



Introduction to Python GUIs: Tkinter

Sources:
Programming Python by Lutz, ed: O'Reilly
Python, How to Program by Deitel et al., ed: Prentice Hall

GUIs

- Most software has it: windows, buttons, menus
- Dynamic & interactive

options

- wxPython: <http://wxpython.org> & Boa Constructor
- JPython
- Qt, designer & KDE
- Gnome & GTK

Tkinter

- Standard
- Comes with Python: if you have Python, you have Tkinter
- Lightweight
- Portable: Microsoft wondows, X Windows and Macintosh
- Well documented

Tkinter extensions

- PIL
- IDLE
- PMW



Hello world



```
from Tkinter import Label
widget = Label(None, text='Hello World!')
widget.pack()
widget.mainloop()
```

None is the parent, which defaults to Tk; usually widgets are attached to other containers.

Alternative 'hello world'

```
from Tkinter import *  
root = Tk()  
Label(root, text='Hello World!').pack(side=TOP)  
root.mainloop()
```

7

Alternative 'hello world'

```
from Tkinter import *  
root = Tk()  
Label(root, text='Hello World!').pack(side=TOP)  
root.mainloop()
```

Doesn't grow
when parent
grows:



8

Expanding widgets

Pass text to widget constructor

```
from Tkinter import *  
root = Tk()  
Label(text='Hello World!').pack(expand=YES, fill=BOTH)  
root.mainloop()
```

Label grows along
with its parent to
fill all space:



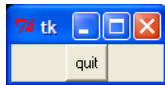
9

Don't have to pass text to constructor

```
from Tkinter import *  
widget = Label()  
widget['text'] = 'Hello World!'  
widget.pack(expand=YES, fill=BOTH)  
mainloop()
```

10

Using button instead of label



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

11

Same thing

```
from Tkinter import *  
root = Tk()  
but = Button(root, text='quit', command=sys.exit)  
but.pack(side=TOP)  
root.mainloop()
```

12

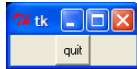
User-defined callback handlers

```
from Tkinter import *
```

```
def quit():  
    print "hello, I'm outta here!"  
    import sys; sys.exit()
```

```
root = Tk()  
but = Button(root, text = 'quit', command=quit)  
but.pack(side = TOP)  
root.mainloop()
```

Same button, but prints message upon termination. In previous examples we used existing callback function, now we use our own!



13

Using a lambda function

```
from Tkinter import *  
from sys import stdout, exit
```

```
but = Button(None, text = 'quit',  
            command=(lambda: stdout.write("I'm outta here") or exit()  
                    )  
            )  
but.pack()  
but.mainloop()
```

Lambda can only take an expression; or forces two expressions to be run.

14

Bound method callbacks

- Class **bound methods** work well as callback handlers
- They record both an instance to send the event to, and
- An associated method to call

15

Using a class to bind the callback

```
from Tkinter import *  
  
class HelloClass:  
    def __init__(self):  
        but = Button(None, text = 'quit', command=self.quit)  
        but.pack()  
  
    def quit(self):  
        print "hello, I'm outta here!"  
        import sys; sys.exit()  
  
HelloClass()  
mainloop()
```

16

Other callback protocols

- Other kinds of buttons: radio, check buttons, scales
- Menu command buttons
- Scrollbar protocols
- Window manager protocols: e.g. window close event
- Scheduled event protocols: timers, input data arrival

17

Adding multiple widgets

```
from Tkinter import *
```

```
def greet():  
    print "hello world"
```



```
win = Frame()  
win.pack()  
Label(win, text="Hello world").pack(side=TOP)  
Button(win, text = 'hello', command=greet).pack(side=LEFT)  
Button(win, text = 'quit', command=win.quit).pack(side=RIGHT)  
  
win.mainloop()
```

18

Frames

```
win = Frame()
win.pack()
Label(win, text="Hello world").pack(side=TOP)
Button(win, text = 'hello', command=greet).pack(side=LEFT)
Button(win, text = 'quit', command=win.quit).pack(side=RIGHT)
```

Here, we create a Frame and attach three buttons to it:
a Label and 2 Buttons
Two of the buttons have callbacks

19

clipping

Widgets that are packed first, are clipped last; i.e., packing order determines which are cut out if the widget is resized and becomes too small.



20

Packing order matters

from Tkinter import *

```
def greet():
    print "hello world"
```

```
win = Frame()
win.pack()
```

```
Button(win, text = 'hello', command=greet).pack(side=LEFT)
Label(win, text="Hello world").pack(side=TOP)
Button(win, text = 'quit', command=win.quit).pack(side=RIGHT)
```

```
win.mainloop()
```



21

Layout managers

- Arrange GUI components for programmer
- Allows programmer to focus on look/feel
- Managers in Tkinter:
 - Pack – arranges components in order added
 - Grid – arranges components in rows and columns
 - Place – allows programmer to specify size and location of components

22

Layout managers

- Choosing best layout manager can make it easier to program a GUI
- Using more than one type in Tkinter can cause problems

23

pack

- Places components in a container from top to bottom in order in which they are added to the GUI
- A container is a GUI component in which other components are placed
- When the edge of a container is reached, the container expands, if possible
- If container cannot expand, component is invisible

24

packing

- Packer starts out with an available space cavity that includes parent container
- As each widget is packed on a side, it's given the entire side
- Later pack requests are given the entire side of what is left
- After all widgets are pack, **expand** divides up the space that is left.
- **fill** can be used to stretch the widget

25

Using expand & fill

```
from Tkinter import *
```

```
def greet():  
    print "hello world"
```

```
win = Frame()  
win.pack()
```

```
Button(win, text = 'hello', command=greet).pack(side=LEFT,fill=Y)  
Label(win, text="Hello world").pack(side=TOP)  
Button(win,text='quit',command=win.quit).pack(side=RIGHT,expand=YES,fill=X)
```

```
win.mainloop()
```



26

Reusable GUIs with classes

```
from Tkinter import *
```

```
class Hello(Frame):  
    def __init__(self, parent=None):  
        Frame.__init__(self, parent)  
        self.pack()  
        self.data = 42  
        self.makeWidgets()  
    def makeWidgets(self):  
        widget = Button(self, text = 'Hello', command=self.message)  
        widget.pack(side=LEFT)
```

```
    def message(self):  
        self.data = self.data + 1  
        print "hello world %s" % self.data
```

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



```
Hello world 43  
Hello world 44  
...
```

27

Using inheritance to extend

```
from Tkinter import *  
from hello import Hello
```

```
class HelloQuit(Hello):  
    def makeWidgets(self):  
        Hello.makeWidgets(self)  
        Button(self, text='quit',command=self.quit).pack(side=RIGHT)  
    def message(self):  
        print 'hello', self.data
```

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

28

grid

- Grid layout manager divides container into rows and columns
- Can place components into a specific cell

29