



# Visual Programming

**College Of Computer Sciences and Mathematics**

**Dr. Firas Aswad**

**Dept. of Networks**

**Feb. 5, 2025**



# Tkinter includes 21 core widgets



**Top-level**

**Canvas**

**Frame**

**Menu**

**OptionMenu**

**Scale**

**Text**

**Label**

**Checkbutton**

**LabelFrame**

**Menubutton**

**PanedWindow**

**Scrollbar**

**Bitmap**

**Button**

**Entry**

**Listbox**

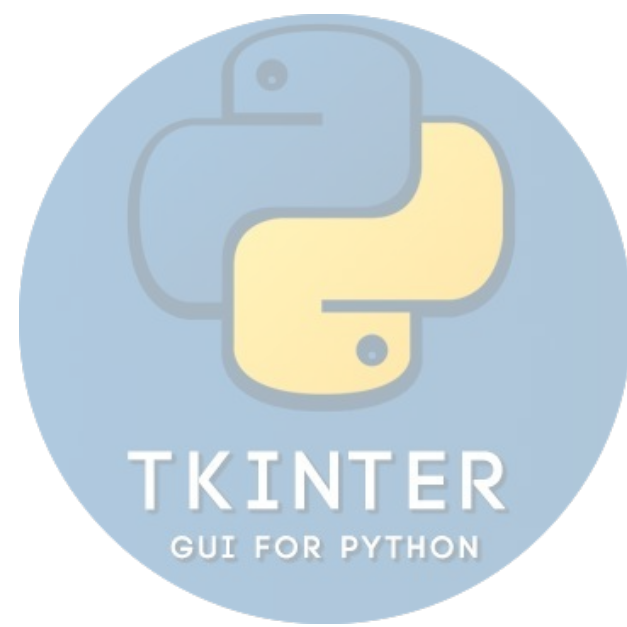
**Message**

**Radiobutton**

**Spinbox**

**Image**

**Label and Button**



# The Label widget

This is a label



The Ttk Label widget shares most of the same options as the Tk version, the most common of which are listed here:

Argument	Values	Description
text	String	The text content of the label
textvariable	StringVar	The variable to bind to the contents of the label
anchor	Cardinal direction	The position of the text relative to the inner margins
justify	left, right, or center	The alignment of the lines of text relative to one another
foreground	Color string	The color of the text
wraplength	Integer	Number of pixels before the text is wrapped to the next line
underline	Integer	The index of a character in text to underline
font	Font string or tuple	The font to be used



# The Entry widget

Text entry box



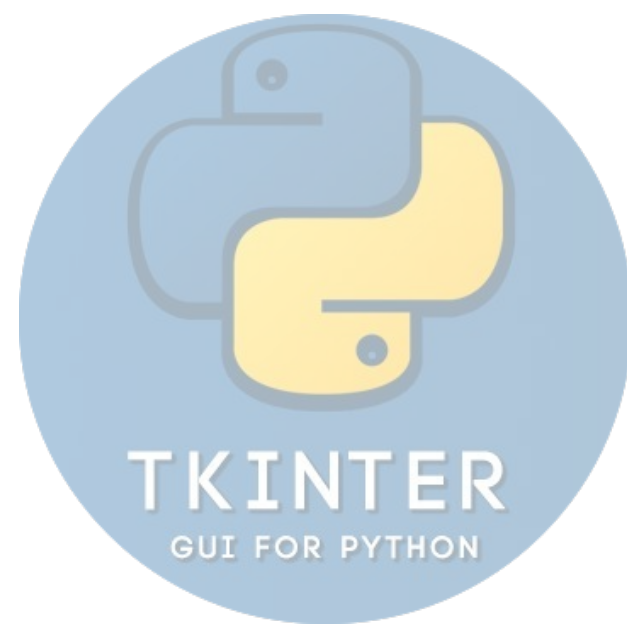
We can create an Entry widget using this code:

```
myentry = ttk.Entry(root, textvariable=my_string_var, width=20)
```

The Ttk Entry is very similar to the Tkinter Entry widget we've already seen, and supports many of the same arguments. Here is a selection of the more common Entry options:

Argument	Values	Description
textvariable	StringVar	Tkinter control variable to bind.
show	String	Character or string to show when the user types. Useful for password fields, for example.
justify	left, right, or center	Alignment of the text in the entry. left is default.
foreground	Color string	Color of text.





# The Spinbox widget



```
mypinbox = ttk.Spinbox(
    root,
    from_=0, to=100, increment=.01,
    textvariable=my_int_var,
    command=my_callback
)
```

Argument	Values	Description
from_	Float or Int	Minimum value the arrows will decrement to.
to	Float or Int	Maximum value the arrows will increment to.
increment	Float or Int	Value that will be added or subtracted by the arrows.
command	Python function	Callback to be executed when either button is pushed.
textvariable	Control variable (any type)	Variable bound to the field value.
values	List of strings or numbers	Set of choices the buttons will scroll through. Overrides the from_ and to values.



# The Checkbutton widget

☐ Would you like Pi?



```
mycheckbutton = ttk.Checkbutton(  
    root,  
    variable=my_bool_var,  
    textvariable=my_string_var,  
    command=my_callback  
)
```

Checkbutton widgets can take a number of arguments in addition to those listed above, as shown in this table:

Argument	Values	Description
variable	Control variable	The variable to which the checked/ unchecked state of the box is bound
text	String	The label text
textvariable	StringVar	The variable to which the label text is bound
command	Python function	A callback to execute whenever the box is checked or unchecked
onvalue	Any	Value to set <code>variable</code> when the box is checked
offvalue	Any	Value to set <code>variable</code> when the box is unchecked
underline	Integer	Index of a character in <code>text</code> to underline



# The Radiobutton widget



The following code shows how to create these buttons:

```
buttons = tk.Frame(root)
r1 = ttk.Radiobutton(
    buttons,
    variable=my_int_var,
    value=1,
    text='One'
)
r2 = ttk.Radiobutton(
    buttons,
    variable=my_int_var,
    value=2,
    text='Two'
)
```



# The Radiobutton widget



This table shows some of the various arguments you can use with a Radiobutton:

Argument	Values	Description
variable	Control variable	A variable to be bound to the button's selected state
value	Any	A value to set the variable to when the button is selected
command	Python function	A callback to execute when the button is clicked
text	String	The label connected to the radio button
textvariable	StringVar	A variable bound to the button's label text
underline	Integer	Index of a character in text to underline





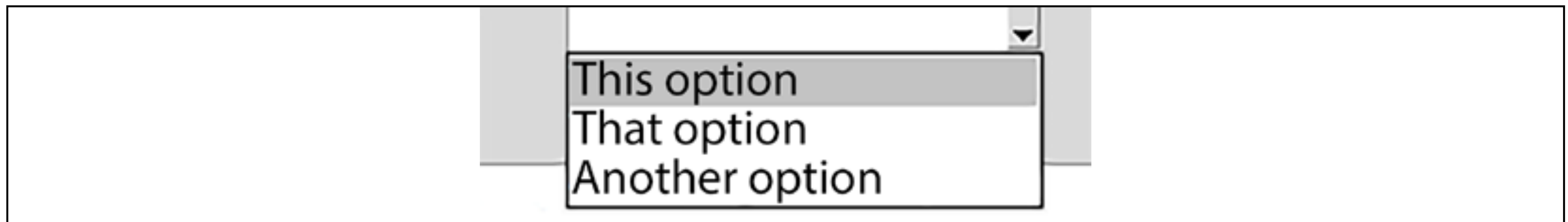
# The Combobox widget



We can create a Combobox widget like so:

```
mycombo = ttk.Combobox(  
    root, textvariable=my_string_var,  
    values=['This option', 'That option', 'Another option']  
)
```

Running that code will give us a combo box that looks something like this:





# The Combobox widget



This table shows some of the common arguments used with a Combobox:

Argument	Values	Description
textvariable	StringVar	Variable bound to the contents of the Combobox
values	List of strings	Values to populate the drop-down listbox
postcommand	Python function	Callback to run just before the listbox is displayed
justify	left, right, or center	Alignment of text in the box

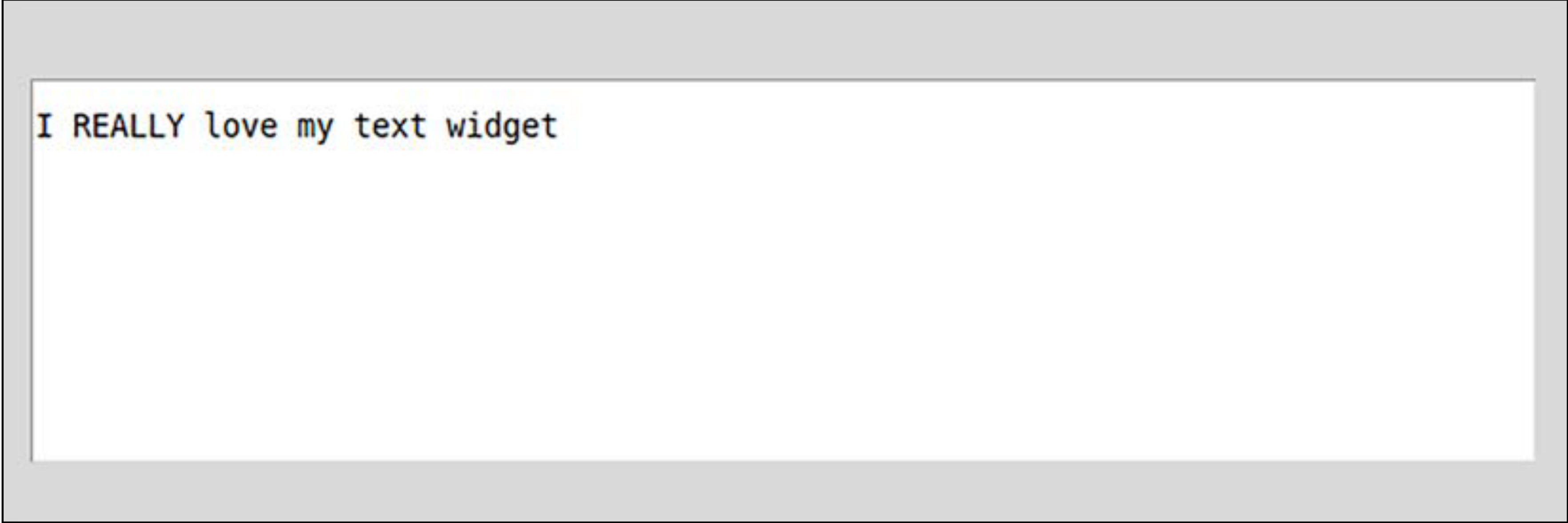


# The Text widget



```
mytext = tk.Text(  
    root,  
    undo=True, maxundo=100,  
    spacing1=10, spacing2=2, spacing3=5,  
    height=5, wrap='char'  
)
```

The above code will produce something that looks like this:



I REALLY love my text widget



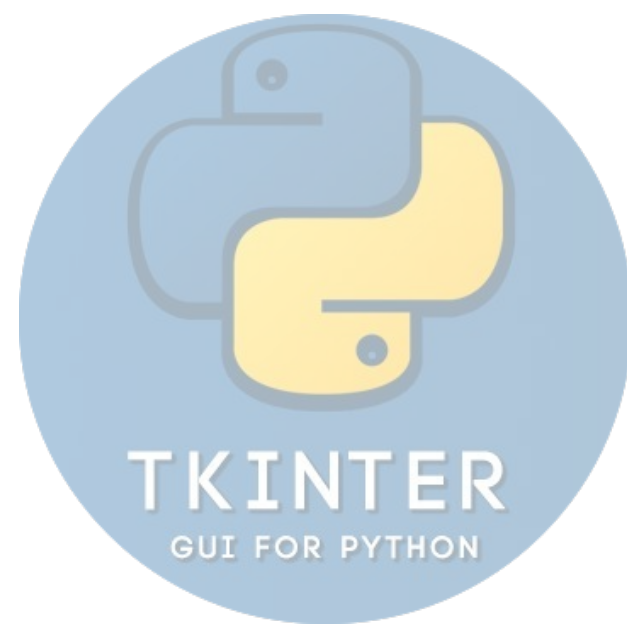
# The Text widget



The Text widget has a large number of arguments we can specify to control its appearance and behavior. Some of the more useful ones are listed in this table:

Argument	Values	Description
height	Integer	Height of the widget in lines of text.
width	Integer	Width of the widget in number of characters. For variable-width fonts, the width of a "0" character is used to calculate the width.
undo	Boolean	Activates or deactivates the undo functionality. Undo and redo actions are activated using the platform's default shortcuts.
maxundo	Integer	Maximum number of edits that will be stored for undo.
wrap	none, char, or word	Specifies how a line of text will be broken and wrapped when it exceeds the width of the widget.
spacing1	Integer	The number of pixels to pad above each complete line of text.
spacing2	Integer	The number of pixels to pad between displayed lines of wrapped text.
spacing3	Integer	The number of pixels to pad below each complete line of text.





# The Button widget



```
mybutton = ttk.Button(  
    root,  
    command=my_callback,  
    text='Click Me!',  
    default='active'  
)
```

The button is a pretty straightforward widget, but it has a few options that can be used to configure it. These are shown in the table below:

Arguments	Values	Description
text	String	Label text on the button.
textvariable	StringVar	Variable bound to the label text of the button.
command	Python function	Callback to be executed when the button is clicked.
default	normal, active, disabled	If the button executes when Enter is pushed. active means it will execute in response to Enter, normal means it will only if selected first, and disabled means it will not respond to Enter.
underline	Integer	Index of a character in text to underline.

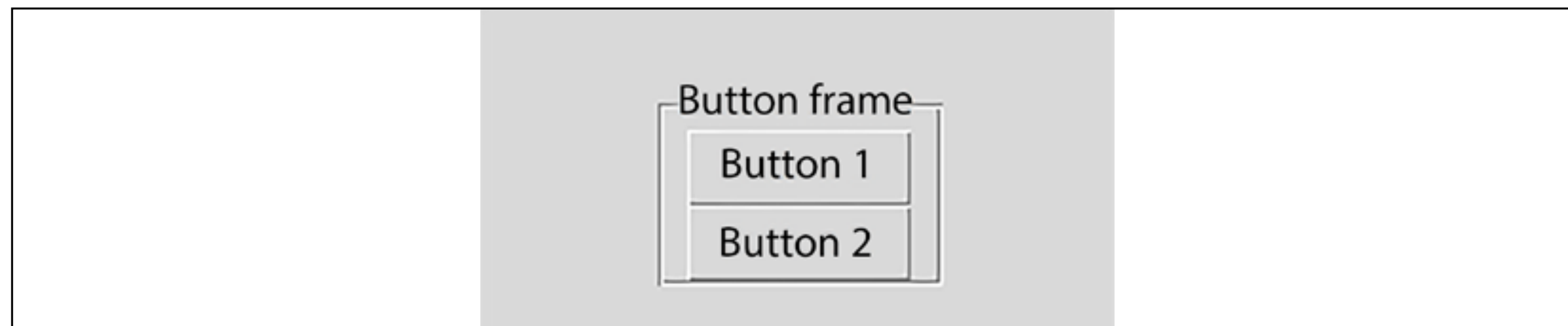


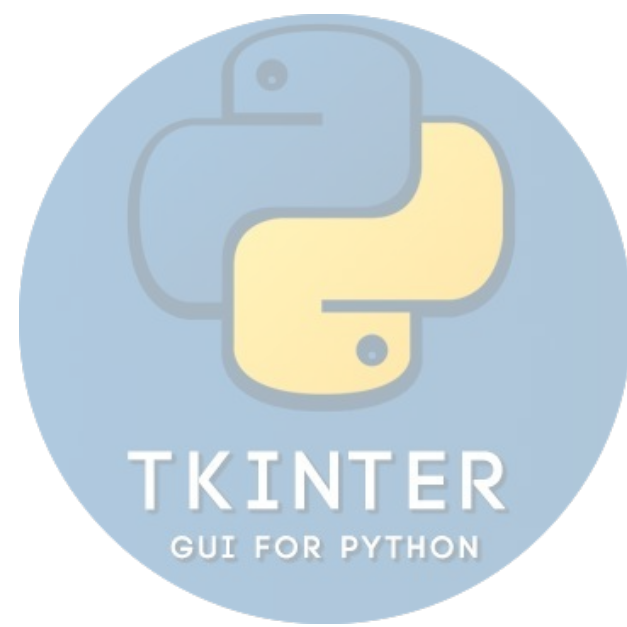
# The LabelFrame widget



```
mylabelframe = ttk.LabelFrame(  
    root,  
    text='Button frame'  
)  
  
b1 = ttk.Button(  
    mylabelframe,  
    text='Button 1'  
)  
b2 = ttk.Button(  
    mylabelframe,  
    text='Button 2'  
)  
b1.pack()  
b2.pack()
```

The resulting GUI would look like this:





# The LabelFrame widget



The LabelFrame widget offers us a few arguments for configuration, shown here:

Argument	Values	Description
text	String	The text of the label to display.
labelanchor	Cardinal direction	Where to anchor the text label.
labelwidget	ttk.Label object	A label widget to use for the label. Overrides text.
underline	Integer	The index of a character in text to underline.



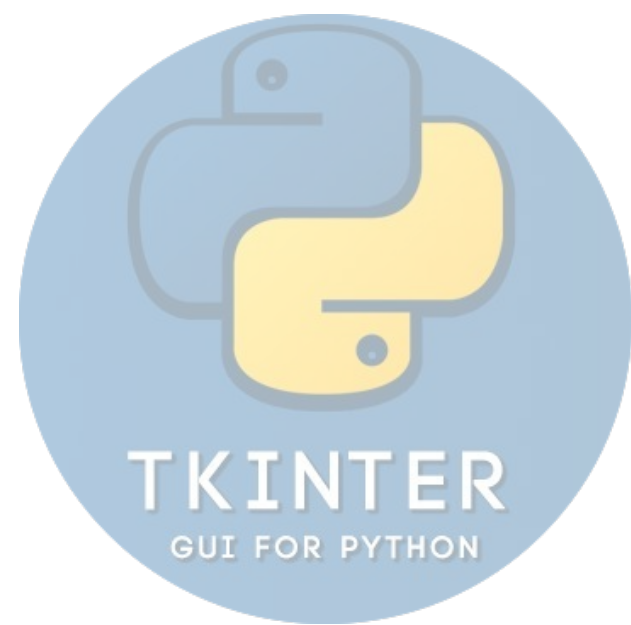
# 1. Common Options Across Most Widgets

These options are shared by most widgets in `tkinter` . They control layout, appearance, and behavior.

OPTION	DESCRIPTION	EXAMPLE
<code>anchor</code>	Specifies where text or content is positioned within the widget. Values: <code>n</code> , <code>ne</code> , <code>e</code> , etc.	<code>Label(root, text="Hello", anchor="nw")</code>
<code>background</code>	Sets the background color of the widget.	<code>Button(root, text="Click", background="")</code>
<code>bd</code> / <code>borderwidth</code>	Sets the border width of the widget.	<code>Label(root, text="Border", bd=5)</code>
<code>cursor</code>	Changes the mouse cursor when hovering over the widget.	<code>Button(root, text="Hover", cursor="hand")</code>
<code>font</code>	Specifies the font used for text.	<code>Label(root, text="Font", font=("Arial",</code>
<code>foreground</code> / <code>fg</code>	Sets the text or foreground color.	<code>Label(root, text="Red Text", fg="red")</code>
<code>height</code>	Sets the height of the widget (in characters for text-based widgets).	<code>Text(root, height=10)</code>
<code>highlightbackground</code>	Sets the color of the focus highlight when the widget does not have focus.	<code>Entry(root, highlightbackground="yellow")</code>
<code>highlightcolor</code>	Sets the color of the focus highlight when the widget has focus.	<code>Entry(root, highlightcolor="green")</code>
<code>highlightthickness</code>	Sets the thickness of the focus highlight.	<code>Entry(root, highlightthickness=2)</code>
<code>justify</code>	Aligns text within the widget. Values: <code>left</code> , <code>center</code> , <code>right</code> .	<code>Label(root, text="Justify", justify="ce")</code>
<code>padx</code> / <code>pady</code>	Adds horizontal ( <code>padx</code> ) or vertical ( <code>pady</code> ) padding inside the widget.	<code>Button(root, text="Pad", padx=10, pady=)</code>
<code>relief</code>	Specifies the border style. Values: <code>flat</code> , <code>raised</code> , <code>sunken</code> , <code>groove</code> , <code>ridge</code> .	<code>Label(root, text="Relief", relief="rais")</code>
<code>state</code>	Controls whether the widget is active, disabled, or normal.	<code>Button(root, text="Disabled", state="di")</code>
<code>takefocus</code>	Determines if the widget can receive focus during tab navigation.	<code>Entry(root, takefocus=True)</code>
<code>width</code>	Sets the width of the widget (in characters for text-based widgets).	<code>Entry(root, width=20)</code>







# Thank You!