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

* **Tkinter** − Tkinter is the Python interface to the Tk GUI toolkit shipped with Python. We would look this option in this chapter.
* **wxPython** − This is an open-source Python interface for wxWindows [http://wxpython.org](http://wxpython.org/).
* **JPython** − JPython is a Python port for Java which gives Python scripts seamless access to Java class libraries on the local machine [http://www.jython.org](http://www.jython.org/).

**What is GUI?**

* A *graphical user interface* (GUI) is a *human-computer interface* (i.e., a way for humans to interact with computers) that uses [*windows*](http://www.linfo.org/window.html), [*icons*](http://www.linfo.org/icon.html) and *menus* and which can be manipulated by a mouse (and often to a limited extent by a keyboard as well).
* GUIs stand in sharp contrast to *command line interfaces* (CLIs), which use only text and are accessed solely by a keyboard. The most familiar example of a CLI to many people is [MS-DOS](http://www.linfo.org/ms-dos.html). Another example is [Linux](http://www.linfo.org/linuxdef.html) when it is used in [*console*](http://www.linfo.org/console.html) mode (i.e., the entire screen shows text only).
* A window is a (usually) rectangular portion of the monitor screen that can display its contents (e.g., a [program](http://www.linfo.org/program.html), icons, a text file or an image) seemingly independently of the rest of the display screen. A major feature is the ability for multiple windows to be open simultaneously. Each window can display a different application, or each can display different files (e.g., text, image or spreadsheet files) that have been opened or created with a single application.
* An icon is a small picture or symbol in a GUI that represents a program (or [command](http://www.linfo.org/command.html)), a file, a directory or a device (such as a hard disk or floppy). Icons are used both on the *desktop* and within application programs. Examples include small rectangles (to represent files), file folders (to represent directories), a trash can (to indicate a place to dispose of unwanted files and directories) and buttons on web browsers (for navigating to previous pages, for reloading the current page, etc.).
* Commands are issued in the GUI by using a mouse, trackball or touchpad to first move a pointer on the screen to, or on top of, the icon, menu item or window of interest in order to *select* that object. Then, for example, icons and windows can be moved by *dragging* (moving the mouse with the held down) and objects or programs can be opened by clicking on their icons.

**Advantages of GUIs**

* A major advantage of GUIs is that they make computer operation more intuitive, and thus easier to learn and use. For example, it is much easier for a new user to move a file from one directory to another by dragging its icon with the mouse than by having to remember and type seemingly arcane commands to accomplish the same task.
* Adding to this intuitiveness of operation is the fact that GUIs generally provide users with immediate, visual feedback about the effect of each action. For example, when a user deletes an icon representing a file, the icon immediately disappears, confirming that the file has been deleted (or at least sent to the trash can). This contrasts with the situation for a CLI, in which the user types a delete command (inclusive of the name of the file to be deleted) but receives no automatic feedback indicating that the file has actually been removed.
* In addition, GUIs allow users to take full advantage of the powerful [*multitasking*](http://www.linfo.org/multitasking.html) (the ability for multiple programs and/or multiple instances of single programs to run simultaneously) capabilities of modern [operating systems](http://www.linfo.org/operating_systems_list.html) by allowing such multiple programs and/or instances to be displayed simultaneously. The result is a large increase in the flexibility of computer use and a consequent rise in user productivity.
* But the GUI has became much more than a mere convenience. It has also become the standard in human-computer interaction, and it has influenced the work of a generation of computer users. Moreover, it has led to the development of new types of applications and entire new industries. An example is desktop publishing, which has revolutionized (and partly wiped out) the traditional printing and typesetting industry.
* Despite the great convenience of the GUI however, system administrators and other advanced users tend to prefer the CLI for many operations because it is frequently more convenient and generally more powerful1. On [Unix-like](http://www.linfo.org/unix-like.html) operating systems, GUIs are actually just attractive, convenient coverings for [*command line programs*](http://www.linfo.org/command_line_program.html) (i.e., programs which operate from a CLI), and they rely on them for their operation.
* One of the great attractions of Unix-like operating systems is that they have maintained their CLI capabilities while continuing to improve their GUIs, thereby allowing advanced users to harness the full power of the computer while simultaneously making it easier for beginning and intermediate users. In contrast, the newer versions of Microsoft Windows (such as 2000 and XP) have downgraded their CLIs to marginal roles.

**Intro to GUI Libraries in Python**

Python has many different GUI frameworks that are built for either cross-platform compatibility or platform-specific. The Tkinter framework is traditionally bundled with Python which uses the Tk. A full list of available frameworks can be seen here: [GUI Programming in Python](http://wiki.python.org/moin/GuiProgramming). We will be using the Tkinter framework. A full tutorial on how to use the different widgets can be found here: [An Introduction to Tkinter](http://www.pythonware.com/library/tkinter/introduction/index.htm)

Tkinter is the standard Python interface (Tk interface) for the Tk GUI toolkit. The Tk interface module consists of a number of other modules as well. We will cover some but not all of them. In a command line based interface the command line determines the order of execution. The user is directed from one prompt to the next and the order is strictly enforced. With a GUI based interface the user influences the order of events more strongly, therefore GUI interfaces are event driven.

import Tkinter

## Tkinter widgets

Tkinter provides various controls, such as buttons, labels and text boxes used in a GUI application. These controls are commonly called widgets.

There are currently 15 types of widgets in Tkinter. We present these widgets as well as a brief description in the following table −

|  |  |
| --- | --- |
| **Sr.No.** | **Operator & Description** |
| 1 | [**Button**](https://www.tutorialspoint.com/python/tk_button.htm)  The Button widget is used to display buttons in your application. |
| 2 | [**Canvas**](https://www.tutorialspoint.com/python/tk_canvas.htm)  The Canvas widget is used to draw shapes, such as lines, ovals, polygons and rectangles, in your application. |
| 3 | [**Checkbutton**](https://www.tutorialspoint.com/python/tk_checkbutton.htm)  The Checkbutton widget is used to display a number of options as checkboxes. The user can select multiple options at a time. |
| 4 | [**Entry**](https://www.tutorialspoint.com/python/tk_entry.htm)  The Entry widget is used to display a single-line text field for accepting values from a user. |
| 5 | [**Frame**](https://www.tutorialspoint.com/python/tk_frame.htm)  The Frame widget is used as a container widget to organize other widgets. |
| 6 | [**Label**](https://www.tutorialspoint.com/python/tk_label.htm)  The Label widget is used to provide a single-line caption for other widgets. It can also contain images. |
| 7 | [**Listbox**](https://www.tutorialspoint.com/python/tk_listbox.htm)  The Listbox widget is used to provide a list of options to a user. |
| 8 | [**Menubutton**](https://www.tutorialspoint.com/python/tk_menubutton.htm)  The Menubutton widget is used to display menus in your application. |
| 9 | [**Menu**](https://www.tutorialspoint.com/python/tk_menu.htm)  The Menu widget is used to provide various commands to a user. These commands are contained inside Menubutton. |
| 10 | [**Message**](https://www.tutorialspoint.com/python/tk_message.htm)  The Message widget is used to display multiline text fields for accepting values from a user. |
| 11 | [**Radiobutton**](https://www.tutorialspoint.com/python/tk_radiobutton.htm)  The Radiobutton widget is used to display a number of options as radio buttons. The user can select only one option at a time. |
| 12 | [**Scale**](https://www.tutorialspoint.com/python/tk_scale.htm)  The Scale widget is used to provide a slider widget. |
| 13 | [**Scrollbar**](https://www.tutorialspoint.com/python/tk_scrollbar.htm)  The Scrollbar widget is used to add scrolling capability to various widgets, such as list boxes. |
| 14 | [**Text**](https://www.tutorialspoint.com/python/tk_text.htm)  The Text widget is used to display text in multiple lines. |
| 15 | [**Toplevel**](https://www.tutorialspoint.com/python/tk_toplevel.htm)  The Toplevel widget is used to provide a separate window container. |
| 16 | [**Spinbox**](https://www.tutorialspoint.com/python/tk_spinbox.htm)  The Spinbox widget is a variant of the standard Tkinter Entry widget, which can be used to select from a fixed number of values. |
| 17 | [**PanedWindow**](https://www.tutorialspoint.com/python/tk_panedwindow.htm)  A PanedWindow is a container widget that may contain any number of panes, arranged horizontally or vertically. |
| 18 | [**LabelFrame**](https://www.tutorialspoint.com/python/tk_labelframe.htm)  A labelframe is a simple container widget. Its primary purpose is to act as a spacer or container for complex window layouts. |
| 19 | [**tkMessageBox**](https://www.tutorialspoint.com/python/tk_messagebox.htm)  This module is used to display message boxes in your applications. |

## Example of Creating a GUI

#!/usr/bin/python

import Tkinter

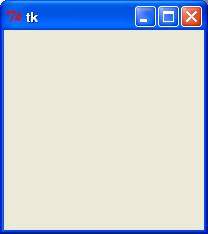
top = Tkinter.Tk()

# Code to add widgets will go here...

top.mainloop()

**OUTPUT:**

This would create a following window −



## Geometry Management

All Tkinter widgets have access to specific geometry management methods, which have the purpose of organizing widgets throughout the parent widget area. Tkinter exposes the following geometry manager classes: pack, grid, and place.

* [The *pack()* Method](https://www.tutorialspoint.com/python/tk_pack.htm) − This geometry manager organizes widgets in blocks before placing them in the parent widget.
* [The *grid()* Method](https://www.tutorialspoint.com/python/tk_grid.htm) − This geometry manager organizes widgets in a table-like structure in the parent widget.
* [The *place()* Method](https://www.tutorialspoint.com/python/tk_place.htm) − This geometry manager organizes widgets by placing them in a specific position in the parent widget.

Let us study the geometry management methods briefly –

This geometry manager organizes widgets in blocks before placing them in the parent widget.

# Python - Tkinter pack() Method

## Syntax

widget.pack( pack\_options )

Here is the list of possible options −

* **expand** − When set to true, widget expands to fill any space not otherwise used in widget's parent.
* **fill** − Determines whether widget fills any extra space allocated to it by the packer, or keeps its own minimal dimensions: NONE (default), X (fill only horizontally), Y (fill only vertically), or BOTH (fill both horizontally and vertically).
* **side** − Determines which side of the parent widget packs against: TOP (default), BOTTOM, LEFT, or RIGHT.

## Example

Try the following example by moving cursor on different buttons −

from Tkinter import \*

root = Tk()

frame = Frame(root)

frame.pack()

bottomframe = Frame(root)

bottomframe.pack( side = BOTTOM )

redbutton = Button(frame, text="Red", fg="red")

redbutton.pack( side = LEFT)

greenbutton = Button(frame, text="Brown", fg="brown")

greenbutton.pack( side = LEFT )

bluebutton = Button(frame, text="Blue", fg="blue")

bluebutton.pack( side = LEFT )

blackbutton = Button(bottomframe, text="Black", fg="black")

blackbutton.pack( side = BOTTOM)

root.mainloop()

When the above code is executed, it produces the following result −



# Python - Tkinter grid() Method

This geometry manager organizes widgets in a table-like structure in the parent widget.

## Syntax

widget.grid( grid\_options )

Here is the list of possible options −

* **column** − The column to put widget in; default 0 (leftmost column).
* **columnspan** − How many columns widgetoccupies; default 1.
* **ipadx, ipady** − How many pixels to pad widget, horizontally and vertically, inside widget's borders.
* **padx, pady** − How many pixels to pad widget, horizontally and vertically, outside v's borders.
* **row** − The row to put widget in; default the first row that is still empty.
* **rowspan** − How many rowswidget occupies; default 1.
* **sticky** − What to do if the cell is larger than widget. By default, with sticky='', widget is centered in its cell. sticky may be the string concatenation of zero or more of N, E, S, W, NE, NW, SE, and SW, compass directions indicating the sides and corners of the cell to which widget sticks.

## Example

Try the following example by moving cursor on different buttons −

import Tkinter

root = Tkinter.Tk( )

for r in range(3):

for c in range(4):

Tkinter.Label(root, text='R%s/C%s'%(r,c),

borderwidth=1 ).grid(row=r,column=c)

root.mainloop( )

This would produce the following result displaying 12 labels arrayed in a 3 × 4 grid −

# TK grid

# Python - Tkinter 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.
* **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.

## Example

Try the following example by moving cursor on different buttons −

from Tkinter import \*

import tkMessageBox

import Tkinter

top = Tkinter.Tk()

def helloCallBack():

tkMessageBox.showinfo( "Hello Python", "Hello World")

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

B.pack()

B.place(bordermode=OUTSIDE, height=100, width=100)

top.mainloop()

When the above code is executed, it produces the following result −

# TK place

# Python - Tkinter 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. You can attach a function or a method to a button which is called automatically when you click the button.

## Syntax

Here is the simple syntax to create this widget −

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

## Parameters

* **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.

|  |  |
| --- | --- |
| **Sr.No.** | **Option & Description** |
| 1 | **activebackground**  Background color when the button is under the cursor. |
| 2 | **activeforeground**  Foreground color when the button is under the cursor. |
| 3 | **bd**  Border width in pixels. Default is 2. |
| 4 | **bg**  Normal background color. |
| 5 | **command**  Function or method to be called when the button is clicked. |
| 6 | **fg**  Normal foreground (text) color. |
| 7 | **font**  Text font to be used for the button's label. |
| 8 | **height**  Height of the button in text lines (for textual buttons) or pixels (for images). |
| 9 | **highlightcolor**  The color of the focus highlight when the widget has focus. |
| 10 | **image**  Image to be displayed on the button (instead of text). |
| 11 | **justify**  How to show multiple text lines: LEFT to left-justify each line; CENTER to center them; or RIGHT to right-justify. |
| 12 | **padx**  Additional padding left and right of the text. |
| 13 | **pady**  Additional padding above and below the text. |
| 14 | **relief**  Relief specifies the type of the border. Some of the values are SUNKEN, RAISED, GROOVE, and RIDGE. |
| 15 | **state**  Set this option to DISABLED to gray out the button and make it unresponsive. Has the value ACTIVE when the mouse is over it. Default is NORMAL. |
| 16 | **underline**  Default is -1, meaning that no character of the text on the button will be underlined. If nonnegative, the corresponding text character will be underlined. |
| 17 | **width**  Width of the button in letters (if displaying text) or pixels (if displaying an image). |
| 18 | **wraplength**  If this value is set to a positive number, the text lines will be wrapped to fit within this length. |

## Methods

Following are commonly used methods for this widget −

|  |  |
| --- | --- |
| **Sr.No.** | **Method & Description** |
| 1 | **flash()**  Causes the button to flash several times between active and normal colors. Leaves the button in the state it was in originally. Ignored if the button is disabled. |
| 2 | **invoke()**  Calls the button's callback, and returns what that function returns. Has no effect if the button is disabled or there is no callback. |

## Example

Try the following example yourself −

import Tkinter

import tkMessageBox

top = Tkinter.Tk()

def helloCallBack():

tkMessageBox.showinfo( "Hello Python", "Hello World")

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

B.pack()

top.mainloop()

When the above code is executed, it produces the following result −

# TK Button

# Python - Tkinter 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.

You can also display images in place of text.

## Syntax

Here is the simple syntax to create this widget −

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

## Parameters

* **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.

|  |  |
| --- | --- |
| **Sr.No.** | **Option & Description** |
| 1 | **activebackground**  Background color when the checkbutton is under the cursor. |
| 2 | **activeforeground**  Foreground color when the checkbutton is under the cursor. |
| 3 | **bg**  The normal background color displayed behind the label and indicator. |
| 4 | **bitmap**  To display a monochrome image on a button. |
| 5 | **bd**  The size of the border around the indicator. Default is 2 pixels. |
| 6 | **command**  A procedure to be called every time the user changes the state of this checkbutton. |
| 7 | **cursor**  If you set this option to a cursor name (*arrow, dot etc.*), the mouse cursor will change to that pattern when it is over the checkbutton. |
| 8 | **disabledforeground**  The foreground color used to render the text of a disabled checkbutton. The default is a stippled version of the default foreground color. |
| 9 | **font**  The font used for the text. |
| 10 | **fg**  The color used to render the text. |
| 11 | **height**  The number of lines of text on the checkbutton. Default is 1. |
| 12 | **highlightcolor**  The color of the focus highlight when the checkbutton has the focus. |
| 13 | **image**  To display a graphic image on the button. |
| 14 | **justify**  If the text contains multiple lines, this option controls how the text is justified: CENTER, LEFT, or RIGHT. |
| 15 | **offvalue**  Normally, a checkbutton's associated control variable will be set to 0 when it is cleared (off). You can supply an alternate value for the off state by setting offvalue to that value. |
| 16 | **onvalue**  Normally, a checkbutton's associated control variable will be set to 1 when it is set (on). You can supply an alternate value for the on state by setting onvalue to that value. |
| 17 | **padx**  How much space to leave to the left and right of the checkbutton and text. Default is 1 pixel. |
| 18 | **pady**  How much space to leave above and below the checkbutton and text. Default is 1 pixel. |
| 19 | **relief**  With the default value, relief=FLAT, the checkbutton does not stand out from its background. You may set this option to any of the other styles |
| 20 | **selectcolor**  The color of the checkbutton when it is set. Default is selectcolor="red". |
| 21 | **selectimage**  If you set this option to an image, that image will appear in the checkbutton when it is set. |
| 22 | **state**  The default is state=NORMAL, but you can use state=DISABLED to gray out the control and make it unresponsive. If the cursor is currently over the checkbutton, the state is ACTIVE. |
| 23 | **text**  The label displayed next to the checkbutton. Use newlines ("\n") to display multiple lines of text. |
| 24 | **underline**  With the default value of -1, none of the characters of the text label are underlined. Set this option to the index of a character in the text (counting from zero) to underline that character. |
| 25 | **variable**  The control variable that tracks the current state of the checkbutton. Normally this variable is an *IntVar*, and 0 means cleared and 1 means set, but see the offvalue and onvalue options above. |
| 26 | **width**  The default width of a checkbutton is determined by the size of the displayed image or text. You can set this option to a number of characters and the checkbutton will always have room for that many characters. |
| 27 | **wraplength**  Normally, lines are not wrapped. You can set this option to a number of characters and all lines will be broken into pieces no longer than that number. |

## Methods

Following are commonly used methods for this widget −

|  |  |
| --- | --- |
| **Sr.No.** | **Method & Description** |
| 1 | **deselect()**  Clears (turns off) the checkbutton. |
| 2 | **flash()**  Flashes the checkbutton a few times between its active and normal colors, but leaves it the way it started. |
| 3 | **invoke()**  You can call this method to get the same actions that would occur if the user clicked on the checkbutton to change its state. |
| 4 | **select()**  Sets (turns on) the checkbutton. |
| 5 | **toggle()**  Clears the checkbutton if set, sets it if cleared. |

## Example

Try the following example yourself −

from Tkinter import \*

import tkMessageBox

import Tkinter

top = Tkinter.Tk()

CheckVar1 = IntVar()

CheckVar2 = IntVar()

C1 = Checkbutton(top, text = "Music", variable = CheckVar1, \

onvalue = 1, offvalue = 0, height=5, \

width = 20)

C2 = Checkbutton(top, text = "Video", variable = CheckVar2, \

onvalue = 1, offvalue = 0, height=5, \

width = 20)

C1.pack()

C2.pack()

top.mainloop()

When the above code is executed, it produces the following result −

# TK Checkbox

# Python - Tkinter Canvas

The Canvas is a rectangular area intended for drawing pictures or other complex layouts. You can place graphics, text, widgets or frames on a Canvas.

## Syntax

Here is the simple syntax to create this widget −

w = Canvas ( master, option=value, ... )

## Parameters

* **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.

|  |  |
| --- | --- |
| **Sr.No.** | **Option & Description** |
| 1 | **bd**  Border width in pixels. Default is 2. |
| 2 | **bg**  Normal background color. |
| 3 | **confine**  If true (the default), the canvas cannot be scrolled outside of the scrollregion. |
| 4 | **cursor**  Cursor used in the canvas like *arrow, circle, dot etc.* |
| 5 | **height**  Size of the canvas in the Y dimension. |
| 6 | **highlightcolor**  Color shown in the focus highlight. |
| 7 | **relief**  Relief specifies the type of the border. Some of the values are SUNKEN, RAISED, GROOVE, and RIDGE. |
| 8 | **scrollregion**  A tuple (w, n, e, s) that defines over how large an area the canvas can be scrolled, where w is the left side, n the top, e the right side, and s the bottom. |
| 9 | **width**  Size of the canvas in the X dimension. |
| 10 | **xscrollincrement**  If you set this option to some positive dimension, the canvas can be positioned only on multiples of that distance, and the value will be used for scrolling by scrolling units, such as when the user clicks on the arrows at the ends of a scrollbar. |
| 11 | **xscrollcommand**  If the canvas is scrollable, this attribute should be the .set() method of the horizontal scrollbar. |
| 12 | **yscrollincrement**  Works like xscrollincrement, but governs vertical movement. |
| 13 | **yscrollcommand**  If the canvas is scrollable, this attribute should be the .set() method of the vertical scrollbar. |

The Canvas widget can support the following standard items −

**arc** − Creates an arc item, which can be a chord, a pieslice or a simple arc.

coord = 10, 50, 240, 210

arc = canvas.create\_arc(coord, start=0, extent=150, fill="blue")

**image** − Creates an image item, which can be an instance of either the BitmapImage or the PhotoImage classes.

filename = PhotoImage(file = "sunshine.gif")

image = canvas.create\_image(50, 50, anchor=NE, image=filename)

**line** − Creates a line item.

line = canvas.create\_line(x0, y0, x1, y1, ..., xn, yn, options)

**oval** − Creates a circle or an ellipse at the given coordinates. It takes two pairs of coordinates; the top left and bottom right corners of the bounding rectangle for the oval.

oval = canvas.create\_oval(x0, y0, x1, y1, options)

**polygon** − Creates a polygon item that must have at least three vertices.

oval = canvas.create\_polygon(x0, y0, x1, y1,...xn, yn, options)

## Example

Try the following example yourself −

import Tkinter

import tkMessageBox

top = Tkinter.Tk()

C = Tkinter.Canvas(top, bg="blue", height=250, width=300)

coord = 10, 50, 240, 210

arc = C.create\_arc(coord, start=0, extent=150, fill="red")

C.pack()

top.mainloop()

When the above code is executed, it produces the following result −

# TK Canvas

# Python - Tkinter Entry

The Entry widget is used to accept single-line text strings from a user.

* If you want to display multiple lines of text that can be edited, then you should use the *Text* widget.
* If you want to display one or more lines of text that cannot be modified by the user, then you should use the *Label* widget.

## Syntax

Here is the simple syntax to create this widget −

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

## Parameters

* **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.

|  |  |
| --- | --- |
| **Sr.No.** | **Option & Description** |
| 1 | **bg**  The normal background color displayed behind the label and indicator. |
| 2 | **bd**  The size of the border around the indicator. Default is 2 pixels. |
| 3 | **command**  A procedure to be called every time the user changes the state of this checkbutton. |
| 4 | **cursor**  If you set this option to a cursor name (*arrow, dot etc.*), the mouse cursor will change to that pattern when it is over the checkbutton. |
| 5 | **font**  The font used for the text. |
| 6 | **exportselection**  By default, if you select text within an Entry widget, it is automatically exported to the clipboard. To avoid this exportation, use exportselection=0. |
| 7 | **fg**  The color used to render the text. |
| 8 | **highlightcolor**  The color of the focus highlight when the checkbutton has the focus. |
| 9 | **justify**  If the text contains multiple lines, this option controls how the text is justified: CENTER, LEFT, or RIGHT. |
| 10 | **relief**  With the default value, relief=FLAT, the checkbutton does not stand out from its background. You may set this option to any of the other styles |
| 11 | **selectbackground**  The background color to use displaying selected text. |
| 12 | **selectborderwidth**  The width of the border to use around selected text. The default is one pixel. |
| 13 | **selectforeground**  The foreground (text) color of selected text. |
| 14 | **show**  Normally, the characters that the user types appear in the entry. To make a .password. entry that echoes each character as an asterisk, set show="\*". |
| 15 | **state**  The default is state=NORMAL, but you can use state=DISABLED to gray out the control and make it unresponsive. If the cursor is currently over the checkbutton, the state is ACTIVE. |
| 16 | **textvariable**  In order to be able to retrieve the current text from your entry widget, you must set this option to an instance of the StringVar class. |
| 17 | **width**  The default width of a checkbutton is determined by the size of the displayed image or text. You can set this option to a number of characters and the checkbutton will always have room for that many characters. |
| 18 | **xscrollcommand**  If you expect that users will often enter more text than the onscreen size of the widget, you can link your entry widget to a scrollbar. |

## Methods

Following are commonly used methods for this widget −

|  |  |
| --- | --- |
| **Sr.No.** | **Method & Description** |
| 1 | **delete ( first, last=None )**  Deletes characters from the widget, starting with the one at index first, up to but not including the character at position last. If the second argument is omitted, only the single character at position first is deleted. |
| 2 | **get()**  Returns the entry's current text as a string. |
| 3 | **icursor ( index )**  Set the insertion cursor just before the character at the given index. |
| 4 | **index ( index )**  Shift the contents of the entry so that the character at the given index is the leftmost visible character. Has no effect if the text fits entirely within the entry. |
| 5 | **insert ( index, s )**  Inserts string s before the character at the given index. |
| 6 | **select\_adjust ( index )**  This method is used to make sure that the selection includes the character at the specified index. |
| 7 | **select\_clear()**  Clears the selection. If there isn't currently a selection, has no effect. |
| 8 | **select\_from ( index )**  Sets the ANCHOR index position to the character selected by index, and selects that character. |
| 9 | **select\_present()**  If there is a selection, returns true, else returns false. |
| 10 | **select\_range ( start, end )**  Sets the selection under program control. Selects the text starting at the start index, up to but not including the character at the end index. The start position must be before the end position. |
| 11 | **select\_to ( index )**  Selects all the text from the ANCHOR position up to but not including the character at the given index. |
| 12 | **xview ( index )**  This method is useful in linking the Entry widget to a horizontal scrollbar. |
| 13 | **xview\_scroll ( number, what )**  Used to scroll the entry horizontally. The what argument must be either UNITS, to scroll by character widths, or PAGES, to scroll by chunks the size of the entry widget. The number is positive to scroll left to right, negative to scroll right to left. |

## Example

Try the following example yourself −

from Tkinter import \*

top = Tk()

L1 = Label(top, text="User Name")

L1.pack( side = LEFT)

E1 = Entry(top, bd =5)

E1.pack(side = RIGHT)

top.mainloop()

When the above code is executed, it produces the following result −

# TK Entry

# Python - Tkinter Frame

The Frame widget is very important for the process of grouping and organizing other widgets in a somehow friendly way. It works like a container, which is responsible for arranging the position of other widgets.

It uses rectangular areas in the screen to organize the layout and to provide padding of these widgets. A frame can also be used as a foundation class to implement complex widgets.

## Syntax

Here is the simple syntax to create this widget −

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

## Parameters

* **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.

|  |  |
| --- | --- |
| **Sr.No.** | **Option & Description** |
| 1 | **bg**  The normal background color displayed behind the label and indicator. |
| 2 | **bd**  The size of the border around the indicator. Default is 2 pixels. |
| 3 | **cursor**  If you set this option to a cursor name (*arrow, dot etc.*), the mouse cursor will change to that pattern when it is over the checkbutton. |
| 4 | **height**  The vertical dimension of the new frame. |
| 5 | **highlightbackground**  Color of the focus highlight when the frame does not have focus. |
| 6 | **highlightcolor**  Color shown in the focus highlight when the frame has the focus. |
| 7 | **highlightthickness**  Thickness of the focus highlight. |
| 8 | **relief**  With the default value, relief=FLAT, the checkbutton does not stand out from its background. You may set this option to any of the other styles |
| 9 | **width**  The default width of a checkbutton is determined by the size of the displayed image or text. You can set this option to a number of characters and the checkbutton will always have room for that many characters. |

## Example

Try the following example yourself −

from Tkinter import \*

root = Tk()

frame = Frame(root)

frame.pack()

bottomframe = Frame(root)

bottomframe.pack( side = BOTTOM )

redbutton = Button(frame, text="Red", fg="red")

redbutton.pack( side = LEFT)

greenbutton = Button(frame, text="Brown", fg="brown")

greenbutton.pack( side = LEFT )

bluebutton = Button(frame, text="Blue", fg="blue")

bluebutton.pack( side = LEFT )

blackbutton = Button(bottomframe, text="Black", fg="black")

blackbutton.pack( side = BOTTOM)

root.mainloop()

When the above code is executed, it produces the following result −

# TK Frame

# Python - Tkinter Label

This widget implements a display box where you can place text or images. The text displayed by this widget can be updated at any time you want.

It is also possible to underline part of the text (like to identify a keyboard shortcut) and span the text across multiple lines.

## Syntax

Here is the simple syntax to create this widget −

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

## Parameters

* **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.

|  |  |
| --- | --- |
| **Sr.No.** | **Option & Description** |
| 1 | **anchor**  This options controls where the text is positioned if the widget has more space than the text needs. The default is anchor=CENTER, which centers the text in the available space. |
| 2 | **bg**  The normal background color displayed behind the label and indicator. |
| 3 | **bitmap**  Set this option equal to a bitmap or image object and the label will display that graphic. |
| 4 | **bd**  The size of the border around the indicator. Default is 2 pixels. |
| 5 | **cursor**  If you set this option to a cursor name (*arrow, dot etc.*), the mouse cursor will change to that pattern when it is over the checkbutton. |
| 6 | **font**  If you are displaying text in this label (with the text or textvariable option, the font option specifies in what font that text will be displayed. |
| 7 | **fg**  If you are displaying text or a bitmap in this label, this option specifies the color of the text. If you are displaying a bitmap, this is the color that will appear at the position of the 1-bits in the bitmap. |
| 8 | **height**  The vertical dimension of the new frame. |
| 9 | **image**  To display a static image in the label widget, set this option to an image object. |
| 10 | **justify**  Specifies how multiple lines of text will be aligned with respect to each other: LEFT for flush left, CENTER for centered (the default), or RIGHT for right-justified. |
| 11 | **padx**  Extra space added to the left and right of the text within the widget. Default is 1. |
| 12 | **pady**  Extra space added above and below the text within the widget. Default is 1. |
| 13 | **relief**  Specifies the appearance of a decorative border around the label. The default is FLAT; for other values. |
| 14 | **text**  To display one or more lines of text in a label widget, set this option to a string containing the text. Internal newlines ("\n") will force a line break. |
| 15 | **textvariable**  To slave the text displayed in a label widget to a control variable of class *StringVar*, set this option to that variable. |
| 16 | **underline**  You can display an underline (\_) below the nth letter of the text, counting from 0, by setting this option to n. The default is underline=-1, which means no underlining. |
| 17 | **width**  Width of the label in characters (not pixels!). If this option is not set, the label will be sized to fit its contents. |
| 18 | **wraplength**  You can limit the number of characters in each line by setting this option to the desired number. The default value, 0, means that lines will be broken only at newlines. |

## Example

Try the following example yourself −

from Tkinter import \*

root = Tk()

var = StringVar()

label = Label( root, textvariable=var, relief=RAISED )

var.set("Hey!? How are you doing?")

label.pack()

root.mainloop()

When the above code is executed, it produces the following result −

# TK Label

# Python - Tkinter Listbox

The Listbox widget is used to display a list of items from which a user can select a number of items.

## Syntax

Here is the simple syntax to create this widget −

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

## Parameters

* **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.

|  |  |
| --- | --- |
| **Sr.No.** | **Option & Description** |
| 1 | **bg**  The normal background color displayed behind the label and indicator. |
| 2 | **bd**  The size of the border around the indicator. Default is 2 pixels. |
| 3 | **cursor**  The cursor that appears when the mouse is over the listbox. |
| 4 | **font**  The font used for the text in the listbox. |
| 5 | **fg**  The color used for the text in the listbox. |
| 6 | **height**  Number of lines (not pixels!) shown in the listbox. Default is 10. |
| 7 | **highlightcolor**  Color shown in the focus highlight when the widget has the focus. |
| 8 | **highlightthickness**  Thickness of the focus highlight. |
| 9 | **relief**  Selects three-dimensional border shading effects. The default is SUNKEN. |
| 10 | **selectbackground**  The background color to use displaying selected text. |
| 11 | **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. |
| 12 | **width**  The width of the widget in characters. The default is 20. |
| 13 | **xscrollcommand**  If you want to allow the user to scroll the listbox horizontally, you can link your listbox widget to a horizontal scrollbar. |
| 14 | **yscrollcommand**  If you want to allow the user to scroll the listbox vertically, you can link your listbox widget to a vertical scrollbar. |

## Methods

Methods on listbox objects include −

|  |  |
| --- | --- |
| **Sr.No.** | **Option & Description** |
| 1 | **activate ( index )**  Selects the line specifies by the given index. |
| 2 | **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. |
| 3 | **delete ( first, last=None )**  Deletes the lines whose indices are in the range [first, last]. If the second argument is omitted, the single line with index first is deleted. |
| 4 | **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. |
| 5 | **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. |
| 6 | **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. |
| 7 | **nearest ( y )**  Return the index of the visible line closest to the y-coordinate y relative to the listbox widget. |
| 8 | **see ( index )**  Adjust the position of the listbox so that the line referred to by index is visible. |
| 9 | **size()**  Returns the number of lines in the listbox. |
| 10 | **xview()**  To make the listbox horizontally scrollable, set the command option of the associated horizontal scrollbar to this method. |
| 11 | **xview\_moveto ( fraction )**  Scroll the listbox so that the leftmost fraction of the width of its longest line is outside the left side of the listbox. Fraction is in the range [0,1]. |
| 12 | **xview\_scroll ( number, what )**  Scrolls the listbox horizontally. For the what argument, use either UNITS to scroll by characters, or PAGES to scroll by pages, that is, by the width of the listbox. The number argument tells how many to scroll. |
| 13 | **yview()**  To make the listbox vertically scrollable, set the command option of the associated vertical scrollbar to this method. |
| 14 | **yview\_moveto ( fraction )**  Scroll the listbox so that the top fraction of the width of its longest line is outside the left side of the listbox. Fraction is in the range [0,1]. |
| 15 | **yview\_scroll ( number, what )**  Scrolls the listbox vertically. For the what argument, use either UNITS to scroll by lines, or PAGES to scroll by pages, that is, by the height of the listbox. The number argument tells how many to scroll. |

## Example

Try the following example yourself −

from Tkinter import \*

import tkMessageBox

import Tkinter

top = Tk()

Lb1 = Listbox(top)

Lb1.insert(1, "Python")

Lb1.insert(2, "Perl")

Lb1.insert(3, "C")

Lb1.insert(4, "PHP")

Lb1.insert(5, "JSP")

Lb1.insert(6, "Ruby")

Lb1.pack()

top.mainloop()

When the above code is executed, it produces the following result −

# TK Listbox

# Python - Tkinter Menubutton

A menubutton is the part of a drop-down menu that stays on the screen all the time. Every menubutton is associated with a Menu widget that can display the choices for that menubutton when the user clicks on it.

## Syntax

Here is the simple syntax to create this widget −

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

## Parameters

* **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.

|  |  |
| --- | --- |
| **Sr.No.** | **Option & Description** |
| 1 | **activebackground**  The background color when the mouse is over the menubutton. |
| 2 | **activeforeground**  The foreground color when the mouse is over the menubutton. |
| 3 | **anchor**  This options controls where the text is positioned if the widget has more space than the text needs. The default is anchor=CENTER, which centers the text. |
| 4 | **bg**  The normal background color displayed behind the label and indicator. |
| 5 | **bitmap**  To display a bitmap on the menubutton, set this option to a bitmap name. |
| 6 | **bd**  The size of the border around the indicator. Default is 2 pixels. |
| 7 | **cursor**  The cursor that appears when the mouse is over this menubutton. |
| 8 | **direction**  Set direction=LEFT to display the menu to the left of the button; use direction=RIGHT to display the menu to the right of the button; or use direction='above' to place the menu above the button. |
| 9 | **disabledforeground**  The foreground color shown on this menubutton when it is disabled. |
| 10 | **fg**  The foreground color when the mouse is not over the menubutton. |
| 11 | **height**  The height of the menubutton in lines of text (not pixels!). The default is to fit the menubutton's size to its contents. |
| 12 | **highlightcolor**  Color shown in the focus highlight when the widget has the focus. |
| 13 | **image**  To display an image on this menubutton, |
| 14 | **justify**  This option controls where the text is located when the text doesn't fill the menubutton: use justify=LEFT to left-justify the text (this is the default); use justify=CENTER to center it, or justify=RIGHT to right-justify. |
| 15 | **menu**  To associate the menubutton with a set of choices, set this option to the Menu object containing those choices. That menu object must have been created by passing the associated menubutton to the constructor as its first argument. |
| 16 | **padx**  How much space to leave to the left and right of the text of the menubutton. Default is 1. |
| 17 | **pady**  How much space to leave above and below the text of the menubutton. Default is 1. |
| 18 | **relief**  Selects three-dimensional border shading effects. The default is RAISED. |
| 19 | **state**  Normally, menubuttons respond to the mouse. Set state=DISABLED to gray out the menubutton and make it unresponsive. |
| 20 | **text**  To display text on the menubutton, set this option to the string containing the desired text. Newlines ("\n") within the string will cause line breaks. |
| 21 | **textvariable**  You can associate a control variable of class StringVar with this menubutton. Setting that control variable will change the displayed text. |
| 22 | **underline**  Normally, no underline appears under the text on the menubutton. To underline one of the characters, set this option to the index of that character. |
| 23 | **width**  The width of the widget in characters. The default is 20. |
| 24 | **wraplength**  Normally, lines are not wrapped. You can set this option to a number of characters and all lines will be broken into pieces no longer than that number. |

## Example

Try the following example yourself −

from Tkinter import \*

import tkMessageBox

import Tkinter

top = Tk()

mb= Menubutton ( top, text="condiments", relief=RAISED )

mb.grid()

mb.menu = Menu ( mb, tearoff = 0 )

mb["menu"] = mb.menu

mayoVar = IntVar()

ketchVar = IntVar()

mb.menu.add\_checkbutton ( label="mayo",

variable=mayoVar )

mb.menu.add\_checkbutton ( label="ketchup",

variable=ketchVar )

mb.pack()

top.mainloop()

When the above code is executed, it produces the following result −

# TK Menubutton

# Python - Tkinter Message

This widget provides a multiline and noneditable object that displays texts, automatically breaking lines and justifying their contents.

Its functionality is very similar to the one provided by the Label widget, except that it can also automatically wrap the text, maintaining a given width or aspect ratio.

## Syntax

Here is the simple syntax to create this widget −

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

## Parameters

* **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.

|  |  |
| --- | --- |
| **Sr.No.** | **Option & Description** |
| 1 | **anchor**  This options controls where the text is positioned if the widget has more space than the text needs. The default is anchor=CENTER, which centers the text in the available space. |
| 2 | **bg**  The normal background color displayed behind the label and indicator. |
| 3 | **bitmap**  Set this option equal to a bitmap or image object and the label will display that graphic. |
| 4 | **bd**  The size of the border around the indicator. Default is 2 pixels. |
| 5 | **cursor**  If you set this option to a cursor name (*arrow, dot etc.*), the mouse cursor will change to that pattern when it is over the checkbutton. |
| 6 | **font**  If you are displaying text in this label (with the text or textvariable option, the font option specifies in what font that text will be displayed. |
| 7 | **fg**  If you are displaying text or a bitmap in this label, this option specifies the color of the text. If you are displaying a bitmap, this is the color that will appear at the position of the 1-bits in the bitmap. |
| 8 | **height**  The vertical dimension of the new frame. |
| 9 | **image**  To display a static image in the label widget, set this option to an image object. |
| 10 | **justify**  Specifies how multiple lines of text will be aligned with respect to each other: LEFT for flush left, CENTER for centered (the default), or RIGHT for right-justified. |
| 11 | **padx**  Extra space added to the left and right of the text within the widget. Default is 1. |
| 12 | **pady**  Extra space added above and below the text within the widget. Default is 1. |
| 13 | **relief**  Specifies the appearance of a decorative border around the label. The default is FLAT; for other values. |
| 14 | **text**  To display one or more lines of text in a label widget, set this option to a string containing the text. Internal newlines ("\n") will force a line break. |
| 15 | **textvariable**  To slave the text displayed in a label widget to a control variable of class *StringVar*, set this option to that variable. |
| 16 | **underline**  You can display an underline (\_) below the nth letter of the text, counting from 0, by setting this option to n. The default is underline=-1, which means no underlining. |
| 17 | **width**  Width of the label in characters (not pixels!). If this option is not set, the label will be sized to fit its contents. |
| 18 | **wraplength**  You can limit the number of characters in each line by setting this option to the desired number. The default value, 0, means that lines will be broken only at newlines. |

## Example

Try the following example yourself −

from Tkinter import \*

root = Tk()

var = StringVar()

label = Message( root, textvariable=var, relief=RAISED )

var.set("Hey!? How are you doing?")

label.pack()

root.mainloop()

When the above code is executed, it produces the following result −

# TK Message

# Python - Tkinter Radiobutton

This widget implements a multiple-choice button, which is a way to offer many possible selections to the user and lets user choose only one of them.

In order to implement this functionality, each group of radiobuttons must be associated to the same variable and each one of the buttons must symbolize a single value. You can use the Tab key to switch from one radionbutton to another.

## Syntax

Here is the simple syntax to create this widget −

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

## Parameters

* **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.

|  |  |
| --- | --- |
| **Sr.No.** | **Option & Description** |
| 1 | **activebackground**  The background color when the mouse is over the radiobutton. |
| 2 | **activeforeground**  The foreground color when the mouse is over the radiobutton. |
| 3 | **anchor**  If the widget inhabits a space larger than it needs, this option specifies where the radiobutton will sit in that space. The default is anchor=CENTER. |
| 4 | **bg**  The normal background color behind the indicator and label. |
| 5 | **bitmap**  To display a monochrome image on a radiobutton, set this option to a bitmap. |
| 6 | **borderwidth**  The size of the border around the indicator part itself. Default is 2 pixels. |
| 7 | **command**  A procedure to be called every time the user changes the state of this radiobutton. |
| 8 | **cursor**  If you set this option to a cursor name (*arrow, dot etc.*), the mouse cursor will change to that pattern when it is over the radiobutton. |
| 9 | **font**  The font used for the text. |
| 10 | **fg**  The color used to render the text. |
| 11 | **height**  The number of lines (not pixels) of text on the radiobutton. Default is 1. |
| 12 | **highlightbackground**  The color of the focus highlight when the radiobutton does not have focus. |
| 13 | **highlightcolor**  The color of the focus highlight when the radiobutton has the focus. |
| 14 | **image**  To display a graphic image instead of text for this radiobutton, set this option to an image object. |
| 15 | **justify**  If the text contains multiple lines, this option controls how the text is justified: CENTER (the default), LEFT, or RIGHT. |
| 16 | **padx**  How much space to leave to the left and right of the radiobutton and text. Default is 1. |
| 17 | **pady**  How much space to leave above and below the radiobutton and text. Default is 1. |
| 18 | **relief**  Specifies the appearance of a decorative border around the label. The default is FLAT; for other values. |
| 19 | **selectcolor**  The color of the radiobutton when it is set. Default is red. |
| 20 | **selectimage**  If you are using the image option to display a graphic instead of text when the radiobutton is cleared, you can set the selectimage option to a different image that will be displayed when the radiobutton is set. |
| 21 | **state**  The default is state=NORMAL, but you can set state=DISABLED to gray out the control and make it unresponsive. If the cursor is currently over the radiobutton, the state is ACTIVE. |
| 22 | **text**  The label displayed next to the radiobutton. Use newlines ("\n") to display multiple lines of text. |
| 23 | **textvariable**  To slave the text displayed in a label widget to a control variable of class *StringVar*, set this option to that variable. |
| 24 | **underline**  You can display an underline (\_) below the nth letter of the text, counting from 0, by setting this option to n. The default is underline=-1, which means no underlining. |
| 25 | **value**  When a radiobutton is turned on by the user, its control variable is set to its current value option. If the control variable is an *IntVar*, give each radiobutton in the group a different integer value option. If the control variable is a *StringVar*, give each radiobutton a different string value option. |
| 26 | **variable**  The control variable that this radiobutton shares with the other radiobuttons in the group. This can be either an IntVar or a StringVar. |
| 27 | **width**  Width of the label in characters (not pixels!). If this option is not set, the label will be sized to fit its contents. |
| 28 | **wraplength**  You can limit the number of characters in each line by setting this option to the desired number. The default value, 0, means that lines will be broken only at newlines. |

## Methods

|  |  |
| --- | --- |
| **Sr.No.** | **Method & Description** |
| 1 | **deselect()**  Clears (turns off) the radiobutton. |
| 2 | **flash()**  Flashes the radiobutton a few times between its active and normal colors, but leaves it the way it started. |
| 3 | **invoke()**  You can call this method to get the same actions that would occur if the user clicked on the radiobutton to change its state. |
| 4 | **select()**  Sets (turns on) the radiobutton. |

## Example

Try the following example yourself −

from Tkinter import \*

def sel():

selection = "You selected the option " + str(var.get())

label.config(text = selection)

root = Tk()

var = IntVar()

R1 = Radiobutton(root, text="Option 1", variable=var, value=1,

command=sel)

R1.pack( anchor = W )

R2 = Radiobutton(root, text="Option 2", variable=var, value=2,

command=sel)

R2.pack( anchor = W )

R3 = Radiobutton(root, text="Option 3", variable=var, value=3,

command=sel)

R3.pack( anchor = W)

label = Label(root)

label.pack()

root.mainloop()

When the above code is executed, it produces the following result −

# TK Radiobutton

# 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.

## Syntax

Here is the simple syntax to create this widget −

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

## Parameters

* **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.

|  |  |
| --- | --- |
| **Sr.No.** | **Option & Description** |
| 1 | **bg**  The default background color of the text widget. |
| 2 | **bd**  The width of the border around the text widget. Default is 2 pixels. |
| 3 | **cursor**  The cursor that will appear when the mouse is over the text widget. |
| 4 | **exportselection**  Normally, text selected within a text widget is exported to be the selection in the window manager. Set exportselection=0 if you don't want that behavior. |
| 5 | **font**  The default font for text inserted into the widget. |

## Methods

Text objects have these methods −

|  |  |
| --- | --- |
| **Sr.No.** | **Methods & Description** |
| 1 | **delete(startindex [,endindex])**  This method deletes a specific character or a range of text. |
| 2 | **get(startindex [,endindex])**  This method returns a specific character or a range of text. |
| 3 | **index(index)**  Returns the absolute value of an index based on the given index. |
| 4 | **insert(index [,string]...)**  This method inserts strings at the specified index location. |
| 5 | **see(index)**  This method returns true if the text located at the index position is visible. |

Text widgets support three distinct helper structures: Marks, Tabs, and Indexes −

Marks are used to bookmark positions between two characters within a given text. We have the following methods available when handling marks −

|  |  |
| --- | --- |
| **Sr.No.** | **Methods & Description** |
| 1 | **index(mark)**  Returns the line and column location of a specific mark. |
| 2 | **mark\_gravity(mark [,gravity])**  Returns the gravity of the given mark. If the second argument is provided, the gravity is set for the given mark. |
| 3 | **mark\_names()**  Returns all marks from the Text widget. |
| 4 | **mark\_set(mark, index)**  Informs a new position to the given mark. |
| 5 | **mark\_unset(mark)**  Removes the given mark from the Text widget. |

Tags are used to associate names to regions of text which makes easy the task of modifying the display settings of specific text areas. Tags are also used to bind event callbacks to specific ranges of text.

Following are the available methods for handling tabs −

|  |  |
| --- | --- |
| **Sr.No.** | **Methods & Description** |
| 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. |

## Example

Try the following example yourself −

from Tkinter import \*

def onclick():

pass

root = Tk()

text = Text(root)

text.insert(INSERT, "Hello.....")

text.insert(END, "Bye Bye.....")

text.pack()

text.tag\_add("here", "1.0", "1.4")

text.tag\_add("start", "1.8", "1.13")

text.tag\_config("here", background="yellow", foreground="blue")

text.tag\_config("start", background="black", foreground="green")

root.mainloop()

When the above code is executed, it produces the following result −

# TK Text

# Python - Tkinter Spinbox

The Spinbox widget is a variant of the standard Tkinter Entry widget, which can be used to select from a fixed number of values.

## Syntax

Here is the simple syntax to create this widget −

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

## Parameters

* **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.

|  |  |
| --- | --- |
| **Sr.No.** | **Option & Description** |
| 1 | **activebackground**  The color of the slider and arrowheads when the mouse is over them. |
| 2 | **bg**  The color of the slider and arrowheads when the mouse is not over them. |
| 3 | **bd**  The width of the 3-d borders around the entire perimeter of the trough, and also the width of the 3-d effects on the arrowheads and slider. Default is no border around the trough, and a 2-pixel border around the arrowheads and slider. |
| 4 | **command**  A procedure to be called whenever the scrollbar is moved. |
| 5 | **cursor**  The cursor that appears when the mouse is over the scrollbar. |

## Methods

Spinbox objects have these methods −

|  |  |
| --- | --- |
| **Sr.No.** | **Methods & Description** |
| 1 | **delete(startindex [,endindex])**  This method deletes a specific character or a range of text. |
| 2 | **get(startindex [,endindex])**  This method returns a specific character or a range of text. |
| 3 | **identify(x, y)**  Identifies the widget element at the given location. |
| 4 | **index(index)**  Returns the absolute value of an index based on the given index. |
| 5 | **insert(index [,string]...)**  This method inserts strings at the specified index location. |
| 6 | **invoke(element)**  Invokes a spinbox button. |

## Example

Try the following example yourself −

from Tkinter import \*

master = Tk()

w = Spinbox(master, from\_=0, to=10)

w.pack()

mainloop()

When the above code is executed, it produces the following result −

# TK Spinbox

# Events and Bindings

As was mentioned earlier, a Tkinter application spends most of its time inside an event loop (entered via the **mainloop** method). Events can come from various sources, including key presses and mouse operations by the user, and redraw events from the window manager (indirectly caused by the user, in many cases).

Tkinter provides a powerful mechanism to let you deal with events yourself. For each widget, you can **bind** Python functions and methods to events.

widget.bind(event, handler)

If an event matching the *event* description occurs in the widget, the given *handler* is called with an object describing the event.