GUI Programming (Tkinter)

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

- **Tkinter:** Tkinter is the Python interface to the Tk GUI toolkit shipped with Python. We would look at this option in this chapter.
- **wxPython:** This is an open-source Python interface for wxWidgets GUI toolkit.
- **JPython:** JPython is a Python port for Java, which gives Python scripts seamless access to the Java class libraries on the local machinehttp://www.jython.org.

There are many other interfaces available, which you can find them on the net.

Tkinter Programming

Tkinter is the standard GUI library for Python. 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.

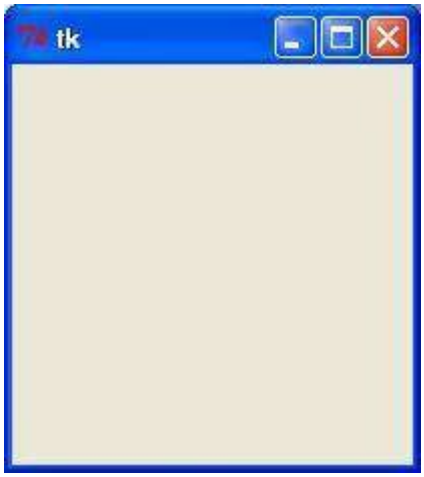
Creating a GUI application using Tkinter is an easy task. All you need to do is perform the following steps –

- Import the *Tkinter* module.
- Create the GUI application main window.
- Add one or more of the above-mentioned widgets to the GUI application.
- Enter the main event loop to take action against each event triggered by the user.

Example

```
import tkinter # note that module name has changed from Tkinter in Python 2 to tkinter in Python 3

top = tkinter.Tk()

# Code to add widgets will go here...

top.mainloop()
```

This would create a following window-



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-

Operator	Description
Button	The Button widget is used to display the buttons in your application.
Canvas	The Canvas widget is used to draw shapes, such as lines, ovals, polygons and rectangles, in your application.
Checkbutton	The Checkbutton widget is used to display a number of options as checkboxes. The user can select multiple options at a time.
Entry	The Entry widget is used to display a single-line text field for accepting values from a user.
Frame	The Frame widget is used as a container widget to organize other widgets.
Label	The Label widget is used to provide a single-line caption for other widgets. It can also contain images.

Listbox	The Listbox widget is used to provide a list of options to a user.
Menubutton	The Menubutton widget is used to display menus in your application.
Menu	The Menu widget is used to provide various commands to a user. These commands are contained inside Menubutton.
Message	The Message widget is used to display multiline text fields for accepting values from a user.
Radiobutton	The Radiobutton widget is used to display a number of options as radio buttons. The user can select only one option at a time.
Scale	The Scale widget is used to provide a slider widget.
Scrollbar	The Scrollbar widget is used to add scrolling capability to various widgets, such as list boxes.
Text	The Text widget is used to display text in multiple lines.
Toplevel	The Toplevel widget is used to provide a separate window container.
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.
PanedWindow	A PanedWindow is a container widget that may contain any number of panes, arranged horizontally or vertically.
LabelFrame	A labelframe is a simple container widget. Its primary purpose is to act as a spacer or container for complex window layouts.
tkMessageBox	This module is used to display the message boxes in your applications.

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.

Option	Description
activebackground	Background color when the button is under the cursor.
activeforeground	Foreground color when the button is under the cursor.
bd	Border width in pixels. Default is 2.
bg	Normal background color.
command	Function or method to be called when the button is clicked.
fg	Normal foreground (text) color.
font	Text font to be used for the button's label.
height	Height of the button in text lines (for textual buttons) or pixels (for images).

highlightcolor	The color of the focus highlight when the widget has focus.
image	Image to be displayed on the button (instead of text).
justify	How to show multiple text lines: LEFT to left-justify each line; CENTER to center them; or RIGHT to right-justify.
padx	Additional padding left and right of the text.
pady	Additional padding above and below the text.
relief	Relief specifies the type of the border. Some of the values are SUNKEN, RAISED, GROOVE, and RIDGE.
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.
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.
width	Width of the button in letters (if displaying text) or pixels (if displaying an image).
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-

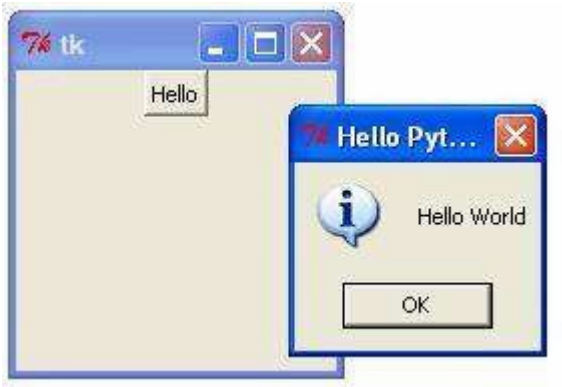
Method	Description
flash()	Causes the button to flash several times between active and normal colors. Leaves the button in the state it was in originally. Ignored if the button is disabled.
invoke()	Calls the button's callback, and returns what that function returns. Has no effect if the button is disabled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

Example

Try the following example yourself-

```
from tkinter import *
from tkinter import
messagebox
top = Tk()
top.geometry("100x100")
def helloCallBack():
    msg=messagebox.showinfo( "Hello Python", "Hello
World") B = Button(top, text ="Hello", command =
helloCallBack) B.place(x=50,y=50)
top.mainloop()
```

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



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.

Option	Description
activebackground	Background color when the checkbutton is under the cursor.
activeforeground	Foreground color when the checkbutton is under the cursor.
bg	The normal background color displayed behind the label and indicator.
bitmap	To display a monochrome image on a button.
bd	The size of the border around the indicator. Default is 2 pixels.
command	A procedure to be called every time the user changes the state of this checkbutton.
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.*
disabledforeground	The foreground color used to render the text of a disabled checkbutton. The default is a stippled version of the default foreground color.
font	The font used for the text.
fg	The color used to render the text.
height	The number of lines of text on the checkbutton. Default is 1.
highlightcolor	The color of the focus highlight when the checkbutton has the focus.
image	To display a graphic image on the button.
justify	If the text contains multiple lines, this option controls how the text is justified: CENTER, LEFT, or RIGHT.

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.
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.

padx	How much space to leave to the left and right of the checkbutton and text. Default is 1 pixel.
pady	How much space to leave above and below the checkbutton and text. Default is 1 pixel.
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
selectcolor	The color of the checkbutton when it is set. Default is selectcolor="red".
selectimage	If you set this option to an image, that image will appear in the checkbutton when it is set.
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.
text	The label displayed next to the checkbutton. Use newlines ("\n") to display multiple lines of text.
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.
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.
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.

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-

Method	Description
deselect()	Clears (turns off) the checkbutton.
flash()	Flashes the checkbutton a few times between its active and normal colors, but leaves it the way it started.
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.
select()	Sets (turns on) the checkbutton.
toggle()	Clears the checkbutton if set, sets it if cleared.

Example

Try the following example yourself-

```
from tkinter import *

import tkinter

top =  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 –

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.

Option	Description

bg	The normal background color displayed behind the label and indicator.
bd	The size of the border around the indicator. Default is 2 pixels.

command	A procedure to be called every time the user changes the state of this checkbutton.
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.
font	The font used for the text.
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.
fg	The color used to render the text.
highlightcolor	The color of the focus highlight when the checkbutton has the focus.
justify	If the text contains multiple lines, this option controls how the text is justified: CENTER, LEFT, or RIGHT.
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
selectbackground	The background color to use displaying selected text.
selectborderwidth	The width of the border to use around selected text. The default is one pixel.
selectforeground	The foreground (text) color of selected text.
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="*".
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.

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.

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.
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-

Method	Description
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.
get()	Returns the entry's current text as a string.
icursor (index)	Set the insertion cursor just before the character at the given index.
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.
insert (index, s)	Inserts string s before the character at the given index.
select_adjust (index)	This method is used to make sure that the selection includes the character at the specified index.

select_clear()	Clears the selection. If there isn't currently a selection, has no effect.

select_from (index)	Sets the ANCHOR index position to the character selected by index, and selects that character.
select_present()	If there is a selection, returns true, else returns false.
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.
select_to (index)	Selects all the text from the ANCHOR position up to but not including the character at the given index.
xview (index)	This method is useful in linking the Entry widget to a horizontal scrollbar.
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-

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.

Options	Description
bg	The normal background color displayed behind the label and indicator.
bd	The size of the border around the indicator. Default is 2 pixels.
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.
height	The vertical dimension of the new frame.
highlightbackground	Color of the focus highlight when the frame does not have focus.
highlightcolor	Color shown in the focus highlight when the frame has the focus.

highlightthickness	Thickness of the focus highlight.
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
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-

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.

Options	Description
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.
bg	The normal background color displayed behind the label and indicator.
bitmap	Set this option equal to a bitmap or image object and the label will display that graphic.
bd	The size of the border around the indicator. Default is 2 pixels.
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.

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.

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.
height	The vertical dimension of the new frame.
image	To display a static image in the label widget, set this option to an image object.
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.
padx	Extra space added to the left and right of the text within the widget. Default is 1.
pady	Extra space added above and below the text within the widget. Default is 1.
relief	Specifies the appearance of a decorative border around the label. The default is FLAT; for other values.
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.
textvariable	To slave the text displayed in a label widget to a control variable of class StringVar, set this option to that variable.
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.
width	Width of the label in characters (not pixels!). If this option is not set, the label will be sized to fit its contents.
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-

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.

Options	Description
activebackground	The background color when the mouse is over the radiobutton.
activeforeground	The foreground color when the mouse is over the radiobutton.
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.
bg	The normal background color behind the indicator and label.

bitmap	To display a monochrome image on a radiobutton, set this option to a bitmap.
borderwidth	The size of the border around the indicator part itself. Default is 2 pixels.
command	A procedure to be called every time the user changes the state of this radiobutton.
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.
font	The font used for the text.
fg	The color used to render the text.
height	The number of lines (not pixels) of text on the radiobutton. Default is 1.
highlightbackground	The color of the focus highlight when the radiobutton does not have focus.
highlightcolor	The color of the focus highlight when the radiobutton has the focus.
image	To display a graphic image instead of text for this radiobutton, set this option to an image object.
justify	If the text contains multiple lines, this option controls how the text is justified: CENTER (the default), LEFT, or RIGHT.

padx	How much space to leave to the left and right of the radiobutton and text. Default is 1.
pady	How much space to leave above and below the radiobutton and text. Default is 1.
relief	Specifies the appearance of a decorative border around the label. The default is FLAT; for other values.
selectcolor	The color of the radiobutton when it is set. Default is red.

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.
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.
text	The label displayed next to the radiobutton. Use newlines ("\n") to display multiple lines of text.
textvariable	To slave the text displayed in a label widget to a control variable of class *StringVar*, set this option to that variable.
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.
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.
variable	The control variable that this radiobutton shares with the other radiobuttons in the group. This can be either an IntVar or a StringVar.
width	Width of the label in characters (not pixels!). If this option is not set, the label will be sized to fit its contents.
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

These methods are available.

Methods	Description
deselect()	Clears (turns off) the radiobutton.

flash()	Flashes the radiobutton a few times between its active and normal colors, but leaves it the way it started.
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.
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-

Tkinter tkMessageBox

The tkMessageBox module is used to display message boxes in your applications. This module provides a number of functions that you can use to display an appropriate message.

Some of these functions are showinfo, showwarning, showerror, askquestion, askokcancel, askyesno, and askretryignore.

Syntax

Here is the simple syntax to create this widget-

```
tkMessageBox.FunctionName(title, message [, options])
```

Parameters

- **FunctionName:** This is the name of the appropriate message box function.
- **title:** This is the text to be displayed in the title bar of a message box.
- **message:** This is the text to be displayed as a message.
- **options:** options are alternative choices that you may use to tailor a standard message box. Some of the options that you can use are default and parent. The

  default option is used to specify the default button, such as ABORT, RETRY, or IGNORE in the message box. The parent option is used to specify the window on top of which the message box is to be displayed.

You could use one of the following functions with dialogue box-

- showinfo()
- showwarning()
- showerror ()
- askquestion()
- askokcancel()
- askyesno ()
- askretrycancel ()

Example

Try the following example yourself-

```
from tkinter import *

from tkinter import messagebox

top = Tk()
top.geometry("100x10
0") def hello():
  messagebox.showinfo("Say Hello", "Hello World")

B1 = Button(top, text = "Say Hello", command = hello)
B1.place(x=35,y=50)

top.mainloop()
```

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

Standard Attributes

Let us look at how some of the common attributes, such as sizes, colors and fonts are specified.

- Dimensions
- Colors
- Fonts
- Anchors
- Relief styles
- Bitmaps
- Cursors

Tkinter Dimensions

Various lengths, widths, and other dimensions of widgets can be described in many different units.

- If you set a dimension to an integer, it is assumed to be in pixels.
- You can specify units by setting a dimension to a string containing a number followed by.

Character	Description
c	Centimeters
i	Inches
m	Millimeters
p	Printer's points (about 1/72")

Length options

Tkinter expresses a length as an integer number of pixels. Here is the list of common length options-

- **borderwidth:** Width of the border which gives a three-dimensional look to the widget.
- **highlightthickness:** Width of the highlight rectangle when the widget has focus .
- **padX padY:** Extra space the widget requests from its layout manager beyond the minimum the widget needs to display its contents in the x and y directions.
- **selectborderwidth:** Width of the three-dimentional border around selected

items of the widget.

- **wraplength:** Maximum line length for widgets that perform word wrapping.
- **height:** Desired height of the widget; must be greater than or equal to 1.
- **underline:** Index of the character to underline in the widget's text (0 is the first character, 1 the second one, and so on).
- **width:** Desired width of the widget.

Tkinter Colors

Tkinter represents colors with strings. There are two general ways to specify colors in Tkinter-

- You can use a string specifying the proportion of red, green and blue in hexadecimal digits. For example, "#fff" is white, "#000000" is black, "#000fff000" is pure green, and "#00ffff" is pure cyan (green plus blue).
- You can also use any locally defined standard color name. The colors "white", "black", "red", "green", "blue", "cyan", "yellow", and "magenta" will always be available.

Color options

The common color options are-

- **activebackground:** Background color for the widget when the widget is active.
- **activeforeground:** Foreground color for the widget when the widget is active.
- **background:** Background color for the widget. This can also be represented as *bg*.
- **disabledforeground:** Foreground color for the widget when the widget is disabled.
- **foreground:** Foreground color for the widget. This can also be represented as *fg*.
- **highlightbackground:** Background color of the highlight region when the widget has focus.
- **highlightcolor:** Foreground color of the highlight region when the widget has focus.
- **selectbackground:** Background color for the selected items of the widget.
- **selectforeground:** Foreground color for the selected items of the widget.

Tkinter Fonts

There may be up to three ways to specify type style.

Simple Tuple Fonts

As a tuple whose first element is the font family, followed by a size in points, optionally followed by a string containing one or more of the style modifiers bold, italic, underline and overstrike.

Example

- ("Helvetica", "16") for a 16-point Helvetica regular.
- ("Times", "24", "bold italic") for a 24-point Times bold italic.

Font object Fonts

You can create a "font object" by importing the tkFont module and using its Font class constructor –

```
import tkFont
font = tkFont.Font ( option, ... )
```

Here is the list of options-

- **family:** The font family name as a string.
- **size:** The font height as an integer in points. To get a font n pixels high, use -n.
- **weight:** "bold" for boldface, "normal" for regular weight.
- **slant:** "italic" for italic, "roman" for unslanted.
- **underline:** 1 for underlined text, 0 for normal.
- **overstrike:** 1 for overstruck text, 0 for normal.

Example

```
helv36 = tkFont.Font(family="Helvetica",size=36,weight="bold")
```

X Window Fonts

If you are running under the X Window System, you can use any of the X font names.

For example, the font named "-*-lucidatypewriter-medium-r-*-*-*-140-*-*-*-*-*-*" is the author's favorite fixed-width font for onscreen use. Use the*xfontsel* program to help you select pleasing fonts.

Tkinter Anchors

Anchors are used to define where text is positioned relative to a reference point. Here is list of possible constants, which can be used for Anchor attribute.

- NW
- N
- NE
- W
- CENTER
- E
- SW
- S
- SE

For example, if you use CENTER as a text anchor, the text will be centered horizontally and vertically around the reference point.

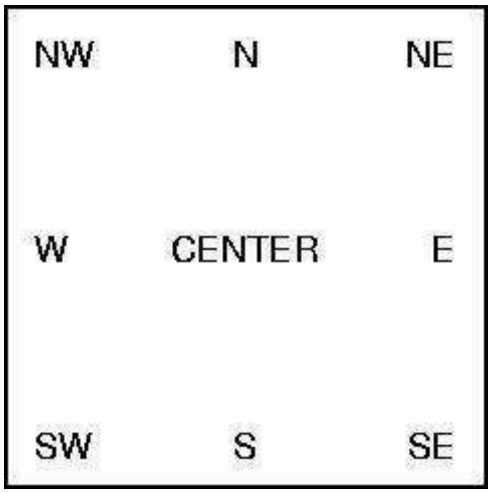
Anchor NW will position the text so that the reference point coincides with the northwest (top left) corner of the box containing the text.

Anchor W will center the text vertically around the reference point, with the left edge of the text box passing through that point, and so on.

If you create a small widget inside a large frame and use the anchor=SE option, the widget will be placed in the bottom right corner of the frame. If you used anchor=N instead, the widget would be centered along the top edge.

Example

The anchor constants are shown in this diagram-



Tkinter Relief styles

The relief style of a widget refers to certain simulated 3-D effects around

the outside of the widget. Here is a screenshot of a row of buttons exhibiting all the possible relief styles-

Here is list of possible constants which can be used for relief attribute-

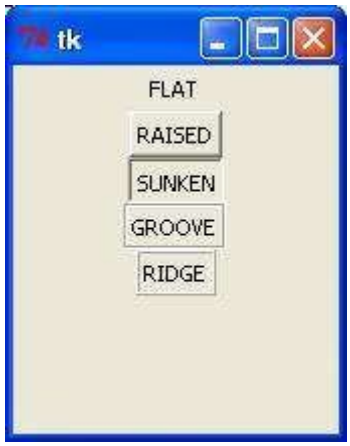
- FLAT
- RAISED
- SUNKEN
- GROOVE
- RIDGE

Example

```
from tkinter import *
import tkinter

top = Tk()

B1 = Button(top, text ="FLAT", relief=FLAT )
```

```
B2 = Button(top, text ="RAISED", relief=RAISED
) B3 = Button(top, text ="SUNKEN",
relief=SUNKEN ) B4 = Button(top, text
="GROOVE", relief=GROOVE ) B5 = Button(top,
text ="RIDGE", relief=RIDGE )

B1.pack()
B2.pack()
B3.pack()
B4.pack()
B5.pack()
top.mainloop(
)
```

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



Tkinter Bitmaps

This attribute to displays a bitmap. There are following type of bitmaps available-

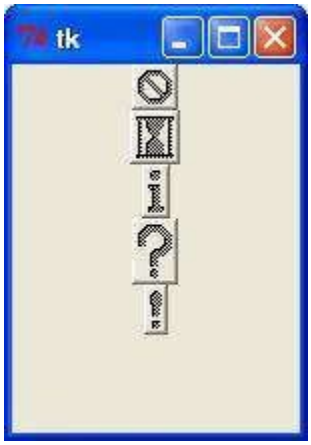
- "error"
- "gray75"
- "gray50"
- "gray25"
- "gray12"
- "hourglass"
- "info"
- "questhead"
- "question"
- "warning"

Example

```
from tkinter import *
import tkinter

top = Tk()

B1 = Button(top, text ="error", relief=RAISED,\
                                bitmap="error")
B2 = Button(top, text ="hourglass", relief=RAISED,\
```

```
                              bitmap="hourglass")
B3 = Button(top, text ="info", relief=RAISED,\
                              bitmap="info")
B4 = Button(top, text ="question", relief=RAISED,\
                              bitmap="question")
B5 = Button(top, text ="warning", relief=RAISED,\
                              bitmap="warning")

B1.pack()
B2.pack()
B3.pack()
B4.pack()
B5.pack()
top.mainloop(
)
```

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



Tkinter Cursors

Python Tkinter supports quite a number of different mouse cursors available. The exact graphic may vary according to your operating system.

Here is the list of interesting ones-

- "arrow"
- "circle"
- "clock"
- "cross"
- "dotbox"
- "exchange"

- "fleur"
- "heart"
- "heart"
- "man"
- "mouse"
- "pirate"
- "plus"
- "shuttle"
- "sizing"
- "spider"
- "spraycan"
- "star"
- "target"
- "tcross"
- "trek"
- "watch"

Example

Try the following example by moving cursor on different buttons-

```
from tkinter import *
import tkinter

top = Tk()

B1 = Button(top, text ="circle", relief=RAISED,\
                              cursor="circle")
B2 = Button(top, text ="plus", relief=RAISED,\
                              cursor="plus")
B1.pack()
B2.pack()
top.mainloop(
)
```

Geometry Management

All Tkinter widgets have access to the 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 - This geometry manager organizes widgets in blocks before placing them in the parent widget.
- The grid() Method - This geometry manager organizes widgets in a table-like structure in the parent widget.
- The place() Method -This geometry manager organizes widgets by placing themin a specific position in the parent widget.

Let us study the geometry management methods briefly –

Tkinter pack() Method

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

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-

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-

```
#/python/keerti
from        tkinter
import * root = Tk(
    )
```

```
b=0
for r in  range(6): for c
    in range(6):
        b=b+1
        Button(root, text=str(b),
            borderwidth=1 ).grid(row=r,column=c)
root.mainloop()
```

This would produce the following result displaying 12 labels arrayed in a 3 x 4 grid-

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-

```
ffrom tkinter import *

top = Tk()
L1 = Label(top, text="Physics")
L1.place(x=10,y=10)
E1 = Entry(top, bd =5)
E1.place(x=60,y=10)
L2=Label(top,text="Maths")
L2.place(x=10,y=50)
E2=Entry(top,bd=5)
E2.place(x=60,y=50)

L3=Label(top,text="Total")
L3.place(x=10,y=150)
E3=Entry(top,bd=5)
E3.place(x=60,y=150)

B = Button(top, text ="Add")
B.place(x=100, y=100)
top.geometry("250x250+10+10
")
top.mainloop()
```

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