Tema 8

Crear una GUI con Tkinter

Manual del Estudiante

Maria del Carmen Sierra Fernández

Contenido

Nivel II: Representación, proceso y visualización de Datos

Crear una Interfaz Gráfica de Usuario (GUI)

La interfaz gráfica de usuario (GUI) es una forma de interfaz de usuario que permite a los usuarios interactuar con los ordenadores a través de objetos visuales utilizando elementos como iconos, menús, ventanas, etc.

Tkinter es un módulo de interfaz gráfica de usuario (GUI) para Python, que nos permite crear aplicaciones GUI de escritorio. Podemos crear ventanas, botones, mostrar texto e imágenes, entre otras.

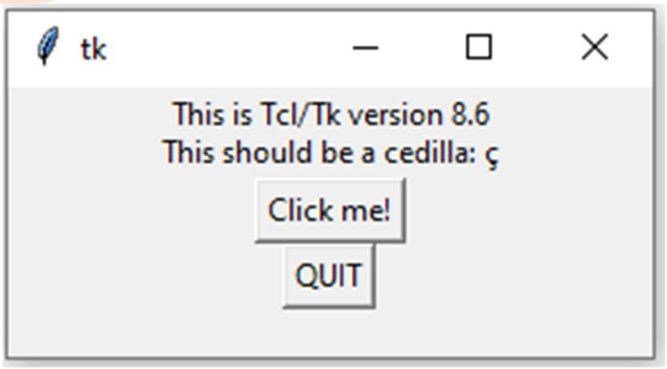
El módulo Tkinter es una interfaz estándar de Python para el kit de herramientas Tk GUI. Como Tk y Tkinter están disponibles en la mayoría de las plataformas Unix, así como en el sistema Windows y Mac OS X, el desarrollo de aplicaciones GUI con Tkinter es rápido y fácil.

Otras librería disponibles para crear aplicaciones GUI son:

- Kivy
- Qt
- wxPython

Para comprobar la librería Tkinter, desde la consola de Python ejecutamos:

```
>> import tkinter
>> tkinter._test()
```



`tkinter._test()` nos muestra una ventana con la versión de Tk y dos botones. Seleccione el botón `QUIT` para salir.

Importar las librerías Tkinter

El módulo de tkinter se importa al igual que cualquier otro módulo:

```
# Importar módulo tkinter
import tkinter
```

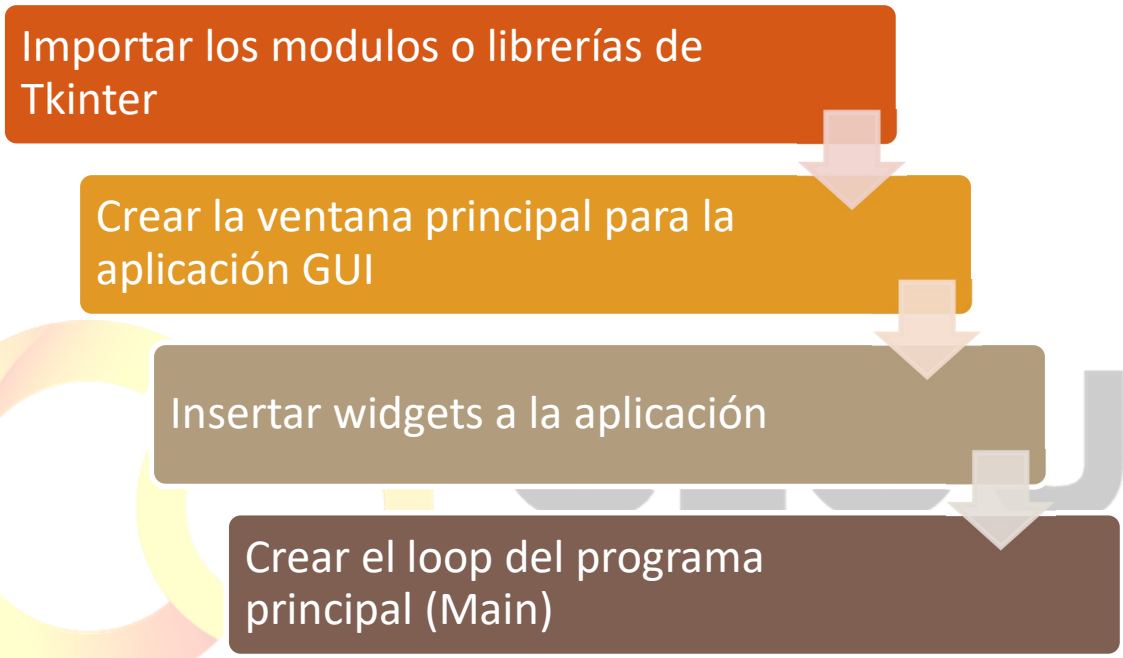
si necesitamos importar todos los módulos:

```
# Importar los  módulos tkinter
from tkinter import *
from tkinter.ttk import *
```

El objetivo de `tkinter.ttk` es separar, el código que implementa el comportamiento de un widget del código que implementa su apariencia.

Estructura de una aplicación Tkinter

La estructura de un programa o aplicación Tkinter es la siguiente:



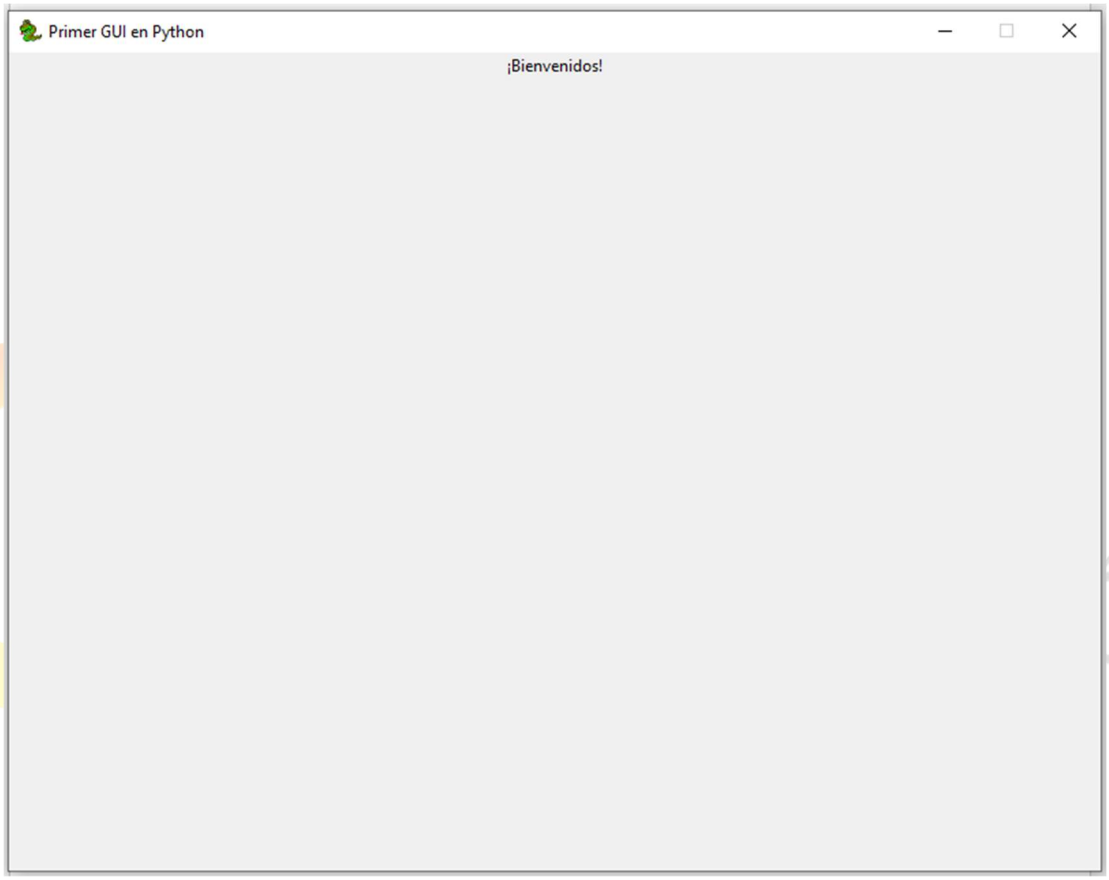
¿Qué son los widgets?

Los widgets en Tkinter son los elementos de la aplicación GUI que proporcionan varios controles (como etiquetas, botones, cuadros combinados, casillas de verificación, barras de menú, botones de radio y muchos más) para que los usuarios interactúen con la aplicación.

Ejemplo (EjemploClase_1_CrearVentana.py)

```
# Importar los módulos o  librerías Tkinter
from tkinter import *
from tkinter.ttk import *

# Crear la ventana principal para la aplicación GUI
root = Tk()
root.geometry(newGeometry='800x600')
root.resizable(width=False, height=False)

# Insertar widgets a la aplicación
```

```
# Titulo de la ventana
root.title("Primer GUI en Python")
root.iconbitmap('python.ico')


# Etiqueta
label = Label(root, text="¡Bienvenidos!").pack()

# Crear el loop del programa principal (Main)
root.mainloop()
```



Para ejecutar nuestro programa desde la terminal, nos ubicamos en el directorio donde se encuentra nuestro programa `EjemploClase_1_CrearVentana.py` y ejecutamos:

```
> python EjemploClase_1_CrearVentana.py
```

Widgets

Como hemos comentamos un Widget, es un elemento de la interfaz gráfica de usuario (GUI) que muestra información o proporciona una forma para que el usuario interactúe con el sistema operativo. En otras palabras, es todo lo que vemos en la ventana.

En Tkinter, los widgets o controles son objetos de Python; instancias de clases que representan botones, marcos, etc. Al crear un widget, debemos pasar su padre (parent) como parámetro a la función de creación de widgets. Una excepción es la ventana principal (root), que contendrá todo los objetos y no tiene un padre.

Widgets básicos de Tkinter

Widgets	Descripción de los widgets
Label	Se utiliza para mostrar texto o imagen en la pantalla
Button	Se utiliza para agregar botones a su aplicación
Canvas	Se utiliza para dibujar imágenes y otros diseños como textos, gráficos, etc.
ComboBox	Contiene una flecha hacia abajo para seleccionar de la lista de opciones disponibles
Checkbutton	Muestra una cantidad de opciones para el usuario como botones de alternancia desde los cuales el usuario puede seleccionar cualquier cantidad de opciones.
Radiobutton	Se utiliza para implementar una selección de muchos, ya que permite seleccionar solo una opción.
Entry	Se utiliza para ingresar una entrada de texto de una sola línea del usuario
Frame	Se utiliza como contenedor para contener y organizar los widgets.
Message	Funciona igual que el de la etiqueta y se refiere a texto de varias líneas y no editable.
Scale	Se utiliza para proporcionar un control deslizante gráfico que permite seleccionar cualquier valor de esa escala
Scrollbar	Se utiliza para desplazarse hacia abajo por los contenidos. Proporciona un controlador deslizante.
SpinBox	Permite al usuario seleccionar de un conjunto dado de valores
Text	Permite al usuario editar texto de varias líneas y formatear la forma en que debe mostrarse
Menu	Se utiliza para crear todo tipo de menú utilizado por una aplicación

Administrador de geometría

Crear un widget no significa que aparecerá en la pantalla. Para mostrarlo, debemos llamar a un método especial: `grid`, `pack` o `place`.

Método	Descripción del método
pack()	El administrador de geometría pack empaqueta widgets en filas o columnas.
grid()	El administrador de geometría de cuadrícula coloca los widgets en una tabla bidimensional (filas y columnas), donde cada "celda" de la tabla resultante puede contener un widget.
place()	El administrador de geometría place permite establecer la posición y el tamaño de una ventana.

`pack()` es recomendable para diseños simples, porque no se adapta tan bien a diseños más complejos sin una cantidad excesiva de Frame, en estos casos la recomendación es utilizar `grid()`, que nos permite diseñar widgets en una cuadrícula bidimensional, como un documento de hoja de cálculo o una tabla HTML.

Por defecto, una llamada a `grid()` colocará el widget en la primera columna (columna 0) de la siguiente fila vacía. Por lo tanto, si tuviéramos que llamar simplemente a `grid()` en el siguiente widget, sería directamente debajo de la primera.

```
título.grid()
```

Sin embargo, también podemos ser explícitos indicando los argumentos o parámetros de fila y columna, así:

```
etiqueta.grid(row=1, column=0)
```

Las filas y columnas cuentan desde la esquina superior izquierda del widget, comenzando con 0. Por lo tanto, fila=1, columna=0 coloca el widget en la segunda fila de la primera columna. Si queremos una columna adicional, todo lo que tenemos que hacer es colocar un widget en ella, así:

```
nombre.grid(fila=1, columna=1)
```

La cuadrícula se expande automáticamente cada vez que agregamos un widget a una nueva fila o columna.

Si un widget es más grande que el ancho actual de la columna o el alto de la fila, todas las celdas de esa columna o fila se expanden para acomodarla. Podemos hacer que un widget abarque varias columnas o varias filas utilizando los parámetros `columnpan` y `rowspan`. Por ejemplo, si necesitamos que el titulo abarque toda la línea:

```
título.grid(columnspan=2)
```

A medida que se expanden las columnas y las filas, los widgets no se expanden con ellas de forma predeterminada. Si queremos que se expandan, necesitamos usar el argumento `sticky`, como este:

```
eater_inp.grid(row=2, columnspan=2, sticky='we')
```

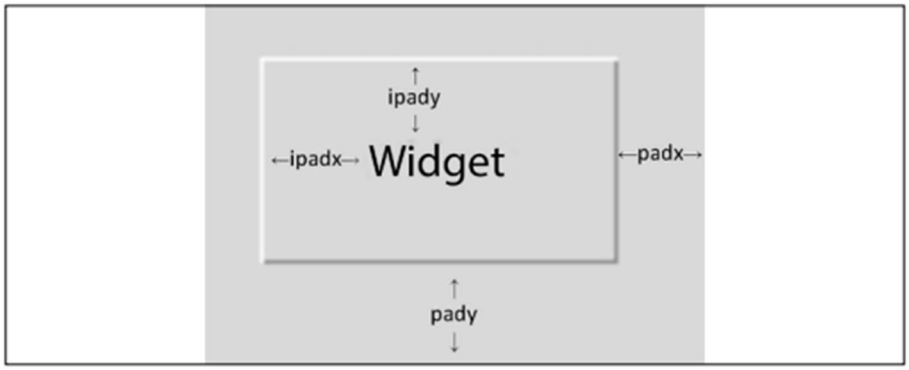
`sticky` le dice a Tkinter que pegue los lados del widget a los lados de la celda que lo contiene para que el widget se estire a medida que se expande la celda. Sticky toma direcciones cardinales: n, s, e y w. En este caso hemos especificado Oeste y Este, lo que hará que el widget se estire horizontalmente si ella columna se expande aún más.

Como alternativa a las cadenas, también podemos usar las constantes de Tkinter como argumentos para sticky:

El método `grid()` nos permite agregar relleno a nuestros widgets también, así:

```
etiqueta.grid(row=4, columnpan=2, sticky='we', pady=10)
etiqueta2.grid(row=5, columnpan=2, sticky='we', padx=25)
```

padx y pady indican relleno externo, es decir, expandirán la celda contenedora, pero no el widget. ipadx e ipady, por otro lado, indican relleno interno. Especificar estos argumentos expandirá el propio widget (y, en consecuencia, la celda contenedora)



Tkinter no nos permite mezclar administradores de geometría en el mismo widget principal; una vez que hemos llamado a grid() en cualquier widget secundario, una llamada al método pack() o place() en un widget hermano generará un error y viceversa.

Sin embargo, podemos usar un administrador de geometría diferente en los hijos del widget hermano. Por ejemplo, podemos usar `pack()` para colocar los widgets secundarios en un widgets frame.

```
# Crear un frame en la ventana principal
frame = Frame(root)
frame.pack()

# Agregar botones en el frame de root con pack
button = Button(frame, text='Mi botón', bd = '2')
button.pack()
```

Ejemplo (EjemploClase_2_geometry.py):

```
from tkinter import *

# Crear la ventana principal
root = Tk()
root.geometry(newGeometry='800x600')
root.resizable(width=False, height=False)
root.title("Crear un Frame y un Botón")

# Agregar botón con pack
button = Button(root, text='Mi botón', bd = '2')
button.pack()

# Agregar etiqueta con place
etiqueta = Label(root, text="Etiqueta con Place")
etiqueta.place(x=40,y=60)

# Agregar controles con grid
button.grid(row=0,column=0)
etiqueta.grid(row=1,column=1)

# Tkinter event loop
root.mainloop()
```

Widget Label

`Label` es un widget que muestra texto o imágenes, que los usuarios sólo visualizarán (no pueden interactuar con ellas). Se utilizan para identificar controles u otras partes de la interfaz de usuario. El texto de una etiqueta (Label) podemos cambiarlo en cualquier momento desde la aplicación.

Sintaxis:

```
w = Label(parent, option, … )
```

donde `parent` es la ventana principal y `option` son pares clave-valor separados por coma. Algunas opciones disponibles son:

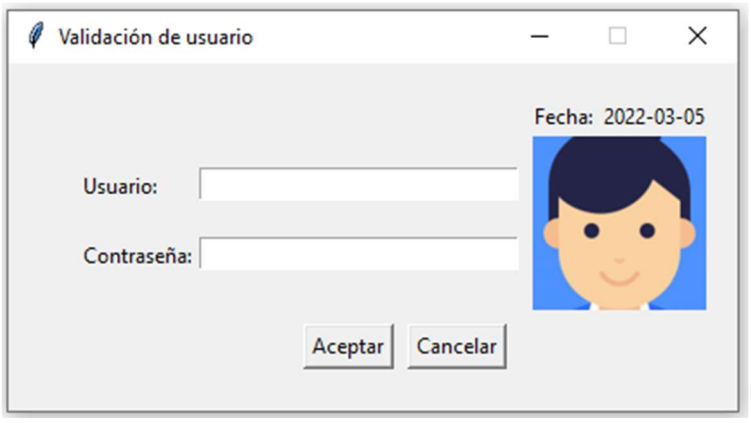
- **text:** Se utilizar para indicar el texto a mostrar
- **anchor:** esta opción se utiliza para controlar la posición del texto. El valor predeterminado es ancla=CENTER.
- **bg:** Esta opción se utiliza para establecer el color de fondo de la etiqueta.
- **height:** Esta opción se utiliza para establecer la dimensión vertical del nuevo marco.
- **width:** ancho de la etiqueta en caracteres. Si esta opción no está configurada, la etiqueta se ajustará al tamaño de su contenido.

- **bd:** esta opción se utiliza para establecer el tamaño del borde. El valor bd predeterminado se establece en 2 píxeles.
- **font:** si estamos mostrando texto en la etiqueta, esta opción nos permite definir el tipo de fuente a utilizar.
- **cursor:** se utiliza para especificar qué cursor mostrar cuando se mueve el mouse sobre la etiqueta. El valor predeterminado es utilizar el cursor estándar.
- **textvariable:** como su nombre lo indica, está asociado con una variable Tkinter (generalmente un StringVar) con la etiqueta. Si se cambia la variable, se actualiza el texto de la etiqueta.
- **bitmap:** se utiliza para establecer el mapa de bits en el objeto gráfico especificado para que la etiqueta pueda representar los gráficos en lugar de texto.
- **image:** esta opción se utiliza para mostrar una imagen estática en el widget de etiqueta.
- **padx:** esta opción se usa para agregar espacios adicionales entre la izquierda y la derecha del texto dentro de la etiqueta. El valor predeterminado para esta opción es 1.
- **pady:** esta opción se usa para agregar espacios adicionales entre la parte superior e inferior del texto dentro de la etiqueta. El valor predeterminado para esta opción es 1.
- **justify:** esta opción se utiliza para definir cómo alinear varias líneas de texto. Utilice LEFT, RIGHT o CENTER. El valor predeterminado CENTER.
- **relief:** esta opción se utiliza para especificar la apariencia de un borde decorativo alrededor de la etiqueta. El valor predeterminado para esta opción es FLAT.
- **underline:** mostrar la etiqueta con el efecto de subrayado
- **wraplength:** en lugar de tener una sola línea como texto de la etiqueta, se puede dividir en el número de líneas donde cada línea tiene el número de caracteres especificado para esta opción.

Ejemplo (EjemploClase_3_Labels.py):

```
from datetime import date
from tkinter import *
from PIL import Image, ImageTk

# Crear la ventana principal para la aplicación GUI
root = Tk()
root.geometry(newGeometry='420x200')
root.resizable(width=False, height=False)


# Insertar widgets a la aplicación
# Titulo de la ventana
root.title("Validación de usuario")

# Etiquetas de controles Usuario y Contraseña
```

```
l_usuario = Label(root, text="Usuario: ").place(x=40,y=60)  # ,
bg="blue"
l_contrasena = Label(root, text="Contraseña: ").place(x=40,
y=100) #,relief="solid", font="Times 8 bold"

# Etiquetas generadas en tiempo de ejecución (Variables)
fecha = StringVar()
Label(root, text="Fecha: ").place(x=300,y=20)
Label(root,  textvariable=fecha).place(x=340,y=20)
fecha.set(date.today()) # Asignamos el texto en tiempo de
ejecución

# Etiquetas con imágenes
imagen = Image.open("perfil.png")  # librería PIL
imagen = imagen.resize((100, 100))
imagen = ImageTk.PhotoImage(imagen)
Label(root, image=imagen).place(x=300, y=40)  #

# Botones Aceptar y Cancelar
boton_aceptar = Button(root, text="Aceptar").place(x=170,
y=150)
boton_cancelar = Button(root, text="Cancelar").place(x=230,
y=150)

# Entrada de Datos
e_usuario = Entry(root, width=30).place(x=110,  y=60)
e_contrasena = Entry(root, width=30).place(x=110, y=100)

root.mainloop()
```



Widget Entry

El widget `Entry()` se utiliza para ingresar o mostrar una sola línea de texto.

Sintaxis:

```
entrada = tk.Entry(parent, option, …)
```

donde `parent` es la ventana principal en el que se mostrará el widget y `option` las diversas opciones proporcionadas por el widget:

- **bg:** El color de fondo que se muestra detrás de la etiqueta y el indicador.
- **bd:** El tamaño del borde alrededor del indicador. El valor predeterminado es 2 píxeles.
- **font:** La fuente utilizada para el texto.
- **fg:** El color utilizado para representar el texto.
- **justify:** si el texto contiene varias líneas, esta opción controla cómo se justifica el texto: `LEFT, RIGHT o CENTER.`
- **relief:** Con el valor por defecto, `relief=FLAT`. Podemos configurar esta opción para cualquiera de los otros estilos como: `SUNKEN, RIGID, RAISED, GROOV`
- **show:** Normalmente, los caracteres que escribe el usuario aparecen en la entrada. Para hacer una .contraseña. entrada que repite cada carácter como un asterisco, configure `show="*"`.
- **textvariable:** para poder recuperar el texto actual del widget de entrada, debemos establecer esta opción en una instancia de la clase `StringVar`.

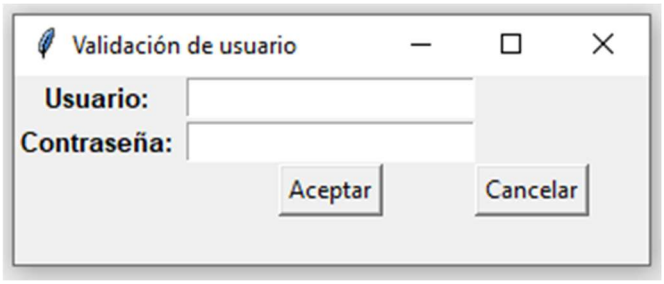
Métodos: los diversos métodos proporcionados por el widget de entrada son:

- **get():** Devuelve el texto actual de la entrada como una cadena.
- **delete():** Elimina caracteres del widget
- **insert(index, 'name'):** inserta la cadena 'name' antes del carácter en el índice dado.

Ejemplo (EjemploClase_4_Entry.py):

```
from tkinter import *

# Función para obtener los datos
def obtener_datos():
    nombre = usuario.get()
    passw = password.get()

    print("El nombre es : " + nombre)
    print("La contraseña es : " + passw)

    usuario.set("")
    password.set("")

# Crear la ventana principal para la aplicación GUI
root = Tk()

# Insertar widgets a la aplicación
# Titulo de la ventana
root.title("Validación de usuario")
```

```
# Declarar variable para almacenar el usuario y la contraseña
usuario = StringVar()
password = StringVar()

# Etiquetas de controles Usuario y Contraseña
l_usuario = Label(root, text="Usuario: ", font=('calibre',10,
'bold'))
l_password = Label(root, text="Contraseña: ",
font=('calibre',10, 'bold'))

# Entrada de Datos
e_usuario = Entry(root, textvariable = usuario,
font=('calibre',10, 'bold'))
e_password = Entry(root, textvariable = password, show="*",
font=('calibre',10, 'bold'))

# Botones Aceptar y Cancelar
boton_aceptar = Button(root, text="Aceptar", command =
obtener_datos)
boton_cancelar = Button(root, text="Cancelar",
command=root.quit)

# Ordenar los controles en un grid
l_usuario.grid(row=0,column=0)
e_usuario.grid(row=0,column=1)
l_password.grid(row=1,column=0)
e_password.grid(row=1,column=1)
boton_aceptar.grid(row=2,column=1)
boton_cancelar.grid(row=2,column=2)

root.mainloop()
```



Widget Button

Un botón, a diferencia de un marco o una etiqueta, nos permiten interactuar. Los usuarios presionan un botón para realizar una acción. Al igual que las etiquetas, pueden mostrar texto o imágenes y acepta opciones adicionales para cambiar su conducta.

Los botones se crean utilizando la clase `ttk.Button`:

```
btn = ttk.Button(parent, option = value, ...)
```

donde `parent` es la ventana principal en el que se mostrará el widget y `option` las diversas opciones proporcionadas por el widget `button`.

Opciones:

La opción `style` se utiliza para alterar su apariencia y comportamiento.

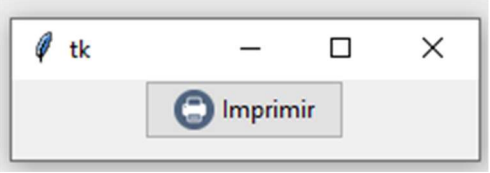
Ejemplo (EjemploClase_5_Stylebutton.py)

```
# Crear un objeto Style
style = Style()

# Definir un estilo (TButton es utilizado por ttk.Button)
style.configure('W.TButton', font=('calibri', 10, 'bold'))

# Aplicar el estilo al botón
boton = Button(root, text='Salir', style='W.TButton',
command=root.destroy)
boton.grid(row=0, column=3, padx=100)
```

Los botones tienen al igual que las etiquetas, opciones de configuración `text`, `textvariable` (no muy utilizada), `image`, y `compound`. Estos controlan si el botón muestra un texto y/o una imagen.

Ejemplo (EjemploClase_6_Imagebutton.py)

```
from tkinter import *
from tkinter.ttk import *
from PIL import Image, ImageTk

root = Tk()

# Ajustar el tamaño de la imagen y Crear un objeto Photoimagen
imagen = Image.open("printer.png")
imagen = imagen.resize((20, 20))
imagen = ImageTk.PhotoImage(imagen)

# Mostrar la imagen en el botón
Button(root, text='Imprimir', image=imagen,
compound=LEFT).pack(side=TOP)

mainloop()
```

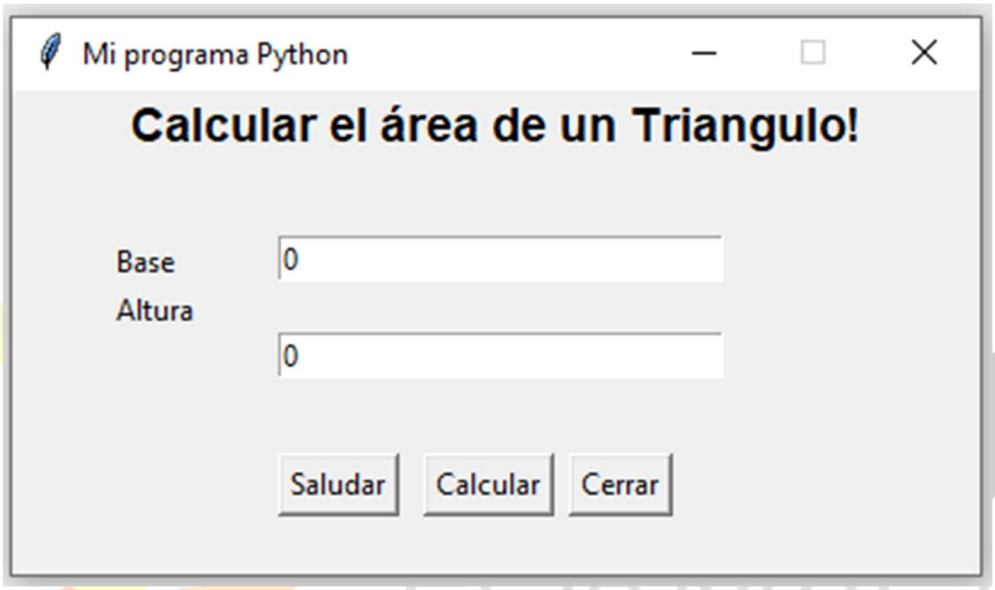


La opción `command`, especifica la función que se llamará cuando se presione el botón, es decir, conecta la acción del botón y la aplicación. Cuando un usuario presiona el botón, el script proporcionado por la opción es evaluado por el intérprete.

EjemploClase_7_CalcularArea.py

```
from tkinter import *

def limpiar_label():
    label = Label(root, text="                                        ")
    label.place(x=50, y=30)

def saludar():
    limpiar_label()
    label = Label(root, text="Bienvenido")
    label.place(x=50, y=30)

def calcular():
    limpiar_label()
    a = base.get() * altura.get()
    label = Label(root, text=f"El área del triángulo es: {a} ")
    label.place(x=50, y=30)

if __name__ == '__main__':
    # Ventana principal
    root = Tk()
    root.geometry('400x200')
    root.resizable(False, False)
    root.title("Mi programa Python")

    # Variables
    base = IntVar(value=0)
    altura = IntVar(value=0)

    # Etiquetas y entradas
    etiqueta = Label(root, text="Calcular el área de un Triángulo!",
                           font=('calibre', 14, 'bold')).pack()

    etiquetabase = Label(root, text="Base").place(x=40, y=60)
    entrada_base = Entry(root, textvariable=base,
                                width=30).place(x=110, y=60)

    etiquetaaltura = Label(root, text="Altura").place(x=40, y=80)
    entrada_altura = Entry(root, textvariable=altura,
                                     width=30).place(x=110, y=100)

    # Botones
    botonSaludo = Button(root, text="Saludar")
    botonSaludo.place(x=120, y=150)
```

```
botonCalcular = Button(root, text="Calcular")
botonCalcular.place(x=170, y=150)

botonCerrar = Button(root, text="Cerrar", command=root.quit)
botonCerrar.place(x=230, y=150)

botonCalcular.configure(command=calcular)
botonSaludo.configure(command=saludar)

root.mainloop()
```



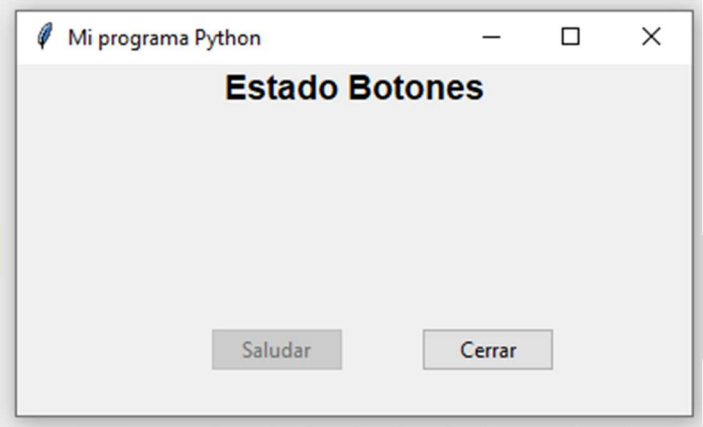
La opción `default`, si se especifica como `active`, esto le dice a Tk que el botón es el botón predeterminado en la interfaz de usuario; de lo contrario es `normal`. Los botones predeterminados se invocan si los usuarios pulsen la tecla ENTER. Algunas plataformas y estilos dibujarán este botón predeterminado con un borde o resaltado diferente.

Estado del botón

Los botones y muchos otros widgets comienzan en un estado normal. Un botón responderá a los movimientos del mouse, se puede presionar e invocará a una función. Los botones también se pueden poner en un estado deshabilitado, donde el botón está atenuado, no responde a los movimientos del mouse, y no se puede presionar. Esto nos permite deshabilitar los botones cuando su comando no es aplicable en un momento dado.

Todos los widgets temáticos mantienen un estado interno, representado como una serie de banderas binarias. Cada indicador puede activarse o desactivarse. Podemos configurar o borrar estos indicadores y comprobar la configuración actual mediante los métodos `state` e `instate`. Los botones hacen uso de `disable` para controlar si los usuarios pueden o no presionar el botón. Por ejemplo:

```
# Habilitar y/o deshabilitar un botón
botonSaludo.state(['disabled']) # Deshabilita el botón
botonSaludo.state(['!disabled']) # Habilita el botón

# Conocer el estado actual del botón

# Retorna True si está deshabilitado
botonSaludo.instate(['disabled'])

# Retorna False si está deshabilitado
botonSaludo.instate(['!disabled'])

# Ejecutar el comando si no está deshabilitado
botonSaludo.instate(['!disabled'], cmd)
```



Widget Text

El widget de `Text` se utiliza cuando un usuario desea insertar un texto de varias líneas. Este widget se puede usar para una variedad de aplicaciones donde se requiere el texto de varias líneas, como mensajería, envío de información o visualización de información y muchas otras tareas. Podemos insertar archivos multimedia como imágenes y enlaces.

Sintaxis:

```
T = Text(parent, bg, fg, bd, width, height, font, ...)
```

donde `parent` es la ventana principal en la que se mostrará el widget.

Opciones o parámetros opcionales

- **bg:** color de fondo
- **fg:** color de primer plano
- **bd:** borde del widget.
- **height:** altura del widget.
- **width:** ancho del widget.
- **font:** tipo de fuente del texto.

- **cursor:** el tipo de cursor que se utilizará.
- **insetofftime:** el tiempo en milisegundos durante el cual el parpadeo del cursor está apagado.
- **insertontime:** el tiempo en milisegundos durante el cual el cursor parpadea.
- **padx:** acolchado horizontal o espacio externo.
- **pady:** acolchado vertical o espacio externo.
- **estate:** define si el widget responderá a los movimientos del mouse o del teclado.
- **highlightthickness**: define el grosor del resaltado de enfoque.
- **insertionwidth:** define el ancho del carácter de inserción.
- **relief:** tipo de borde que puede ser HUNDIDO, ELEVADO, RANURADO y CRESTA.
- **yscrollcommand:** para hacer que el widget se pueda desplazar verticalmente.
- **xscrollcommand:** para hacer que el widget se pueda desplazar horizontalmente.

Algunos métodos comunes

- **index(index):** Para obtener el índice especificado.
- **insert(index):** Para insertar una cadena en un índice especificado.
- **see(index):** comprueba si una cadena está visible o no en un índice dado.
- **get(startindex, endindex):** para obtener caracteres dentro de un rango dado.
- **delete(startindex, endindex):** elimina caracteres dentro del rango especificado.

Métodos de manejo de etiquetas

- **tag_delete(tagname):** Para eliminar una etiqueta determinada.
- **tag_add(tagname, startindex, endindex):** para etiquetar la cadena en el rango especificado
- **tag_remove(tagname, startindex, endindex):** para eliminar una etiqueta del rango especificado

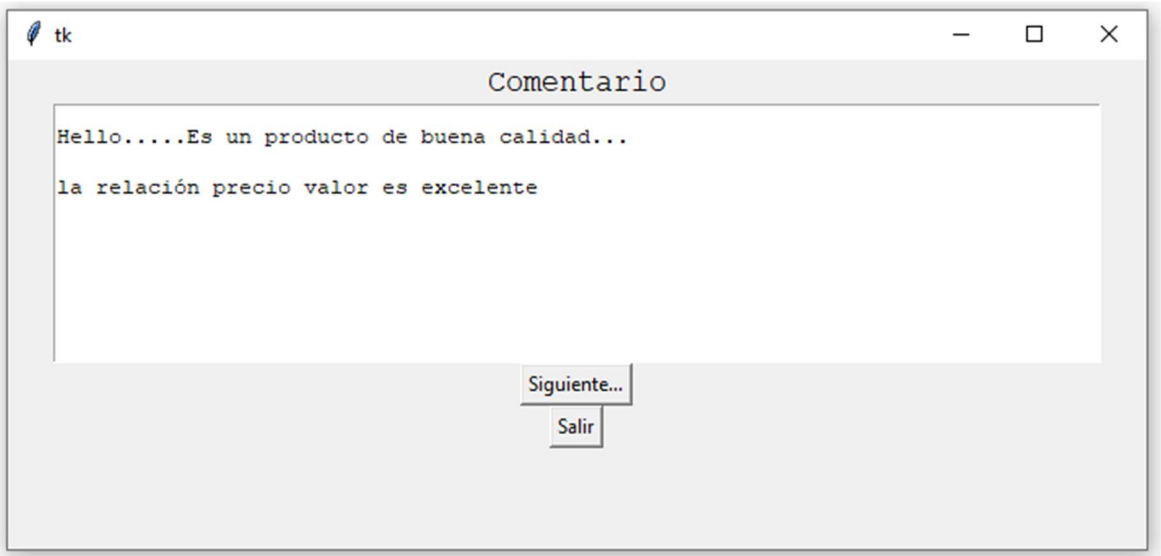
Métodos de manejo de marcas

- **mark_names():** para obtener todas las marcas en el rango dado.
- **index(mark):** para obtener el índice de una marca.
- **mark_gravity():** para obtener la gravedad de una marca determinada.

EjemploClase_9_Text.py

```
from tkinter import *

root = Tk()
root.geometry("700x300")

# Create text widget and specify size.
T = Text(root, undo=True, maxundo=100,
         spacing1=10, spacing2=2,
```

```
            spacing3=5, height=5, wrap='char'
)

# Crear una etiqueta
etiqueta = Label(root, text="Comentario")
etiqueta.config(font=("Courier", 14))

Comentario = """Es un producto de buena calidad...
la relación precio valor es excelente"""

# Crear botón Siguiente
boton_1 = Button(root, text="Siguiente...", )

# Crear botón Salir
boton_2 = Button(root, text="Salir",   command=root.destroy)

etiqueta.pack()
T.pack()
boton_1.pack()
boton_2.pack()

# Insertar Texto y la variable Comentario
T.insert(INSERT, "Comentario: ")
T.insert(END, Comentario)

# Obtener el contenido del texto
contents = T.get('1.0', 'end') # Extrae todo el texto
print(contents)

root.mainloop()
```



Widget MessageBox

Tkinter proporciona una clase `messagebox` que se puede usar para mostrar una variedad de mensajes para que el usuario pueda responder. Mensajes como mensaje de confirmación, mensaje de error, mensaje de advertencia, etc.

Para usar esta clase, se debe importar esta clase como se muestra a continuación:

```
from tkinter import messagebox
```

Sintaxis:

```
messagebox.Function_Name(title, message [, options])
```

Parámetros:

Function_Name: este parámetro se utiliza para representar una función de la ventana mensaje . Las funciones o métodos disponibles en el widget del cuadro de mensajes son:

- **showinfo():** muestra información relevante para el usuario.
- **showwarning():** muestra la advertencia al usuario.
- **showerror():** muestra el mensaje de error al usuario.
- **askquestion ():** haga una pregunta y el usuario debe responder sí o no.
- **askokcancel():** Confirma la acción del usuario con respecto a alguna actividad de la aplicación.
- **askyesno():** El usuario puede responder sí o no para alguna acción.
- **askretrycancel():** pregunta al usuario si puede volver a realizar una tarea en particular o no.

title: este parámetro es una cadena que se muestra como título de un cuadro de mensaje.

message: este parámetro es la cadena que se mostrará como un mensaje en el cuadro de mensaje.

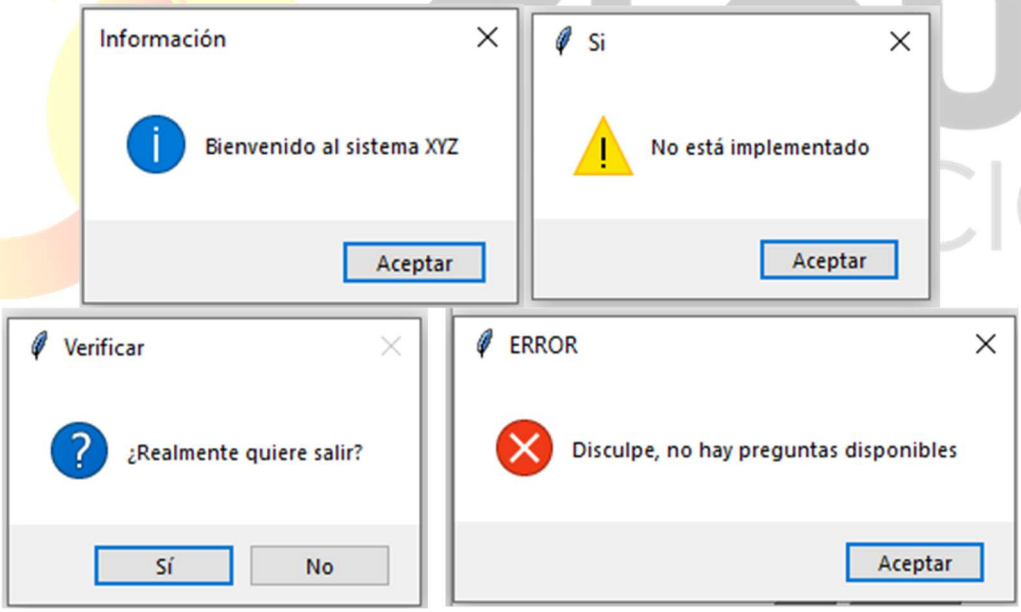
options: Hay dos opciones que se pueden utilizar son:

- **default:** esta opción se usa para especificar el botón predeterminado como ABORT, RETRY o IGNORE en el cuadro de mensaje.
- **parent:** esta opción se utiliza para especificar la ventana en la parte superior de la cual se mostrará el cuadro de mensaje.

Ejemplo (EjemploClase_10_MessageBox.py)

```
from tkinter import *
from tkinter import messagebox

def pregunta():
    # Mensaje de Error
```

```
    messagebox.showerror("ERROR", "Disculpe, no hay preguntas
disponibles")

def salir():
    # Mensaje Si o No
    if  messagebox.askyesno('Verificar', '¿Realmente quiere
salir?'):
        # Mensaje de Alerta
        messagebox.showwarning('Si', 'No está implementado')
    else:
        # Mensaje de tipo información
        messagebox.showinfo('No', 'Salir fue cancelado')

root = Tk()
root.geometry("300x200")

messagebox.showinfo('Información', 'Bienvenido al sistema XYZ')
Button(text='Salir', command=salir).place(x=170, y=150)
Button(text='Pregunta', command=pregunta).place(x=210, y=150)

root.mainloop()
```



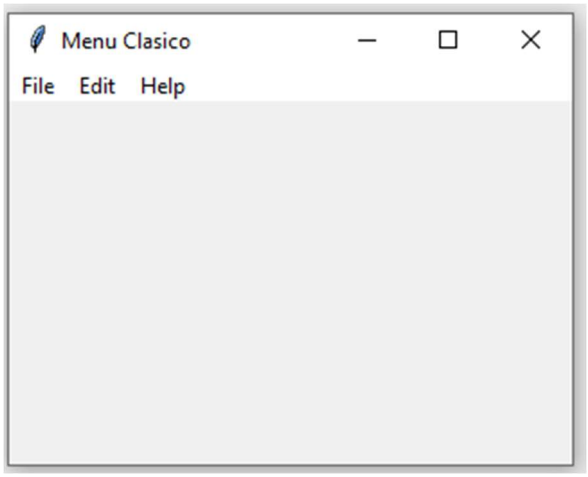
Widget Menu

Los menús son la parte importante de cualquier GUI. Un uso común de los menús es brindar acceso a varias operaciones, como guardar o abrir un archivo, salir de un programa o manipular datos. Los menús de nivel superior se muestran justo debajo de la barra de título de la raíz o cualquier otra ventana de nivel superior.

```
menu = Menu(parent, **options)
```

donde `parent` es la ventana principal en la que se mostrará el widget.

Ejemplo (EjemploClase_11_Menu.py)




```
from tkinter import *

# Crear la ventana
root = Tk()
root.title('Menu Clasico')

# Crear la Barra de Menu
menubar = Menu(root)

# Creando el Menú File y sus
opciones
file = Menu(menubar, tearoff=0)
menubar.add_cascade(label='File', menu=file)
file.add_command(label='New File', command=None)
file.add_command(label='Open...', command=None)
file.add_command(label='Save', command=None)
file.add_separator()
file.add_command(label='Exit', command=root.destroy)

# Creando el Menú Edit y sus opciones
edit = Menu(menubar, tearoff=0)
menubar.add_cascade(label='Edit', menu=edit)
edit.add_command(label='Cut', command=None)
edit.add_command(label='Copy', command=None)
edit.add_command(label='Paste', command=None)
edit.add_command(label='Select All', command=None)
edit.add_separator()
edit.add_command(label='Find...', command=None)
edit.add_command(label='Find again', command=None)

# Creando el Menú File y sus opciones
help_ = Menu(menubar, tearoff=0)
menubar.add_cascade(label='Help', menu=help_)
help_.add_command(label='Tk Help', command=None)
help_.add_command(label='Demo', command=None)
help_.add_separator()
help_.add_command(label='About Tk', command=None)

# Mostrar Menú
root.config(menu=menubar)

mainloop()
```

Widget Frame

Un `frame` o marco es un widget que se muestra como un simple rectángulo. Nos ayuda a organizar la interfaz de usuario, a menudo tanto visualmente como en la codificación. Los frames o marcos a menudo actúan como widgets maestros para un administrador de geometría como grid, que gestiona los widgets esclavos contenidos en el marco.

Los frames o marcos se crean utilizando la clase ttk.Frame:

```
marco = ttk.Frame(parent)
```

donde `parent` es la ventana principal en la que se mostrará el widget. Los frames se pueden configurar de formas diferentes y alterar la forma en que se muestran.

```
# Bordes
frame['borderwidth'] = 2  # ancho del borde
frame['relief'] = 'sunken' # tipo de borde

# Definir un estilo
s = ttk.Style() s.configure('Danger.TFrame', background='red',
                            borderwidth=5, relief='raised')
ttk.Frame(root, width=200, height=200,
style='Danger.TFrame').grid()
```

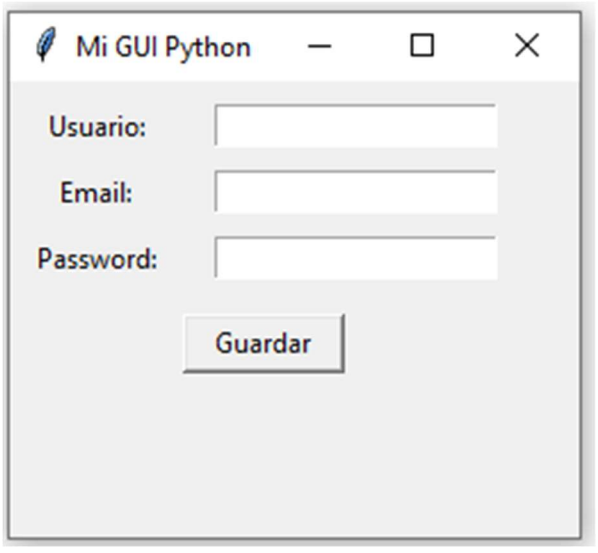
Ejemplo (EjemploClase_12_Frame.py)

```
from tkinter import *
from tkinter.ttk import *

# Crear la ventana principal
root = Tk()
root.geometry(newGeometry='800x600')
root.resizable(width=False, height=False)
root.title("Crear un Frame y un Botón")

# Definir un estilo para el frame
s = Style()
s.configure('Danger.TFrame', background='red', borderwidth=5,
             relief='raised'  ) #'sunken', background='red'

# Crear un frame en la ventana principal
frame = Frame(root, width=200, height=200,
style='Danger.TFrame')
frame.grid()

# Agregar botones en el frame de root
button = Button(frame, text='Mi botón').place(x=40,y=80)
```

```
    # Agregar etiqueta en el frame de root
    etiqueta = Label(frame, text="Etiqueta")
    etiqueta.place(x=40,y=60)

    root.mainloop()
```

Ejemplo (EjemploClase_13_Frame.py)

```
from tkinter import *

root = Tk()
root.title('Mi GUI Python')
root.geometry('250x200')

# Crear el frame para las etiquetas
frame1 = Frame(root, padx=5, pady=5)
frame1.grid(row=0, column=1)

Label(frame1, text='Usuario: ', padx=5, pady=5).pack()
Label(frame1, text='Email: ', padx=5, pady=5).pack()
Label(frame1, text='Password: ', padx=5, pady=5).pack()

# Crear el frame para las entradas
frame2 = Frame(root, padx=5, pady=5)
frame2.grid(row=0, column=2)

Entry(frame2).pack(padx=5, pady=5)
Entry(frame2).pack(padx=5, pady=5)
Entry(frame2).pack(padx=5, pady=5)

Button(root, text='Guardar', padx=10).grid(row=1, columnspan=5,
pady=5)

root.mainloop()
```



Widget LabelFrame

El widget `LabelFrame` en tkinter se usa para dibujar un borde alrededor de los widgets secundarios que contiene junto con su propia etiqueta.

La sintaxis para el `Labelframe` es:

```
Labelframe_tk = LabelFrame (parent, options)
```

donde `parent` es la ventana principal en la que se mostrará el widget y options son las principales propiedades de LabelFrame:

- **bg:** Muestra el color de fondo del widget.
- **bd:** Muestra el ancho del borde. Igual que el ancho del borde.
- **cursor:** Puede usar el cursor en diferentes formas en la pantalla como círculo, punto, etc.
- **fg:** Determina el color de primer plano de la fuente que se utiliza para el widget.
- **font:** Determina el tipo de fuente que se utiliza para el widget.
- **width:** Especifica el ancho del widget.
- **height:** Determina la altura del widget.
- **labelAnchor:** Especifica la posición del texto dentro del widget.
- **labelwidget:** Especifica el widget utilizado para identificar las etiquetas. El texto se usa por defecto si no se define ningún valor.
- **highlightbackground:** Muestra el color de resaltado del fondo del widget de texto cuando se hace clic en él.
- **highlightcolor:** Muestra el color de resaltado del widget de marco de etiqueta cuando se hace clic en él.
- **highlightthickness:** Especifica el grosor del resaltado.
- **padx:** Agrega relleno en la dirección horizontal
- **pady**: Agrega relleno en la dirección vertical
- **relief:** Esto muestra diferentes tipos de bordes. Por defecto, tiene un borde FLAT.
- **text:** Especifica la cadena que tiene el texto de la etiqueta.

Ejemplo (EjemploClase_14_LabelFrame.py)

```
from tkinter import *

root = Tk()
root.geometry('300x300')

# Crear el Label Frame
labelframe_tk = LabelFrame(root, text="Titulo del Label Frame")
labelframe_tk.pack(fill="both", expand="yes")
```

```
# Insertar etiqueta
etiqueta = Label(labelframe_tk, text="Etiqueta")
etiqueta.pack()

root.mainloop()
```

Widget Canvas (lienzo)

El widget Canvas nos permite mostrar varios gráficos en la aplicación. Se puede utilizar para dibujar desde formas simples hasta gráficos complicados. También podemos mostrar varios tipos de widgets personalizados según nuestras necesidades.

Sintaxis:

```
C = Lienzo(parent, height, width, bd, bg, ..)
```

donde `parent` es la ventana principal en la que se mostrará el widget.

Parámetros opcionales:

- **height:** altura del widget de lienzo.
- **width:** ancho del widget de lienzo.
- **bg:** color de fondo del lienzo.
- **bd:** borde de la ventana del lienzo.
- **scrollregion (w, n, e, s):** tupla definida como una región para desplazarse hacia la izquierda, arriba, abajo y a la derecha.
- **highlightcolor**: que se muestra en el resaltado de enfoque.
- **cursor:** Se puede definir como un cursor para el lienzo que puede ser un círculo, un do, una flecha, etc.
- **confine:** decide si se puede acceder al lienzo fuera de la región de desplazamiento.
- **relief:** tipo de borde que puede ser UNKEN, RAISED, GROOVE y RIDGE.

Algunos métodos de dibujo comunes:

- Crear un óvalo

```
oval = C.create_oval(x0, y0, x1, y1, options)
```

- Crear un arco

```
arc = C.create_arc(20, 50, 190, 240, start=0,
                   extent=110, fill="red")
```

- Crear una línea

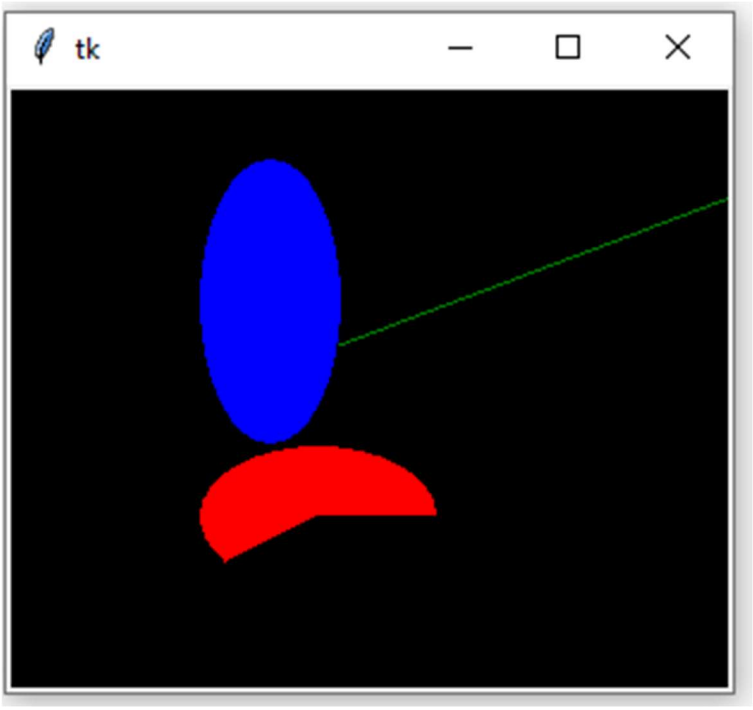
```
line = C.create_line(x0, y0, x1, y1, ..., xn, yn, options)
```

- Crear un polígono

```
poligono = C.create_polygon(x0, y0, x1, y1, ...xn, yn,
options)
```

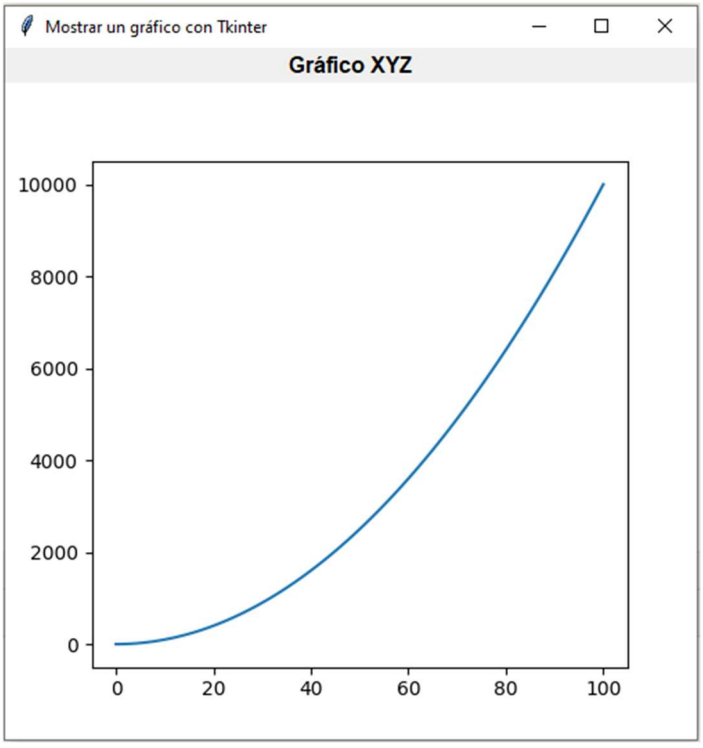
Ejemplo (EjemploClase_15_CanvasFG.py)

```
from tkinter import *

root = Tk()

C = Canvas(root, bg="black", height=250, width=300)

line = C.create_line(108, 120,
                     320, 40,
                     fill="green")

arc = C.create_arc(180, 150, 80,
                   210, start=0,
                   extent=220,
                   fill="red")

oval = C.create_oval(80, 30, 140,
                     150,
                     fill="blue")

C.pack()
root.mainloop()
```



Ejemplo (EjemploClase_16_CanvasGrafico.py)

```
from tkinter import *
from matplotlib.figure import Figure
from matplotlib.backends.backend_tkagg import (FigureCanvasTkAgg,
                                               NavigationToolbar2Tk)

def mostrar_grafico():
    fig = Figure(figsize=(5, 5), dpi=100)
    y = [i ** 2 for i in range(101)]
    plot1 = fig.add_subplot(111)
    plot1.plot(y)

    # Creamos el canvas o área donde se mostrara el gráfico
    canvas = FigureCanvasTkAgg(fig, master=root)
    canvas.draw()
    canvas.get_tk_widget().pack()

    toolbar = NavigationToolbar2Tk(canvas, root)
    toolbar.update()

root = Tk()
root.title('Mostrar un gráfico con Tkinter')
root.geometry("500x500")

Label(root, text="Gráfico XYZ", font=('calibre',12, 'bold')).pack()

mostrar_grafico()
root.mainloop()
```



Widget Checkbutton

El widget `Checkbutton` es como un botón que contiene un valor binario (1 o 0), utilizado para activar o desactivar opciones. Los botones de verificación pueden contener texto o imágenes. Cuando se presiona el botón, Tkinter llama a la función o método.

Sintaxis:

```
w = Checkbutton(parent, options)
```

donde `parent` es la ventana principal y `options` son pares clave-valor separados por coma, algunas opciones disponibles son:

- **activebackground:** esta opción solía representar el color de fondo cuando el botón de verificación está debajo del cursor.
- **activeforeground:** esta opción solía representar el color de primer plano cuando el botón de verificación está debajo del cursor.
- **bg:** esta opción solía representar el color de fondo normal que se muestra detrás de la etiqueta y el indicador.
- **bitmap:** esta opción solía mostrar una imagen monocromática en un botón.
- **bd:** esta opción solía representar el tamaño del borde alrededor del indicador y el valor predeterminado es de 2 píxeles.
- **command:** Esta opción está asociada con una función que se llamará cuando se cambie el estado del botón de verificación.
- **cursor:** al usar esta opción, el cursor del mouse cambiará a ese patrón cuando esté sobre el botón de verificación.
- **disabledforeground:** el color de primer plano utilizado para representar el texto de un botón de verificación deshabilitado. El valor predeterminado es una versión punteada del color de primer plano predeterminado.
- **font:** Esta opción se utiliza para representar la fuente utilizada para el texto.
- **fg:** esta opción solía representar el color utilizado para representar el texto.
- **height:** esta opción solía representar el número de líneas de texto en el botón de verificación y su valor predeterminado es 1.
- **highlightcolor:** esta opción solía representar el color del resaltado de enfoque cuando el botón de verificación tiene el foco.
- **image:** Esta opción solía mostrar una imagen gráfica en el botón.
- **justify:** esta opción se usa para controlar cómo se justifica el texto: CENTER, LEFT o RIGHT.
- **offvalue:** la variable de control asociada se establece en 0 de forma predeterminada si el botón no está marcado. Podemos cambiar el estado de una variable no marcada a alguna otra.

- **onvalue:** la variable de control asociada se establece en 1 de forma predeterminada si el botón está marcado. Podemos cambiar el estado de la variable marcada a alguna otra.
- **padx:** esta opción solía representar cuánto espacio dejar a la izquierda y derecha del botón de verificación y el texto. Su valor predeterminado es 1 píxel.
- **pady:** esta opción solía representar cuánto espacio dejar arriba y debajo del botón de verificación y el texto. Su valor predeterminado es 1 píxel.
- **relief:** El tipo de borde del botón de verificación. Su valor predeterminado se establece en FLAT.
- **selectcolor:** esta opción representa el color del botón de verificación cuando se configura. El valor predeterminado es selectcolor=”red”.
- **selectimage:** la imagen se muestra en el botón de verificación cuando se establece.
- **state:** Representa el estado del checkbutton. De forma predeterminada, se establece en NORMAL. Podemos cambiarlo a DISABLED para que el botón de verificación no responda. El estado del botón de verificación es ACTIVE cuando está bajo el foco.
- **text:** esta opción utiliza líneas nuevas ("\n") para mostrar varias líneas de texto.
- **underline:** Esta opción se utiliza para representar el índice del carácter en el texto que se va a subrayar. La indexación comienza con cero en el texto.
- **variable:** Esta opción solía representar la variable asociada que rastrea el estado del botón de verificación.
- **width:** Esta opción solía representar el ancho del botón de verificación. y también representado en la cantidad de caracteres que se representan en forma de textos.
- **wraplength:** esta opción dividirá el texto en el número de piezas.

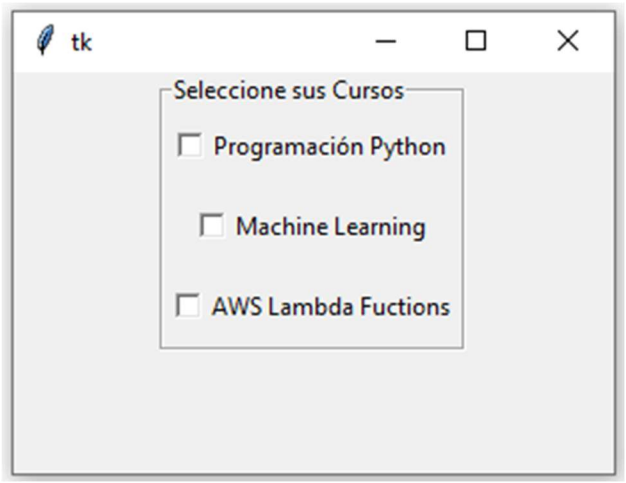
Métodos:

- **deselect():** Este método se llama para desactivar el botón de verificación.
- **flash():** el botón de verificación parpadea entre los colores activo y normal.
- **invoke():** este método invocará el método asociado con el botón de verificación.
- **select():** este método se llama para activar el botón de verificación.
- **toggle():** este método se utiliza para alternar entre los diferentes botones de verificación.

Ejemplo (EjemploClase_19_Checkbutton.py)

```
from tkinter import *
from tkinter import messagebox

def isChecked():
    if Checkbutton1.get() == 1:
        messagebox.showinfo('Cursos', 'Bienvenido a Programación
Python!')
    elif Checkbutton2.get() == 1:
        messagebox.showinfo('Cursos', 'Bienvenido a Machine
```

```
Learning!')
    elif Checkbutton3.get() == 1:
        messagebox.showinfo('Cursos', 'Bienvenido a AWS Lambda
Functions!')


root = Tk()
root.geometry("300x200")

l_frame = LabelFrame(root, text='Seleccione sus Cursos')
l_frame.pack()

Checkbutton1 = IntVar()
Checkbutton2 = IntVar()
Checkbutton3 = IntVar()

Button1 = Checkbutton(l_frame, text="Programación Python",
                      variable=Checkbutton1,
                      onvalue=1,
                      offvalue=0,
                      height=2,
                      padx=2,
                      command=isChecked)

Button2 = Checkbutton(l_frame, text="Machine Learning",
                      variable=Checkbutton2,
                      onvalue=1,
                      offvalue=0,
                      height=2,
                      padx=2,
                      command=isChecked)

Button3 = Checkbutton(l_frame, text="AWS Lambda Fuctions",
                      variable=Checkbutton3,
                      onvalue=1,
                      offvalue=0,
                      height=2,
                      padx=2,
                      command=isChecked)

Button1.pack()
Button2.pack()
Button3.pack()

mainloop()
```



Widget Radiobutton

`Radiobutton` es un widget estándar de Tkinter que se utiliza para implementar una de múltiples selecciones. Los `Radiobutton` pueden contener texto o imágenes, y puede asociar una función o método de Python con cada botón. Cuando se presiona el botón, Tkinter llama automáticamente a esa función o método.

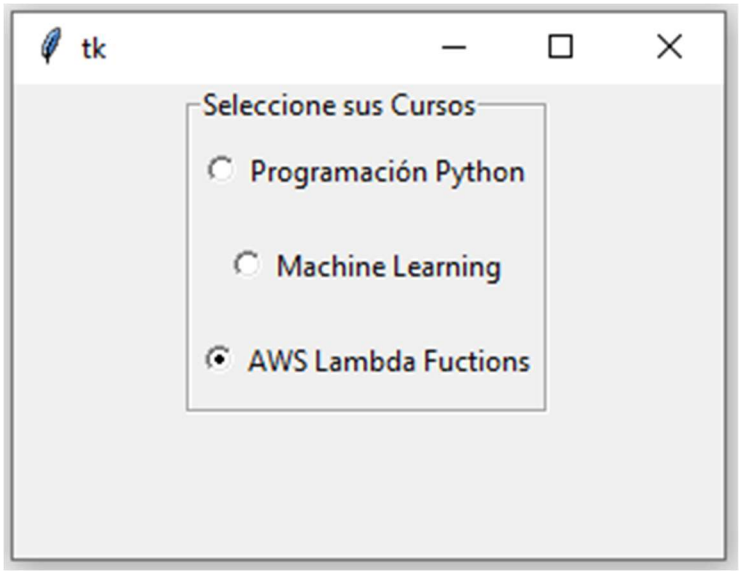
Sintaxis:

```
button = Radiobutton(parent, text, variable, value, options =
                     values, …)
```

Ejemplo (EjemploClase_20_Radiobutton.py)

```
from tkinter import *
from tkinter import messagebox

def isChecked():
    if seleccion.get() == 1:
        messagebox.showinfo('Cursos', 'Bienvenido a Programación
Python!')
    elif seleccion.get()  == 2:
        messagebox.showinfo('Cursos', 'Bienvenido a Machine
Learning!')
    elif seleccion.get()  == 3:
        messagebox.showinfo('Cursos', 'Bienvenido a AWS Lambda
Functions!')


root = Tk()
root.geometry("300x200")

l_frame = LabelFrame(root, text='Seleccione sus Cursos')
```

```
l_frame.pack()

seleccion = IntVar(root, "0")


Button1 = Radiobutton(l_frame, text="Programación Python",
                      variable=seleccion,
                      value=1,
                      height=2,
                      padx=2,
                      command=isChecked)

Button2 = Radiobutton(l_frame, text="Machine Learning",
                      variable=seleccion,
                      value=2,
                      height=2,
                      padx=2,
                      command=isChecked)

Button3 = Radiobutton(l_frame, text="AWS Lambda Fuctions",
                      variable=seleccion,
                      value=3,
                      height=2,
                      padx=2,
                      command=isChecked)

Button1.pack()
Button2.pack()
Button3.pack()

mainloop()
```

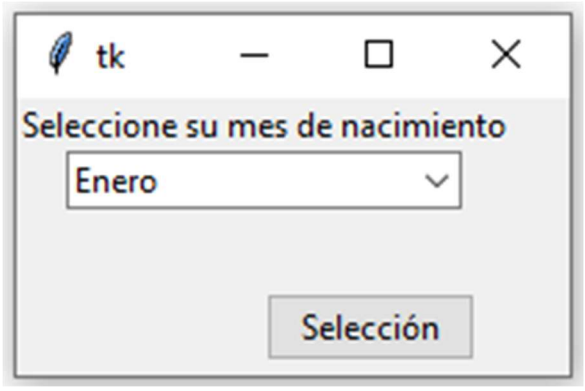


Widget Combobox

Tkinter Combobox es una lista desplegable para que el usuario pueda elegir. Es una combinación de Entry y de widgets desplegables. Al hacer clic en la flecha de la izquierda, muestra un menú desplegable con todas las opciones disponibles, al hace clic en una de ellas, reemplazará el contenido actual de Entry.

Ejemplo (EjemploClase_21_Combobox.py)

```
from tkinter import *
from tkinter.ttk import *

def nueva_seleccion():
    Label(root, text=comboExample.get()).place(x=20, y=50)

root = Tk()
root.geometry('200x100')

labelTop = Label(root, text="Seleccione su mes de nacimiento")
labelTop.grid(column=0, row=0)

comboExample = Combobox(root,
                        values=[
                                "Enero", "Febrero", "Marzo",
                                "Abril", "Mayo", "Junio",
                                "Julio", "Agosto", "Septiembre",
                                "Octubre", "Noviembre", "Diciembre",
                               ])
comboExample.grid(column=0, row=1)
comboExample.current(0)

botonCalcular = Button(root, text="Selección")
botonCalcular.place(x=90, y=70)
botonCalcular.configure(command=nueva_seleccion)

root.mainloop()
```



Widget Panedwindow

`Panedwindow` en tkinter es un contenedor que mantiene juntas una o más sub-paned windows que contienen sus propias sub-panedwindows y así sucesivamente. Siempre podemos cambiar el tamaño de estos paneles a nuestra elección usando un mouse. Cada panel contiene un widget. Podemos implementar los paneles en forma horizontal y vertical. La sintaxis de panedwindow es:

```
Panedwindow_tk = PanedWindow (ventana, options)
```

Las características y propiedades de Panedwindow son:

- **bg:** Declara el color de fondo del widget.
- **bd:** Muestra el tamaño del borde 3D.
- **borderwidth:** Especifica el ancho del borde de representación del widget. El ancho predeterminado es 2px.
- **cursor:** Muestra el tipo de cursor cuando se pasa el mouse sobre PanedWindow. El valor predeterminado es un cursor de flecha puntiaguda.
- **handlepad:** 8 es el tamaño predeterminado.
- **handlesize:** 8 es el tamaño predeterminado.
- **height:** Decide la altura de la ventana acristalada.
- **orient:** La orientación por defecto es HORIZONTAL.
- **relief:** Esto muestra diferentes tipos de bordes como SUNKEN, RAISED, GROOVE o RIDGE.
- **sashpad:** Especifica el relleno alrededor de cada hoja. El valor predeterminado es 0.
- **sashrelief:** Esto muestra diferentes tipos de bordes de marco.
- **sashwidth:** Especifica el ancho de la hoja. El valor predeterminado es 2px.
- **showhandle:** Este método se establece en True para mostrar los identificadores. El valor predeterminado es FALSE.
- **width:** Esta característica establece el ancho de la ventana.

Los métodos para los widgets de una ventana de panel son:

- **add(child, options):** Este método se usa para agregar ventanas con paneles dentro del rango especificado.
- **get(startindex, endindex):** Obtiene el índice absoluto del índice especificado
- **config(opciones):** Se configura con widget dentro de las opciones especificadas.

Ejemplo (EjemploClase_17_PanedWindows.py)

```
from tkinter import *

# Crear primer Panel
screen = PanedWindow(bg="red")
```

```
screen.pack(fill=BOTH, expand=1)

# Insertar etiqueta al primer panel
left = Label(screen, text="Panel Izquierdo" , bg="green")
screen.add(left)

# Crear segundo Panel
m2 = PanedWindow(screen, orient=VERTICAL)
screen.add(m2)

# Insertar etiquetas
top = Label(m2, text="Panel superior")
m2.add(top)
top1 = Label(m2, text="Panel superior 2")
m2.add(top1)

# Crear tercer Panel
m3 = PanedWindow(screen,bg="blue" ,orient=HORIZONTAL)
m2.add(m3)

# Insertar etiqueta
bottom = Label(m3, text="Panel inferior",bg="blue")
m3.add(bottom)

mainloop()
```

Widget Toplevel (Abrir una nueva ventana)

Cuando se ejecuta un programa Tkinter, ejecuta un bucle principal (`mainloop`) que es responsable de ejecutar un programa GUI. Solo puede haber una instancia de `mainloop` activa, por lo que para abrir una nueva ventana tenemos que usar un widget, `Toplevel`.

El widget `Toplevel` funciona de manera muy similar a un marco, pero se abre en una ventana de nivel superior separada, tales ventanas tienen todas las propiedades que debería tener una ventana principal.

Ejemplo (EjemploClase_18_NuevaVentana.py)

```
from tkinter import *
from tkinter.ttk import *

# Crear la ventana principal
root = Tk()
root.geometry("200x200")

# Function para crear una nueva ventana
def abrir_nueva_ventana():

    # Objeto de nivel superior que será tratado como una nueva
```

```
ventana
    newWindow = Toplevel(root)
    newWindow.title("Nueva Ventana")  # título de la nueva
ventana
    newWindow.geometry("200x200")

    # Crear una etiqueta en la nueva ventana
    Label(newWindow,
          text="Esta es una nueva Ventana").pack()

# Crear una etiqueta en la ventana principal
label = Label(root, text="Esta es la Ventana Principal")
label.pack(pady=10)

btn = Button(root,
             text="Nueva Ventana",
             command=abrir_nueva_ventana)
btn.pack(pady=10)

mainloop()
```