Tkinter:
Input and Output Bindings

Marquette University

Tkinter Variables

- Tkinter contains a useful mechanism to connect widgets to variables
    - This allows us to have variables change when widgets do and vice versa

Radio Buttons

- Radio buttons are used to select from a menu
  - Define an integer variable in order to connect it to the radio button widget
  - Use trace in order to be informed when the variable changes.

Radio Buttons

- Normal window creation

```
class GUI:
    def __init__(self):
        self.window = tk.Tk()
        self.window.title = "Radio"
        self.window.geometry("250x200")
        self.make_widgets()
        self.window.mainloop()
```

Radio Buttons

- In make_widgets:
  - Create two labels
  - Create list of radiobuttons

Radio Buttons

```
def make_widgets(self):
        self.int_var = tk.IntVar()
        self.int_var.set(0)
        self.label1 = tk.Label(self.window, text = "Make a choice")
        self.label1.pack(anchor=tk.N)
        self.radios = []
        counter = 0
        for language in /
          ["C", "C++", "C# ", "Fortran", "Java", "Python"]:
    counter += 1
    b = tk.Radiobutton self.window, text = language,
                                                    )
    self.rad
    b.pack(anchor = tk.W)
self.int_var.trace("w", self.show_choice)
self.label2 = tk.Label(self.window, text = self.int_var.get(
self.label2.pack(anchor=tk.S)
```

Create Integer Variable

Radio Buttons

```
def make_widgets(self):
        self.int_var = tk.IntVar()
        self.int_var.set(0)
        self.label1 = tk.Label(self.window, text = "Make a choice")
        self.label1.pack(anchor=tk.N)
        self.radios = []
        counter = 0
        for language in /
          ["C", "C++", "C#", "Fortran", "Java", "Python"]:
    counter += 1
    b = tk.Radiobutton(self.window, text = language,
                       variable=self.int_var
                       )
    self.radios.append(b)
    b.pack(anchor = tk.W)
self.int_var.trace("w", self.show_choice)
self.label2 = tk.Label(self.window, text = self.int_var.get(
self.label2.pack(anchor=tk.S)
```

Start creating radio buttons in a list

Radio Buttons

```
def make_widgets(self):
        self.int_var = tk.IntVar()
        self.int_var.set(0)
        self.label1 = tk.Label(self.window, text = "Make a choice")
        self.label1.pack(anchor=tk.N)
        self.radios = []
        counter = 0
        for language in /
            ["C", "C++", "C#", "Fortran", "Java", "Python"]:
        counter += 1
        b = tk.RadioButton(self.window, text = language,
                           variable = self.int_var
                                                    )
        self.radios.append(b)
        b.pack(anchor = tk.W)
self.int_var.trace("w", self.show_choice)
self.label2 = tk.Label(self.window, text = self.int_var.get(
self.label2.pack(anchor=tk.S)
```

A way to place python statements on two lines

Radio Buttons

```
def make_widgets(self):
        self.int_
        self.int_
        self.labe                                          ke a choice")
        self.labe
        self.radios = []
        counter = 0
        for language in /
            ["C", "C++", "C#", "Fortran", "Java", "Python"]:
            counter += 1
            b = tk.Radiobutton(self.window, text = language,
                               variable = self.int_var,
                               value = counter)
                               #command = self.show_choice)
            self.radios.append(b)
            b.pack(anchor = tk.W)
    self.int_var.trace("w", self.show_choice)
    self.label2 = tk.Label(self.window, text = self.int_var.get(
    self.label2.pack(anchor=tk.S)
```

Specify connection to int_var via the variable parameter

Radio Buttons

```
def make_widgets(self):
        self.int_
        self.int_
        self.labe                                              ke a choice")
        self.labe
        self.radios = []
        counter = 0
        for language in /
            ["C", "C++", "C#", "Fortran", "Java", "Python"]:
            counter += 1
            b = tk.Radiobutton(self.window, text = language,
                               variable = self.int_var,
                               value = counter)
                               #command = self.show_choice)
            self.radios.append(b)
            b.pack(anchor = tk.W)
    self.int_var.trace("w", self.show_choice)
    self.label2 = tk.Label(self.window, text = self.int_var.get(
    self.label2.pack(anchor=tk.S)
```

Assign it a value

Radio Buttons

```
def make_widgets(self):
        self.int_
        self.int_
        self.labe                                          ke a choice")
        self.labe
        self.radios = []
        counter = 0
        for language in /
            ["C", "C++", "C#", "Fortran", "Java", "Python"]:
            counter += 1
            b = tk.Radiobutton(self.window, text = language,
                               variable = self.int_var,
                               value = counter)
                               #command = self.show_choice)
            self.radios.append(b)
            b.pack(anchor = tk.W)
    self.int_var.trace("w", self.show_choice)
    self.label2 = tk.Label(self.window, text = self.int_var.get(
    self.label2.pack(anchor=tk.S)
```

Alternative to trace selections by user by a command fucntion

Radio Buttons

```
def make_widgets(self):
        self.int_
        self.int_
        self.labe                                          ke a choice")
        self.labe
        self.radios = []
        counter = 0
        for language in /
          ["C", "C++"  "C#", "Fortran", "Java", "Python"]:
            counter += 1
            b = tk.Radiobutton(self.window, text = language,
                               variable = self.int_var,
                               value = counter)
                               #command = self.show_choice)
            self.radios.append(b)
            b.pack(anchor = tk.W)
    self.int_var.trace("w", self.show_choice)
    self.label2 = tk.Label(self.window, text = self.int_var.get(
    self.label2.pack(anchor=tk.S)
```

Anchor is another way in pack to explain where you want elements to be placed

Radio Buttons

```
def make_widgets(self):
        self.int_
        self.int_
        self.labe                                          ke a choice")
        self.labe
        self.radios = []
        counter = 0
        for language in /
            ["C", "C++", "C ", "Fortran", "Java", "Python"]:
            counter += 1
            b = tk.Radiobutto (self.window, text = language,
                               variable = self.int_var,
                               value = counter)
                               #command = self.show_choice)
            self.radios.appen (b)
            b.pack(anchor = tk.W)
    self.int_var.trace("w", self.show_choice)
    self.label2 = tk.Label(self.window, text = self.int_var.get(
    self.label2.pack(anchor=tk.S)
```

Set the trace to have a command executed whenever the value of the variable changes

Radio Buttons

- Reaction to selection is just changing the second label

```
def show_choice(self, *args):
    print(args)
    self.label2.configure(text = self.int_var.get())
```

Using Variable Classes

Variable Classes

- To recapitulate:
  - Create TkInter variables: `var = StringVar()`
  - Use set and get to write and read
    - `var.set("hi")  var.get()`
- Can use trace to attach an observing callback function
  - `var.trace("w", callback)`

Input / Output in Tkinter

- Entry widget allows user to enter text in tkinter
- Create widget with entry box plus four buttons
- Changes distances between miles and kilometers and yards and meters

Distance Calculator

- Create window with a string tkinter variable

```
import tkinter as tk

class MyApp:
    dictionary = {"km2m": 0.621371, "m2km": 1.60934,
                  "m2y": 1.09361, "y2m": 0.9144}
    def __init__(self):
        MyApp.main_window = tk.Tk()
        MyApp.main_window.title("Converter")
        MyApp.entry_variable = tk.StringVar()
        MyApp.entry_variable.set("hello")
        MyApp.make_widgets()
        MyApp.main_window.mainloop()
```

Distance Calculator

- Create widgets:
  - Entry box connected to the tkinter variable we just defined via textvariable

```
def make_widgets():
        MyApp.entry = tk.Entry(MyApp.main_window,
                textvariable=MyApp.entry_variable)
        MyApp.entry.grid(row=0, column =0, columnspan = 4)
        MyApp.button1 = tk.Button(MyApp.main_window, text="KM2M",
                        command = lambda : MyApp.callback("km2m"))
        MyApp.button1.grid(row=1, column = 0)
        MyApp.button2 = tk.Button(MyApp.main_window, text="M2KM",
                        command = lambda : MyApp.callback("m2km"))
        MyApp.button2.grid(row=1, column = 1)
        MyApp.button3 = tk.Button(MyApp.main_window, text="M2Y",
                          command = lambda : MyApp.callback("m2y"))
        MyApp.button3.grid(row=1, column = 2)
        MyApp.button4 = tk.Button(MyApp.main_window, text="Y2M",
```

Distance Calculator

- Create a bunch of buttons and place them
- Use the lambda trick to pass the parameter to callback

```
MyApp.button1 = tk.Button(MyApp.main_window, text="KM2M",
                          command = lambda : MyApp.callback("km2m"))
MyApp.button1.grid(row=1, column = 0)
MyApp.button2 = tk.Button(MyApp.main_window, text="M2KM",
             command = lambda : MyApp.callback("m2km"))
MyApp.button2.grid(row=1, column = 1)
MyApp.button3 = tk.Button(MyApp.main_window, text="M2Y",
             command = lambda : MyApp.callback("m2y"))
MyApp.button3.grid(row=1, column = 2)
MyApp.button4 = tk.Button(MyApp.main_window, text="Y2M",
              command = lambda : MyApp.callback("y2m"))
MyApp.button4.grid(row=1, column = 3)
```

Distance Calculator

- Get the string from the string tkinter variable
  - Convert it to a number (if possible)
  - Change in the required manner
  - Reset the text in the Entry box

```
def callback(cmd):
        number = MyApp.entry_variable.get()
        try:
            num = float(number)
            MyApp.entry_variable.set("{:7.3f}".format(round(num *
                                        MyApp.dictionary[cmd],3)))
        except:
            MyApp.entry_variable.set("Error")
```

Binding

Binding

- Binding allows us to bind events to windows, so that they can be handled
    - We can bind at different levels and bindings can be very sophisticated.
        - For example, we can have an action if the mouse cursor enters a window or leaves a window, …
- The usual types of events are mouse clicks and keyboard entries.

Binding

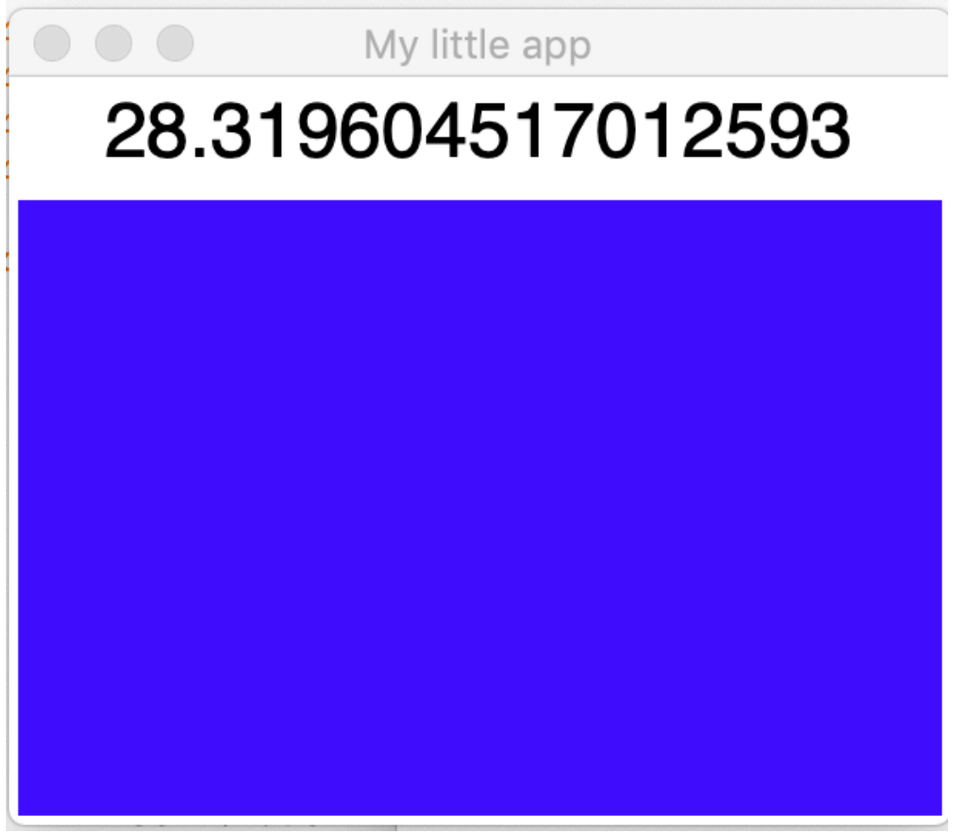
- To illustrate binding mouse clicks
  - Use event type <Button-1>
    - Can have up to three buttons on your mouse with special ways to emulate three-button mice with two button mice.
    - Three button mice were part of the SUN workstations

Bindings

- Event structure allows us to access information about an event
  - Mouse-clicks: get x and y position

Guessing Game

- Guessing a secret location on the canvas given only the distance
- Need to get within 5 points



Guessing Game

- Create a string tkinter variable
- Bind the mouse click to the callback function
- Have the callback function use the components of the event to determine the distance from the mouse click to the secret location

Guessing Game

```
class My_App:
    def distance(pt1, pt2):
        return math.sqrt((pt1[0]-pt2[0])**2 + (pt1[1]-pt2[1])**2)
    def __init__(self):
        self.main_window = tk.Tk()
        self.main_window.title("My little app")
        self.secret = (rd.randint(0,299), rd.randint(0,199))
        self.text = tk.StringVar()
        self.text.set("Hello")
        self.make_widgets()
        self.main_window.mainloop()
```

Guessing Game

- Create label and canvas
- Bind event to the canvas

```
def make_widgets(self):
        self.label = tk.Label(self.main_window,
          textvariable = self.text,
          font=("Helvetica", 25))
self.label.pack()
self.canvas = tk.Canvas(self.main_window,
      height = 200, width=300,
      background = "#4010ff")
self.canvas.pack()
self.canvas.bind("<Button-1>",
        lambda e: self.callback(e))
```

Guessing Game

- Callback can use event
- print out event details for debug purposes

```
def callback(self, event):
        print(repr(event))
        pos = event.x, event.y
        dis = My_App.distance(pos, self.secret)
        if dis<5:
            self.text.set("Winner")
        else:
            self.text.set(round(dis,1))
```