Outline

- Introduction & Demo
- Widgets
- Bindings
- Canvas
- Tkinter-specific topics
Python at a glance

- Interactive
- Interpreted
- Dynamically typed
- Object-oriented
- Multi-platform
Tkinter at a glance

- Object-oriented
- Multi-platform

Tkinter (Python)

Tk (Tcl)

Native (C)
A simple application

```python
from Tkinter import *
root = Tk()
root.title("Sample application")
root.mainloop()
```
Designing UIs

- Two parts
  - Graphical: *widgets*
  - Functional: *bindings*
Widgets

- Widgets are visual basic components
  - Buttons
  - Text labels
  - Check boxes
  - ...
- UIs contain multiple widgets
## Widget classes

- Button
- Canvas
- CheckButton
- Entry
- Frame
- Label
- Listbox
- Menu
- MenuButton
- Message
- RadioButton
- Scale
- Scrollbar
- Text
A simple application using widgets

```python
from Tkinter import *

def quit():
    import sys; sys.exit()

root = Tk()
lbl = Label(root, text="Press the button below to exit")
lbl.pack()
btn = Button(root, text="Quit", command=quit)
btn.pack()
root.mainloop()
```
from tkinter import *

class Example:
    def __init__(self, master):
        self.lbl = Label(master, \
                         text="Press the button below to exit")
        self.lbl.pack()
        self.btn = Button(master, \
                          text="Quit", command=self.quit)
        self.btn.pack()

    def quit(self):
        import sys; sys.exit()

if __name__ == "__main__":
    root = Tk()
    ex3 = Example(root)
    root.mainloop()
Geometry management

- Responsible for:
  - negotiating widget positions
  - negotiating widget dimensions
  - reacting to changes (resizing)

- Widgets have a geometry master

- Three possible managers
  - Placer
  - Grid
  - Packer
Placer geometry manager

- Simple
- Precise
- Allows positions and dimensions to be:
  - absolute
  - relative
- Supports any combination
- Seldom used
from Tkinter import *
root = Tk()

# Absolute positioning
Button(root, text="Corner").place(x=10, y=10, height=35)

# Relative positioning
Button(root, text="Centered").place(relx=0.25, rely=0.25, 
       relwidth=0.5, relheight=0.5)

root.mainloop()
Grid geometry manager

- Simple
- Especially useful for dialog boxes
- Allows widgets to be organized in complex grid layouts
- Closely resembles HTML tables
  - Each cell can hold one widget
  - Widgets can span multiple cells
  - Widget can be expanded within cells in all directions
- Not compatible with packer (infinite negotiation loop)
from Tkinter import *
root = Tk()
Label(root, text="Name").grid(row=0, sticky=W)
Label(root, text="Address").grid(row=1, sticky=W)
Entry(root, width=16).grid(row=0, column=1, sticky=W)
Entry(root, width=16).grid(row=1, column=1, sticky=W)
root.mainloop()
Packer geometry manager

- Most common
- Quickest
- Attributes portions of the space left in the master to widgets
- Complex layouts require \texttt{Frame} widgets
from Tkinter import *
root = Tk()
Button(root, text="Left").pack(side=LEFT, expand=YES, fill=Y)
Button(root, text="Top").pack(side=TOP, expand=YES, fill=BOTH)
Button(root, text="Right").pack(side=RIGHT, expand=YES, fill=X)
root.mainloop()
Tkinter class hierarchy

- Basewidget
- Pack
- Grid
- Place
- Toplevel
- Widget
- Button
- ...
Widget properties

- Set at instantiation time
- Can be changed by calling
  `widget.configure(**options)`

```python
from Tkinter import *
root = Tk()
b = Button(root, text="Button caption")
b.pack()
# Change the caption color from black to blue
b.configure(foreground="blue")
root.mainloop()
```
Event bindings

- Tkinter *event-driven*
- Reacting to events requires associating *callbacks* to specific events
- Callbacks can be:
  - lambda expressions
  - (bound) methods
  - functions
Selecting events

- Events are specified using descriptors
- General form of descriptors:
  
  `<Modifiers-Type-Qualifier>`

- Multiple modifiers can be specified
- Not all sections are required
Event types

- **Keyboard events:**
  - KeyPress
  - KeyRelease

- **Mouse Events**
  - ButtonPress
  - ButtonRelease
  - Motion
  - Enter
  - Leave
  - MouseWheel

- **Window Events**
  - Activate
  - Deactivate
  - Destroy
  - Expose
  - FocusIn
  - FocusOut
  - Property
  - Reparent
  - Visibility
  - ...

Tkinter from the Ground Up — p.22/40
Event qualifiers

- **Mouse buttons:**
  - 1: left button
  - 2: middle button
  - 3: right button
  - 4: wheel up (XFree86)
  - 5: wheel down (XFree86)

- **Keysyms:**
  - A..Z
  - BackSpace
  - ...

Tkinter from the Ground Up – p.23/40
Event modifiers

- Modifier keys:
  - Control
  - Shift
  - Alt
  - Meta

- Repetition
  - Double
  - Triple
  - Any

- Mouse buttons (drag)
  - B1: Left button
  - B2: Middle button
  - B3: Right button
  - B4
  - B5
Examples of selectors

- `<Double-Button-1 (<Double-1>): double clicks`
- `<keyPress-Tab> (<Tab>): tabulation key`
- `<Control-B1-Motion>: Dragging with mouse button 1 while Control is held down`
Event propagation

- 4 binding levels (in order):
  - Widget
  - Toplevel
  - Class
  - Application

- Most specific descriptors have precedence

- Events go down the chain until a registered callback is found

- Propagation can be stopped by returning "break" from callback
Binding events to callbacks

- 4 methods:
  - `widget.bind(descriptor, callback, add = None)`
  - `toplevel.bind(descriptor, callback, add = None)`
  - `widget.bind_class(class_name, descriptor, callback, add = None)`
  - `widget.bind_all(descriptor, callback, add = None)`
The Canvas widget

- Provides drawing facilities
- Supports various advanced features
  - Splines
  - Postscript output
  - Embedded widgets
- All drawing operations rely on canvas items
  - Items are not widgets
Canvas item classes

- Arc
- Bitmap
- Image
- Line
- Oval
- Polygon
- Rectangle
- Text
- Window
Canvas coordinates

- 2 systems are used simultaneously
  - Canvas coordinate system
  - Window coordinate system
- Converting from window to canvas system:
  - `canvas.canvasx(screenx, gridspacing=None)`
  - `canvas.canvasy(screeny, gridspacing=None)`
- Origin is top-left corner (standard)
Creating items:

```
canvas.create_X(*coords, **options)
```

- returns and item ID (integer)
- Item IDs allow access to the item’s configuration
Modifying items

- `canvas.coords(item, x0, y0, ...)`
- `canvas.delete(item)`
- `canvas.itemcget(item, option)`
- `canvas.itemconfigure(item, **options)`
- `canvas.lift(item)`
- `canvas.lower(item)`
- `canvas.move(item, dx, dy)`
- `canvas.scale(item, xscale, yscale, xoffset, yoffset)`
Finding items

- `canvas.find_above(item)`
- `canvas.find_all()`
- `canvas.find_below(item)`
- `canvas.find_closest(x, y)`
- `canvas.find_enclosed(x1, y1, x2, y2)`
- `canvas.find_overlapping(x1, y1, x2, y2)`
- `canvas.find_withtag(tag)`
Tagging canvas items

- Various Tkinter widgets support tagging
  - Canvas
  - Text
- Canvas items can have an arbitrary number of tags
- Tags are usually strings
- Tags can be shared (grouping)
- Tags and IDs are interchangeable in Canvas methods
Reserved tags

- 2 special, reserved tags
  - `CURRENT` ("current"): added to the item under the cursor
  - `ALL` ("all"): belongs to all items
- Adding the `CURRENT` tag to an item is not an error, but it will get overwritten
Manipulating tags

- `addtag_above(tag, item)`
- `addtag_all(tag)`
- `addtag_below(tag, item)`
- `addtag_closest(tag, x, y, halo, start)`
- `addtag_enclosed(tag, x1, y1, x2, y2)`
- `addtag_overlapping(tag, x1, y1, x2, y2)`
- `addtag_withntag(tag, old_tag)`
- `dtag(item, tag)`
Specifying colors

- 2 possibilities:
  - By name: standard X color names are always available
    - e.g. "Blue"
    - e.g. "Red"
  - By hex strings:
    - "#RGB"
    - "#RRGGBB"
    - "#RRRRGGGGGBBBB"
Specifying fonts

- As tuples of the form 
  `(family, size, options)`
  - e.g. `("arial", 10, "bold italic")`

- As Font instances:
  - instances of the class `tkFont.Font`
  - e.g. `f = tkFont.Font(family="arial", size=10, weight=tkFont.BOLD, slant=tkFont.ITALIC)`
Specifying fonts

- as system font names
  - e.g. "6x10"

- as X font descriptors
  - e.g.
    -*-courier-medium-r-**--14--**-*-*-*-*-*-*
For more information . . .


- http://www.python.org/