GUI PROGRAMMING USING TKINTER

Cuauhtémoc Carbajal

ITESM CEM

April 17, 2013

Agenda

- Introduction
- Tkinter and Python Programming
- Tkinter Examples

INTRODUCTION

Introduction

- In this lecture, we will give you a brief introduction to the subject of graphical user interface (GUI) programming.
- We cannot show you everything about GUI application development in just one lecture, but we will give you a very solid introduction to it.
- The primary GUI toolkit we will be using is Tk, Python's default GUI. We'll access Tk from its Python interface called Tkinter (short for "Tk interface").
- Tk is not the latest and greatest, nor does it have the most robust set of GUI building blocks, but it is fairly simple to use, and with it, you can build GUIs that run on most platforms.
- Once you have completed this lecture, you will have the skills to build more complex applications and/or move to a more advanced toolkit. Python has bindings or adapters to most of the current major toolkits, including commercial systems.

What Are Tcl, Tk, and Tkinter?

- Tkinter is Python's default GUI library. It is based on the Tk toolkit, originally designed for the Tool Command Language (Tcl). Due to Tk's popularity, it has been ported to a variety of other scripting languages, including Perl (Perl/Tk), Ruby (Ruby/Tk), and Python (Tkinter).
- The combination of Tk's GUI development portability and flexibility along with the simplicity of a scripting language integrated with the power of systems language gives you the tools to rapidly design and implement a wide variety of commercial-quality GUI applications.
- Python, along with Tkinter, provides a fast and exciting way to build useful applications that would have taken much longer if you had to program directly in C/C++ with the native windowing system's libraries.
- Once you have designed the application and the look and feel that goes along with your program, you will use basic building blocks known as widgets to piece together the desired.
- Once you get Tkinter up on your system, it will take less than 15 minutes to get your first GUI application running.

Getting Tkinter Installed and Working

- Tkinter is not necessarily turned on by default on your system. You can determine whether Tkinter is available for your Python interpreter by attempting to import the Tkinter module (in Python 1 and 2; renamed to tkinter in Python 3). If Tkinter is available, then no errors occur, as demonstrated in the following:

```
>>> import tkinter
>>>
```

- If your Python interpreter was not compiled with Tkinter enabled, the module import fails. You might need to recompile your Python interpreter to gain access to Tkinter. This usually involves editing the Modules/Setup file and then enabling all the correct settings to compile your Python interpreter with hooks to Tkinter, or choosing to have Tk installed on your system.

Getting Tkinter Installed and Working on the RPi

- Type the following line into a terminal window:
  - `sudo apt-get install python-tk`
- Open a Python Shell:
  - `idle3`
- Import the Tkinter module:

```
>>> import tkinter
```

The Tkinter Module: Adding Tk to your Applications

- Do you need to do to have Tkinter as part of your application?
- First, it is not necessary to have an application already. You can create a pure GUI if you want, but it probably isn't too useful without some underlying software that does something interesting.
- There are basically five main steps that are required to get your GUI up and running:
  1. Import the Tkinter module (or from Tkinter import *).
  2. Create a top-level windowing object that contains your entire GUI application.
  3. Build all your GUI components (and functionality) on top (or within) of your top-level windowing object.
  4. Connect these GUI components to the underlying application code.
  5. Enter the main event loop.

Introduction to GUI Programming

- Before going to the examples, we will give you a brief introduction to GUI application development. This will provide you with some of the general background you need to move forward.
- Setting up a GUI application is similar to how an artist produces a painting. Conventionally, there is a single canvas onto which the artist must put all the work. Here’s how it works: you start with a clean slate, a “top-level” windowing object on which you build the rest of your components.
- Think of it as a foundation to a house or the easel for an artist. In other words, you have to pour the concrete or set up your easel before putting together the actual structure or canvas on top of it. In Tkinter, this foundation is known as the top-level window object.

Windows and Widgets

- In GUI programming, a top-level root windowing object contains all of the little windowing objects that will be part of your complete GUI application. These can be text labels, buttons, list boxes, etc. These individual little GUI components are known as widgets.
- So when we say create a top-level window, we just mean that you need a place where you put all your widgets. In Python, this would typically look like this line:
  - `top = Tkinter.Tk()` # or just Tk() with "from Tkinter import *"
- The object returned by **`Tkinter.Tk()`** is usually referred to as the **root** window; hence, the reason why some applications use **root** rather than top to indicate as such. Top-level windows are those that show up stand-alone as part of your application. You can have more than one top-level window for your GUI, but only one of them should be your root window.

Windows and Widgets (2)

- You can choose to completely design all your widgets first, and then add the real functionality, or do a little of this and a little of that along the way.
- Widgets can be stand-alone or be containers. If a widget contains other widgets, it is considered the parent of those widgets. Accordingly, if a widget is contained in another widget, it's considered a child of the parent, the parent being the next immediate enclosing container widget.
- Usually, widgets have some associated behaviors, such as when a button is pressed, or text is filled into a text field. These types of user behaviors are called **events**, and the GUI's response to such events are known as **callbacks**.

Event-Driven Processing

- Events can include the actual button press (and release), mouse movement, hitting the Return or Enter key, etc. The entire system of events that occurs from the beginning until the end of a GUI application is what drives it. This is known as **event-driven processing**.
- One example of an event with a callback is a simple mouse move. Suppose that the mouse pointer is sitting somewhere on top of your GUI application. If you move the mouse to another part of your application, something has to cause the movement of the mouse to be replicated by the cursor on your screen so that it looks as if it is moving according to the motion of your hand. These are mouse move events that the system must process portray your cursor moving across the window. When you release the mouse, there are no more events to process, so everything just remains idle on the screen again.

Event-Driven Processing (2)

- The event-driven processing nature of GUIs fits right in with client/server architecture.
  - When you start a GUI application, it must perform some setup procedures to prepare for the core execution, just as how a network server must allocate a socket and bind it to a local address.
  - The GUI application must establish all the GUI components, then draw (a.k.a. render or paint) them to the screen. This is the responsibility of the **geometry manager** (more about this in a moment). When the geometry manager has completed arranging all of the widgets, including the top-level window, GUI applications enter their server-like infinite loop.
  - This loop runs forever waiting for GUI events, processing them, and then going to wait for more events to process.

Geometry Managers

Geometry managers allow us to organize widgets inside of a container

Place geometry manager
http://effbot.org/tkinterbook/place.htm
Pack geometry manager
http://effbot.org/tkinterbook/pack.htm
Grid geometry manager
http://effbot.org/tkinterbook/grid.htm

Geometry Managers

- Tk has three geometry managers that help with positioning your widgetset:
  - **Placer:** You provide the size of the widgets and locations to place them; the manager then places them for you. The problem is that you have to do this with all the widgets, burdening the developer with coding that should otherwise take place automatically.
  - **Packer:** it packs widgets into the correct places (namely the containing parent widgets, based on your instruction), and for every succeeding widget, it looks for any remaining "real estate" into which to pack the next one. The process is similar to how you would pack elements into a suitcase when traveling.
  - **Grid:** It is used to specify GUI widget placement, based on grid coordinates. The Grid will render each object in the GUI in their grid position.
- We will stick with the Packer.

Packer

- Once the Packer has determined the sizes and alignments of all your widgets, it will then place them on the screen for you.
- When all the widgets are in place, we instruct the application to enter the aforementioned infinite main loop. In Tkinter, the code that does this is:
  - `Tkinter.mainloop()`
- This is normally the last piece of sequential code your program runs.
- When the main loop is entered, the GUI takes over execution from there.
- All other actions are handled via callbacks, even exiting your application. When you select the File menu and then click the Exit menu option or close the window directly, a callback must be invoked to end your GUI application.

Top-Level Window: Tkinter.Tk()

- We mentioned earlier that all main widgets are built on the top-level window object. This object is created by the Tk class in Tkinter and is instantiated as follows:

```
>>> import Tkinter
>>> top = Tkinter.Tk()
```

- Within this window, you place individual widgets or multiple-component pieces together to form your GUI.

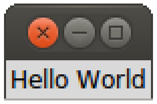
Hello World

options

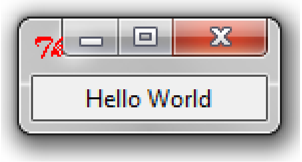
```
from Tkinter import Label                          # get a widget
widget = Label(None, text='Hello World')           # make a Label
widget.pack()                                      # arrange it in its parent
widget.mainloop()                                  # start the event loop
```

parent widget

1. Load a widget class from Tkinter
2. Make an instance of it
   (repeat 1 and 2 as needed)
3. Arrange the widget in its parent widget
4. Enter the event loop



Hello World



Hello World

python3

```
  from tkinter import Label
```

python

```
  from Tkinter import Label
```

Tkinter Events and Binding

<Button-1>	- left mouse button	Mouse events
<Button-2>	- middle mouse button (on 3 button mouse)	
<Button-3>	- rightmost mouse button	
<B1-Motion>	- mouse moved with left button depressed	
<ButtonRelease-1>	- left button released	
<Double-Button-1>	- double click on button 1	
<Enter>	- mouse pointer entered widget	
<Leave>	- mouse pointer left the widget	
<FocusIn>	- Keyboard focus moved to a widget	Keyboard events
<FocusOut>	- Keyboard focus moved to another widget	
<Return>	- Enter key depressed	
<Key>	- A key was depressed	
<Shift-Up>	- Up arrow while holding Shift key	
<Configure>	- widget changed size or location	

http://effbot.org/tkinterbook/tkinter-events-and-bindings.htm

Event Handling

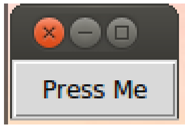
- Event sources (widgets) can specify their handlers
  - command handlers
  - callbacks

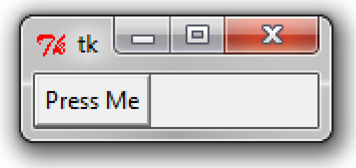
Command Handlers

use the 'command=' keyword followed by the command you want executed

ex:

```
from Tkinter import *
root = Tk()
Button (root, text='Press Me', command=root.quit).pack(side=LEFT)
root.mainloop()
```

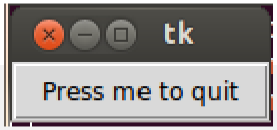




Callbacks

- A callback is the name of the function that is to be run in response of an event
- Callbacks can be defined as a free standing **function** in our program or as a **class** member.

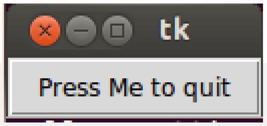
ex.



```
from Tkinter import *

def quit():
    print 'Hello, getting out of here'
    import sys; sys.exit()

widget = Button(None, text='Press me to quit' , command=quit)
widget.pack()
widget.mainloop()
```

Bound Method Callbacks

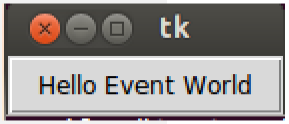
Let’s make a Hello Class and use it:



```
from Tkinter import *
class HelloClass:
     # create the window in the class constructor
    def __init__(self):
        widget = Button(None, text='Press Me to quit', command=self.quit)
        widget.pack()
    def quit(self):
         print 'leaving now'
         import sys ; sys.exit()

HelloClass()      # create a HelloClass object
mainloop()
```

Binding Events

```
from Tkinter import *

def  hello(event):
    print 'Double click to exit'

def  quit(event):
    print 'caught a double click, leaving'
    import sys ; sys.exit()

widget = Button(None, text='Hello Event World')
widget.pack()
widget.bind('<Button-1>', hello)
widget.bind('<Double-1>' , quit)
widget.mainloop()
```



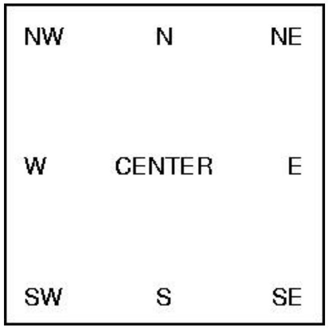
TKINTER WIDGETS

Tk Widgets

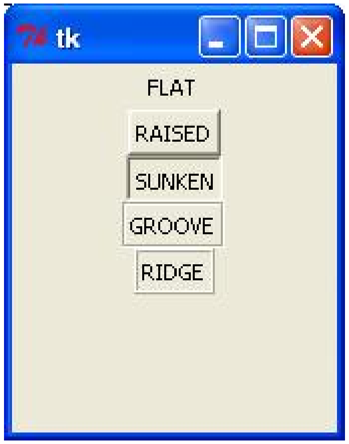
Widget	Description
Button	Similar to a Label but provides additional functionality for mouse-overs, presses, and releases, as well as keyboard activity/events
Canvas	Provides ability to draw shapes (lines, ovals, polygons, rectangles); can contain images or bitmaps
Checkbutton	Set of boxes, of which any number can be "checked"
Entry	Single-line text field with which to collect keyboard input
Frame	Pure container for other widgets
Label	Used to contain text or images
LabelFrame	Combo of a label and a frame but with extra label attributes
Listbox	Presents the user with a list of choices from which to choose
Menu	Actual list of choices "hanging" from a Menubutton from which the user can choose
Menubutton	Provides infrastructure to contain menus (pulldown, cascading, etc.)
Message	Similar to a Label, but displays multiline text
PanedWindow	A container widget with which you can control other widgets placed within it
Radiobutton	Set of buttons, of which only one can be "pressed"
Scale	Linear "slider" widget providing an exact value at current setting; with defined starting and ending values
Scrollbar	Provides scrolling functionality to supporting widgets, for example, Text, Canvas, Listbox, and Entry
Spinbox	Combination of an entry with a button letting you adjust its value
Text	Multiline text field with which to collect (or display) text from user
Toplevel	Similar to a Frame, but provides a separate window container

Standard attributes

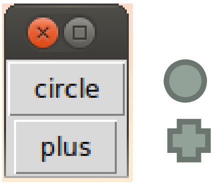
- Dimensions
- Colors
- Fonts
- Anchors
  - used to define where text is positioned relative to a reference point.



- Relief styles
  - refers to certain simulated 3-D effects around the outside of the widget.



- Bitmaps
  - used to display a bitmap type: "error“, "gray75“, "gray50“. "gray25“, "gray12“, "hourglass“, "info“, "questhead“, "question“, "warning"
- Cursors



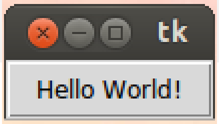
Button

Button(master=None, **options) (class) [#] A command button. *master* Parent widget. ***options* Widget options. See the description of the **config** method for a list of available options.

- Buttons can contain text or images, and you can associate a Python function or method with each button. When the button is pressed, Tkinter automatically calls that function or method.
- The button can only display text in a single font, but the text may span more than one line. In addition, one of the characters can be underlined, for example to mark a keyboard shortcut. By default, the Tab key can be used to move to a button widget.

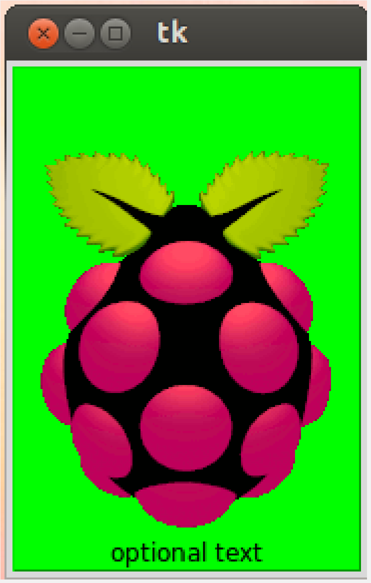
Button Widget Demo

```
import Tkinter

top = Tkinter.Tk()
quit = Tkinter.Button(top, text='Hello World!',
        command=top.quit)
quit.pack()
Tkinter.mainloop()
```



tkhello2.py

```
import Tkinter as tk
button_flag = True
def click():
    """
    respond to the button click
    """
    global button_flag
    # toggle button colors as a test
    if button_flag:
        button1.config(bg="white")
        button_flag = False
    else:
        button1.config(bg="green")
        button_flag = True
root = tk.Tk()
# create a frame and pack it
frame1 = tk.Frame(root)
frame1.pack(side=tk.TOP, fill=tk.X)
# pick a (small) image file you have in the working directory ...
photo1 = tk.PhotoImage(file="rpi.gif")
# create the image button, image is above (top) the optional text
button1 = tk.Button(frame1, compound=tk.TOP, width=148, height=240, image=photo1,
    text="optional text", bg='green', command=click)
button1.pack(side=tk.LEFT, padx=2, pady=2)
# save the button's image from garbage collection (needed?)
button1.image = photo1
# start the event loop
root.mainloop()
```



"padx" puts a bit of extra space to the left and right of the widget, while "pady" adds extra space top and bottom.

button-image.py

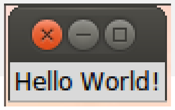
Label

Label(master=None, **options) (class) [#] Display a single line of text, or an image.
master Parent widget. ***options* Widget options. See the description of the **config** method for a list of available options.

- Label is used to display a text or image on the screen. The label can only display text in a single font, but the text may span more than one line.
- Can be used as a label for another widget (like an Entry Widget) or as an output only widget on the screen.
  - associate a text variable with the label and whenever the text variable changes value the label field will change. Use the set() method to update the Text variable (or get() to retrieve is current value).
  - text may be anchored **N**, **NE**, **E**, **SE**, **S**, **SW**, **W**, **NW**, or **CENTER**. Default is CENTER

Label Widget Demo

```
import Tkinter

top = Tkinter.Tk()
label = Tkinter.Label(top, text='Hello World!')
label.pack()
Tkinter.mainloop()
```



Hello World!

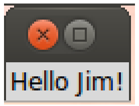
tkhello1.py

Widget Configuration

- Configuration can be specified at creation time or changed as the program is running by using the config method

- ex.

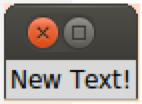
```
import Tkinter
top = Tkinter.Tk()
w = Tkinter.Label(top, text="Hello, world!")
w.pack()
w.config(text="Hello Jim!")
Tkinter.mainloop()
```



Hello Jim!

Updating the text variable

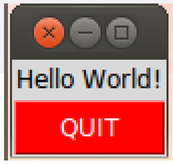
```
from Tkinter import *

root = Tk()

# To create a Tkinter variable, call the corresponding constructor
v = StringVar()
label = Label(root, textvariable = v).pack()
v.set("New Text!")

print v.get()

root.mainloop()
```



New Text!

Label and Button Widget Demo

```
import Tkinter
top = Tkinter.Tk()

hello = Tkinter.Label(top, text='Hello World!')
hello.pack()

quit = Tkinter.Button(top, text='QUIT', command=top.quit,
       bg='red', fg='white')
quit.pack(fill=Tkinter.X, expand=1)

Tkinter.mainloop()
```

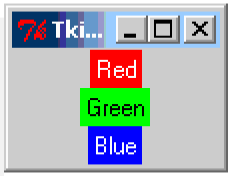


You can use the *fill=X* and *expand* options to make a widget as wide as the parent widget, even if the user resizes the window.

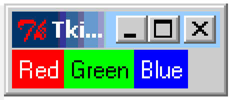
tkhello3.py

Pack Geometry Manager

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

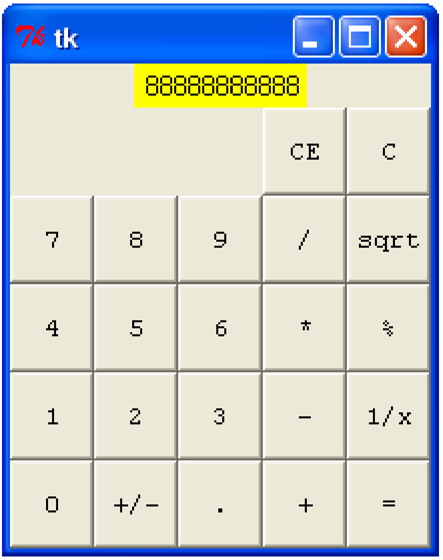


```
from Tkinter import *
root = Tk()
w = Label(root, text="Red", bg="red", fg="white").pack(side=LEFT)
w = Label(root, text="Green", bg="green", fg="black").pack(side=LEFT)
w = Label(root, text="Blue", bg="blue", fg="white").pack(side=LEFT)
mainloop()
```



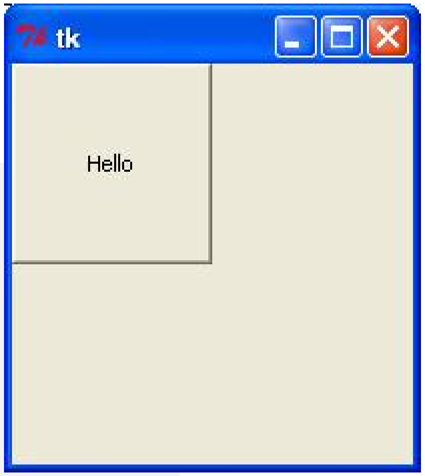
Grid Geometry Manager

```
Button(win, font=('courier',10), text='CE',width=4,height=2, command=lambda: click('CE')).grid(row=0,column=4)
Button(win, font=('courier',10), text='C',width=4,height=2, command=lambda: click('C')).grid(row=0,column=5)
Button(win, font=('courier',10), text='7',width=4,height=2, command=lambda: click('7')).grid(row=1,column=1)
Button(win, font=('courier',10), text='8',width=4,height=2, command=lambda: click('8')).grid(row=1,column=2)
Button(win, font=('courier',10), text='9',width=4,height=2, command=lambda: click('9')).grid(row=1,column=3)
Button(win, font=('courier',10), text='4',width=4,height=2, command=lambda: click('4')).grid(row=2,column=1)
Button(win, font=('courier',10), text='5',width=4,height=2, command=lambda: click('5')).grid(row=2,column=2)
Button(win, font=('courier',10), text='6',width=4,height=2, command=lambda: click('6')).grid(row=2,column=3)
Button(win, font=('courier',10), text='1',width=4,height=2, command=lambda: click('1')).grid(row=3,column=1)
Button(win, font=('courier',10), text='2',width=4,height=2, command=lambda: click('2')).grid(row=3,column=2)
Button(win, font=('courier',10), text='3',width=4,height=2, command=lambda: click('3')).grid(row=3,column=3)
Button(win, font=('courier',10), text='0',width=4,height=2, command=lambda: click('0')).grid(row=4,column=1)
Button(win, font=('courier',10), text='+/-',width=4,height=2, command=lambda: click('+/-')).grid(row=4,column=2)
Button(win, font=('courier',10), text='.',width=4,height=2, command=lambda: click('.')).grid(row=4,column=3)
```



Place Geometry Manager

- Simplest of the three geometry managers, do not use for ordinary windows as it is too much work. Instead use it for making more complex containers.
- Place is available for all widgets



```
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()
```

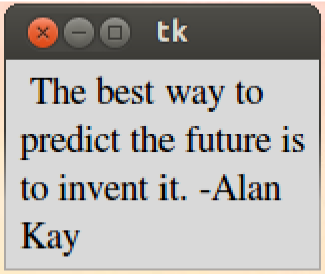
MORE WIDGETS

ENTRY , CHECKBUTTONS, RADIO BUTTONS, SCALE

Message

The Message widget is used to display text. This is a widget not a text window. Use for on screen instructions or to make a custom message window

```
from Tkinter import *
master =Tk()
msg = Message(master, text=' The best way to predict the future is to invent it. -Alan Kay')
msg.config(font=('times',14))
msg.pack()
mainloop()
```



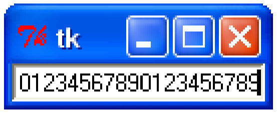
The best way to
predict the future is
to invent it. -Alan
Kay

options:

- anchor
- aspect
- background/bg
- borderwidth/bd
- cursor
- font
- foreground/fg
- highlightbackground
- highlightcolor
- highlightthickness
- justify
- padx
- pady
- relief
- takefocus
- text
- textvariable
- width

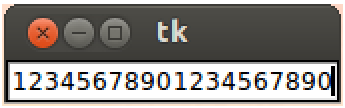
Entry

- used to enter or display a single line of text
  - To enter multiple lines of text, use the Text widget.



01234567890123456789

```
from Tkinter import *
master = Tk()
e = Entry(master)
e.pack()
mainloop()
```



12345678901234567890

options:

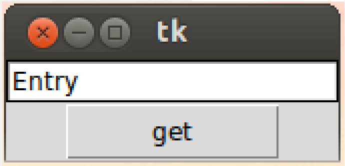
- anchor
- aspect
- background/bg
- borderwidth/bd
- cursor
- font
- foreground/fg
- highlightbackground
- highlightcolor
- highlightthickness
- relief
- takefocus
- text
- textvariable
- width
- validate
- validatecommand
- justify

methods:

- get()
- set()
- delete(first,last/END)
- insert(index)
- insert(index,string)
- icursor(index)
- index(index)

Entry

```
from Tkinter import *

master = Tk()

e = Entry(master)
e.pack()

def callback():
    print e.get()

b = Button(master, text="get", width=10, command=callback)
b.pack()
mainloop()
```



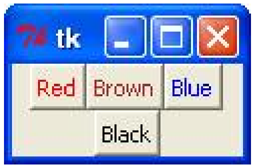
Frame Widget

- Frames are used as containers for other widgets. Widgets placed in a frame can use any of the geometry managers (but only one per frame)
- You can have multiple frames per window
- Options:

bg	background color
bd	border width (default=2 pixels)
cursor	cursor to use when hovering over the frame
height	vertical dimension of the frame
highlightbackground	color when frame doesn't have focus
highlightcolor	color when frame has focus
highlightthickness	thickness of the focus highlight
relief	FLAT, Raised, Sunken, GROOVE, RIDGE; default = FLAT
width	width of the frame

Frame Example

```
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()
```

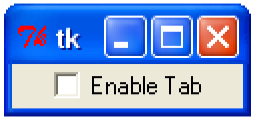


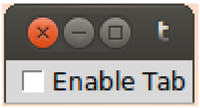
Checkbuttons and Radiobuttons

- Checkbuttons are used for multiple choice situations, i.e. choosing *m* of *n* possible options. This is done by assigning each checkbox a variable of its own.
- Radiobuttions are used for choosing one of n possible choices; i.e. a mutually exclusive single choice by giving each button a unique value of the same Tkinter variable.

Checkbutton

```
from Tkinter import *

def cb():
  print "variable is", var.get()

win = Tk()
var = IntVar()
c = Checkbutton(
   win, text="Enable Tab",
   variable=var,
   command= (lambda: cb()))
c.pack()

mainloop()
```





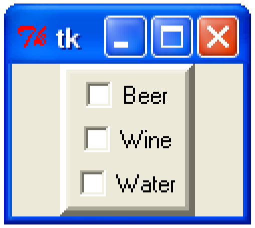
checkbutton.py

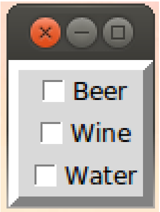
Checkbutton (2)

```
from Tkinter import *

def cb():
  print "beer is", var1.get()
  print "Wine is", var2.get()
  print "Water is", var3.get()

win = Tk()
f = Frame(relief=RAISED , borderwidth=5)
var1 = IntVar()
var2 = IntVar()
var3 = IntVar()

c1 = Checkbutton(
   f, text="Beer",
   variable=var1,
   command= (lambda: cb()))
c1.pack(side=TOP)
```

```
c2 = Checkbutton(
   f, text="Wine",
   variable=var2,
   command= (lambda: cb()))
c2.pack(side=TOP)

c3 = Checkbutton(
   f, text="Water",
   variable=var3,
   command= (lambda: cb()))
c3.pack(side=TOP)
f.pack()

mainloop()
```

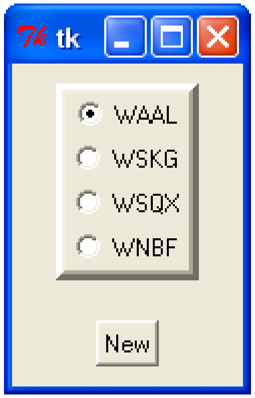
checkbutton2.py

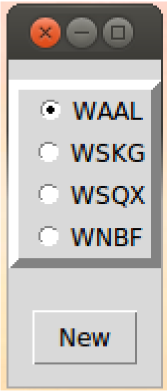




Radiobuttons

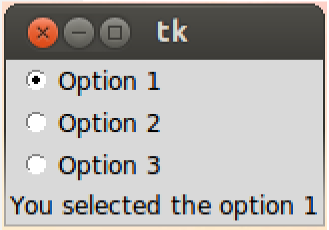
```
from Tkinter import *

def change():
   print 'Station = ' , var.get()

root = Tk()

stations = 'WAAL' , 'WSKG' , 'WSQX' , 'WNBF'

f = Frame(relief=RAISED , borderwidth=5)
var = StringVar()

for station in stations:
   radio = Radiobutton(f, text=station, variable=var ,value=station)
   radio.pack(side=TOP)

f.pack(pady=10)
Button(root,text='New' , command=(lambda: change())).pack(pady=10)
var.set('WAAL') #initalize the set of radio buttons

mainloop()
```





Radiobutton (2)

```
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()
```



● Option 1
○ Option 2
○ Option 3
You selected the option 1

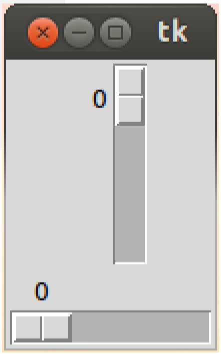
radiobutton2.py

Sliders

- A slider is a Tkinter object with which a user can set a value by moving an indicator. Sliders can be vertically or horizontally arranged. A slider is created with the Scale method().
- Using the Scale widget creates a graphical object, which allows the user to select a numerical value by moving a knob along a scale of a range of values. The minimum and maximum values can be set as parameters, as well as the resolution. We can also determine if we want the slider vertically or horizontally positioned. A Scale widget is a good alternative to an Entry widget, if the user is supposed to put in a number from a finite range, i.e. a bounded numerical value.

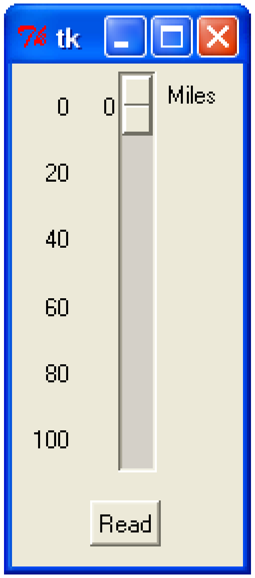
Slider: A Simple Example

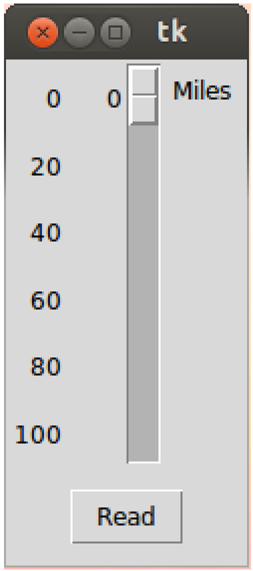
```
from Tkinter import *

master = Tk()
w = Scale(master, from_=0, to=42)
w.pack()
w = Scale(master, from_=0, to=200, orient=HORIZONTAL)
w.pack()

mainloop()
```

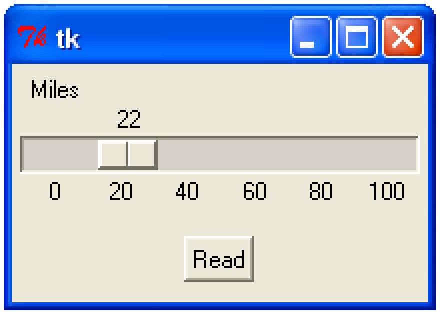


Scale/Sliders

```
from Tkinter import *

class SliderDemo(Frame):
  def __init__(self,parent=None):
    Frame.__init__(self,parent)
    self.pack()
    self.var = IntVar()
    Scale(self,label='Miles',
        command=self.onMove,
        variable = self.var,
        from_=0 , to=100 ,length=200,
        tickinterval=20).pack()
    Button(self , text='Read', command=self.readScale).pack(pady=10)

  def onMove(self, value):
    print 'onMove = ' , value

  def readScale(self):
    print 'readscale = ' , self.var.get()

if __name__ == '__main__' :
  SliderDemo().mainloop()
```

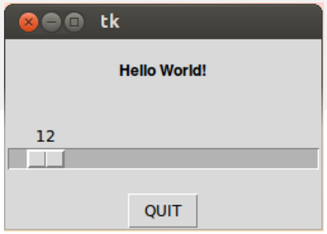






Label, Button, and Scale Demonstration

```
from Tkinter import *
def resize(ev=None):
  label.config(font='Helvetica -%d bold' % scale.get())
top = Tk()
top.geometry('250x150')
label = Label(top, text='Hello World!', \
      font='Helvetica -12 bold')
label.pack(fill=Y, expand=1)
scale = Scale(top, from_=10, to=40, orient=HORIZONTAL, \
      command=resize)
scale.set(12)
scale.pack(fill=X, expand=1)
quit = Button(top, text='QUIT', command=top.quit, \
      activeforeground='white', activebackground='red')
quit.pack()
mainloop()
```



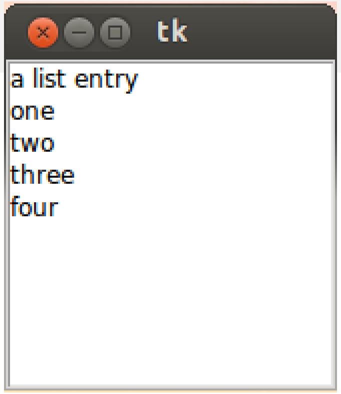
tkhello4.py

Listbox

- The Listbox widget is used to display a list of alternatives. The listbox can only contain text items, and all items must have the same font and color. Depending on the widget configuration, the user can choose one or more alternatives from the list.
- When to use the Listbox Widget
  - Listboxes are used to select from a group of textual items. Depending on how the listbox is configured, the user can select one or many items from that list.
- Patterns
  - When you first create the listbox, it is empty. The first thing to do is usually to insert one or more lines of text. The insert method takes an index and a string to insert. The index is usually an item number (0 for the first item in the list), but you can also use some special indexes, including ACTIVE, which refers to the “active” item (set when you click on an item, or by the arrow keys), and END, which is used to append items to the list.

Listbox example

```
from Tkinter import *
master = Tk()
listbox = Listbox(master)
listbox.pack()
listbox.insert(END, "a list entry")
for item in ["one", "two", "three", "four"]:
    listbox.insert(END, item)
mainloop()
```



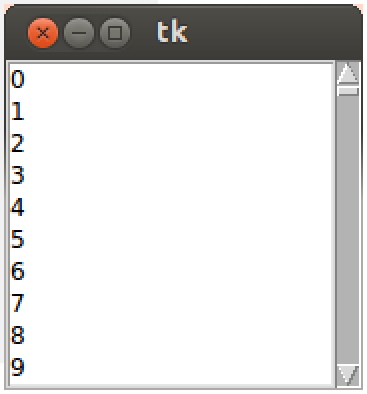
a list entry
one
two
three
four

Scrollbar

- This widget is used to implement scrolled listboxes, canvases, and text fields.
- Patterns
    - The Scrollbar widget is almost always used in conjunction with a Listbox, Canvas, or Text widget. Horizontal scrollbars can also be used with the Entry widget.
    - To connect a vertical scrollbar to such a widget, you have to do two things:
        - Set the widget’s yscrollcommand callbacks to the set method of the scrollbar.
        - Set the scrollbar’s command to the yview method of the widget.

Scrollbar Example

```
from Tkinter import *

master = Tk()

scrollbar = Scrollbar(master)
scrollbar.pack(side=RIGHT, fill=Y)

listbox = Listbox(master, yscrollcommand=scrollbar.set)
for i in range(1000):
    listbox.insert(END, str(i))
listbox.pack(side=LEFT, fill=BOTH)

scrollbar.config(command=listbox.yview)

mainloop()
```

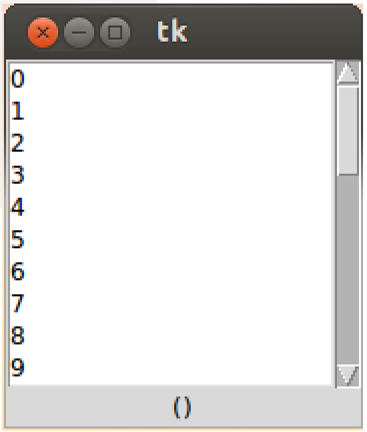


```
0
1
2
3
4
5
6
7
8
9
```

Get selected value from Listbox

```
import Tkinter

F1 = Tkinter.Frame()
s = Tkinter.Scrollbar(F1)
L = Tkinter.Listbox(F1)

s.pack(side=Tkinter.RIGHT, fill=Tkinter.Y)
L.pack(side=Tkinter.LEFT, fill=Tkinter.Y)

s['command'] = L.yview
L['yscrollcommand'] = s.set

for i in range(30):
   L.insert(Tkinter.END, str(i))

F1.pack(side=Tkinter.TOP)
```

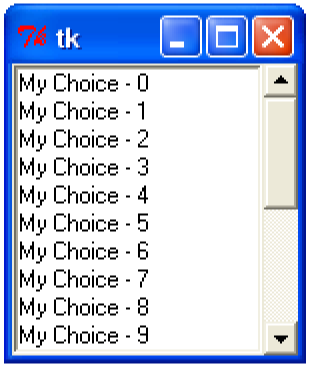
```
F2 = Tkinter.Frame()
lab = Tkinter.Label(F2)

def poll():
    lab.after(200, poll)
    sel = L.curselection()
    lab.config(text=str(sel))

lab.pack()
F2.pack(side=Tkinter.TOP)

poll()
Tkinter.mainloop()
```



```
0
1
2
3
4
5
6
7
8
9
()
```

Listboxes and Scrollbars

```
from Tkinter import *

class ScrolledList(Frame):
  def __init__(self,options,parent=None):
    Frame.__init__(self,parent)
    self.pack(expand=YES , fill=BOTH)
    self.makeWidgets(options)

  def handleList(self,event):
    index = self.list.curselection()
    label = self.list.get(index)
    self.runCommand(label)

  def makeWidgets(self , options):
    sbar = Scrollbar(self)
    list = Listbox(self,relief=SUNKEN)
    sbar.config(command=list.yview)
    list.config(yscrollcommand=sbar.set)
    sbar.pack(side=RIGHT , fill=Y)
    list.pack(side=LEFT , expand=YES , fill=BOTH)
    pos = 0
    for label in options:
      list.insert(pos,label)
      pos += 1
    #list.config(selectmode=SINGLE , setgrid = 1)
    list.bind('<Double-1>' , self.handleList)
    self.list = list

  def runCommand(self,selection):
    print 'You selected ' , selection

if __name__ == '__main__' :
  options = map((lambda x : 'My Choice - ' + str(x)) , range(20))
  ScrolledList(options).mainloop()
```



My Choice - 0
My Choice - 1
My Choice - 2
My Choice - 3
My Choice - 4
My Choice - 5
My Choice - 6
My Choice - 7
My Choice - 8
My Choice - 9

```
$ python listbox.py
You selected  My Choice - 12
You selected  My Choice - 15
You selected  My Choice - 8
You selected  My Choice - 10
```

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:
- 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.

Canvas (2)

The Canvas widget can support the following standard items:

- arc . Creates an arc item.
  - `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.
  - `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)`

Canvas (3)

```
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()
```