UNIVERSITY *of* TASMANIA

KIT100 PROGRAMMING PREPARATION

Lecture Ten:

GUI Components part 1

Lecture Objectives

- Introduction to Graphical User Interfaces (GUI)
- Tkinter (TK Interface)
- Processing Events
- The Widget Classes
- Designing your GUI

What’s a GUI?

- It is a user interface that includes graphical elements, such as windows, icons and buttons.
- Many types of digital devices lead to many different designs of Graphical User Interfaces (GUI).
- Computer, mobiles, cars, electronic equipment can all have GUIs. This now extends to a huge variety of portable devices like watches, fitness trackers, bathroom scales etc.

History of GUIs

- Interfaces devices used prior - command line only, punched paper tape input and output, banks of switches and lights.
- Earliest GUIs
  - WIMP (Window, Icon, Menu, Pointer) paradigm.

w.i.m.p
window
icon
menu
pointer
Window
Finder File Edit View Go
About Finder
Preferences...
Empty Trash...
Secure Empty Trash...
Services
Hide Finder
Hide Others
Show All

- Development GUIs over 5 decades.
  - 1970: Xerox PARC – first computer to demonstrate the desktop metaphor and the GUI.
  - 1980: Apple Macs, and graphical interfaces become more in use. Microsoft Windows 2.0. First multimedia computer was released. 1984 Apple Mac
    - https://youtu.be/VtvjbmoDx-I 1984 Apple Advert
    - https://youtu.be/JQ8ZiT1sn88 Mother of all demos
  - 1990: Mainstream use of the desktop.
  - 2000: GUIs continued to be refined. Introduction of mobile interfaces. 2007 Apple iPhone
  - 2010 onwards: iPads, touch tablets. Virtual/Augmented Reality

Command Line vs. GUI

a **CLI** (command line interface) or **GUI** (graphical user interface)?

	CLI	**GUI**
Ease	✗ higher degree of memorization and familiarity	✓ users tend to learn how to use a GUI faster
Control	✓ good bit of control over both the file and operating systems	✓ offers access to files, software features, and the operating system as a whole
Multitasking	✗ do not offer to view multiple things at once on one screen.	✓ have windows that enable a user to view, control, manipulate, and toggle through multiple programs
Speed	✓ only need to utilize a keyboard to navigate the interface	✗ they require a mouse, slower than using the keyboard
Resources	✓ only using the command line takes a lot	✗ Require more system resources because of the elements that require loading

Command Line vs. GUI

	CLI	GUI
Scripting	✓ requires users to already know scripting commands and syntax	✗ with the help of programming software – provides guides and tips
Remote access	✗ you must know the commands to do so and is not as easy for new users.	✓ easy to navigate with little experience
Diversity	✓ After you've learned how to navigate and use a command line, it's not going to change as much as a new GUI.	✗ Each GUI has a different design and structure when it comes to performing different tasks.

Take away:

- Overall, a GUI is used by more users today than a CLI.
- Programmers may lean towards using a CLI for efficiency and speed.
- GUI is more user-friendly and preferred by most users.

How do GUIs work?

- A GUI uses windows, icons, and menus to carry out commands.
- GUI systems
  - much easier to learn – provides a better user experience.
  - Users don’t need to know any programming languages to perform tasks through a GUI.
- Different programming languages will use different libraries and methods to create GUI platforms for their programs.
  - Python source code includes (e.g, import library)

Python and GUIs

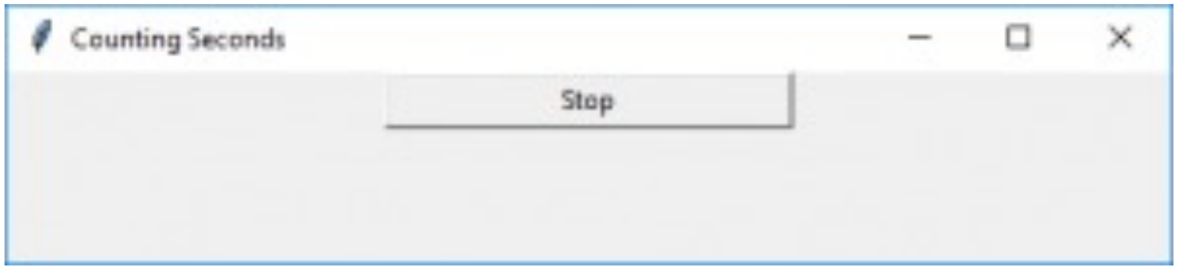
- Users respond to the GUI, because it's friendlier and requires less thought than the command-line prompt.

- There are many products available to provide a GUI for your Python program.
  - `tkinter` (*TK Interface*) is the standard GUI package – that is what we'll be using. It's a cross-platform (Windows, macOS, Linux), accepted standard.

Getting started with tkinter

- The `tkinter` **module** contains several classes for **creating GUIs**.
  - In other words, the module contains the (compiled) source code of several classes (that start with the keyword `Class`).

- The `Tk` class, which is part of the `tkinter` module,
  - creates a **window** for holding GUI **widgets** (a widget is a visual component, like buttons, labels etc)
  - tkinter.Tk ( )

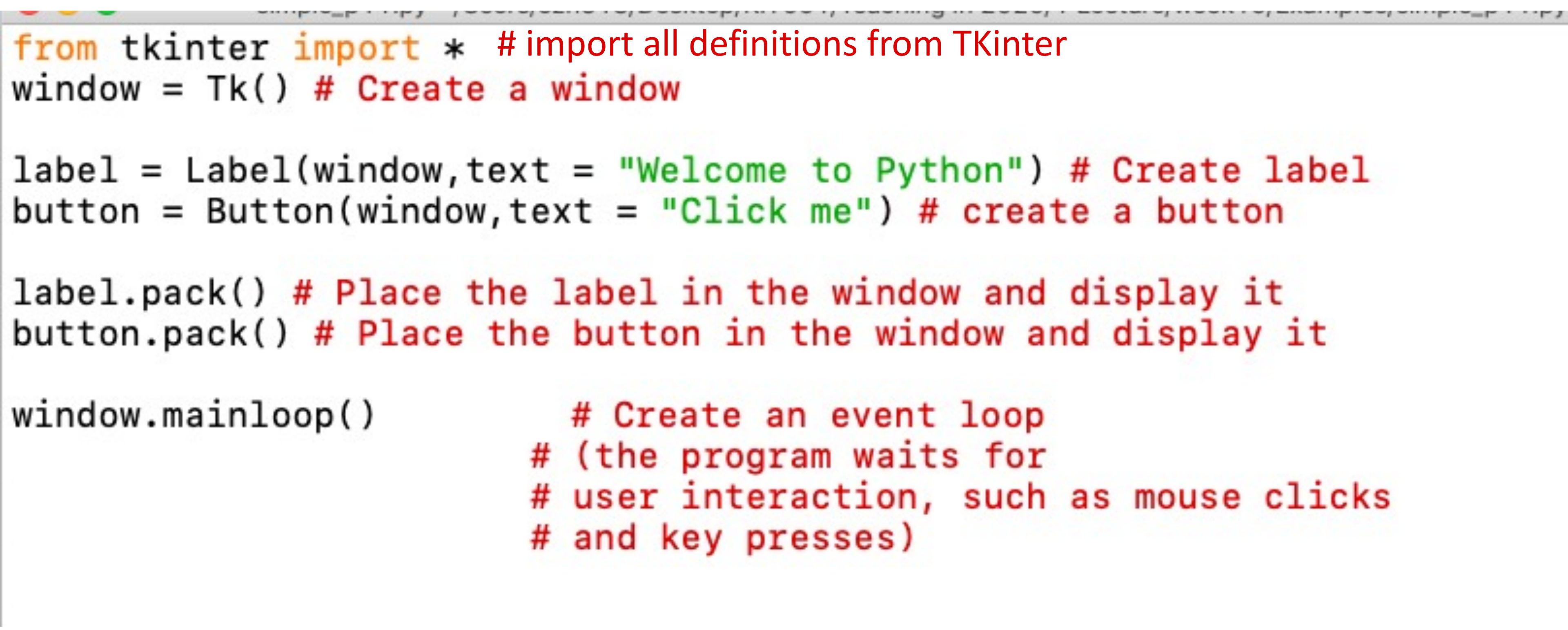


General form for all tk *widgets*

- Create the widget (general syntax):
  - `widgetVar = Widget(parent, attributes...)`
  - Widget is something like Label, Button, Frame etc,
    - `Label(parent, attributes...)`
    - `Button(parent, attributes...)`

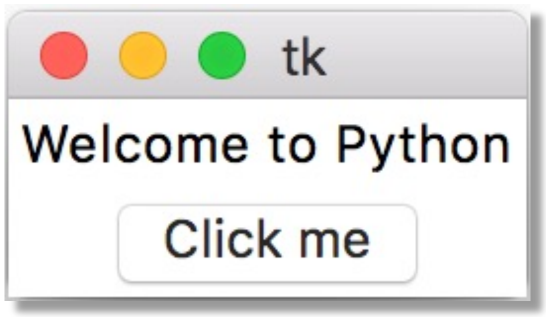
- Layout:
  - `widgetVar.pack()`
    - "Pack" (place) the widget on screen and make it visible.
    - doing layouts where everything is on a single row or in a single column (think rows of buttons in a toolbar)
  - `widgetVar.grid()`
    - place the widget on screen and make it visible in a grid.
    - arrange widgets along row and column boundaries (great for creating tables)

A simple GUI

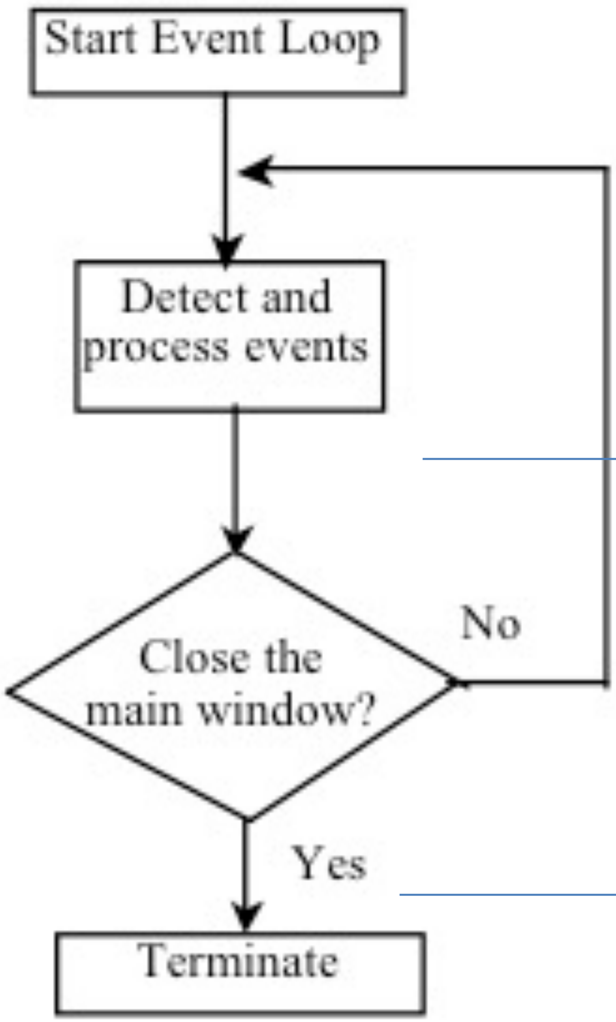


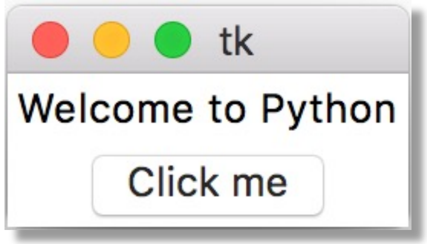
```
from tkinter import *  # import all definitions from TKinter
window = Tk() # Create a window

label = Label(window,text = "Welcome to Python") # Create label
button = Button(window,text = "Click me") # create a button

label.pack() # Place the label in the window and display it
button.pack() # Place the button in the window and display it

window.mainloop()           # Create an event loop
                          # (the program waits for
                          # user interaction, such as mouse clicks
                          # and key presses)
```

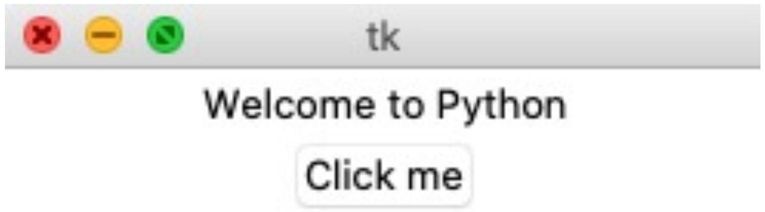
Output



Event-driven programming with tkinter





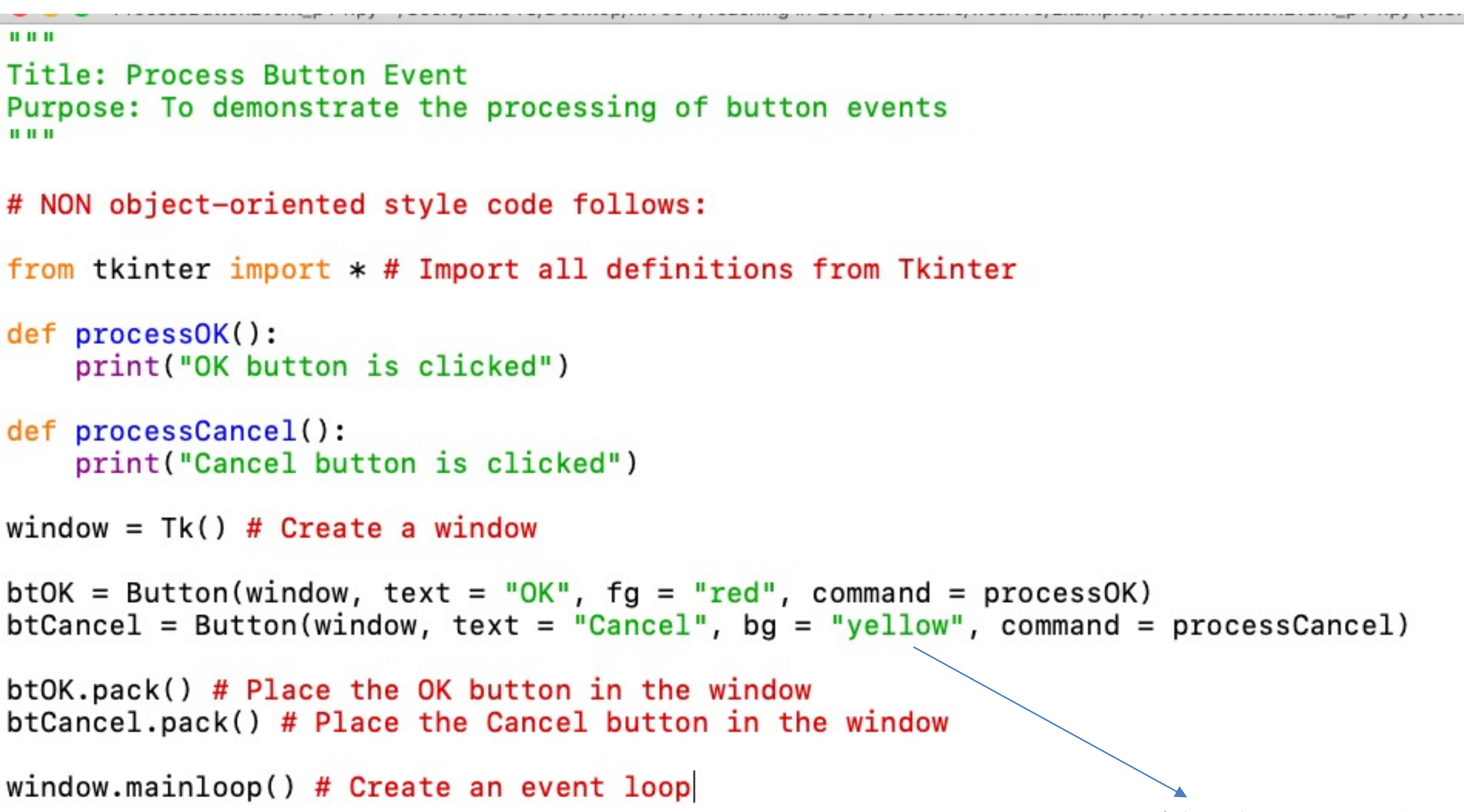


Processing Events

- A `tkinter` widget can be bound to a function (use def statement), which is called when an event occurs.
  - This is known as a *callback*.
  - Example:
    - When the user clicks a button, a *button-click* (specifically the `<Button>` event) occurs and the program should process this event by running the function code.
      - The next slide – the source code for callback.

Processing a Button Event

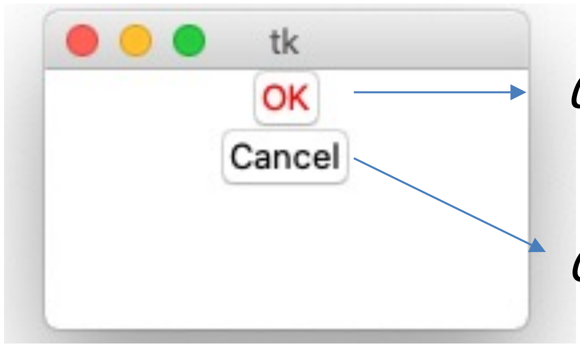
* Procedural programming style (NON object-oriented programming (OOP) style)



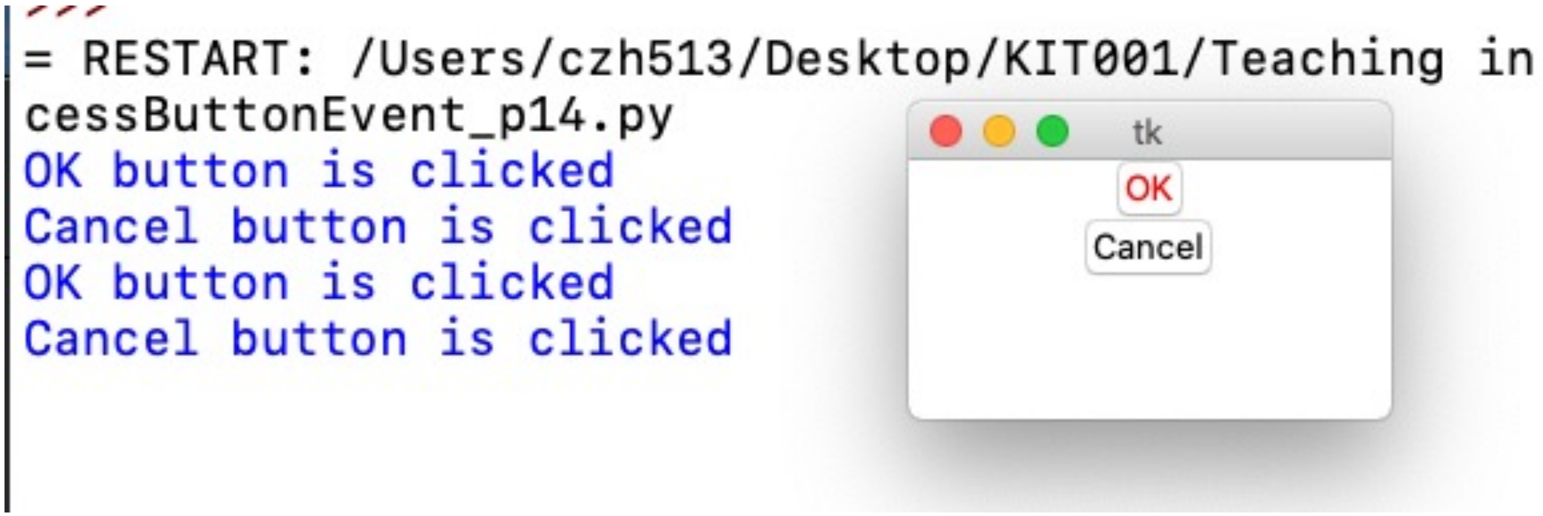
```
"""
Title: Process Button Event
Purpose: To demonstrate the processing of button events
"""

# NON object-oriented style code follows:

from tkinter import * # Import all definitions from Tkinter

def processOK():
    print("OK button is clicked")

def processCancel():
    print("Cancel button is clicked")

window = Tk() # Create a window

btOK = Button(window, text = "OK", fg = "red", command = processOK)
btCancel = Button(window, text = "Cancel", bg = "yellow", command = processCancel)

btOK.pack() # Place the OK button in the window
btCancel.pack() # Place the Cancel button in the window

window.mainloop() # Create an event loop
```

* fg -> foreground colour
* bg->background colour

Output



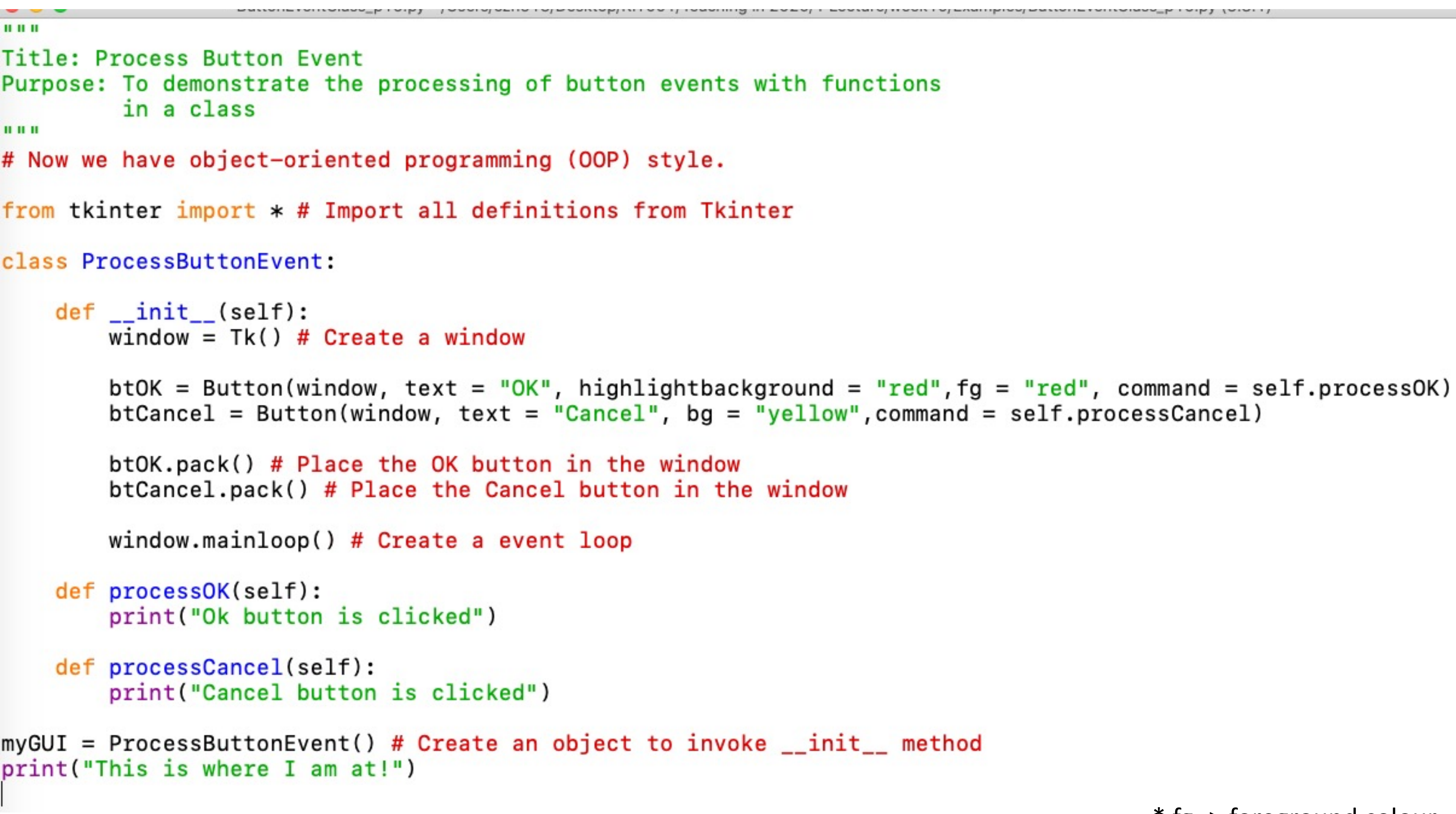
```
OK button is clicked

Cancel button is clicked
```



```
= RESTART: /Users/czh513/Desktop/KIT001/Teaching in
cessButtonEvent_p14.py
OK button is clicked
Cancel button is clicked
OK button is clicked
Cancel button is clicked
```

Classes – Process Events

* object-oriented programming (OOP) style

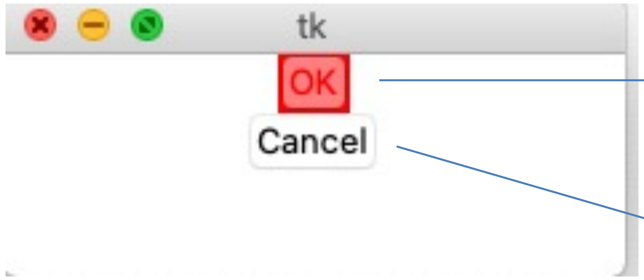


```
"""
Title: Process Button Event
Purpose: To demonstrate the processing of button events with functions
         in a class
"""
# Now we have object-oriented programming (OOP) style.

from tkinter import * # Import all definitions from Tkinter

class ProcessButtonEvent:

    def __init__(self):
        window = Tk() # Create a window

        btOK = Button(window, text = "OK", highlightbackground = "red",fg = "red", command = self.processOK)
        btCancel = Button(window, text = "Cancel", bg = "yellow",command = self.processCancel)

        btOK.pack() # Place the OK button in the window
        btCancel.pack() # Place the Cancel button in the window

        window.mainloop() # Create a event loop

    def processOK(self):
        print("Ok button is clicked")

    def processCancel(self):
        print("Cancel button is clicked")

myGUI = ProcessButtonEvent() # Create an object to invoke __init__ method
print("This is where I am at!")
```

* fg -> foreground colour
* bg->background colour

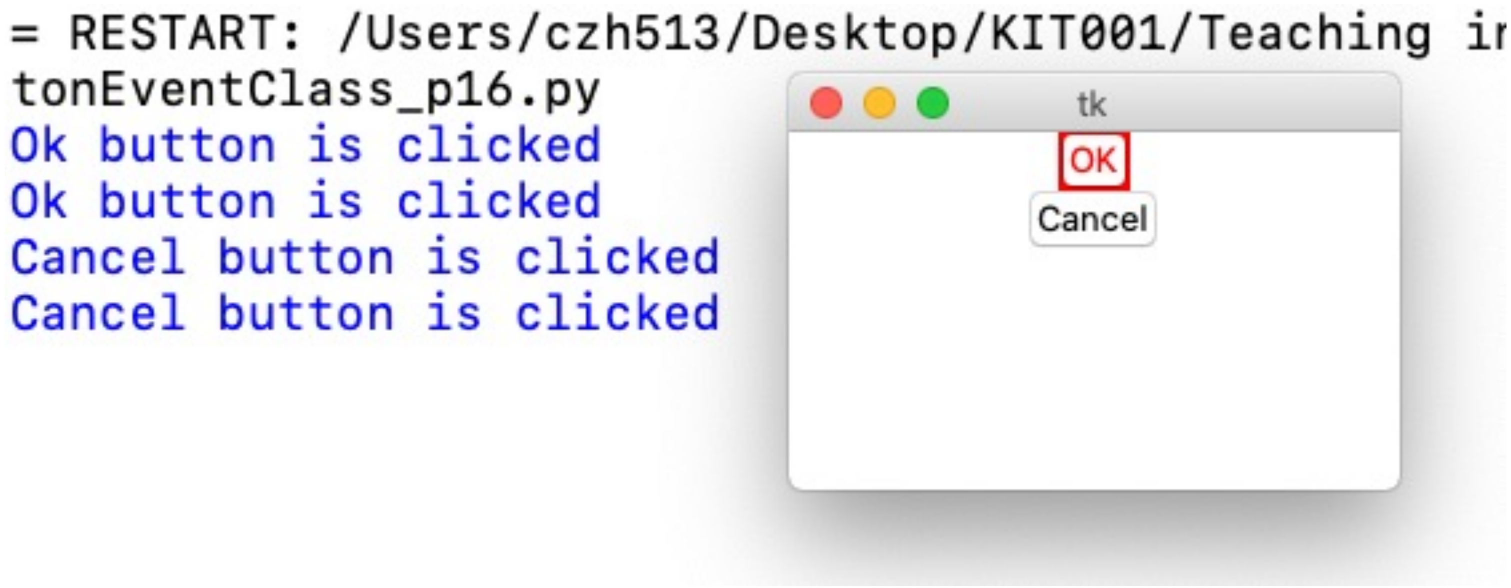
Output

This is where I am at!



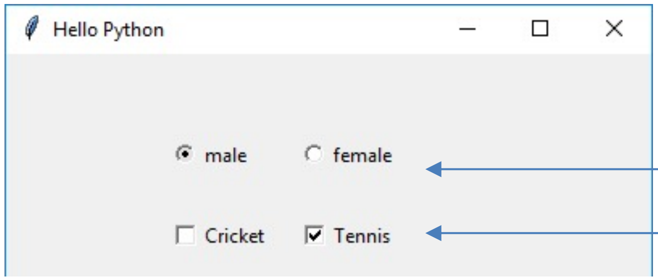
OK button is clicked

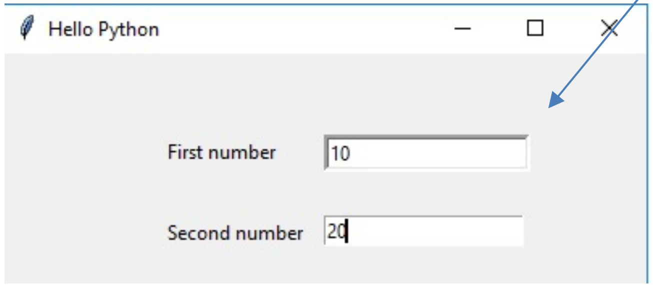
Cancel button is clicked



```
= RESTART: /Users/czh513/Desktop/KIT001/Teaching ir
tonEventClass_p16.py
Ok button is clicked
Ok button is clicked
Cancel button is clicked
Cancel button is clicked
```

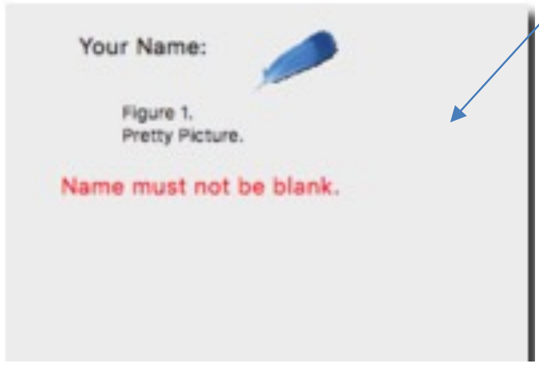
The tk Widget Class





First number 10

Second number 20



Name must not be blank.

Widget Class	Description
Button	A simple button, used to execute a command.
Radio-button	Clicking a radio button sets the variable to the value.
Check-button	Clicking a check button toggles between the values.
Entry	A text entry field or a text box.
Frame	A container widget placed inside a window.
Menu	A menu pane, used to implement pull-down and pop-up menus.
Menu-button	A menu button, used to implement pull down menus.
Label	Displays a text or an image.
Message	Displays a text. Similar to the label widget, but can automatically wrap text to a given width or aspect ratio.
Text	Formatted text display. Allows you to display and edit text with various styles and attributes.

Specifying a Widget 's Colour

- To specify a colour,
  - use either the colour name (there's several pre-defined colours referenced by their name), or explicitly specify the red, green, and blue (RGB) colour components.
- To keep things simple we can use
  - `red`, `yellow`, `green`, `blue`, `white`, `black`, `purple`.
- Note on some platforms (e.g. macOS X)
  - some colour features do not work, as the OS overrides using some *native* widgets to ensure consistent look-and-feel.

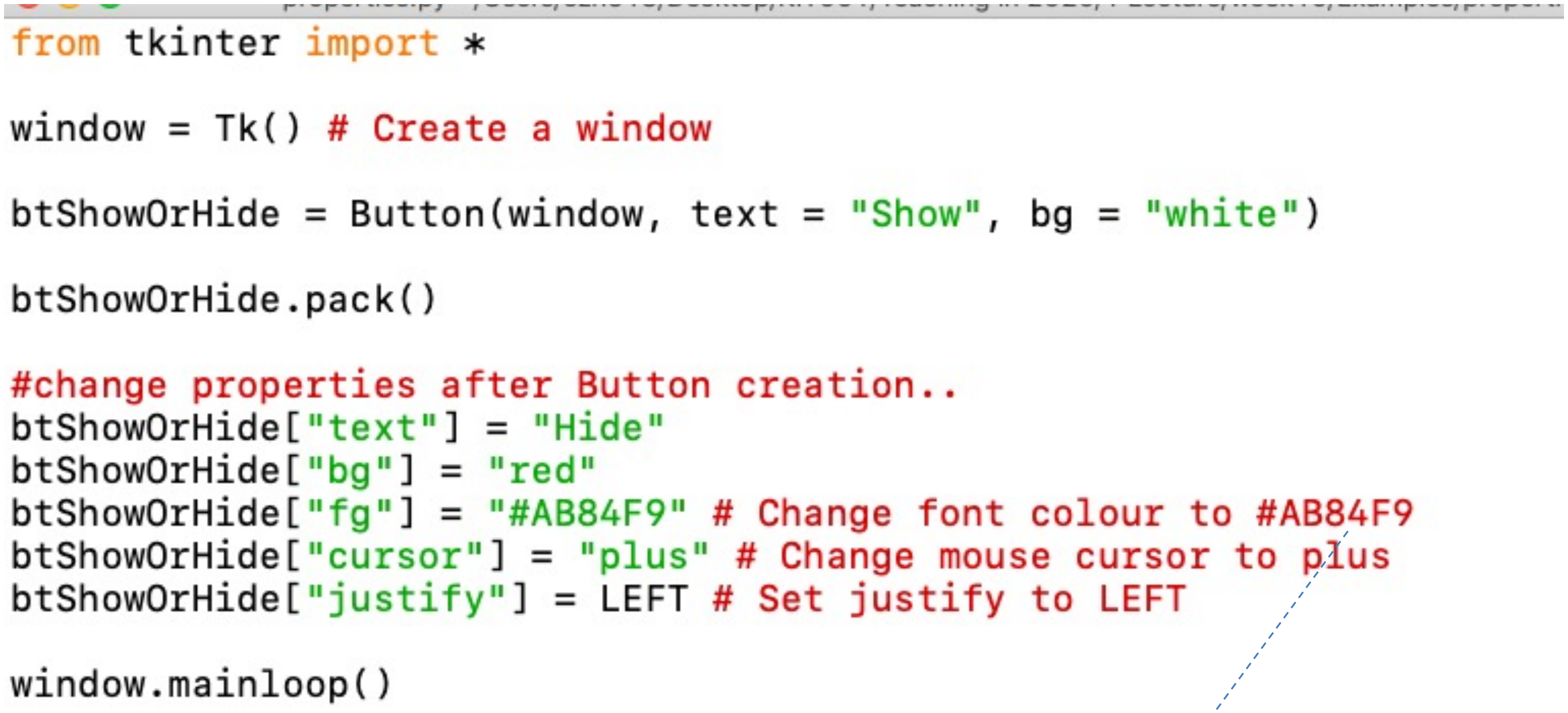
Specify widget font

- We can specify a **font** in a string that includes the font name, size, and style.
  - For example:
    - `Times 10 bold`
    - `Helvetica 10 bold italic`
    - `Courier 20 bold italic overstrike underline`

Text Formatting

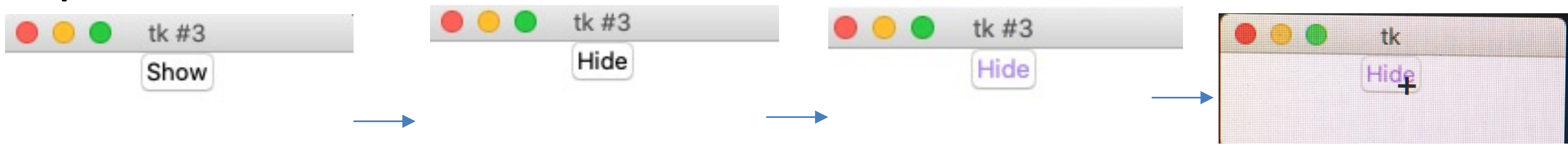
- Text Formatting
  - By default, the text in a label or button is <u>centered</u>.
- Can change using constant variables:
  - `LEFT`
  - `RIGHT`
  - `CENTER`
- Can also use the newline character `\n`

Widget Properties



```
from tkinter import *

window = Tk() # Create a window

btShowOrHide = Button(window, text = "Show", bg = "white")

btShowOrHide.pack()

#change properties after Button creation..
btShowOrHide["text"] = "Hide"
btShowOrHide["bg"] = "red"
btShowOrHide["fg"] = "#AB84F9" # Change font colour to #AB84F9
btShowOrHide["cursor"] = "plus" # Change mouse cursor to plus
btShowOrHide["justify"] = LEFT # Set justify to LEFT

window.mainloop()
```

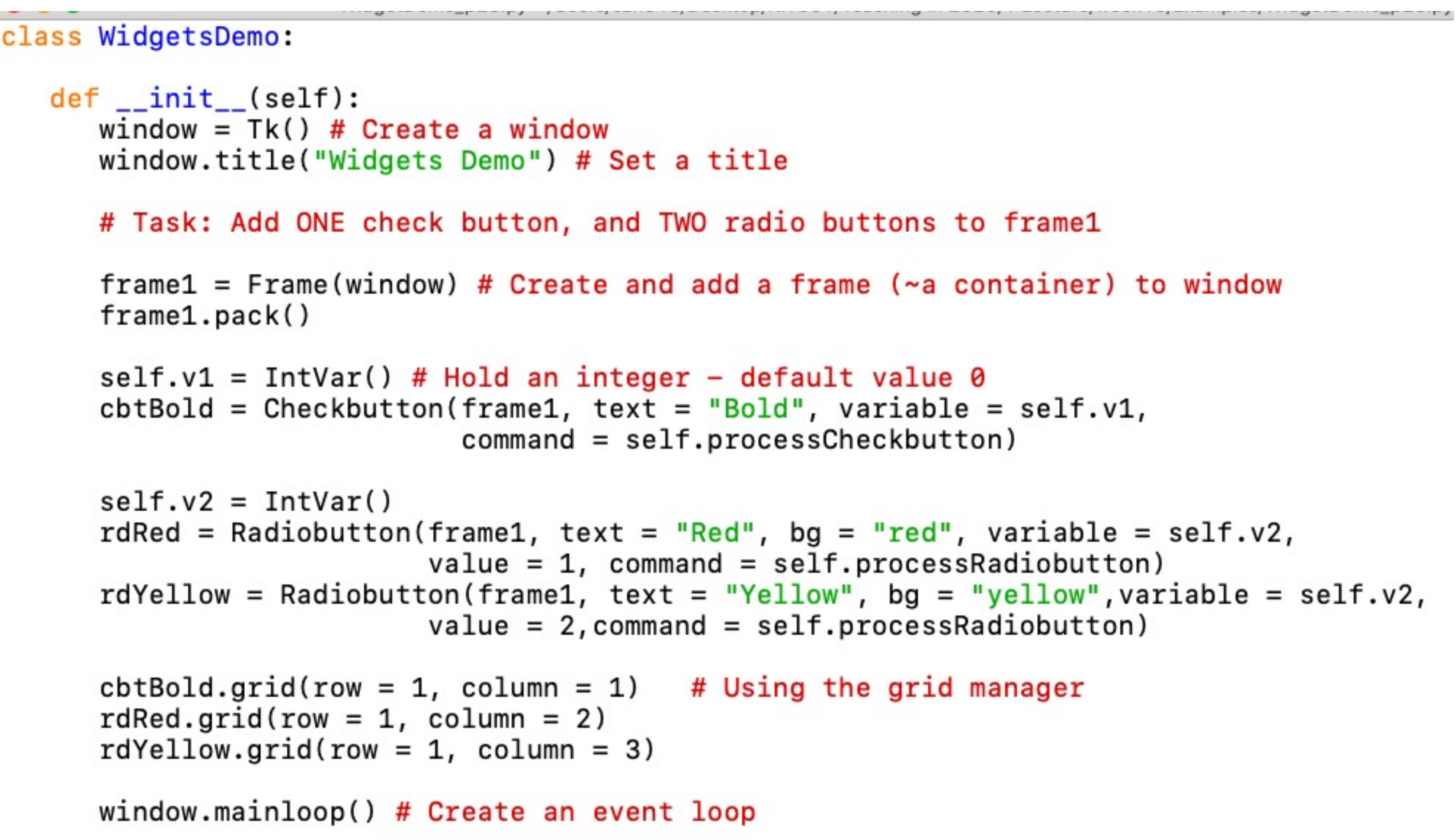
Output:



Widget Demo – 5 examples

- Demo1:
  - Add ONE check button, and TWO radio buttons to a frame
- Demo2
  - Add a label, an entry, a button, and a message to a frame
- Demo3:
  - Add texts
- Demo4:
  - Completed demo - Combine demo 1 + 2 + 3
- Demo5:
  - Change the label (modify from demo 1 + 2)

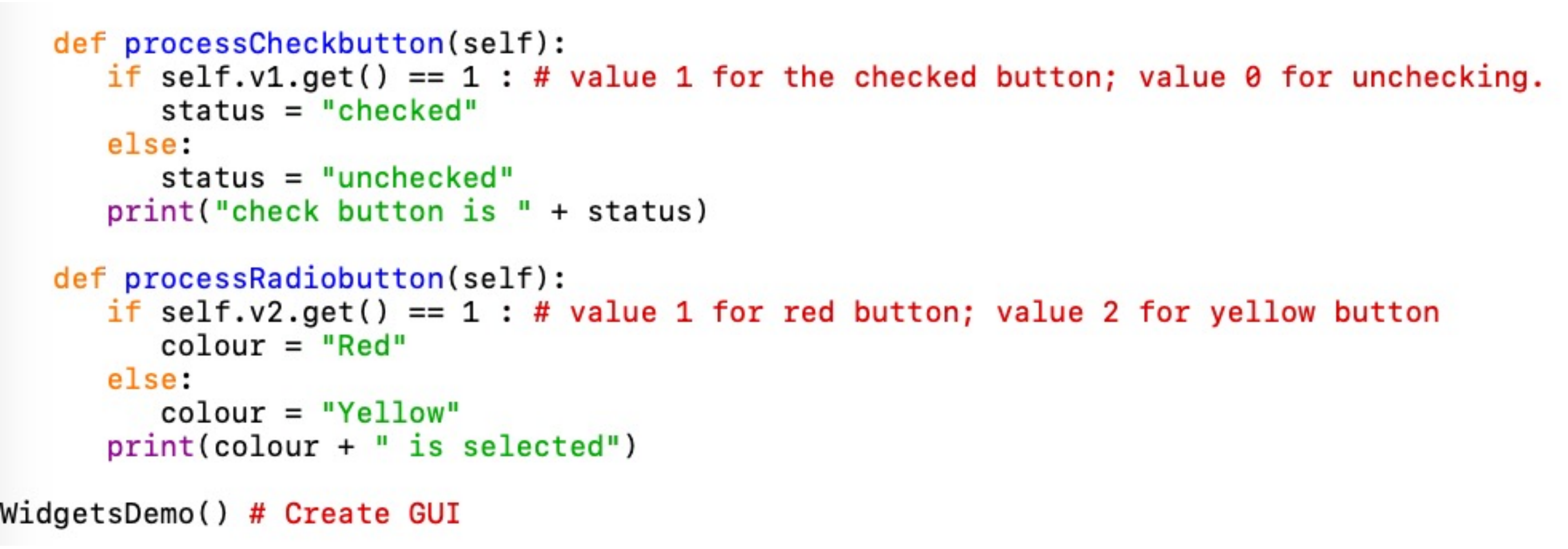
Widget Demo – WidgetDemo1.py



```
class WidgetsDemo:

    def __init__(self):
        window = Tk() # Create a window
        window.title("Widgets Demo") # Set a title

        # Task: Add ONE check button, and TWO radio buttons to frame1

        frame1 = Frame(window) # Create and add a frame (~a container) to window
        frame1.pack()

        self.v1 = IntVar() # Hold an integer - default value 0
        cbtBold = Checkbutton(frame1, text = "Bold", variable = self.v1,
                              command = self.processCheckbutton)

        self.v2 = IntVar()
        rdRed = Radiobutton(frame1, text = "Red", bg = "red", variable = self.v2,
                            value = 1, command = self.processRadiobutton)
        rdYellow = Radiobutton(frame1, text = "Yellow", bg = "yellow",variable = self.v2,
                            value = 2,command = self.processRadiobutton)

        cbtBold.grid(row = 1, column = 1)   # Using the grid manager
        rdRed.grid(row = 1, column = 2)
        rdYellow.grid(row = 1, column = 3)

        window.mainloop() # Create an event loop
```

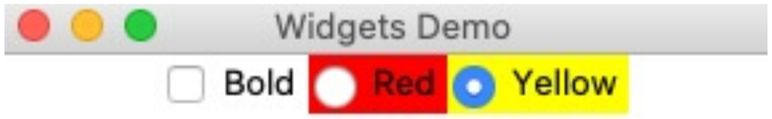
* 'Variable' saves the count for the number of clicking the button.

…continued from the previous slide



```
   def processCheckbutton(self):
      if self.v1.get() == 1 : # value 1 for the checked button; value 0 for unchecking.
         status = "checked"
      else:
         status = "unchecked"
      print("check button is " + status)

   def processRadiobutton(self):
      if self.v2.get() == 1 : # value 1 for red button; value 2 for yellow button
         colour = "Red"
      else:
         colour = "Yellow"
      print(colour + " is selected")

WidgetsDemo() # Create GUI
```

Output Part 1

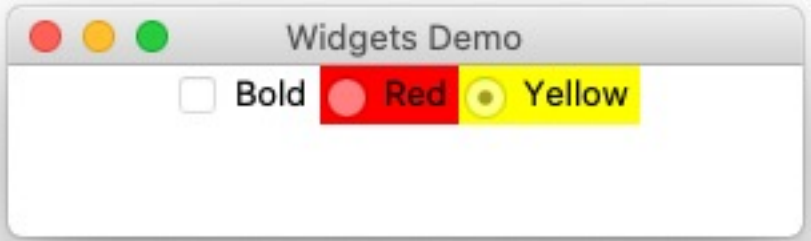


check button is checked

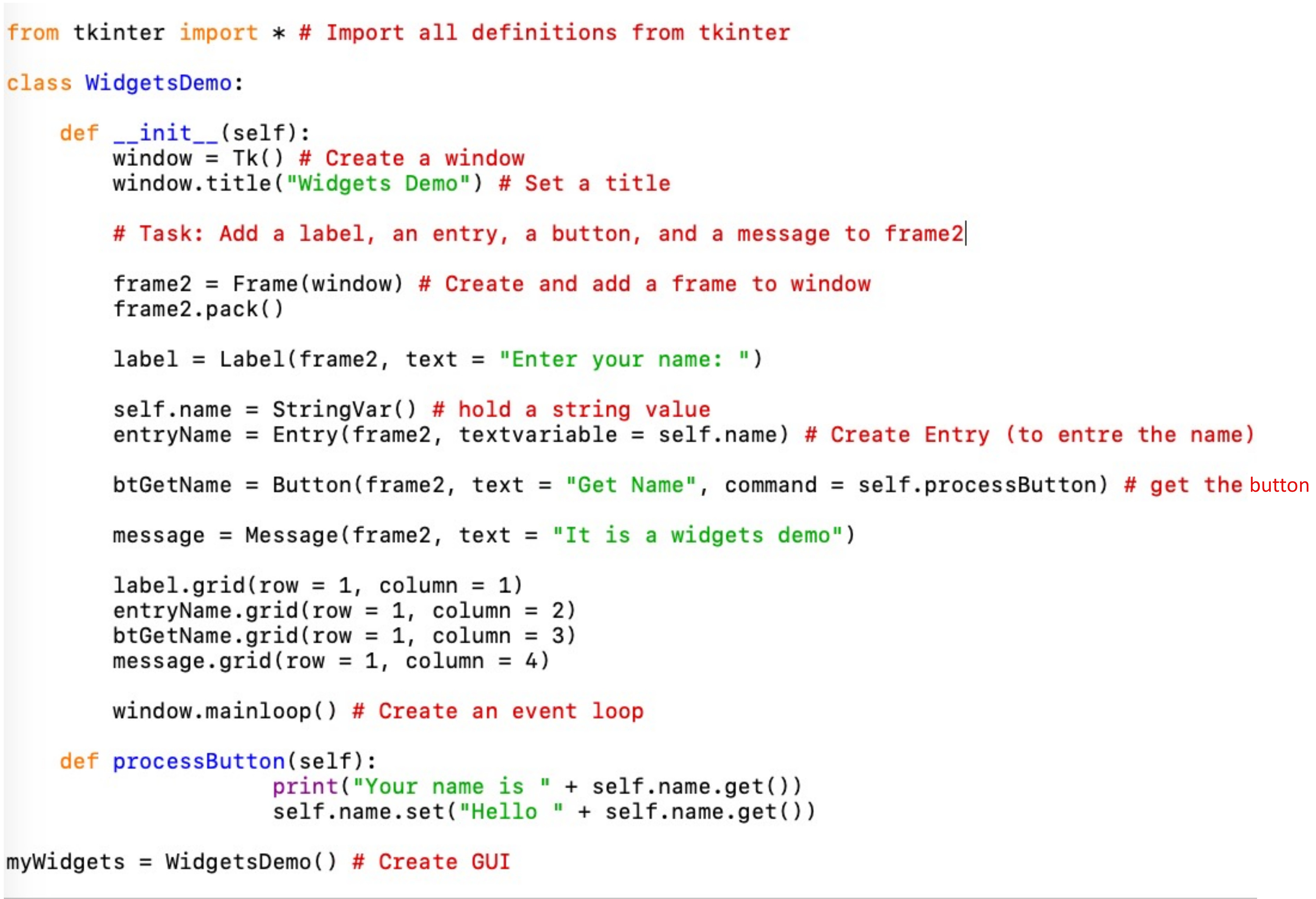
Red is selected

Yellow is selected

```
getDemo_p23.py
check button is checked
check button is unchecked
Red is selected
Yellow is selected
```

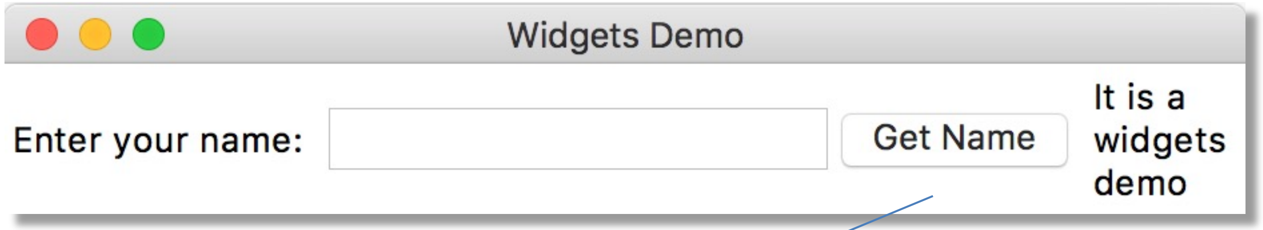


Widget Demo – WidgetDemo2.py

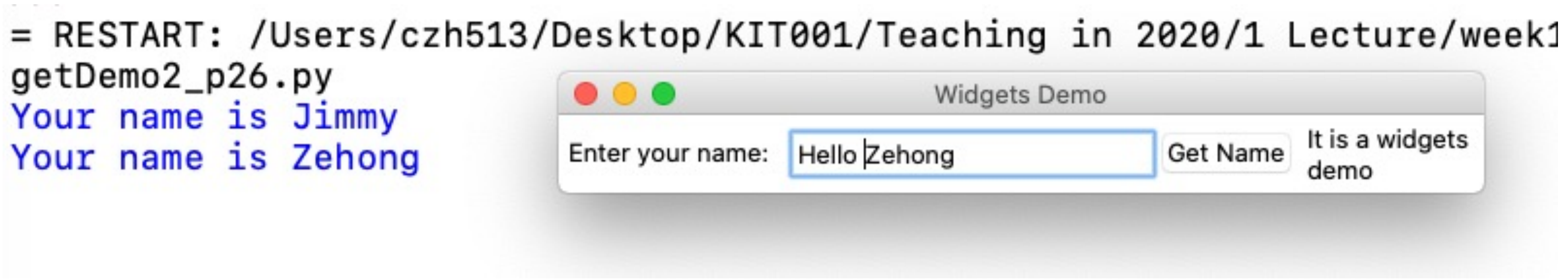


```python
from tkinter import * # Import all definitions from tkinter

class WidgetsDemo:

    def __init__(self):
        window = Tk() # Create a window
        window.title("Widgets Demo") # Set a title

        # Task: Add a label, an entry, a button, and a message to frame2

        frame2 = Frame(window) # Create and add a frame to window
        frame2.pack()

        label = Label(frame2, text = "Enter your name: ")

        self.name = StringVar() # hold a string value
        entryName = Entry(frame2, textvariable = self.name) # Create Entry (to entre the name)

        btGetName = Button(frame2, text = "Get Name", command = self.processButton) # get the button

        message = Message(frame2, text = "It is a widgets demo")

        label.grid(row = 1, column = 1)
        entryName.grid(row = 1, column = 2)
        btGetName.grid(row = 1, column = 3)
        message.grid(row = 1, column = 4)

        window.mainloop() # Create an event loop

    def processButton(self):
                    print("Your name is " + self.name.get())
                    self.name.set("Hello " + self.name.get())

myWidgets = WidgetsDemo() # Create GUI
```

Output Part 2

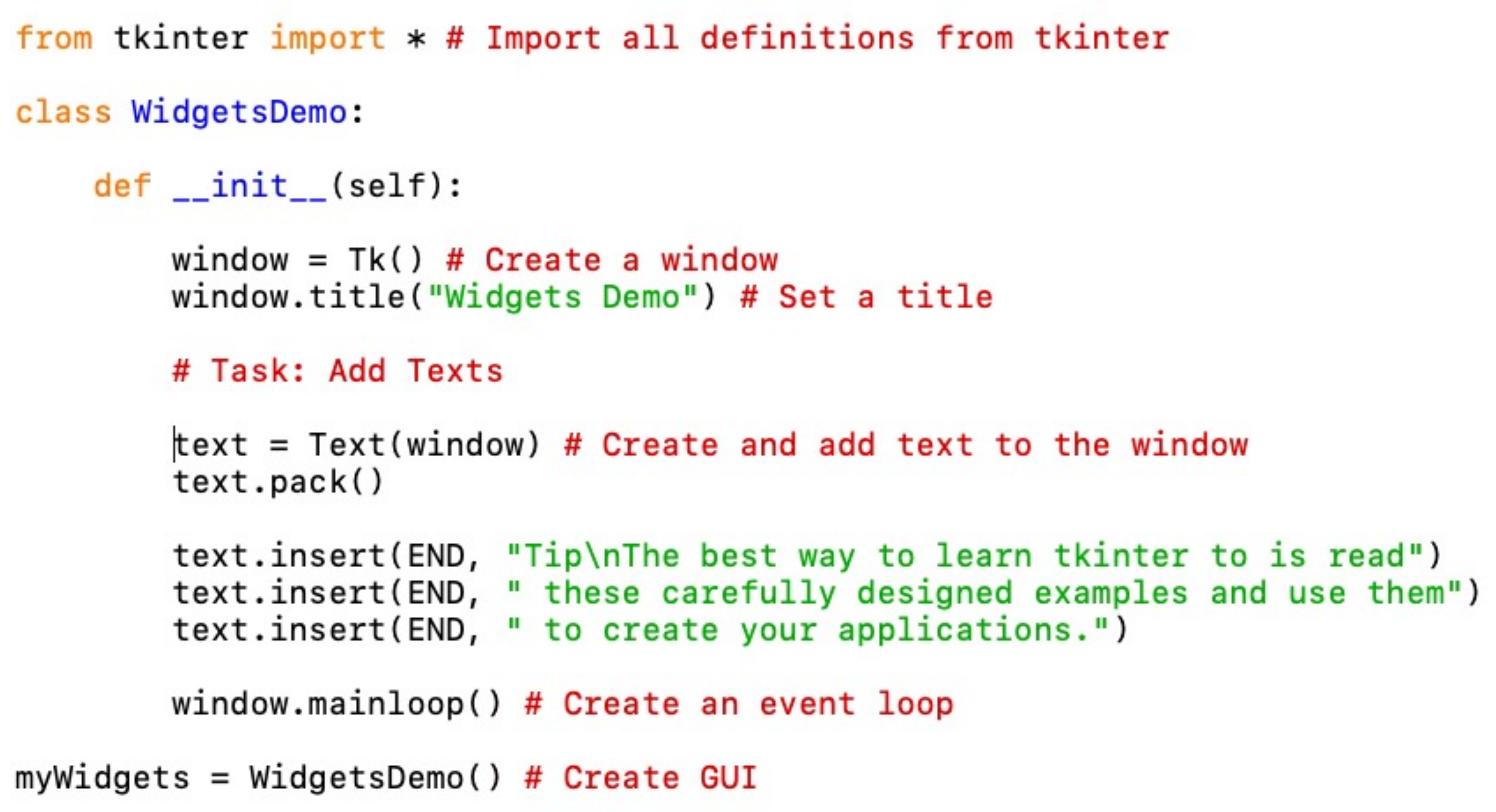


```
Your name is Jimmy
```



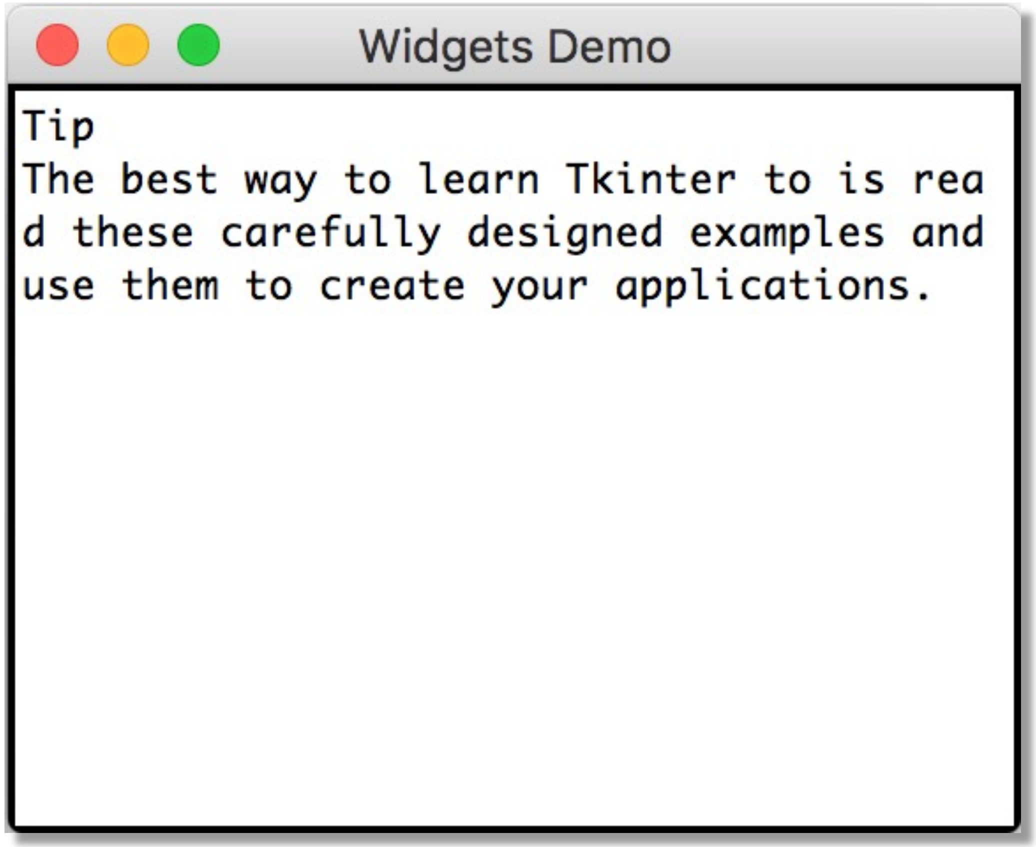
```
= RESTART: /Users/czh513/Desktop/KIT001/Teaching in 2020/1 Lecture/week1
getDemo2_p26.py
Your name is Jimmy
Your name is Zehong
```

Widget Demo – WidgetDemo3.py



```
from tkinter import * # Import all definitions from tkinter

class WidgetsDemo:

    def __init__(self):

        window = Tk() # Create a window
        window.title("Widgets Demo") # Set a title

        # Task: Add Texts

        text = Text(window) # Create and add text to the window
        text.pack()

        text.insert(END, "Tip\nThe best way to learn tkinter to is read")
        text.insert(END, " these carefully designed examples and use them")
        text.insert(END, " to create your applications.")

        window.mainloop() # Create an event loop

myWidgets = WidgetsDemo() # Create GUI
```

Output Part 3



```
Tip
The best way to learn Tkinter to is rea
d these carefully designed examples and
use them to create your applications.
```

Widget Demo – WidgetDemo4.py

```
"""
Title:Widget Demo
Purpose: To demonstrate the use of frame, button, checkbutton, radiobutton,
label, entry, mesCosage and text widgets
"""

from tkinter import * # Import all definitions from tkinter

class WidgetsDemo:
    def __init__(self):
        window = Tk() # Create a window
        window.title("Widgets Demo") # Set a title

        # Task: Add ONE check button, and TWO radio button to frame1
        frame1 = Frame(window) # Create and add a frame to window
        frame1.pack()
        self.v1 = IntVar()
        cbtBold = Checkbutton(frame1, text = "Bold", variable = self.v1, command = self.processCheckbutton)
        self.v2 = IntVar()
        rdRed = Radiobutton(frame1, text = "Red", bg = "red", variable = self.v2, value = 1, command = self.processRadiobutton)
        rdYellow = Radiobutton(frame1, text = "Yellow", bg = "yellow", variable = self.v2, value = 2, command = self.processRadiobutton)
        cbtBold.grid(row = 1, column = 1)   # Using the grid manager
        rdRed.grid(row = 1, column = 2)
        rdYellow.grid(row = 1, column = 3)

        # Task: Add a label, an entry, a button, and a message to frame2
        frame2 = Frame(window) # Create and add a frame to window
        frame2.pack()
        label = Label(frame2, text = "Enter your name: ")
        self.name = StringVar()
        entryName = Entry(frame2, textvariable = self.name) # Create Entry
        btGetName = Button(frame2, text = "Get Name",
                           command = self.processButton)
        message = Message(frame2, text = "It is a widgets demo")
        label.grid(row = 1, column = 1)
        entryName.grid(row = 1, column = 2)
        btGetName.grid(row = 1, column = 3)
        message.grid(row = 1, column = 4)
```

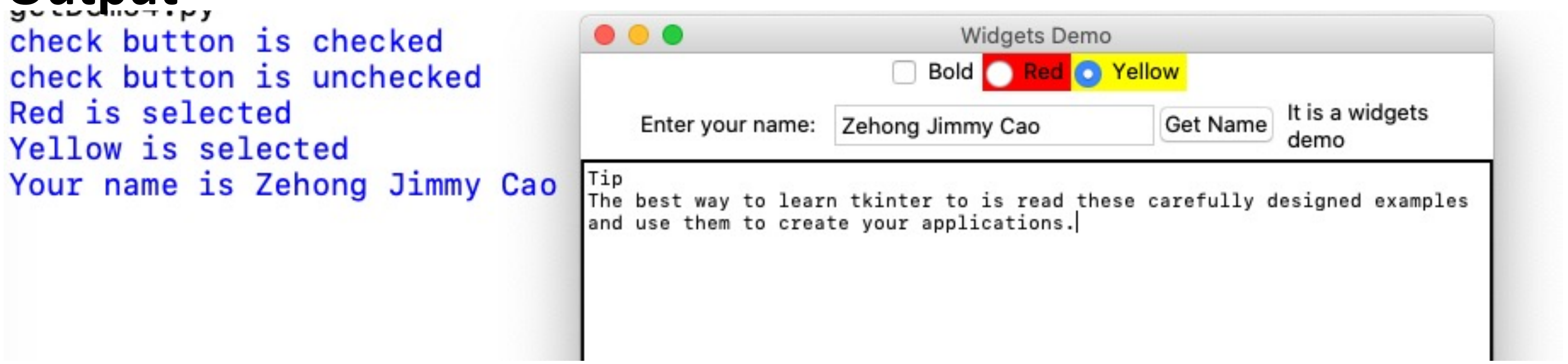
Widget Demo – WidgetDemo4.py

... Continued from the previous slide

```
        # Task: Add Text
        text = Text(window) # Create and add text to the window
        text.pack()
        text.insert(END, "Tip\nThe best way to learn tkinter to is read")
        text.insert(END, " these carefully designed examples and use them")
        text.insert(END, " to create your applications.")

        window.mainloop() # Create an event loop

    def processCheckbutton(self):
       if self.v1.get() == 1:
          status = "checked"
       else:
          status = "unchecked"
       print("check button is " + status)

    def processRadiobutton(self):
       if self.v2.get() == 1:
          colour = "Red"
       else:
          colour = "Yellow"
       print(colour + " is selected")

    def processButton(self):
       print("Your name is " + self.name.get())

myWidgets = WidgetsDemo() # Create GUI
```

Complete Output

Output



```
check button is checked
check button is unchecked
Red is selected
Yellow is selected
Your name is Zehong Jimmy Cao
```

Zehong Jimmy Cao

```
Tip
The best way to learn tkinter to is read these carefully designed examples
and use them to create your applications.
```

Widget Demo – WidgetDemo5.py

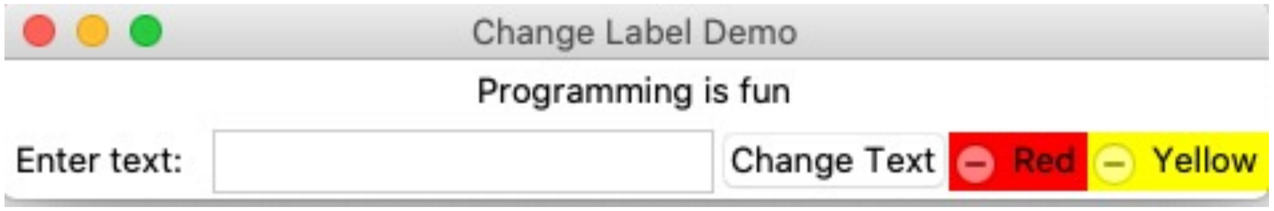
```python
from tkinter import * # Import tkinter

class ChangeLabelDemo:

    def __init__(self):
        window = Tk() # Create a window
        window.title("Change Label Demo") # Set a title

        # Task1: Add a label to frame1

        frame1 = Frame(window) # Create and add a frame to window
        frame1.pack()
        self.lbl = Label(frame1, text = "Programming is fun")
        self.lbl.pack()

        # Task 2: Add a label, an entry, a button, and two radio buttons to frame2

        frame2 = Frame(window) # Create and add a frame to window
        frame2.pack()

        label = Label(frame2, text = "Enter text: ") # create a label

        self.msg = StringVar() #hold a string value
        entry = Entry(frame2, textvariable = self.msg) # Create entry
        btChangeText = Button(frame2, text = "Change Text", command = self.processButton) # Button callback

        self.v1 = StringVar() # hold another string value
        rbRed = Radiobutton(frame2, text = "Red", bg = "red",variable = self.v1,
                            value = 'R', command = self.processRadiobutton) # Radio Button callback
        rbYellow = Radiobutton(frame2, text = "Yellow", bg = "yellow",variable = self.v1,
                               value = 'Y',command = self.processRadiobutton)
```

Widget Demo – WidgetDemo5.py

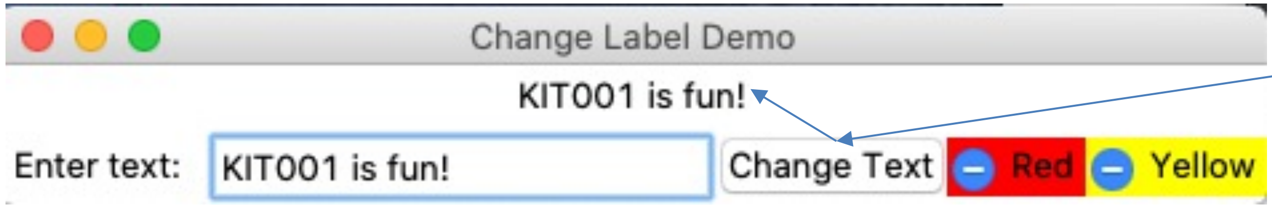
... Continue from the previous slide

```python
        label.grid(row = 1, column = 1)
        entry.grid(row = 1, column = 2)
        btChangeText.grid(row = 1, column = 3)
        rbRed.grid(row = 1, column = 4)
        rbYellow.grid(row = 1, column = 5)

        window.mainloop() # Create an event loop

    def processRadiobutton(self):
        if self.v1.get() == 'R':
            self.lbl["fg"] = "red" # set a new fg
        elif self.v1.get() == 'Y':
            self.lbl["fg"] = "yellow" # set a new fg

    def processButton(self):
        self.lbl["text"] = self.msg.get() # New text for the label

myWidgets = ChangeLabelDemo() # Create GUI
```

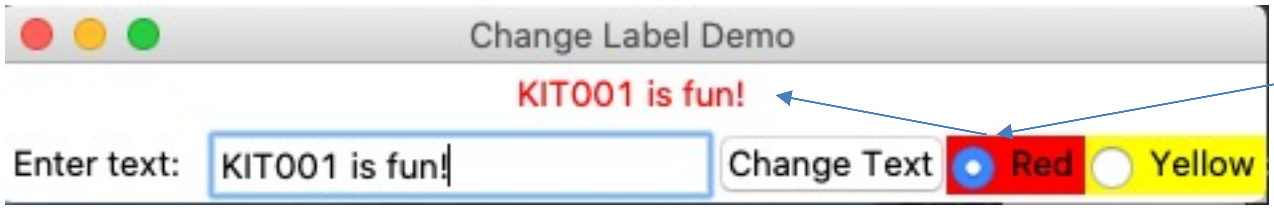
Output



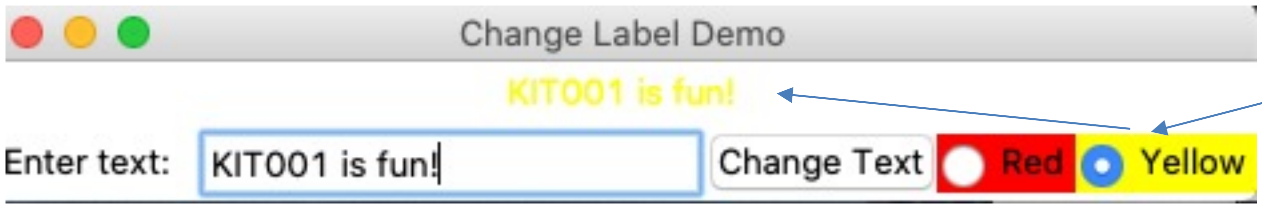
Default



Change Text button pressed



Red radio button selected



Yellow radio button selected

Designing your GUI

- Before you start to write the code for your GUI
  - you will need to plan it, draw a picture.

- Back to the pen and paper
- Think about
  - How do you want your GUI to look like?
  - Where do you want things placed?

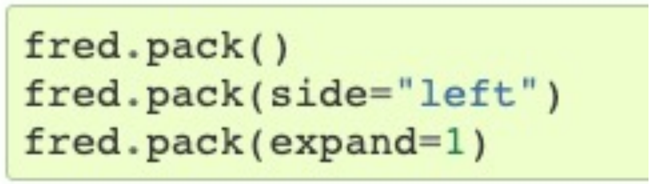
Organising Widgets with Frames

- So far when we have packed the widgets into the window, they always go under (beneath) the previous widget.

- What if we want to get them to go side-by-side (paralleled) or some other place?
  - Most windowing toolkits have layout management systems to help you arrange widgets!

Using pack()

- `pack()` will **center** the widget in the frame by default.
- `myWidget.pack(side=LEFT)`
  - this tells pack to put this widget to the left of the next widget.
  - `LEFT`, `RIGHT`, `TOP`, `BOTTOM`
- https://docs.python.org/3/library/tkinter.html#the-packer

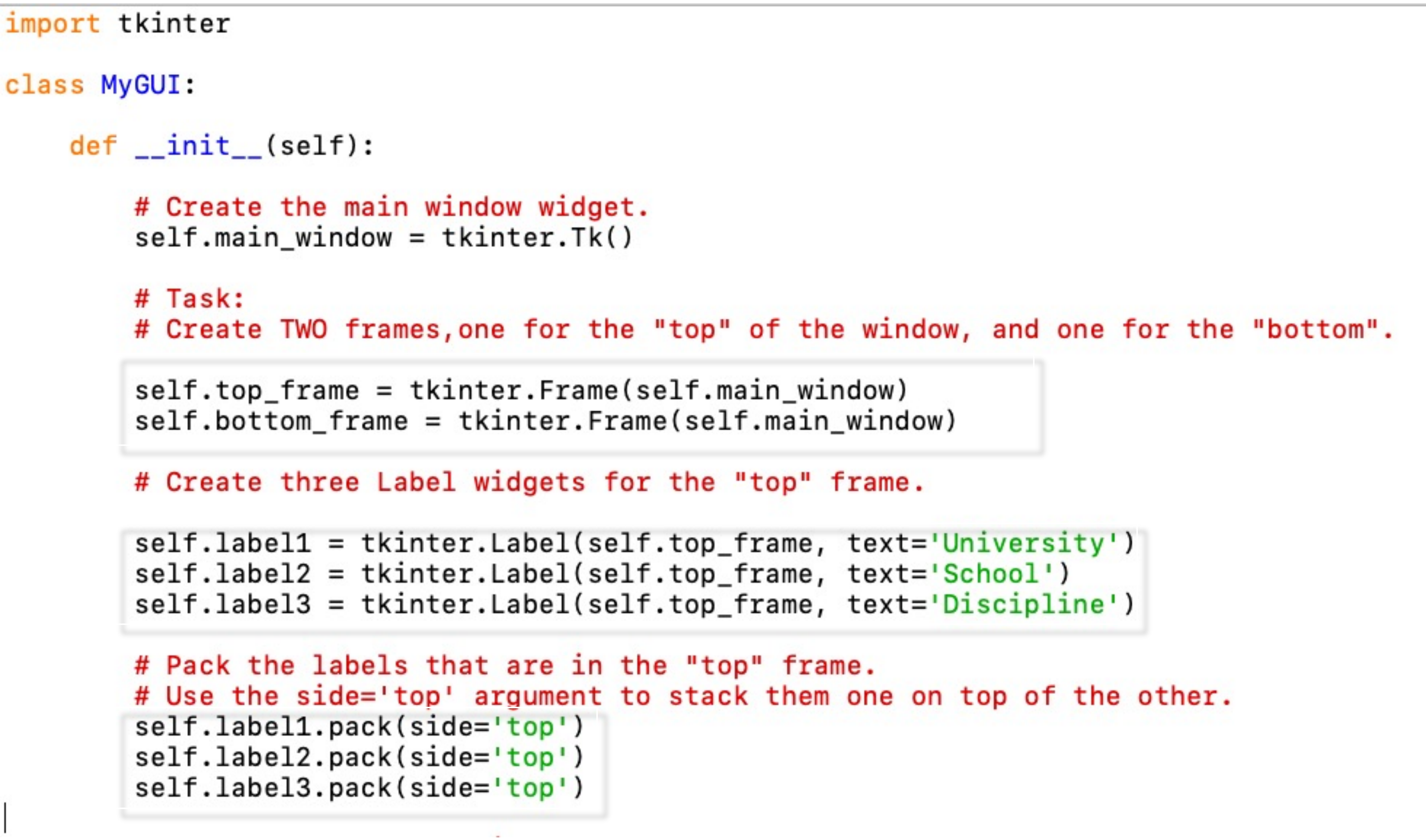


```
fred.pack()
fred.pack(side="left")
fred.pack(expand=1)
```

Packing Frames

- Usually you cannot get the desired look with pack
  - unless you use Frames
- Frames are widgets that contain in a window.
  - It works like a container.
  - lets us organize and group widgets.

Widget Demo – FrameDemo.py

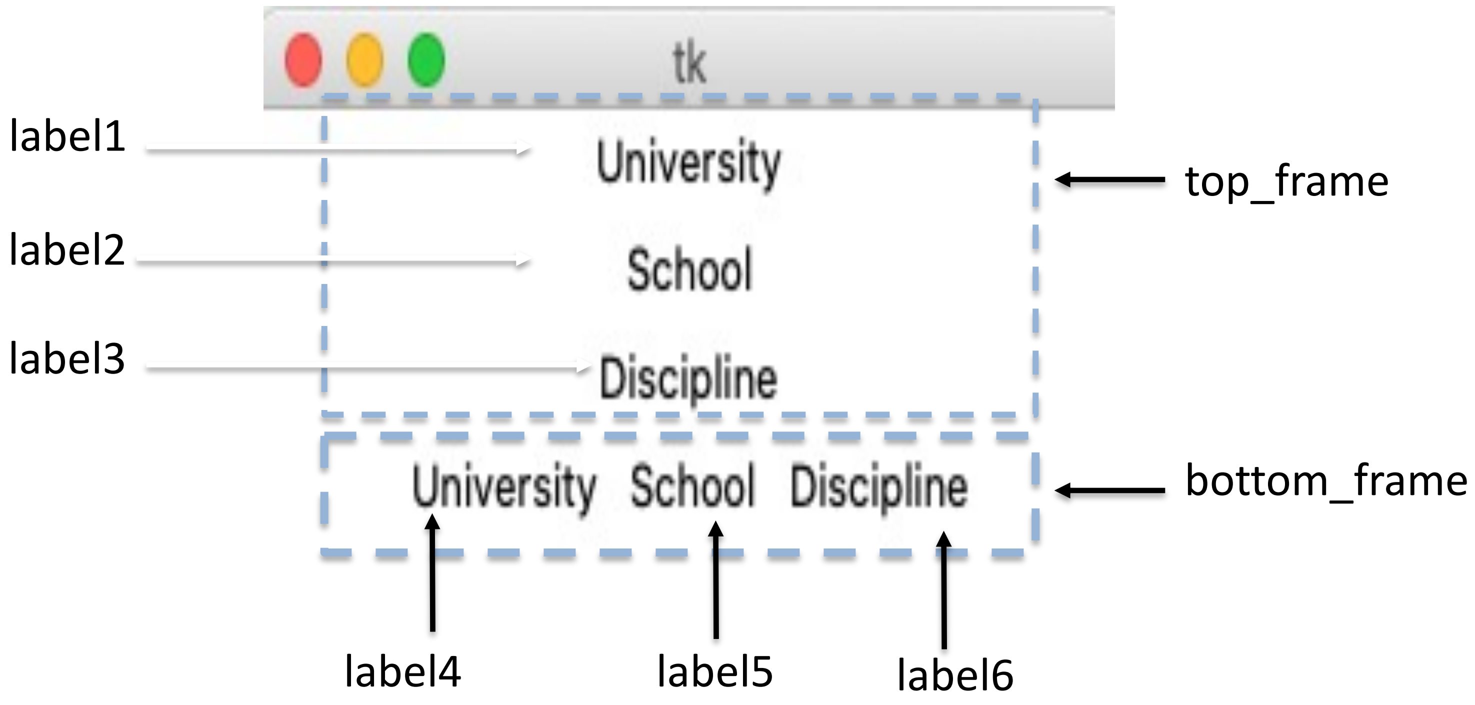


```
import tkinter

class MyGUI:

    def __init__(self):

        # Create the main window widget.
        self.main_window = tkinter.Tk()

        # Task:
        # Create TWO frames,one for the "top" of the window, and one for the "bottom".

        self.top_frame = tkinter.Frame(self.main_window)
        self.bottom_frame = tkinter.Frame(self.main_window)

        # Create three Label widgets for the "top" frame.

        self.label1 = tkinter.Label(self.top_frame, text='University')
        self.label2 = tkinter.Label(self.top_frame, text='School')
        self.label3 = tkinter.Label(self.top_frame, text='Discipline')

        # Pack the labels that are in the "top" frame.
        # Use the side='top' argument to stack them one on top of the other.
        self.label1.pack(side='top')
        self.label2.pack(side='top')
        self.label3.pack(side='top')
```

Widget Demo – FrameDemo.py

```
        # Create three Label widgets for the "bottom" frame.

        self.label4 = tkinter.Label(self.bottom_frame, text='University')
        self.label5 = tkinter.Label(self.bottom_frame, text='School')
        self.label6 = tkinter.Label(self.bottom_frame, text='Discipline')

        # Pack the labels that are in the "bottom" frame.
        # Use the side='left' argument to arrange them horizontally from the left of the frame.
        self.label4.pack(side='left')
        self.label5.pack(side='left')
        self.label6.pack(side='left')

        # Yes, we have to pack the frames too!
        self.top_frame.pack()
        self.bottom_frame.pack()

        # Enter the tkinter main loop.
        tkinter.mainloop()

# Create an instance of the MyGUI class.
myGui = MyGUI()
```

Organising your GUI



Questions?