Tkinter – GUIs in Python

Dan Fleck

CS112

George Mason University

NOTE: This information is not in your textbook!
See references for more information!

What is it?

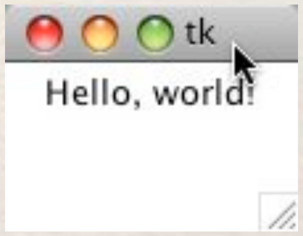
- Tkinter is a Python interface to the Tk graphics library.
  - Tk is a graphics library widely used and available everywhere
- Tkinter is included with Python as a library. To use it:
  - import * from Tkinter
    - or
  - from Tkinter import *

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.

First Tkinter Window

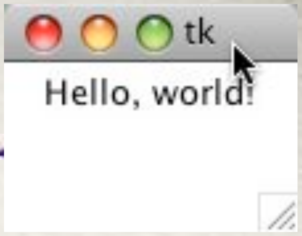
```
# File: hello1.py
from Tkinter import *
root = Tk()  # Create the root (base) window where all widgets go
w = Label(root, text="Hello, world!") # Create a label with words
w.pack() # Put the label into the window
root.mainloop() # Start the event loop
```



```
Hello, world!
```

Coming up: Explain the code

Explain the code



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

root = Tk()

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

w.pack()

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 and display. Without calling pack 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

- We haven't discussed objects, but in graphical programming we will use them.
- An int is a data type that holds a number and allows you to do things to it (add, subtract, etc…)
- An class is a CUSTOM data type that holds information and defines operations you can do to it

Classes and objects

- A class is the definition of a something or the "blueprint"
- An object is an instantiation of that class.
- For example:

Class

3 objects of class BMW CS

Objects

- Again… Objects combine *data* and *operations*
- *For example, you could create a Car class that has:*
  - *data – amount of gas in tank, odometer reading, year built, etc…*
  - *operations – start car, apply brakes, start windshield wipers, etc…*

Do all objects of class Car have the same data values?
No! Amount of gas in the tank is different for each object

Tkinter objects

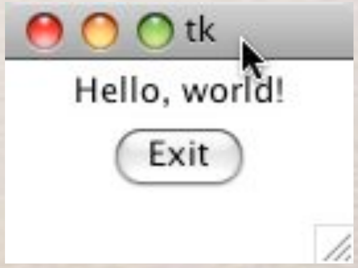
- Label is a class, w is an object
  - w = Label(root, text="Hello, world!")
  - Call the "pack" operation:
    - w.pack()
    - Hint: An operation is just a function… nothing more, nothing less.. it is just defined inside the class to act upon the object's current data.

      Objects usually hide their data from anyone else and let other programmers access the data only through operations. (This is an OO concept called encapsulation)

Build it (called instantiation)

More objects we can build

```
#Button1.py
from Tkinter import *

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

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

myButton = Button(root, text="Exit")
myButton.pack()

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

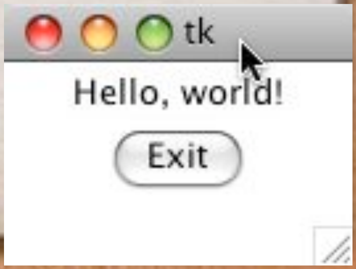


But nothing happens when we push the button! Lets fix that with an event!

Making the button do something

```
#Button2.py
from Tkinter import *

def buttonPushed():
    print "Button pushed!”

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

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

myButton = Button(root, text="Exit",command=buttonPushed)
myButton.pack()

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

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



Coming up: Making the button close the window

Making the button close the window

```
#Button3.py
from Tkinter import *
# Hold onto a global reference for the root window
root = None

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

def main():
    global root
    root = Tk()  # Create the root (base) window where all widgets go
    w = Label(root, text="Hello, world!") # Create a label with words
    w.pack() # Put the label into the window
    myButton = Button(root, text="Exit",command=buttonPushed)
    myButton.pack()
    root.mainloop() # Start the event loop

main()
```

Need later

Close the global root window

Use the global root window

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

Coming up: Creating text entry box

Creating text entry box

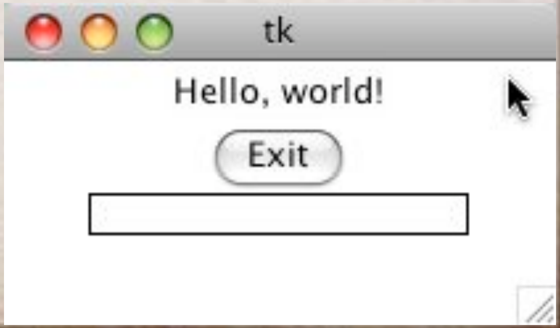
General form for all widgets:

1. # Create the widget
   widget = <widgetname>(parent, attributes…)
2. widget.pack()
   pack the widget to make it show up

```
def createTextBox(parent):
    tBox = Entry(parent)
    tBox.pack()
```

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)

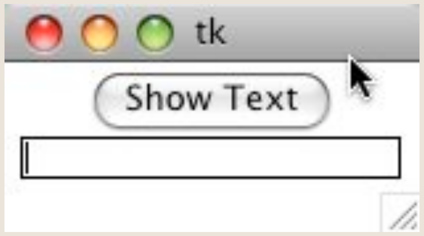
For us, this means make the entry box global so we can get the info when a button is pressed

Using a text entry box

```
#Textentrybox1.py
from Tkinter import *

# Hold onto a global reference for the root window
root = None

# Hold onto the Text Entry Box also
entryBox = None

def buttonPushed():
    global entryBox
    txt = entryBox.get()
    print "The text is:",txt

def createTextBox(parent):
    global entryBox
    entryBox = Entry(parent)
    entryBox.pack()

def main():
    global root
    root = Tk()  # Create the root (base) window where all widgets go

    myButton = Button(root, text="Show Text",command=buttonPushed)
    myButton.pack()
    createTextBox(root)
    root.mainloop() # Start the event loop

main()
```

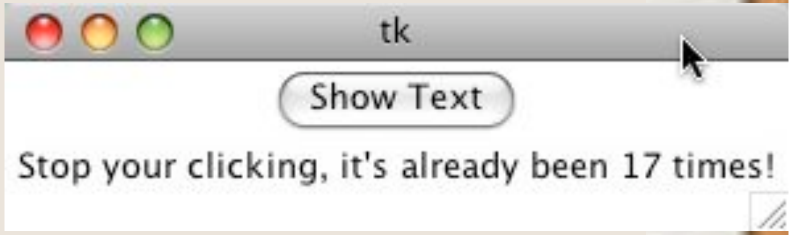
Call the get() operation on the entry box to get the text when button is pushed

Create the global entry box!



Creating a label you can change

```
#changeable_label.py
# Use a StringVar to create a changeable label
from Tkinter import *

# Hold onto a global reference for the root window
root = None

# Changeable text that will go inside the Label
myText = None
count = 0 # Click counter

def buttonPushed():
   global myText
   global count
   count += 1
   myText.set("Stop your clicking, it's already been %d times!" %(count))

def addTextLabel(root):
   global myText
   myText = StringVar()
   myText.set("")
   myLabel = Label(root, textvariable=myText)
   myLabel.pack()

def main():
   global root
   root = Tk()  # Create the root (base) window where all widgets go
   myButton = Button(root, text="Show Text",command=buttonPushed)
   myButton.pack()
   addTextLabel(root)
   root.mainloop() # Start the event loop

main()
```



Stop your clicking, it's already been 17 times!

Set the text in the label (call set method with a string actual parameter)

Create a StringVar to hold text

Link the label to the StringVar

Coming

Layout management

- You may have noticed as we pack widgets into the window they always go under the previous widget
- What if we want to get them to go side-by-side or some other place?

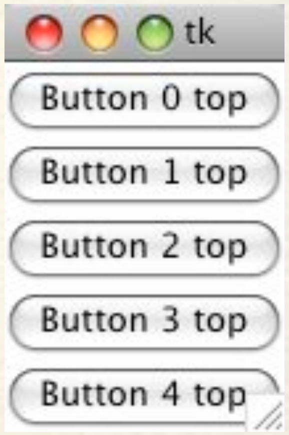
- Most windowing toolkits have layout management systems to help you arrange widgets!

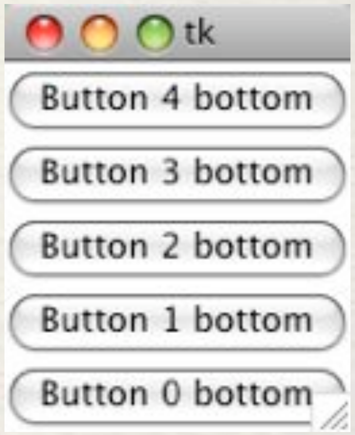
Layout management

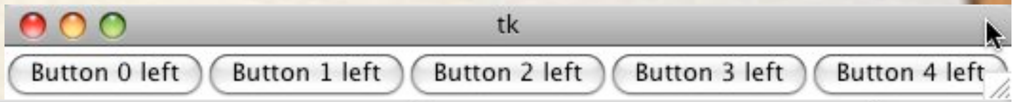
- You've been using one – the packer is called when you pack()
- pack can have a side to pack on:
  - myWidget.pack(side=LEFT)
  - this tells pack to put this widget to the left of the next widget
  - Let's see other options for pack at:
  - http://epydoc.sourceforge.net/stdlib/Tkinter.Pack-class.html#pack

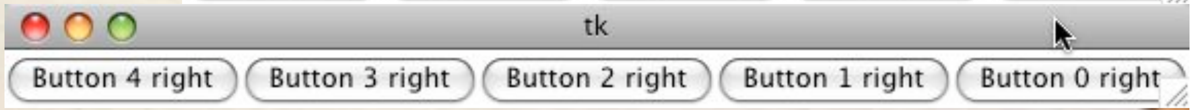
Pack Examples

```
#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()
```



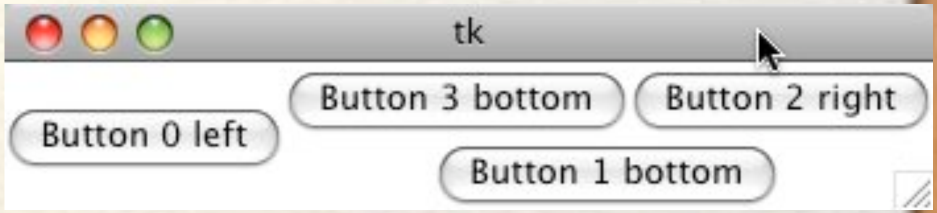






Pack Examples

```
#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()
```

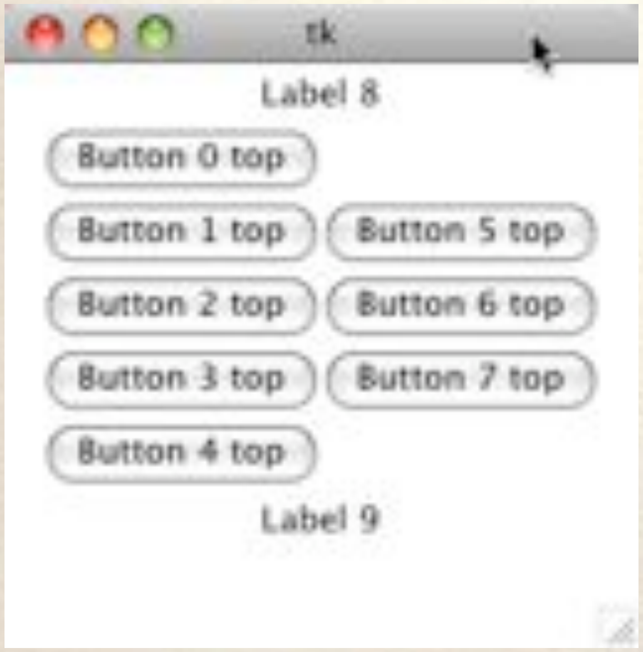


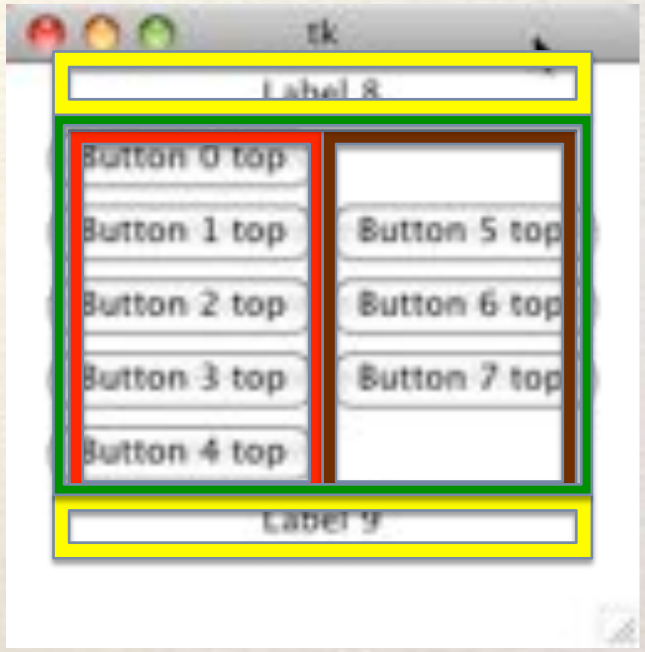
Packing Frames

- 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

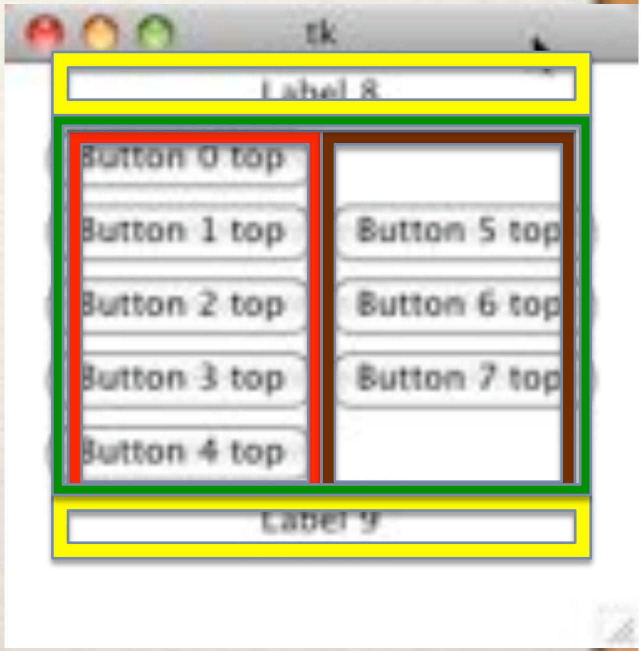




- Lets look at the frames

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:



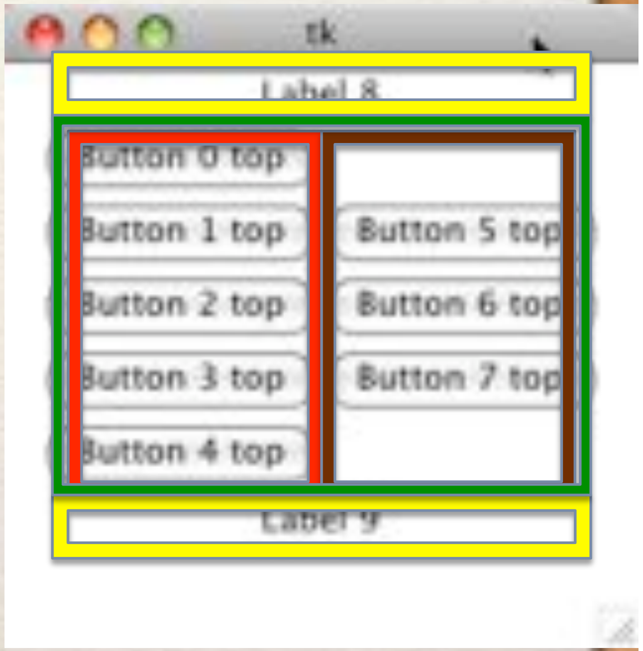
- addButton(root, TOP)
- addButton(root, TOP)
- addButton(root, TOP)
- addButton(root, TOP)
- addButton(root, TOP)

Packing Frames

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

Create the frame like any other widget!

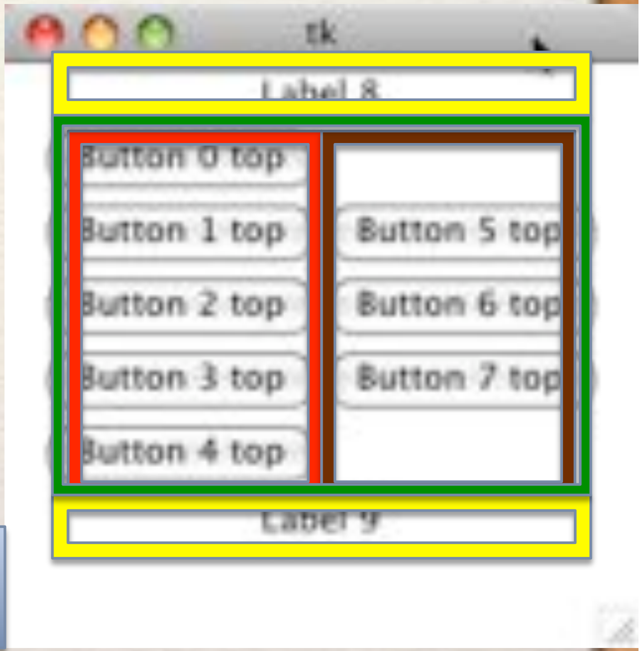
- frame1 = Frame(root)
- addButton(frame1 , TOP)
- addButton(frame1 , TOP)
- addButton(frame1 , TOP)
- addButton(frame1 , TOP)
- addButton(frame1 , TOP)



- 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!

Coming up: Other geometry managers

Other geometry managers

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

- 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
- In this class we will only use the pack manager, but for very complicated GUIs you probably want the grid manager

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

Coming up: Adding Menus

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)
```

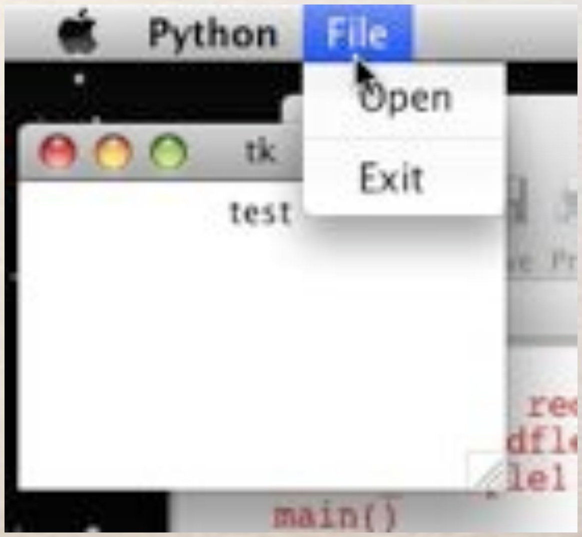
Add the filemenu as a menu item under the menubar

```
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)
```



Coming up: Adding Sub-Menus

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)
```

Coming up: Showing Images

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.pack() # 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).

A canvas was the widget that Turtle Graphics uses to draw on!

```
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():
    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()
```

Step 2: Write functions to handle events. Notice: event object automatically passed into event handler!

Step 1: Bind events to functions

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 will 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

Coming up: Common problem!

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!
```

Coming up: How mouse-clicks work: the event loop

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 outpus 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…

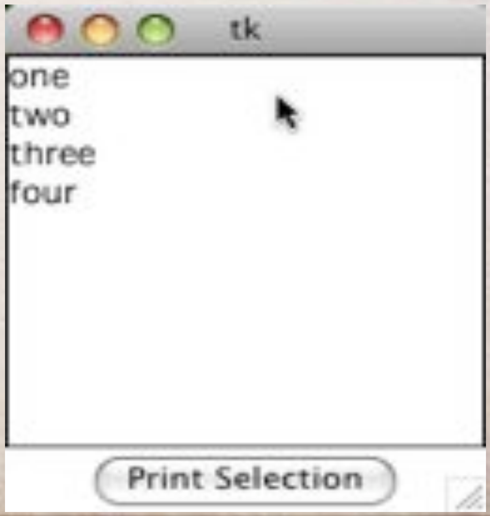
EVENT-BASED
User clicked “answer q1 button”
User clicked “answer q3 button”
User clicked “answer q2 button”
Etc…

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. 401K Lab — Batch

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



Message Dialog Boxes

- A dialog box is a small modal window that lets you ask a question, show a message or do many other things in a separate window from the main window (File->Open usually opens a dialog box)
- 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.

Coming up: Question Dialog Boxes

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!

Coming up: File Dialog Boxes

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
```

Coming up: More Info

More Info

- More information about dialogs of all types is at:
- http://www.pythonware.com/library/tkinter/introduction/standard-dialogs.htm

Coming up: Adding a title to your window

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()

Coming up: References

Fixing some problems

- My widgets don’t show up!
  - did you pack everything? and the frames to?
- How to “see” your frame:
  - x = Frame(parent, bg=‘green’, borderwidth=10)
  - Lots of colors work
- My stuff shows up in the middle, not on the left or right
  - Use anchor… next slide

Fixing some problems

- My stuff shows up in the middle, not on the left or right
  - Use anchor… next slide
    - pack(side=TOP, anchor=‘e’) # Anchor EAST
    - Anchor says where should this widget go if I have a lot more space!

References

- http://www.ibm.com/developerworks/library/l-tkprg/index.html#h4
- 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!

End of presentation