Graphical User Interfaces

- Basic `tkinter` Widgets
- Event-Based `tkinter` Widgets
- Designing GUIs
- OOP for GUIs

Graphical user interfaces (GUIs)

Almost all computer apps have a GUI

- A GUI gives a better overview of what an application does
- A GUI makes it easier to use the application.

A graphical user interface (GUI) consists of basic visual building blocks, called widgets, packed inside a standard window.

- widgets include buttons, labels, text entry forms, menus, check boxes, scroll bars, …

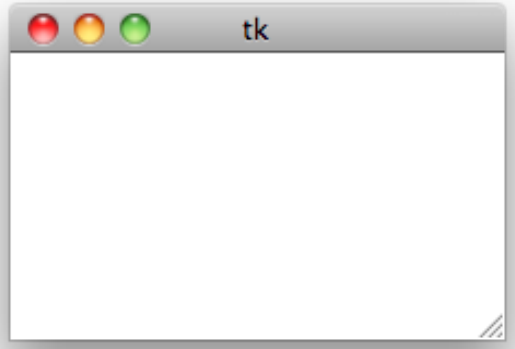
In order to develop GUIs, we need a module that makes widgets available; we will use module `tkinter` that is included in the Standard Library.

Widget `Tk`

We introduce some of the commonly used `tkinter` widgets

Widget `Tk` represents the GUI window

```
>>> from tkinter import Tk
>>> root = Tk()
>>> root.mainloop()
```

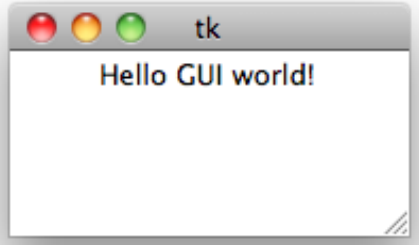


As usual, the constructor creates the widget (i.e., GUI object) …
… but method `mainloop()` really starts the GUI

The window is currently empty; normally it contains other widgets

Widget `Label` (for displaying text)

The widget `Label` can be used to display text inside a window.



Method `pack()` specifies the placement of the widget within its master

Option	Description
`master`	**The master of the widget**
`text`	**Text to display on the widget**
`image`	Image to display
`width`	Width of widget (in pixels or characters)
`height`	Height of widget (in pixels or characters)
`relief`	Border style (FLAT, RAISED, RIDGE, …)
`borderwidth`	Width of border (no border is 0)
`background`	Background color (e.g., string “white”)
`foreground`	Foreground color
`font`	Font descriptor (as a tuple

- axis

```
>>> from tkinter import Tk, Label
>>> root = Tk()
>>> hello = Label(master = root, text = 'Hello GUI world!')
>>> hello.pack() # widget placed against top boundary of master (default)
>>> root.mainloop()
```

Widget `Label` (for displaying images)

The widget `Label` can be used to display images too

Option	Description
`master`	The master of the widget
`text`	Text to display
`image`	Image to display
`width`	Width of widget (in pixels or characters)
`height`	Height of widget (in pixels or characters)
`relief`	Border style (FLAT, RAISED, RIDGE, …)
`borderwidth`	Width of border (no border is 0)
	color (e.g., string “white”)
	color
	tor (as a tuple
	ed along the x- or y- axis

Option `image` must refer to an image in a format that `tkinter` can display. The `PhotoImage` class, defined in module `tkinter`, is used to transform a GIF image into an object with such a format.

```
from tkinter import Tk, Label, PhotoImage
root = Tk()
# transform GIF image to a format tkinter can display
photo = PhotoImage(file='peace.gif')

peace = Label(master=root,
              image=photo,
              width=300,    # width of label, in pixels
              height=180)   # height of label, in pixels
peace.pack()
root.mainloop()
```

peace.py

Packing widgets

Method `pack()` specifies the placement of the widget within its master

Option	Description
side	LEFT, RIGHT, TOP, BOTTOM,
fill	'both', 'x', 'y', or 'none'
expand	True or False

```
from tkinter import Tk, Label, PhotoImage, BOTTOM,
LEFT, RIGHT, RIDGE
root = Tk()

text = Label(root,
             font=('Helvetica', 16, 'bold italic'),
             foreground='white',
             background='black',
             pady=10,
             text='Peace begins with a smile.')
text.pack(side=BOTTOM)

peace = PhotoImage(file='peace.gif')
peaceLabel = Label(root,
                   borderwidth=3,
                   relief=RIDGE,
                   image=peace)
peaceLabel.pack(side=LEFT)

smiley = PhotoImage(file='smiley.gif')
smileyLabel = Label(root,
                    image=smiley)
smileyLabel.pack(side=RIGHT)

root.mainloop()
```

smileyPeace.py

Arranging widgets into a grid

Method `grid()` is used to place widgets in a grid format

`pack()` and `grid()` use different algorithms to place widgets within a master; You must use one or the other for all widgets with the same master.

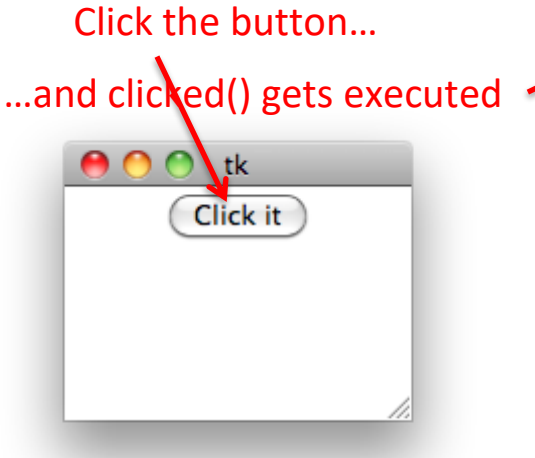
Options
column
columnspan
row
rowspan

```
from tkinter import Tk, Label, RAISED

root = Tk()
labels = [['1', '2', '3'],
          ['4', '5', '6'],
          ['7', '8', '9'],
          ['*', '0', '#']]

for r in range(4):
    for c in range(3):
        # create label for row r and column c
        label = Label(root,
                      relief=RAISED,
                      padx=10,
                      text=labels[r][c])
        # place label in row r and column c
        label.grid(row=r, column=c)

root.mainloop()
```

phone.py

Widget Button

Widget `Button` represents the standard clickable GUI button




```
>>> === RESTART ===
>>>
Day:  13 Apr 2012
Time: 15:50:05 PM

Day:  13 Apr 2012
Time: 15:50:07 PM

Day:  13 Apr 2012
Time: 15:50:11 PM
```

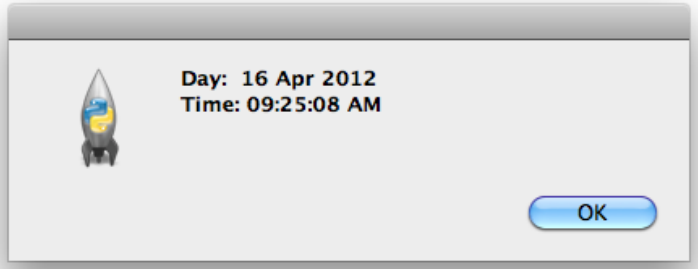
Option `command` specifies the function that is executed every time the button is clicked

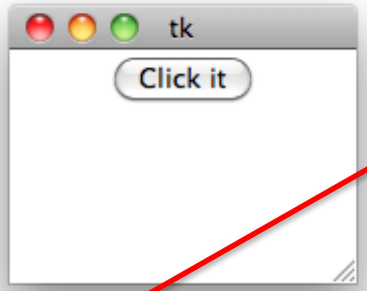
This function is called an event handler: it handles the event of clicking this particular button

```
from tkinter import Tk, Button
from time import strftime, localtime

def clicked():
    'prints day and time info'
    time = strftime('Day:  %d %b %Y\nTime: %H:%M:%S %p\n',
                    localtime())
    print(time)

root = Tk()
button = Button(root,
                text='Click it',
                command=clicked)
button.pack()
root.mainloop()
```

clickit.py

Widget Button






```
>>> === RESTART ===
>>>
Day:  13 Apr 2012
Time: 15:50:05 PM

Day:  13 Apr 2012
Time: 15:50:07 PM

Day:  13 Apr 2012
Time: 15:50:07 PM
```

Suppose we want the date and time to be printed in a window, rather than in the shell

```
from tkinter import Tk, Button
from time import strftime, localtime
from tkinter.messagebox import showinfo

def clicked():
    'prints day and time info'
    time = strftime('Day:  %d %b %Y\nTime: %H:%M:%S %p\n',
                    localtime())
    showinfo(message = time)

root = Tk()
button = Button(root,
                text='Click it',
                command=clicked)
button.pack()
root.mainloop()
```

clickit.py

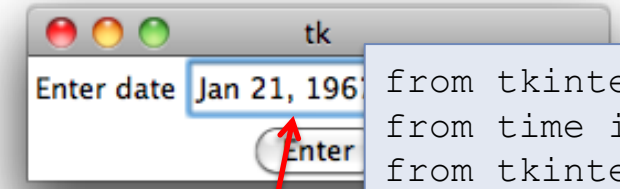
Event-driven programming

When a GUI is started with the `mainloop()` method call, Python starts an infinite loop called an event loop

```
while True:

    1. wait for an event to occur

    2. run the associated event handler
```

Event-driven programming is the programming approach used to build applications whose execution flow is determined by events and described using an event loop

Widget `Entry`



Widget `Entry` represents the single-line text entry/display form

To illustrate it, let's build an app that takes a date and prints the day of the week corresponding to the date

```
from tkinter import Tk, Button, Entry, Label, END
from time import strptime, strftime
from tkinter.messagebox import showinfo

def compute():
    global dateEnt    # dateEnt is a global variable
    date = dateEnt.get()
    weekday = strftime('%A', strptime(date, '%b %d, %Y'))
    showinfo(message = '{} was a {}'.format(date, weekday))
    dateEnt.delete(0, END)

root = Tk()

label = Label(root, text='Enter date')
label.grid(row=0, column=0)

dateEnt = Entry(root)
dateEnt.grid(row=0, column=1)

button = Button(root, text='Enter', command=compute)
button.grid(row=1, column=0, columnspan=2)

root.mainloop()
```

Event handler `compute()` should:

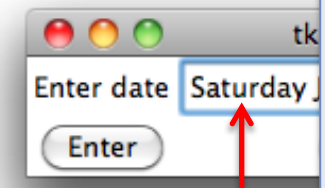
1. Read the date from entry `dateEnt`
2. Compute the weekday corresponding to the date
3. Display the weekday message in a pop-up window
4. Erase the date from entry `dateEnt` (to make it easier to enter another date)

Widget `Entry`

```
from tkinter import Tk, Button, Entry, Label, END
from time import strptime, strftime
from tkinter.messagebox import showinfo

def compute():
    global dateEnt   # dateEnt is a global variable
    date = dateEnt.get()
    weekday = strftime('%A', strptime(date, '%b %d, %Y'))
    showinfo(message = '{} was a {}'.format(date, weekday))
    dateEnt.delete(0, END)
...
dateEnt = Entry(root)
dateEnt.grid(row=0, column=1)
...
```

Method	Description
`e.get()`	return string in entry `e`
`e.insert(idx, text)`	insert text into entry `e` starting at index `idx`
`e.delete(from, to)`	delete text from index `from` to index `to` inside entry `e`

Exercise



Modify the app so th
the weekday messag
window, insert it in f
entry box.

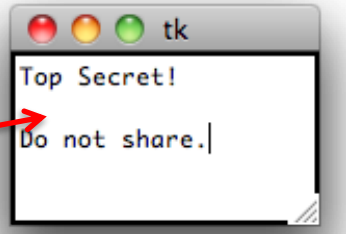
Also add a button lab
erases the entry box

```
from tkinter import Tk, Button, Entry, Label, END
from time import strptime, strftime
from tkinter.messagebox import showinfo

def compute():
    global dateEnt   # dateEnt is a global variable
    date = dateEnt.get()
    weekday = strftime('%A', strptime(date, '%b %d, %Y'))
    dateEnt.insert(0, weekday + ' ')

def clear():
    global dateEnt   # dateEnt is a global variable
    dateEnt.delete(0, END)

root = Tk()

label = Label(root, text='Enter date')
label.grid(row=0, column=0)

dateEnt = Entry(root)
dateEnt.grid(row=0, column=1)

button = Button(root, text='Enter', command=compute)
button.grid(row=1, column=0)

button = Button(root, text='Clear', command=clear)
button.grid(row=1, column=1)

root.mainloop()
```

Widget `Text`

We use a `Text` widget to develop an application that looks like a text editor, but "secretly" records and prints every keystroke the user types

Widget `Text` represents the multi-line text entry/display form




```
>>>
char = T
char = o
char = p
char = space
char = S
char = e
char = c
char = r
char = e
char = t
char = exclam
char = Return
char = Return
char = D
char = o
char = space
char = n
char = o
char = t
char = space
char = s
char = h
char = a
char = r
char = e
char = period
```

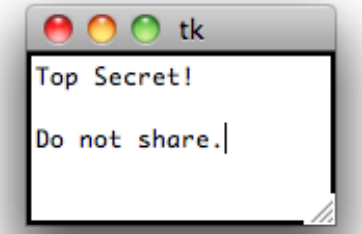
Like widget `Entry`, it supports methods `get()`, `insert()`, `delete()`

- except that the index has the format `row.column`

Method	Description
`t.get(from, to)`	return text from index `from` to index
`t.insert(idx, text)`	insert `text` into text entry `t` starting
`t.delete(from, to)`	delete text from index `from` to index `to` inside text entry `t`

Widget `Text`

We use a `Text` widget to develop an application that looks like a text editor, but "secretly" records and prints every keystroke the user types




```
>>>
char = T
char = o
char = p
char = space
char = S
char = e
char = c
char = r
char = e
char = t
char = exclam
char = Return
char = Return
char = D
char = o
char = space
char = n
char = o
char = t
char = space
char = s
char = h
char = a
char = r
char = e
char = period
```

In order to record every keystroke, we need to associate an event-handling function with keystrokes

Widget method `bind()` method "binds" (i.e., associates) an event type to an event handler. For example

```
text.bind('<KeyPress>', record)
```

binds a keystroke, described with string `'<KeyPress>'`, within widget `text` to event handler `record()`

Widget Text

Event-handling function `record()` takes as input an object of type `Event`; this object is created by Python when an event occurs

```
from tkinter import Tk, Text, BOTH

def record(event):
    '''event handling function for key press events;
       input event is of type tkinter.Event'''
    print('char = {}'.format(event.keysym)) # print key symbol

root = Tk()

text = Text(root,
            width=20,  # set width to 20 characters
            height=5)  # set height to 5 rows of characters

# Bind a key press event with the event handling function record()
text.bind('<KeyPress>', record)

# widget expands if the master does
text.pack(expand=True, fill=BOTH)

root.mainloop()
```

An `Event` object contains information about the event, such as the symbol of the pressed key

Keystroke events are bound to event handling function `record()`

Event pattern and `tkinter` class `Event`

Type	Description
`Button`	Mouse button
`Return`	Enter/Return key
`KeyPress`	Press of a keyboard key
`KeyRelease`	Release of a keyboard key
`Motion`	Mouse motion

Modifier	Description
`Control`	Ctrl key
`Button1`	Left mouse button
`Button3`	Right mouse button
`Shift`	Shift key

Detail	Description
`<button number>`	Ctrl key
`<key symbol>`	Left mouse button

The first argument of method `bind()` is the type of event we want to bind

The type of event is described by a string that is the concatenation of one or more event patterns

An event pattern has the form

```
<modifier-modifier-type-detail>
```

- **`<Control-Button-1>`: Hitting Ctrl and the left mouse button simultaneously**
- **`<Button-1><Button-3>`: Clicking the left mouse button and then the right one**
- **`<KeyPress-D><Return>`: Hitting the keyboard key and then Return**
- `<Button1-Motion>`: Mouse motion while holding left mouse button

Event pattern and `tkinter` class `Event`

The second argument of method `bind()` is the event handling function

The event handling function must be defined to take exactly one argument, an object of type `Event`, a class defined in `tkinter`

When an event occurs, Python will create an object of type `Event` associated with the event and then call the event-handling function with the `Event` object passed as the single argument

An `Event` object has many attributes that store information about the event

Attribute	Event Type	Description
`num`	ButtonPress, ButtonRelease	Mouse button pressed
`time`	all	Time of event
`x`	all	x-coordinate of mouse
`y`	all	x-coordinate of mouse
`keysum`	KeyPress, KeyRelease	Key pressed as string
`keysum_num`	KeyPress, KeyRelease	Key pressed as Unicode number

Widget `Canvas`

Widget `Canvas` represents a drawing board in which lines and other geometrical objects can be drawn

We illustrate widget `Canvas` by developing a pen drawing app

- the user starts the drawing of the curve by pressing the left mouse button
- the user then draws the curve by moving the mouse, while still pressing the left mouse button

Widget Canvas

Every time the mouse is moved while pressing the left mouse button, the handler `draw()` is called with an `Event` object storing the new mouse position.

To continue drawing the curve, we need to connect this new mouse position to the previous one with a straight line.

```
from tkinter import Tk, Canvas

# event handlers begin() and draw() to be defined

root = Tk()
canvas = Canvas(root, height=100, width=150)

# bind left mouse button click event to function begin()
canvas.bind("<Button-1>", begin)

# bind mouse motion while pressing left button event
canvas.bind("<Button1-Motion>", draw)

canvas.pack()
root.mainloop()
```

We illustrate widget `Canvas` by developing a pen drawing app

- the user starts the drawing of the curve by pressing the left mouse button
- the user then draws the curve by moving the mouse, while still pressing the left mouse button

Widget Canvas

Therefore the previous mouse position must be stored

But where?

In global variables `x` and `y`

Handler `begin()` sets the initial values of `x` and `y`

Method `create_line()` creates a line segment between `(x, y)` and `(newx, newy)`

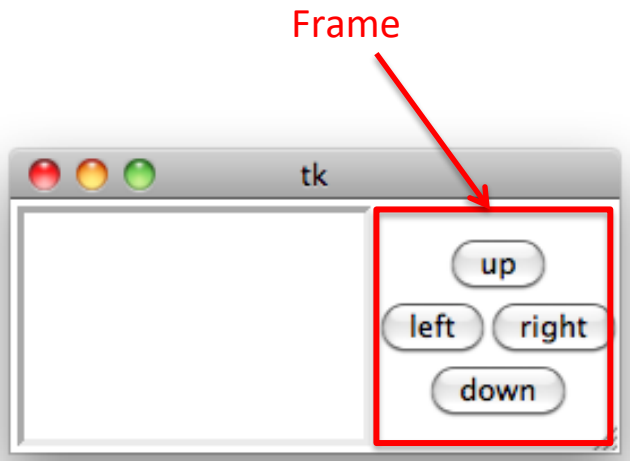
We illustrate widget `Can`

- the user starts th
- the user then dra
  the left mouse bu

```
from tkinter import Tk, Canvas

def begin(event):
    global x, y
    x, y = event.x, event.y

def draw(event):
    global x, y, canvas
    newx, newy = event.x, event.y
    # connect previous mouse position to current one
    canvas.create_line(x, y, newx, newy)
    # new position becomes previous
    x, y = newx, newy

root = Tk()
x, y = 0, 0 # mouse coordinates (global variables)
canvas = Canvas(root, height=100, width=150)

# bind left mouse button click event to function begin()
canvas.bind("<Button-1>", begin)

# bind mouse motion while pressing left button event
canvas.bind("<Button1-Motion>", draw)

canvas.pack()
root.mainloop()
```

Widget `Frame`

Widget `Frame` is a key widget whose primary purpose is to serve as the master of other widgets and help define a hierarchical structure of the GUI and its geometry



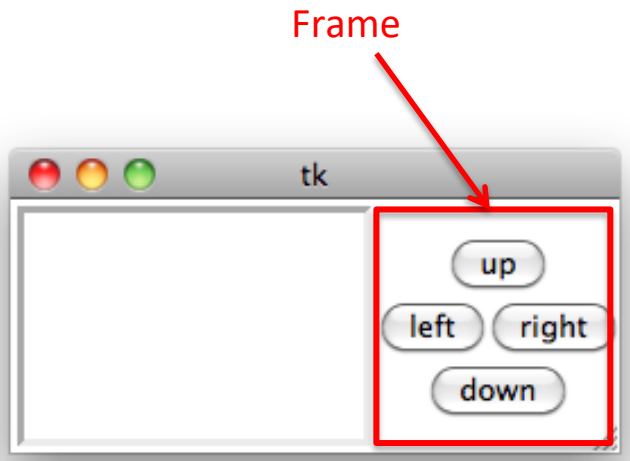
We illustrate widget `Frame` by developing an *Etch-A-Sketch* drawing app

- Pressing a button moves the pen 10 pixels in the indicated direction

To facilitate the specification of the geometry of the GUI widgets, we use a `Frame` widget to be the master of the 4 buttons

Widget Frame

```
from tkinter import Tk, Canvas, Frame, Button,
SUNKEN, LEFT, RIGHT

# event handlers to be defined here

root = Tk()
canvas = Canvas(root, height=100, width=150,
                relief=SUNKEN, borderwidth=3)
canvas.pack(side=LEFT)

box = Frame(root) # frame to hold the 4 buttons
box.pack(side=RIGHT)

# buttons have Frame widget as their master
button = Button(box, text='up', command=up)
button.grid(row=0, column=0, columnspan=2)
button = Button(box, text='left', command=left)
button.grid(row=1, column=0)
button = Button(box, text='right', command=right)
button.grid(row=1, column=1)
button = Button(box, text='down', command=down)
button.grid(row=2, column=0, columnspan=2)

x, y = 50, 75      # initial pen position
root.mainloop()
```



Exercise

```
def up():
    'move pen up 10 pixels'
    global y, canvas
    canvas.create_line(x, y, x, y-10)
    y -= 10

def down():
    'move pen down 10 pixels'
    global y, canvas
    canvas.create_line(x, y, x, y+10)
    y += 10

def left():
    'move pen left 10 pixels'
    global x, canvas
    canvas.create_line(x, y, x-10, y)
    x -= 10

def right():
    'move pen right 10 pixels'
    global x, canvas
    canvas.create_line(x, y, x+10, y)
    x += 10
```

```
x, y = 50, 75      # initial pen position
root.mainloop()
```

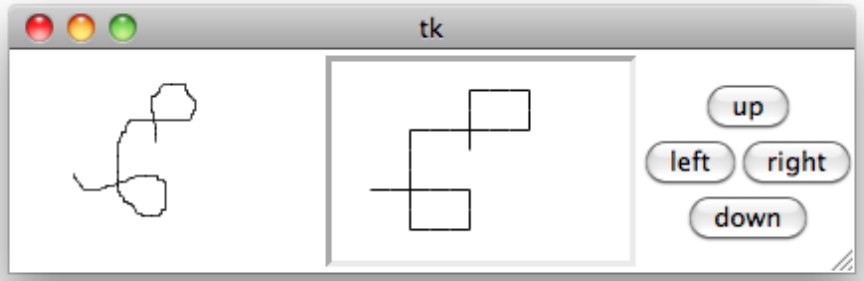
Implement the 4 event handlers

Note: the x coordinates increase from left to right, while the y coordinates increase from top to bottom

OOP for GUIs

Suppose we want to build a new GUI that incorporates GUIs we have already developed

- For example, GUIs draw and Etch-A-Sketch



Ideally, we would like to reuse the code we have already developed

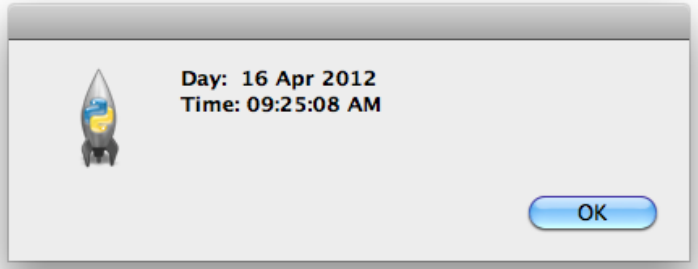
OOP for GUIs

```
from tkinter import Tk, Car
SUNKEN, LEFT, RIGHT

# event handlers to be defi

root = Tk()
canvas = Canvas(root, heigh
                relief=SUNK
canvas.pack(side=LEFT)

box = Frame(root) # frame t
box.pack(side=RIGHT)

# buttons have Frame widget
button = Button(box, text='
button.grid(row=0, column=0
button = Button(box, text='
button.grid(row=1, column=0
button = Button(box, text='
button.grid(row=1, column=1
button = Button(box, text='
button.grid(row=2, column=0

x, y = 50, 75      # initial
root.mainloop()
```

```
from tkinter import Tk, Canvas

def begin(event):
    global x, y
    x, y = event.x, event.y

def draw(event):
    global x, y, canvas
    newx, newy = event.x, event.y
    # connect previous mouse position to current one
    canvas.create_line(x, y, newx, newy)
    # new position becomes previous
    x, y = newx, newy

root = Tk()
x, y = 0, 0 # mouse coordinates (global variables)
canvas = Canvas(root, height=100, width=150)

# bind left mouse button click event to function begin()
canvas.bind("<Button-1>", begin)

# bind mouse motion while pressing left button event
canvas.bind("<Button1-Motion>", draw)

canvas.pack()
root.mainloop()
```

Need to rename x and y

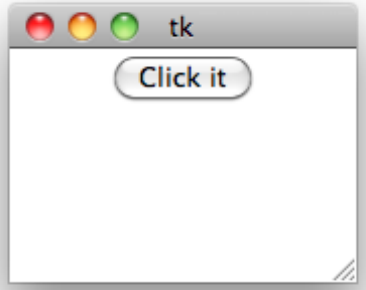
OOP for GUIs



Our GUI programs do not encapsulate the implementation, making code reuse problematic

We now redevelop our GUIs as classes using OOP, so that they are easily reusable

Let's start simple, with the ClickIt app




```
from tkinter import Tk, Button
from time import strftime, localtime
from tkinter.messagebox import showinfo

def clicked():
    time = strftime('Day:  %d %b %Y\nTime: %H:%M:%S %p\n', localtime())
    showinfo(message = time)

root = Tk()
button = Button(root, text='Click it', command=clicked)
button.pack()
root.mainloop()
```

Class `ClickIt`

Main idea: incorporating a widget into a GUI is easy, so develop the user-defined GUI so it is a widget

How? By developing the user-defined GUI as a subclass of a built-in widget class

- Class Frame, for example

```
class ClickIt(Frame):

    # class methods to be defined
```

Usage

```
>>> from tkinter import Tk
>>> root = Tk()
>>> clickit = ClickIt(root)
>>> clickit.pack()
>>> root.mainloop()
```

`ClickIt` constructor takes as input the master widget

```
from tkinter import Tk, Button
from time import strftime, localtime
from tkinter.messagebox import showinfo

def clicked():
    time = strftime('Day:  %d %b %Y\nTime: %H:%M:%S %p\n', localtime())
    showinfo(message = time)

root = Tk()
button = Button(root, text='Click it', command=clicked)
button.pack()
root.mainloop()
```

Class `ClickIt`

```
# from ...

class ClickIt(Frame):

    def __init__(self, master):
        Frame.__init__(self, master)
        button = Button(self, text='Click it', command=self.clicked)
        button.pack()

    def clicked(self):
        time = strftime('Day:  %d %b %Y\nTime: %H:%M:%S %p\n', localtime())
        showinfo(message=time)
```

constructor input argument: the master widget

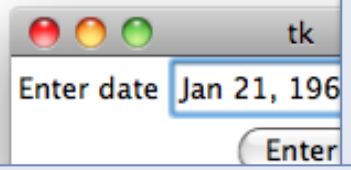
`ClickIt` should be initialized just like `Frame`

event handler is a class method (for encapsulation)

`ClickIt` widget `self` contains a `Button` widget that packs itself inside its master (`self`)

```
from tkinter import Tk, Button
from time import strftime, localtime
from tkinter.messagebox import showinfo

def clicked():
    time = strftime('Day:  %d %b %Y\nTime: %H:%M:%S %p\n', localtime())
    showinfo(message = time)

root = Tk()
button = Button(root, text='Click it', command=clicked)
button.pack()
root.mainloop()
```

Instance variables for shared widgets




```
from tkinter import Tk, Button, Entry, Label, END
from time import strptime, strftime
from tkinter.messagebox import showinfo
```

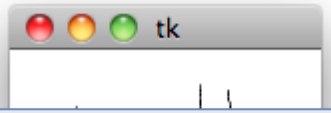
```
class Day(Frame):

    def __init__(self, master):
        Frame.__init__(self, master)

        label = Label(self, text='Enter date')
        label.grid(row=0, column=0)

        self.dateEnt = Entry(self)              # instance variable
        self.dateEnt.grid(row=0, column=1)

        button = Button(self, text='Enter', command=self.compute)
        button.grid(row=1, column=0, columnspan=2)

    def compute(self):
        date = self.dateEnt.get()
        weekday = strftime('%A', strptime(date, '%b %d, %Y'))
        showinfo(message = '{} was a {}'.format(date, weekday))
        self.dateEnt.delete(0, END)
```

Entry widget is assigned to an instance variable …

… so it is accessible by the event handler without global variables

day.py

Instance variables for shared data




```
from tkinter import Tk, Canvas

def begin(event):
    global x, y
```

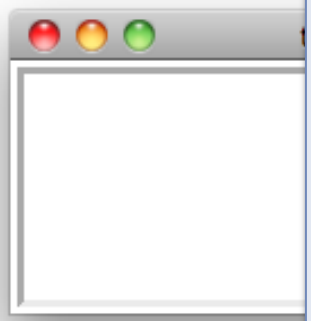
In addition to the Canvas widget, variables x and y are
y event handlers

```
from tkinter import Canvas, Frame, BOTH
class Draw(Frame):

    def __init__(self, parent):
        Frame.__init__(self, parent)

        # mouse coordinates are instance variables
        self.oldx, self.oldy = 0, 0

        # create canvas and bind mouse events to handlers
        self.canvas = Canvas(self, height=100, width=150)
        self.canvas.bind("<Button-1>", self.begin)
        self.canvas.bind("<Button1-Motion>", self.draw)
        self.canvas.pack(expand=True, fill=BOTH)

    def begin(self, event):
        self.oldx, self.oldy = event.x, event.y

    def draw(self, event):
        newx, newy = event.x, event.y
        self.canvas.create_line(self.oldx, self.oldy, newx, newy)
        self.oldx, self.oldy = newx, newy
```

```
o current one
y)

 variables)
=150)

 function begin()

 button event
```

Exercise

Redevelop the app as a class



```
from tkinter import Tk, Canvas, Frame, Button,
SUNKEN, LEFT, RIGHT
```

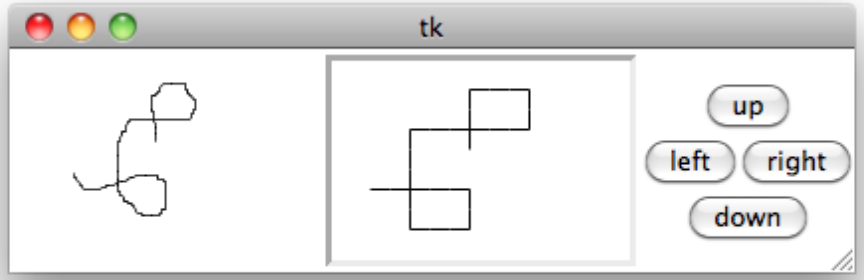
```
from tkinter import Tk, Canvas, Frame, Button, SUNKEN, LEFT, RIGHT
class Plotter(Frame):

    def __init__(self, parent=None):
        Frame.__init__(self, parent)
        self.x, self.y = 75, 50

        self.canvas = Canvas(self, height=100, width=150,
                             relief=SUNKEN, borderwidth=3)
        self.canvas.pack(side=LEFT)

        buttons = Frame(self)
        buttons.pack(side=RIGHT)
        b = Button(buttons, text='up', command=self.up)
        b.grid(row=0, column=0, columnspan=2)
        b = Button(buttons, text='left', command=self.left)
        b.grid(row=1, column=0)
        b = Button(buttons, text='right', command=self.right)
        b.grid(row=1, column=1)
        b = Button(buttons, text='down', command=self.down)
        b.grid(row=2, column=0, columnspan=2)

    def up(self):
        self.canvas.create_line(self.x, self.y, self.x, self.y-10)
        self.y -= 10

    # remaining event handlers omitted
```

OOP for GUIs

Let's now develop the GUI combining our draw and Etch-A-Sketch apps

```
class App(Frame):

    def __init__(self, master):
        Frame.__init__(self, master)
        draw = Draw(self)
        draw.pack(side=LEFT)
        plotter = Plotter(self)
        plotter.pack(side=RIGHT)
```

Yes, that's it!



To get it started:

```
>>> from tkinter import Tk
>>> root = Tk()
>>> app = App(root)
>>> app.pack()
>>> root.mainloop()
```

The encapsulation and abstraction resulting from implementing our GUIs as classes makes code reuse easy