Python Object-Oriented Program with Libraries

Unit 4: Tkinter GUI Programming

CHAPTER 1: TKINTER GUI ENVIRONMENT

DR. ERIC CHOU

IEEE SENIOR MEMBER

Objectives

In this chapter, you will:

- Become aware of using Tkinter module to build graphical user interfaces
- Create and manipulate labels, text fields, buttons, check boxes, and radio buttons
- Learn to use mouse events and keyboard events

Graphics User Interface

LECTURE 1

What Is a User Interface?

A set of hardware devices (touch screen, monitor, keyboard, mouse, microphone, speakers)

Software (input/output functions)

Allows human beings to use a computer effectively


Crush
AGA

Text-Based User Interface (TUI)

- Supports input via the keyboard and output via the monitor
- In Python, the I/O functions are **input** and **print**

```
import math
radius = float(input('Enter the radius: '))
area = math.pi * radius ** 2
print('The area is', area)
```

Text-Based User Interface (TUI)

- Supports input via the keyboard and output via the monitor
- In Python, the I/O functions are **`input`** and **`print`**

```
import math
radius = float(input('Enter the radius: '))
area = math.pi * radius ** 2
print('The area is', area)
```

```
ken — bash — 44×7
Last login: Sat Aug 25 12:56:53 on ttys000
Madison:~ ken$ python3 circlearea.py
Enter the radius: 34.5
The area is 3739.280655935251
Madison:~ ken$
```

Problems with a TUI

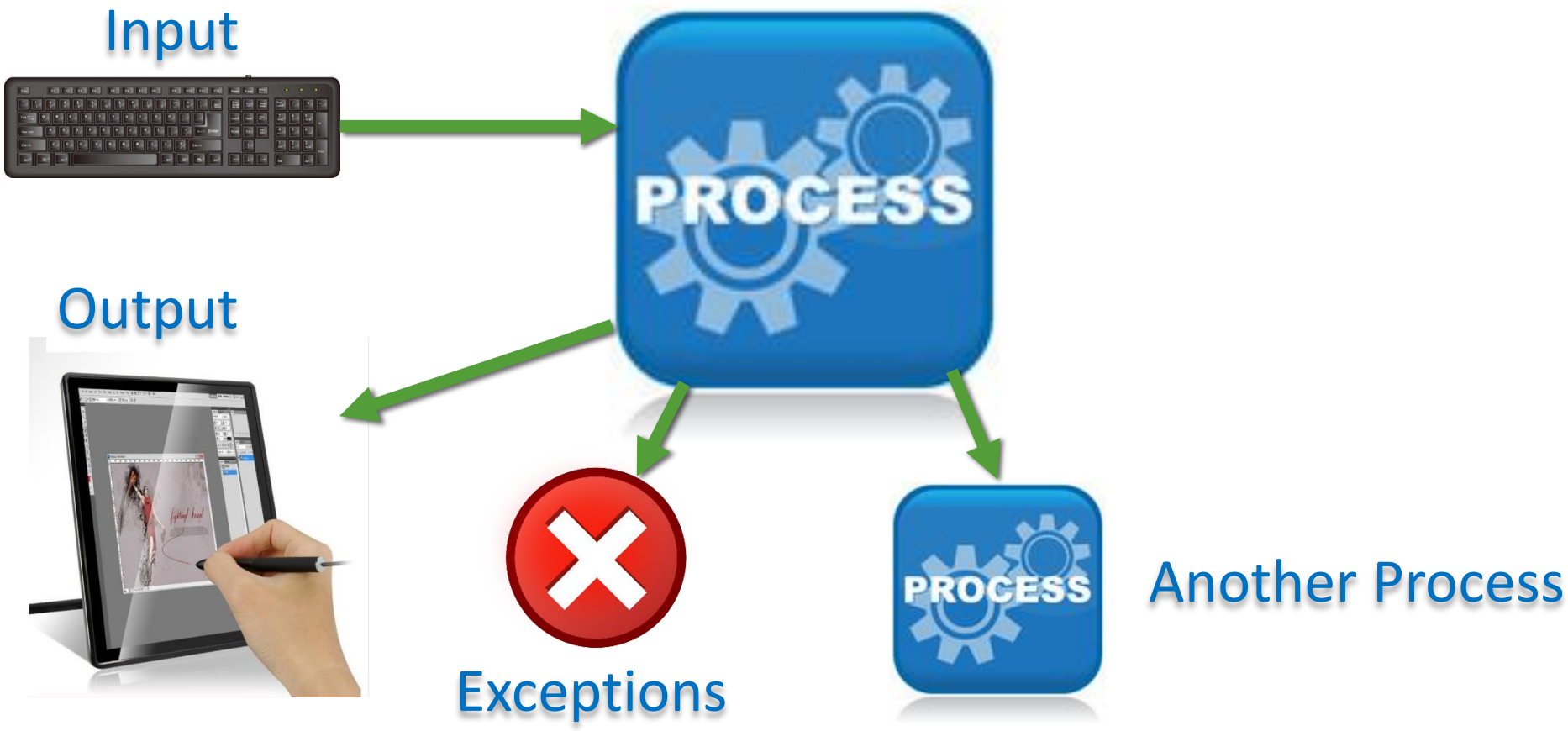
- Must enter inputs in a certain order
- Cannot back up to correct input mistakes or change one’s mind
- Must re-enter all inputs to change just one
- I/O restricted to text

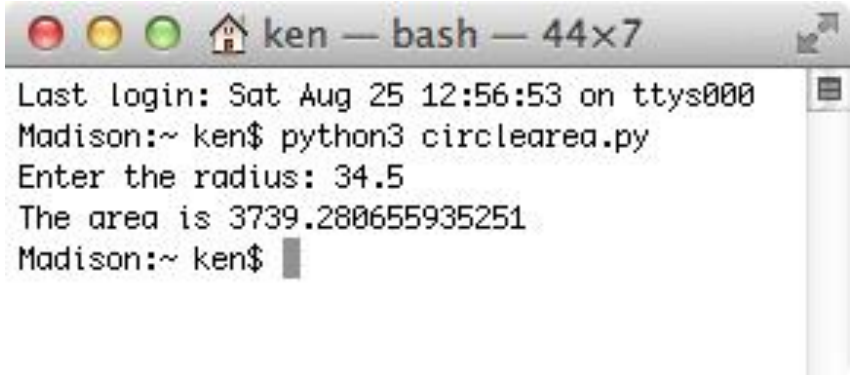
Graphical User Interface (GUI)

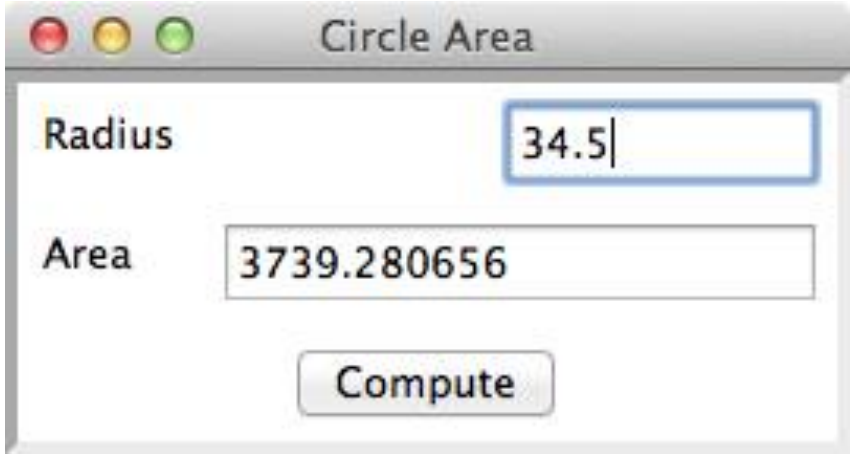
- Supports input via the keyboard, touch screen, audio and etc.
- Can output text and also graphical shapes representing desktop elements, such as windows, command buttons, data fields, and drop-down menus (also called “widgets”)
- Supports direct manipulation of desktop elements via the mouse or touchscreen

Graphics User Interface is

Modern-Day Input/Output Interface for Computer Programs







TUI vs GUI

- Non-programmers (the 99%) do not use a TUI, they use a GUI
- Only programmers (the 1%) use a TUI (and also a GUI)
- Most beginning programmers program to a TUI, not a GUI

Models of Computation

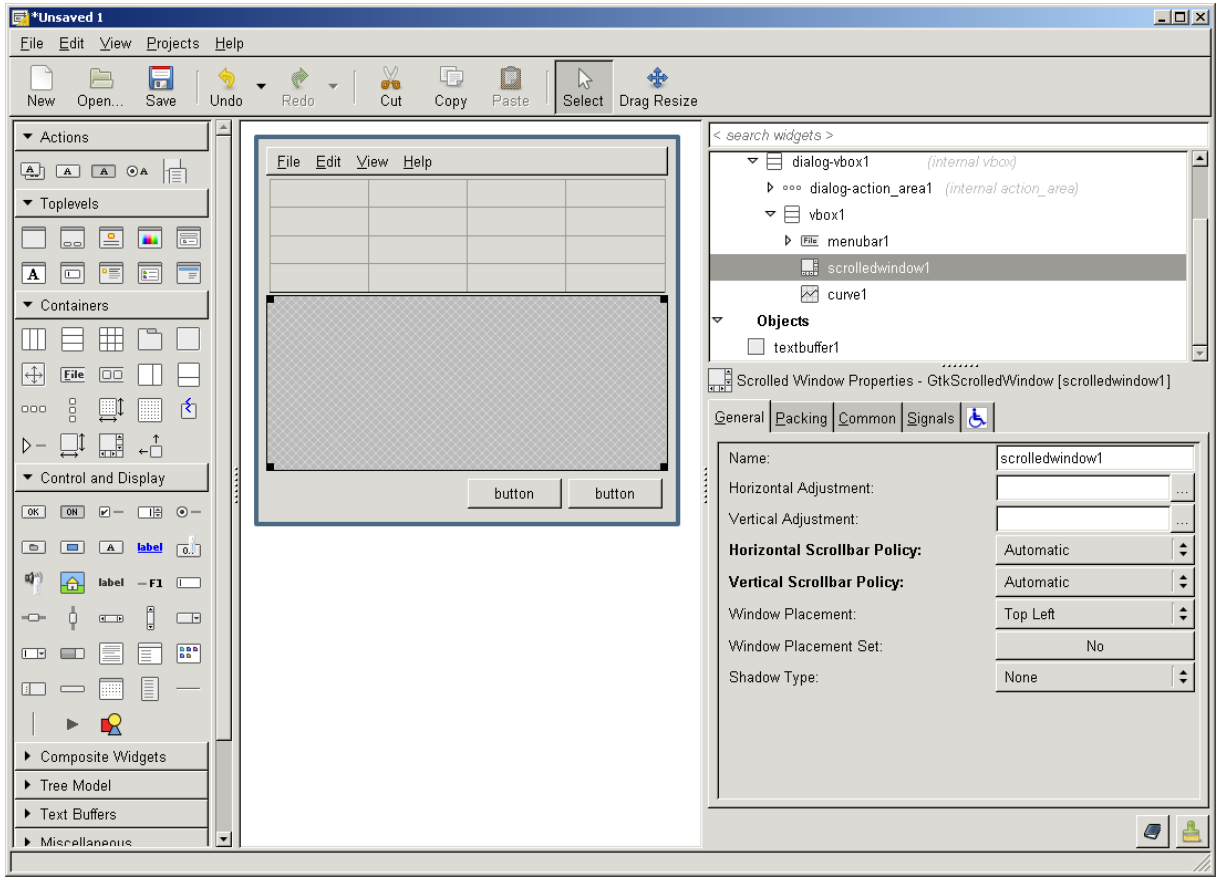
Text-Based User Interface	Graphics User Interface
1. Obtain user inputs	1. Layout and pop up the window
2. Perform computations	2. Wait for user events
3. Print results	3. Handle a user event
	4. Goto step 2

GUI Packages for Python

LECTURE 2

Connect GUI with Python

- Direct via script
- GUI design tools and IDEs (**I**ntegrated **D**evelopment **E**nvironments)
  - GUI builder
  - GUI to build a GUI application
  - Useful especially for larger projects
  - Output usually xml
    - Use directly
    - Translate to Python script



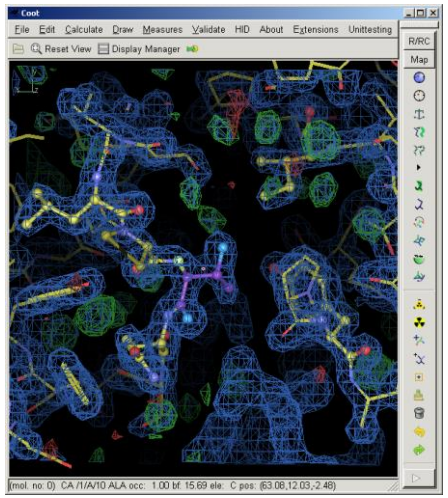
GUI modules for Python (I)(Toolkits, cross-platform)

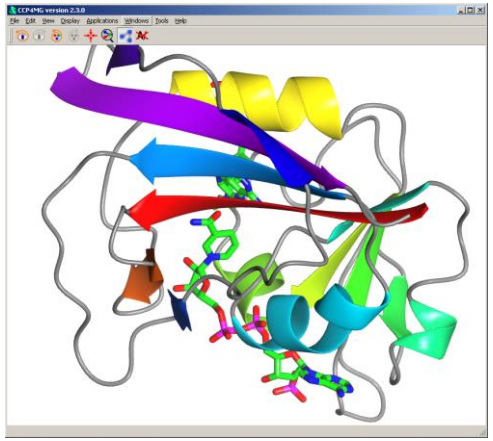
PyGTK (Gnome):

- TK: GTK+ (GIMP Tool Kit)
- Well supported
- Builder (Glade)
  - xml direct
- Not native on Mac

PyQt (KDE):

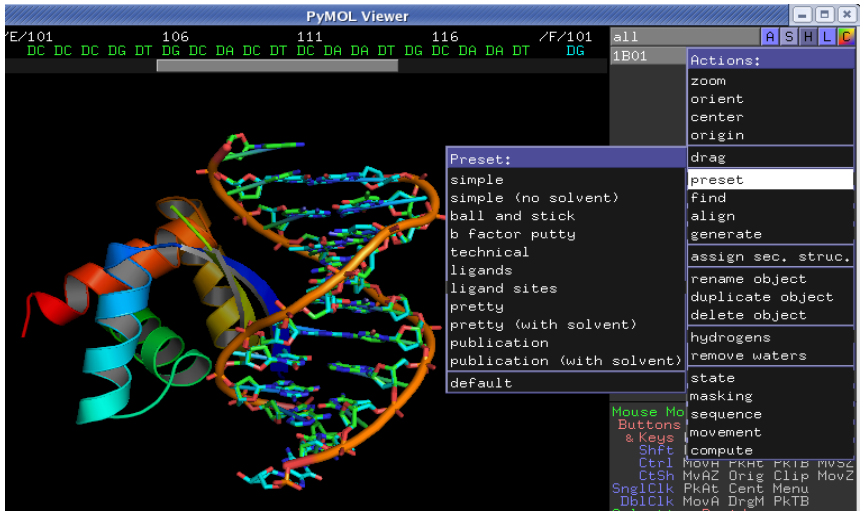
- TK: Qt (“cu-te”)
- Licence issue?
- Native
- Builder (Designer)
  - xml->python

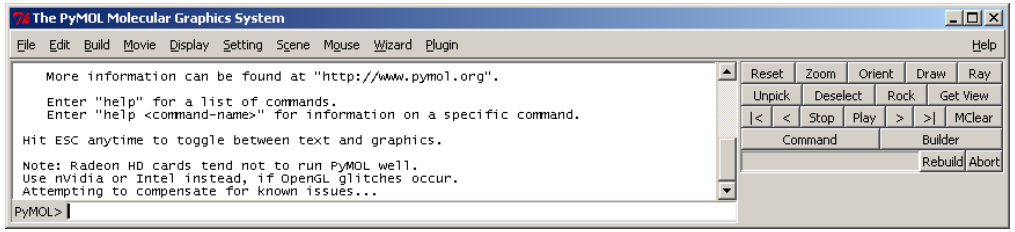


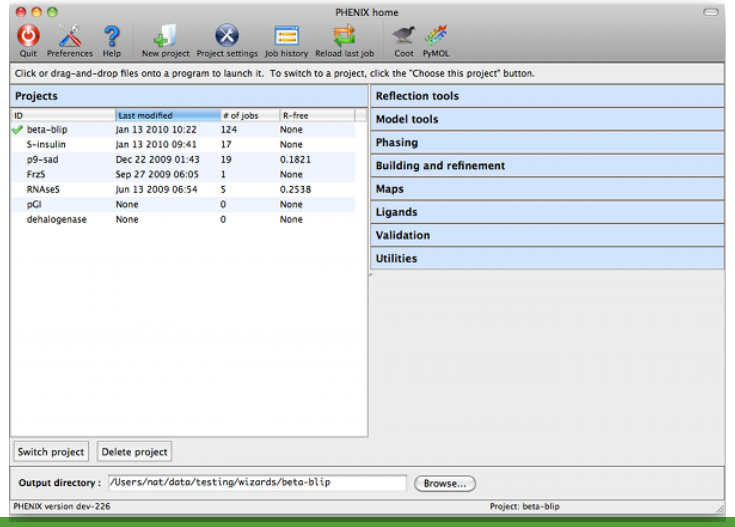


GUI modules for Python (II)

- Tk(Inter):
  - TK: Tk
  - Tkinter Python *Tk interface*
  - Distributed with python
  - Builder (GUI builder)
- wxPython: **Phenix**
  - TK: wxWidgets
  - Builder (wxGlade)
  - Native







7 Top Python GUI Frameworks for 2017

How to choose between all these options for Python GUI?

- Platforms: Windows, Mac, and Linux
- Interpreter: Python 2, or 3.
- Toolkits: Gtk, Qt, Tk, and wxWidgets)
- Frameworks (Kivy, PyQt, gui2Py, libavg, wxPython, Pyforms, and PyGOBjects)
- GUI Designer (for Tkinter): VisualTkinter, PAGE, and Pygubu

Our Examples are on Windows, Python 3, Tk, Tkinter, PAGE (Because of simplicity and availability in Python built-in library)

tkinter module

LECTURE 3

tkinter

The **Tkinter** module (“Tk interface”) is the standard Python interface to the Tk GUI toolkit from **Scriptics** (formerly developed by Sun Labs).

Both **Tk** and **Tkinter** are available on most Unix platforms, as well as on Windows and Macintosh systems.

Starting with the 8.0 release, **Tk** offers native look and feel on all platforms.

Tkinter consists of a number of modules. The **Tk** interface is provided by a binary extension module named **_tkinter**. This module contains the low-level interface to **Tk**, and should never be used directly by application programmers. It is usually a shared library (or DLL), but might in some cases be statically linked with the Python interpreter.

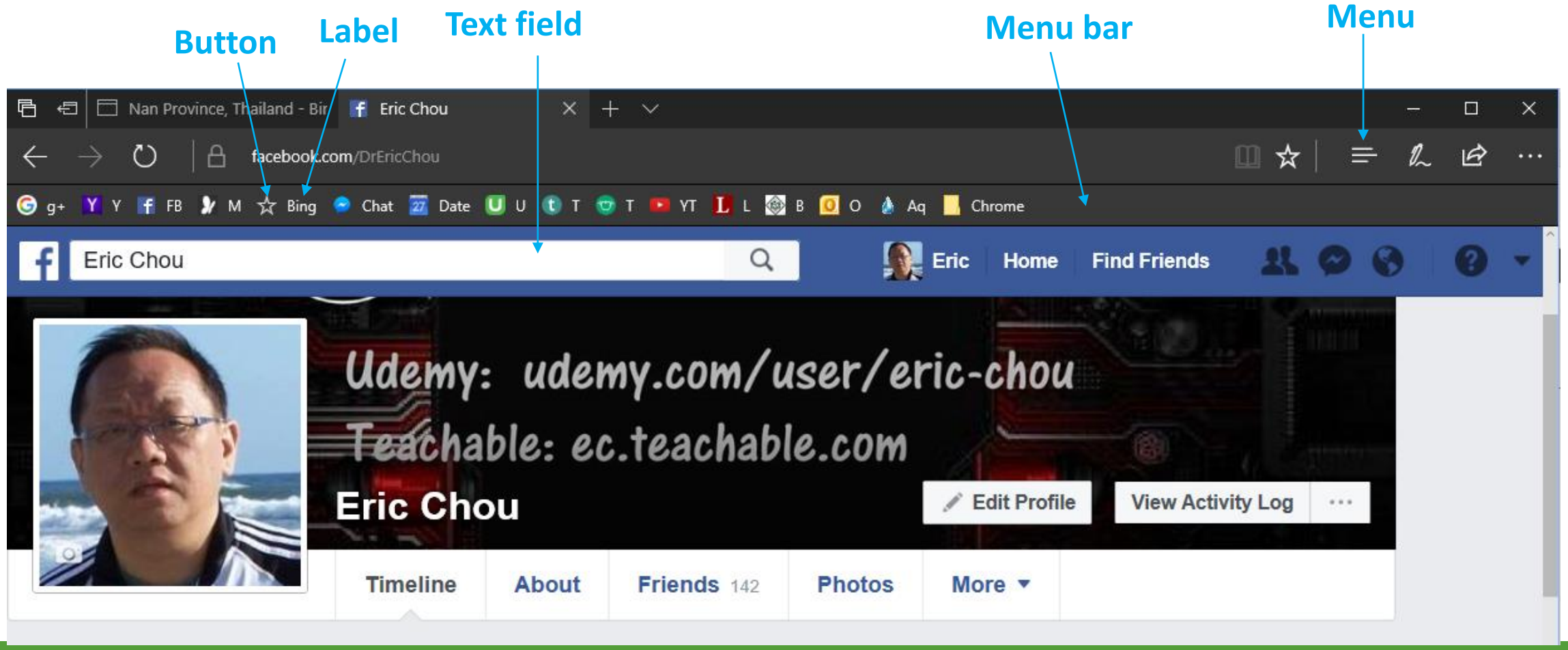
Windowing system/hierarchy

Our script (in Python): Real application
TK (in Python): glue to TK
TK Python plugin (usually in C): translate to TK calls
TK widgets (usually in C): Widget implementation
TK library (usually C): glue to windowing system
Windows managing system (e.g. X)

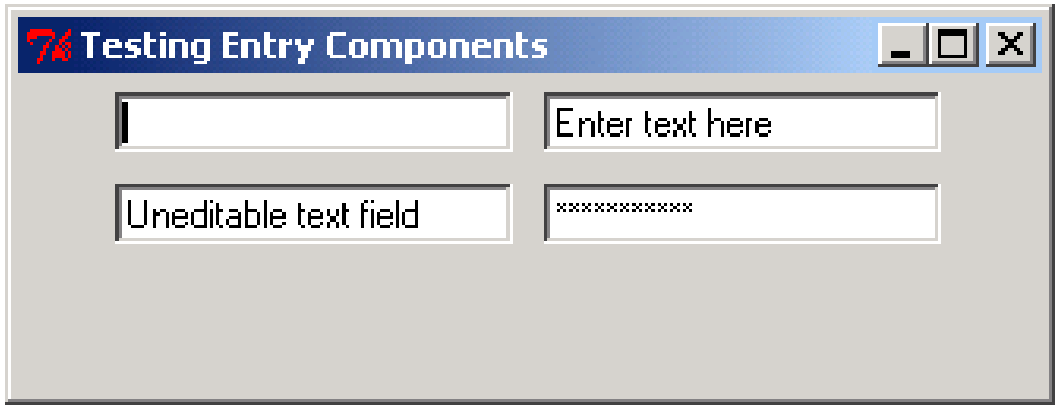
Tk Overview

- Set of widgets designed by John K. Ousterhout, 1987
- Based on Apple Hypercard idea of putting together graphics program
- Tk means Tool Kit
- Mean to be driven by Tcl (Toolkit Control Language)
  - Many people find Tcl limited
  - Can also drive Tk with Perl, Python
- Tkinter is the Python Tk Interface
  - Very easy to use

An Example of GUI Components in an Edge browser



Python GUIs Components



Event Handling

GUI components generate events due to user interaction

Events drive the program to perform a task

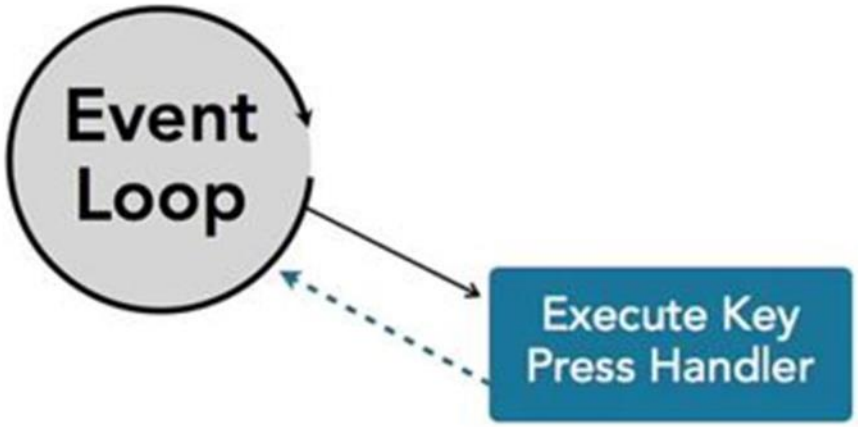
- Click (onClick)
- Change (onChange)
- Time (Sleep)

Event handler is what you write

Event-driven programming

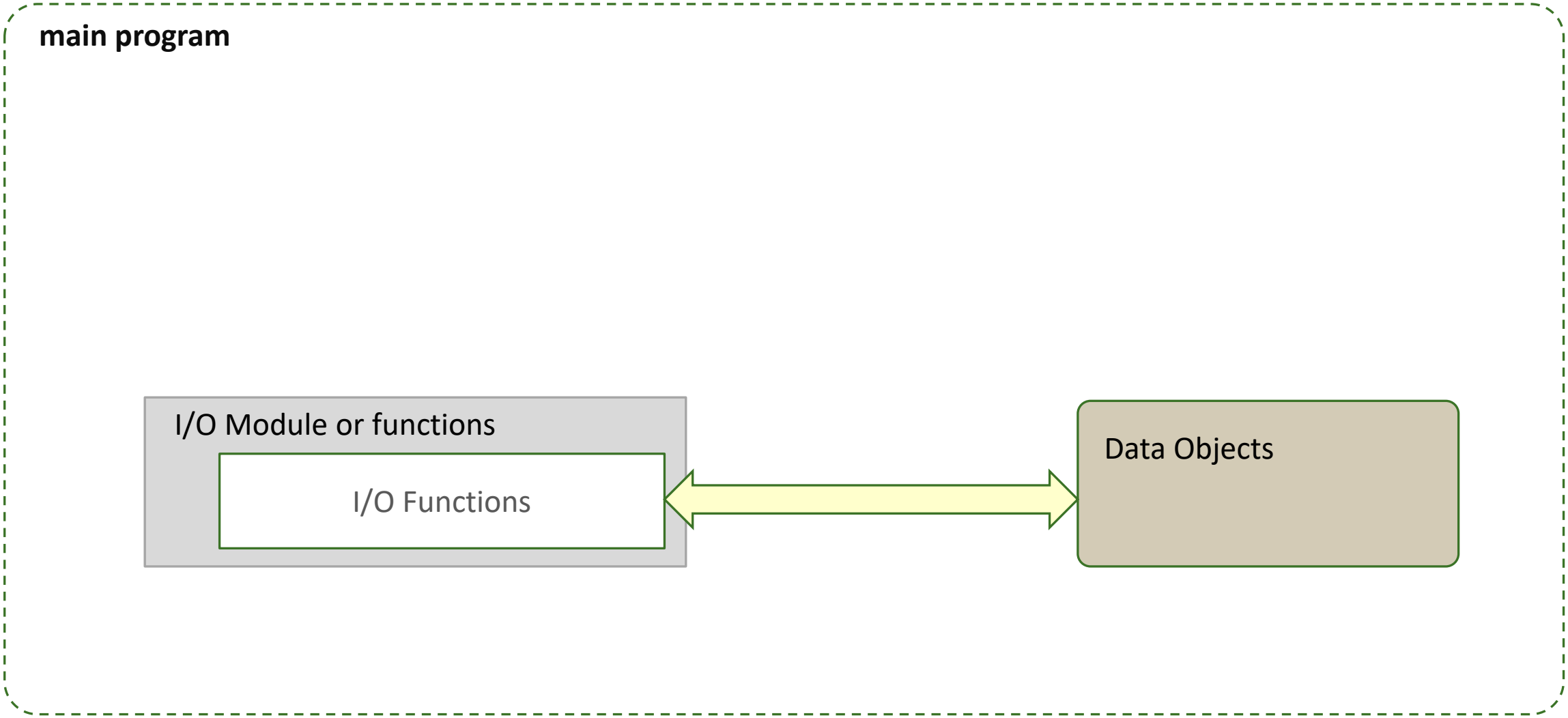
When a GUI is started with the `mainloop()` method call, Python starts an infinite loop called an event loop

```
while True:

    1. wait for an event to occur

    2. run the associated event handler
```

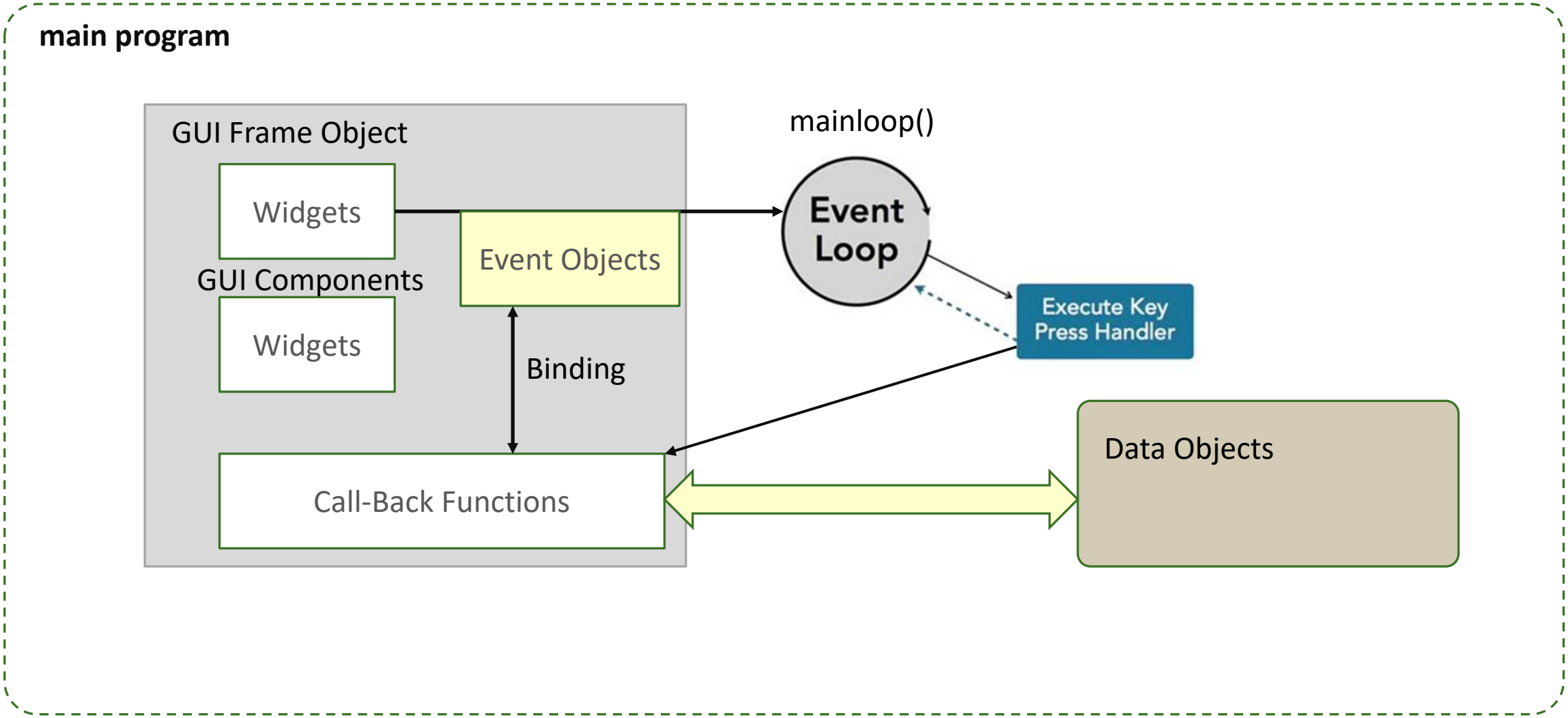


Event-driven programming is the programming approach used to build applications whose execution flow is determined by events and described using an event loop

Program with TUI



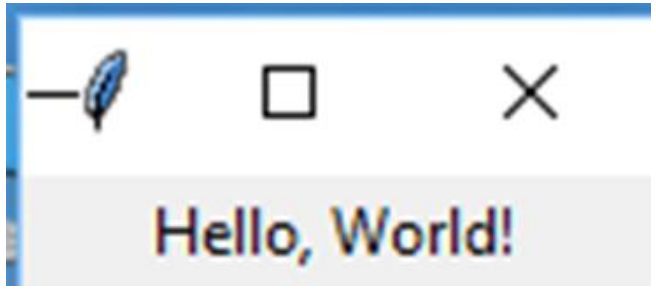
Program with tkinter GUI



Demo Program

helloworld.py and helloworldG.py

Go PyCharm!!!



Hello, World

No-Frame Container – Minimum GUI Program

```
from Tkinter import *
w=Label(text="Hello, World!")
w.pack()
w.mainloop()
```

G

```
print("Hello World!")
```

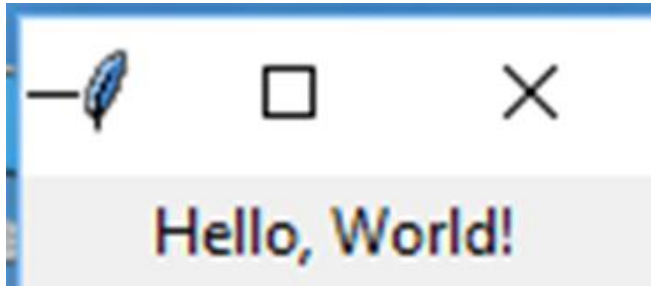
T

- **Label()** defines a label to be displayed
  - text= specifies a parameter to be passed in
- **pack()** resizes the window to the proper size
- **mainloop()** enters the event loop, and the program idles until a button is pushed, a menu is pulled, etc. It has to idle until the program is killed, since we didn't define any events.

Demo Program

helloworld2.py and helloworld2G.py

Go PyCharm!!!



Events (Hello, Goodbye)

No-Frame Container – Minimum GUI Program with Default Event Handler

```
from tkinter import *
w=Label(text="Hello, World)")
w.pack()
b=Button(text="Goodbye",command='exit')
b.pack()
mainloop()
```

Note: For button, default event is on-click

G

```
print("Hello World!")
```

T

- **Button** label defined by text parameter
- **Button** defines a **callback function**, something to run when it is pushed.
- Now mainloop() has an event to catch, so when we push the button, mainloop() executes the **exit** command.

widgets

LECTURE 4

tkinter Module

- Python's standard GUI package – **tkinter** module
- **tkinter** library provides object-oriented interface to Tk GUI toolkit
- Each GUI component class inherits from class **Widget**
- GUI consists of top-level (or parent) component that can contain children components
- Class **Frame** serves as a top-level component

```
try:
    from Tkinter import *     # for Python2
except ImportError:
    from tkinter import *     # for Python3
```

widget

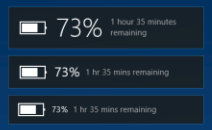
Pronounced wih-jit.

- Widget is a generic term for the part of a GUI that allows the user to interface with the application and operating system.
- Widgets display information and invite the user to act in a number of ways.
- Typical widgets include **buttons, dialog boxes, pop-up windows, pull-down menus, icons, scroll bars, resizable window edges, progress indicators, selection boxes, windows, tear-off menus, menu bars, toggle switches** and **forms**.

WIN10 **WIDGETS**

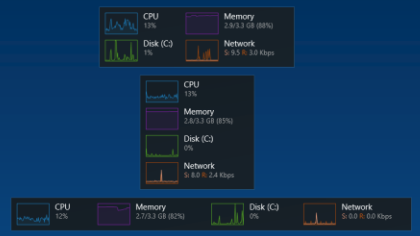
*FOR RAINMETER **BY TJ MARKHAM***

BATTERY

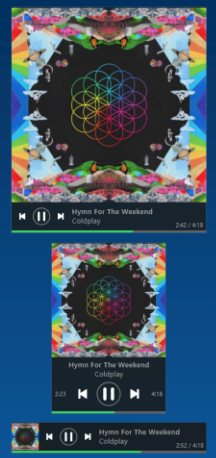


HARD DRIVE

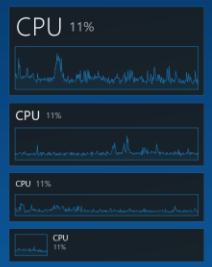
PERFORMANCE



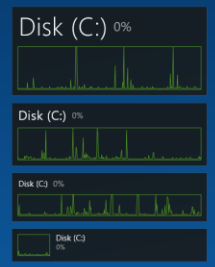
SPOTIFY



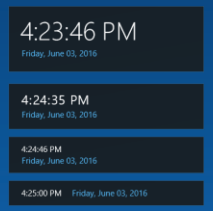
CPU



DISK

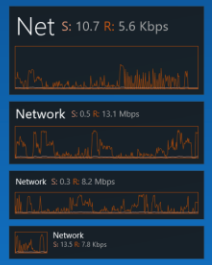


DATE TIME



WIFI

NETWORK

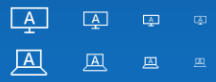


MEMORY

VOLUME

LOCK

LAYOUT SWITCHER



WEATHER

win10widgets.com

widget

- The term also refers to the program that is written in order to make the graphic widget in the GUI look and perform in a specified way, depending on what action the user takes while interfacing with the GUI.
- The term widget is used to refer to either the graphic component or its controlling program or to refer to the combination of both.

Python 2.X to Python 3.X

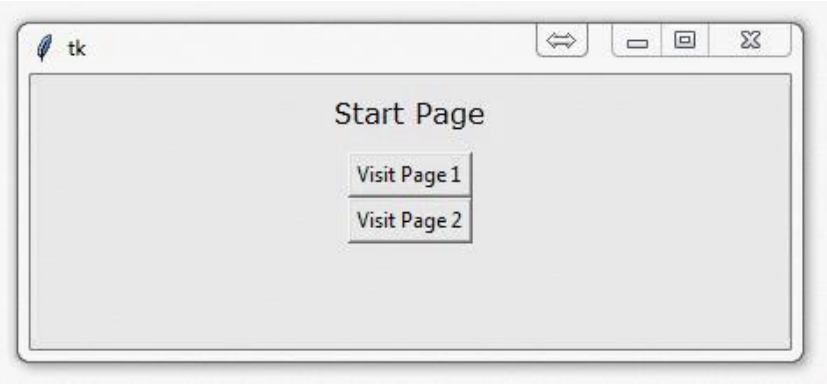
Tkinter → **tkinter**
tkMessageBox → **tkinter.messagebox**
tkColorChooser → tkinter.colorchooser
tkFileDialog → tkinter.filedialog
tkCommonDialog → tkinter.commondialog
tkSimpleDialog → tkinter.simpledialog
tkFont → tkinter.font
Tkdnd → tkinter.dnd
ScrolledText → tkinter.scrolledtext
Tix → tkinter.tix
ttk → tkinter.ttk

GUI Design (Web Page-Equivalent)

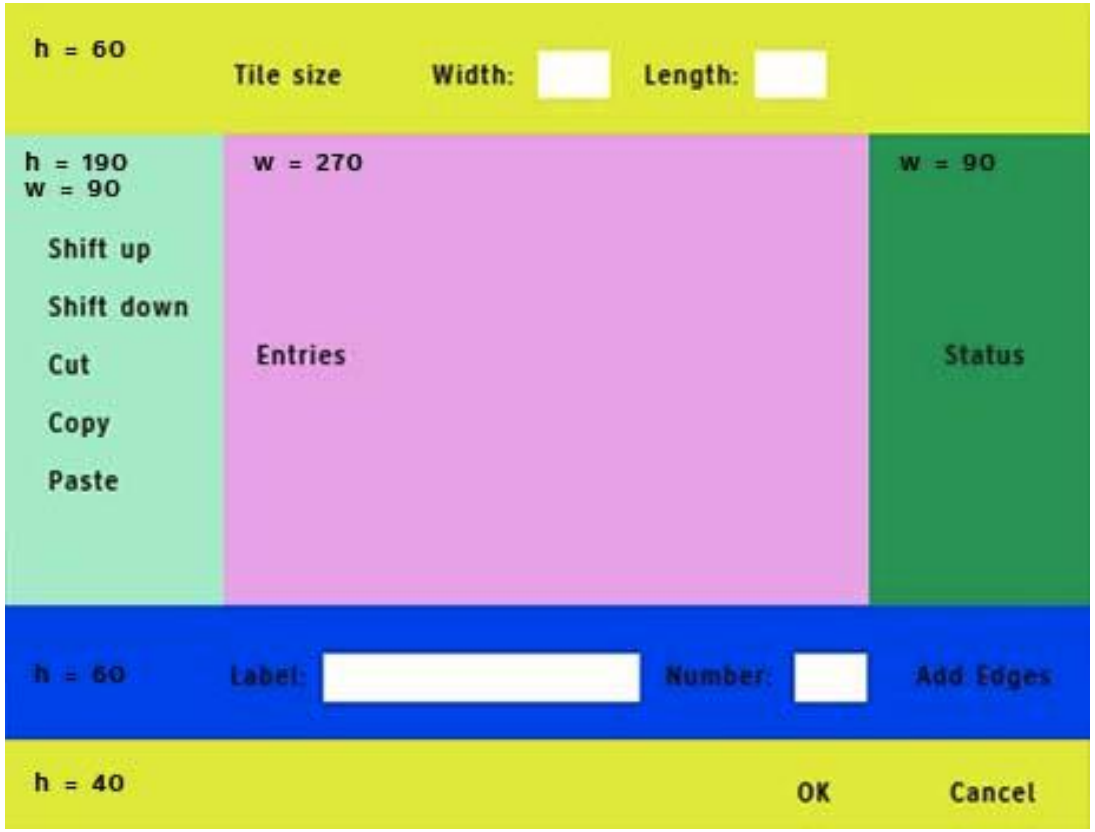
- Structural (HTML)
- Content and Style (CSS)
- Behavioral (Javascript)

Tkinter GUI Technology (I)

Container – Frame (Structural)

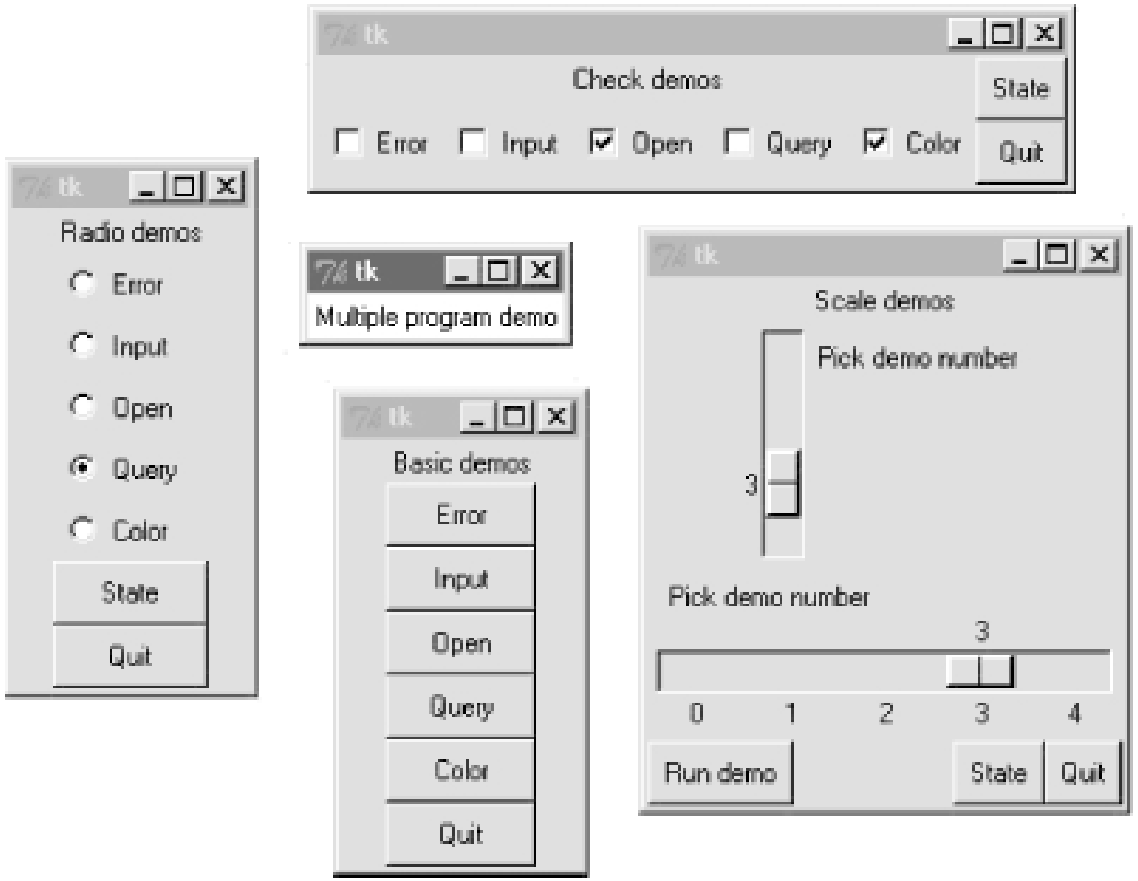


Layout Management (Structural)

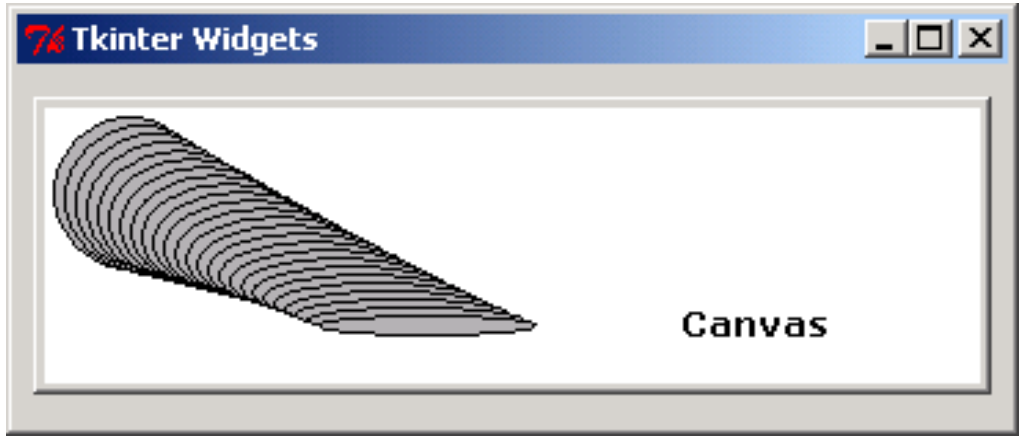


Tkinter GUI Technology (II)

Components (Content)

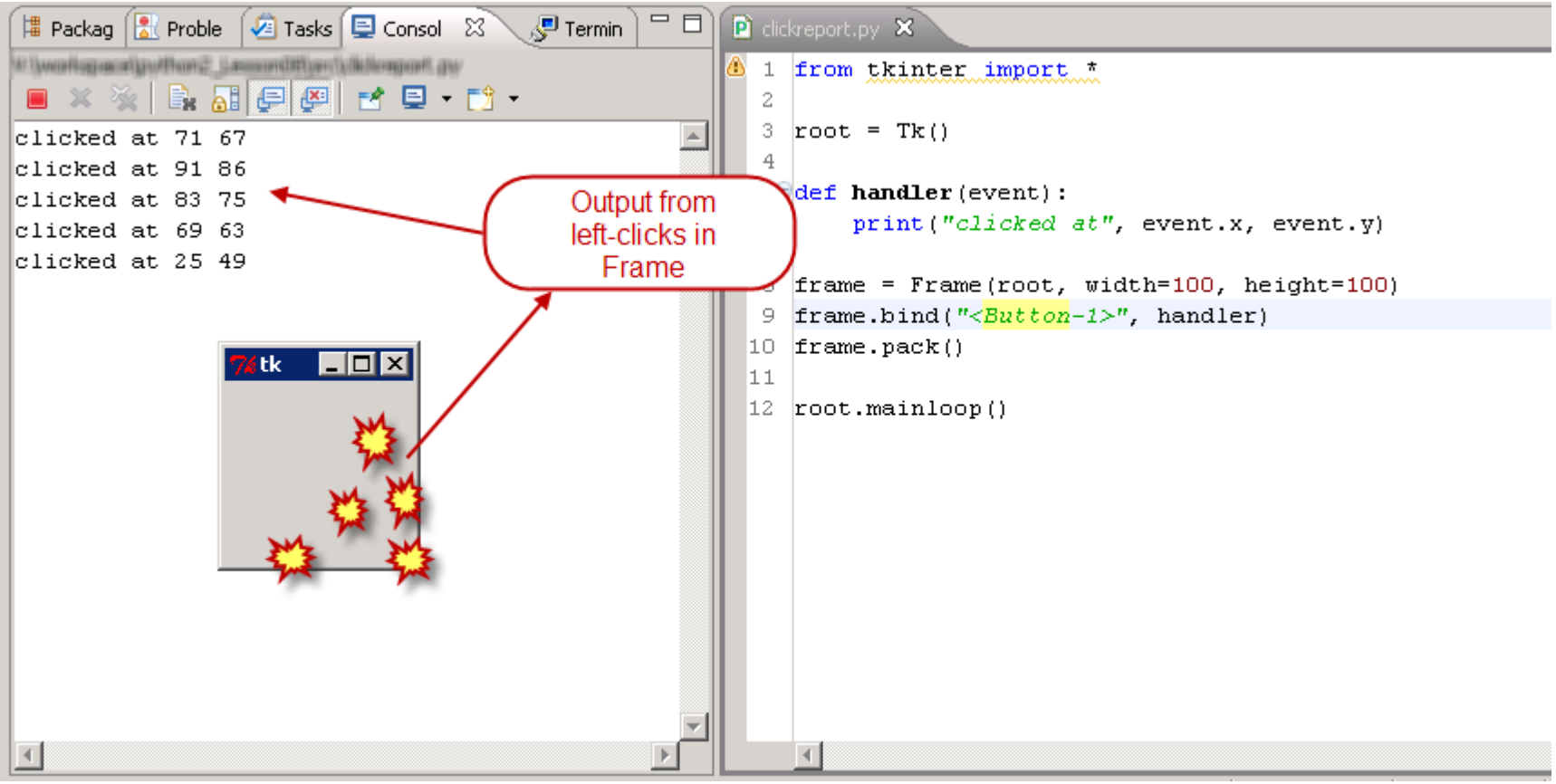


Canvas for Drawing (Content)



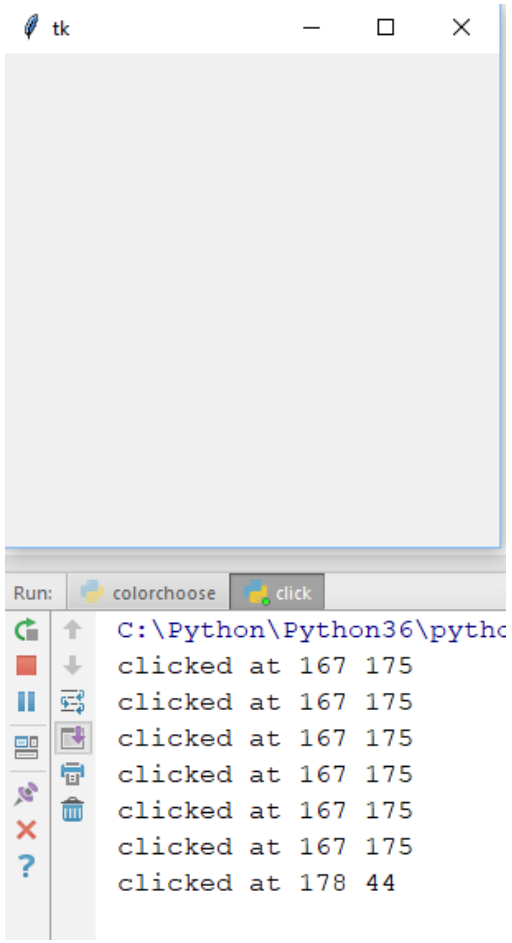
Tkinter GUI Technology (III)

Event (Source, Target and Handler) - Behavioral



Demo Program

click.py



S Structural Definition

main

↓

frame

C Only one frame

```
1   from tkinter import *
2
3   main = Tk()
4
5   def handler(event):
6       print("clicked at",event.x, event.y)
7
8   frame = Frame(main, width=300, height=300)
9   main.bind("<Button-1>", handler)
10  frame.pack()
11  main.mainloop()
```

Create a Tk object (main window application)

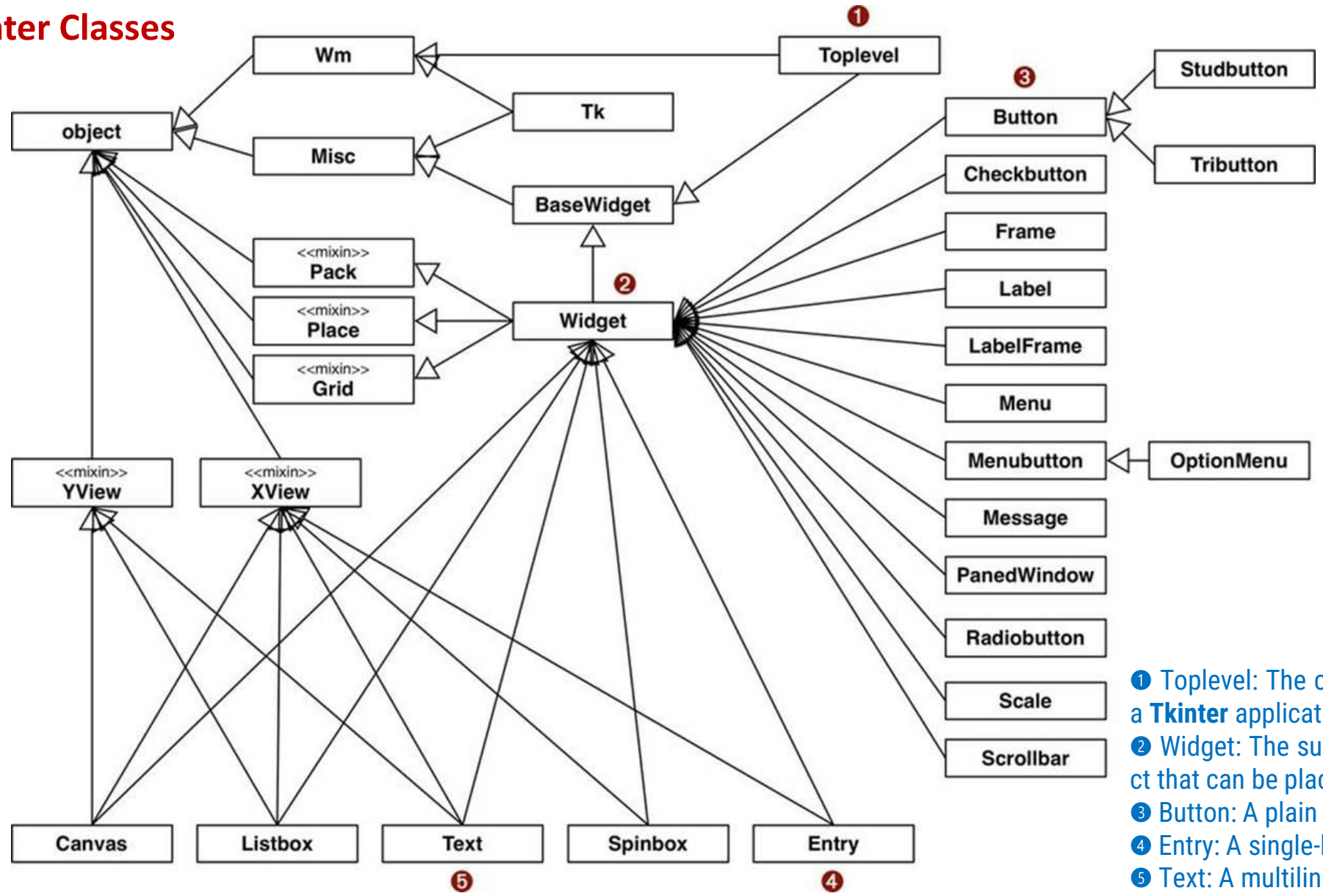
Create a frame in main

Pack the frame

Bind all left mouse click on main with handler

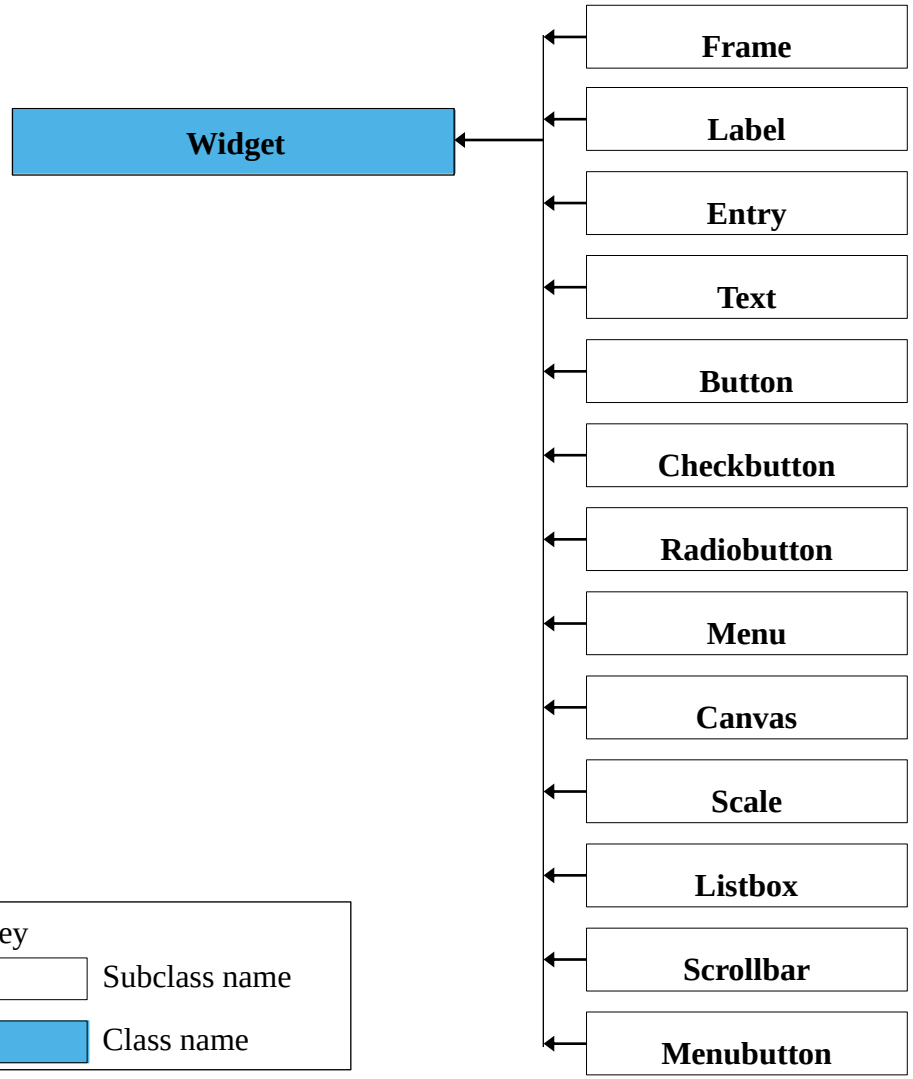
main window go into loop

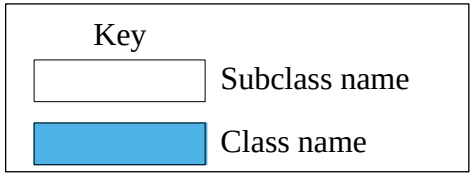
Tkinter Classes



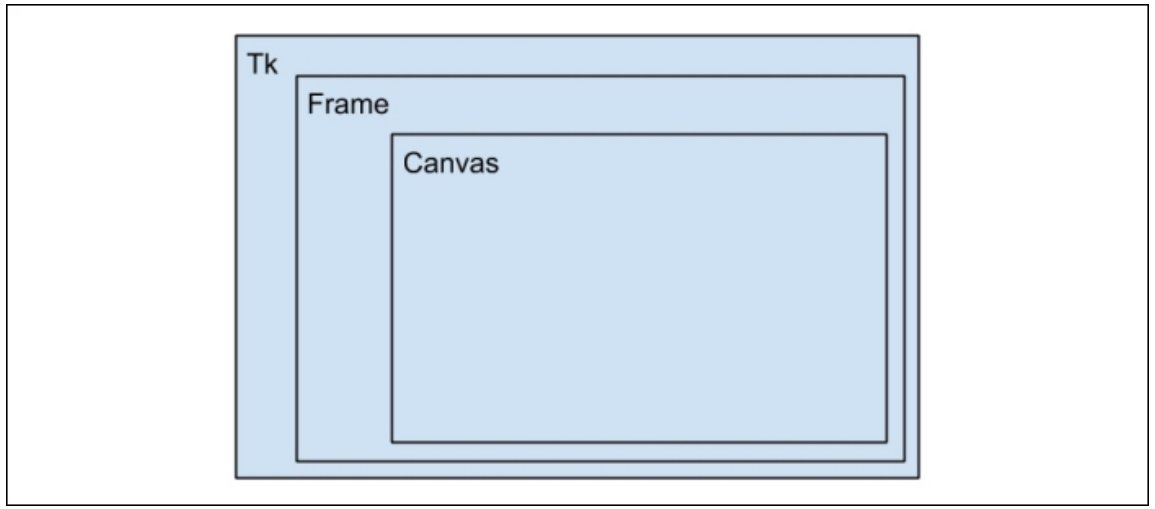
❶ Toplevel: The class of a top-level window in a **Tkinter** application.
❷ Widget: The superclass of every visible object that can be placed on a window.
❸ Button: A plain button widget.
❹ Entry: A single-line editable text field.
❺ Text: A multiline editable text field.

Tkinter Components and Canvas (Contents)





Tkinter Basic Layout for Drawing



tkinter widgets

tkinter has many controls/widgets available to build a GUI with.

Window Widgets: Toplevel

Container Widgets: PanedWindow, Frame, Canvas

Text Widgets: Label, LabelFrame, Text, Message, Entry

Box Widgets: Listbox, Spinbox

Button Widgets: Button, Checkbutton, Radiobutton

Menu Widgets: Menu, Menubutton

Slider Widgets: Scale, Scrollbar

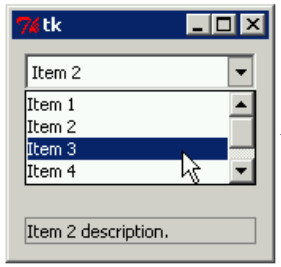
tkinter.ttk — More Widget

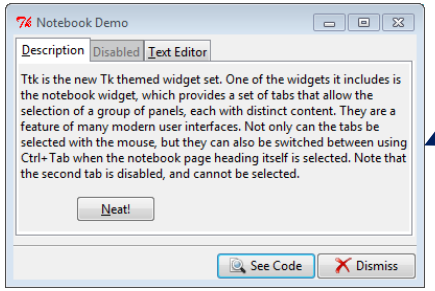
If you're going to use tkinter for your Course Project, check out some of these other widgets that are available outside of tkinter.

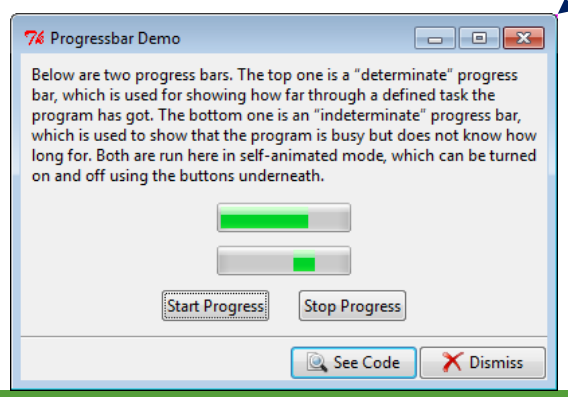
```
from tkinter import ttk
```

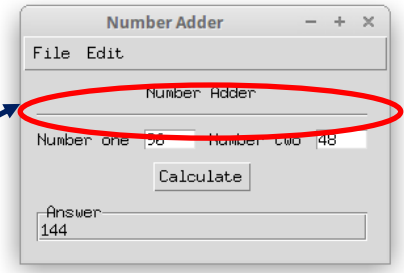
ttk widgets available:

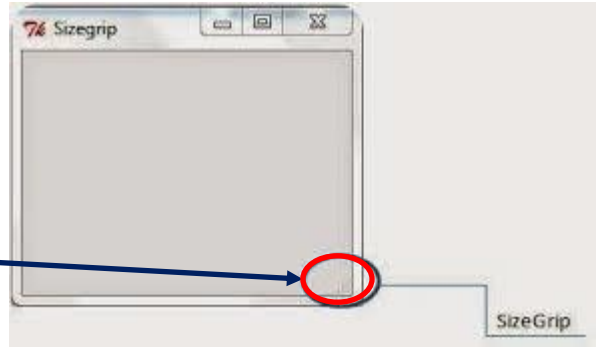
- Combobox
- Notebook
- Progressbar
- Separator
- Sizegrip
- Treeview











Note: tkinter.**ttk** replaces an older module called tkinter.tix.

Widget Methods

LECTURE 6

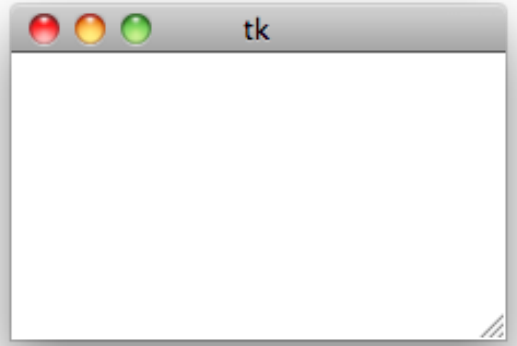
Widget `Tk`

mini.py

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

Basic Widget Methods

The following methods are provided by all widgets (including the root window).

Configuration

```
w.config(option=value)
value = w.cget("option")
k = w.keys()
```

Event processing

```
mainloop()
w.mainloop()
w.quit()
w.wait_variable(var)
w.wait_visibility(window)
w.wait_window(window)
w.update()
w.update_idletasks()
```

Event callbacks

```
w.bind(event, callback)
w.unbind(event)
w.bind_class(event, callback)
w.bindtags()
w.bindtags(tags)
```

Alarm handlers and other non-event callbacks

```
id = w.after(time, callback)
id = w.after_idle(callback)
w.after_cancel(id)
```

Window management

```
w.lift()
w.lower()
```

Window-related information

```
w.winfo_width(),w.winfo_height()
w.winfo_reqwidth(),w.winfo_reqheight()
w.winfo_id()
```

The option database

```
w.option_add(pattern, value)
w.option_get(name, class)
```

The Tkinter Frame Widget (Container)

Demo Program: HelloWorld/frame.py

A frame is rectangular region on the screen. The frame widget is mainly used as a geometry master for other widgets, or to provide padding between other widgets.

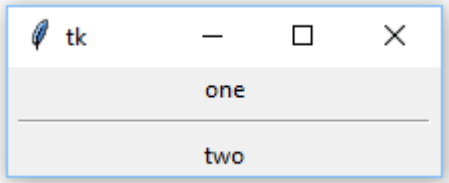
When to use the Frame Widget

Frame widgets are used to group other widgets into complex layouts. They are also used for padding, and as a base class when implementing compound widgets.

Patterns

The frame widget can be used for decorations:

```
from Tkinter import *
master = Tk()
Label(text="one").pack()
separator = Frame(height=2, bd=1, relief=SUNKEN)
separator.pack(fill=X, padx=5, pady=5)
Label(text="two").pack()
mainloop()
```



The Tkinter Toplevel Widget (Container)

The **Toplevel** widget work pretty much like **Frame**, but it is displayed in a separate, top-level window. Such windows usually have title bars, borders, and other “window decorations”.

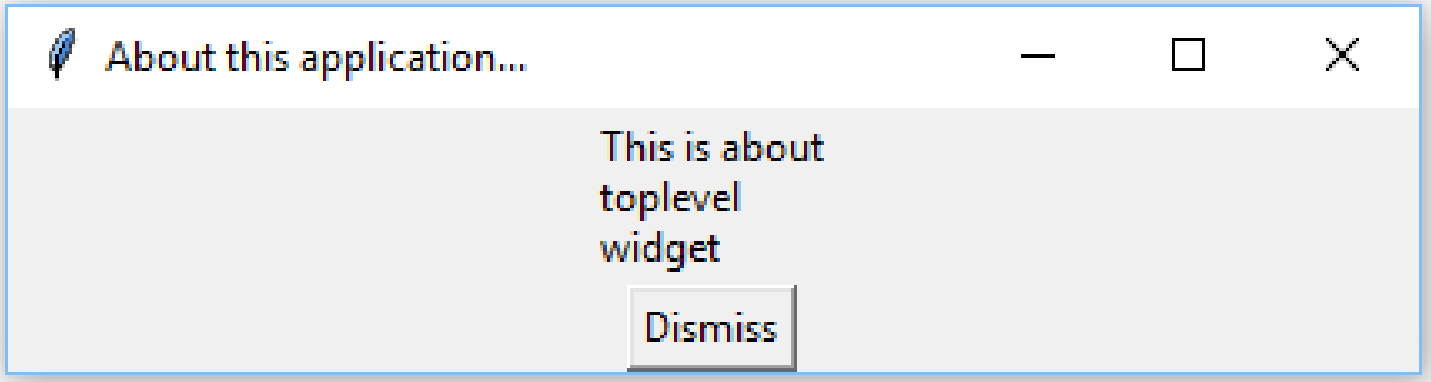
When to use the Toplevel Widget

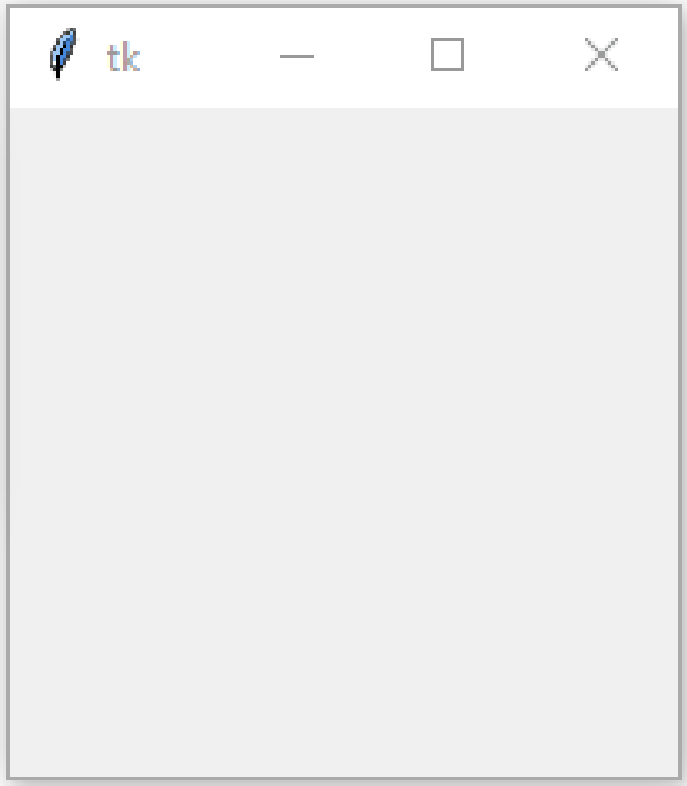
The Toplevel widget is used to display extra application windows, dialogs, and other “pop-up” windows.

Patterns

```
top = Toplevel()
top.title("About this application...")
msg = Message(top, text=about_message)
msg.pack()
button = Button(top, text="Dismiss", command=top.destroy)
button.pack()
```

toplevel widget





Structured GUI Program

LECTURE 5

Structured blank Window Program

Demo Program: blank.py

```
1   from tkinter import *
2
3   class Application(Frame):
4       def __init__(self):
5           super().__init__()
6           self.master.title("Title")      # set title for the window
7           self.pack(fill=BOTH, expand=1)
8
9   def main():
10      win = Tk()                          # create a window application
11      win.geometry("300x200+100+100")     # setting window geometry
12      app = Application()                 # start the frame inside the window
13      win.mainloop()                      # window go into event loop
14
15  if __name__ == "__main__":
16      main()
```

Frame Definition

Main program for the window

Structured Program with Toplevel Object

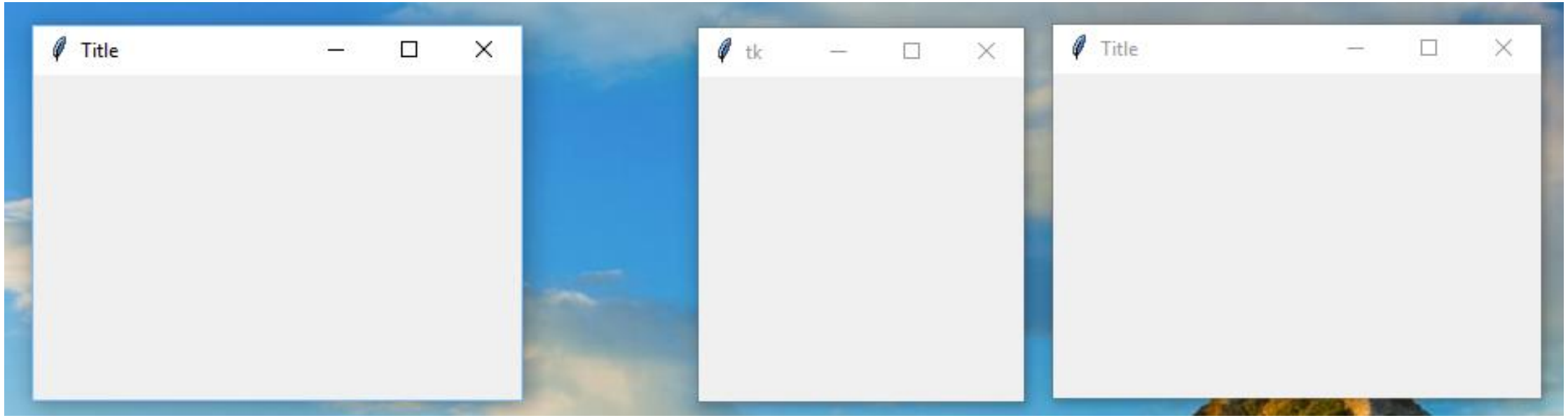
Demo Program: blank2.py

Go PyCharm!!!

```
from tkinter import *

class Application(Toplevel):
    def __init__(self):
        super().__init__()
        self.master.title("Title")      # set title for the window
        #self.pack(fill=BOTH, expand=1)# Toplevel has no pack

def main():
    win = Tk()                          # create a window application
    win.geometry("300x200+100+100")     # setting window geometry
    app = Application()                 # start the frame inside the window
    win.mainloop()                      # window go into event loop

if __name__ == "__main__":
    main()
```

Result for blank.py using Frame.
Tile is updated in the same window.

Result for blank2.py using Toplevel.
Tile is updated in the different window.

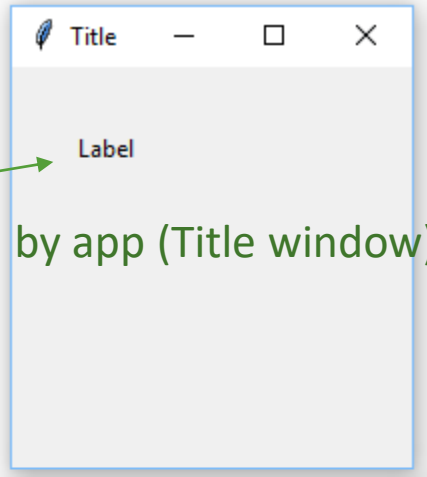


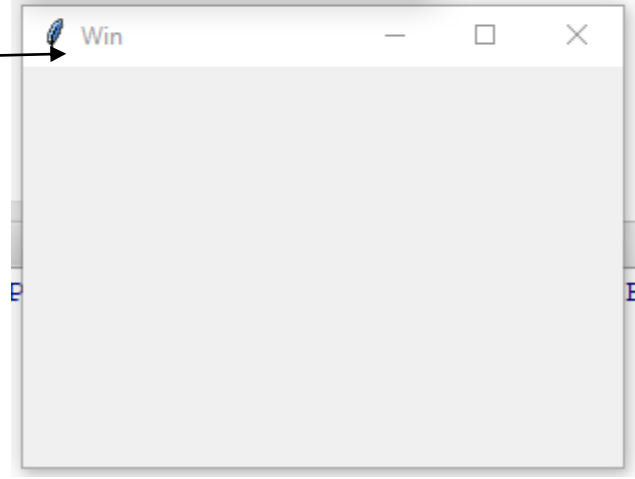
Revised blank2.py

```
class Application(Toplevel):
    def __init__(self):
        super().__init__()
        self.title("Title")      # set title for the window
        self.master.title("Win")
        self.lb = Label(self, text="Label")
        self.lb.place(x=30, y=30)
        #self.pack(fill=BOTH, expand=1)# Toplevel has no pack

def main():
    win = Tk()                          # create a window application
    win.geometry("300x200+100+100")     # setting window geometry
    app = Application()                 # start the frame inside the window
    win.mainloop()                      # window go into event loop

if __name__ == "__main__":
    main()
```

Owner of app (Toplevel). The owner is win main window

lb is owned by app (Title window)





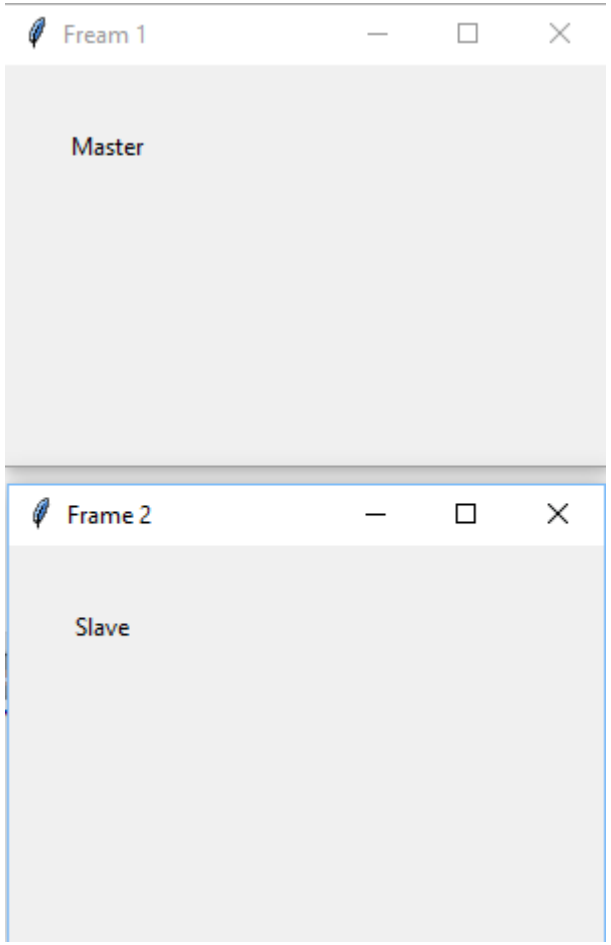
Note:
toplevel is a new slave window. If the win window is closed, the **toplevel** window will also be closed.

Three kinds of program relationship in a GUI Program

- **Inheritance of classes:**
  - Class Toplevel and Class Application have the Base class and the Derived class relationship.
- **Calling structure of functions:**
  - Main() function and the Application() constructor function has the caller and callee relationship.
- **Ownership of components:**
  - In this blank2.py example, the two windows has the owner(win) and slave(Title) relationship

Demo Program: blank3.py

Master and Slave Frames




```python
from tkinter import *
class Frame1(Frame):                          # master frame
    def __init__(self, owner):
        super().__init__(owner)
        self.initUI()
    def initUI(self):
        self.master.title("Fream 1")
        self.pack(fill=BOTH, expand=1)
        self.lb = Label(self, text="Master")
        self.lb.place(x=30, y=30)

class Frame2(Toplevel):                       # slave frame
    def __init__(self, owner):
        super().__init__(owner)
        self.initUI()
    def initUI(self):
        self.title("Frame 2")
        self.lb = Label(self, text="Slave")
        self.lb.place(x=30, y=30)

def main():
    win = Tk()                            # create a window application
    win.geometry("300x200+100+100")       # setting window geometry
    app = Frame1(win)                        # start the frame inside the window
    app2 = Frame2(win)                         # start the frame inside the window
    win.mainloop()                        # window go into event loop

if __name__ == "__main__":
    main()
```

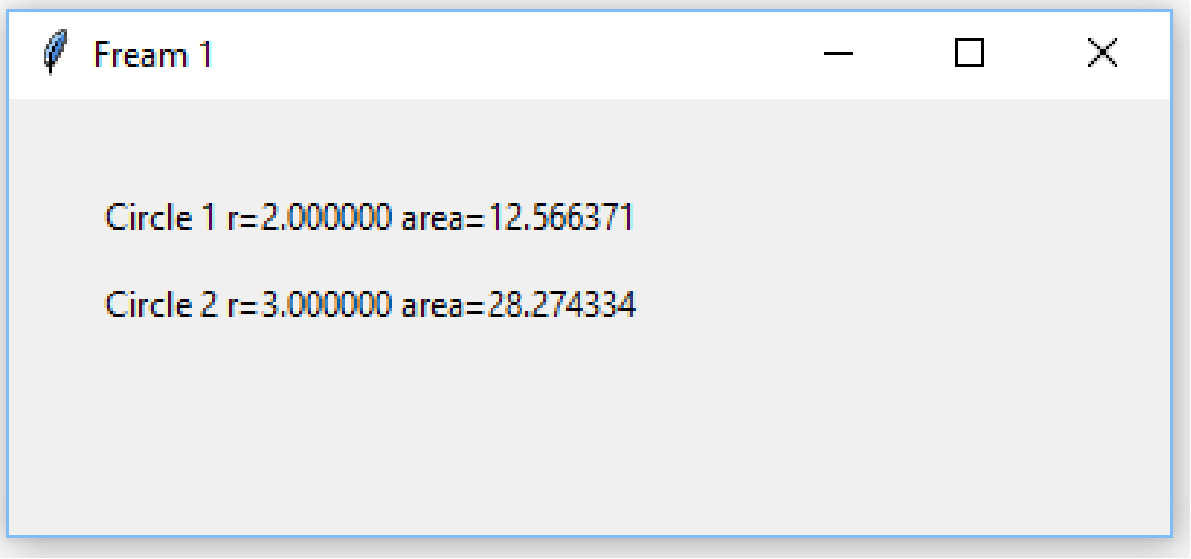
Merging Data and GUI

- **__init__():**
  - Constructor function used to build a frame or window
- **initUI():**
  - Function used to build the data view.
- **initModel():**
  - Function used to build the data model
- **showData():**
  - Function used to show data from Model to View

Demo Program:

blank4.py

Go PyCharm!!!



```
from math import *
from tkinter import *

class circle:
    NoOfCircle =0
    def __init__(self, r=1.0):
        self.__r = r
    def getR(self):
        return self.__r
    def getArea(self):
        return pi * self.__r**2
```

```python
class Application(Frame):                          # master frame
    def __init__(self, owner):
        super().__init__(owner)
        self.initModel()
        self.initUI()
        self.showData()

    def initModel(self):
        self.c1 = circle(2.0)
        self.a = self.c1.getArea()      # need to be frame instance variable to be shown on GUI
        self.c2 = circle(3.0)
        self.b = self.c2.getArea()

    def initUI(self):
        self.master.title("Fream 1")
        self.pack(fill=BOTH, expand=1)

    def showData(self):
        self.lba = Label(self, text=("Circle 1 r=%f area=%f" % (self.c1.getR(), self.a)))
        self.lba.place(x=30, y=30)
        self.lbb = Label(self, text=("Circle 2 r=%f area=%f" % (self.c2.getR(), self.b)))
        self.lbb.place(x=30, y=60)
```

```python
def main():
    win = Tk()                          # create a window application
    win.geometry("400x150+100+100")     # setting window geometry
    app = Application(win)                      # start the frame inside the window
    win.mainloop()                      # window go into event loop

if __name__ == "__main__":
    main()
```