GLS UNIVERSITY

0301601 INTRODUCTION TO PYTHON
UNIT– V

Tkinter

Introduction to Python GUI

- Python provides various options for developing graphical user interfaces (GUIs). Most important are listed below.

- **Tkinter:** It is the easiest among all to get started with. It is Python’s standard GUI (Graphical User Interface) package. It is the most commonly used toolkit for GUI Programming in Python.
- **JPython:** It is the Python platform for Java that is providing Python scripts seamless access o Java class Libraries for the local machine.
- **wxPython:** It is open-source cross-platform GUI toolkit written in C++. It one of the alternatives to Tkinter, which is bundled with Python.

- There are many other interfaces available for GUI. But these are the most commonly used ones.

Python GUI Programming - Tkinter

- It is the standard GUI toolkit for Python.
- It was written by Fredrik Lundh.
- Python when combined with Tkinter provides a fast and easy way to create GUI applications.
- Tkinter provides a powerful object-oriented interface to the Tk GUI toolkit.
- Creating a GUI application using Tkinter is an easy task.
- Tkinter is an acronym for "Tk interface".

Note: Tkinter comes pre-installed with Python3, and you need not bother about installing it.

Python GUI Programming - Tkinter

Install Tkinter

- To see if you have Tkinter, launch python; then at the Python prompt, type:
  - import Tkinter
  - Or in Python 3: import tkinter

sudo apt-get install python-tk (for version 2)

sudo apt-get install python3-tk (for versoin 3)

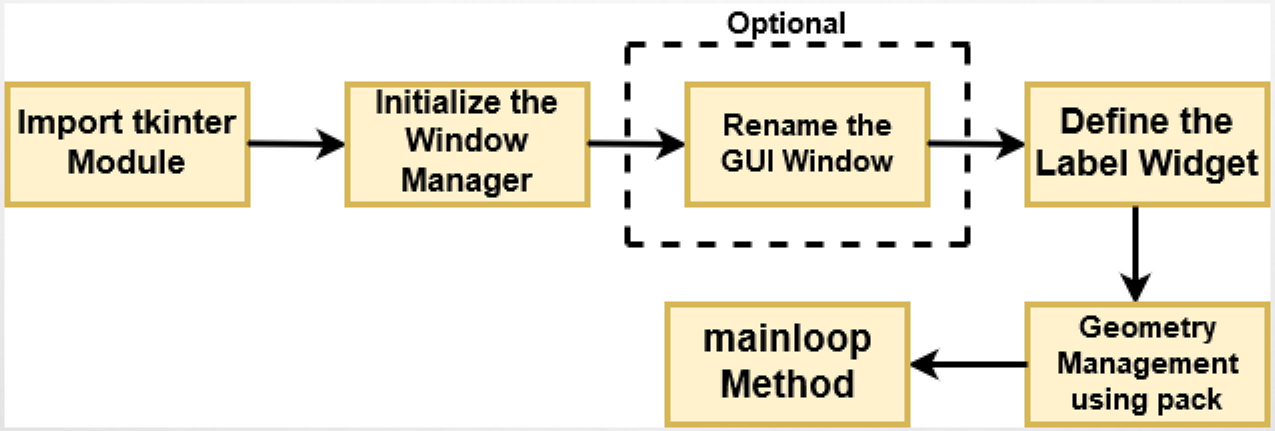
Python GUI Programming - Tkinter

There is method used you the user need to remember while creating the Python application with GUI.

Tk(screenName=None, baseName=None, className=’Tk’, useTk=1):

The basic code used to create the main window of the application is:

m=tkinter.Tk() where m is the name of the main window object



Steps:

- First, you import the key component, i.e., the Tkinter module.
- As a next step, you initialize the window manager with the tkinter.Tk() method and assign it to a variable. This method creates a blank window with close, maximize, and minimize buttons on the top as a usual GUI should have.
- Then as an optional step, you will Rename the title of the window as you like with window.title(title_of_the_window).
- Next, you make use of a widget called Label, which is used to insert some text into the window.
- Then, you make use of Tkinter's geometry management attribute called pack() to display the widget in size it requires.
- Finally, as the last step, you use the mainloop() method to display the window until you manually close it. It runs an infinite loop in the backend.

Python GUI Programming - Tkinter

mainloop():

- There is a method known by the name mainloop() is used when you are ready for the application to run.
- mainloop() is an infinite loop used to run the application, wait for an event to occur and process the event till the window is not closed.

```
m.mainloop()
```

Example:

```
import tkinter
m = tkinter.Tk()
m.mainloop()
```

Python GUI Programming - Tkinter

tkinter also offers access to the **geometric configuration** of the widgets which can organize the widgets in the parent windows.
There are mainly three geometry manager classes class.

pack() method:
It organizes the widgets in blocks before placing in the parent widget.

grid() method:
It organizes the widgets in grid (table-like structure) before placing in the parent widget.

place() method:
It organizes the widgets by placing them on specific positions directed by the programmer.

Pack() method

- Pack is the easiest to use of the three geometry managers of Tk and Tkinter.
- Instead of having to declare precisely where a widget should appear on the display screen, we can declare the positions of widgets with the pack command relative to each other.
- The pack command takes care of the details.
- Though the pack command is easier to use, this layout managers is limited in its possibilities compared to the grid and place mangers.
- For simple applications it is definitely the manager of choice.
- For example simple applications like placing a number of widgets side by side, or on top of each other.

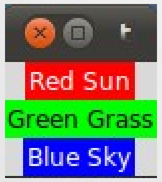
Pack() method

```
from tkinter import *

root = Tk()

Label(root, text="Red Sun", bg="red", fg="white").pack()
Label(root, text="Green Grass", bg="green", fg="black").pack()
Label(root, text="Blue Sky", bg="blue", fg="white").pack()

mainloop()
```

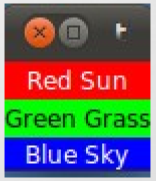


Pack() method

Fill:
If you want to make the widgets as wide as the parent widget, you have to use the fill=X option:

```
from Tkinter import *

root = Tk()
w = Label(root, text="Red Sun", bg="red", fg="white")
w.pack(fill=X)
w = Label(root, text="Green Grass", bg="green", fg="black")
w.pack(fill=X)
w = Label(root, text="Blue Sky", bg="blue", fg="white")
w.pack(fill=X)
mainloop()
```


Red Sun
Green Grass
Blue Sky

Pack() method

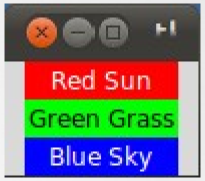
Padding:
The pack() manager knows four padding options, i.e. internal and external padding and padding in x and y direction:

padx	External padding, horizontally
pady	External padding, vertically
ipadx	Internal padding, horizontally.
ipady	Internal padding, vertically

Pack() method

padx   External padding, horizontally

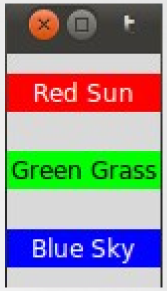
```
from tkinter import *
root = Tk()
w = Label(root, text="Red Sun", bg="red", fg="white")
w.pack(fill=X,padx=10)
w = Label(root, text="Green Grass", bg="green", fg="black")
w.pack(fill=X,padx=10)
w = Label(root, text="Blue Sky", bg="blue", fg="white")
w.pack(fill=X,padx=10)
mainloop()
```



Pack() method

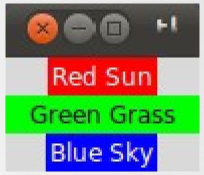
pady   External padding, vertically

```
from tkinter import *
root = Tk()
w = Label(root, text="Red Sun", bg="red", fg="white")
w.pack(fill=X,pady=10)
w = Label(root, text="Green Grass", bg="green", fg="black")
w.pack(fill=X,pady=10)
w = Label(root, text="Blue Sky", bg="blue", fg="white")
w.pack(fill=X,pady=10)
mainloop()
```



Pack() method

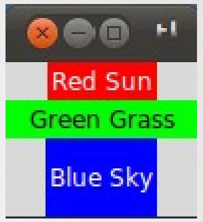
ipadx   Internal padding, horizontally.



```
from Tkinter import *
root = Tk()
w = Label(root, text="Red Sun", bg="red", fg="white")
w.pack()
w = Label(root, text="Green Grass", bg="green", fg="black")
w.pack(ipadx=10)
w = Label(root, text="Blue Sky", bg="blue", fg="white")
w.pack()
mainloop()
```

Pack() method

ipady Internal padding, vertically



```
from Tkinter import *
root = Tk()
w = Label(root, text="Red Sun", bg="red", fg="white")
w.pack()
w = Label(root, text="Green Grass", bg="green", fg="black")
w.pack(ipadx=10)
w = Label(root, text="Blue Sky", bg="blue", fg="white")
w.pack(ipady=10)
mainloop()
```

The default value in all cases is 0.

Pack() method

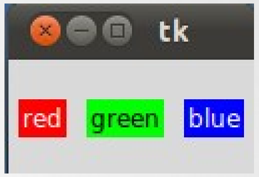
Placing widgets side by side

```
from Tkinter import *

root = Tk()

w = Label(root, text="red", bg="red", fg="white")
w.pack(padx=5, pady=10, side=LEFT)
w = Label(root, text="green", bg="green", fg="black")
w.pack(padx=5, pady=20, side=LEFT)
w = Label(root, text="blue", bg="blue", fg="white")
w.pack(padx=5, pady=20, side=LEFT)

mainloop()
```



Place() method

This geometry manager organizes widgets by placing them in a specific position in the parent widget.

Syntax:
widget.place(place_options)

Here is the list of possible options −
anchor − The exact spot of widget other options refer to: may be N, E, S, W, NE, NW, SE, or SW, compass directions indicating the corners and sides of widget; default is NW (the upper left corner of widget)

bordermode − INSIDE (the default) to indicate that other options refer to the parent's inside (ignoring the parent's border); OUTSIDE otherwise.

Place() method

height, width − Height and width in pixels.

relheight, relwidth − Height and width as a float between 0.0 and 1.0, as a fraction of the height and width of the parent widget.

relx, rely − Horizontal and vertical offset as a float between 0.0 and 1.0, as a fraction of the height and width of the parent widget.

x, y − Horizontal and vertical offset in pixels.

Place() method

Example

```
from tkinter import *
from tkinter import messagebox

top = Tk()

def helloCallBack():
   messagebox.showinfo("Say Hello", "Hello World")

B = Button(top, text ="Hello", command = helloCallBack)

B.place(x = 35,y = 50)
top.mainloop()
```

Grid manager

- The first geometry manager of Tk had been pack.
- The algorithmic behaviour of pack is not easy to understand and it can be difficult to change an existing design.
- Grid was introduced in 1996 as an alternative to pack.
- Though grid is easier to learn and to use and produces nicer layouts, lots of developers keep using pack.
- Grid is in many cases the best choice for general use.
- While pack is sometimes not sufficient for changing details in the layout, place gives you complete control of positioning each element, but this makes it a lot more complex than pack and grid.

Grid manager

- The Grid geometry manager places the widgets in a **2-dimensional table**, which consists of a number of rows and columns.
- The position of a widget is defined by a **row and a column number.**
- Widgets with the same column number and different row numbers will be above or below each other.
- Correspondingly, widgets with the same row number but different column numbers will be on the same "line" and will be beside of each other, i.e. to the left or the right.
- Using the grid manager means that you create a widget, and use the grid method to tell the manager in which row and column to place them.
- The size of the grid doesn't have to be defined, because the manager automatically determines the best dimensions for the widgets used.

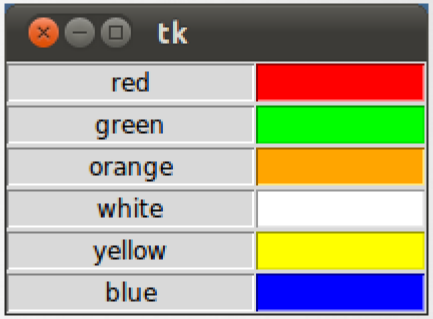
Grid manager

```
from Tkinter import *

colours = ['red','green','orange','white','yellow','blue']

r = 0
for c in colours:
    Label(text=c, relief=RIDGE,width=15).grid(row=r,column=0)
    Entry(bg=c, relief=SUNKEN,width=10).grid(row=r,column=1)
    r = r + 1

mainloop()
```



Tkinter Variable Classes

- Variables can be used with most entry widgets to track changes to the entered value.
- The Checkbutton and Radiobutton widgets require variables to work properly.
- Some widgets (like text entry widgets, radio buttons and so on) can be connected directly to application variables by using special options: variable, textvariable, onvalue, offvalue, and value.

- Create a Tkinter variable:
  - x = StringVar() # Holds a string; default value ""
  - x = IntVar() # Holds an integer; default value 0
  - x = DoubleVar() # Holds a float; default value 0.0
  - x = BooleanVar() # Holds a boolean, returns 0 for False and 1 for True

Example:

- var = StringVar()
- var.set("hello") **#to set the value, call the set method**

Tkinter Widgets

- Tkinter provides various controls, such as buttons, labels and text boxes used in a GUI application.
- These controls are commonly called **widgets**.
  - Label
  - Button
  - Canvas
  - combo-box
  - frame
  - level
  - check-button
  - entry
  - level-frame
  - menu
  - list – box
  - menu button
  - message
  - tk_optoinMenu
  - progress-bar
  - radio button
  - scroll bar
  - Separator
  - tree-view and many more...

Tkinter Widgets - Label

Label: A label is a widget that displays text or images, typically that the user will just view does not interact with. Labels are used for such things as identifying controls or other parts of the user interface, providing textual feedback or results, etc.

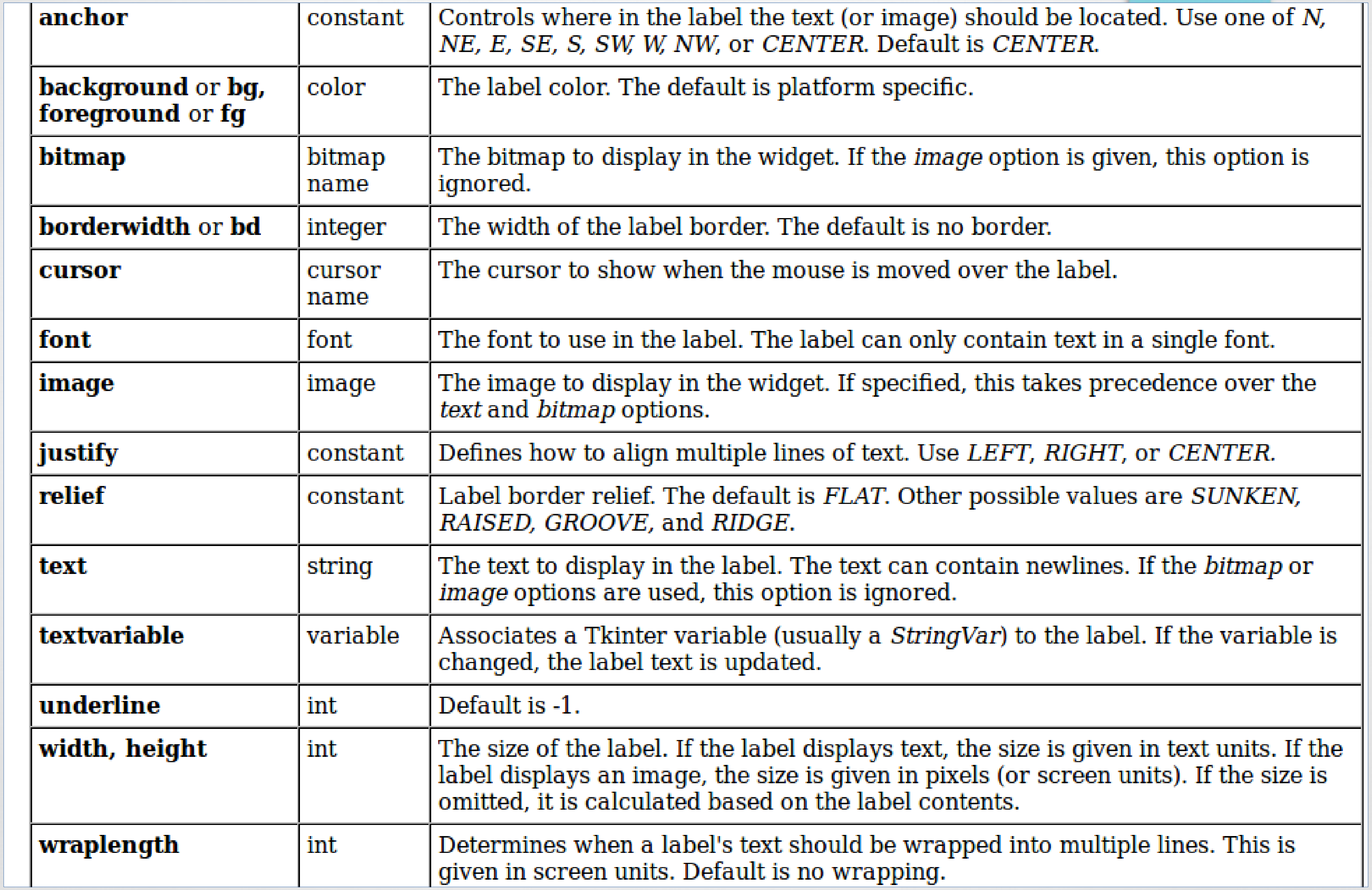
Syntax:

w = Label (master, option, ...)

master − represents the parent window.

options − list of commonly used options for this widget.

Tkinter Widgets - Label



anchor	constant	Controls where in the label the text (or image) should be located. Use one of *N, NE, E, SE, S, SW, W, NW*, or *CENTER*. Default is *CENTER*.
background or **bg**, **foreground** or **fg**	color	The label color. The default is platform specific.
bitmap	bitmap name	The bitmap to display in the widget. If the *image* option is given, this option is ignored.
borderwidth or **bd**	integer	The width of the label border. The default is no border.
cursor	cursor name	The cursor to show when the mouse is moved over the label.
font	font	The font to use in the label. The label can only contain text in a single font.
image	image	The image to display in the widget. If specified, this takes precedence over the *text* and *bitmap* options.
justify	constant	Defines how to align multiple lines of text. Use *LEFT*, *RIGHT*, or *CENTER*.
relief	constant	Label border relief. The default is *FLAT*. Other possible values are *SUNKEN, RAISED, GROOVE*, and *RIDGE*.
text	string	The text to display in the label. The text can contain newlines. If the *bitmap* or *image* options are used, this option is ignored.
textvariable	variable	Associates a Tkinter variable (usually a *StringVar*) to the label. If the variable is changed, the label text is updated.
underline	int	Default is -1.
width, height	int	The size of the label. If the label displays text, the size is given in text units. If the label displays an image, the size is given in pixels (or screen units). If the size is omitted, it is calculated based on the label contents.
wraplength	int	Determines when a label's text should be wrapped into multiple lines. This is given in screen units. Default is no wrapping.

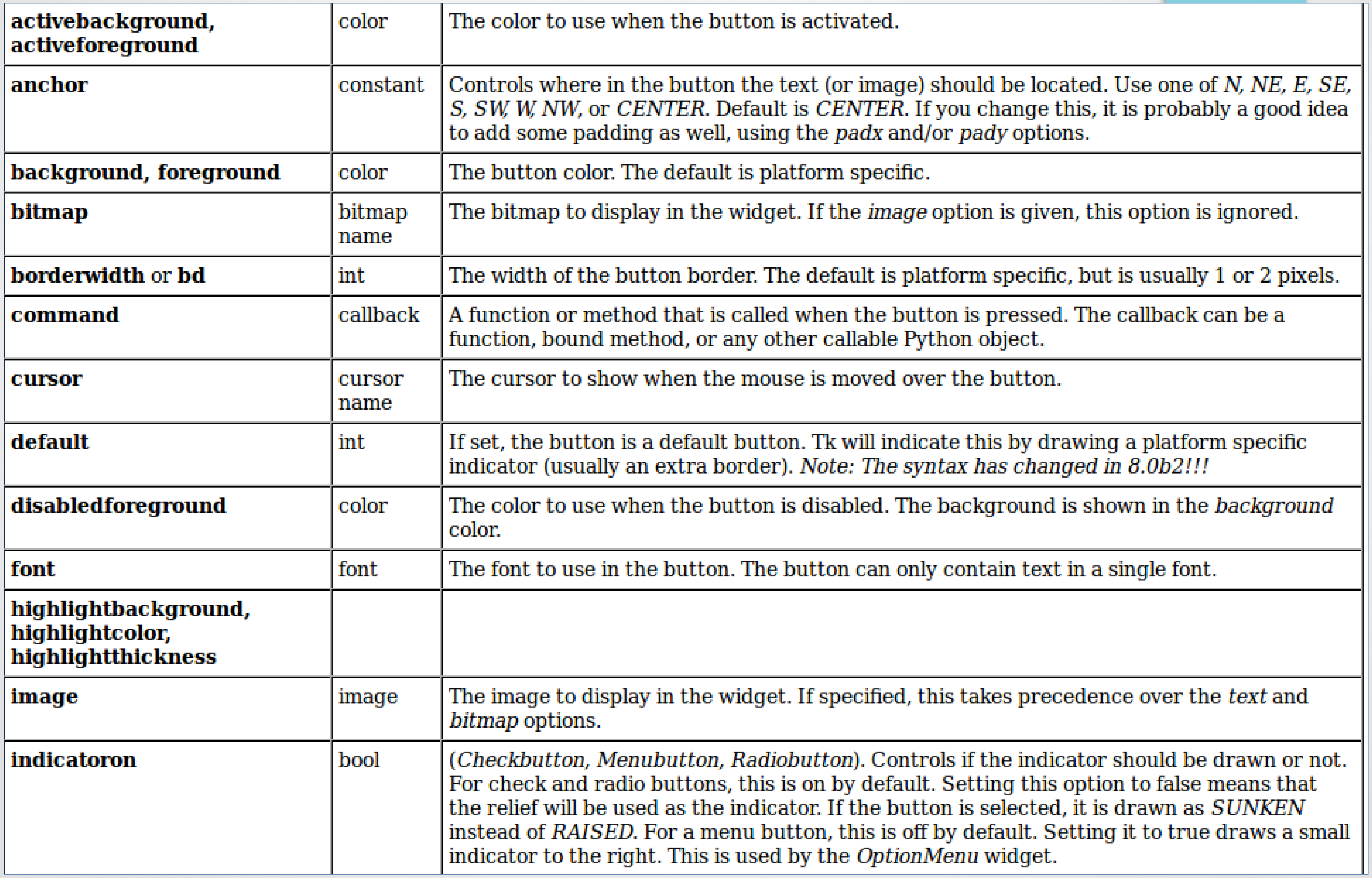
Tkinter Widgets - Button

Button: The Button widget is used to add buttons in a Python application. These buttons can display text or images that convey the purpose of the buttons

Syntax:

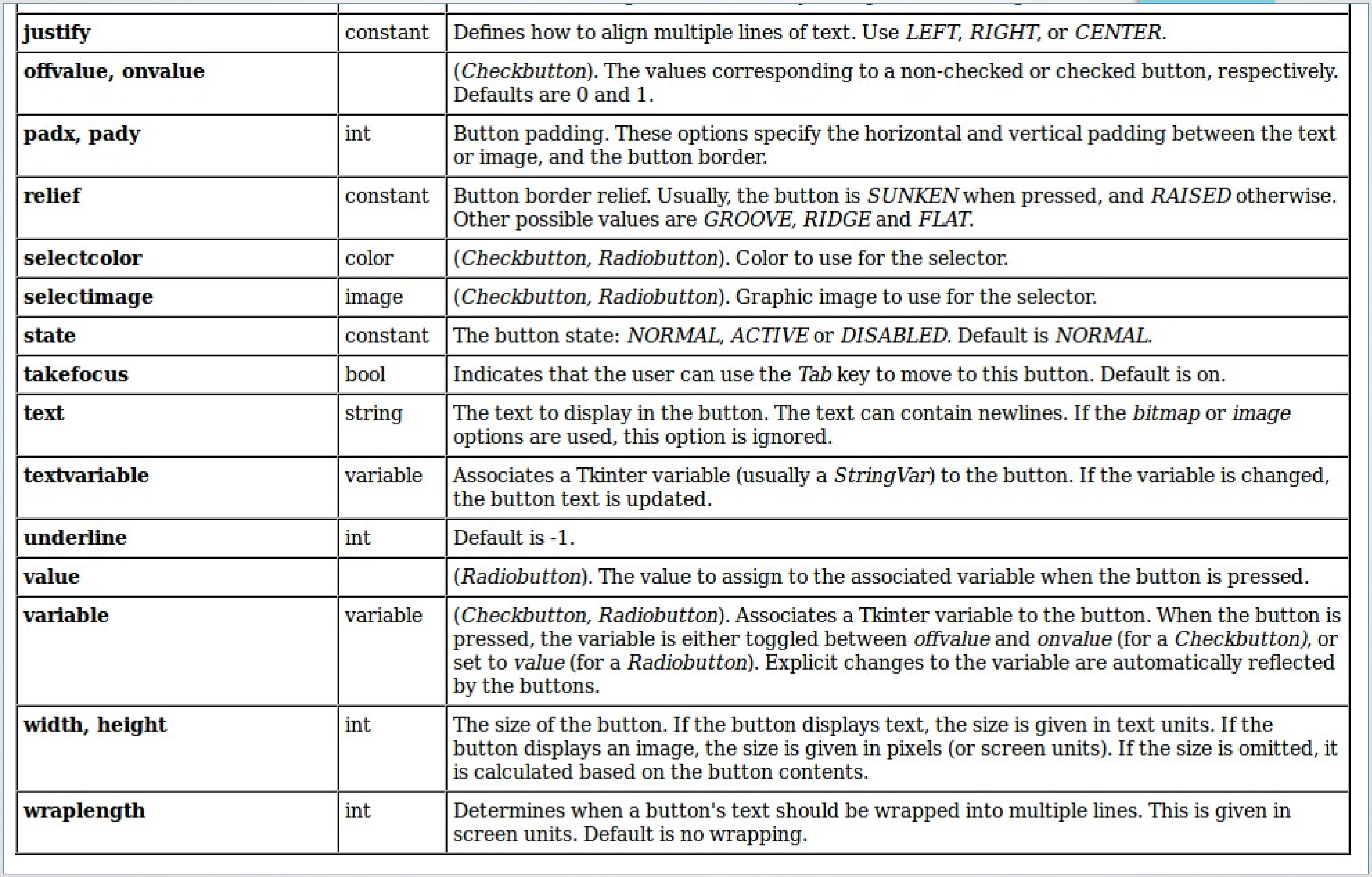
```
w = Button ( master, option=value, ... )
    master − represents the parent window.
    options − list of commonly used options for this widget.
```

Tkinter Widgets - Button



activebackground, activeforeground	color	The color to use when the button is activated.
anchor	constant	Controls where in the button the text (or image) should be located. Use one of *N, NE, E, SE, S, SW, W, NW*, or *CENTER*. Default is *CENTER*. If you change this, it is probably a good idea to add some padding as well, using the *padx* and/or *pady* options.
background, foreground	color	The button color. The default is platform specific.
bitmap	bitmap name	The bitmap to display in the widget. If the *image* option is given, this option is ignored.
borderwidth or **bd**	int	The width of the button border. The default is platform specific, but is usually 1 or 2 pixels.
command	callback	A function or method that is called when the button is pressed. The callback can be a function, bound method, or any other callable Python object.
cursor	cursor name	The cursor to show when the mouse is moved over the button.
default	int	If set, the button is a default button. Tk will indicate this by drawing a platform specific indicator (usually an extra border). *Note: The syntax has changed in 8.0b2!!!*
disabledforeground	color	The color to use when the button is disabled. The background is shown in the *background* color.
font	font	The font to use in the button. The button can only contain text in a single font.
highlightbackground, highlightcolor, highlightthickness		
image	image	The image to display in the widget. If specified, this takes precedence over the *text* and *bitmap* options.
indicatoron	bool	(*Checkbutton, Menubutton, Radiobutton*). Controls if the indicator should be drawn or not. For check and radio buttons, this is on by default. Setting this option to false means that the relief will be used as the indicator. If the button is selected, it is drawn as *SUNKEN* instead of *RAISED*. For a menu button, this is off by default. Setting it to true draws a small indicator to the right. This is used by the *OptionMenu* widget.

Tkinter Widgets - Button



justify	constant	Defines how to align multiple lines of text. Use *LEFT, RIGHT*, or *CENTER*.
offvalue, onvalue		(*Checkbutton*). The values corresponding to a non-checked or checked button, respectively. Defaults are 0 and 1.
padx, pady	int	Button padding. These options specify the horizontal and vertical padding between the text or image, and the button border.
relief	constant	Button border relief. Usually, the button is *SUNKEN* when pressed, and *RAISED* otherwise. Other possible values are *GROOVE, RIDGE* and *FLAT*.
selectcolor	color	(*Checkbutton, Radiobutton*). Color to use for the selector.
selectimage	image	(*Checkbutton, Radiobutton*). Graphic image to use for the selector.
state	constant	The button state: *NORMAL, ACTIVE* or *DISABLED*. Default is *NORMAL*.
takefocus	bool	Indicates that the user can use the *Tab* key to move to this button. Default is on.
text	string	The text to display in the button. The text can contain newlines. If the *bitmap* or *image* options are used, this option is ignored.
textvariable	variable	Associates a Tkinter variable (usually a *StringVar*) to the button. If the variable is changed, the button text is updated.
underline	int	Default is -1.
value		(*Radiobutton*). The value to assign to the associated variable when the button is pressed.
variable	variable	(*Checkbutton, Radiobutton*). Associates a Tkinter variable to the button. When the button is pressed, the variable is either toggled between *offvalue* and *onvalue* (for a *Checkbutton)*, or set to *value* (for a *Radiobutton*). Explicit changes to the variable are automatically reflected by the buttons.
width, height	int	The size of the button. If the button displays text, the size is given in text units. If the button displays an image, the size is given in pixels (or screen units). If the size is omitted, it is calculated based on the button contents.
wraplength	int	Determines when a button's text should be wrapped into multiple lines. This is given in screen units. Default is no wrapping.

Tkinter Widgets - CheckButton

Checkbutton:
The Checkbutton widget is used to display a number of options to a user as toggle buttons.
The user can then select one or more options by clicking the button corresponding to each option.

- Checkboxes are shown on the screen as square boxes.
- A Checkbox has two states: on or off.

Syntax:
w = Checkbutton (master, option, ...)

   master − represents the parent window.

   options − list of commonly used options for this widget.

Tkinter Widgets – Radio Button

Radiobutton:

A radio button, sometimes called option button, is a graphical user interface element of Tkinter, which allows the user to choose (exactly) one of a predefined set of options.

Syntax:

```
w = Radiobutton ( master, option, ...  )
```

master − represents the parent window.

options − list of commonly used options for this widget.

Tkinter Widgets – Entry

Entry:
The Entry widget is used to accept single-line text strings from a user. Entry widgets are the basic widgets of Tkinter used to get input, i.e. text strings, from the user of an application.

Syntax: w = Entry(master, option, ...)

master − represents the parent window.

options − list of commonly used options for this widget.

Tkinter Widgets – Text

Text: A text widget is used for multi-line text area.
The Text widget provides formatted text display. It allows you to display and edit text with various styles and attributes. It allows user to edit a multiline text and format the way it has to be displayed, such as changing its color and font.

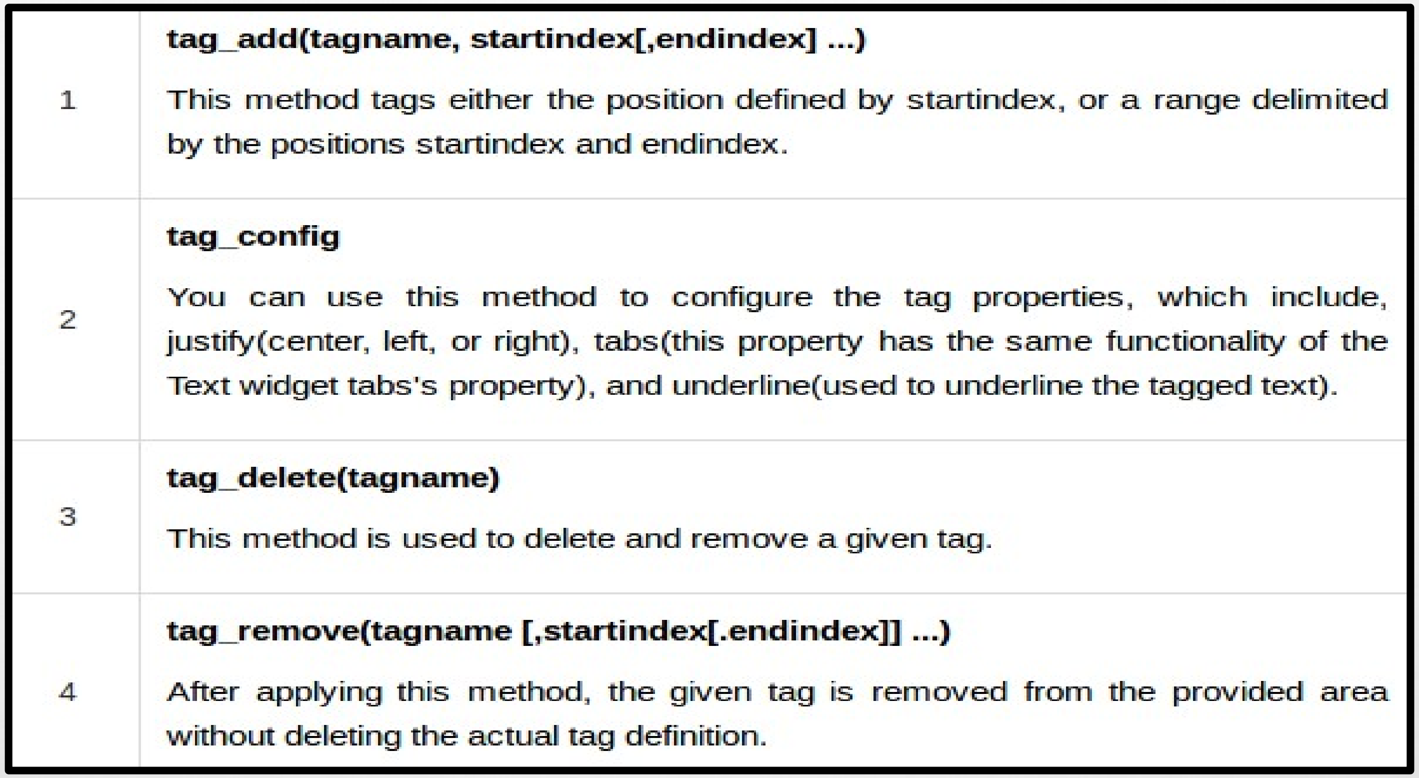
Syntax: w = Text(master, option, ...)

master − represents the parent window.

options − list of commonly used options for this widget.

Tkinter Widgets – Entry

Text: A text widget is used for multi-line text area.
Tags are used to associate names to regions of text which makes easy the task of modifying the display settings of specific text areas.



1	**tag_add(tagname, startindex[,endindex] ...)** This method tags either the position defined by startindex, or a range delimited by the positions startindex and endindex.
2	**tag_config** You can use this method to configure the tag properties, which include, justify(center, left, or right), tabs(this property has the same functionality of the Text widget tabs's property), and underline(used to underline the tagged text).
3	**tag_delete(tagname)** This method is used to delete and remove a given tag.
4	**tag_remove(tagname [,startindex[.endindex]] ...)** After applying this method, the given tag is removed from the provided area without deleting the actual tag definition.

Python GUI Programming - Tkinter

Tkinter Widgets

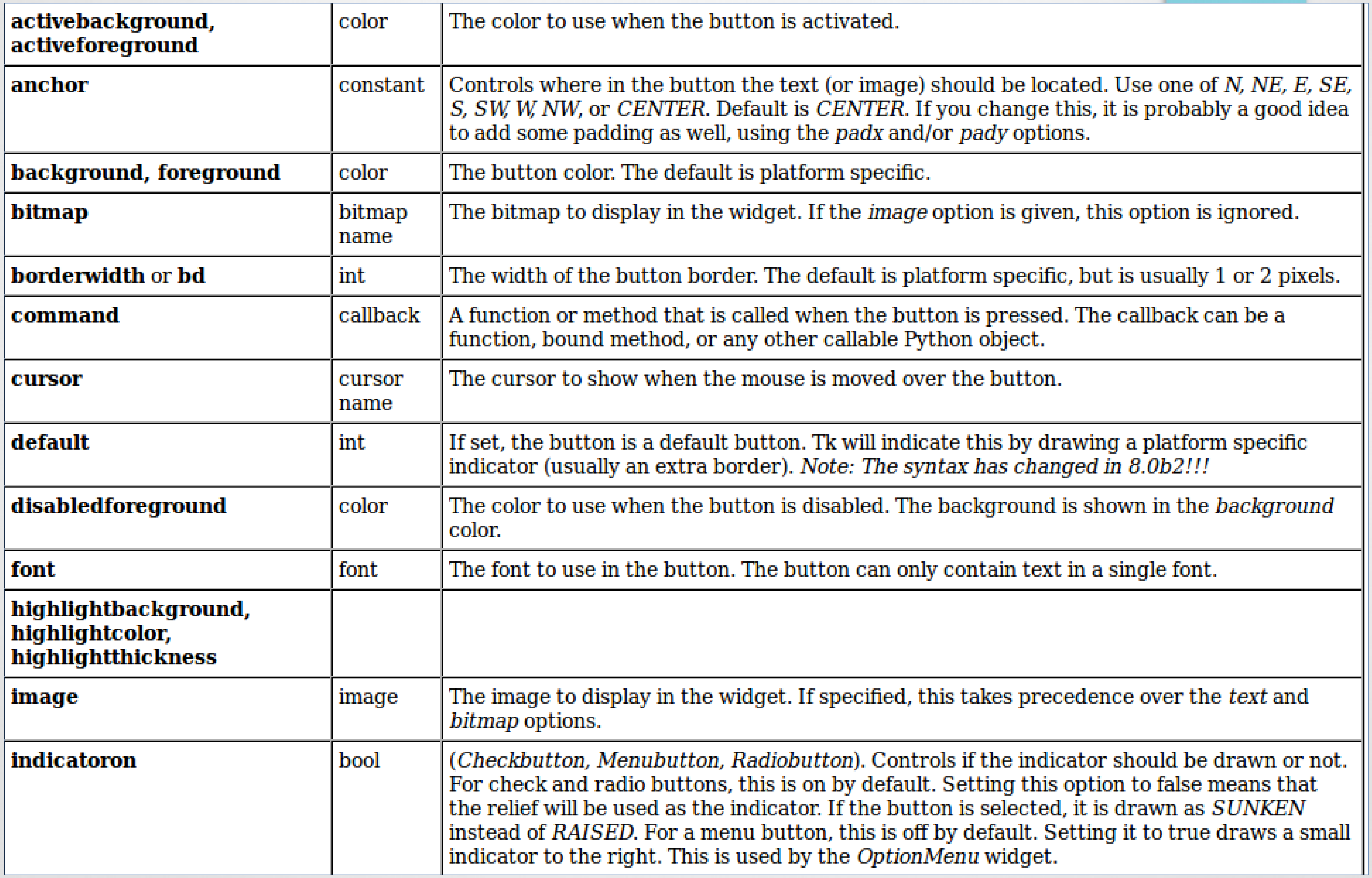
Button: The Button widget is used to add buttons in a Python application. These buttons can display text or images that convey the purpose of the buttons

Syntax: w = Button (master, option=value, ...)

master − represents the parent window.

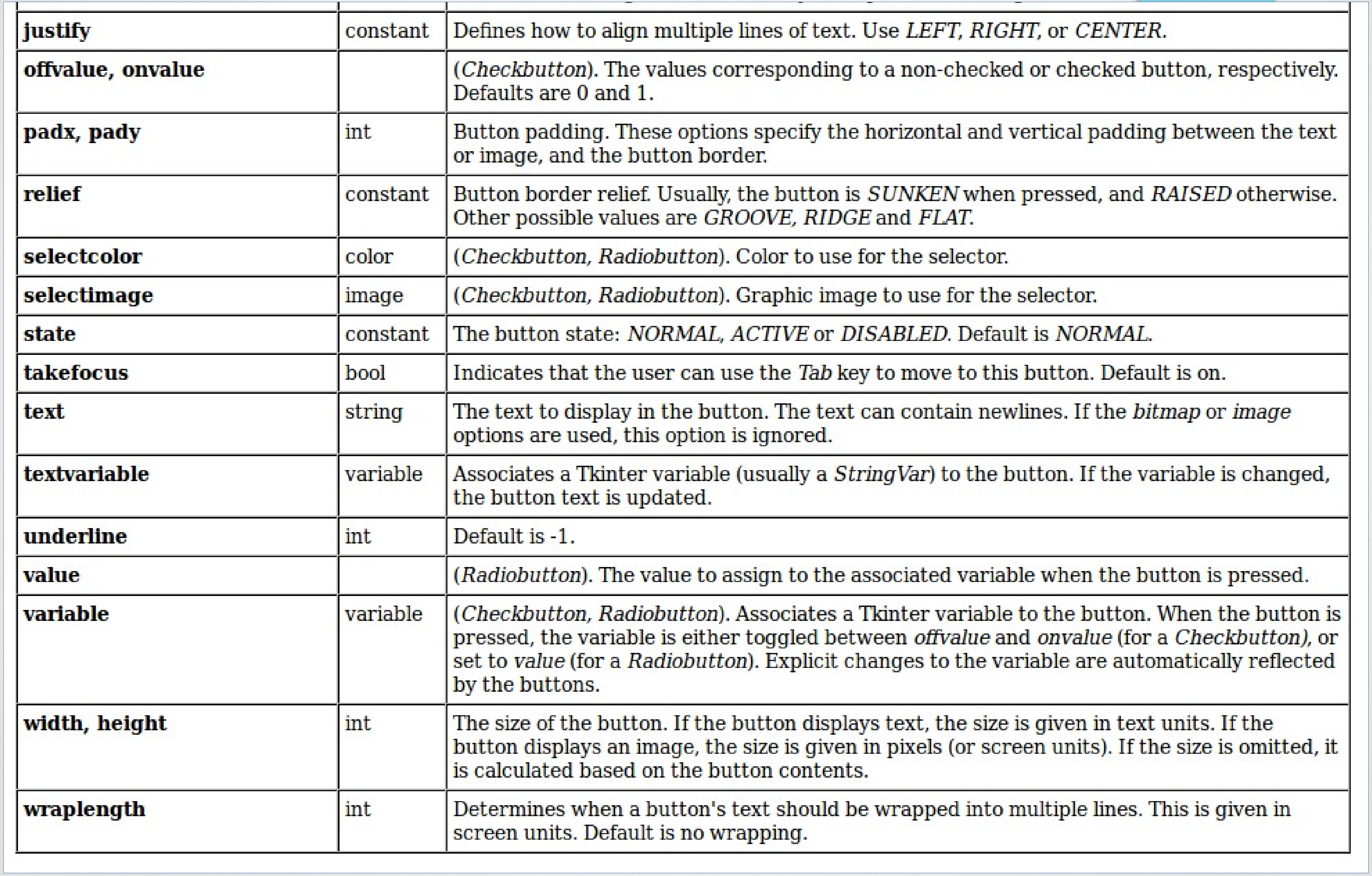
options − list of commonly used options for this widget.

Python GUI Programming - Tkinter



activebackground, activeforeground	color	The color to use when the button is activated.
anchor	constant	Controls where in the button the text (or image) should be located. Use one of *N, NE, E, SE, S, SW, W, NW*, or *CENTER*. Default is *CENTER*. If you change this, it is probably a good idea to add some padding as well, using the *padx* and/or *pady* options.
background, foreground	color	The button color. The default is platform specific.
bitmap	bitmap name	The bitmap to display in the widget. If the *image* option is given, this option is ignored.
borderwidth or **bd**	int	The width of the button border. The default is platform specific, but is usually 1 or 2 pixels.
command	callback	A function or method that is called when the button is pressed. The callback can be a function, bound method, or any other callable Python object.
cursor	cursor name	The cursor to show when the mouse is moved over the button.
default	int	If set, the button is a default button. Tk will indicate this by drawing a platform specific indicator (usually an extra border). *Note: The syntax has changed in 8.0b2!!!*
disabledforeground	color	The color to use when the button is disabled. The background is shown in the *background* color.
font	font	The font to use in the button. The button can only contain text in a single font.
highlightbackground, highlightcolor, highlightthickness		
image	image	The image to display in the widget. If specified, this takes precedence over the *text* and *bitmap* options.
indicatoron	bool	(*Checkbutton, Menubutton, Radiobutton*). Controls if the indicator should be drawn or not. For check and radio buttons, this is on by default. Setting this option to false means that the relief will be used as the indicator. If the button is selected, it is drawn as *SUNKEN* instead of *RAISED*. For a menu button, this is off by default. Setting it to true draws a small indicator to the right. This is used by the *OptionMenu* widget.

Python GUI Programming - Tkinter



justify	constant	Defines how to align multiple lines of text. Use *LEFT, RIGHT,* or *CENTER*.
offvalue, onvalue		(*Checkbutton*). The values corresponding to a non-checked or checked button, respectively. Defaults are 0 and 1.
padx, pady	int	Button padding. These options specify the horizontal and vertical padding between the text or image, and the button border.
relief	constant	Button border relief. Usually, the button is *SUNKEN* when pressed, and *RAISED* otherwise. Other possible values are *GROOVE, RIDGE* and *FLAT*.
selectcolor	color	(*Checkbutton, Radiobutton*). Color to use for the selector.
selectimage	image	(*Checkbutton, Radiobutton*). Graphic image to use for the selector.
state	constant	The button state: *NORMAL, ACTIVE* or *DISABLED*. Default is *NORMAL*.
takefocus	bool	Indicates that the user can use the *Tab* key to move to this button. Default is on.
text	string	The text to display in the button. The text can contain newlines. If the *bitmap* or *image* options are used, this option is ignored.
textvariable	variable	Associates a Tkinter variable (usually a *StringVar*) to the button. If the variable is changed, the button text is updated.
underline	int	Default is -1.
value		(*Radiobutton*). The value to assign to the associated variable when the button is pressed.
variable	variable	(*Checkbutton, Radiobutton*). Associates a Tkinter variable to the button. When the button is pressed, the variable is either toggled between *offvalue* and *onvalue* (for a *Checkbutton*), or set to *value* (for a *Radiobutton*). Explicit changes to the variable are automatically reflected by the buttons.
width, height	int	The size of the button. If the button displays text, the size is given in text units. If the button displays an image, the size is given in pixels (or screen units). If the size is omitted, it is calculated based on the button contents.
wraplength	int	Determines when a button's text should be wrapped into multiple lines. This is given in screen units. Default is no wrapping.

Python GUI Programming - Tkinter

Tkinter Widgets

Checkbutton: The Checkbutton widget is used to display a number of options to a user as toggle buttons. The user can then select one or more options by clicking the button corresponding to each option.

- Checkboxes are shown on the screen as square boxes.
- A Checkbox has two states: on or off.

Syntax: w = Checkbutton (master, option, ...)

master − represents the parent window.

options − list of commonly used options for this widget.

Python GUI Programming - Tkinter

Tkinter Widgets

Radiobutton: A radio button, sometimes called option button, is a graphical user interface element of Tkinter, which allows the user to choose (exactly) one of a predefined set of options.

Syntax: w = Radiobutton (master, option, ...)

master − represents the parent window.

options − list of commonly used options for this widget.

Python GUI Programming - Tkinter

Tkinter Widgets

Entry: The Entry widget is used to accept single-line text strings from a user. Entry widgets are the basic widgets of Tkinter used to get input, i.e. text strings, from the user of an application.

Syntax: w = Entry(master, option, ...)

master − represents the parent window.

options − list of commonly used options for this widget.

Python GUI Programming - Tkinter

Tkinter Widgets

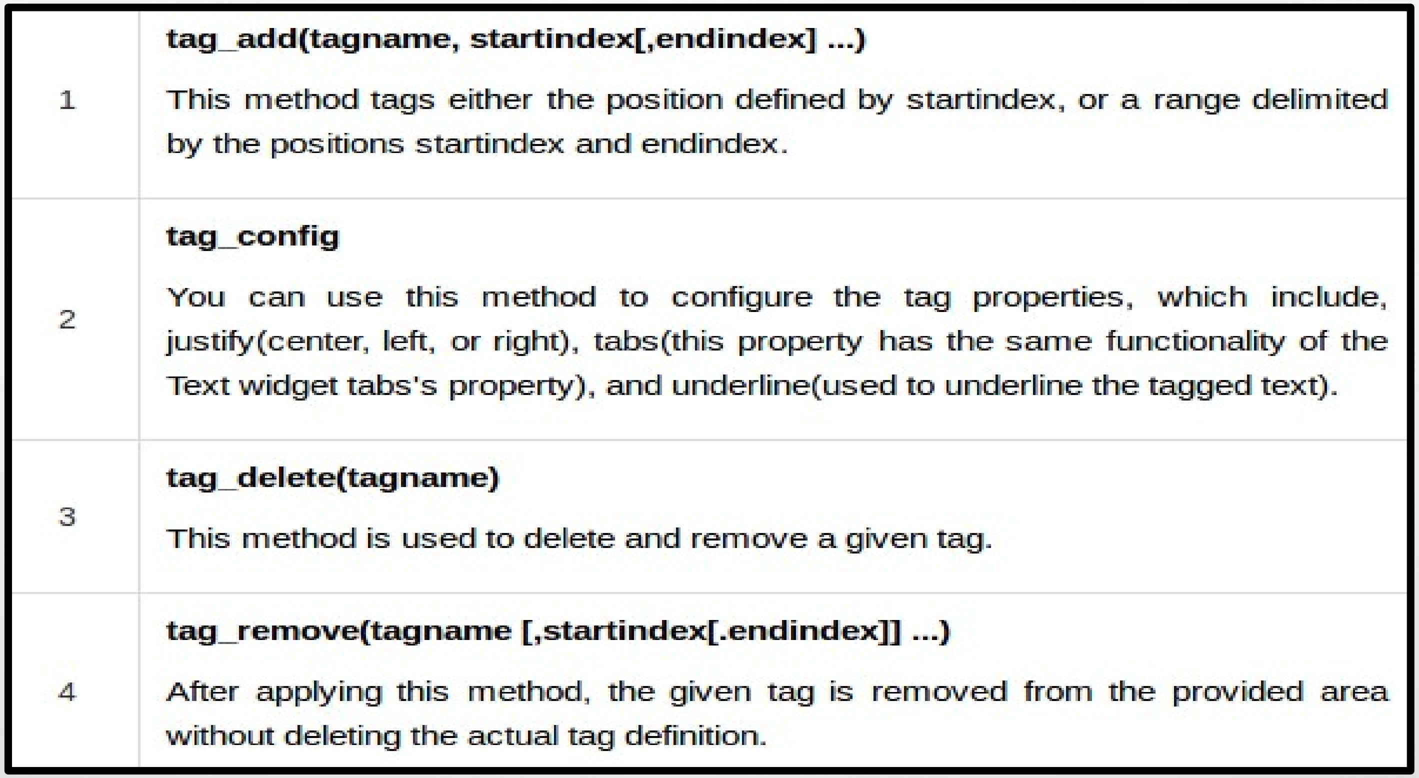
Text: A text widget is used for multi-line text area.
The Text widget provides formatted text display. It allows you to display and edit text with various styles and attributes. It allows user to edit a multiline text and format the way it has to be displayed, such as changing its color and font.

Syntax: w = Text(master, option, ...)
master − represents the parent window.
options − list of commonly used options for this widget.

Python GUI Programming - Tkinter

Tkinter Widgets

Text: A text widget is used for multi-line text area.
Tags are used to associate names to regions of text which makes easy the task of modifying the display settings of specific text areas.



1	**tag_add(tagname, startindex[,endindex] ...)** This method tags either the position defined by startindex, or a range delimited by the positions startindex and endindex.
2	**tag_config** You can use this method to configure the tag properties, which include, justify(center, left, or right), tabs(this property has the same functionality of the Text widget tabs's property), and underline(used to underline the tagged text).
3	**tag_delete(tagname)** This method is used to delete and remove a given tag.
4	**tag_remove(tagname [,startindex[.endindex]] ...)** After applying this method, the given tag is removed from the provided area without deleting the actual tag definition.

Python GUI Programming - Tkinter

Tkinter Widgets

Text: A text widget is used for multi-line text area.
Tags are used to associate names to regions of text which makes easy the task of modifying the display settings of specific text areas.

Layout Manager (Geometry Manager) in Python

Tkinter has three built-in layout managers that use geometric methods to position widgets in an application frame:

- *pack()* organizes widgets in horizontal and vertical boxes that are limited to left, right, top, bottom positions. Each box is offset and relative to each other.
- *place()* places widgets in a two dimensional grid using x and y absolute coordinates.
- *grid()* locates widgets in a two dimensional grid using row and column absolute coordinates.

pack()

The pack method is used to organize widget in to cavity on parent window.

Syntax

```
w.pack(options)
```

pack is the easiest layout manager to use with Tkinter. Instead of declaring the precise location of a widget, *pack()* declares the positioning of widgets in relation to each other. However, *pack*() is limited in precision compared to *place()* and *grid()* which feature absolute positioning. For simple positioning of widgets vertically or horizontally in relation to each other, *pack()* is the layout manager of choice.

They are used for positioning,arranging and registering widgets on tkinter window.

There are three possible options of pack method : expand, fill and side

- expand : It can have boolean values , True and False. If value is set to True then widget expands to fill any space.If value is set to False then it is used in widget's parent
- fill : It is used to fill extra spaces allocated to it by pack(). This fill option has dimensions like – NONE (default), X(fill horizontally), Y(fill vertically) or BOTH(fill horizontally and vertically)
- side : It is used to specify the side to which widget must be packed in parent window – TOP(default), BOTTOM,LEFT or RIGHT

Pack method with internal and external padding

- padx : This option is used for external padding in X direction / horizontal padding.
- pady : This option is used for external padding in Y direction / vertical padding.
- ipadx : This option is used for internal padding in X direction / horizontal padding.
- ipady : This option is used for internal padding in Y direction / vertical padding.

grid()

The grid method is used to organize widget in to grid / table like structure.
Syntax

```
w.grid(options)
```

There are various possible options of grid method : column, columnspan, row, rowspan, padx, pady, ipadx, ipady and sticky

- Column : It is the column number where you want to organize widget.The default value is zero.
- Row : It is the row number where you wnat to organize the widget. the default value is zero.
- Sticky : It is used in the condition where cell is larger than widget.If sticky is not used than by default the widget is placed in the center of the cell. Sticky can have values like N,E,S,W,NE,NW,SE & SW.
  i. If you set sticky option to NE, the position of widget is top right.
  ii. If you set sticky option to SE, the position of widget is bottom right.
  iii. If you set sticky option to NW, the position of widget is top left.
  iv. If you set sticky option to SW, the position of widget is bottom left.

place()

The place method is used to organize widget in to parent window placing them in specific position. In absolute positioning we specify the position and size of each widget in pixels. The size and position of widget do not change if we resize the window
Syntax

```
w.place(options)
```

There are various possible options of grid method : anchor, bordermode, height, width, relheight, relwidth, relx, rely, x, y

- anchor : It is used in the condition where cell is larger than widget.If anchor is not used than by default the widget is placed in the upper left corner ie NW. Anchor can have values like N,E,S,W,NE,NW,SE & SW.
- bordermode : INSIDE (the default) to indicate that other options refer to the parent’s inside (ignoring the parent’s border); OUTSIDE otherwise.
- height and width : It is used to specify height and width in pixels.
- x and y : It is used to determine horizontal and vertical offset in pixels.

Python Tkinter Scrollbar

Introduction to the Tkinter scrollbar widget

A scrollbar allows you to view all parts of another widget whose content is typically larger than available space.

Tkinter scrollbar widget is not a part of any other widgets such as Text and Listbox. Instead, a scrollbar an independent widget.

To use the scrollbar widget, you need to:

- First, create a scrollbar widget.
- Second, link the scrollbar with a scrollable widget.

```
scrollbar = tk.Scrollbar(container,orient='vertical',command=widget.yview)
)
```

In this syntax:

- The `container` is the window or frame on which the scrollbar locates.
- The `orient` argument specifies whether the scrollbar needs to scroll horizontally or vertically.
- The `command` argument allows the scrollbar widget to communicate with the scrollable widget.

The scrollable widget also needs to communicate back to the scrollbar about the percentage of the entire content area that is currently visible.

Every scrollable widget has a `yscrollcommand` and/or `xscrollcommand` options. And you can assign the `scrollbar.set` method to it:

```
widget['yscrollcommand'] = scrollbar.set
```

Tkinter canvas

The canvas is a general purpose widget: you can use it to make any kind of graphics including plots, drawings, charts, show images and much more.
A tkinter canvas can be used to draw in a window. Use this widget to draw graphs or plots. You can even use it to create graphical editors.
A canvas widget manages a 2D collection of graphical objects — lines, circles, images, or other widgets. It is suitable for drawing or building more complex widgets.
A canvas is added with one line of code:

```
myCanvas = Canvas(root, bg="white", height=300, width=300)
```

Tkinter geometric shapes

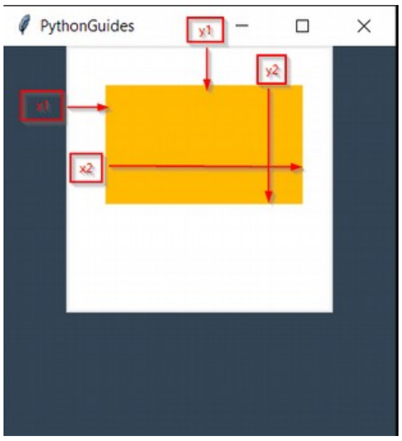
We can draw various shapes on the `Canvas`.

Python Tkinter Canvas Rectangle

- **Python Tkinter Canvas** has built-in features to create shapes.
- To create a rectangle create_rectangle() method is used.
- This method accepts 4 parameters x1, y1, x2, y2. Here x1 and y1 are the coordinates for the top left corner and x2 and y2 are the coordinates for the bottom right corner.
- Syntax:

```
create_rectangle(x1, y1, x2, y2, option, ...)
```

```
canvas.create_rectangle(30, 30, 180, 120,outline="#fb0",fill="#fb0")
```



x1 pushes the object to the right side or in the East., y1 pushes the object towards the South direction. X2 and Y2 expand the rectangle in East & South directions.

Python Tkinter Canvas Text

- The text refers to the arrangement of the alphabet(s).
- We can place text in python Tkinter canvas using `canvas.create_text(x, y)`. Here x & y is the position of the tex

```
canvas.create_text(200,100,fill="darkblue",font="Times 20 italic
bold",text="with great power comes \ngreat responsibility")
```

A text widget is used for multi-line text area. The tkinter text widget is very powerful and flexible and can be used for a wide range of tasks. Though one of the main purposes is to provide simple multi-line areas, as they are often used in forms, text widgets can also be used as simple text editors or even web browsers.

Python Tkinter Canvas Oval

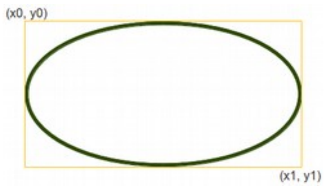
Ovals, mathematically, are ellipses, including circles as a special case.

The ellipse is fit into a rectangle defined by the coordinates (`x0`, `y0`) of the top left corner and the coordinates (`x1`, `y1`) of a point just *outside of* the bottom right corner.

```
We can create an oval on a canvas c with the following method:
```

```
create_oval ( x0, y0, x1, y1, option, ... )


create_oval(50,50,100,100)
```



ListBox widget

The ListBox widget is used to display different types of items. These items must be of the same type of font and having the same font color. The items must also be of Text type. The user can select one or more items from the given list according to the requirement.

```
w = Listbox ( master, option, ... )
```

- **master** − This represents the parent window.
- **options** − Here is the list of most commonly used options for this widget. These options can be used as key-value pairs separated by commas.

Options:

bg

The normal background color displayed behind the label and indicator.

bd

The size of the border around the indicator. Default is 2 pixels.

selectmode

Determines how many items can be selected, and how mouse drags affect the selection −

- **BROWSE** − Normally, you can only select one line out of a listbox. If you click on an item and then drag to a different line, the selection will follow the mouse. This is the default.
- **SINGLE** − You can only select one line, and you can't drag the mouse.wherever you click button 1, that line is selected.

- **MULTIPLE** − You can select any number of lines at once. Clicking on any line toggles whether or not it is selected.
- **EXTENDED** − You can select any adjacent group of lines at once by clicking on the first line and dragging to the last line.

xscrollcommand

If you want to allow the user to scroll the listbox horizontally, you can link your listbox widget to a horizontal scrollbar.

yscrollcommand

If you want to allow the user to scroll the listbox vertically, you can link your listbox widget to a vertical scrollbar.

Methods

activate (index)

Selects the line specifies by the given index.

curselection()

Returns a tuple containing the line numbers of the selected element or elements, counting from 0. If nothing is selected, returns an empty tuple.

get (first, last=None)

Returns a tuple containing the text of the lines with indices from first to last, inclusive. If the second argument is omitted, returns the text of the line closest to first.

index (i)

If possible, positions the visible part of the listbox so that the line containing index i is at the top of the widget.

insert (index, *elements)

Insert one or more new lines into the listbox before the line specified by index. Use END as the first argument if you want to add new lines to the end of the listbox.

xview()

To make the listbox horizontally scrollable, set the command option of the associated horizontal scrollbar to this method.

yview()

To make the listbox vertically scrollable, set the command option of the associated vertical scrollbar to this method

Python - Tkinter Text

Text widgets provide advanced capabilities that allow you to edit a multiline text and format the way it has to be displayed, such as changing its color and font.

You can also use elegant structures like tabs and marks to locate specific sections of the text, and apply changes to those areas. Moreover, you can embed windows and images in the text because this widget was designed to handle both plain and formatted text.

```
w = Text ( master, option, ... )
```

- **master** − This represents the parent window.

- **options** − Here is the list of most commonly used options for this widget. These options can be used as key-value pairs separated by commas.

Option

bg

The default background color of the text widget.

bd

The width of the border around the text widget. Default is 2 pixels

font

The default font for text inserted into the widget

xscrollcommand

To make the text widget horizontally scrollable, set this option to the set() method of the horizontal scrollbar.

yscrollcommand

To make the text widget vertically scrollable, set this option to the set() method of the vertical scrollbar.

Methods

delete(startindex [,endindex])

This method deletes a specific character or a range of text.

get(startindex [,endindex])

This method returns a specific character or a range of text.

index(index)

Returns the absolute value of an index based on the given index.

insert(index [,string]...)

This method inserts strings at the specified index location.