An Introduction to Tkinter

David Beazley
Copyright (C) 2008
http://www.dabeaz.com

Note: This is an supplemental subject component to Dave's Python training classes. Details at:

http://www.dabeaz.com/python.html

Last Update : March 22, 2009

Overview

• A brief introduction to Tkinter
• Some basic concepts that make it work
• Some GUI-related programming techniques
• This is not an exhaustive reference
Tkinter

• The only GUI packaged with Python itself

• Based on Tcl/Tk. Popular open-source scripting language/GUI widget set developed by John Ousterhout (90s)

• Tk used in a wide variety of other languages (Perl, Ruby, PHP, etc.)

• Cross-platform (Unix/Windows/MacOS)

• It's small (~25 basic widgets)

Tkinter Hello World

• A very short example:

```python
>>> from Tkinter import Label
>>> x = Label(None,text="Hello World")
>>> x.pack()
>>> x.mainloop()
```

• Output (Windows)
Tkinter Hello World

• A more interesting example: A button

```python
>>> def response():
...     print "You did it!"
...
>>> from Tkinter import Button
>>> x = Button(None,text="Do it!",command=response)
>>> x.pack()
>>> x.mainloop()
```

• Clicking on the button....

  You did it!
  You did it!
  ...

Tkinter in a nutshell

• Typical steps in using Tkinter
  • You create and configure widgets (labels, buttons, sliders, etc.)
  • You pack them (geometry)
  • You implement functions that respond to various GUI events (event handling)
  • You run an event loop
The Big Picture

- A GUI lives in at least one graphical window
- Here it is.... an empty window (no widgets)

- This window is known as the "root" window
- Usually only one root window per application

Root Window

- To create a new root window:

    >>> from Tkinter import *
    >>> root = Tk(className="ApplicationName")
    >>>

- To start running the GUI, start its loop

    >>> root.mainloop()

- This isn't very exciting. Just a blank window
Widgets

- Widgets are graphical elements

```python
>>> from Tkinter import *
>>> root = Tk()
>>> b = Button(root, text="A Button")
>>> b.pack()
```

- All widgets belong to some window (parent)
- e.g., no free floating widgets

Widget Configuration

- Widgets have configuration options

```python
>>> b = Button(root, text="A Button", bg="blue", fg="white")
```

- Widgets can later be reconfigured

```python
>>> b.config(bg="red")  # Change background
```

- Get current settings with cget()

```python
>>> b.cget("bg")
'red'
>>> ```
Widget Events

• Most widgets respond to various events

```python
>>> def pressed():
...     print "You pressed it!"
...     

>>> b = Button(root,text="A Button",command=pressed)
```

Event handler

• Types of events and handler protocol depend on the widget (e.g., different for buttons than for scrollbars)

Widget State

• Widgets sometimes rely on "linked variables"

```python
ivar = IntVar()
svar = StringVar()
dvar = DoubleVar()
bvar = BooleanVar()
```

• Example: Text entry

```python
>>> svalue = StringVar()
>>> w = Entry(root,textvariable=svalue)
```

Holds current value of entry text

```python
>>> svalue.get()
'This is a test'
```
Widgets as Building Blocks

- Widgets are the basic building blocks

**Widget Tour**

- Labels:
  
  ```python
  >>> w = Label(root, text="A label")
  ```

  - Usually used for small text-labels
Widget Tour

• Messages

>>> w = Message(root, text="Stay tuned. A very important message concerning your mental stability is about to appear")

![Image]

• Used for informative messages/dialogs

Widget Tour

• Buttons:

```python
>>> def when_pressed():
...     print "Do something"
...
>>> w = Button(root, text="Press Me!", command=when_pressed)
```

![Image]
Widget Tour

• Checkbutton

```python
>>> debug_mode = IntVar(value=0)
>>> w = Checkbutton(root,text="Debug mode",
... variable=debug_mode)
... 
>>> debug_mode.get()
1
```
Widget Tour

• Scales/Sliders

```python
>>> temp = IntVar()
>>> def on_move(value):
...     print "moved", value
...     
... >>> w = Scale(root,label="Temperature",variable=temp,
...             from_=0,to=100,tickinterval=50,
...             orient='horizontal',command=on_move)
... >>>
```

![Scale widget example](image)

• Text entry

```python
>>> value = StringVar(root)
>>> w = Entry(root,textvariable=value)

>>> value.get()
'This is a test'
>>>```
Widget Tour

• Scrollbar

```python
>>> w = Scrollbar(root, orient="vertical")
```

• Note: Have omitted many details

Widget Tour

• Text-widget

```python
>>> sometext = open('README.TXT').read()
>>> w = Text(root, relief=SUNKEN)
>>> w.insert("1.0", sometext)
```
**Widget Tour**

- **Canvas**

```python
canvas = Canvas(root, width=250, height=250)
canvas.create_line(20, 30, 200, 100)
canvas.create_rectangle(40, 50, 100, 90)
canvas.create_text(150, 140, text="A test")
```

- **Menus**

```python
menu = Menu(root)
file = Menu(menu)
file.add_command(label='Open', command=open_cmd)
file.add_command(label='Close', command=close_cmd)
menu.add_cascade(label='File', menu=file)
edit = Menu(menu)
edit.add_command(label='Cut', command=cut_cmd)
edit.add_command(label='Paste', command=paste_cmd)
menu.add_cascade(label='Edit', menu=edit)
root.config(menu=menu)
```
Commentary

• Have covered some of the basic widgets
• There are many more, but same idea
• For complete details: consult a Tk reference
• Next step: arranging them within a window

Packing

• Widgets have to be placed somewhere within a window (geometry)
• The pack() method does this
• By default, pack places a widget centered at the top of a window
Choosing Sides

- You can pack a widget on any side

```python
w.pack(side=TOP)
w.pack(side=LEFT)
w.pack(side=BOTTOM)
w.pack(side=RIGHT)
```

Anchoring

- A widget can also be anchored in its space

```python
w.pack(side=TOP, anchor=W)  
w.pack(side=TOP, anchor=E)
```

- Anchoring is "directional" (East, West, etc.)

E, W, N, S, NW, NE, SW, SE
Multiple Widgets

• More than one widget can be packed

```python
>>> root = Tk()
>>> b1 = Button(root,text="Button 1")
>>> b2 = Button(root,text="Button 2")
>>> b1.pack(side=TOP)
>>> b2.pack(side=LEFT)
>>> root.mainloop()
```

Pop Quiz

• Let's add a third button

```python
>>> root = Tk()
>>> b1 = Button(root,text="Button 1")
>>> b2 = Button(root,text="Button 2")
>>> b3 = Button(root,text="Button 3")
>>> b1.pack(side=TOP)
>>> b2.pack(side=LEFT)
>>> b3.pack(side=BOTTOM)
>>> root.mainloop()
```

• ????
Pop Quiz

• Let's add a third button

```python
>>> root = Tk()
>>> b1 = Button(root, text="Button 1")
>>> b2 = Button(root, text="Button 2")
>>> b3 = Button(root, text="Button 3")
>>> b1.pack(side=TOP)
>>> b2.pack(side=LEFT)
>>> b3.pack(side=BOTTOM)
>>> root.mainloop()
```

Commentary: Packer

• Figuring out the Tk packer is probably the most mind-boggling aspect of Tk

• Keep in mind: It works hierarchically

• It packs things in order and carves up space
Filling/Expanding

- **Filling:** Widget expands to use all of the space that's been allocated to it
- **Expanding:** Widget expands to use all of its allocated space and adjacent free space
- Both specified by special options

```python
w.pack(side=SIDE, fill=X)
w.pack(side=SIDE, fill=Y)
w.pack(side=SIDE, fill=BOTH)
w.pack(side=SIDE, fill=FILL, expand=True)
```

Filling

- Consider two widgets:

```
>>> Button(root, text="tiny").pack()
>>> Button(root, text="humongous").pack()
>>>  
```

- Result looks terrible
### Filling

- Now, two widgets with filling

```python
>>> Button(root,text="tiny").pack(fill=X)
>>> Button(root,text="humongous").pack(fill=X)
```

- Result looks better

![Image showing two buttons with filling]

- Buttons fill out their horizontal space (X)

### Expanding

- Now consider this example:

```python
>>> Button(root,text="tiny").pack(fill=X)
>>> Button(root,text="humongous").pack(fill=X)
>>> w = Label(root,text="Label",bg="blue",fg="white")
>>> w.pack(fill=X)
```

Now, watch what happens if the window is expanded

![Image showing expanded window]

Note the empty space here
Expanding

• Expanding and filling

```python
>>> Button(root, text="tiny").pack(fill=X)
>>> Button(root, text="humongous").pack(fill=X)
>>> w = Label(root, text="Label", bg="blue", fg="white")
>>> w.pack(fill=BOTH, expand=True)
```

Now, watch what happens if the window is expanded

Label now takes up all remaining space

Frames

• Frames are like a sub-window
• A space to hold widgets
• Used to group widgets together

```python
>>> root = Tk()
>>> f = Frame(root)
>>> Label(f, text="Name: ").pack(side=LEFT)
>>> Entry(f).pack(side=RIGHT, fill=X, expand=True)
>>> f.pack()
>>> root.mainloop()
```
Using Frames

• Typically used to subdivide a window into logical components

```python
>>> root = Tk()
>>> f1 = Frame(root)
>>> f2 = Frame(root)
>>> f3 = Frame(root)
>>> f1.pack(side=TOP)
>>> f2.pack(side=LEFT)
>>> f3.pack(side=RIGHT)
```

• Widgets are then placed into each frame

• Frame is used as the "parent" window

Frame Example

• An entry field widget

```python
class EntryField(Frame):
    def __init__(self, parent, label, labelwidth=12):
        Frame.__init__(self, parent)
        l = Label(self, text=label, width=labelwidth, anchor=W)
        l.pack(side=LEFT, fill=X)
        Entry(self).pack(side=RIGHT, fill=X, expand=True)
```

• Creates an enclosing frame

• Packs two other widgets inside
Frame Example

• Example:

```python
root = Tk()
find = EntryField(root,"Find:")
find.pack(side=TOP,fill=X,pady=3)
replace = EntryField(root,"Replace with:")
replace.pack(side=TOP,fill=X,pady=3)
```

![Frame Example](image)

Frame Example

• Another widget: An option bar

```python
class Optionbar(Frame):
    def __init__(self,parent,label,options,labelwidth=12):
        Frame.__init__(self,parent)
        l = Label(self,text=label,width=labelwidth,anchor=W)
        l.pack(side=LEFT)
        for option in options:
            cb = Checkbutton(self,text=option)
            cb.pack(side=LEFT,anchor=W,expand=True)
```
Frame Example

• Example:

```python
root = Tk()
options = OptionBar(root, "Options",
                   ["Regular expression", "Match case", "Whole word",
                    "Wrap around"])
```

Frame Example

• Another widget: A radio button bar

```python
class RadioChoice(Frame):
    def __init__(self, parent, label, choices, default_labelwidth=12):
        Frame.__init__(self, parent)
        l = Label(self, text=label, width=labelwidth, anchor=W)
        l.pack(side=LEFT)
        self.choice = StringVar(self, default)
        for choice in choices:
            rb = Radiobutton(self, text=choice,
                             variable=self.choice, value=choice)
            rb.pack(side=LEFT, anchor=W, expand=True)
```
Frame Example

Example:

```python
root = Tk()
options = RadioChoice(root,"Direction", ["Up","Down"], "Down")
```

Another widget: A series of buttons

```python
class ButtonList(Frame):
    def __init__(self,parent,buttons):
        Frame.__init__(self,parent)
        for b in buttons:
            Button(self,text=b).pack(side=TOP,fill=X,pady=1)
```
Frame Example

• Example:

```python
root = Tk()
buttons = ButtonList(root,"close","Find","Replace",
                     "Replace+Find","Replace All")
buttons.pack()
```

Frame Example

• A Find/Replace Dialog

```python
class FindReplace(Frame):
    def __init__(self,parent):
        Frame.__init__(self,parent)
        but = ButtonList(self,"close","Find","Replace",
                         "Replace+Find","Replace All")
        but.pack(side=RIGHT,fill=X,padx=2)
        find = EntryField(self,"Find:")
        find.pack(side=TOP,fill=X,pady=3)
        replace = EntryField(self,"Replace:")
        replace.pack(side=TOP,fill=X,pady=3)
        opt = OptionBar(self,"Options",["Regular expression", "Match case", "Whole word", "Wrap around")
        opt.pack(side=TOP,fill=X,pady=3)
        dir = RadioChoice(self,"Direction",["Up","Down"])
        dir.pack(side=TOP,anchor=W,pady=3)
```

• Uses widgets we created earlier
Frame Example

• Example:

```python
root = Tk()
FindReplace(root).pack()
```

Frame Example

• Example:

```python
root = Tk()
FindReplace(root).pack()
```
Commentary

- Can see how GUI is built up from pieces
- I have omitted several key parts
  - Managing state
  - Callbacks

Maintaining State

- Widgets often need to store internal information
- Values of entry fields, button selections, etc.
- Other code needs to get that data
- Two approaches: Objects, Functions
Widgets as Objects

- Define each widget as a class (often inheriting from Frame)
- Store all state as attribute of the object
- Provide methods to access data as needed

Example: EntryField widget

```python
class EntryField(Frame):
    def __init__(self, parent, label, labelwidth=12):
        Frame.__init__(self, parent)
        self.value = StringVar(self)
        l = Label(self, text=label, anchor=W, width=labelwidth)
        l.pack(side=LEFT)
        e = Entry(self, textvariable=self.value)
        e.pack(side=RIGHT, fill=X, expand=True)
    def get_value(self):
        return self.value.get()
```
Widgets as Objects

• Example: EntryField widget

class EntryField(Frame):
    def __init__(self, parent, label, labelwidth=12):
        Frame.__init__(self, parent)
        self.value = StringVar(self)
        self.value = StringVar(self)
        l = Label(self, text=label, anchor=W, width=labelwidth)
        l.pack(side=LEFT)
        e = Entry(self, textvariable=self.value)
        e.pack(side=RIGHT, fill=X, expand=True)
    def get_value(self):
        return self.value.get()
Widgets as Objects

• Example: EntryField Widget Use

```python
class FindReplace(Frame):
    def __init__(self, parent):
        Frame.__init__(self, parent)
        self.find = EntryField(self, "Find:")
        self.replace = EntryField(self, "Replace:")
        self.find.pack(side=TOP, fill=X)
        self.replace.pack(side=TOP, fill=X)
        Button(self, text="Go", command=self.do_it)
    def do_it(self):
        ftext = self.find.get_value()
        rtext = self.replace.get_value()
        print "Replacing '%s' with '%s'" % (ftext, rtext)
```

Invoked on button press

Value of entry fields retrieved

Example: EntryField Widget Use
Widgets as Functions

- Write a function that simply creates a widget
- Store all state inside function using closures
- Return a function for accessing state
- This is a more sly approach

```python
def entryfield(parent, label, labelwidth=12, **packopts):
    f = Frame(parent)
    f.pack(**packopts)
    l = Label(f, text=label, width=labelwidth)
    l.pack(side=LEFT, anchor=W)
    value = StringVar(f)
    e = Entry(f, textvariable=value)
    e.pack(side=RIGHT, fill=X, expand=True)
    return lambda: value.get()
```

Example: EntryField function
Widgets as Functions

• Example: EntryField function

```python
def entryfield(parent, label, labelwidth=12, **packopts):
    f = Frame(parent)
    f.pack(**packopts)
    l = Label(f, text=label, width=labelwidth)
    l.pack(side=LEFT, anchor=W)
    value = StringVar(f)
    e = Entry(f, textvariable=value)
    e.pack(side=RIGHT, fill=X, expand=True)
    return lambda: value.get()
```

• Example: EntryField function

```python
def entryfield(parent, label, labelwidth=12, **packopts):
    f = Frame(parent)
    f.pack(**packopts)
    l = Label(f, text=label, width=labelwidth)
    l.pack(side=LEFT, anchor=W)
    value = StringVar(f)
    e = Entry(f, textvariable=value)
    e.pack(side=RIGHT, fill=X, expand=True)
    return lambda: value.get()
```
Widgets as Functions

• Example: Using the EntryField function

```python
def find_replace(ftext, rtext):
    print 'Replacing %s with %s' % (ftext, rtext)

def find_replace_gui(parent):
    findv = entryfield(parent, "Find:", side=TOP, fill=X)
    replacev = entryfield(parent, "Replace", side=TOP, fill=X)
    b = Button(parent, text="Go",
               command=lambda: find_replace(findv(), replacev()))
    b.pack(side=TOP, fill=X)

root = Tk()
find_replace_gui(root)
```

• Example: Using the EntryField function

```python
def find_replace(ftext, rtext):
    print 'Replacing %s with %s' % (ftext, rtext)

def find_replace_gui(parent):
    findv = entryfield(parent, "Find:", side=TOP, fill=X)
    replacev = entryfield(parent, "Replace", side=TOP, fill=X)
    b = Button(parent, text="Go",
               command=lambda: find_replace(findv(), replacev()))
    b.pack(side=TOP, fill=X)

root = Tk()
find_replace_gui(root)
```
Widgets as Functions

- Example: Using the EntryField function

```python
def find_replace(ftext, rtext):
    print "Replacing '%s' with '%s'" % (ftext, rtext)

def find_replace_gui(parent):
    findv = entryfield(parent, "Find:", side=TOP, fill=X)
    replacev = entryfield(parent, "Replace", side=TOP, fill=X)
    b = Button(parent, text="Go",
               command=lambda: find_replace(findv(), replacev()))
    b.pack(side=TOP, fill=X)

root = Tk()
find_replace_gui(root)
```

Callback Handling

- Most TK widgets have some kind of callback
- Callback is often a simple function
- Example:

```python
def button_press():
    print "Button pressed"

Button(root, text="Go", command=button_press)
```

- If callback takes arguments, need to use lambda or other functional trick
Callbacks and Lambda

• Using lambda to supply extra arguments

```python
def button_press(which):
    print "You pressed", which

Button(root,text="Go",
    command=lambda:button_press('go'))
Button(root,text="Cancel",
    command=lambda:button_press('cancel'))
```

• Note: used this in find/replace example

Callback Alternatives

• Instead of lambda, may several alternatives

• Partial Function Evaluation

```python
from functools import *
def button_press(which):
    print "You pressed", which

Button(root,text="Go",
    command=partial(button_press,'go'))
Button(root,text="Cancel",
    command=partial(button_press,'cancel'))
```

• Similar to lambda, but subtle differences
Callback Alternatives

• Callable object

```python
def button_press(which):
    print "You pressed", which

class Pressed(object):
    def __init__(self,name):
        self.name = name
    def __call__(self):
        button_press(self.name)

Button(root,text="Go", command=Pressed('go'))
Button(root,text="Cancel", command=Pressed('cancel'))
```

• Uses fact that overriding `__call__()` lets an object be called like a function

Pre-built Widgets

• Tkinter has a number of prebuilt widgets
• Standard dialogs
• Simple data entry
• Filename and color selection
• Useful if quickly putting something together
Standard Dialogs

• Informational dialog

```python
>>> from tkMessageBox import *
>>> showinfo("FYI","I am about to destroy your computer")
```

![Informational dialog example](image1)

• Warning dialog

```python
>>> from tkMessageBox import *
>>> showwarning("Warning","Operation Unsuccessful")
```

![Warning dialog example](image2)
Standard Dialogs

• Error dialog

```python
>>> from tkMessageBox import *
>>> showerror("Fatal Error","Everything is hosed!")
```

![Error dialog example]

• Yes/No dialog

```python
>>> from tkMessageBox import *
>>> askyesno("Confirm","Are you sure you're ready?")
```

• Returns True/False
Standard Dialogs

• Ok/Cancel Dialog

```python
>>> from tkMessageBox import *
>>> askokcancel("Confirm","About to run a loop")
```

• Returns True/False

Standard Dialogs

• Retry/Cancel Dialog

```python
>>> from tkMessageBox import *
>>> askretrycancel("Try Again","Not responding")
```

• Returns True/False
Entry Dialogs

• Enter string, integers, floats

```python
>>> from tkSimpleDialog import *
>>> askinteger("The value", "Enter a value")
42
```

• Variants:
  askinteger()
  askfloat()
  askstring()

Filename Dialog

• Select a filename for opening

```python
>>> from tkFileDialog import *
>>> askopenfilename()
'C:/Python25/README.txt'
```
Directory Dialog

• Select a folder

```python
>>> from tkFileDialog import *
>>> askdirectory()
'C:/Python25/Doc'
```
Color Chooser

- Selecting a color

```python
>>> from tkColorChooser import *
>>> askcolor()
((0, 0, 255), '#0000ff')
```

Commentary

- Using standard dialogs may be useful for simple scripts (especially if no command line)

```python
from tkFileDialog import *
from tkSimpleDialog import *

filename = askopenfilename()
pat      = askstring("Pattern", "Enter search regex")
output   = asksaveasfilename()

# Go run the program (whatever)
...
```

- Unsophisticated, but it works
Summary

- A high-level overview of using Tkinter
- Tour of popular widgets
- Some details on geometry, packing, etc.
- How to create more complex widgets
- Pre-built widgets
- Have omitted a lot of detail

More Information

- "Programming Python, 3rd Ed." by Mark Lutz (O'Reilly)
- "Python and Tkinter Programming" by John Grayson.
- "Practical Programming in Tcl and Tk, 4th Ed." by Brent Welch, Ken Jones, and Jeffrey Hobbs