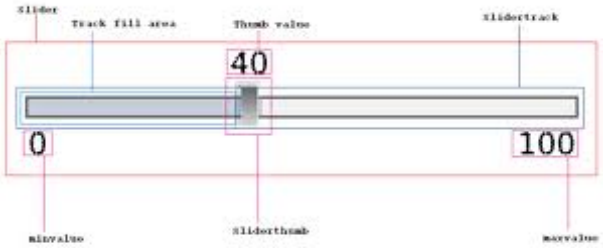
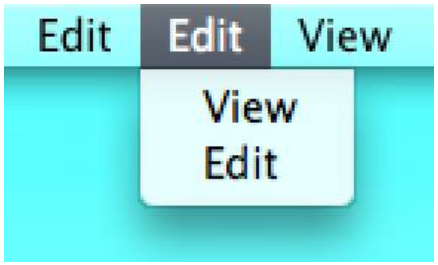
CS 1
Introduction to Computer Programming

"The Lost Lecture": December, 2015

Tkinter Widgets





Tkinter Widgets

- So far have spent a lot of time with the Tkinter **`Canvas`** widget
  - Very good for drawing simple graphics, as long as they're lines, circles, ovals, rectangles, etc.
- Tkinter also provides many other kinds of widgets for building more complex GUIs
  - Buttons, labels, text fields, menus, lists, etc.
  - Simple mechanisms for grouping and laying out widgets

Message Boxes

- Tkinter provides several prebuilt features
- Display a simple informational message:

```
import tkMessageBox
tkMessageBox.showinfo('Hi from Tkinter!',
    'Hello world!\n\nThis is a message box.')
```

- Output:

Message Boxes (2)

- **`tkMessageBox`** module provides other functions as well
- Display warnings or errors:

```
tkMessageBox.showwarning('HAL Warning',
    "I'm sorry Dave. I'm afraid I can't do that.")
# also showerror() function
```

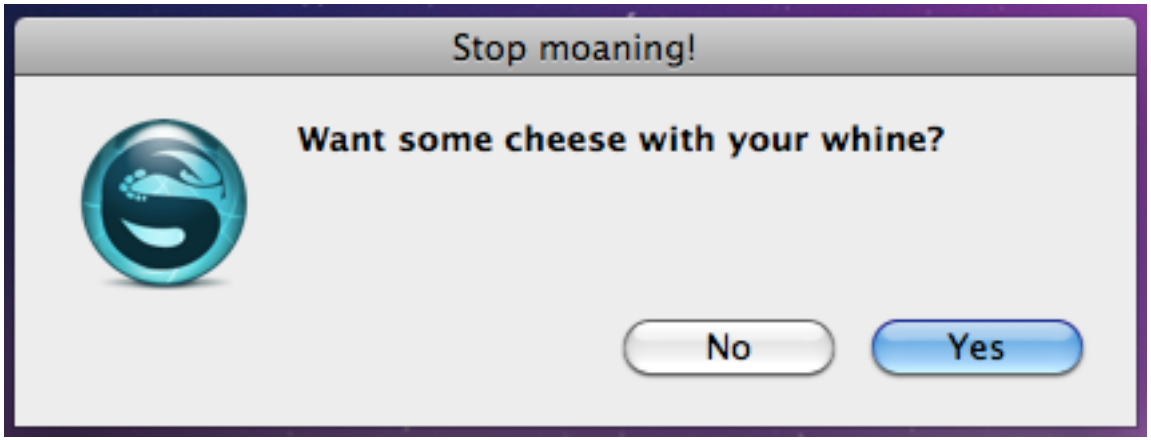
- Output:

Message Boxes (3)

- Can also perform simple interactions:

```
tkMessageBox.askyesno('Stop moaning!',
    'Want some cheese with your whine?')
```

- Output:



  - Returns **`True`** if "Yes" is clicked, or **`False`** if "No" is clicked

Message Boxes (4)

- Other functions with different options:
  - **`askokcancel`** – "Ok" and "Cancel"
  - **`askretrycancel`** – "Retry" and "Cancel"
  - All return **`True`** if "Yes" is clicked, or **`False`** if "No" is clicked
- Also:
  - **`askquestion`** – "Yes" and "No" options
  - Returns string **`'yes'`** if "Yes" is clicked, or **`'no'`** if "No" is clicked
- Message boxes also have keyword options
  - Choose what icon to use, default button, etc.

Buttons

- Buttons are a simple example of a widget
  - Very widely used user-interface component!
- Example:

```
from Tkinter import *
root = Tk()
btn = Button(root, text='Click Me!')
btn.pack()
root.mainloop()
```

- A very simple application:

Buttons (2)

- Buttons have a **`command`** option
  - Set this to a function to invoke when the button is clicked
- Example:

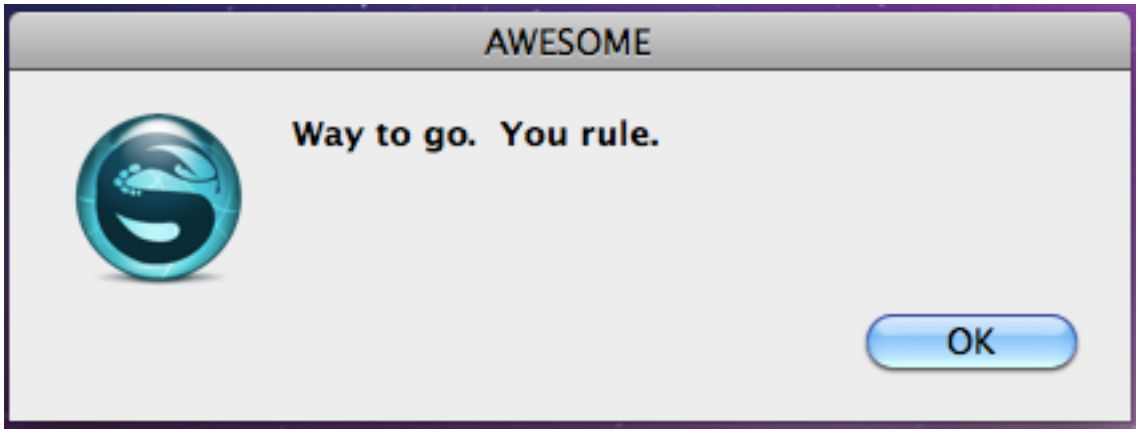
```
def clicked():
    tkMessageBox.showinfo('AWESOME',
        'Way to go.  You rule.')


...
btn = Button(root, text='Click Me!',
  command=clicked)
```

Buttons (3)

- When app is run:

- When you click the button:



Buttons (4)

- Can also set widget options using []

```
def clicked():
    tkMessageBox.showinfo('AWESOME',
        'Way to go.  You rule.')
    btn['text'] = 'Thank you!'


...
btn = Button(root, text='Click Me!')
btn['command'] = clicked
```

Images

- Tkinter supports "bitmap images" and "photo images"
  - Bitmap image: black-and-white image, in **`.xbm`** format
  - Photo image: color image, in **`.gif`**, **`.pgm`** or **`.ppm`** formats
- Example:

```
img = PhotoImage(file='Alien.gif')
```

  - Argument is path to the image file
  - Must specify **`file=`** or will fail with an error!

Images (2)

- Can use images in several different places
- Can display in a **`Canvas`**

```
id = c.create_image(x, y, image=...)
```

- Can use in buttons, check-boxes, etc.

```
img = PhotoImage(file='Alien.gif')
b = Button(root, image=img)
```

  - Displays:

Multiple Widgets

- Frequently want multiple widgets…

```
btn1 = Button(root, text='Click Me!')
btn1.pack()
btn2 = Button(root, text='No, Me!')
btn2.pack()
```

- Shows:

- The **`pack()`** method places a widget within its container

Multiple Widgets (2)

- The `pack()` method invokes the Pack layout manager
  - A special component that determines how widgets should be positioned
  - Pretty simple layout manager
- Also a Grid layout manager, which allows more specific placement of components
  - Components are placed on a 2D grid (surprise!)
  - Invoked via the `grid()` call
- Also a Place layout manager
- If you don't invoke a layout operation on a widget, it won't show up in your GUI!

Multiple Widgets (3)

- As with widget constructors, the **`grid()`** method takes keyword arguments:
  - **`row`** = row to place widget in (zero-indexed)
  - **`column`** = column to place widget in
  - **`rowspan`** = how many rows the widget spans
  - **`columnspan`** = how many columns the widget spans
  - Other options to specify how extra space should be distributed to widgets

Multiple Widgets (4)

- Lay out our buttons horizontally instead:

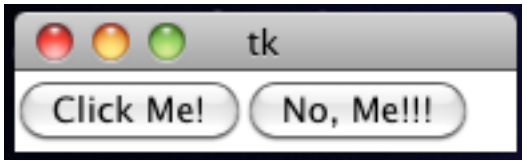
```
from Tkinter import *

root = Tk()

btn1 = Button(root, text='Click Me!')
btn1.grid(row=0, column=0)
btn2 = Button(root, text='No, Me!')
btn2.grid(row=0, column=1)

root.mainloop()
```

- Shows:



Check-Buttons

- Can create **`Checkbutton`** widgets:

```
from Tkinter import *
root = Tk()
cbox = Checkbutton(root,
     text='Disable homicidal computer')
cbox.grid()
root.mainloop()
```

- Shows:

Check-Buttons (2)

- We have our user interface… but how to read out the value of the check-button?
- **`Checkbutton`** also has **`command`** value
  - Is called every time check-box state changes
  - Doesn't receive any arguments, so can't tell exactly what happened
- Tkinter uses *control variables* to associate UI widgets with the values they hold

Control Variables

- Update our code:

```
cbox_val = IntVar()  # The control variable

def cbox_clicked():
    if cbox_val.get() == 1:
        tkMessageBox.showinfo('HAL9000',
            'evil module disabled')
    else:
        tkMessageBox.showwarning('HAL9000',
            'evil module enabled')

cbox = Checkbutton(root,
    text='Disable homicidal computer',
    variable=cbox_val, command=cbox_clicked)
```

Control Variables (2)

- Three kinds of control variables:
  - **`IntVar`** – initial value is 0
  - **`DoubleVar`** – initial value is 0.0
  - **`StringVar`** – initial value is empty string
  - Specify an initial value with **`value`** argument:
    - `s = StringVar(value='hello')`
- All control variables have two methods:
  - **`get()`** returns current value of the variable
  - **`set(val)`** sets the value of the variable

Control Variables (3)

- Control variables encourage a *separation of concerns* within graphical applications
  - Divide code into non-overlapping functional units
- A common approach:
  - Create a *model* class that encapsulates all state for the GUI application
  - Create a *view* class that sets up and manages the actual graphical widgets
  - A *controller* class handles events from the view (e.g. button-press commands), and manipulates the model's state accordingly
  - Called the *Model-View-Controller* (MVC) design pattern

Text Fields

- Can use the **`Entry`** widget to input a single line of text
  - Has a **`textvariable`** option for specifying the control variable
  - Control variable must be of type **`StringVar`**
  - **`StringVar`** has **`get()`** and **`set()`** methods
- Other useful options:
  - **`width`** = width of text-field in characters
  - **`show`** = character to show when text is entered
    - e.g. set to `'*'` for password fields
  - **`justify`** = left/right justify or center text

Text Fields (2)

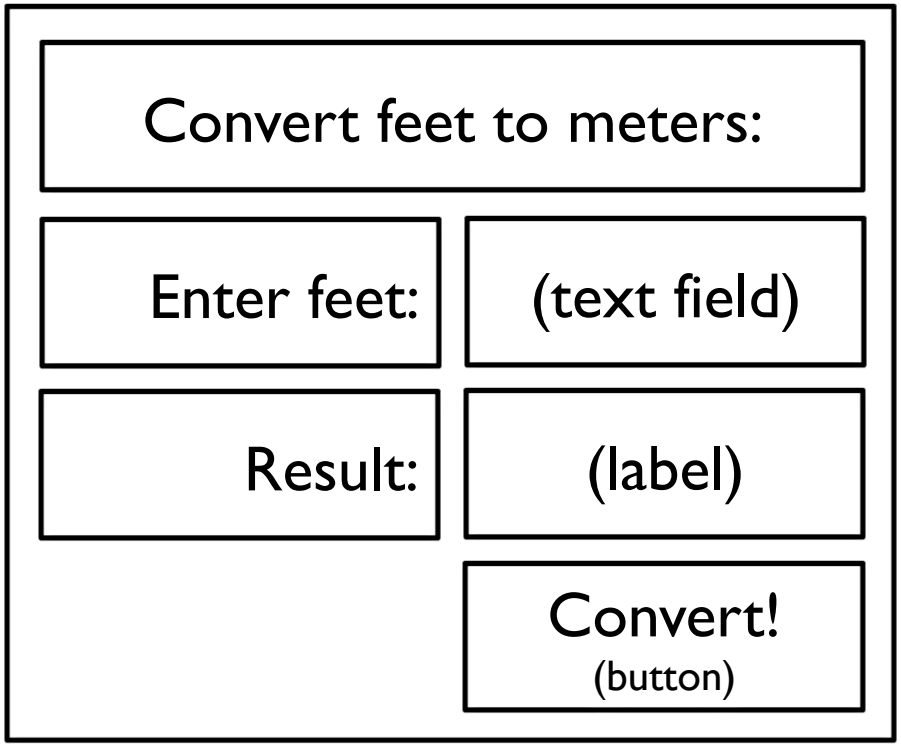
- Use the **`Label`** widget to show read-only text in a user interface
  - Specify label's contents with the **`text`** value
  - (Also has a **`textvariable`** option if you don't want to use **`text`** value)
- Other useful options:
  - **`width`** = width of text-field in characters
  - **`justify`** = left/right justify or center text
  - **`wraplength`** = specify number of characters to appear in each line of the label

Example: Units Converter

- Program to convert feet to meters
  - User enters a distance in feet
  - Program computes distance in meters
- Very helpful to create a UI mock-up:
  - Design the user interface before you build it
  - (Just like you should do with programs…)
- Generally, programmers design very bad UIs
  - Tend to be overcomplicated and difficult to use
- Some people focus *entirely* on UI design
  - User-experience, human-computer interaction

Units Converter UI-Mockup

- A simple user-interface mockup:



- (I'm not a UI designer…)

Units Converter Usage

- Usage:
  - User enters a distance in feet, then presses a button to actually convert the value
- State:
  - Will need a control variable for **`Entry`** field that receives the number of feet
    - Note: It must be a **`StringVar`**…
  - We'll also use a control variable to display the number of meters
    - Use a **`StringVar`** here too

Units Converter Implementation

- Program isn't very complicated, so we'll just use global variables
- Code:

```
from Tkinter import *
import tkMessageBox

root = Tk()

# Our state variables for the app
cvt_from = StringVar()
cvt_to = StringVar()
...
```

Units Converter UI Setup

- Next, set up the user interface
  - (This code tends to be tedious to write…)

```
lbl = Label(root,
     text='Convert from feet to meters:')
lbl.grid(row=0, column=0, columnspan=2)

from_lbl = Label(root, text='Enter feet:')
from_lbl.grid(row=1, column=0)

from_entry = Entry(root,
   textvariable=cvt_from)
from_entry.grid(row=1, column=1)
...
```

Units Converter UI Setup (2)

- Continued:

```
to_lbl = Label(root, text='Result:')
to_lbl.grid(row=2, column=0)

result_lbl = Label(root,
    textvariable=cvt_to)
result_lbl.grid(row=2, column=1)

# TODO:  Need to write do_convert...
convert_btn = Button(root,
    text='Convert!', command=do_convert)
convert_btn.grid(row=3, column=1)

root.mainloop()
```

Doing the Conversion

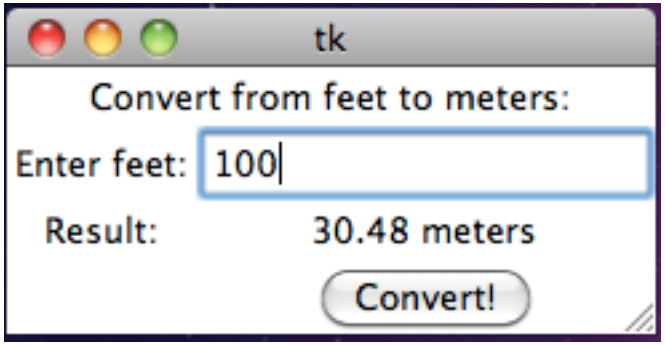
- Finally, implement **`do_convert`** function
- Put it after the command variables, and before the UI setup:

```
def do_convert():
    feet_val = float(cvt_from.get())
    meters_val = feet_val * 0.3048
    cvt_to.set('%s meters' % meters_val)
```

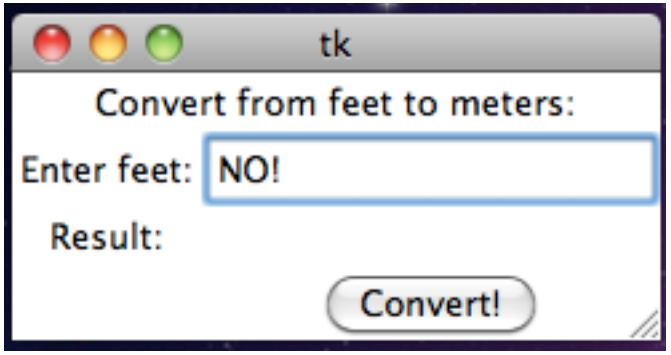
- Big assumption?
  - Users will always enter valid numbers!
  - Will deal with this in a bit.

Running the Application

- Application works great!



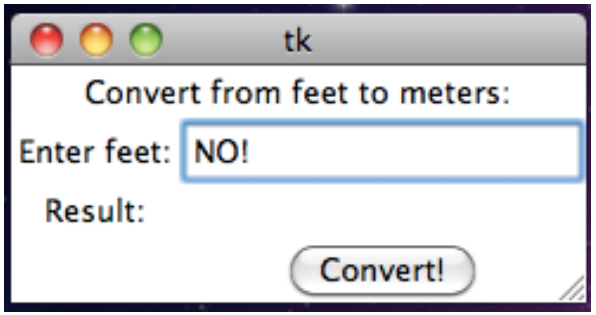
- That is, as long as you enter a number
  - In console, see an error message:




```
ValueError: could not convert string
to float: NO!
```

Handling Bad Inputs

- Applications should always expect bad input from users



  - (Also, design the user interface to minimize opportunities for bad input…)
- Problem occurs in **`do_convert`**:

```
def do_convert():
    feet_val = float(cvt_from.get())
    meters_val = feet_val * 0.3048
    cvt_to.set('%s meters' % meters_val)
```

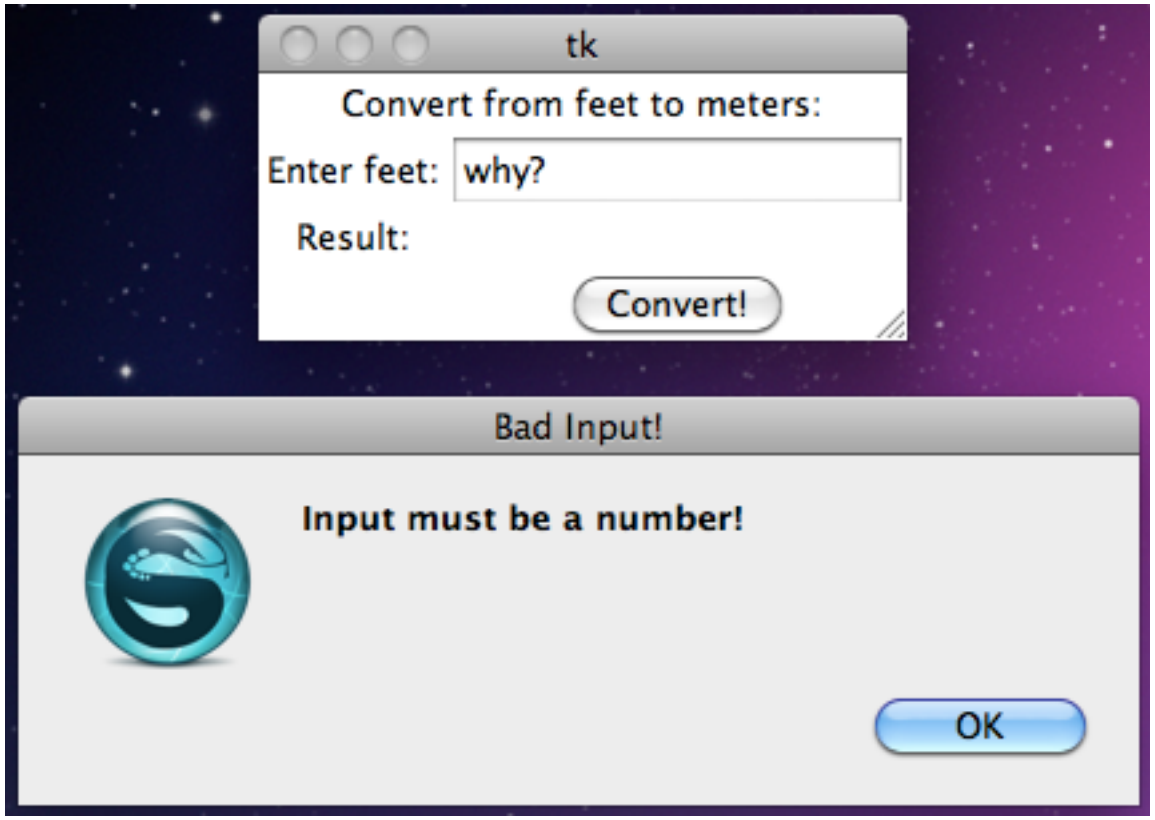
Handling Bad Inputs (2)

- When **`float()`** can't convert a value, it raises a **`ValueError`** exception
- This is easy enough to handle:

```
def do_convert():
    cvt_to.set('')
    try:
        feet_val = float(cvt_from.get())
        meters_val = feet_val * 0.3048
        cvt_to.set('%s meters' % meters_val)
    except ValueError:
        tkMessageBox.showerror(...)
```

Handling Bad Inputs (3)

- Now, user interface responds more gracefully to bad input:



Combo-Boxes

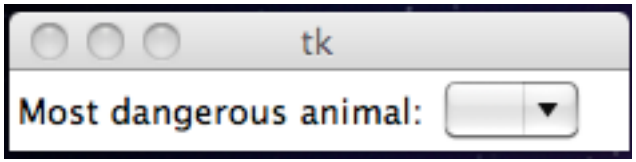
- A common user-interface interaction:
  - Allow the user to choose one of a fixed set of options
  - Often provided by a *drop-down combo-box*
- Unit-conversion example: let user choose one of a number of conversions
  - feet → meters, meters → feet, miles → km
  - furlongs → feet, light-years → beard-seconds
  - Fahrenheit → Celsius, kilograms → slugs

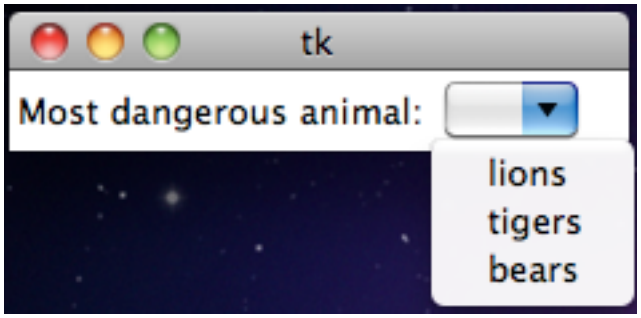
Combo-Boxes (2)

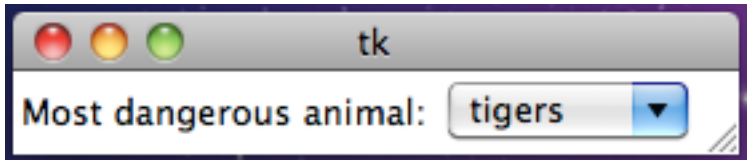
- Tkinter has an **`OptionMenu`** widget

```
choice_var = StringVar()
opt_menu = OptionMenu(root, choice_var,
    'lions', 'tigers', 'bears')
```

- Displays as:







Combo-Boxes (3)

- Can separate out the option-list if needed

```
choice_var = StringVar()
choices = ['lions', 'tigers', 'bears']
opt_menu = OptionMenu(root, choice_var,
    *choices)
```

  - Effectively inserts contents of **`choices`** as remaining args to **`OptionMenu`** constructor
  - Must be the last argument
- Can use this to set the option-menu's current state:

```
choice_var = choices[0]
```

Combo-Boxes (4)

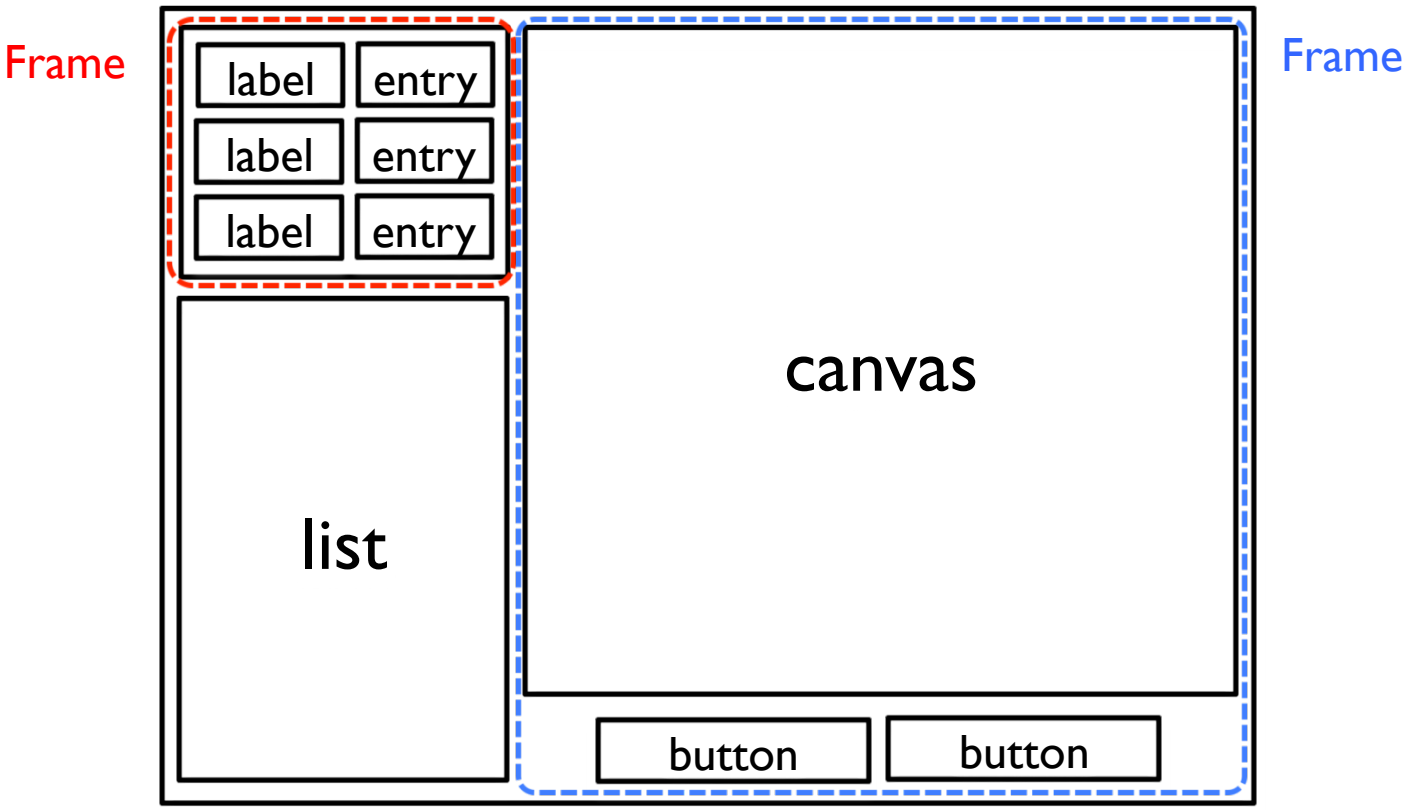
- **`OptionMenu`** widgets aren't very smart
- Don't have a **`command`** option to handle when the user changes its value
  - Basically can only query the **`OptionMenu`**'s command variable for its current state
- Also, can't have separate "display values" and "choice values"
  - Can only display the actual values specified to the **`OptionMenu`**'s constructor

Groups of Widgets

- Sometimes want more sophisticated user-interface layouts
- The **`Frame`** widget simply holds other widgets
  - Doesn't have any visual appearance
  - Just groups widgets, and controls their layout
- Also a **`LabelFrame`** widget
  - Draws a labeled border around the widgets it contains

Groups of Widgets (2)

- Example: want a layout like this:



Enabling/Disabling Widgets

- Most widgets (buttons, lists, etc.) have a state value to enable/disable the widget
  - "Disabled" means the widget is still displayed, but the user can't actually interact with it
  - Usually shown as grayed-out
- Example:

```
cbox = Checkbutton(
    text='Disable homicidal computer')
cbox['state'] = DISABLED
```

  - To reenable, set to the constant **NORMAL**

File Chooser

- Last component for today: the file chooser
- Frequently want to open a file for reading, or save data to an output file
- Like many frameworks, Tkinter provides a built-in file-chooser component
  - `import tkFileDialog`
- Two functions:
  - `tkFileDialog.askopenfilename(...)`
  - `tkFileDialog.asksavefilename(...)`

File Chooser (2)

- Main difference between **`askopenfilename()`** and **`asksavefilename()`**:
  - "Open file" requires that the file already exists
  - "Save file" allows you to "create a new file" (i.e. select a file that doesn't exist)
- Can specify options to these functions:
  - **`title`** = title of file-chooser window
  - **`defaultextension`** = default extension to use when user doesn't specify one

Example: Open an Image File

- Want to select an image file for opening

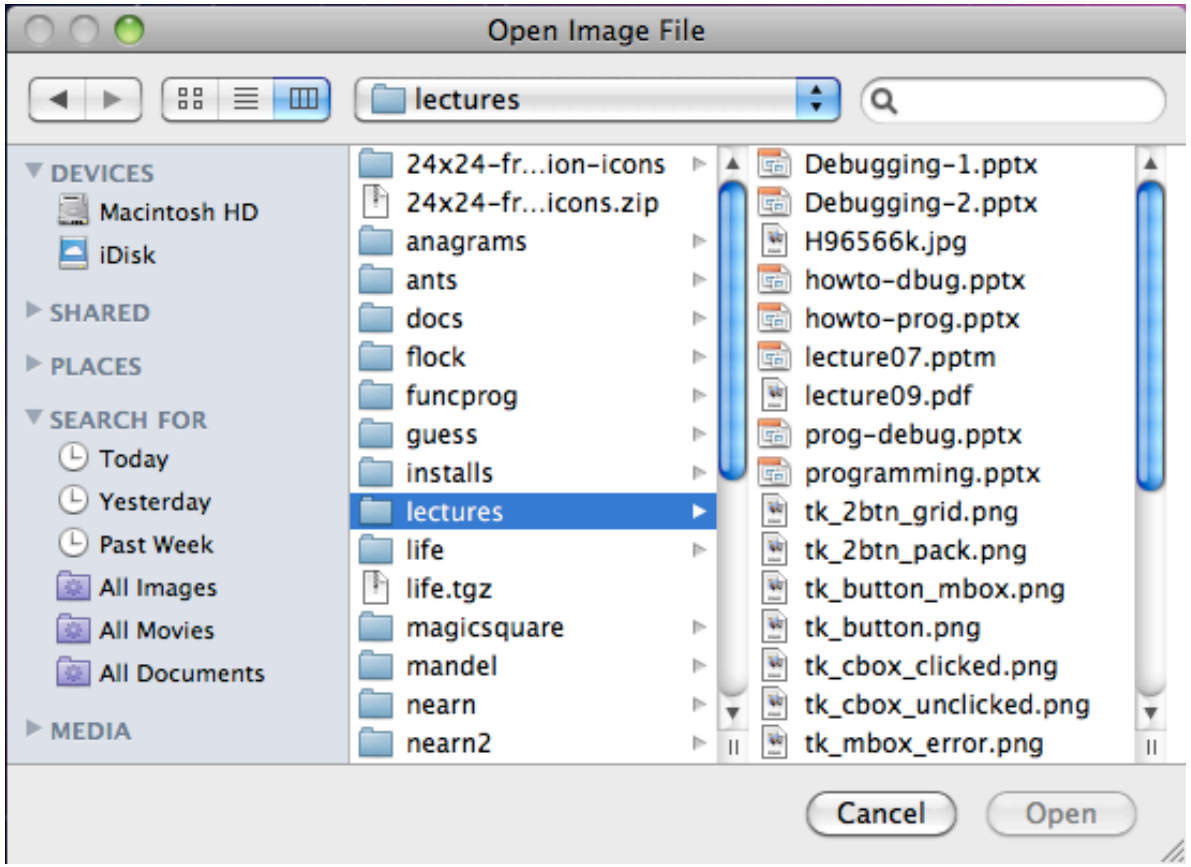
```
import tkFileDialog

path = tkFileDialog.askopenfilename(
    title='Open Image File')
print 'Selected path:  %s' % path
```

- If user selects a file, **`path`** will be the full path to the file
- If user selects "Cancel" button, **`path`** will be set to the empty string

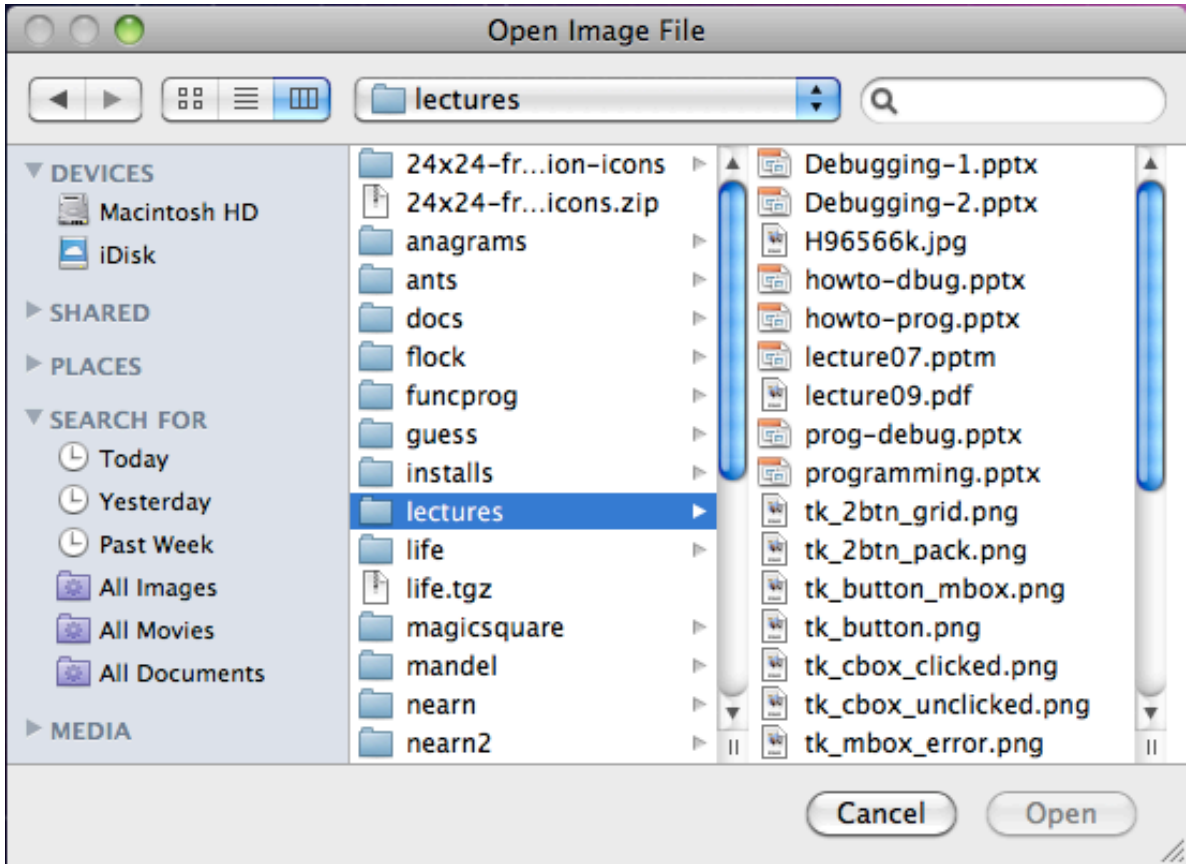
Example: Open an Image File

- Displays:



Opening an Image File (2)

- Problem: shows a lot of non-image files



Opening an Image File (3)

- Can also specify types of files the dialog should show:

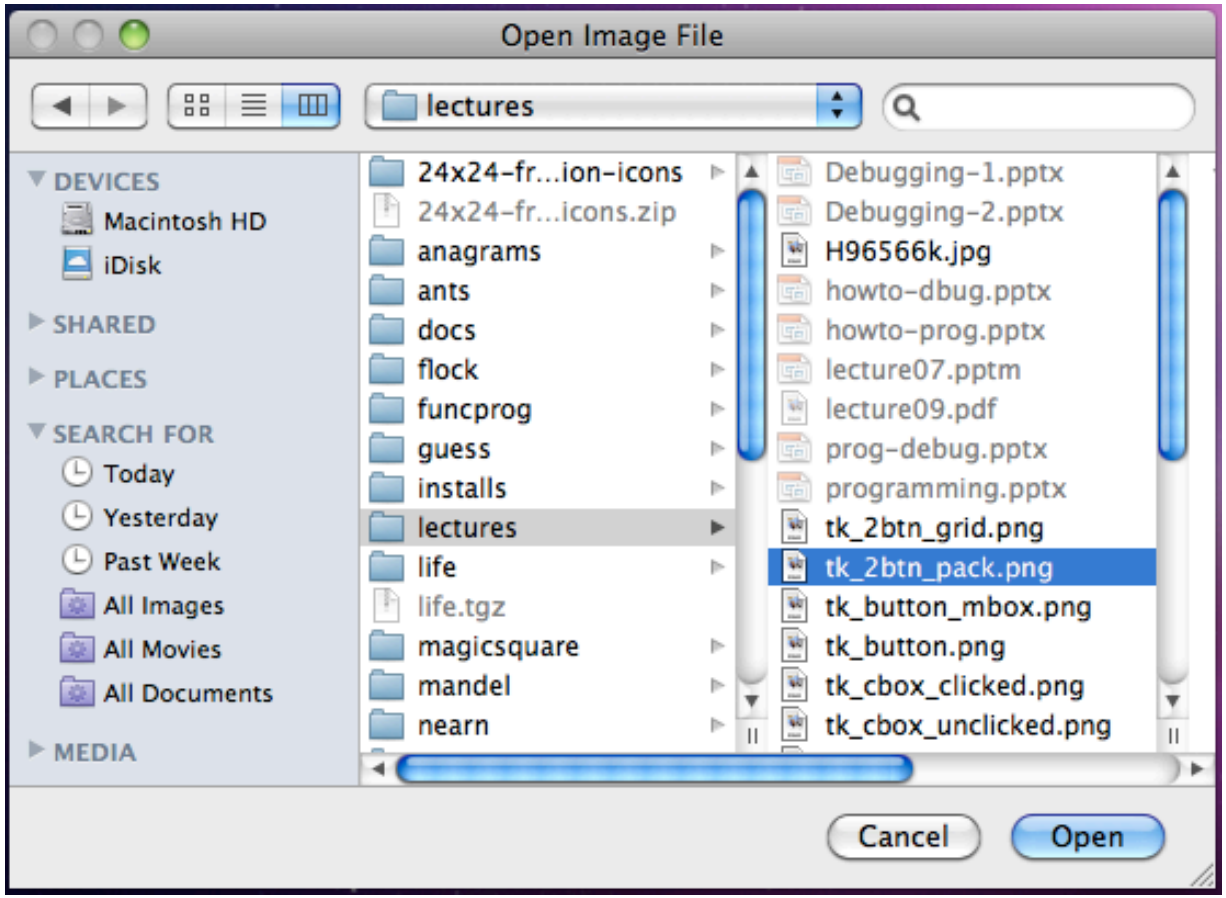
```
types = [('JPEG files', '*.jpg'),
         ( 'PNG files', '*.png'),
         ( 'GIF files', '*.gif')]
```

  - Each element of the list is a tuple specifying a label and a pattern

```
path = tkFileDialog.askopenfilename(
     title='Open Image File',
     filetypes=types)
```

Opening an Image File (4)

- Now, non-matching file types are disabled:



Other Tkinter Widgets!

- Tkinter provides many other kinds of widgets
  - Lists, menus, multi-line text areas, radio buttons, images, sliders, paned windows, etc.
- Simply too much to cover in one class
- Feel free to consult the Tkinter docs:
  - http://infohost.nmt.edu/tcc/help/pubs/tkinter/
- Write little programs to experiment with various widgets to learn more

That is all!