**UNIT-V GRAPHICAL USER INTERFACES**

**Graphical User Interfaces:** Behavior of terminal based programs and GUI-based programs, Coding simple GUI-based programs, other useful GUI resources. GUI Programming: Graphical User Interfaces, Using the Tkinter Module, Display text with Label Widgets, Organizing Widgets with Frames, Button Widgets and Info Dialog Boxes, Getting Input with Entry Widget, Using Labels as Output Fields, Radio Buttons, Check Buttons.

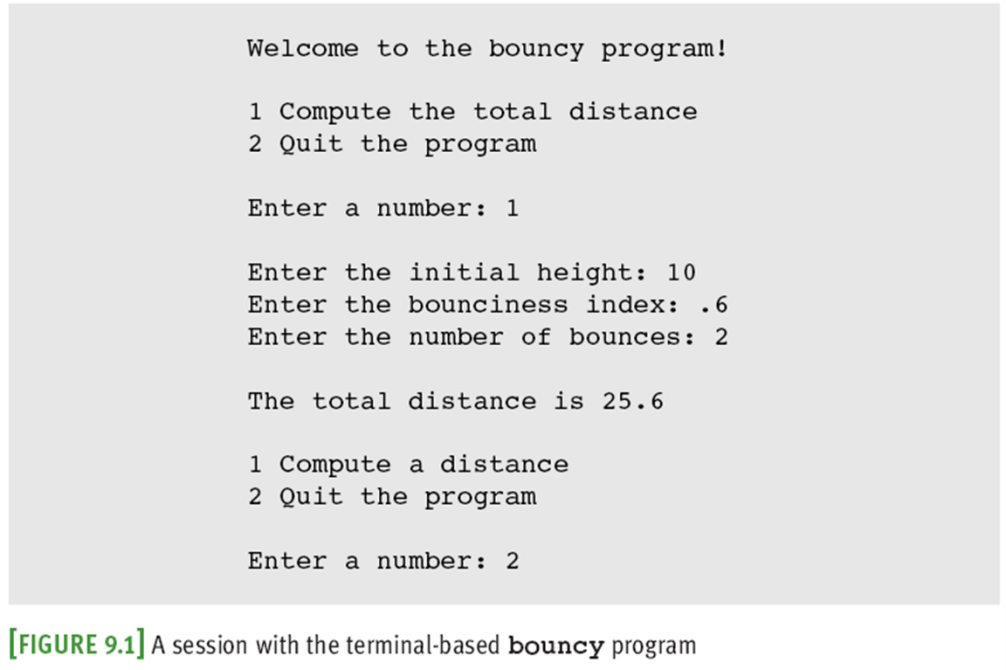
**Graphical User Interfaces:**

* Most modern computer software employs a graphical user interface or GUI
* A GUI displays text as well as small images (called icons) that represent objects such as directories, files of different types, command buttons, and drop-down menus
* In addition to entering text at keyboard, the user of a GUI can select an icon with pointing device, such as mouse, and move that icon around on the display

**Behavior of terminal based programs and GUI-based programs:**

* Two different versions of the **bouncy** program from a user’s point of view:
  + Terminal-based user interface
  + Graphical user interface
* Both programs perform exactly the same function
  + However, their behavior, or look and feel, from a user’s perspective are quite different

**The Terminal-Based Version:**



**Problems with The Terminal-Based Version**:

* User is constrained to reply to a definite sequence of prompts for inputs
  + Once an input is entered, there is no way to change it
* To obtain results for a different set of input data, user must wait for command menu to be displayed again
  + At that point, the same command and all of the other inputs must be re-entered
* User can enter an unrecognized command

**The GUI-Based Version:**

* Uses a window that contains various components
  + Called **window objects** or **widgets**
* Solves problems of terminal-based version

**Coding simple GUI-based programs:**

* Define a new class to represent the main window
* Instantiate the classes of window objects needed for this application (e.g., labels, command buttons)
* Position these components in the window
* Instantiate the data model and provide for the display of any default data in the window objects
* Register controller methods with each window object in which a relevant event might occur
* Define these controller methods
* Define a **main** that launches the GUI

**Other useful GUI resources:**

* There are many libraries and toolkits of GUI components available to the Python programmer
  + For example, **Tkinter** and **tkMessageBox** 
    - Standard Python modules
* **Tkinter** includes classes for windows and numerous types of window objects
* **tkMessageBox** includes functions for several standard pop-up dialog boxes
* Layout of GUI components can be specified in more detail
  + Groups of components can be nested in panes
* Paragraphs can be displayed in scrolling text boxes
* Lists of information can be presented for selection in scrolling list boxes and drop-down menus
* Color, size, and style of text and of some GUI components can be adjusted
* GUI-based programs can be configured to respond to various keyboard events and mouse events

**GUI Programming: Graphical User Interfaces:**

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..
* **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/)

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

**Using the tkinter Module:** 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 top = Tkinter.Tk()

# Code to add widgets will go here... top.mainloop()

**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 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 message boxes in your applications. |

**Display text with Label Widgets:** A Label is a Tkinter Widget class, which is used to display text or an image. The label is a widget that the user just views but not interact with. There is hardly any book or introduction into a programming language, which doesn't start with the "Hello World" example.

The following Python script uses Tkinter to create a window with the text "Hello Tkinter". You can use the Python interpretor to type this script line after line, or you can save it in a file, for example, "hello.py":

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. |

import tkinter as tk

# if you are still working under a Python 2 version,

# comment out the previous line and uncomment the following line

# import Tkinter as tk

root = tk.Tk()

w = tk.Label(root, text="Hello Tkinter!")

w.pack()

root.mainloop()

**Output:**

Hello Tkinter Windows

**Organizing Widgets with Frames:** Python Tkinter Frame widget is used to organize the group of widgets. It acts like a container which can be used to hold the other widgets. The rectangular areas of the screen are used to organize the widgets to the python application.

The syntax to use the Frame widget is given below.

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

from tkinter import \*

top = Tk()

top.geometry("140x100")

frame = Frame(top)

frame.pack()

leftframe = Frame(top)

leftframe.pack(side = LEFT)

rightframe = Frame(top)

rightframe.pack(side = RIGHT)

btn1 = Button(frame, text="Submit", fg="red",activebackground = "red")

btn1.pack(side = LEFT)

btn2 = Button(frame, text="Remove", fg="brown", activebackground = "brown")

btn2.pack(side = RIGHT)

btn3 = Button(rightframe, text="Add", fg="blue", activebackground = "blue")

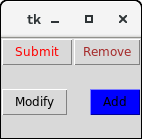
btn3.pack(side = LEFT)

btn4 = Button(leftframe, text="Modify", fg="black", activebackground = "white")

btn4.pack(side = RIGHT)

top.mainloop()

**Output:**



**Button Widgets and Info Dialog Boxes:** The button widget is used to add various types of buttons to the python application. Python allows us to configure the look of the button according to our requirements. Various options can be set or reset depending upon the requirements. We can also associate a method or function with a button which is called when the button is pressed.

## 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

**from** tkinter **import** \*

**from** tkinter **import** messagebox

top = Tk()

top.geometry("200x100")

**def** fun():

    messagebox.showinfo("Hello", "Red Button clicked")

b1 = Button(top,text = "Red",command = fun,activeforeground = "red",activebackground = "pink",pady=10)

b2 = Button(top, text = "Blue",activeforeground = "blue",activebackground = "pink",pady=10)

b3 = Button(top, text = "Green",activeforeground = "green",activebackground = "pink",pady = 10)

b4 = Button(top, text = "Yellow",activeforeground = "yellow",activebackground = "pink",pady = 10)

b1.pack(side = LEFT)

b2.pack(side = RIGHT)

b3.pack(side = TOP)

b4.pack(side = BOTTOM)

top.mainloop()

**Output:**

**  
**

**Getting Input with Entry Widget:** The Entry widget is used to provde the single line text-box to the user to accept a value from the user. We can use the Entry widget to accept the text strings from the user. It can only be used for one line of text from the user. For multiple lines of text, we must use the text 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:**

from tkinter import \*

top = Tk()

top.geometry("400x250")

name = Label(top, text = "Name").place(x = 30,y = 50)

email = Label(top, text = "Email").place(x = 30, y = 90)

password = Label(top, text = "Password").place(x = 30, y = 130)

sbmitbtn = Button(top, text = "Submit",activebackground = "pink", activeforeground = "blue").place(x = 30, y = 170)

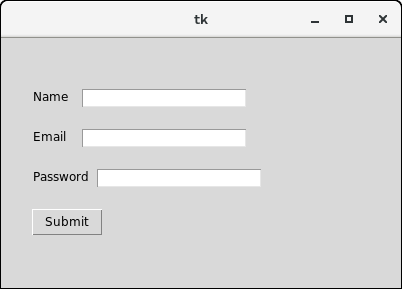
e1 = Entry(top).place(x = 80, y = 50)

e2 = Entry(top).place(x = 80, y = 90)

e3 = Entry(top).place(x = 95, y = 130)

top.mainloop()

**Output:**



**Using Labels as Output Fields**

from Tkinter import \*

master = Tk()

master.geometry('200x90')

master.title('Input Test')

def UserName():

global usrE

global usrN

global lbl

usrE1 = usrE.get()

usrN2 = usrN.get()

InputExcept = usrE1 + " " + usrN2

print InputExcept

lbl.config(text='User expected:'+InputExcept)

usrE = Entry(master, relief=SUNKEN)

usrE.pack()

usrN = Entry(master, relief=SUNKEN)

usrN.pack()

Btn1 = Button(master, text="Input", command=UserName)

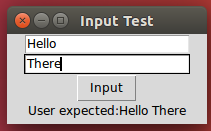
Btn1.pack()

lbl = Label(master)

lbl.pack()

master.mainloop()

**Output:**



**Radio Buttons:** The Radiobutton widget is used to implement one-of-many selection in the python application. It shows multiple choices to the user out of which, the user can select only one out of them. We can associate different methods with each of the radiobutton.

We can display the multiple line text or images on the radiobuttons. To keep track the user's selection the radiobutton, it is associated with a single variable. Each button displays a single value for that particular variable.

## 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:**

**from** tkinter **import** \*

**def** selection():

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

   label.config(text = selection)

top = Tk()

top.geometry("300x150")

radio = IntVar()

lbl = Label(text = "Favourite programming language:")

lbl.pack()

R1 = Radiobutton(top, text="C", variable=radio, value=1, command=selection)

R1.pack( anchor = W )

R2 = Radiobutton(top, text="C++", variable=radio, value=2, command=selection)

R2.pack( anchor = W )

R3 = Radiobutton(top, text="Java", variable=radio, value=3, command=selection)

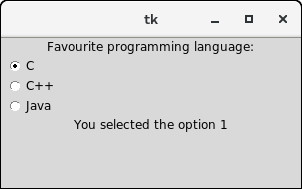
R3.pack( anchor = W)

label = Label(top)

label.pack()

top.mainloop()

**Output:**

****

**Check Buttons:** The Checkbutton is used to track the user's choices provided to the application. In other words, we can say that Checkbutton is used to implement the on/off selections. The Checkbutton can contain the text or images. The Checkbutton is mostly used to provide many choices to the user among which, the user needs to choose the one. It generally implements many of many selections.

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

**from** tkinter **import** \*

top = Tk()

top.geometry("200x200")

checkvar1 = IntVar()

checkvar2 = IntVar()

checkvar3 = IntVar()

chkbtn1 = Checkbutton(top, text = "C", variable = checkvar1, onvalue = 1, offvalue = 0, height = 2, width = 10)

chkbtn2 = Checkbutton(top, text = "C++", variable = checkvar2, onvalue = 1, offvalue = 0, height = 2, width = 10)

chkbtn3 = Checkbutton(top, text = "Java", variable = checkvar3, onvalue = 1, offvalue = 0, height = 2, width = 10)

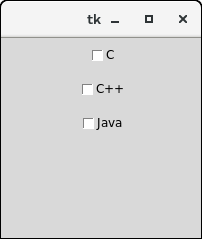
chkbtn1.pack()

chkbtn2.pack()

chkbtn3.pack()

top.mainloop()

**Output:**

****

## 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.

* T[he *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.

**T**[**he *pack()* Method**](https://www.tutorialspoint.com/python/tk_pack.htm)**:**

**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.

[**The *grid()* Method**](https://www.tutorialspoint.com/python/tk_grid.htm)

## 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.

## [The *place()* Method](https://www.tutorialspoint.com/python/tk_place.htm)

## 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.

**Simple Graphics and Image Processing:**

**Overview of Turtle Graphics:**

* Graphics is the discipline that underlies the representation and display of geometric shapes in two and three-dimensional space.
* A Turtle graphics library provides an enjoyable and easy way to draw shapes in a window and gives you an opportunity to run several functions with an object.
* Turtle graphics were originally developed as part of the children‟s programming language called Logo, created by Seymour Papert and his colleagues at MIT in the late 1960s.
* Imagine a turtle crawling on a piece of paper with a pen tied to its tail.
* Commands direct the turtle as it moves across the paper and tells it to lift or lower its tail, turn some number of degrees left or right and move a specified distance.
* Whenever the tail is down, the pen drags along the paper, leaving a trail.
* In the context of computer, of course, the sheet of paper is a window on a display screen and the turtle is an invisible pen point.
* At any given moment of time, the turtle coordinates. The position is specified with (x, y) coordinates.
* The coordinate system for turtle graphics is the standard Cartesian system, with the origin (0, 0) at the centre of a window. The turtle‟s initial position is the origin, which is also called the home.

**Turtle Operations:**

Turtle is an object; its operations are also defined as methods. In the below table the list of methods of Turtle class.

|  |  |
| --- | --- |
| **Turtle Methods** | **WHAT IT DOES** |
| home | Moves the turtle to the origin – coordinates (0, 0) – and set its  heading to its start-orientation. |
| fd | forward | Moves the turtle forward for a specified distance, in the direction  where the turtle is headed. |
| bk | backward | Moves the turtle backward for a specified distance, in the  direction where the turtle is headed. Do not change the turtle‟s heading. |
| right | rt | Turns the turtle right by angle units. Units are by default degrees,  but can be set via the degrees ( ) and radians ( ) functions. |
| left | lt | Turns the turtle left by angle units. Units are by default degrees,  but can be set via the degrees ( ) and radians ( ) functions. |
| setx | Set the turtle‟s first coordinate to x, leaves the second coordinate  unchanged. |
| sety | Set the turtle‟s second coordinate to y, leaves the first coordinate  unchanged. |
| goto | Moves the turtle to an absolute position. If the pen is down,  draws a line. Do not change the turtle‟s orientation. |
| degrees | Set the angle measurement unit to radians. Equivalent to degrees  (2 \* math.pi ) |
| radians | Set the angle measurement unit, i.e., set the number of degrees  for a full circle. The default value is 3600. |
| seth | Sets the orientation of the turtle to to\_angle. |

**Turtle Object:**

**t=Turtle( )** creates a new turtle object and open sits window. The window’s drawing area is 200 pixels wide and 200 pixels high.

**t=Turtle( width, height )** creates a new turtle object and open sits window. The window’s drawing area has given width and height.

**Two dimensional Shapes:**

**Example-1:** Write a program to draw square.

import turtle

turtle.bgcolor('orange')

turtle.pensize(8)

turtle.forward(100)

turtle.left(90)

turtle.forward(100)

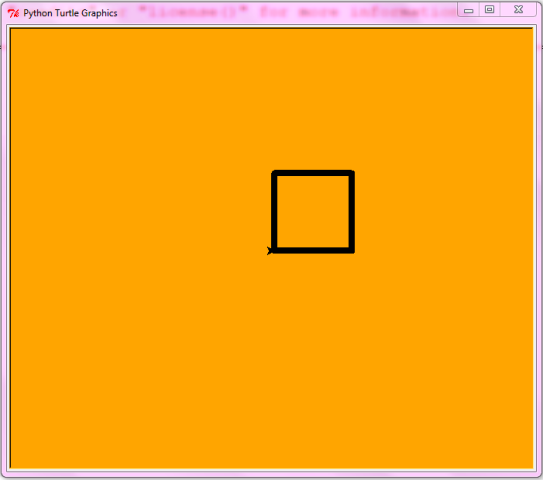
turtle.left(90)

turtle.forward(100)

turtle.left(90)

turtle.forward(100)

turtle.left(90)



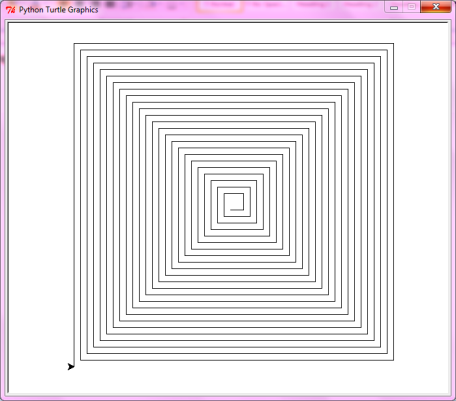
**Example-2:**

import turtle

for i in range(20,500,5):

turtle.forward(i)

turtle.left(90)



**Example-3:**

import turtle

c=["red","green","blue"]

i=0

turtle.pensize(5)

for angle in range(0,360,30):

if i>2:

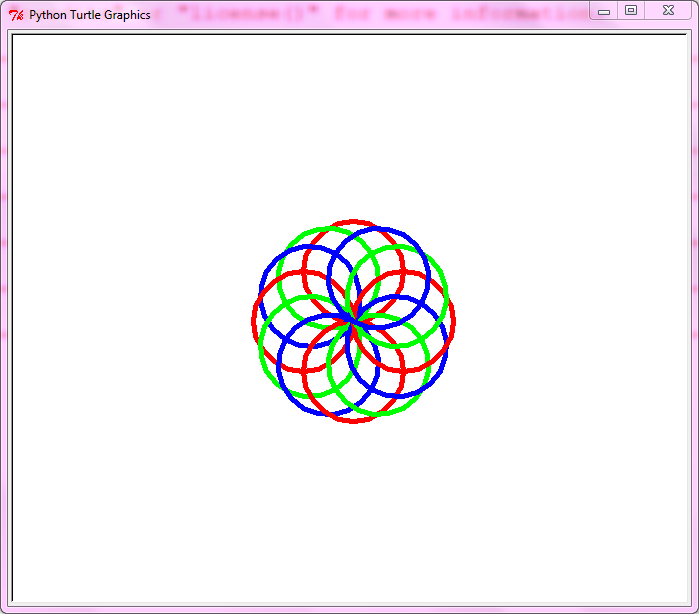
i=0

turtle.color(c[i])

turtle.seth(angle)

turtle.circle(50)

i=i+1

****

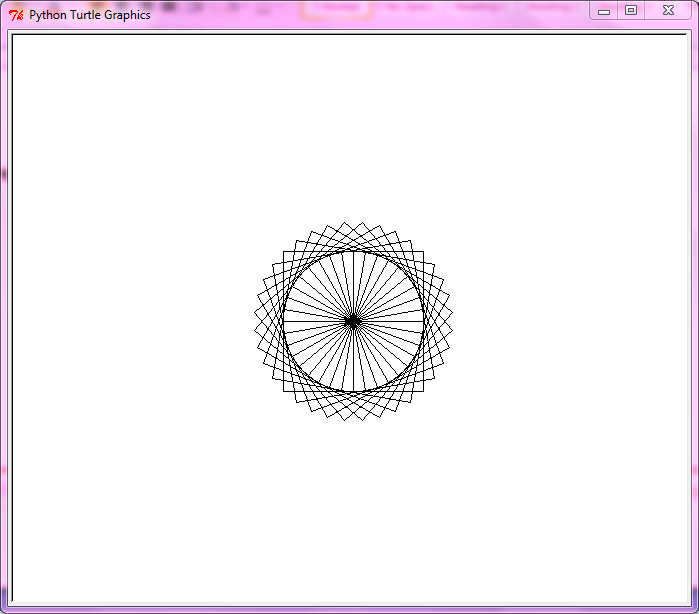
**Example-4:**

import turtle

for i in range(36):

for j in range(4):

turtle.forward(70)



**Colors and RBG System:**

* **Tkinter** module supports the RGB
  + Values expressed in hex notation (e.g., **#ff0000**)
  + Some commonly used colors have been defined as string values (e.g., **"white"**, **"black"**, **"red"**)
* For most components, you can set two color attributes:
  + A foreground color (**fg**) and a background color (**bg**)

****

****

**Image Processing:** It has never as easy as it is nowadays to take a picture. All it usually needs is a mobile phone. These are the bare essentials to shoot and to view an image. Taking a photograph is free, if we don't take the costs for the mobile phone into considerations. Just a generation ago, hobby artists and real artists needed special and often expensive and the costs per picture were far from being free. We take pictures to preserve great moments in time. Pickled memories ready to be "opened" in the future at will.

Similar to pickling things, we have to pay attention to the right preservatives. Of course, mobile phone also provide us with a range of image processing software, but as soon as we need to manipulate a huge quantity of photographs we need other tools. This is when programming and Python comes into play. Python and its modules like Numpy, Scipy, Matplotlib and other special modules provide the optimal functionality to be able to cope with the flood of pictures.

To provide you with the necessary knowledge this chapter of our Python tutorial deals with basic image processing and manipulation. For this purpose we use the modules NumPy, Matplotlib and SciPy.

We start with the scipy package misc. The helpfile says that scipy.misc contains "various utilities that don't have another home".

# the following line is only necessary in Python notebook:

%matplotlib inline

from scipy import misc

ascent = misc.ascent()

import matplotlib.pyplot as plt

plt.gray()

plt.imshow(ascent)

plt.show()

Additionally to the image, we can see the axis with the ticks. This may be very interesting, if you need some orientations about the size and the pixel position, but in most cases, you want to see the image without this information. We can get rid of the ticks and the axis by adding the command plt.axis("off").