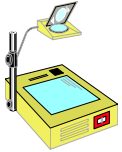




Python: Widgets



Learning Python, Lutz & Ascher

1

Widget appearance

```
from Tkinter import *
```



```
root = Tk()
labelfont = ('times', 20, 'bold', 'underline')
widget = Label(root, text = "Hello world");
widget.config(bg="blue", fg="yellow")
widget.config(font=labelfont)
```

```
widget.config(height=3, width=20)
widget.pack(expand=YES, fill=BOTH)
root.mainloop()
```

2

appearance

- bg is background, fg is foreground
- Font
 - Three-item tuple for font family, size and style
 - Font style can be normal, bold, roman, italic, underline, overstrike and combinations of these
- Tkinter guarantees Times, Courier and Helvetica font families will be available

3

More appearance

- Border & relief: bd=n can set border width
- Relief can be set to FLAT, SUNKEN, RAISED, GROOVE, SOLID, or RIDGE
- Cursor – cursor='gumby', watch, pencil, cross, and hand2
- Padding – space around widgets padx=n, pady=n

4

appearance

```
from Tkinter import *
```



```
root = Tk()
widget = Label(root, text = "Hello world", padx=10, pady=10);
widget.config(font=('helvetica', 20, 'underline italic'))
widget.config(cursor='gumby')
widget.config(bg='dark green', fg='white')
widget.config(bd=8, relief=RAISED)
```

```
widget.config(height=3, width=20)
widget.pack(padx=10, pady=10)
root.mainloop()
```

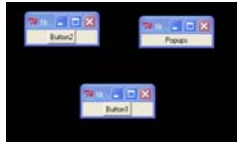
5

Top level windows

- Toplevel() and Tk() are root windows
- Tkinter GUIs always have a root window
- Root window opens when you run the program and where you pack important widgets
- Can create many toplevel windows
- They are automatically added to the event loop
- They are independently active but separate processes: when program exits, they all die

6

Multiple windows



```
import sys
from Tkinter import Toplevel, Button, Label, Tk
root = Tk()
win1=Toplevel()
win2=Toplevel()
Button(win1, text="Button1", command=sys.exit).pack()
Button(win2, text="Button2", command=sys.exit).pack()
Label(root, text="Popups").pack()
win1.mainloop()
```

7

dialogs

- Popped up to provide or request info
- Two kinds:
 - Modal – block everything until dismissed
 - Nonmodal – remain without interfering with other windows
- Three ways to present them:
 - Standard
 - Old style
 - custom

8

Standard dialogs

- Tkinter comes with collection of precoded dialog windows
 - File selection
 - Error and warning
 - Question and answer prompts
- All standard boxes are modal – block main window
- Easy to use!

9

Dialog box example

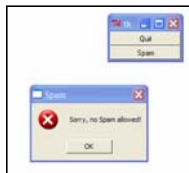
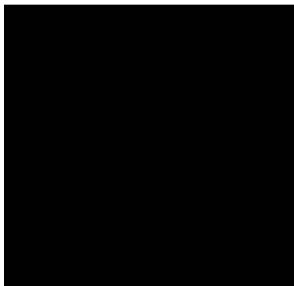
```
from Tkinter import *
from tkMessageBox import *

def callback():
    if askyesno('Verify', 'Do you really want to quit?'):
        showwarning('yes', 'Quit not yet implemented')

root = Tk()
errmsg = 'Worry, no Spam allowed!'
Button(root, text="Quit", command=callback).pack(fill=X)
Button(root, text="Spam", \
        command=(lambda: showerror('Spam', errmsg))).pack(fill=X)

root.mainloop()
```

10



11

Reusable quit

```
from Tkinter import *
from tkMessageBox import *
```

```
class Quit(Frame):
    def __init__(self, parent=None):
        Frame.__init__(self, parent)
        self.pack()
        widget = Button(self, text='Quit', command=self.quit)
        widget.pack(side=LEFT)
    def quit(self):
        ans = askokcancel('Verify exit', "Really quit?")
        if ans: Frame.quit(self)
```

```
if __name__ == "__main__":
    Quit().mainloop()
```



12

Reusable dialog boxes

- Let's write a reusable dialog module
- And reuse the quit class
- Demonstrate the power of Tkinter and GUIs

13

Dialog table (dictionary)

```
from tkinter import askopenfilename
from tkinter import askcolor
from tkinter import askquestion, showerror
from tkinter import askfloat

demos = {
    'Open': askopenfilename,
    'Color': askcolor,
    'Query': lambda: askquestion('Warning', 'You typed rm *"\nConfirm?'),
    'Error': lambda: showerror('Error!', 'You shall not pass!'),
    'Input': lambda: askfloat('Entry', 'Enter credit card number')
}
```

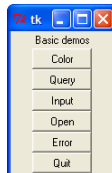
14

Client of dialog table

```
from tkinter import *
from dialogtable import demos
from tkinter import Quit
```

```
class Demo(Frame):
    def __init__(self, parent=None):
        Frame.__init__(self, parent)
        self.pack()
        Label(self, text="Basic demos").pack()
        for (key, value) in demos.items():
            Button(self, text=key, command=value).pack(side=TOP, fill=BOTH)
            Quit(self).pack(side=TOP, fill=BOTH)
```

```
if __name__ == "__main__":
    Demo().mainloop()
```



15

File dialog



16

Binding events

- The **bind** widget is used to catch button presses
- Most often used with other widgets
- Part of the event-driven protocol

17

```
from Tkinter import *
def showPosEvent(event):
    print 'Widget=%s X=%s Y=%s' % (event.widget, event.x, event.y)
def showAllEvent(event):
    print event
    for attr in dir(event):
        print attr, '=', getattr(event, attr)
def onKeyPress(event):
    print 'Got key press:', event.char
def onArrowKey(event):
    print 'Got arrow key'
def onReturnKey(event):
    print 'Got return key'
def onLeftClick(event):
    print 'Got left mouse button click'
    showPosEvent(event)
def onRightClick(event):
    print 'Got Right mouse button click'
    showPosEvent(event)
def onMiddleClick(event):
    print 'Got Middle mouse button click'
    showPosEvent(event)
    showAllEvent(event)
```

Lots of callback handler functions

18

```
tkroot = Tk()
labelfont=('courier', 20, 'bold')
widget = Label(tkroot, text='Hello World')
widget.config(bg='red', font=labelfont)
widget.config(height=5, width=20) #lines and chars
widget.pack(expand=YES, fill=BOTH)
```

```
widget.bind('<Button-1>', onLeftClick)
widget.bind('<Button-3>', onRightClick)
widget.bind('<Button-2>', onMiddleClick)
```

```
widget.bind('<KeyPress>', onKeyPress)
widget.bind('<Up>', onArrowKey)
widget.bind('<Return>', onReturnKey)
widget.focus()
tkroot.title('Click Me')
tkroot.mainloop()
```

19

What it looks like



This script demonstrates events that are caught with **bind**

20

Callback handlers

- The top part of the file consists of callback handler functions
- These functions are triggered when a bound event occurs
- These callback functions receive an event argument that contains details about the event that fired
- Each argument to the callback function is an instance of the Tkinter Event class; the details of the event are attributes of Event

21

<KeyPress>

- The key that is pressed is returned in ascii form in the Event object passed to the **callback handler** (event.char)
- Other attributes in the Event identify the key pressed in more detail
- Key presses can be intercepted by the top-level root window widget, or a widget that has been assigned keyboard focus (using focus method)

22

Additional bindable events

- <ButtonPress> fires when button goes down
- <ButtonRelease> fires when a button is released
- <Motion> fires when mouse pointer is moved
- <Enter> and <Leave> fire when mouse enters or exits a window
- <Destroy> fires when window widget is destroyed
- <FocusIn> and <FocusOut> runs as widget gains/loses focus
- <Down>, <Left>, and <Right> catch arrow key presses

23

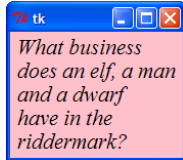
Message and Entry

- Allow for display and input of simple text
- Both are subsets of the Text widget
- Message widget:
 - a place to display text
 - Splits up long strings automatically
 - Can be embedded inside container widgets
 - Over a dozen configuration options

24

The Message widget

```
from Tkinter import *
msg = Message(text="What business does an elf, a man and a dwarf
    have in the riddermark?")
msg.config(bg='pink', font=('times', 16, 'italic'))
msg.pack()
mainloop()
```



25

Entry widget

- Simple, single-line text input field
- For form-like dialogs where user types a value into a field of a larger display
- Supports scrolling & key binding (for edit)

26

Example of Entry

```
from Tkinter import *
from quitclass import Quit
```

```
def fetch():
    print 'Input => "%s"' % ent.get()

root = Tk()
ent = Entry(root)
ent.insert(0, 'Type name: ')
ent.pack(side=TOP, fill=X) #grow horizontally

ent.focus()
ent.bind('<Return>', (lambda event: fetch()))
btn = Button(root, text='Fetch', command=fetch)
btn.pack(side=LEFT)
Quit(root).pack(side=RIGHT)
root.mainloop()
```



Both <Return> and fetch button trigger the script's fetch callback function; thus, either event get and displays the text.

27

Programming Entry widgets

- Value is set with **insert(pos, text)**
 - First param is position where text is inserted
 - "0" means the front because offset starts at 0 (integer 0 and string 0 are same in Python)
 - If Entry widget already contains text, you can delete it with **ent.delete(0,END)**

28

```
from Tkinter import *
from quitclass import Quit
fields = 'Name', 'Job', 'Pay'
```

```
def fetch(entries):
    for entry in entries:
        print 'Input => "%s"' % entry.get()
```

```
def makeform(root, fields):
    entries = []
    for field in fields:
        row = Frame(root)
        lab = Label(row, width=5, text=field)
        ent = Entry(row)
        row.pack(side=TOP, fill=X)
        lab.pack(side=LEFT)
        ent.pack(side=RIGHT, expand=YES, fill=X)
    entries.append(ent)
    return entries
```

```
if __name__ == '__main__':
    root = Tk()
    ents = makeform(root, fields)
    root.bind('<Return>', (lambda event, e=ents: fetch(e)))
    Button(root, text='Fetch',
        command=(lambda e=ents: fetch(e))).pack(side=LEFT)
    Quit(root).pack(side=RIGHT)
    root.mainloop()
```



29

CheckBox

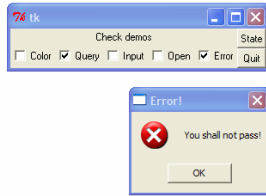
- A multiple choice input pidget
- We'll reuse the dialog table
- Use Tkinter variables to communicate

30

```

from Tkinter import *
from dialogtable import demos
from quitclass import Quit
class CheckBoxDemo(Frame):
    def __init__(self, parent=None, **args):
        Frame.__init__(self, parent, args)
        self.pack()
        self.tools()
        Label(self, text="Check demos").pack()
        self.vars=[]
        for key in demos.keys():
            var = IntVar()
            Checkbutton(self, text=key, variable=
                command=demos[key]).pack(
                self.vars.append(var)
        def report(self):
            for var in self.vars:
                print var.get(),
            print
        def tools(self):
            frm = Frame(self)
            frm.pack(side=RIGHT)
            Button(frm, text="State", command=self.report).pack(fill=X)
            Quit(frm).pack(fill=X)
if __name__ == '__main__':
    CheckBoxDemo().mainloop()

```



31

Other widgets

- **Radiobutton**: single-choice input
- **Scale**: aka slider

32

Images

- In Tkinter, graphical images are displayed by creating independent **PhotoImage** or **BitmapImage** objects
- These objects are then attached to other widgets via **image** attribute settings
- Buttons, labels, canvases, text and menus can all display images this way

33

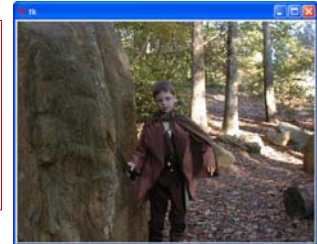
```

gifdir = "../gifs/"
from Tkinter import *
win = Tk()
img = PhotoImage(file=gifdir+"smallfrodo.gif")
Button(win, image=img).pack()
win.mainloop()

```

Here, PhotoImage loads the file and allows it to be shown in a button.

Note: buttons are automatically sized to fit the photo.



34

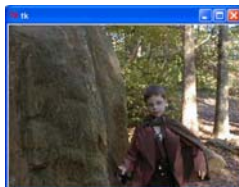
```

gifdir = "../gifs/"
from Tkinter import *
win = Tk()
img = PhotoImage(file=gifdir+"smallfrodo.gif")
can = Canvas(win)
can.pack(fill=BOTH)
can.create_image(2, 2, image=img, anchor=NW)
win.mainloop()

```

Here we use a canvas to hold the picture.

Note that canvases are not automatically sized to fit the image!



35

Some gotchas (subject to change)

- photoImage widget only supports GIF, PPM and PGM graphic file formats
- BitmapImage support X Windows-style .xbm bitmap images
- There are other viewing tools: PIL, PyView
- Unlike other widgets, an image is lost if Python image object is garbage-collected. Need global reference.

36

Fun with buttons and images

- We'll display a button that changes images randomly each time it's pressed
- **glob** gives a list of files that end *.gif*
- **random.choice** picks and returns an item from a list at random
- We'll call `config` with new option settings

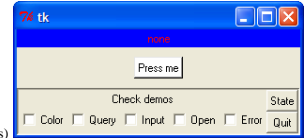
37

```
from Tkinter import *
from glob import glob
import checkboxdemo
import random
gifdir='./gifs/'
```

```
def draw():
    name, photo=random.choice(images)
    lbl.config(text=name)
    pix.config(image=photo)
```

```
root = Tk()
lbl = Label(root, text="none", bg='blue', fg='red')
pix = Button(root, text="Press me", command=draw, bg='white')
lbl.pack(fill=BOTH)
pix.pack(pady=10)
checkboxdemo.CheckBoxDemo(root, relief=SUNKEN, bd=2).pack(fill=BOTH)
```

```
files = glob(gifdir+ "*.gif")
images=map(lambda x: (x, PhotoImage(file=x)), files)
print files
root.mainloop()
```



38



39



PhotoImage automatically
resizes to match the pic

40