def archivo_nuevo_presionado(event=None):
    print("¡Has presionado para crear un nuevo archivo!")

ventana = tk.Tk()
ventana.title("Barra de menús en Tk")
ventana.config(width=400, height=300)
barra_menus = tk.Menu()
menu_archivo = tk.Menu(barra_menus, tearoff=False)
img_menu_nuevo = tk.PhotoImage(file="nuevo_archivo.png")
menu_archivo.add_command(
    label="Nuevo",
    accelerator="Ctrl+N",
    command=archivo_nuevo_presionado,
    image=img_menu_nuevo,
    compound=tk.LEFT
)
ventana.bind_all("<Control-n>", archivo_nuevo_presionado)
menu_archivo.add_separator()
menu_archivo.add_command(label="Salir",
    command=ventana.destroy)
barra_menus.add_cascade(menu=menu_archivo, label="Archivo")
ventana.config(menu=barra_menus)
ventana.mainloop()
```

