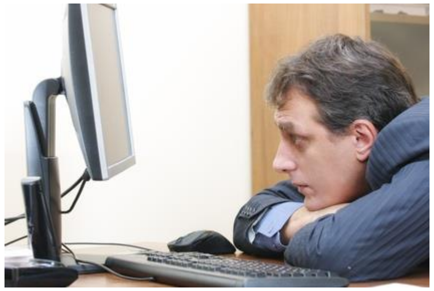
Event Loops and GUI

Intro2CS – weeks 11-12

The taxi dispatcher

- Imagine a taxi cab dispatcher.
- His job is:
  - to keep track of the location of taxis
  - to answer requests for taxis (and dispatch a taxi)
- Most of the time he is bored.

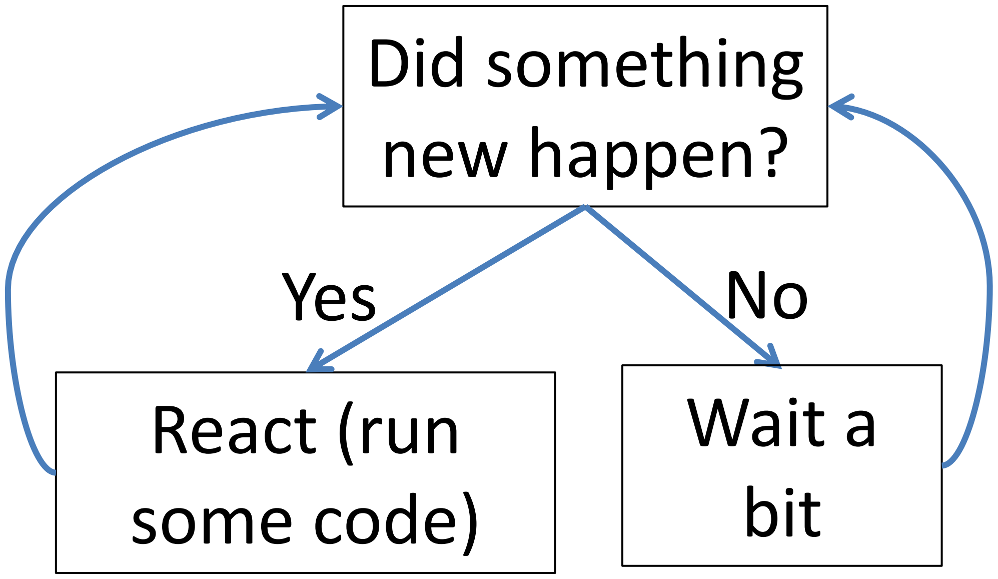


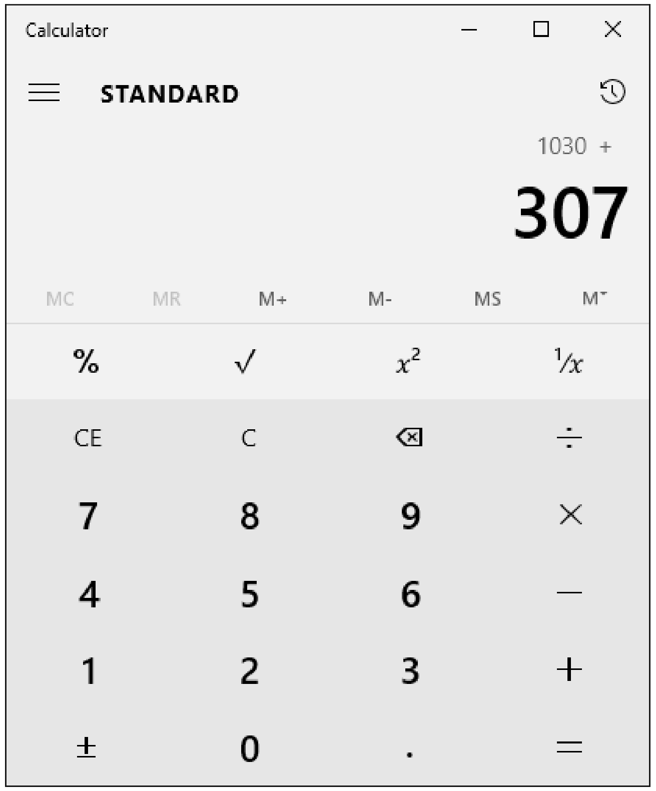
How do we describe his job?

- When a taxi reports dropping off a passenger:
  - if customers are waiting, send cab
  - Else add cab to "waiting cabs" list
- When a customer calls in:
  - If there are waiting taxis: send a taxi
  - Else add the customer to the waiting customers list
- Notice that we are describing the job of the dispatcher when meaningful <u>**events**</u> happen

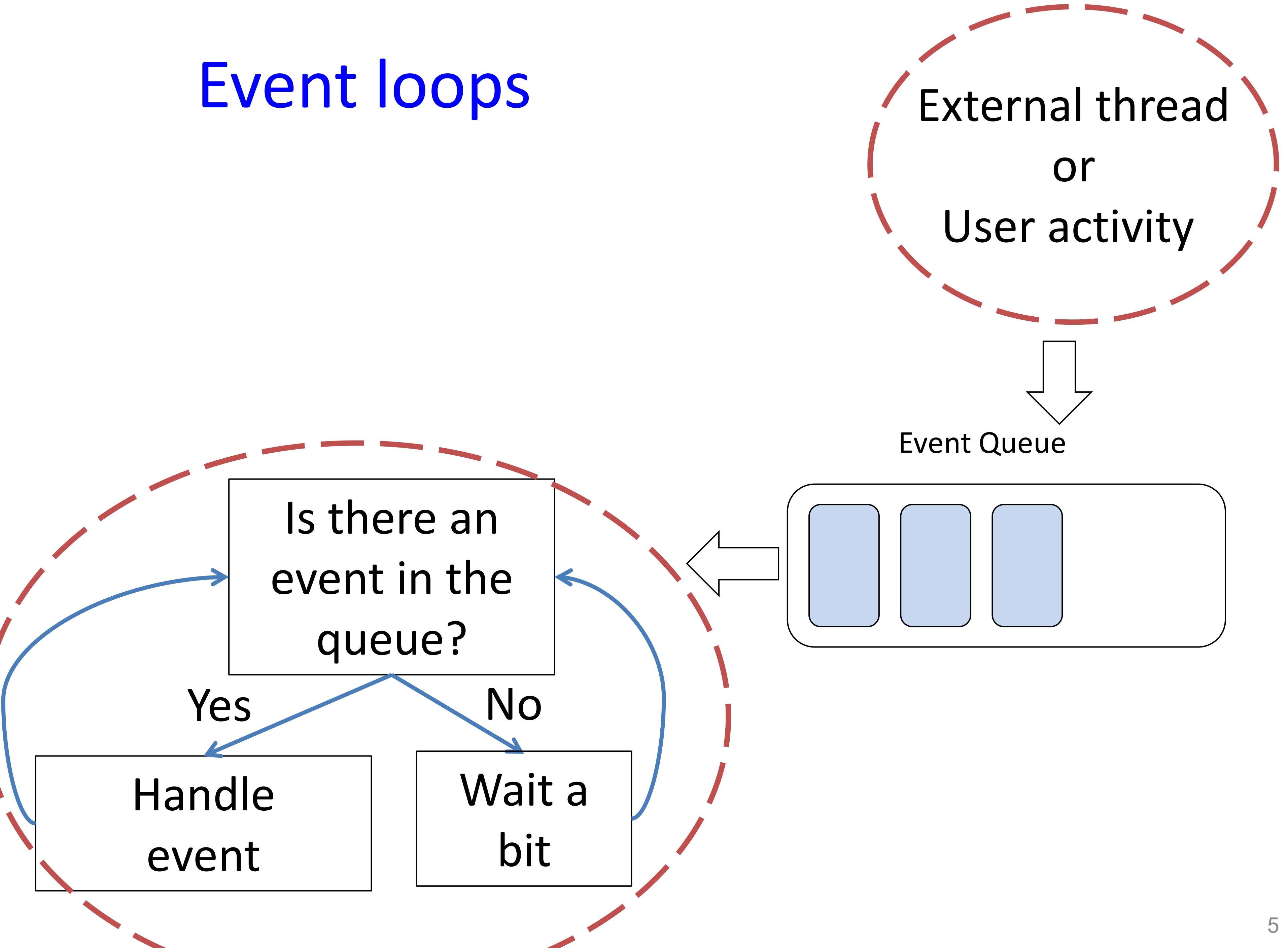
The event loop

- Programs often sit around waiting for input.





Event loops


External thread
or
User activity
Event Queue
Is there an event in the queue?
Yes
No
Handle event
Wait a bit

Graphical User Interfaces (GUI)

- GUI programs are usually constructed with an event loop. It is already implemented for you.

```
import tkinter

root = tkinter.Tk()

button1 = tkinter.Button(root, text = "Hello!")
button1.pack()

root.mainloop()
print("main loop ended. We are exiting.")
```

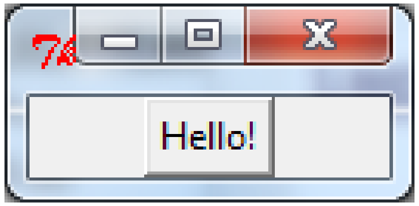
The GUI package we will use (comes with every python distribution)

Create a window

Create a button

Set location of the button in the window and make it visible

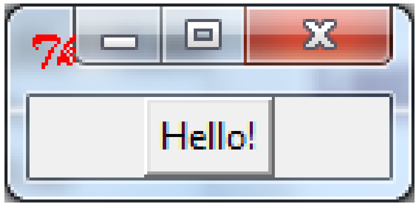
Start the event loop. You MUST do this.



Graphical User Interfaces (GUI)

- As program runs, execution stays with the event loop.
- Your code runs only when events occur.
- The user can freely interact with the GUI.
  - Resize window, click button, close window…

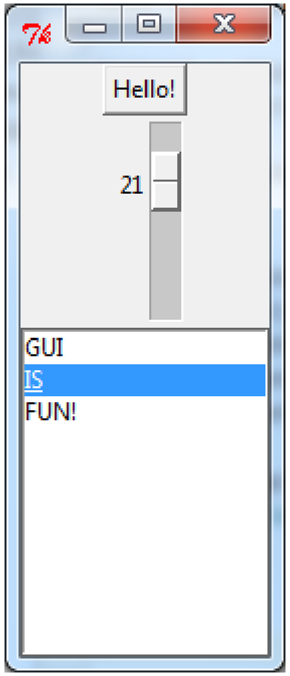
```
import tkinter

root = tkinter.Tk()

button1 = tkinter.Button(root, text = "Hello!")
button1.pack()

root.mainloop()
print("main loop ended. We are exiting.")
```



Widgets

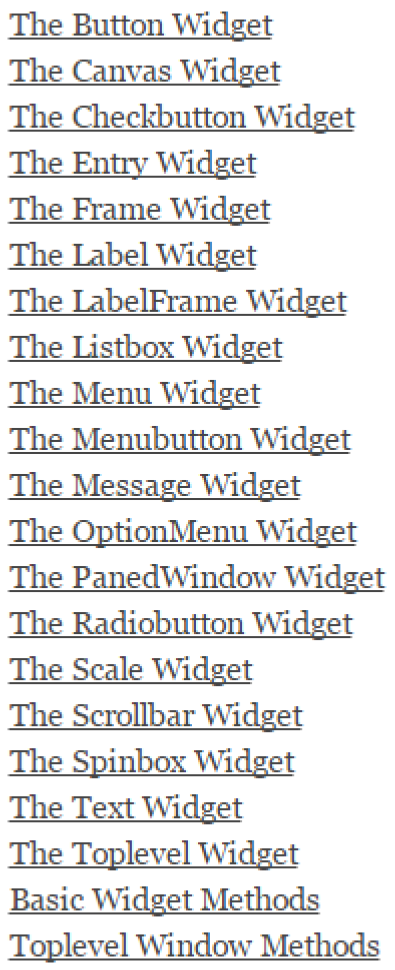
- There are many types of widgets

```
import tkinter

root = tkinter.Tk()

button1 = tkinter.Button(root, text="Hello!")
button1.pack()

w = tkinter.Scale(root, from_=0, to=100)
w.pack()

listbox = tkinter.Listbox(root)
listbox.pack()
listbox.insert(tkinter.END, "GUI")
listbox.insert(tkinter.END, "IS")
listbox.insert(tkinter.END, "FUN!")

root.mainloop()
print("main loop ended. We are exiting.")
```



Many widgets…

- You can create GUIs just like any “windows” program you know.
- Freely control color, fonts, behavior when resizing etc.
- Impossible to cover it all in class.
- Search online for details!



The Button Widget
The Canvas Widget
The Checkbutton Widget
The Entry Widget
The Frame Widget
The Label Widget
The LabelFrame Widget
The Listbox Widget
The Menu Widget
The Menubutton Widget
The Message Widget
The OptionMenu Widget
The PanedWindow Widget
The Radiobutton Widget
The Scale Widget
The Scrollbar Widget
The Spinbox Widget
The Text Widget
The Toplevel Widget
Basic Widget Methods
Toplevel Window Methods

http://effbot.org/tkinterbook/tkinter-index.htm

GUI programs

Two main things to take care of:

- Adding components and making it look “okay”
- Adding behavior.

(We will focus on behavior)

We will be defining events. What to do when things happen.

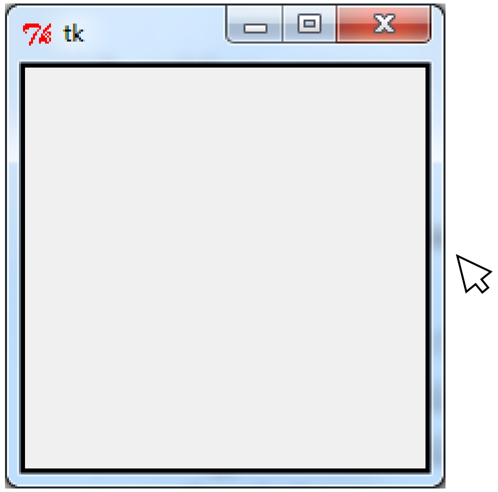
```
import tkinter as tki

class MyApp:
    def __init__(self,parent):
        self._parent = parent

        # add a canvas to draw on
        self._canvas = tki.Canvas(parent, width=200, height=200,
                                  highlightbackground='black')
        self._canvas.pack()

        # bind an event to the entry into the canvas
        self._canvas.bind("<Enter>", self._enter_event_handler)
        self._canvas.bind("<Leave>",
            lambda event: self._canvas.config(highlightbackground="black"))

    def _enter_event_handler(self,event):
        self._canvas.config(highlightbackground="red")

root = tki.Tk()
MyApp(root)
root.mainloop()
```

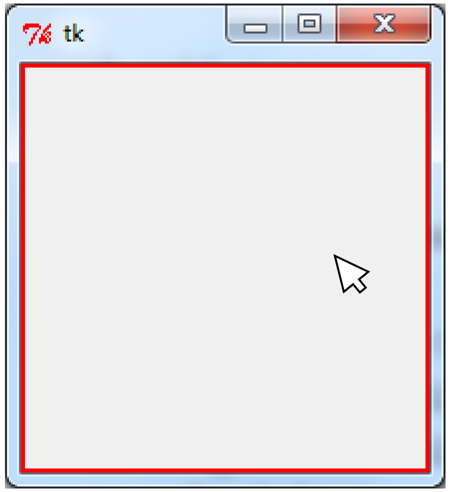
A class to handle our GUI

Bind an event to a function

We can do it with lambda expressions too!

Create and run everything


tk


tk

Events

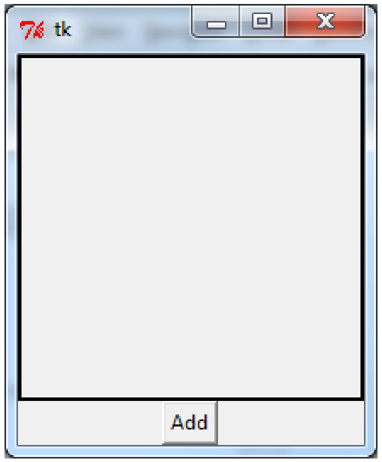
General events

- Mouse clicks,
- Keys getting pressed
- Focus changes
- Windows got resized, or somehow changed
- …

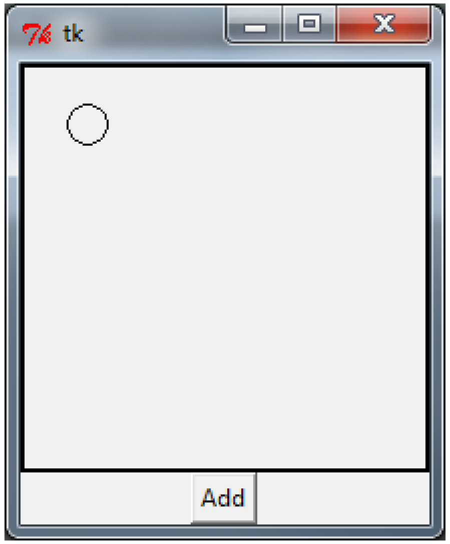
Action events from widgets

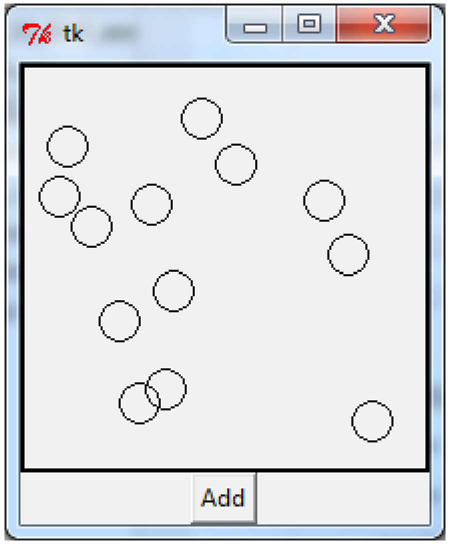
- Button widget clicked
- List selections changed
- Sliders moved
- …

```python
import tkinter as tki
import random

CANVAS_SIZE = 200
BALL_SIZE = 20
STEP_SIZE = 2

class MyApp:
    def __init__(self,parent):
        self._parent = parent

        # add a canvas to draw on
        self._canvas = tki.Canvas(parent,width=200, height=200,
                                  highlightbackground='black')
        self._canvas.pack()

        # add a button
        button = tki.Button(parent, text = "Add", command=self._add_ball)
        button.pack()
        self._balls = []
        self._move()
```



What to do when button pressed

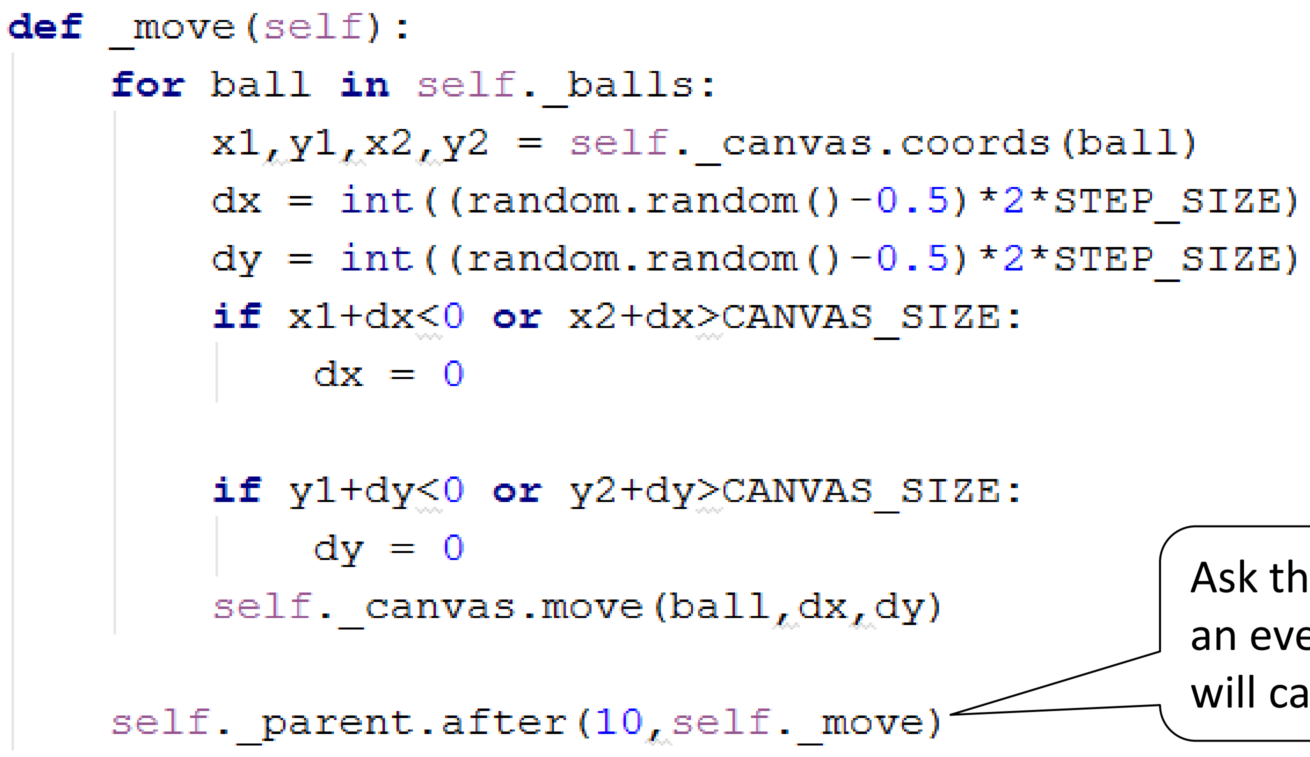




```
def _add_ball(self):
    x = random.randrange(CANVAS_SIZE-BALL_SIZE)
    y = random.randrange(CANVAS_SIZE-BALL_SIZE)
    self._balls.append(self._canvas.create_oval(x, y, x+BALL_SIZE, y+BALL_SIZE))
```

We want to get the balls to move all the time on the screen.

Writing a loop to do it would be bad. (WHY?)
Instead:



```
def _move(self):
    for ball in self._balls:
        x1,y1,x2,y2 = self._canvas.coords(ball)
        dx = int((random.random()-0.5)*2*STEP_SIZE)
        dy = int((random.random()-0.5)*2*STEP_SIZE)
        if x1+dx<0 or x2+dx>CANVAS_SIZE:
            dx = 0

        if y1+dy<0 or y2+dy>CANVAS_SIZE:
            dy = 0
        self._canvas.move(ball,dx,dy)

    self._parent.after(10,self._move)
```

Ask the event loop to add an event in 10 milisecs that will call this method

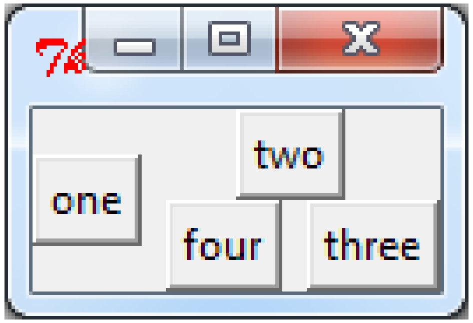
Mouse button events

```
import tkinter as tki

class MyApp:
    def __init__(self,parent):
        self._parent = parent

        label = tki.Label(parent,highlightbackground='black')
        label.pack()

        button = tki.Button(parent, text="click_me")
        button.pack()

        button["command"] = lambda: label.configure(text="Click!")
        button.bind("<Button-1>", lambda event: label.configure(text="Press"))

        label.bind("<Button-1>", lambda event: label.configure(text="Press"))
        label.bind("<ButtonRelease-1>", lambda event: label.configure(text="Release"))
        label.bind("<Double-Button-1>", lambda event: label.configure(text="Double Click"))
        label.bind("<Triple-Button-1>", lambda event: label.configure(text="Triple Click"))

root = tki.Tk()
MyApp(root)
root.mainloop()
```

Layout

- Pack can be asked to place things at the top,bottom,right or left (top is the default).

```
import tkinter as tki

class MyApp:
    def __init__(self,root):
        b1 = tki.Button(root, text = "one")
        b2 = tki.Button(root, text = "two")
        b3 = tki.Button(root, text = "three")
        b4 = tki.Button(root, text = "four")

        b1.pack(side=tki.LEFT)
        b2.pack(side=tki.TOP)
        b3.pack(side=tki.RIGHT)
        b4.pack(side=tki.BOTTOM)

root = tki.Tk()
MyApp(root)
root.mainloop()
```

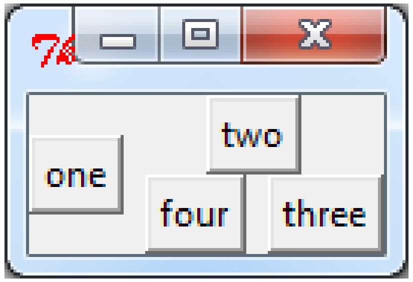


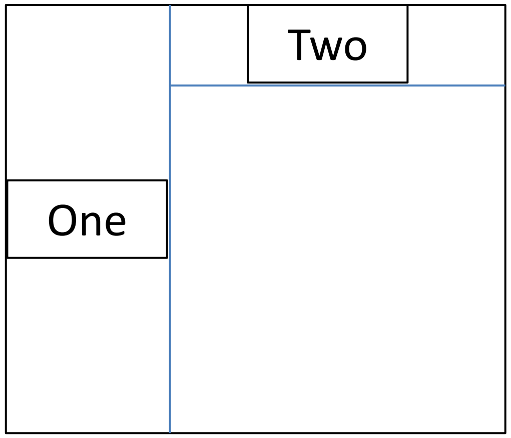
Layout

- Pack can be asked to place things at the top,bottom,right or left (top is the default).

```
import tkinter as tki

class MyApp:
    def __init__(self,root):
        b1 = tki.Button(root, text = "one")
        b2 = tki.Button(root, text = "two")
        b3 = tki.Button(root, text = "three")
        b4 = tki.Button(root, text = "four")

        b1.pack(side=tki.LEFT)
        b2.pack(side=tki.TOP)
        b3.pack(side=tki.RIGHT)
        b4.pack(side=tki.BOTTOM)

root = tki.Tk()
MyApp(root)
root.mainloop()
```

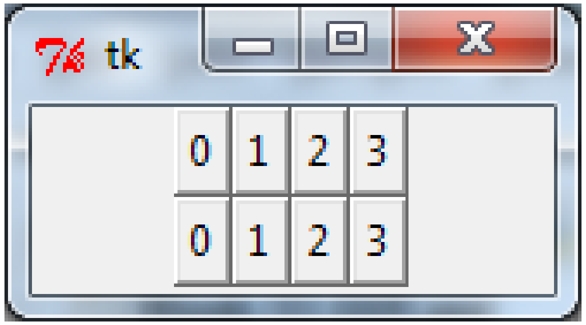
Things that are packed later will be in the remaining "cavity"





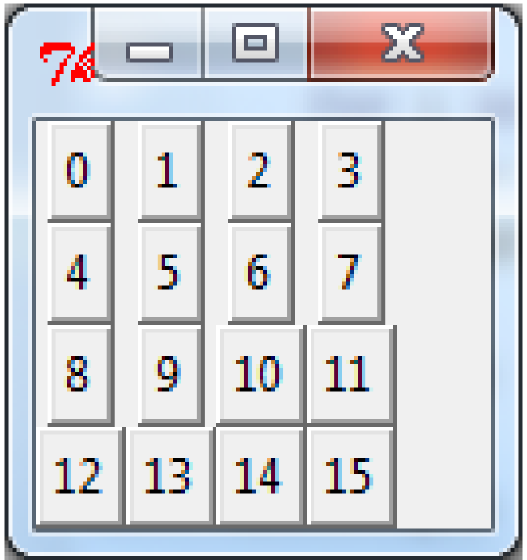
Using frames to organize things

```
import tkinter as tki

class MyApp:
    def __init__(self,root):

        top_frame = tki.Frame(root)
        bottom_frame = tki.Frame(root)
        top_frame.pack()
        bottom_frame.pack()

        for i in range(4):
            b = tki.Button(top_frame,text=str(i))
            b.pack(side = tki.LEFT)
            b = tki.Button(bottom_frame,text=str(i))
            b.pack(side = tki.LEFT)

root = tki.Tk()
MyApp(root)
root.mainloop()
```

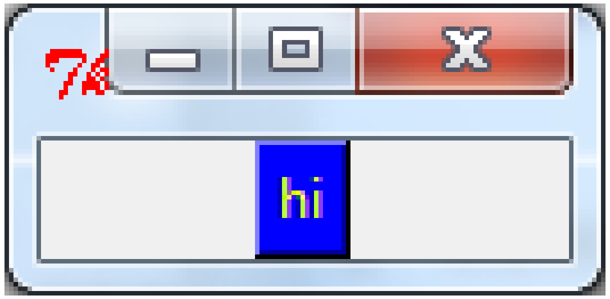


grid

```
import tkinter as tki

class MyApp:
    def __init__(self,root):

        for i in range(16):
            b = tki.Button(root ,text=str(i))
            b.grid(row=i//4, column = i%4)

root = tki.Tk()
MyApp(root)
root.mainloop()
```



Recap (1)



```
import tkinter as tki

# Create a main window
root = tki.Tk()

# Creating widgets
button = tki.Button(root, text = "hi")

# Configuring widgets
button.configure(fg="yellow")
button["bg"] = "blue"

# Packing
button.pack(side=tki.TOP)
```

Container

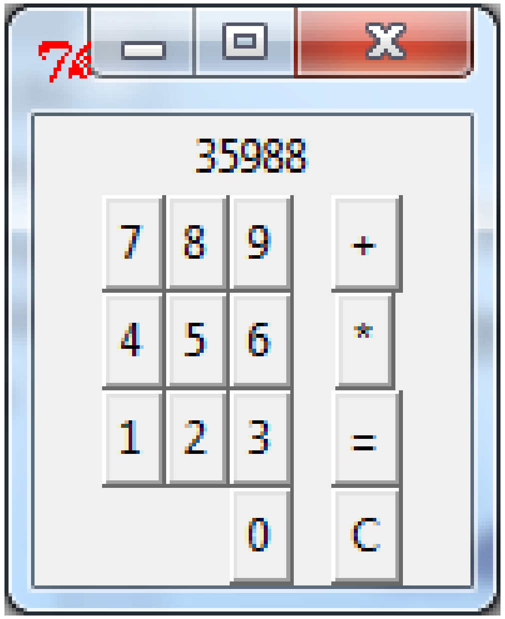
configuration

More configuration

Even more configuration

Recap (2)

```python
# Event handlers
def my_event_handler(event):
    print("entering " + event.widget["text"])

# Binding event handlers to widget
button.bind("<Enter>",my_event_handler)

# Listen to events generated by the button
button.configure(command=lambda: print("clicking hi"))

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


35988
7
8
9
+
4
5
6
*
1
2
3
=
0
C

```python
class MyCalculatorApp:
    NUM_DIGITS = 10

    def __init__(self,parent):
        self._parent = parent
        self._display_label = tki.Label(parent)
        self._display_label.pack(side=tki.TOP)

        lower_frame = tki.Frame(parent)
        lower_frame.pack()

        self._create_digit_buttons(lower_frame)
        self._create_op_buttons(lower_frame)

        self._reset()
```

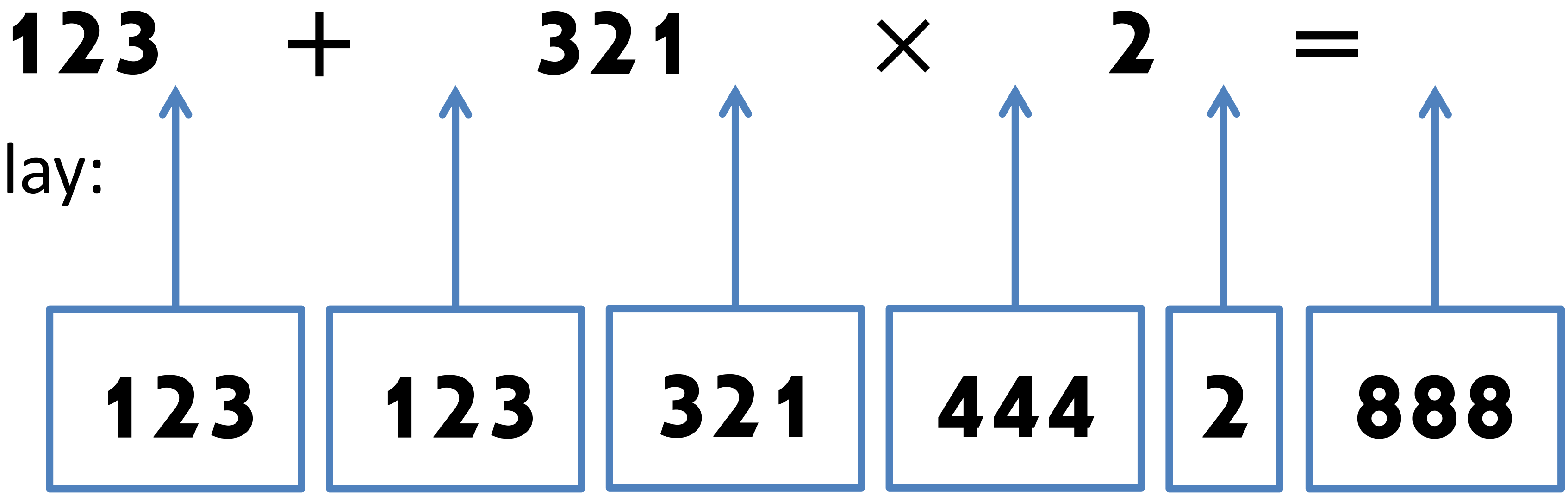
```python
def _reset(self):
    self._display_label.configure(text="0")
    self._current_num = ""
    self._prev_num = 0
    self._prev_op = lambda x, y: x+y
```

```
def _create_digit_buttons(self,parent):
    digit_frame = tki.Frame(parent)
    digit_frame.pack(side=tki.LEFT)

    for digit in range(MyCalculatorApp.NUM_DIGITS):
        button = tki.Button(digit_frame,
                            text=str(digit),
                            command=self._digit_event_h(digit))
        button.grid(row = 3-(digit+2)//3, column=(digit-1)%3)



def _digit_event_h(self,digit):
    def digit_press():
        self._current_num += str(digit)
        self._display_label.configure(text = self._current_num)
    return digit_press
```

What to do when a user presses an operator?

User Pressed:

Display:



When an operator is pressed we apply the **previous** operator to old + new number & store the result.

```
def _create_op_buttons(self,parent):
    separator = tki.Frame(parent, width=10)
    separator.pack(side = tki.LEFT)
    op_frame = tki.Frame(parent)
    op_frame.pack(side=tki.LEFT)

    plus_button = tki.Button(op_frame,text="+",
                             command=self._op_event_h(lambda x, y: x+y))
    times_button = tki.Button(op_frame,text="*",
                              command=self._op_event_h(lambda x, y: x*y))
    op_equals = lambda x, y: x if self._current_num == "" else y
    eq_button = tki.Button(op_frame, text="=",
                           command=self._op_event_h(op_equals))
    clear_button = tki.Button(op_frame, text="C",
                              command=self._reset)

    plus_button.pack(side=tki.TOP)
     …
```

Using many tools

A closure,
with a lambda expression,
That is used as an event handler.

```
plus_button = tki.Button(op_frame,text="+",
                         command=self._op_event_h(lambda x, y: x+y))
```

```
def _op_event_h(self,op_func):
    def op_event():
        if self._current_num == "":
            cur_num = 0
        else:
            cur_num = int(self._current_num)
        self._prev_num = self._prev_op(self._prev_num,int(cur_num))
        self._prev_op = op_func
        self._current_num = ""
        self._display_label.configure(text=str(self._prev_num))
    return op_event
```