

PROGRAMACIÓN AVANZADA CON PYTHON

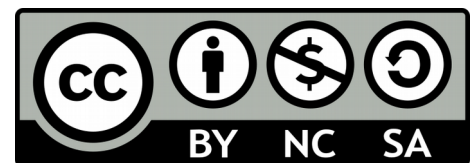
(CEFIRE CTEM)



Interfaces gráficas con Tkinter

Esta obra está sujeta a la licencia Reconocimiento-NoComercial-CompartirIgual 4.0 Internacional de Creative Commons. Para ver una copia de esta licencia, visitad

<http://creativecommons.org/licenses/by-nc-sa/4.0/>.



Autora: María Paz Segura Valero (segura_marval@gva.es)

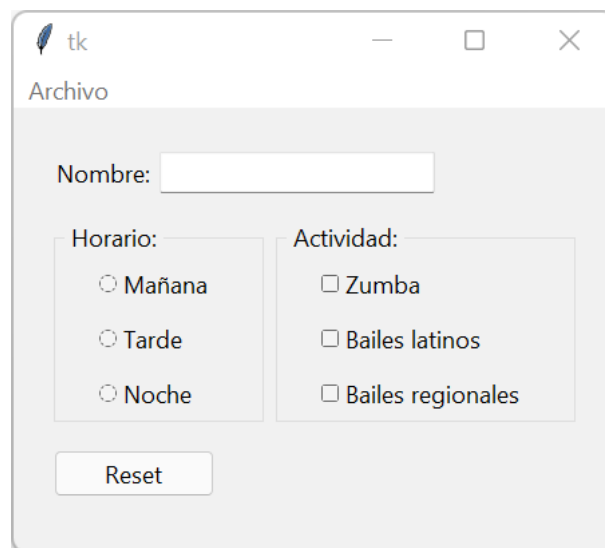
CONTENIDO

1. Introducción.....	2
2.1. Comprueba que tienes Tkinter en tu ordenador.....	2
3. Importar el módulo tkinter.....	3
4. Componentes de la interfaz gráfica.....	4
5. Raíz o ventana principal.....	5
6. Widgets hijos.....	7
6.1. Etiqueta.....	7
6.1.1. Uso del widget.....	10
6.2. Campo de entrada.....	11
6.2.1. Uso del widget.....	13
6.3. Campo multilíneas.....	13
6.3.1. Uso del widget.....	15
6.4. Botones.....	16
6.5. Radiobutton.....	18
6.5.1. Uso del widget.....	19
6.6. Checkbutton.....	19
6.6.1. Uso del widget.....	20
6.7. Menú.....	20
6.8. Ventana emergente.....	23
6.8.1. Módulo <i>messagebox</i>	23
6.8.2. Módulo <i>filedialog</i>	25
6.9. Contenedores.....	27
6.9.1. Contenedor <i>Frame</i>	27
6.9.2. Contenedor <i>LabelFrame</i>	29
7. Gestores de geometría.....	29
7.1. Gestor <i>pack()</i>	30
7.1.1. Parámetros <i>fill</i> y <i>expand</i>	30
7.1.2. Parámetros <i>ipadx</i> e <i>ipady</i>	31
7.1.3. Parámetro <i>side</i>	31
8. Utilizando una clase específica.....	32
9. Fuentes de información.....	33
10. Créditos de imágenes.....	34

1. Introducción

Hasta ahora hemos estado creando programas que interactuaban con los usuarios en modo texto. Es decir, el programa utilizaba una consola para lanzar preguntas al usuario y para que éste le proporcionase la información necesaria.

Pero en Python también podemos crear programas que utilicen interfaces gráficas de usuario compuestas por ventanas, cajas de texto, botones, etc.



Para ello debemos utilizar alguna librería creada con dicho propósito. Una de las más populares en Python es **Tkinter**, entre otras cosas porque viene integrada por defecto en el lenguaje.

Se trata de una capa orientada a objetos de la librería **Tcl/Tk** donde **Tcl** es un sencillo lenguaje de programación de propósito general y **Tk** un conjunto de herramientas para crear aplicaciones gráficas de usuario.

2.1. Comprueba que tienes Tkinter en tu ordenador

Hemos dicho que la librería *Tkinter* se encuentra integrada en el intérprete de comandos. Si quieres comprobar que es así, puedes ejecutar estos comandos desde una consola/terminal:

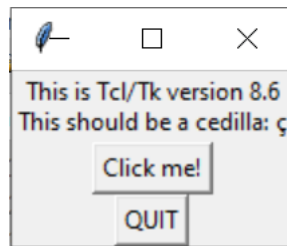
- En Windows ejecuta:

```
python -m tkinter
```

- En Linux ejecuta:

```
python -m Tkinter
```

En cualquier caso, se debería abrir una ventana que muestre una interfaz *Tk* simple similar a la siguiente:



En ella se indica también qué versión de *Tcl/Tk* está instalada. Así podemos leer la documentación de *Tcl/Tk* específica de dicha versión.

3. Importar el módulo tkinter

Para poder trabajar con interfaces gráficas de usuario Tkinter/tkinter, podemos importar la librería de varias formas. Ésta es una de ellas:

```
import tkinter as tk
```

Lo normal es que estés trabajando con una versión de Python reciente y funcione el código anterior pero si estuvieses trabajando con una versión de Python más antigua, necesitarás importar la librería de la siguiente forma:

```
import Tkinter as tk
```

Para no tener problemas de compatibilidad a la hora de utilizar la librería, podemos utilizar el siguiente código:

```
1 try:
2     import Tkinter as tk
3 except ImportError:
4     import tkinter as tk
```

Además, desde la versión 8.5 de Tk, esta librería incluye nuevas versiones para algunos de los *widgets* (elementos de la interfaz) más habituales, también llamados *widgets tematizados*. Si queremos que la definición de las dos versiones de widgets estén habilitadas en el entorno, deberemos realizar la importación de la librería así:

```
1 try:
2     import Tkinter as tk
3     from Tkinter import ttk
4 except ImportError:
5     import tkinter as tk
6     from tkinter import ttk
```

4. Componentes de la interfaz gráfica

Una interfaz gráfica de *Tkinter* está compuesta por un conjunto de componentes o *widgets* que se corresponden con objetos de clases específicas de la librería *Tkinter*. Por ejemplo: `tkinter.Frame`, `tkinter.Button`, `tkinter.Checkbutton`, etc.

El *widget* principal es la raíz o *root*. Se trata de la **ventana principal** de la aplicación y, por tanto, es el primer *widget* que debe crearse. Su aspecto dependerá del sistema operativo en el que se ejecute el programa:

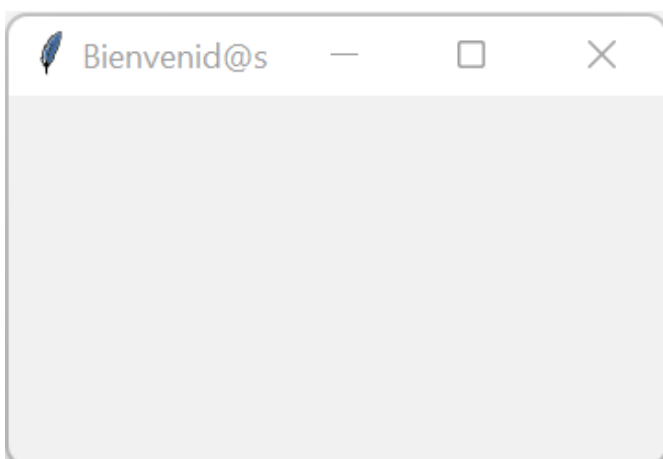


Figura 1: En Windows

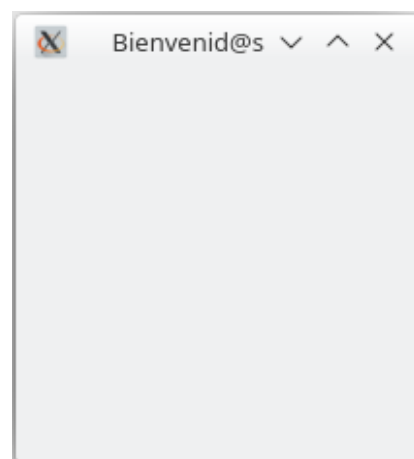
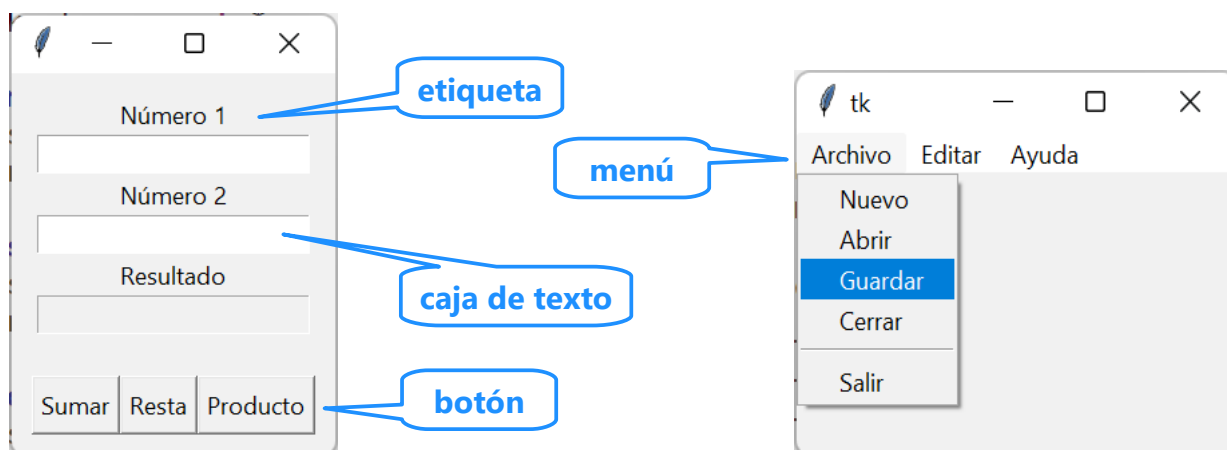


Figura 2: En Lliurex

También existen otros tipos de *widgets* como pueden ser etiquetas, cajas de texto, botones, menús, etc.



Estos *widgets* pueden alojarse directamente en la ventana principal (por defecto) o dentro de *widgets* contenedores (por ejemplo: `Frame`, `Labelframe`) formando así una jerarquía de *widgets* cuya raíz será la ventana principal de la aplicación.

La forma de ubicar los widgets en la raíz o en los contenedores viene determinada por el **Gestor de Geometría** (*Geometry Management*). Existen varios tipos, como podemos ver en la siguiente tabla:

Gestor	Descripción
<code>grid()</code>	Ubica los elementos basándose en una parrilla o tabla e indicando, para cada uno de ellos, la fila y columna que debe ocupar dentro del contenedor.
<code>pack()</code>	Ubica los elementos según la configuración que se utilice. Existen varias: <code>fill</code> , <code>expand</code> , <code>side</code> , <code>ipadx</code> , <code>ipady</code> , <code>padx</code> , y <code>pady</code> . Es recomendable para ubicar los elementos de arriba a abajo (en forma de pila) o cuando queremos ubicarlos contiguos de izquierda a derecha.
<code>place()</code>	Ubica los elementos indicando las coordenadas de cada uno de ellos, es decir, indicando el valor para su posición horizontal (X) y vertical (Y) dentro del contenedor. Es el gestor menos utilizado pero puede ser útil cuando permitimos que el usuario determine la posición de los elementos en pantalla.

Veremos el método **`pack()`** con más detalle, más adelante.

5. Raíz o ventana principal

Como hemos dicho, la *raíz* (*root*) o ventana principal es el primer widget que debe crearse. Se trata del contenedor del resto de widgets y su tamaño se adapta al tamaño de los *widgets* que contiene.

Para crear la raíz de nuestra aplicación debemos seguir los siguientes pasos:

1. Crear un objeto de la clase `Tk()`. Habitualmente se almacena en una variable llamada `root`, pero puede tener otro nombre.
2. Inicializar los parámetros de la raíz o dejar por defecto.
3. Lanzar el bucle `mainloop()` que se encarga de atender los eventos generados durante la ejecución y actualizar la interfaz de usuario.

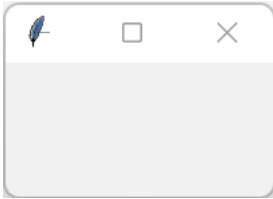
Veamos un ejemplo:

```
1-raíz.py ×
1  try:
2      import Tkinter as tk
3      from Tkinter import ttk
4  except ImportError:
5      import tkinter as tk
6      from tkinter import ttk
7
8
9  #Creamos la raíz "root"
10 root = tk.Tk()
11
12 #Configuramos la raíz "root"
13 root.title("Bienvenid@s") #título de la ventana
14 root.geometry('500x150+250+250') #anchoxalto+x+y
15
16 #Bucle de la aplicación
17 root.mainloop()
```

Con este parámetro podemos indicar el tamaño de la ventana.

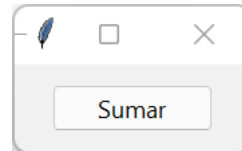
Figura 3: Creamos la ventana principal de la aplicación.

A parte del título, pueden configurarse más parámetros de la ventana principal. Algunos ejemplos son los que se muestran a continuación:

Parámetro	Descripción
geometry	<p>Podemos indicar el tamaño de la ventana pasándole una cadena de texto con la siguiente <u>sintaxis</u>:</p> <p style="text-align: center;">"anchoxalto+coordenada_x+coordenada_y"</p> <p><u>Ejemplo</u>:</p> <pre>13 root = tk.Tk() 14 root.geometry("200x100+50+50")</pre> 
bd	<p>Permite especificar el ancho del borde de un widget. Si lo utilizamos en la raíz, nos permite controlar el margen que habrá entre el borde de la ventana y los <i>widgets</i> que contendrá.</p>

Ejemplo:

```
13 root = tk.Tk()  
14 root.config(bd=15)
```



En la siguiente página web puedes encontrar más información:

- Tkinter Window: <https://www.pythontutorial.net/tkinter/tkinter-window/>

Puedes ver un ejemplo completo en el siguiente fichero: **1-raíz.py**

6. Widgets hijos

Existen muchos *widgets* que podemos utilizar en la interfaz gráfica. En este apartado presentaremos algunos de los más utilizados.

Recuerda que para muchos de ellos, existe una versión antigua y una más nueva. Si queremos utilizar la versión antigua utilizaremos la versión de **tk** y sino la versión **ttk**.

6.1. Etiqueta

Las etiquetas sirven para mostrar texto o imágenes al usuario y pertenecen a la clase **Label**.

Cuando se crea una etiqueta, se debe indicar su contenedor o *widget* padre y una serie de parámetros de configuración. Además, debemos seleccionar el gestor de geometría que queremos utilizar.

Ejemplo: Creamos una etiqueta con el texto "PAISAJE CAMPESTRE" y la guardamos en una variable llamada `etiqueta`. En la instrucción siguiente, indicamos que vamos a utilizar el gestor de geometría `pack()`.

```
12 etiqueta = ttk.Label(root, text="PAISAJE CAMPESTRE")  
13 etiqueta.pack()
```

También podríamos haber compactado las dos instrucciones en una sola.

```
15 etiqueta = ttk.Label(root, text="PAISAJE CAMPESTRE").pack()
```

O, si no vamos a utilizar la etiqueta para nada más, quitar la variable:


```
17 ttk.Label(root, text="PAISAJE CAMPESTRE").pack()
```

En cualquier caso, el aspecto sería el siguiente:

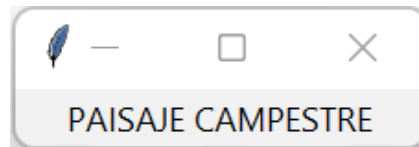


Figura 4: Usando la clase **Label**

Ejemplo: Podemos cambiar el tipo de letra del texto (`font`), su color (`foreground`) y el color de fondo (`background`).

```
26 ttk.Label(root, text="PAISAJE CAMPESTRE",  
27             font= ("Verdana",24),  
28             foreground= "white",  
29             background= "grey"  
30             ).pack()
```



Ejemplo: Si la etiqueta ya existiese, podríamos modificar su configuración a través de la función **config()**.

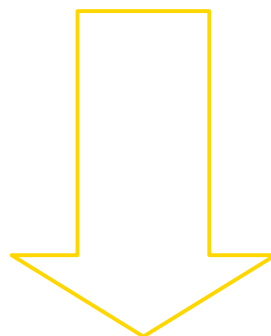
```
32 #Si la etiqueta ya existe, podemos cambiar la configuración con config()  
33 etiqueta.config(font= ("Verdana",24),  
34                 foreground= "white",  
35                 background= "grey")
```

Si, en lugar de un texto, queremos mostrar una imagen en la etiqueta entonces debemos utilizar un objeto de la clase **PhotoImage** para crear un objeto imagen y luego pasarlo como argumento a la clase **Label**.

```
14 imagen = tk.PhotoImage(file="winter.png")  
15 ttk.Label(root, image=imagen).pack()
```

Pasamos el nombre del
fichero de imagen.

Utilizamos el parámetro `image` para pasar el objeto que contiene la imagen.



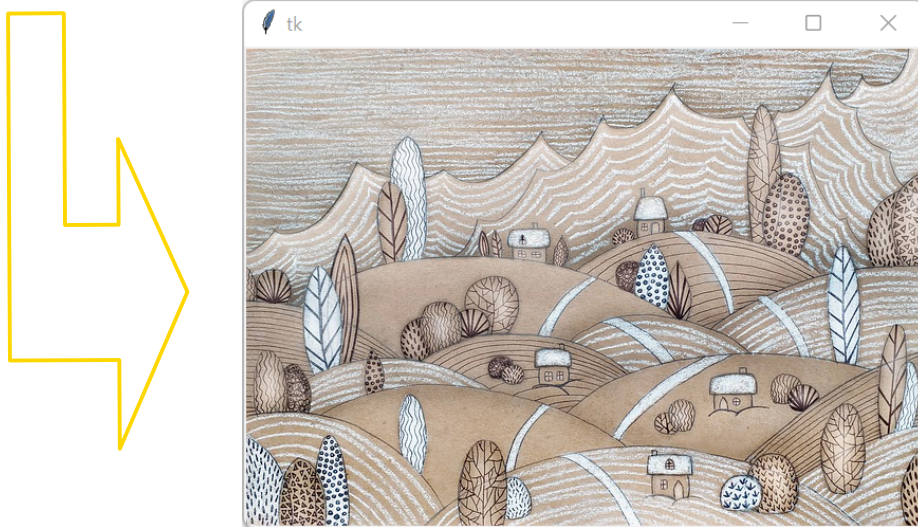


Figura 5: Usamos **Label** y **PhotoImage**

Cosas a tener en cuenta:

- Si la imagen se encuentra en una carpeta diferente de la del programa de Python, deberemos indicar la ruta completa en el parámetro `file`.
- La clase **PhotoImage** solo acepta los formatos de imagen `pgm`, `ppm`, `gif` y `png`. Si queremos utilizar otras extensiones deberemos instalar y hacer uso del *módulo PIL* (Pillow). Para más información, visita la siguiente página: <https://www.pythontutorial.net/tkinter/tkinter-photoimage/>

También podríamos crear una etiqueta que contuviese un texto y una imagen a la vez. En ese caso deberíamos informar los parámetros `text`, `image` y `compound`. Éste último para indicar la manera de disponer el texto y la imagen dentro de la etiqueta.

```

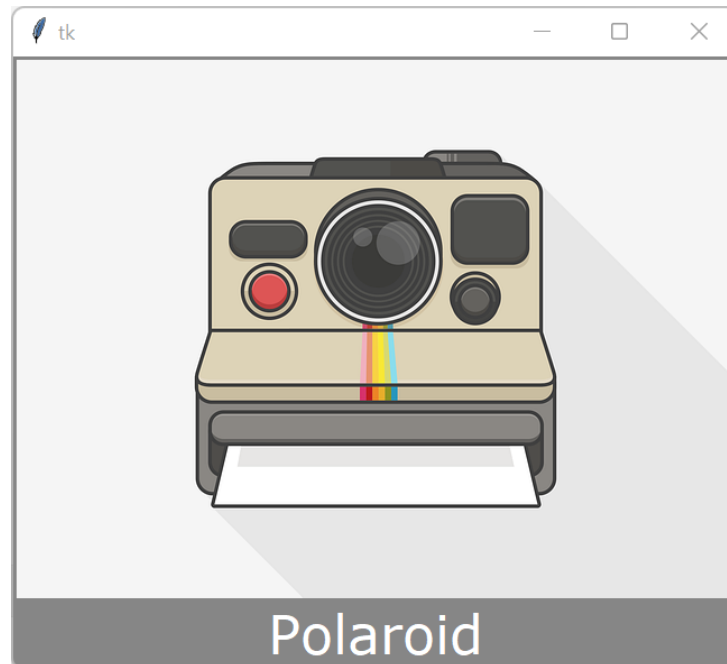
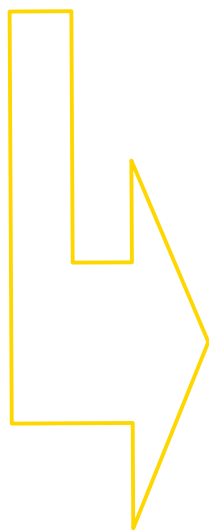
19 imagen = tk.PhotoImage(file="polaroid.png")
20 ttk.Label(root, text="Polaroid",
21           font= ("Verdana",24),
22           foreground= "white",
23           background= "grey",
24           image=imagen,
25           compound="top").pack()

```

Informamos los parámetros del texto

Informamos los parámetros de la imagen

Queremos que la imagen aparezca arriba del texto.
Los valores pueden ser: `bottom`, `left`, `right` o `top`.



Puedes ver [ejemplos completos](#) en los siguientes ficheros: **2-Label_texto.py** y **3-Label_imagen.py**

6.1.1. Uso del widget

En los ejemplos anteriores hemos creados etiquetas que tenían un texto concreto pero, ¿y si queremos cambiar dicho texto durante la ejecución del programa?

Una manera habitual de hacerlo es creando un objeto de tipo **StringVar** y asociándolo a la etiqueta a través del atributo **textvariable**. De esta forma, cada vez que cambiemos el valor del objeto **StringVar** asociado, cambiará automáticamente el valor de la etiqueta correspondiente.

Ejemplo: Podemos asociar el objeto **StringVar** en el momento de crear la etiqueta o añadirlo después gracias a la función **config()** que nos permite modificar la configuración de un widget.

```
21 # Variables dinámicas
22 texto_etiqueta1 = tk.StringVar()
23 texto_etiqueta1.set("Bienvenid@s")
24 etiqueta.config(textvariable = texto_etiqueta1)
```

Creamos el objeto *StringVar*.

Indicamos el texto a mostrar a través de la función *set*.

Asociamos el *StringVar* a la etiqueta.

Si quisiésemos recuperar el valor del widget asociado a la variable **texto_etiqueta1** podríamos utilizar la función **get()**, de la siguiente forma:

```
26 texto_etiqueta1.get()
```

Esta forma de asociar variables dinámicas a los objetos **Label** es extrapolable a otros widgets tal y como veremos en los siguientes apartados.

Por último, cabe señalar que la clase **StringVar** es una variable de control de tipo carácter pero existen otras similares que se pueden en la siguiente tabla:

Gestor	Descripción
IntVar	Para variables de tipo entero.
DoubleVar	Para variables de coma flotante.
BooleanVar	Para variables de tipo lógico.

Para modificar la imagen que se muestra en una etiqueta debemos utilizar la propiedad `config(file=nombre_fichero)`.

Ejemplo: Cambiamos la imagen de una etiqueta que ya está inicializada.

```
imagen = tk.PhotoImage(file="winter.png")
ttk.Label(root, image=imagen).pack()
imagen.config(file="polaroid.png")
```

Para conocer más sobre las variables de control, puedes visitar la siguiente página web: <https://python-para-impacientes.blogspot.com/2016/02/variables-de-control-en-tkinter.html>

6.2. Campo de entrada

La clase **Entry** de Tkinter nos permite crear cajas de texto o campos de entrada de una sola línea. Si, en lugar de eso, quisiésemos utilizar una caja de texto multilínea deberíamos utilizar la clase **Text**, explicado en el siguiente apartado.

Ejemplo: Creamos una caja para solicitar un nombre de usuario.

```
20 usuario_entry = ttk.Entry(root)
21 usuario_entry.pack()
```

Podemos configurar algunos parámetros de la caja de texto. Por ejemplo, podemos enmascarar los caracteres si la utilizamos para solicitar una contraseña.

```
29 contra_entry = ttk.Entry(root, show="*")
30 contra_entry.pack()
```

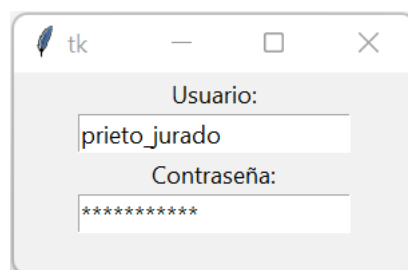
En el parámetro `show` indicamos el carácter que queremos utilizar para enmascarar el texto.

Si en nuestra interfaz vamos a utilizar varias cajas de texto, podemos decidir qué caja de texto será la primera que dispondrá del foco de la aplicación utilizando la función **focus()**.

```
20 usuario_entry = ttk.Entry(root)
21 usuario_entry.pack()
22 usuario_entry.focus()
```

Habitualmente se acompañan las cajas de texto con etiquetas para indicar el tipo de información que esperan. Veamos un ejemplo más completo:

```
1  try:
2      import Tkinter as tk
3      from Tkinter import ttk
4  except ImportError:
5      import tkinter as tk
6      from tkinter import ttk
7
8  # Configuración de la raíz
9  root = tk.Tk()
10 root.geometry('300x150+50+50') #anchoxalto+x+y
11
12 # usuario
13 usuario_label = ttk.Label(root, text="Usuario:")
14 usuario_label.pack()
15
16 usuario_entry = ttk.Entry(root)
17 usuario_entry.pack()
18 usuario_entry.focus()
19
20 # contraseña
21 contra_label = ttk.Label(root, text="Contraseña:")
22 contra_label.pack()
23
24 contra_entry = ttk.Entry(root, show="*")
25 contra_entry.pack()
```



Otros parámetros que podríamos utilizar serían los siguientes:

Parámetro	Descripción
<code>justify</code>	Para alinear el texto dentro del campo. Posibles valores: <code>left</code> , <code>center</code> , <code>right</code> . <u>Ejemplo:</u> Alineamos el texto a la derecha. <code>usuario_entry = ttk.Entry(root, justify='right')</code>
<code>state</code>	Para activar o desactivar el campo. Posibles valores: <code>DISABLED</code> , <code>NORMAL</code> . <u>Ejemplo:</u> Deshabilitamos el campo de contraseña. <code>otro_entry = ttk.Entry(root, state= 'disabled')</code>

Puedes ver un [ejemplo completo](#) en el siguiente fichero: **4-Entry.py**

6.2.1. Uso del widget

Al igual que con las etiquetas, podemos asociar objetos **StringVar** o similares a una caja de texto. De esta forma podremos tener control sobre el valor que contiene, modificarlo, validarlo, etc.

Ejemplo: Creamos un objeto **StringVar** para la caja de texto del nombre de usuario.

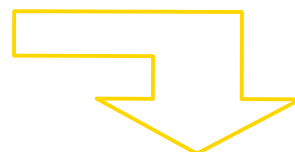
```
21 texto_usuario = tk.StringVar()
22 usuario_entry = ttk.Entry(root, textvariable = texto_usuario)
```

6.3. Campo multilíneas

En el apartado anterior hemos visto como trabajar con un campo de entrada de texto corto pero si necesitáramos mostrar un texto más largo o multilíneas entonces deberíamos trabajar con un objeto de la clase **Text**.

Ejemplo: Creamos un campo multilíneas básico.

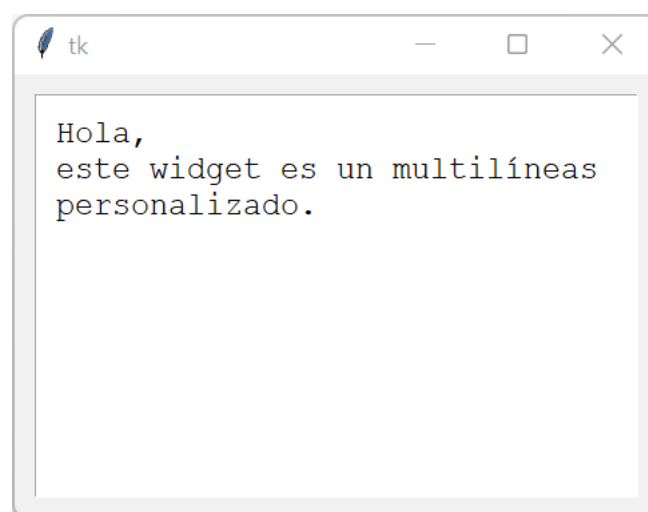
```
9 root = tk.Tk()
10 root.config(bd=15) #tamaño del borde
```





Ejemplo: Creamos un campo multilíneas personalizando algunos parámetros con la función **config()**.

```
12 texto = tk.Text(root)
13 texto.config(width=30, height=10, font=("Courier New",12),
14             padx=15, pady=15, selectbackground="grey")
15 texto.pack()
```



Los parámetros **width** y **height** nos permiten indicar el tamaño del widgets en número de caracteres (y no en píxeles).

Las características de la fuente de texto las configuramos con el parámetro **font** donde indicamos el tipo y tamaño de la misma.

Los parámetros **padx** y **pady** permiten que indiquemos el espacio en blanco que habrá entre el borde del widget y el texto que contenga.

Y, por último, con el parámetro **selectbackground** podemos configurar el color a utilizar cuando se seleccione un texto del widget.

Evidentemente, ésta es solo una muestra de los parámetros que podemos personalizar. Podéis encontrar más información en la siguiente página web: https://www.tutorialspoint.com/python/tk_text.htm

Puedes ver un ejemplo completo en el siguiente fichero: **5-Text.py**

6.3.1. Uso del widget

Veamos como podemos gestionar los objetos de tipo **Text** en un programa.

Acción	Descripción
Inicializar el valor	<p>Para inicializar el valor del <i>widget</i> debemos utilizar el método <code>insert()</code> que tiene esta <u>sintaxis</u>:</p> <pre>objeto_text.insert(posición, texto_a_mostrar)</pre> <p>donde <code>posición</code> es la línea y columna donde queremos empezar a escribir (formato: <code>'línea.columna'</code>) y <code>texto_a_mostrar</code> es el contenido que queremos que aparezca en el widget.</p> <p><u>Ejemplo</u>: Mostrar un texto desde la línea 1, columna 0.</p> <pre>texto = tk.Text(root) texto.insert('1.0', 'Hola a tod@s')</pre>
Recuperar el valor	<p>Para recuperar el valor del <i>widget</i> debemos utilizar el método <code>get()</code> con la siguiente <u>sintaxis</u>:</p> <pre>objeto_text.get(pos_inicial, pos_final)</pre>

	<p>donde <code>pos_inicial</code> es la línea y columna donde queremos empezar a leer (formato: <code>'línea.columna'</code>) y <code>pos_inicial</code> es la línea y columna donde queremos acabar. Si queremos que sea hasta el final entonces <code>pos_inicial</code> será <code>'end'</code>.</p> <p><u>Ejemplo:</u> Recuperamos el contenido completo del widget. <code>texto.get('1.0', 'end')</code></p> <p><u>Ejemplo:</u> Recuperamos una parte del contenido del widget. <code>texto.get('2.1', '3.0')</code></p>
Habilitar o deshabilitar	<p>Para habilitar o deshabilitar el widget debemos configurar la opción <code>state</code>, siendo sus valores posibles <code>normal</code> y <code>disabled</code>.</p> <p><u>Ejemplo:</u> Habilitamos el widget. <code>texto['state'] = 'normal'</code></p> <p><u>Ejemplo:</u> Deshabilitamos el widget. <code>texto['state'] = 'disabled'</code></p>
Borrar el contenido	<p>Para borrar el valor del <i>widget</i> debemos utilizar el método <code>delete()</code> con la siguiente <u>sintaxis</u>:</p> <pre>objeto_text.delete(pos_inicial, pos_final)</pre> <p>donde <code>pos_inicial</code> es la línea y columna donde queremos empezar a leer (formato: <code>'línea.columna'</code>) y <code>pos_inicial</code> es la línea y columna donde queremos acabar. Si queremos que sea hasta el final entonces <code>pos_inicial</code> será <code>'end'</code>.</p> <p><u>Ejemplo:</u> <code>texto.get('1.0', 'end')</code></p>

6.4. Botones

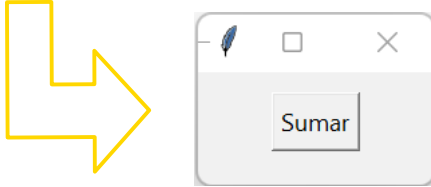
Los **botones** son *widgets* que muestran un texto o una imagen y ejecutan una acción cuando se clican sobre ellos. Además, podemos asociarles un atajo de teclado (*keyboard shortcut*).

Existen muchas opciones para configurar un botón pero aquí nos centraremos en las más habituales.

Ejemplo: Creamos un botón típico.

```
17 boton1 = ttk.Button(root, text="Sumar", command=sumar)
18 boton1.pack()
```

El comando puede ser una función o un método de una clase.



Podemos habilitar o deshabilitar un botón cambiando su atributo **state**, que puede estar en estado normal o deshabilitado.

Ejemplo: Habilitar y deshabilitar un botón.

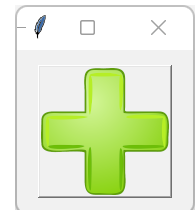
```
17 boton1['state'] = tk.NORMAL
18 boton1['state'] = tk.DISABLED
```

Utilizamos las constantes `tk.NORMAL` y `tk.DISABLED` para cambiar el atributo **state**.

Si queremos asociar una imagen al botón entonces deberemos crear un objeto *PhotoImage* igual que hicimos con las etiquetas.

Ejemplo: El botón muestra una imagen en su interior.

```
26 imagen = tk.PhotoImage(file="suma.png")
27 boton1 = ttk.Button(root, image=imagen, command=sumar)
28 boton1.pack()
```



Si queremos que se muestren tanto un texto como una imagen, deberemos configurar el atributo **compound** igual que anteriormente.

Ejemplo: El botón muestra una imagen y un texto en su interior.

```
33 imagen = tk.PhotoImage(file="suma.png")
34 boton1 = ttk.Button(root, text="suma", image=imagen, compound="right", command=sumar)
35 boton1.pack()
```



Puedes ver un [ejemplo completo](#) en el siguiente fichero: **6-Button.py**

6.5. Radiobutton

Los **Radiobutton** son *widgets* que suelen utilizarse en conjunto para mostrar varias opciones al usuario. El usuario podrá elegir una de las opciones y solo una. Es decir, solo podrá haber una opción seleccionada en cada momento.

Cada opción representa un objeto de la clase *Radiobutton* en el que indicamos el texto a mostrar por pantalla y el valor que representa (puede ser igual al texto o no).

La particularidad es que todos los objetos *Radiobutton* de un conjunto deben compartir la misma variable de control (objeto *StringVar*, *IntVar*, etc) para conocer el valor seleccionado por el usuario.

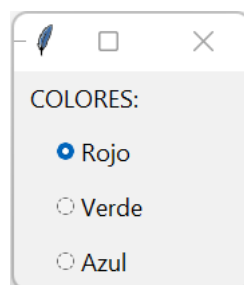
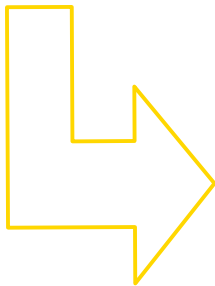
Ejemplo: Definimos un conjunto de *Radiobutton* para tres opciones de colores.

```
#Colores
colores = ttk.Label(root, text="COLORES:")
colores.pack(fill='x', padx=10, pady=5)

opcion = tk.IntVar()
ttk.Radiobutton(root, text="Rojo", value=1, variable=opcion).pack(fill='x', padx=30, pady=5)
ttk.Radiobutton(root, text="Verde", value=2, variable=opcion).pack(fill='x', padx=30, pady=5)
ttk.Radiobutton(root, text="Azul", value=3, variable=opcion).pack(fill='x', padx=30, pady=5)
```

Utilizamos la misma variable para todos los botones.

Más adelante veremos los parámetros de `pack()`.



También podemos asociarle una función o método a cada *Radiobutton*, igual como hacíamos con los botones sencillos.

Puedes ver un [ejemplo completo](#) en el siguiente fichero: **7-Radiobutton.py**

6.5.1. Uso del widget

En este apartado vamos a ver cómo recuperar el valor seleccionado por el usuario y, además, como crear un conjunto de objetos **RadioButton** de forma automática.

Primero, veremos una forma de generar los botones de forma automática basándonos en el uso de una lista.

Ejemplo:

```
22 opcion_selec = tk.IntVar()
23 opciones = [{"Rojo", 1}, {"Verde", 2}, {"Azul", 3}]
24 for opc in opciones:
25     r = ttk.Radiobutton(root, text=opc[0], value=opc[1], variable=opcion_selec,
26                         command=selecciona).pack(fill='x', padx=30, pady=5)
```

Como se puede, primero creamos una lista que contiene, por cada elemento, otra lista donde se indica el *texto* a mostrar en el **RadioButton** y su *valor* equivalente.

Fíjate que utilizamos el parámetro `command` de la clase **RadioButton** para que se ejecute una función llamada `selecciona()`, que se encargará de tratar el valor seleccionado. Para recuperar dicho valor deberemos utilizar el método `get()` de la variable de control asociada a los **RadioButton**, en este caso `opcion_selec`.

Ejemplo:

```
10 def selecciona():
11     selec_texto.set(opcion_selec.get())
```

`selec_texto` es una variable de control asociada a una etiqueta de la interfaz gráfica

Puedes ver un [ejemplo completo](#) en el siguiente fichero: **7-Radiobutton_uso.py**

6.6. Checkbutton

Este tipo de botones se utilizan para marcar o desmarcar alguna opción. A diferencia de los *Radiobutton*, éstos no se utilizan en grupo sino que son independientes entre sí.

Ejemplo: Usamos tres *Checkbox* para elegir si incluir o no ingredientes extra en una pizza.

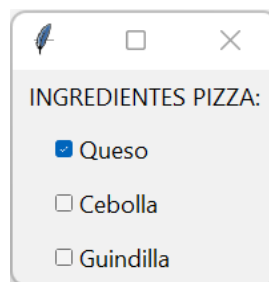
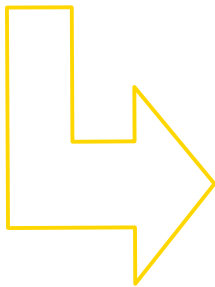
```
#Ingredientes pizza
ingredientes = ttk.Label(root, text="INGREDIENTES PIZZA:")
ingredientes.pack(fill='x', padx=10, pady=5)

queso = tk.BooleanVar()
check1 = ttk.Checkbutton(root, text="Queso", variable=queso, onvalue=True, offvalue=False)
check1.pack(fill='x', padx=30, pady=5)

cebolla = tk.BooleanVar()
check2 = ttk.Checkbutton(root, text="Cebolla", variable=cebolla, onvalue=True, offvalue=False)
check2.pack(fill='x', padx=30, pady=5)

guindilla = tk.BooleanVar()
check3 = ttk.Checkbutton(root, text="Guindilla", variable=guindilla, onvalue=True, offvalue=False)
check3.pack(fill='x', padx=30, pady=5)
```

La variable almacena el valor devuelto onvalue o offvalue, según si se ha marcado o no la opción.



También podemos asociarle una función o método a cada *Radiobutton*, igual como hacíamos con los otros botones.

Puedes ver un [ejemplo completo](#) en el siguiente fichero: **8-Checkbutton.py**

6.6.1. Uso del widget

De forma similar a como hemos utilizado los **RadioButton**, podemos gestionar los valores de los **CheckButton**.

Puedes ver un [ejemplo completo](#) en el siguiente fichero: **8-Checkbutton_uso.py**

6.7. Menú

Cuando implementamos una aplicación con muchas funcionalidades, suele ser habitual crear un menú principal donde agruparlas por categorías.

Los pasos a seguir son los siguientes:

1. Crear un *menú principal* o barra de menú. Suele ser habitual llamarlo *menubar*.

```
12 menubar = tk.Menu(root)
13 root.config(menu=menubar)
```

Indicamos el objeto *Menu* que actuará como menú principal de la aplicación.

2. Crear un *submenú* por cada opción del menú principal.

```
17 archivo_menu = tk.Menu(menubar, tearoff=False)
18 editar_menu = tk.Menu(menubar, tearoff=False)
19 ayuda_menu = tk.Menu(menubar, tearoff=False)
```

El parámetro `tearoff=False` deshabilita la posibilidad de poder desanclar el menú de la barra de menús.

3. En cada submenú, añadir las distintas *opciones* y asignarles la función o método que ejecutará.

```
22 archivo_menu.add_command(label="Nuevo")
23 archivo_menu.add_command(label="Abrir")
24 archivo_menu.add_command(label="Guardar")
25 archivo_menu.add_command(label="Cerrar")
26 archivo_menu.add_separator()
27 archivo_menu.add_command(label="Salir", command=root.destroy)
```

Para cada opción se indica el texto que mostrará (parámetro *label*) y, si procede, el comando que se ejecutará (parámetro *command*). El comando `root.destroy` es el preestablecido para cerrar la aplicación.

También podemos añadir líneas separadoras para ayudar a agrupar las opciones usando la función `add_separator`.

4. Asociar los submenús al menú principal.

```
40 menubar.add_cascade(label="Archivo", menu=archivo_menu)
41 menubar.add_cascade(label="Editar", menu=editar_menu)
42 menubar.add_cascade(label="Ayuda", menu=ayuda_menu)
```

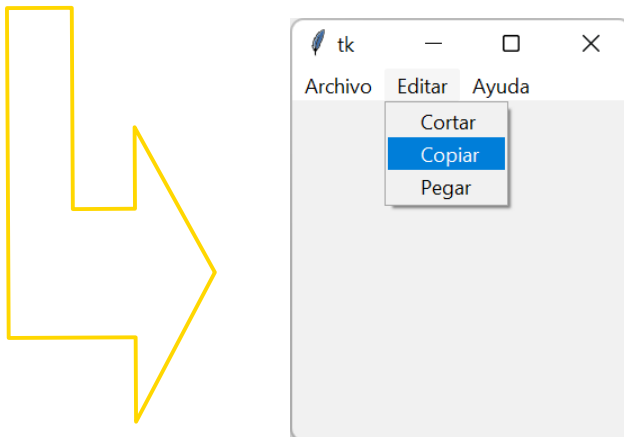
Indicamos el texto a mostrar en la opción del menú principal para cada submenú (parámetro *label*) y el menú que tendrá asociado (parámetro *menu*).

Veamos un ejemplo completo:

```

16 #Creamos los submenús -----
17 archivo_menu = tk.Menu(menu_bar, tearoff=False)
18 editar_menu = tk.Menu(menu_bar, tearoff=False)
19 ayuda_menu = tk.Menu(menu_bar, tearoff=False)
20
21 #Añadimos opciones al submenú "Archivo" -----
22 archivo_menu.add_command(label="Nuevo")
23 archivo_menu.add_command(label="Abrir")
24 archivo_menu.add_command(label="Guardar")
25 archivo_menu.add_command(label="Cerrar")
26 archivo_menu.add_separator()
27 archivo_menu.add_command(label="Salir", command=root.destroy)
28
29 #Añadimos opciones al submenú "Editar" -----
30 editar_menu.add_command(label="Cortar")
31 editar_menu.add_command(label="Copiar")
32 editar_menu.add_command(label="Pegar")
33
34 #Añadimos opciones al submenú "Ayuda" -----
35 ayuda_menu.add_command(label="Ayuda")
36 ayuda_menu.add_separator()
37 ayuda_menu.add_command(label="Acerca de...")
38
39 #Asignamos los submenús al menú principal -----
40 menu_bar.add_cascade(label="Archivo", menu=archivo_menu)
41 menu_bar.add_cascade(label="Editar", menu=editar_menu)
42 menu_bar.add_cascade(label="Ayuda", menu=ayuda_menu)

```



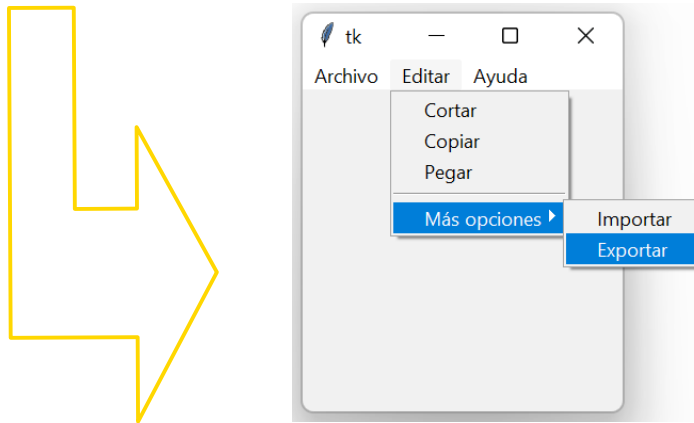
Por último, si queremos que una de las opciones de submenú despliegue otro submenú podemos hacerlo de la siguiente manera:

```

46 mas_menu = tk.Menu(editar_menu, tearoff=False)
47 mas_menu.add_command(label="Importar")
48 mas_menu.add_command(label="Exportar")
49 editar_menu.add_cascade(label="Más opciones", menu=mas_menu)

```

Asociamos el submenú al que queremos que sea su *menú padre* en lugar de al *menú principal*.



Puedes ver un [ejemplo completo](#) en el siguiente fichero: **9-Menu.py**

6.8. Ventana emergente

Cuando estamos ejecutando una aplicación, puede surgir la necesidad de mostrar información al usuario o solicitarle algún tipo de confirmación a través de una ventana emergente o diálogo. Esta funcionalidad podemos conseguirla utilizando funciones de distintos módulos de *Tkinter*. Aquí vamos a ver una muestra de algunos de ellos.

6.8.1. Módulo *messagebox*

Para incluir las funciones necesarias de este módulo debemos realizar una importación. Por ejemplo así:

```
1 try:
2     import Tkinter as tk
3     from Tkinter import ttk
4     import Tkinter.messagebox as tmb
5 except ImportError:
6     import tkinter as tk
7     from tkinter import ttk
8     import tkinter.messagebox as tmb
```

En la siguiente tabla se recogen algunos tipos de ventanas que encontramos en este módulo:

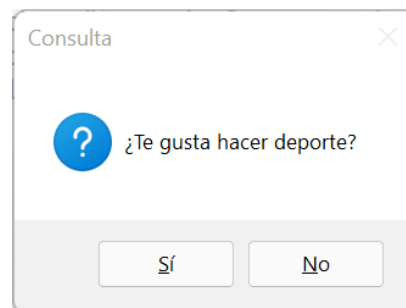
Función	Descripción
<code>showinfo()</code>	Esta función sirve para mostrar un texto informativo.

	<p>Sintaxis: <code>showinfo(título_ventana, mensaje)</code></p> <p>Ejemplo: <code>tmb.showinfo("Información", "Bienvenid@ al mundo de Python.")</code></p> 
<code>showwarning()</code>	<p>Esta función muestra un mensaje de aviso. La diferencia con el diálogo anterior es el tipo de icono que muestra.</p> <p>Sintaxis: <code>showwarning(título_ventana, mensaje)</code></p> <p>Ejemplo: <code>tmb.showwarning("Atención", "Puede haber hackers en esta habitación.")</code></p> 
<code>showerror()</code>	<p>Esta función muestra un mensaje de error. La diferencia con los diálogos anteriores es el tipo de icono que muestra.</p> <p>Sintaxis: <code>showerror(título_ventana, mensaje)</code></p> <p>Ejemplo: <code>tmb.showerror("Error", "El fichero no existe.")</code></p> 
<code>askquestion()</code>	<p>Esta función muestra una pregunta y dos botones: uno para responder "Sí" y otro para responder "No". Podemos recuperar el valor de la respuesta seleccionada por el usuario a través de una variable.</p> <p>Sintaxis: <code>askquestion(título_ventana, mensaje)</code></p>

Ejemplo:

```
respuesta = tmb.askquestion("Consulta",
    "¿Te gusta hacer deporte?")

if respuesta == "yes": #valor "no" para el otro botón
    pass #instrucciones para el "sí"
else:
    pass #instrucciones para el "no"
```



Para conocer más funciones de este módulo puedes visitar la siguiente página web:
<https://docs.python.org/es/3.10/library/tkinter.messagebox.html>

Puedes ver un ejemplo completo en el siguiente fichero: **10-messagebox.py**

6.8.2. Módulo `filedialog`

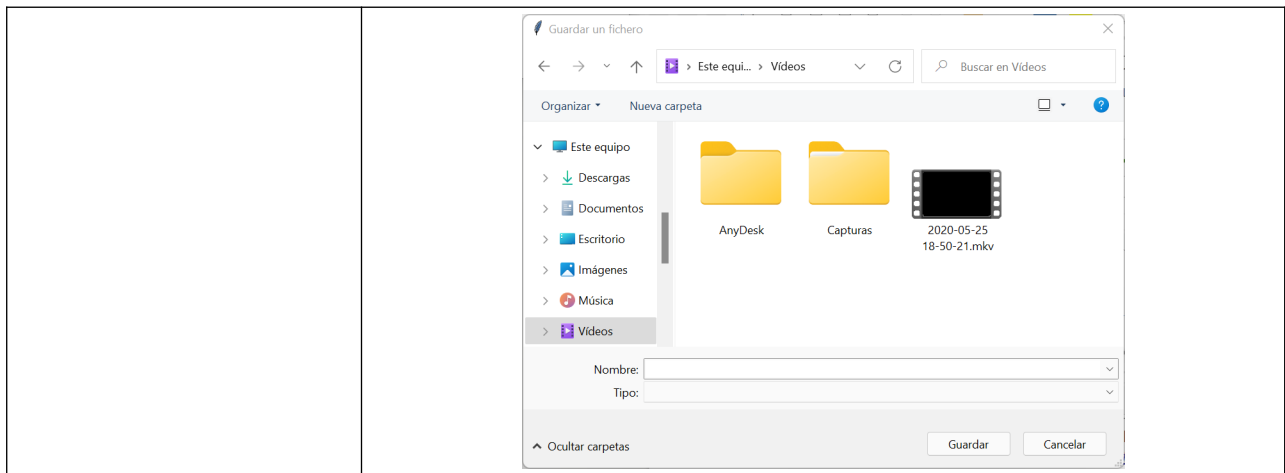
En este módulo encontramos funciones para trabajar con el sistema de archivos del sistema operativo. Por ejemplo: para abrir un fichero del disco duro o salvarlo.

Para incluir las funciones necesarias de este módulo debemos realizar esta importación:

```
1 try:
2     import Tkinter as tk
3     from Tkinter import ttk
4     import Tkinter.filedialog as tfd
5 except ImportError:
6     import tkinter as tk
7     from tkinter import ttk
8     import tkinter.filedialog as tfd
```

En la siguiente tabla se recogen algunos tipos de ventanas que encontramos en este módulo:

Función	Descripción
askopenfilename()	<p>Muestra una ventana que permite navegar por el sistema de archivos del disco duro para seleccionar un fichero.</p> <p>Sintaxis: askopenfilename(título_ventana, directorio_inicial, tipos_ficheros)</p> <p>Ejemplo:</p> <pre> dlg_selec = tfd.askopenfilename(title='Seleccionar un fichero', initialdir='/', filetypes= (('Fichero de texto', '*.txt'), ('Todos los ficheros', '*.*')) </pre> 
asksaveasfilename()	<p>Esta función muestra un mensaje de aviso. La diferencia con el diálogo anterior es el tipo de icono que muestra.</p> <p>Sintaxis: asksavefilename(título_ventana, tipo_fichero)</p> <p>Ejemplo:</p> <pre> dlg_save = tfd.asksaveasfilename(title="Guardar un fichero", defaultextension=".txt") </pre>



Para conocer más funciones de este módulo puedes visitar la siguiente página web: <https://docs.python.org/es/3.10/library/dialog.html?highlight=filedialog#tkinter.filedialog.Open>

Puedes ver un [ejemplo completo](#) en el siguiente fichero: **10-filedialog.py**

6.9. Contenedores

Los contenedores, marcos o *frames* (en inglés) son widgets con forma rectangular que permiten organizar o agrupar otros widgets en su interior. Estos contenedores pueden posicionarse directamente en la raíz o en otro contenedor.

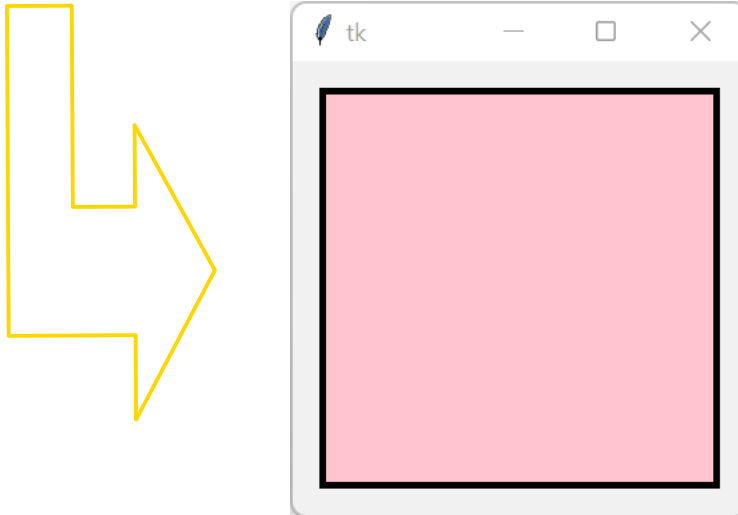
En este apartado hablaremos de dos tipos de contenedores pertenecientes a las clases **Frame** y **LabelFrame**.

6.9.1. Contenedor *Frame*

Para explicar el uso de esta clase nos basaremos en un ejemplo práctico.

Ejemplo: Creamos un contenedor que se ubica directamente en el *root*.

```
13 marco = tk.Frame(root, width='300', height='300')
14 marco.config(background='pink')
15 marco['borderwidth'] = 5 #Ancho del borde
16 marco['relief'] = 'solid' #Relieve, otros: flat, groove, raised, ridge, sunken.
17 marco.pack()
```



Al crear el marco con la clase **Frame** hemos indicado que se ubica directamente en la ventana principal y, además, su anchura y altura.

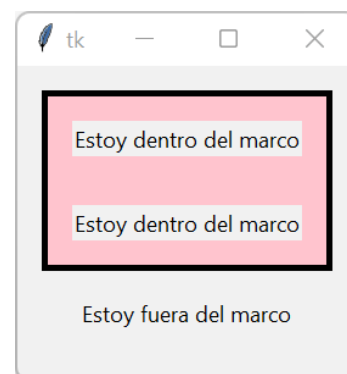
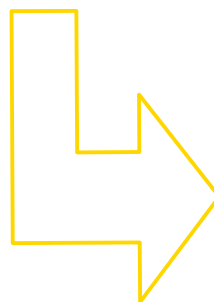
Después hemos utilizado la función **config()** para indicar el color de fondo (**background**) del marco y, por último, hemos utilizado otra nomenclatura (líneas 15 y 16) para indicar el ancho del borde (**borderwidth**) y su relieve (**relief**).

Una vez cread este marco, podríamos ubicar otros widgets dentro de él.

Ejemplo: Ubicamos varias etiquetas dentro del marco anterior y una fuera de él.

```
19 #Widgets dentro del marco
20 ttk.Label(marco, text="Estoy dentro del marco").pack(padx=20, pady=20)
21 ttk.Label(marco, text="Estoy dentro del marco").pack(padx=20, pady=20)
22
23 #Widgets fuera del marco
24 ttk.Label(root, text="Estoy fuera del marco").pack(padx=20, pady=20)
```

En el primer parámetro decidimos si mostramos el widget directamente en la raíz en otro contenedor



Puedes ver un [ejemplo completo](#) en el siguiente fichero: **11-Frame.py**

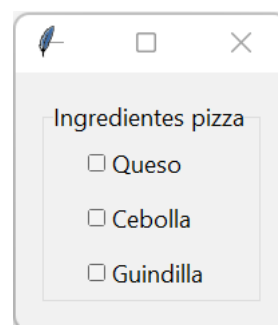
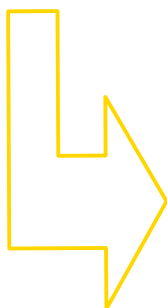
6.9.2. Contenedor **LabelFrame**

Los objetos de la clase **LabelFrame** suelen utilizarse para agrupar otros widgets, como los **RadioButton** o **CheckButton** y muestran un texto en su parte superior.

Ejemplo: Creamos un **LabelFrame** que contiene tres botones de tipo **Checkbutton** para seleccionar los ingredientes extra de una pizza.

```
12 #Ingredientes pizza
13 lf = ttk.LabelFrame(root, text='Ingredientes pizza')
14 lf.pack()
15
16 queso = tk.BooleanVar()
17 check1 = ttk.Checkbutton(lf, text="Queso", variable=queso, onvalue=True, offvalue=False)
18 check1.pack(fill='x', padx=30, pady=5)
19
20 cebolla = tk.BooleanVar()
21 check2 = ttk.Checkbutton(lf, text="Cebolla", variable=cebolla, onvalue=True, offvalue=False)
22 check2.pack(fill='x', padx=30, pady=5)
23
24 guindilla = tk.BooleanVar()
25 check3 = ttk.Checkbutton(lf, text="Guindilla", variable=guindilla, onvalue=True, offvalue=False)
26 check3.pack(fill='x', padx=30, pady=5)
```

A cada **Checkbutton** le indicamos que su contenedor será el **LabelFrame** en lugar de la ventana principal.



Puedes ver un [ejemplo completo](#) en el siguiente fichero: **12-LabelFrame.py**

7. Gestores de geometría

Los gestores de geometría nos permiten establecer la posición relativa de los *widgets* dentro de los marcos o la ventana principal. Existen tres tipos:

- El gestor **pack()** que se encarga de distribuir los *widgets* de un contenedor de forma automática según los parámetros de configuración.
- El gestor **grid()** que se basa en el uso de una rejilla con filas y columnas para que el/la programador/a pueda decidir dónde posicionar los *widgets*.
- El gestor **place()** que permite configurar las coordenadas (x, y) de cada *widget* para establecer su posición dentro del contenedor.

En este documento trataremos los parámetros más importantes del gestor **pack()** y dejaremos para vuestra investigación personal los otros dos. Os recomiendo que visitéis estas páginas:

- Gestor **grid()**: <https://www.pythontutorial.net/tkinter/tkinter-grid/>
- Gestor **place()**: <https://www.pythontutorial.net/tkinter/tkinter-place/>

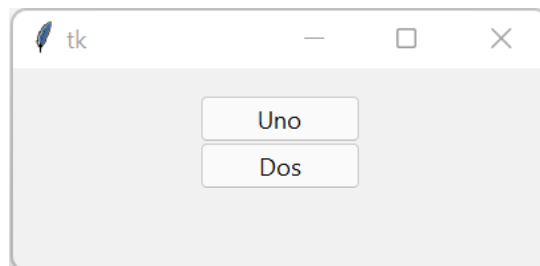
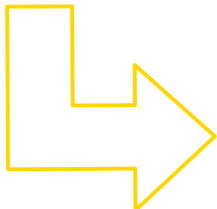
7.1. Gestor pack()

El gestor **pack()** admite múltiples configuraciones. Aquí presentaremos las más habituales.

Para ello, partiremos de un ejemplo sencillo al que iremos configurando determinados parámetros para ver su efecto.

Ejemplo: Configuración por defecto.

```
12 # Configuración de la raíz
13 root = tk.Tk()
14 root.config(bd=20)
15 root.geometry('400x150+500+300')
16
17 #Configuración de los botones básicos -----
18 boton1 = ttk.Button(root, text="Uno", command=sumar)
19 boton1.pack()
20
21 boton2 = ttk.Button(root, text="Dos", command=sumar)
```



Como se puede ver, la función **pack()** sin parámetros ubica los *widgets* centrados y apilados uno encima de otro.

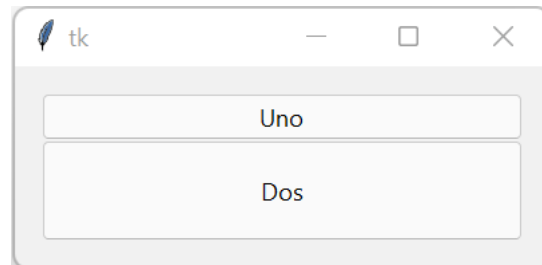
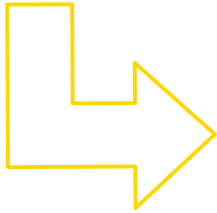
7.1.1. Parámetros *fill* y *expand*

El parámetro **fill** sirve para indicar si queremos que el *widget* rellene todo el espacio disponible. Podemos seleccionar si queremos que se expanda a lo largo del eje de coordenadas **X**, a lo largo del eje de coordenadas **Y** o a lo largo de los dos ejes (**both**).

Si queremos que el *widget* pueda crecer verticalmente debemos utilizar el parámetro **fill** junto con el parámetro **expand**.

Ejemplo: El botón superior se expande a lo ancho y el inferior a lo ancho y alto.

```
25 boton1.pack(fill='x')  
26 boton2.pack(fill='both', expand=True)
```

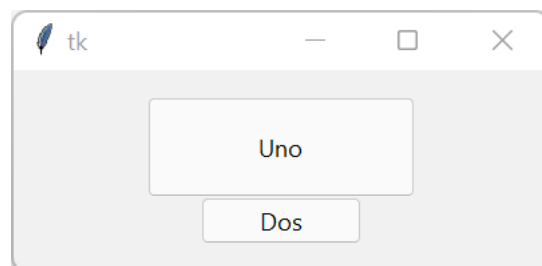
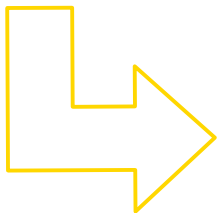


7.1.2. Parámetros *ipadx* e *ipady*

Los parámetros **ipadx** e **ipady** sirven para indicar el relleno que habrá entre el contenido del *widget* y su borde externo. Podemos indicar el relleno en el eje de coordenadas **X** (*ipadx*) y en el eje de coordenadas **Y** (*ipady*).

Ejemplo: Utilizamos los dos rellenos para el primer botón.

```
29 boton1.pack(ipadx=40, ipady=20)
```

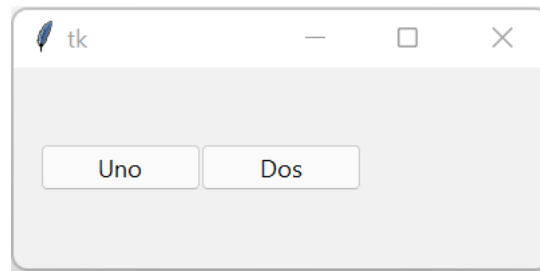
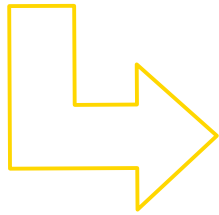


7.1.3. Parámetro *side*

Por último, probaremos el parámetro **side** que nos permite indicar qué lado del *widget* alinearemos con su contenedor. Los posibles valores son los siguientes: **left**, **right**, **top**, **bottom**. Tal y como hemos podido comprobar, el valor por defecto es **top**.

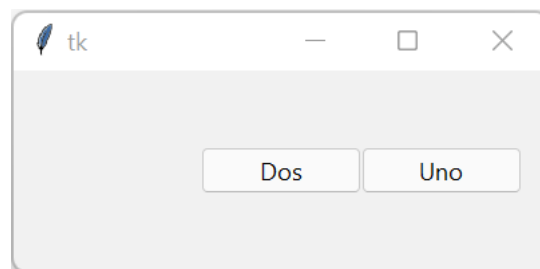
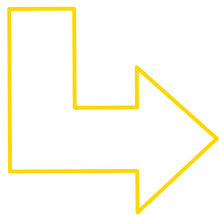
Ejemplo: Alineamos los dos botones a su izquierda.

```
30 boton1.pack(side='left')  
31 boton2.pack(side='left')
```

Ejemplo: Alineamos los dos botones a la derecha .

```
36 boton1.pack(side='right')  
37 boton2.pack(side='right')
```



Puedes ver un [ejemplo completo](#) en el siguiente fichero: **13-gestor_pack.py**

8. Utilizando una clase específica

Aunque la forma de trabajar que hemos visto hasta ahora es correcta, podríamos dar un paso más y crear una clase **Aplicacion** (o con otro nombre) que heredase de la clase **Frame** del módulo **tkinter** y dispusiese de todos los métodos necesarios para trabajar con la aplicación gráfica:

La estructura podría ser algo así:

```
try:  
    import Tkinter as tk  
    from Tkinter import ttk  
except ImportError:  
    import tkinter as tk  
    from tkinter import ttk  
  
#Definición de la clase -----  
class Aplicacion(tk.Frame):
```

```
def __init__(self, master):
    super().__init__(master) #Llamamos al constructor de Frame

    #Inicialización del objeto Aplicacion
    self.master = master

    #Disposición del objeto Aplicacion
    self.pack()

    #Llamamos al método que crea los elementos
    self.crear_widgets()

def crear_widgets(self):
    #Creación de los elementos

#Programa principal -----
root = tk.Tk() #Creamos la raíz
app = Aplicacion(root) #Creamos el objeto Aplicacion
app.mainloop() #Lanzamos a ejecución el objeto Aplicacion
```

En el fichero **14-Ejemplo completo.zip** puedes encontrar un *ejemplo completo* en dos versiones: una sin utilizar la clase *Aplicacion* (**14-Ejemplo_completo_sin_clase.py**) y otra utilizándola (**14-Ejemplo_completo_con_clase.py**).

9. Fuentes de información

- tkinter — Interface de Python para Tcl/Tk: <https://docs.python.org/es/3/library/tkinter.html#module-tkinter>
- Tutorial Tkinter: <https://www.pythontutorial.net/tkinter/>
- Interfaces gráficas con Tkinter de Hektor Profe: <https://docs.hektorprofe.net/python/interfaces-graficas-con-tkinter/>
- Tk Docs: <http://tkdocs.com/tutorial/concepts.html>

10. Créditos de imágenes

	Image by Press Love you from Pixabay
	Image by Sara Torda from Pixabay
	Image by Clker-Free-Vector-Images from Pixabay