CS2021- Week 7

Tkinter Tour

Core tkinter concepts

Widget Classes:

Label

Button

Frame

Toplevel, Tk

Message, Entry,

Checkbutton, Radiobutton

Menubutton, Scrollbar, Listbox

Canvas

Composites: Dialog, ScrolledText,
OptionMenu

Example 8-3.

PP4E/Gui/Tour/ toplevel0.py

```
#independent windows
part of same process

import sys
from tkinter import
Toplevel, Button, Label
win1 = Toplevel() # two
independent windows
win2 = Toplevel() # but
part of same process
Button(win1,
text='Spam',
command=sys.exit).pack()
```

```
Button(win2,
text='SPAM',
command=sys.exit).pack()
Label(text='Popups').pac
k() # on default Tk()
root window
win1.mainloop()
```

Top-Level Window Protocols

toplevel2.py

```
from tkinter import *
root = Tk() # explicit root
trees = [('The Larch!', 'light
blue'),
('The Pine!', 'light green'), ('The
Giant Redwood!', 'red')]
for (tree, color) in trees:
    win = Toplevel(root) # new
```

```
window
    win.title('Sing...') # set border

win.protocol('WM_DELETE_WIN
DOW', lambda:None) # ignore
close
    win.iconbitmap('py-blue-
trans-out.ico') # not red Tk
    msg = Button(win, text=tree,
command=win.destroy) # kills
one win
    msg.pack(expand=YES,
fill=BOTH)
    msg.config(padx=10, pady=10,
bd=10, relief=RAISED)
    msg.config(bg='black',
fg=color, font=('times', 30, 'bold
```

```
italic'))

root.title('Lumberjack demo')
Label(root, text='Main window',
width=30).pack()
Button(root, text='Quit All',
command=root.quit).pack() #
kills all app
root.mainloop()
```

Standard dialogs

generate—file selection dialogs, error and warning pop ups, and question and answer prompts.

“Smart Quit”
quitter.py

```
from tkinter import * # get
widget classes
from tkinter.messagebox import
askokcancel # get canned std
dialog
class Quitter(Frame): # subclass
our GUI
    def __init__(self,
parent=None): # constructor
method
        Frame.__init__(self,
parent)
        self.pack()
        widget = Button(self,
```

```
text='Quit', command=self.quit)
    widget.pack(side=LEFT,
expand=YES, fill=BOTH)

    def quit(self):
        ans = askokcancel('Verify
exit', "Really quit?")
        if ans: Frame.quit(self)
if __name__ == '__main__':
Quitter().mainloop()
```

Creating dialogs

```
#More useful than priniting to
stdout
# define a name:callback demos
table
```

```
from tkinter.filedialog   import
askopenfilename       # get
standard dialogs
from tkinter.colorchooser import
askcolor              # they live in Lib
\tkinter
from tkinter.messagebox   import
askquestion, showerror
from tkinter.simpledialog import
askfloat

demos = {
   'Open':  askopenfilename,
   'Color': askcolor,
   'Query': lambda:
askquestion('Warning', 'You
typed "rm *"\nConfirm?'),
```

```
    'Error': lambda:
showerror('Error!', "He's dead,
Jim"),
    'Input': lambda:
askfloat('Entry', 'Enter credit card
number')
}
```

dialogTable.py
demoDlg.py

```
from tkinter import * # get base
widget set
from dialogTable import demos #
button callback handlers
from quitter import Quitter #
attach a quit object to me
```

```
class Demo(Frame):
    def __init__(self,
parent=None, **options):
    Frame.__init__(self, parent,
**options)
    self.pack()
    Label(self, text="Basic
demos").pack()
    for (key, value) in
demos.items():
        Button(self, text=key,
command=value).pack(side=TOP,
fill=BOTH)
    Quitter(self).pack(side=TOP,
fill=BOTH)
if __name__ == '__main__':
Demo().mainloop()
```

Printing dialog results demoDlg-print.py

```
from tkinter import *
# get base widget set
from dialogTable import
demos       # button
callback handlers
from quitter import
Quitter          # attach
a quit object to me

class Demo(Frame):
    def __init__(self,
parent=None):
```

```
Frame.__init__(self,
parent)
        self.pack()
        Label(self,
text="Basic
demos").pack()
        for key in
demos:
            func =
(lambda key=key:
self.printit(key))
            Button(self,
text=key,
command=func).pack(side=
TOP, fill=BOTH)

Quitter(self).pack(side=
```

```
TOP, fill=BOTH)

    def printit(self,
name):
        print(name,
'returns =>',
demos[name]()) # fetch,
call, print

if __name__ ==
'__main__':
Demo().mainloop()
```

Enclosing scope Lookup

Button-press callbacks are run with no arguments

Actual handler printit needs extra data (i.e. demo key)

must wrap the handler with lambda, enclosing scope lookup

```
 for key in demos:
  func = (lambda
key=key:
self.printit(key))
  Button(self, text=key,

command=func).pack(side=
TOP, fill=BOTH)
```

for loop creates new scope for key so use default argument to pass the current key to the *printit handler*

the lambda remembers and

passes on the state information

Binding Events – Tour/ bind.py

Callback functions triggered when bound events occur.
Callbacks for bind receive Event object argument, an instance of tkinter Event class

```
from tkinter import *
def showPosEvent(event):
    print('Widget=%s X=%s Y=%s' %
(event.widget, event.x, event.y))
def showAllEvent(event):
    print(event)
    for attr in dir(event):
        if not attr.startswith('__'):
            print(attr, '=>',
getattr(event, attr))
def onKeyPress(event):
    print('Got key press:', event.char)
def onArrowKey(event):
    print('Got up arrow key press')
```

```
tkroot = Tk()
labelfont = ('courier',
20, 'bold')
# family, size, style
widget = Label(tkroot,
text='Hello bind world')
widget.config(bg='red',
font=labelfont)
# red background, large
font
widget.config(height=5,
width=20)
# initial size:
lines,chars
widget.pack(expand=YES,
fill=BOTH)
```

```
widget.bind('<Button-1>'
,  onLeftClick)
# mouse button clicks
widget.bind('<Button-3>'
,  onRightClick)
widget.bind('<Button-2>'
,  onMiddleClick)
# middle=both on some
mice
widget.bind('<Double-1>'
,  onDoubleLeftClick)
# click left twice
widget.bind('<B1-
Motion>', onLeftDrag)
# click left and move
```

```
widget.bind('<KeyPress>'
,  onKeyPress)
# all keyboard presses
widget.bind('<Up>',
onArrowKey)
# arrow button pressed
widget.bind('<Return>',
onReturnKey)
# return/enter key
pressed
widget.focus()
# or bind keypress to
tkroot
tkroot.title('Click Me')
tkroot.mainloop()
```

Tour/canvasDraw.py

```
class CanvasEventsDemo:
    def __init__(self, parent=None):
        canvas = Canvas(width=300, height=300,
bg='beige')
        canvas.pack()
        canvas.bind('<ButtonPress-1>',
self.onStart)      # click
        canvas.bind('<B1-Motion>',
self.onGrow)       # and drag
        canvas.bind('<Double-1>',
self.onClear)      # delete all
        canvas.bind('<ButtonPress-3>',
self.onMove)       # move latest
        self.canvas = canvas
        self.drawn  = None
        self.kinds = [canvas.create_oval,
canvas.create_rectangle]

    def onStart(self, event):
        self.shape = self.kinds[0]
        self.kinds = self.kinds[1:] +
self.kinds[:1]      # start dragout
        self.start = event
        self.drawn = None
```

Binding to specific items

canvas-bind.py

```
from tkinter import *
```

```
def onCanvasClick(event):
    print('Got canvas click', event.x, event.y,
event.widget)

def onObjectClick(event):
    print('Got object click', event.x, event.y,
event.widget, end=' ')
    print(event.widget.find_closest(event.x,
event.y))  # find text object's ID

root = Tk()
canv = Canvas(root, width=100, height=100)
obj1 = canv.create_text(50, 30, text='Click me
one')
obj2 = canv.create_text(50, 70, text='Click me
two')

canv.bind('<Double-1>', onCanvasClick)
# bind to whole canvas
canv.tag_bind(obj1, '<Double-1>', onObjectClick)
# bind to drawn item
canv.tag_bind(obj2, '<Double-1>', onObjectClick)
# a tag works here too
canv.pack()
root.mainloop()
```

Homework #7

Two week design project: Due Wed Oct 21

Developing a GUI to meet

- Business need
- Define a business information need
  - Enumerate features through “user stories”
- Create a GUI that addresses that need
- Your solution should include buttons and dialogs
- Free to explore code samples in PP4E/Gui