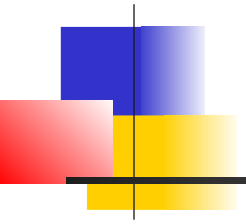
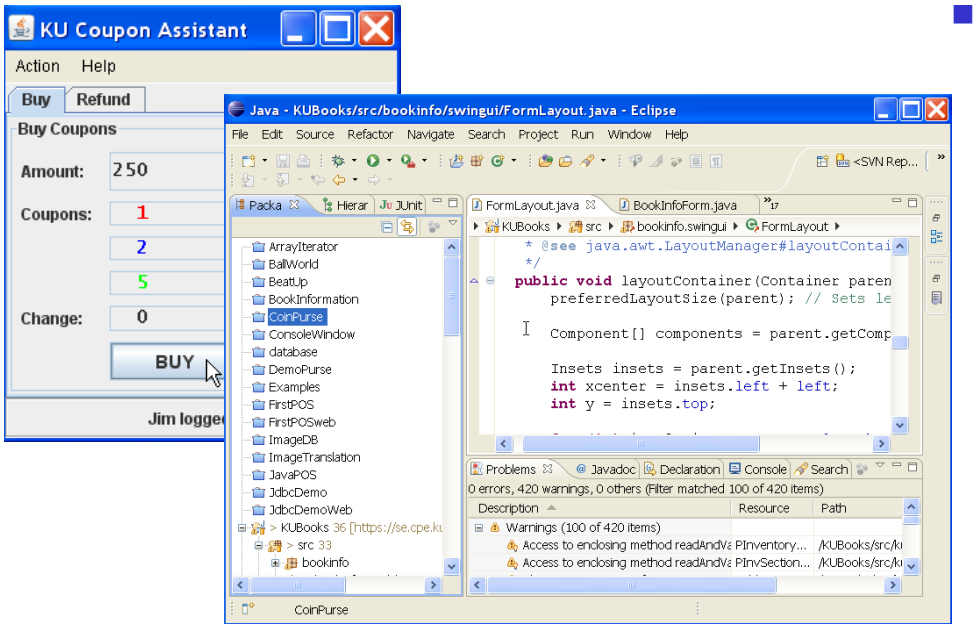


Introduction to Graphical Interfaces

Different Kinds of Apps

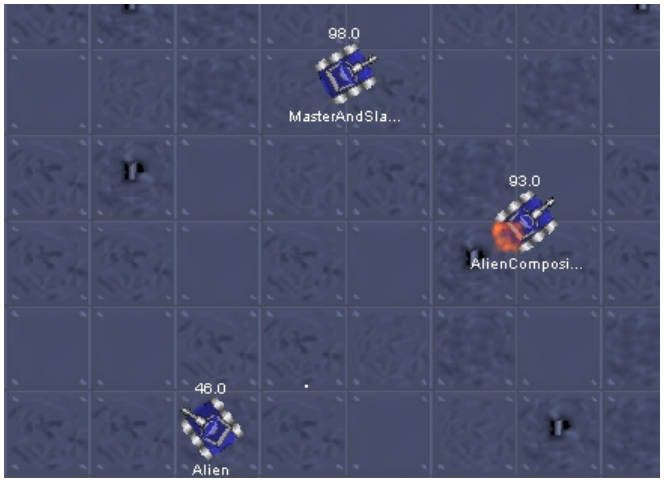
Graphical User Interface

- Display info and accept user input
- Built using components
- "Forms" applications
- IDE, File Manager, etc.



Graphics Programming

- Manipulate graphical elements on a display
- points, lines, curves, shapes
- images, textures,
- animation, 3D effect



Frameworks

Python has frameworks for building graphical apps.

Tkinter - included with Python distribution, easy to use.

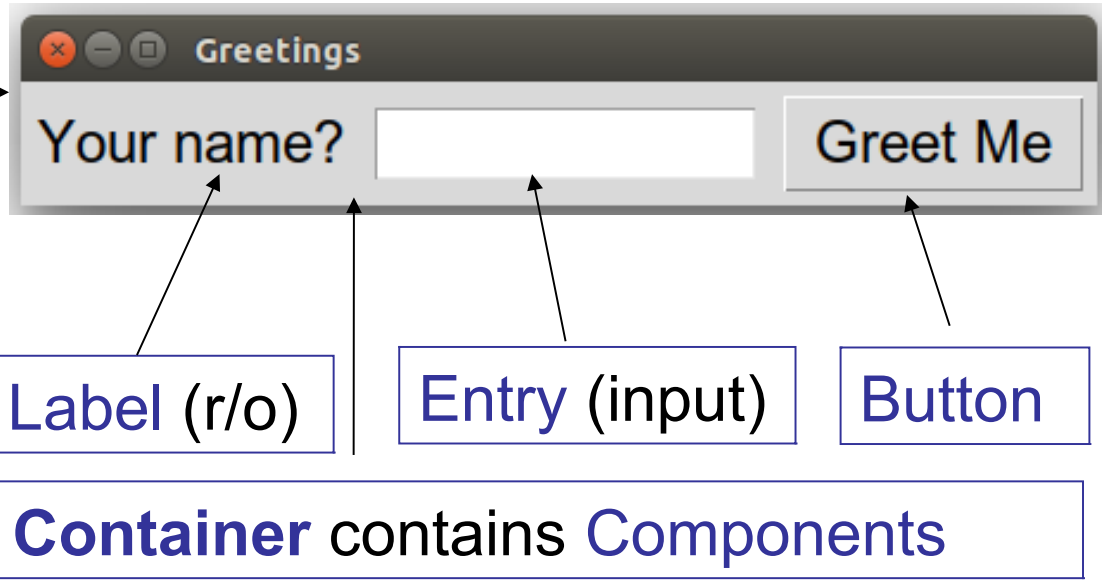
PyQt5 - built on QT cross-platform framework. Powerful, requires Qt be installed on host.

wxPython - wrapper for wsWidgets API for creating cross-platform GUI interfaces.

Kivy - based on OpenGL, suitable for GUI and games.

Parts of a GUI

Window shown on screen



Components ("widgets")

Layout Manager

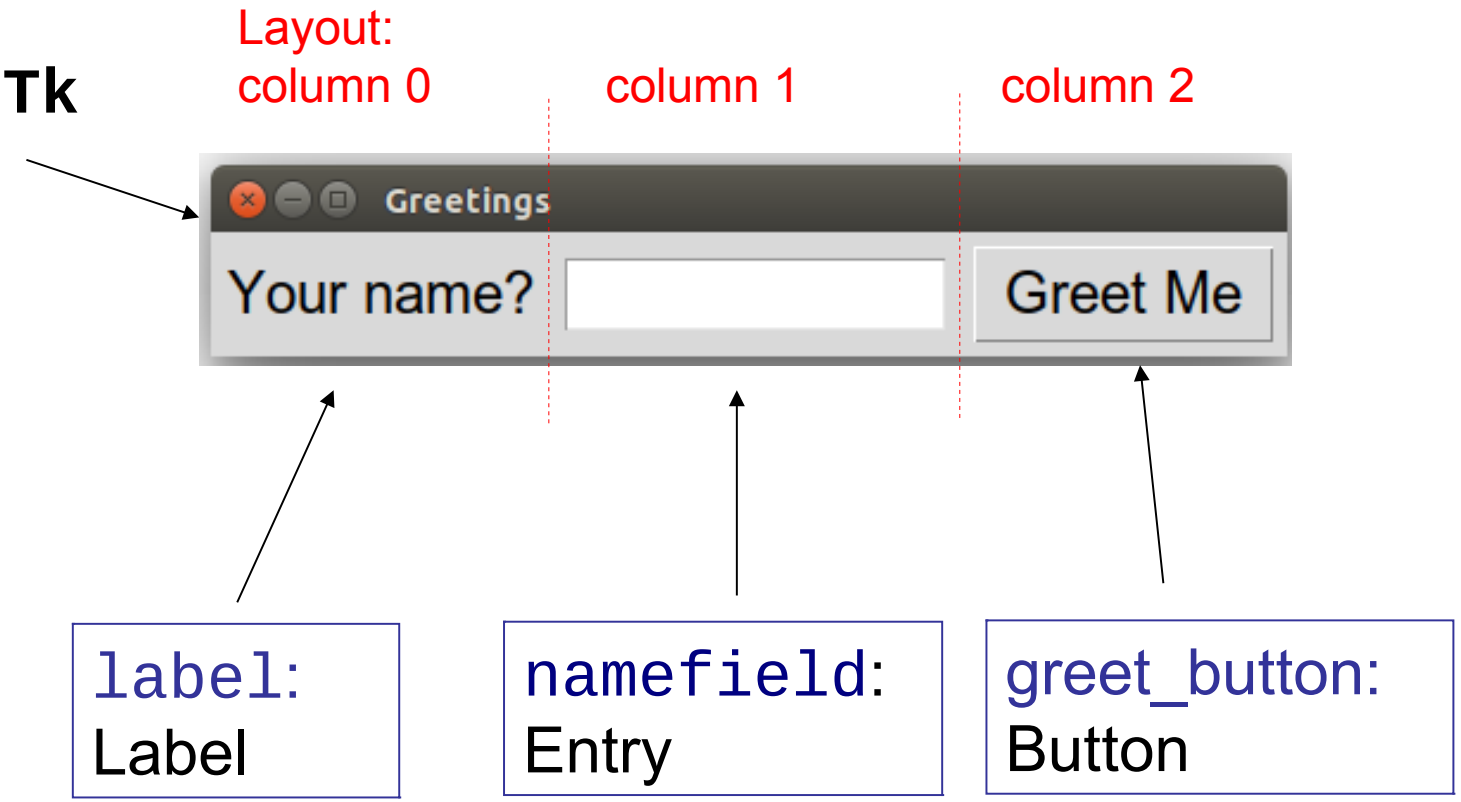
Arrange & align components, resize things when window size changes.

Event Handlers (code)

```
def button_handler():
    username = entry.get()
    print(f"Hello, {username}")
    entry.delete(0, len(username))
```

Define our GUI

`tkinter.Tk`
(window)



Create a Simple GUI

1. create a window (includes a container)

```
import tkinter as tk
root = tk.Tk()
root.title("Greeter")
```

2. create components & add them to the window

```
label = tk.Label(root, text="Your name?")
namefield = tk.Entry(root, width=20)
greet_button = tk.Button(root, text="Greet Me")
```

3. position components using grid layout grid(row,col)

```
label.grid(row=0, column=0)
namefield.grid(row=0, column=1)
greet_button.grid(row=0, column=2)
```

Run it

4. instruct window (root) to wait for events

```
root.mainloop()
```

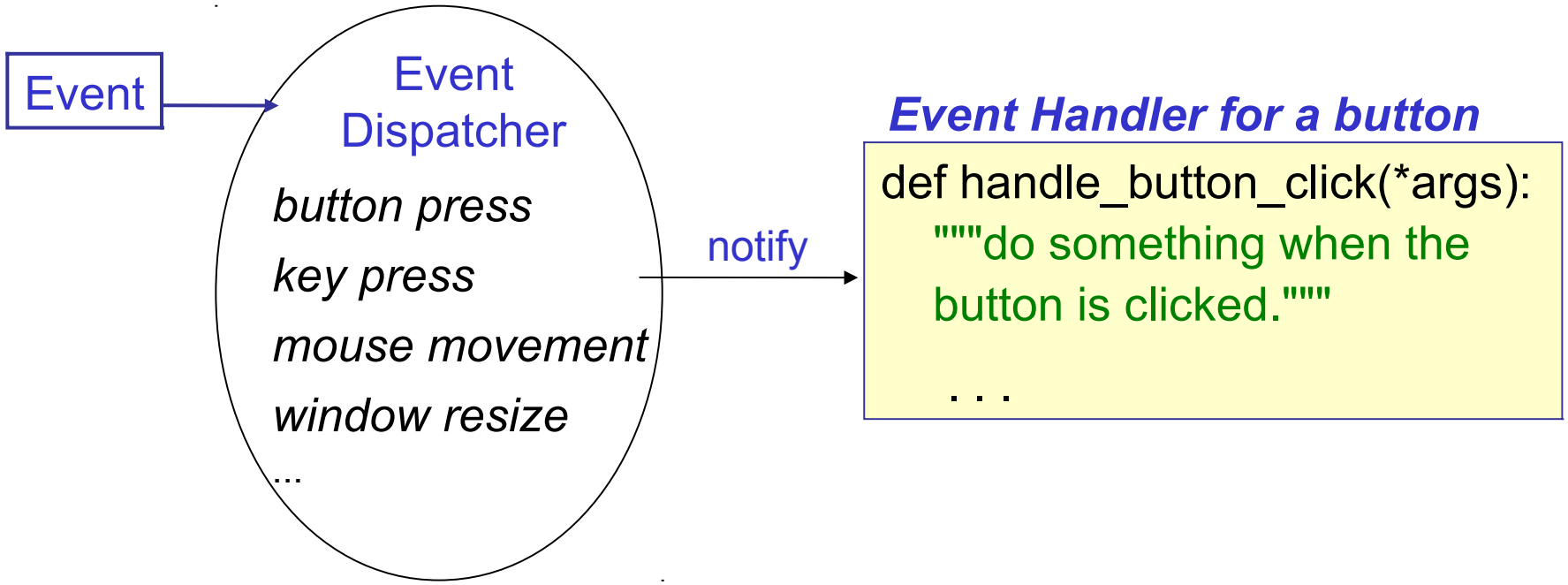
Save file & Run it:

```
> python3 greeter.py
```

Resize the window.
What happens to the component sizes?

Event Driven Programming

- Graphics apps are ***event-driven***.
- An **event dispatcher** receives events and ***notifies*** *event handlers* (code).
- Events come from user actions, operating system, timers, or other code.



Kinds of Events

- mouse button click on a component
- key press
- mouse movement
- text changed in a text component
- a timer triggers an event
- operating system triggers an event

Add an Event Handler (code)

Event: *users presses button*.

5. define an event handler to greet the user

```
def greet_handler(*args):
    username = namefield.get()
    message = f"Hello, {username}"
    print(message)
    # clear text from the input field
    namefield.delete(0, tk.END)
```

Write this function before the greet_button code.

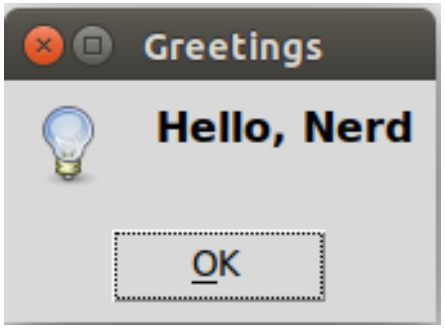
6. add the event handler to Button component

```
greet_button = tk.Button(root, text="Greet Me",
                         command=greet_handler)
```

Run it.

Improve the Behavior

Show a **dialog box** instead of printing on console




```
from tkinter import messagebox

def greet_handler(*args):
    username = namefield.get()
    message = f"Hello, {username}"
    messagebox.infomessage("Greetings", message)
    # clear text from the input field
    namefield.delete(0, tk.END)
```

Run it.

Event Handler for other Events

If user enters a name and presses ENTER, then greet him (no button press).

Bind the `greet_handler` code to keyboard "<Return>" event on the Entry field.

```
namefield.bind('<Return>', greet_handler)
```

List of common event names:

https://python-course.eu/tkinter/events-and-binds-in-tkinter.php

(I really *hate* using strings for event names.)

Improve the Appearance

Components have a common set of **attributes** you can set.

1. Set `namefield` text color & font using dict syntax:

```
namefield['foreground'] = "blue"
namefield['font'] = ('Monospace', 16)
```

2. Customize other components using `.configure`

```
options = {'font': ('Arial', 14)}
label.configure(**options)
greet_button.configure(**options)
```

Grid Spacing

The Grid Layout ("Geometry") has properties you can set.

Add some space between components

```
label.grid(row=0, column=0, padx=5, pady=5)
namefield.grid(row=0, column=1, padx=5, pady=5)
greet_button.grid(row=0, column=2, padx=5, pady=5)
```

Make the `namefield` expand to fill available space:

```
root.columnconfigure(1, weight=1)
namefield.grid(sticky=tk.EW) # EW = east-west sides
```

Run It and resize the window. What happens?

Variables for Component Data

Many tkinter components require a **variable** to set/get component data.

Examples: `Radiobutton, Listbox`

Define a variable for the `namefield` text value

```
username = tk.StringVar()
namefield = tk.Entry(root, width=20,
                     textvariable=username)
```

Use the Variable in Event Handler

`tk.StringVar` has '`get`' and '`set`' methods.

```
def greet_handler(*args):
    message = f"Hello, {username.get()}"
    messagebox.infomessage("Greetings", message)
    # clear text from input field
    username.set("")

#TODO if no name is entered, just ignore the event
```

Less Coupling is Good!

The `greet_handler` code does not depend on the Entry component. It depends only on the `username` StringVar.

Summary: What do you need to know?

- Window - how to create a window
- Components - what components are available
  - how to set attributes of the components?
- Layouts - how to arrange components
- Containers - how to group components in containers to simplify layout
- Events - what events are there?
  - How to I connect an event to my code?
- Dialogs - special purpose dialogs for message and input (e.g. file chooser)
- Event looping - how to create event for later callback?

Know your components

You need to know the available components

... and what they can do.

https://anzeljg.github.io/rin2/book2/2405/docs/tkinter/index.html - *John Shipman's popular Tkinter guide*

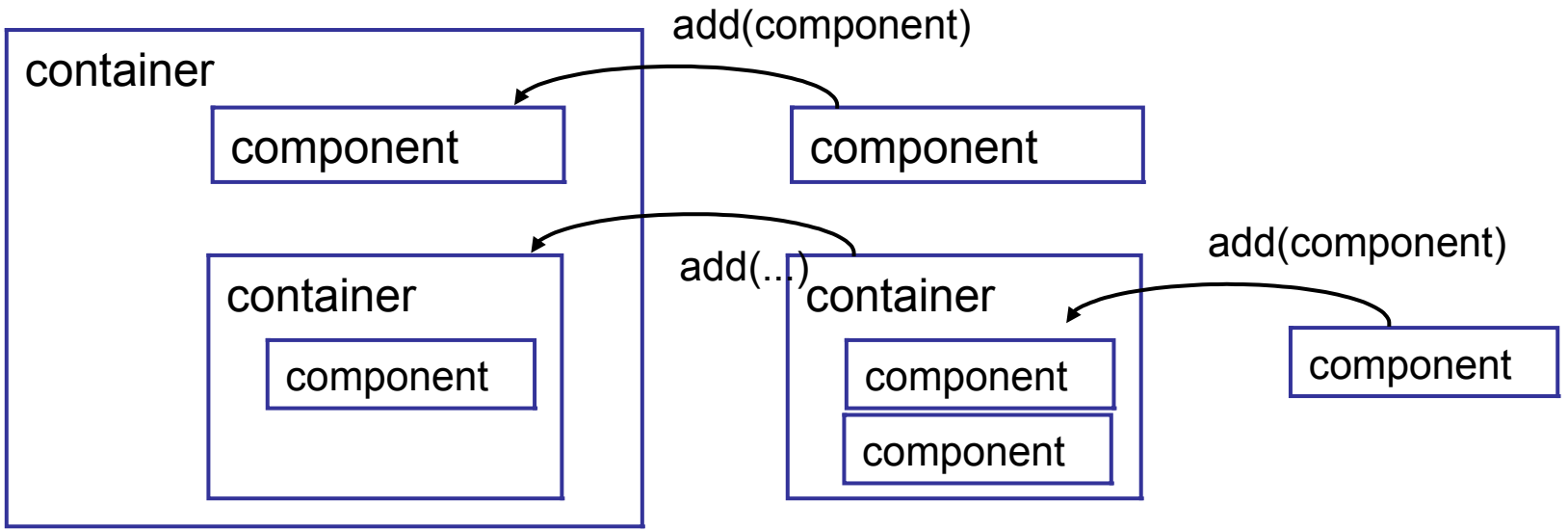
https://python-course.eu/tkinter

https://runestone.academy/ns/books/published/thinkcspy/GUIandEventDrivenProgramming/03_widgets.html

Aj. Chaiporn's slides from Programming 2 in 2021.

Containers and Components

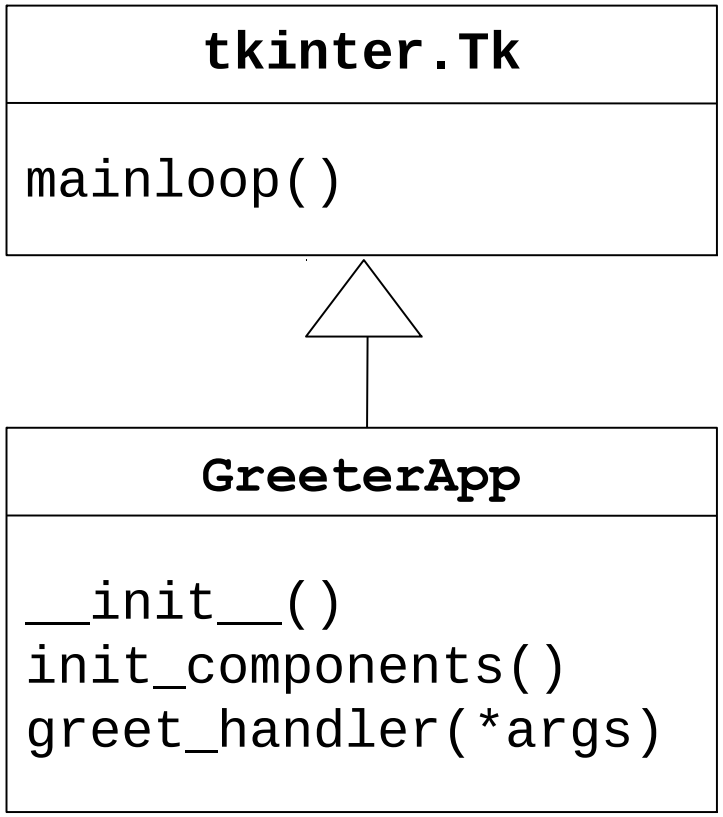
- A GUI has many **components** in **containers.**
- Use **add** to put component in a container.
- A container is also a component; so a container may *contain other containers*.



Object-Oriented Tk Application

Your class can extend tkinter.Tk or a container such as tkinter.ttk.Frame

Your class's __init__ should call superclass __init__ first.



GreeterApp Class

```
class GreeterApp(tk.Tk):

    def __init__(self):
        # call the superclass constructor FIRST
        super().__init__()
        self.title("Greeter")
        # Create components in a separate method
        self.init_components()
```

Use a separate method (`init_components`) to create & layout components.
In this course, points deducted if lots of tkinter code in __init__.

Add components & event handler

The components are assigned to local variables. But `self.username` *is an attribute.*

Why?

```
def init_components(self):
    """Define components and layout"""
    self.username = tk.StringVar()
    label = tk.Label(self, text="Your name?")
    namefield = tk.Entry(self, width=12,
                         textvariable=self.username)
    greet_button = tk.Button(self, text="Greet Me",
                         command=self.greet_handler)
    namefield.bind('<Return>', self.greet_handler)
    # position the components
    label.grid(row=0, column=0, padx=5, pady=5)
    ... (same as before)
    # style the components
    options = {'font': ('Arial',16)}
    ... (same as before)
```

Define the event handler

```
def greet_handler(self, *args):
    """Greet the user."""
    who = self.username.get().strip()
    # clear input field for next use
    self.username.set("")
    message = f"Hello, {who}"
    messagebox.showinfo("Greetings", message)
```

The event handler needs access to **`username`** (StringVar) so `username` must be an attribute.

No event handler needs access to the components, so we don't need a reference to them (no attributes).