Tkinter – GUIs in Python

Dan Fleck
CS112
George Mason University

NOTE: Some of this information is not in your textbook!
See references for more information!

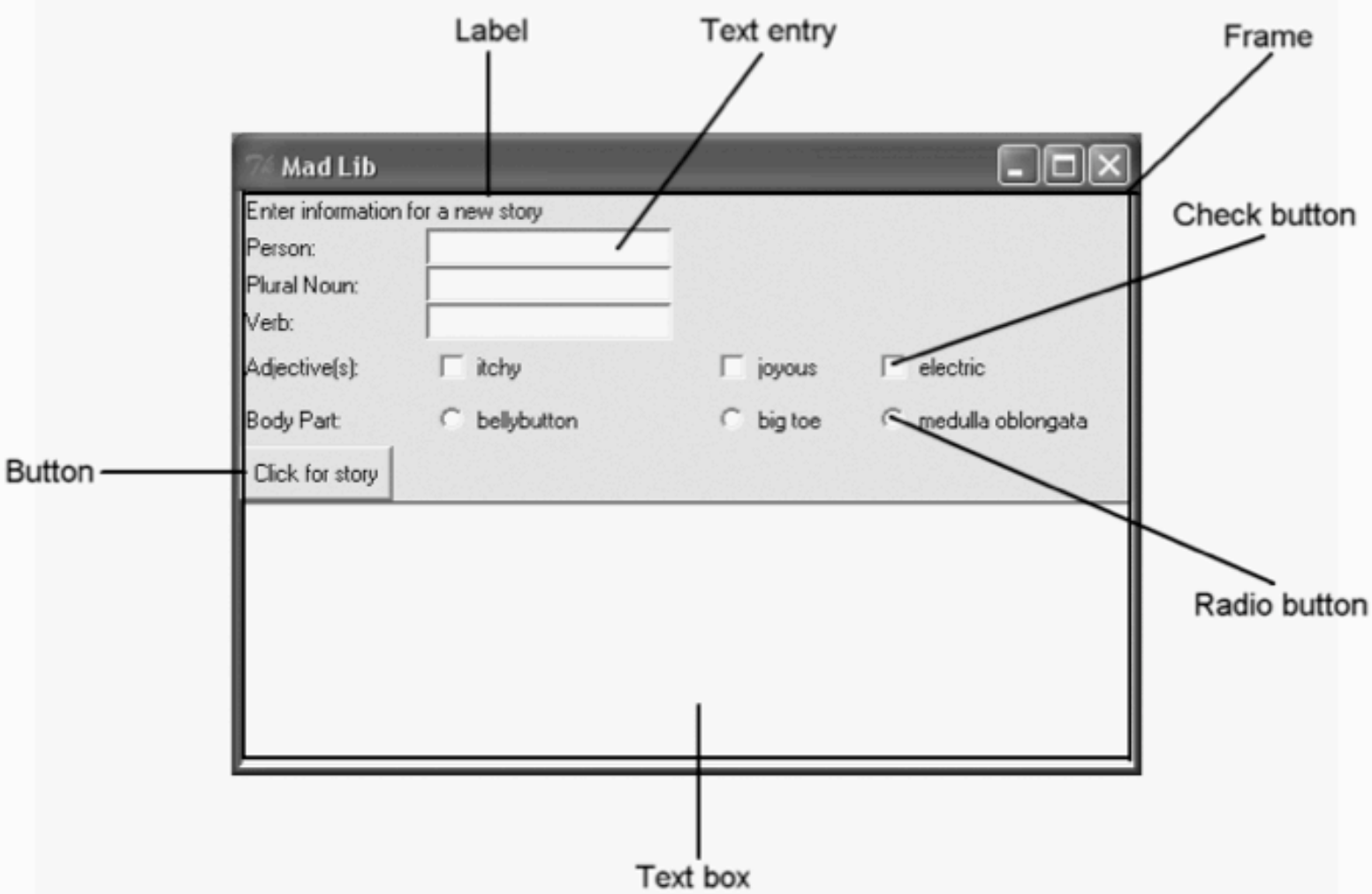
What is it?

- Tkinter is a Python interface to the Tk graphics library.
  - Tk is a graphics library widely used and available everywhere
- Tkinter is included with Python as a library. To use it:
  - import * from Tkinter
    - or
  - from Tkinter import *

What can it do?

- Tkinter gives you the ability to create Windows with widgets in them
- Definition: widget is a graphical component on the screen (button, text label, drop-down menu, scroll bar, picture, etc…)
- GUIs are built by arranging and combining different widgets on the screen.

Widgets (GUI Components)

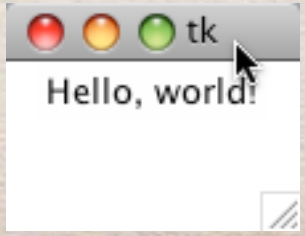


First Tkinter Window

hello1.py – /Users/dfleck/Documents/gmuwebsite/classes/cs112/spring09/samplecode/tkinter/hello1.py

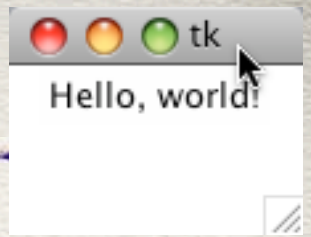
```
# File: hello1.py
from Tkinter import *

root = Tk()  # Create the root (base) window where all widgets go
root.grid()  # Start using the grid layout manager

w = Label(root, text="Hello, world!") # Create a label with words

w.grid() # Put the label into the window

root.mainloop() # Start the event loop
```

Ln: 10 Col: 7



Coming up: Explain the code

Explain the code




```
# File: hello1.py
from Tkinter import *

root = Tk()

w = Label(root, text="Hello, world!")

w.grid()

root.mainloop()
```

Create the parent window. All applications have a “root” window. This is the parent of all other widgets, you should create only one!

A Label is a widget that holds text
This one has a parent of “root”
That is the mandatory first argument to the Label’s constructor

Tell the label to place itself into the root window at row=0, col=0 .
Without calling grid the Label will NOT be displayed!!!

Windows go into an “event loop” where they wait for things to happen (buttons pushed, text entered, mouse clicks, etc…) or Windowing operations to be needed (redraw, etc..). You must tell the root window to enter its event loop or the window won’t be displayed!

Widgets are objects

- Widgets are objects.

- http://www.effbot.org/tkinterbook/tkinter-index.htm#class-reference
- Classes:
  - Button, Canvas, Checkbutton, Entry, Frame, Label, Listbox,Menu,Menubutton, Message, Radiobutton, Scale, ScrollBar, Text, TopLevel, and many more…

More objects we can build

button1.py – /Users/dfleck/Documents/gmuwebsite/classes/cs112/spring09/samplecode/tkinter/...

```
#Button1.py
from Tkinter import *

# Create the root
#(base) window where all widgets go
root = Tk()

# Make the window bigger
root.geometry("200x100")

# Create a label with words
w = Label(root, text="Hello, world!")
w.grid(row=0,column=0) # Put the label into the window

# Create a button and put in the window
myButton = Button(root, text="Exit")
myButton.grid(row=0,column=1)

root.mainloop() # Start the event loop
```

But nothing happens when we push the button!
Lets fix that with an event!

Coming up: Making the button do something

Making the button do something

button2.py – /Users/dfleck/Documents/gmuwebsite/classes/cs112/spring09/samplecode/tkinter/button2...

```
Button2.py
rom Tkinter import *

ef buttonPushed():
   print "Button pushed!"

ef main():

   # Create the root
   #(base) window where all widgets go
   root = Tk()

   # Make the window bigger
   root.geometry("200x100")

   # Create a label with words
   w = Label(root, text="Hello, world!")
   w.grid(row=0,column=0) # Put the label into the window

   # Create a button and put in the window
   myButton = Button(root, text="Exit", command=buttonPushed)
   myButton.grid(row=0,column=1)

   root.mainloop() # Start the event loop
```

Ln: 8 Col: 0

This says, whenever someone pushes the button, call the buttonPushed function. (Generically any method or function called by an action like this is a “callback”)

Coming up: Making the button close the window

Making the button close the window

```
# See Button3.py
# Use a class so we can store the attribute for later use

class MyButtonCloser(object):
    def __init__(self):
        root = Tk()
        self.root = root
        …
        myButton = Button(root, text="Exit", command=self.buttonPushed)

    def buttonPushed(self):
        self.root.destroy() # Kill the root window!
```

Store attribute for later use

Don't forget self!

Close the global root window

Calling this also ends the mainloop() function (and thus ends your program)

Creating text entry box

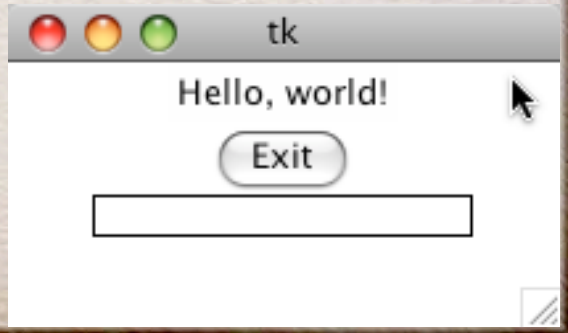
General form for all widgets:

1. # Create the widget
   widget = <widgetname>(parent, attributes…)
2. widget.grid(row=someNum, column=somNum)
   place the widget to make it show up

```
def createTextBox(parent):
    tBox = Entry(parent)
    tBox.grid(row=3, column=1)
```

From main call:

```
createTextBox(root)
```



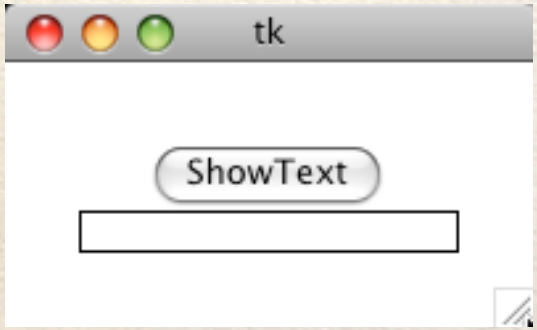
Using a text entry box

To use a text entry box you must be able to get information from it when you need it. (Generally in response to an event)

For us, this means make the entry box an attribute so we can get the info when a button is pressed

Using a text entry box

- Create it as an attribute
- Use “get” method when needed to get the text inside the box



- See TextEntryBox1.py

Creating a Changeable Label

- Create a StringVar object
- Assign the StringVar to the label
- Change the StringVar, and the label text changes

Creating a label you can change

Labels usually cannot be changed after creation

To change label text you must:

```
# Create a StringVar
myText = StringVar()
myText.set("Anything")

# Associate the StringVar with the label
myLabel = Label(parent, textvariable=myText)
```

See ChangeableLabel.py

Many more widgets

- I like this most
- http://effbot.org/tkinterbook/

- 2nd best:
- http://www.pythonware.com/library/tkinter/introduction/index.htm

- Find one you like? Let me know…

Layout management

- You may have noticed as we call “grid”. If not, the widgets will not show up!
- Grid is a layout or geometry manager. It is responsible for determining where widgets go in a window, and what happens as the window changes size, widgets get added, removed, etc…
- Most windowing toolkits have layout management systems to help you arrange widgets!

Grid parameters

- **row, column** – specify the row and column location of each widget.
  - 0,0 is the upper left corner
  - Empty rows are discarded (they do NOT make blank space)
- **rowspan, columnspan** – specify how many rows or columns a single widget should take
- **padx, pady** – specify how much blank space should be put around the widget

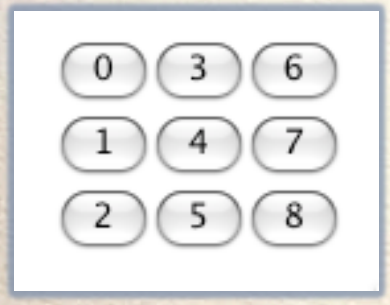
Grid parameters

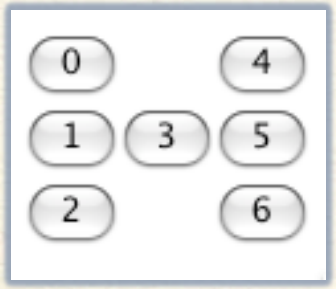
- **sticky** - Defines how to expand the widget if the resulting cell is larger than the widget itself. This can be any combination of the constants S, N, E, and W, or NW, NE, SW, and SE.

- For example, W (west) means that the widget should be aligned to the left cell border. W+E means that the widget should be stretched horizontally to fill the whole cell. W+E+N+S means that the widget should be expanded in both directions.

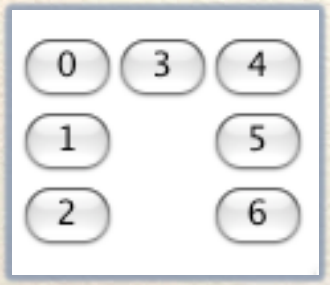
- Default is to center the widget in the cell.

Examples

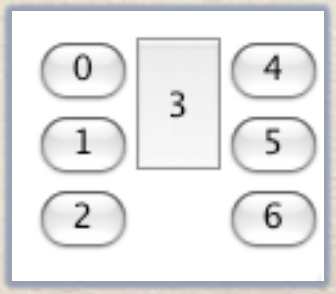





```
self.__createButton(root).
  grid(row=0,column=1, rowspan=3)
```

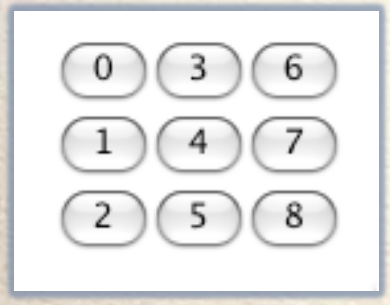


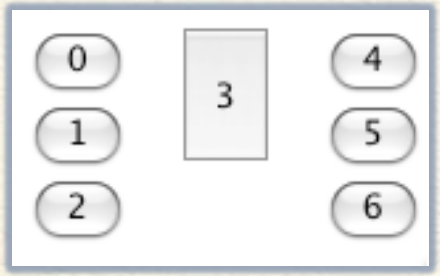

```
self.__createButton(root).
      grid(row=0,column=1,
            rowspan=3, sticky=N)
```



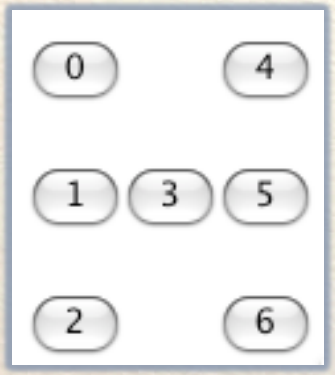

```
self.__createButton(root).
     grid(row=0,column=1,
            rowspan=2, sticky=N+S)
```

Examples






```
self.__createButton(root).
        grid(row=0,column=1, rowspan=2,
        sticky=N+S, padx=20)
```




```
self.__createButton(root).
    grid(row=1,column=1,
        sticky=N+S, pady=20)
```

Other geometry managers

Python has other geometry managers (instead of pack) to create any GUI layout you want

- pack – lets you put items next to each other in different ways, allowing for expansion
- grid – lets you specify a row,column grid location and how many rows and columns each widget should span
- place – specify an exact pixel location of each widget

WARNING: Never use multiple geometry managers in one window! They are not compatible with each other and may cause infinite loops in your program!!

Showing Images

An image is just another widget.

photo = PhotoImage(file=‘somefile.gif’)

Note: Tkinter only supports GIF, PGM, PBM, to read JPGs you need to use the Python Imaging Library

```
im = PhotoImage(file='cake.gif') # Create the PhotoImage widget

# Add the photo to a label:
w = Label(root, image=im) # Create a label with image
w.image = im # Always keep a reference to avoid garbage collection
w.grid() # Put the label into the window
```

Guess how you put an image in a Button?

Showing Images

A Canvas is a container that allows you to show images and draw on the container. Draw graphs, charts, implement custom widgets (by drawing on them and then handling mouse-clicks).

```
myCanvas = Canvas(root, width=400, height=200)
myCanvas.create_line(0, 0, 200, 100)
myCanvas.create_line(0, 100, 200, 0, fill="red", dash=(4, 4))
myCanvas.create_image(0, 0, anchor=NW, image=myPhotoImage)
```

How to use a canvas: http://effbot.org/tkinterbook/canvas.htm

How can we change the background color of a canvas?

Capturing mouse-clicks

- To capture mouse events you can "bind" events to a widget.
  - widget.bind(event, handler)
  - events can be:
    - <Button-1>
      - (1 is left mouse button, 2=right, 3=middle)
    - <Double-Button-1> - double clicked button 1
    - <Enter> - mouse entered the widget
    - <Leave> - mouse left the widget
    - <Return> - user pressed enter key
    - <key> (<a> for example) – user pressed "a"

Capturing mouse-clicks

For example, to make a button beg to be clicked:

```
def mouseEntered(event):
    button = event.widget
    button.config(text = "Please Please click me")

def mouseExited(event):
    button = event.widget
    button.config(text = "Logon")

def main():

    root = Tk()  # Create the root (base) window where all widgets go
    b = Button(root, text="Logon")
    b.bind("<Enter>",mouseEntered)
    b.bind("<Leave>",mouseExited)
    b.grid()
    root.mainloop() # Start the event loop

main()
```

Step 4: Write functions (or methods) to handle events. Notice: event object automatically passed into event handler!

Step 3: Bind events to functions

General Design Strategy

- Design the GUI – Layout what widgets you want, and where they should go
- Code the GUI
- Tell the system what events you want to know about
  - associate events with the appropriate event handlers (typically called **binding** or **registering** an event listener)
- Tell the system to begin accepting events
  - root.mainloop()

Capturing mouse-clicks

```
def mouseEntered(event):
    button = event.widget
    button.config(text = "Please Please click me")
```

Notice how I say “event.widget”… that is because all events store as data the widget that caused the event. In this case it is a button. (This again is because event is an object of class Event. That object stores data items – one of which is named “widget”.

Note: in the project you (might) need to bind left-button mouse events to the canvas and then look at the x,y location of the click. Is x,y stored in the event? Check the link below to see the names ot everything you can get from an event object just by saying:

myVariable = event.attribute

http://www.pythonware.com/library/tkinter/introduction/events-and-bindings.htm

Common problem!

```
def main():
    global root
    root = Tk()  # Create the root (base) window where all widgets go
    b = Button(root, text="Logon")
    b.bind("<Enter>",mouseEntered)
    b.bind("<Leave>",mouseExited)
    b.pack()
    root.mainloop() # Start the event loop

main()
```

WARNING: When you specify a function, you must NOT use parenthesis… using parenthesis CALLS the function once.. you want to pass the function as a parameter!

```
b.bind("<Enter>", mouseEntered)   # GOOD

b.bind("<Enter>", mouseEntered()) # BAD!
```

How mouse-clicks work: the event loop

- In this GUI we are using event based programming."**root.mainloop()**" starts an event loop in Python that looks like this:

  - while (True): # Loop forever
    wait for an event
    handle the event (usually call an event
    handler with the event information object)

- Many events you never see (window resized, iconified, hidden by another window and reshown…) You can capture these events if desired, but Tkinter handles them for you and generally does what you want.

Event Driven Programming

- Event driven programming – a programming paradigm where the flow of the program is driven by sensor outputs or user actions (aka events)

  – Wikipedia

- Batch programming – programming paradigm where the flow of events is determined completely by the programmer

  – Wikipedia

BATCH
Get answer for question 1
Get answer for question 2
Etc…

EVENT-BASED
User clicked “answer q1 button”
User clicked “answer q3 button”
User clicked “answer q2 button”
Etc…

Example: Graphical Project 2

- Lets implement Project 2 with a GUI
- Design:
  - URLImage : This class will be the Image. What are it's attributes? methods?
  - GUIFrame : This class will build the GUI, and run the GUI
  - Driver : This module will start the program, load the data (URLImages) and then call the GUIFrame to display it

Coming up: Which type is it (batch or event based?)

Which type is it (batch or event based?)

1. Take all the grades for this class and calculate final grade for the course — Batch
2. World of Warcraft — Event Based
3. Any video game — Event Based
4. GuessMyNumber Lab — Batch

Event Driven Systems

- An event-driven system is a software system that operates (i.e., is driven) via interactions (i.e., events) by some external entity (i.e., the “user”) with the application GUI.
- There are other forms of event-driven systems, but for the purposes of this class, we will limit the scope as defined above.
- The order of these interactions define the specific operation of a given instance of the application (i.e., it behaves differently depending upon the specific interactions offered by a specific user).
- The operation of an event-driven system is defined by the particular Event Model implemented by the system.

Event Driven vs Procedural

- Procedural System
  - Request input from user via menus, sub-menus, prompts, etc.
  - User must provide each piece of information in some given sequence
  - Process data & respond to user (and optionally repeat process)
- Event-Driven System
  - Present user with GUI
  - User can provide information and use controls in any (pre-determined) order, depending upon the design of the GUI

General Event Model

- Event
  - An object generated as a result of a specific interaction with the system
  - Examples: button press, mouse click, typing text in a field and pressing <ENTER>, etc.
- Event Listener
  - A mechanism that waits for notification that a particular event has occurred
- Event Handler
  - A process that should be executed in response to the event having been generated

Coming up: Lets create a drawing program

Lets create a drawing program

- Goal: Create a drawing program that allows us to draw lines easily

See DrawingCanvas.py versions 1,2,3,4

List boxes

- List boxes allow you to select one (or more) items from a list of items
- See this link: http://www.pythonware.com/library/tkinter/introduction/x5453-patterns.htm
- And the sample code:
  - listbox.py

Adding a title to your window

- This is actually very simple. You simply call the title method of the root window:

```
root.title("This is my window title")
```

- You should do this before you call root.config()

Message Dialog Boxes

- A dialog box is a small modal window that appears on top of the main window
  - used to ask a question, show a message or do many other things
  - File->Open usually opens a dialog box
  - **Definition:** A modal window is one that temporarily stops all other GUI processing (events)

- You may notice that in many programs the dialog box to open a file is very similar, or the dialog box to select a file or choose a color. These are very standard things, and most GUI toolkits (including Tk) provide support to make these tasks easy.

Message Dialog Boxes

- Using tkinter to create a dialog box you do this code:

```
import tkMessageBox   # Another way you can import

tkMessageBox.showinfo(title="Game Over",
    message="You have solved the puzzle… good work!")
```

- You can also call showwarning, showerror the only difference will be the icon shown in the window.

Question Dialog Boxes

Question dialogs are also available

```
from tkMessageBox import *

ans = askyesno("Continue", "Should I continue?”)
```

ans will be True (for Yes) or False (for No).
What do you do with answer then?

Other questions available are: askokcancel, askretrycancel, askquestion

Warning: askquestion by itself will return “yes” or “no” as strings, NOT True and False!

File Dialog Boxes

- See this link for some examples of standard dialogs to
  - open a file
  - select a directory
  - selecting a file to save

http://www.pythonware.com/library/tkinter/introduction/x1164-data-entry.htm

Data Input Dialogs

- You can also use tkSimpleDialog to ask for a number or string using a dialog box:

askstring(title, prompt), askinteger…, askfloat...

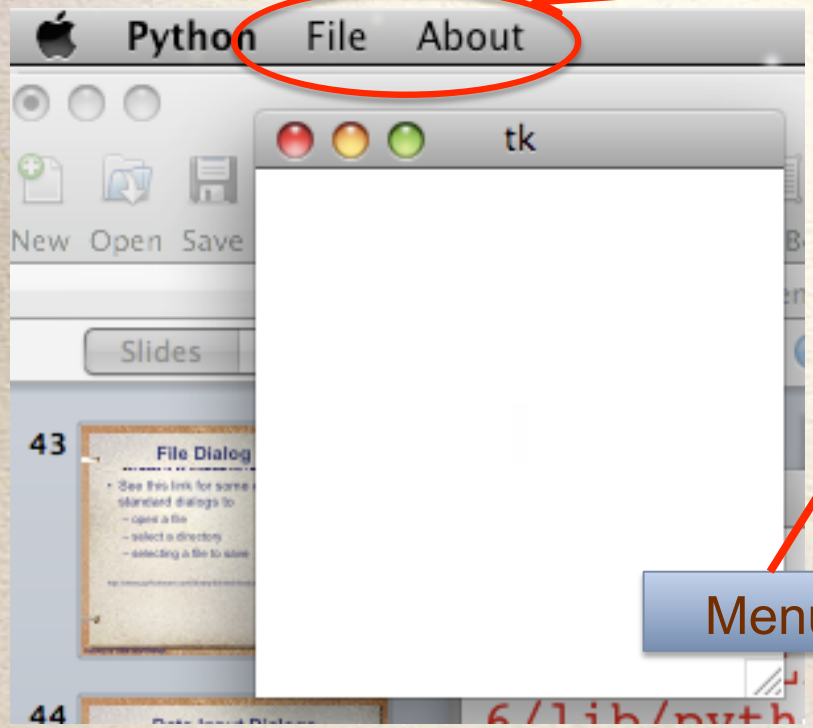
```
from tkSimpleDialog import *
ans = askstring("Title", "Give me your name")
print ans
ans = askinteger(”Dialog Title", "Give me an integer")
print ans
ans = askinteger(”Num", "Give me an integer between 0 and 100",
    minvalue=0, maxvalue=100)
print ans
```

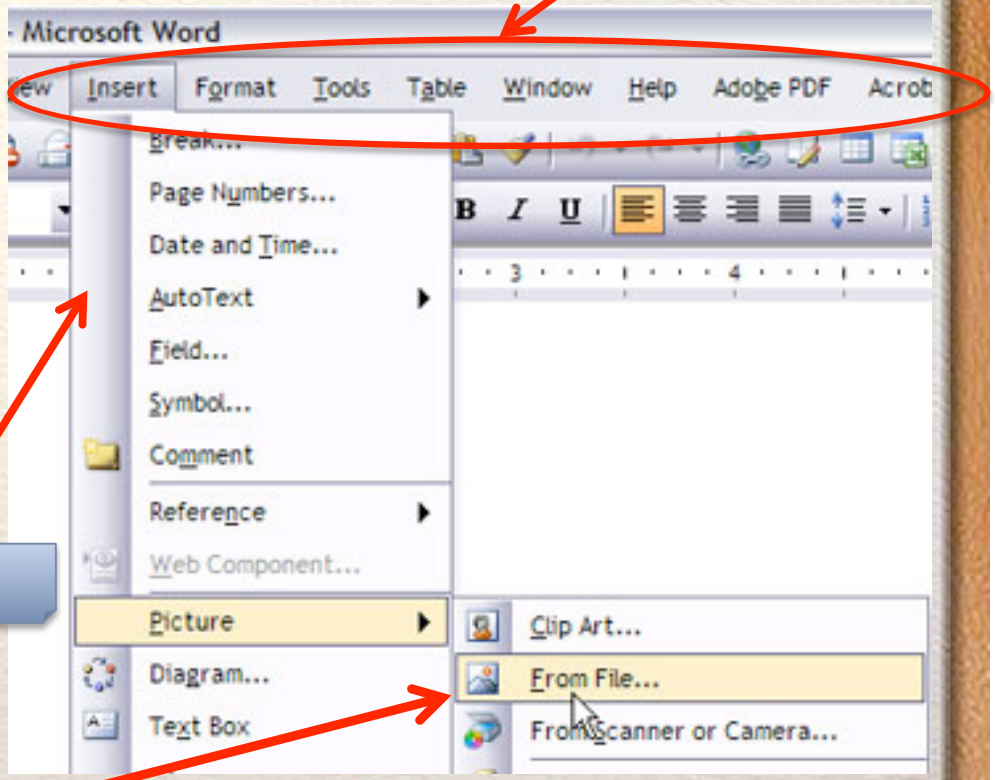
More Info

- More information about dialogs of all types is at:

- http://www.pythonware.com/library/tkinter/introduction/standard-dialogs.htm

Menus





Menu bar

Menu

Cascade (sub) menu

Mac OSX

Windows

Adding Menus

- A menu is simply another type of widget.

```
# create a toplevel menu
menubar = Menu(root)
```

The menubar is a container for Menus

```
# create a pulldown menu, and add it to the menu bar
filemenu = Menu(menubar)
```

Create a single menu

```
filemenu.add_command(label="Open", command=hello)
```

Call the hello function when the Open menu option is chosen

```
filemenu.add_separator()
```

Add a line separator in the menu

```
filemenu.add_command(label="Exit”,command=root.destroy)
```

Call the root.destroy function when the Exit menu option is chosen

```
menubar.add_cascade(label="File", menu=filemenu)
```

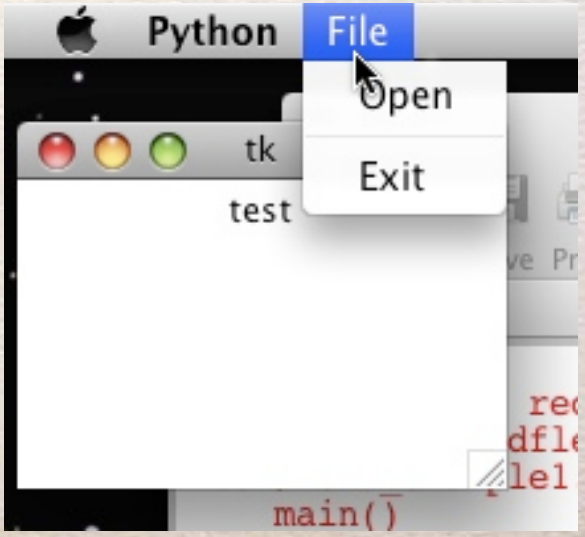
Add the filemenu as a menu item under the menubar

```
root.config(menu=menubar)
```

Tell the root window to use your menubar instead of default

Adding Menus

```
# create a toplevel menu
menubar = Menu(root)

# create a pulldown menu, and add it to the menu bar
filemenu = Menu(menubar)
filemenu.add_command(label="Open", command=hello)
filemenu.add_separator()
filemenu.add_command(label="Exit",command=root.destroy)
menubar.add_cascade(label="File", menu=filemenu)
root.config(menu=menubar)
```



See : MenuExample1.py

Adding Sub-Menus

Adding sub-menus, is done by adding a menu to another menu instead of the menubar.

```
# Create another menu item named Hello
helloMenu = Menu(menubar)
helloMenu.add_command(label="Say hello", command=hello)
menubar.add_cascade(label="Hello", menu=helloMenu)

# Create a submenu under the Hello Menu
subHello = Menu(helloMenu) # My parent is the helloMenu
subHello.add_command(label="English", command=hello) # Menu Item 1
subHello.add_command(label="Spanish", command=hello) # Menu Item 2
subHello.add_command(label="Chinese", command=hello) # Menu Item 3
subHello.add_command(label="French", command=hello)  # Menu Item 4

# Add sub menu into parent with the label International Hello
helloMenu.add_cascade(label="International Hello", menu=subHello)
```

References

- http://epydoc.sourceforge.net/stdlib/Tkinter.Pack-class.html#pack
- http://effbot.org/tkinterbook
- http://www.pythonware.com/library/tkinter/introduction/

If you don't get it, try reading these links! Good stuff!

Backup Slides

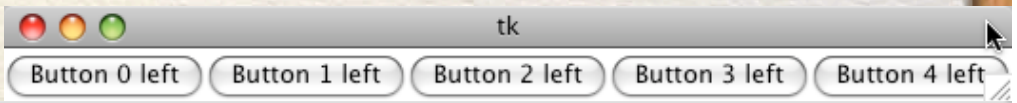
- A discussion of the Pack layout manager follows

Layout management

- You've been using one – the packer is called when you pack()
- pack can have a side to pack on:
  - myWidget.pack(side=LEFT)
  - this tells pack to put this widget to the left of the next widget
  - Let's see other options for pack at:
  - http://epydoc.sourceforge.net/stdlib/Tkinter.Pack-class.html#pack

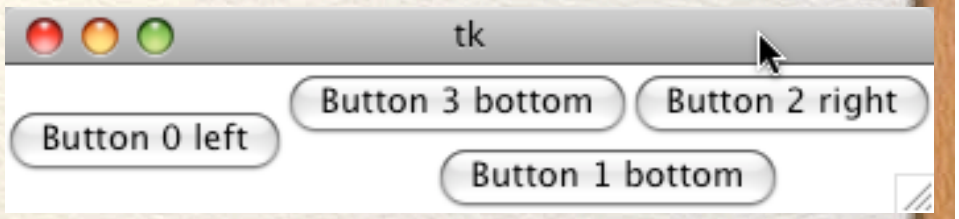
Pack Examples

```
#pack_sample.py
from Tkinter import *

# Hold onto a global reference for the root window
root = None
count = 0 # Click counter

def addButton(root, sideToPack):
    global count
    name = "Button "+ str(count) +" "+sideToPack
    button = Button(root, text=name)
    button.pack(side=sideToPack)
    count +=1

def main():
    global root
    root = Tk()  # Create the root (base) window where all widgets go
    for i in range(5):
        addButton(root, TOP)
    root.mainloop() # Start the event loop

main()
```



Pack Examples

```
#pack_sample.py
from Tkinter import *

# Hold onto a global reference for the root window
root = None
count = 0 # Click counter

def addButton(root, sideToPack):
    global count
    name = "Button "+ str(count) +" "+sideToPack
    button = Button(root, text=name)
    button.pack(side=sideToPack)
    count +=1

def main():
    global root
    root = Tk()  # Create the root (base) window where all widgets go
    addButton(root, LEFT) # Put the left side of the next widget close to me
    addButton(root, BOTTOM) # Put bottom of next widget close to me
    addButton(root, RIGHT) # Put right of next widget close to me
    addButton(root, BOTTOM) # Put bottom of next widget close to me
    root.mainloop() # Start the event loop

main()
```

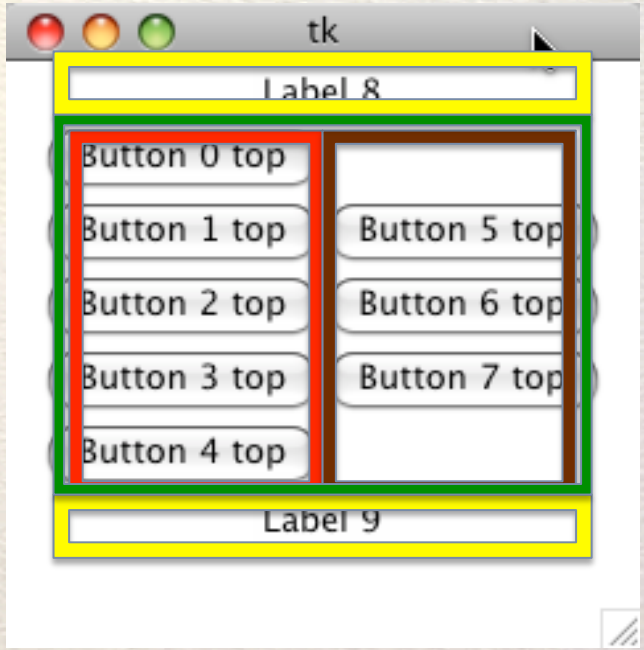


Packing Frames

- Usually you cannot get the desired look with pack unless you use Frames
- Frame are widgets that hold other widgets. (Frames are parents).
- Usually root has Frames as children and Frames have widgets or more Frames as children.

Packing Frames

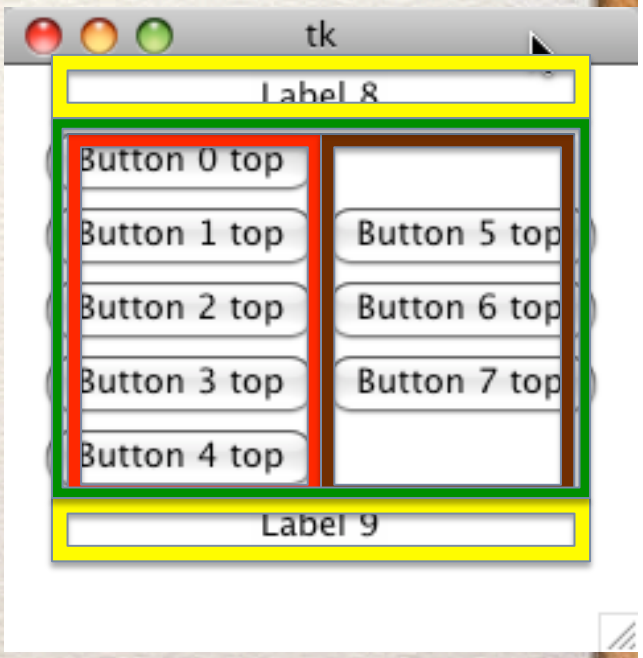
- Lets say you want this GUI



- Lets look at the frames

Packing Frames

- You know how to create any one area already. For example if I said create a window with a list of buttons arranged vertically you would do this:



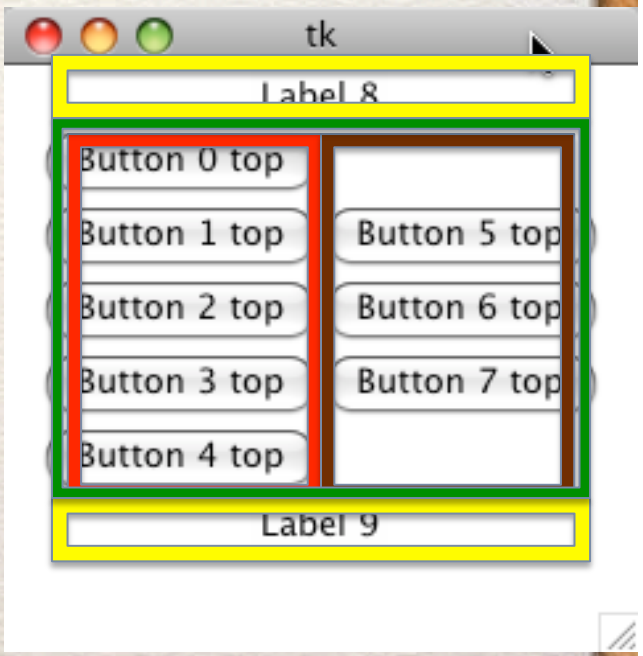
- addButton(root, TOP)
- addButton(root, TOP)
- addButton(root, TOP)
- addButton(root, TOP)
- addButton(root, TOP)

Packing Frames

- To do that with a Frame you just do this instead:

Create the frame like any other widget!

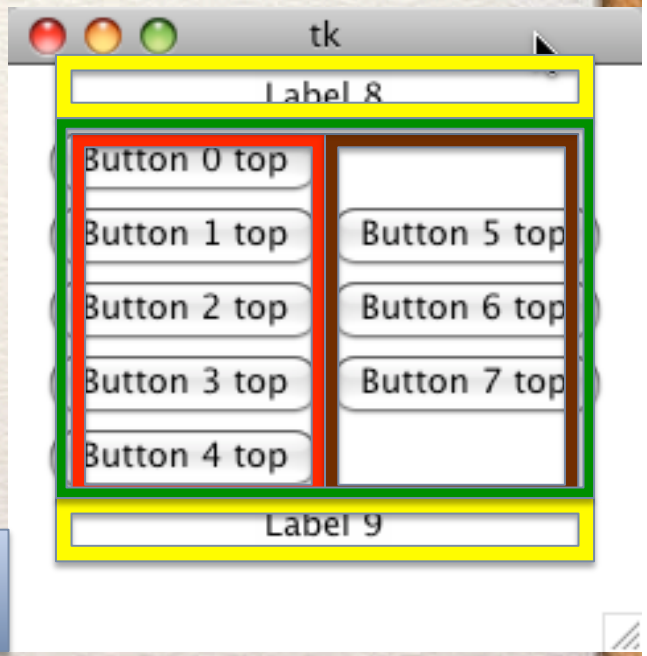
- frame1 = Frame(root)
- addButton(frame1 , TOP)
- addButton(frame1 , TOP)
- addButton(frame1 , TOP)
- addButton(frame1 , TOP)
- addButton(frame1 , TOP)



- Now you can treat the frame as one big widget!

Packing Frames

- To do that with a Frame you just do this instead:
- Now, assuming you created the frames already:
- redFrame.pack(side=LEFT)
- brownFrame.pack(side=LEFT)
- topYellow.pack(side=TOP)
- green.pack(side=TOP)
- bottomYellow.pack(side=TOP)



Who is the parent of the red and brown frames?

Ans: The green frame!