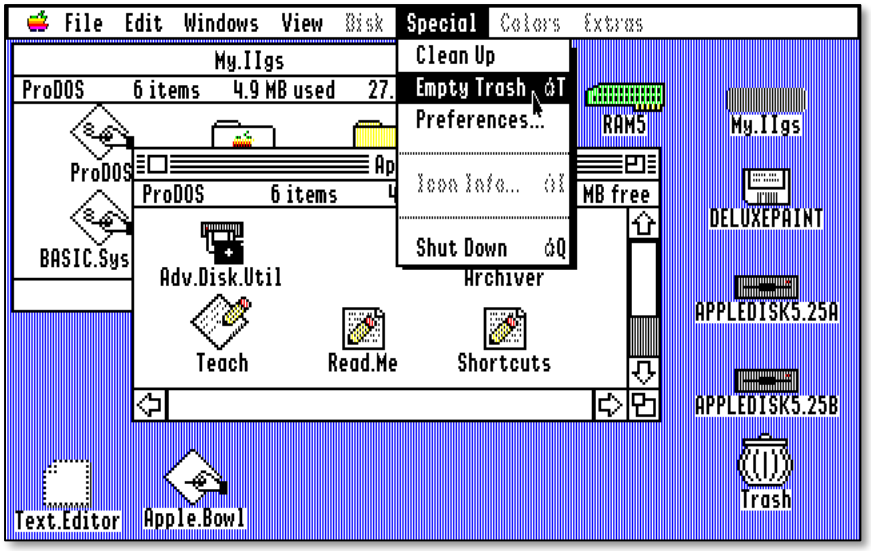
ITP 115 – Programming in Python

GUIs

Graphical User Interface (GUI)

- What the user sees and interacts with
- Type of user interface that allows users to interact with electronic devices with images rather than text commands
- Components
  - Icons
  - Pointer
  - Window
  - Menus
  - Dialog Boxes
  - Toolbars



Graphical User Interface

- A graphical user interface (GUI) provides a visual way for a user to interact with the computer
- Python has a huge number of GUI frameworks (or toolkits) available for it
  - Most popular are wxPython, tkinter, and PyQT
- http://wiki.python.org/moin/GuiProgramming

tkinter References

- Best reference
  - http://infohost.nmt.edu/tcc/help/pubs/tkinter/web/index.html
- Official Python page on tkinter:
  - http://wiki.python.org/moin/Tkinter
  - http://effbot.org/tkinterbook/
  - http://www.tutorialspoint.com/python/python_gui_programming.htm
  - http://www.pythonware.com/library/tkinter/introduction/
- Installation (should be installed by default)
  - For Mac issues
  - https://www.python.org/download/mac/tcltk

Steps to Creating a GUI

1. Import tkinter (`from tkinter import *`)
2. Create top-level window for entire application
3. Build all widgets into application
4. Connect widgets to underlying functions
5. Enter main event loop

First GUI Application (tkinter)

- Simple GUI application that creates a window with "Hello World" as the title

```
# Simple GUI
# Demonstrates creating a window

from tkinter import *

# create the root window
root = Tk()

# modify the window
root.title("Hello World")
root.geometry("200x100")

# kick off the window's event-loop
root.mainloop()
```

First GUI Application (tkinter)

```
from tkinter import *

# create the root window
root = Tk()

# modify the window
root.title("Hello World")
root.geometry("200x100")

# kick off the window's event-loop
root.mainloop()
```

Widgets (GUI Elements)

Element	tkinter Class	Description
Frame	**`Frame`**	Holds other GUI elements
Label	**`Label`**	Displays uneditable text or icons
Button	**`Button`**	Performs an action when clicked
Text entry	**`Entry`**	Accepts / displays one line of text
Text box	**`Text`**	Accepts / displays multiple lines of text
Check button	**`Checkbutton`**	Allows the user to select or not select an option
Radio button	**`Radiobutton`**	Allows, as a group, the user to select one option from several

- Exercise 1

Frame Widget

- The container you can put *other* widgets in
  - Think "corkboard"
- When creating widgets, give argument for the **parent** widget
  - Called the **master**
- Always call **grid()** method to ensure widget is visible

```
root = Tk()
myFrame = Frame(root)
myFrame.grid()
```

Adding Widgets to a Frame

```
myFrame = Frame(root)
myFrame.grid()
```

- Create the widget (as a variable)

```
lbl1 = Label(myFrame, text="I'm a
  Label!")
```

- Call the grid method to make widget appear

```
lbl1.grid()
```

Widgets

- A type of class representing on-screen objects (e.g. buttons, text fields, etc.)
- Types of tkinter widgets:
  - Application windows
  - Buttons
  - Check boxes and radio buttons
  - Text boxes
  - Labels
  - and more...

Labels

```
Label(rootWindow, text="your text here")
```

- Creates a non-interactive piece of text
- Good for labels or other text that never changes

Buttons

```
Button(rootWindow, text="This is a button!",
                                  command=function)
```

- Makes a button that executes a specified function when pressed
- Buttons can have text labels
- By default, buttons use a standard appearance depending on operating system

User Input – Text Entry (single line)

- `Entry(rootWindow)`

- **`Entry()`** widgets can be used for output (display) as well as input
- **`Entry()`** may be of any length, but only one line

`Entry` methods

- Enabled / disabled typing in **`entry`**

```
.config(state = "normal")
.config(state = "disabled")
```

- Retrieve all text in **`entry`**

  `.get()` to retrieve text

`Entry` methods

- Delete text from **x**th to **y**th character

```
.delete(x, y)
```

- Delete all text (END is defined by tkinter)

```
.delete(0, END)
```

- Insert text at 0th character

```
.insert(0, your text here)
```

User Input – Text Field (multiple lines)

```
Text(rootWindow, width = x, height = y)
```

- Width and height are in *characters* and *lines*, not in pixels

`Text` methods

- Get text from **x**th *line*, **y**th *character* to **a**th *line*, **b**th *character*

```
.get(x.y, a.b)
```

- Get text from beginning to end

```
.get(0.0, END)
```

`Text` methods

- Delete text from beginning to end

```
.delete(0.0, END)
```

- Insert text at beginning

```
.insert(0.0, text to be inserted)
```

Common Properties

Property	Description	Example
`fg`	Sets the text color	`fg="red"`
`bg`	Sets the background color	`bg="#44ab3d"`
`activeforeground`	Sets text color when button is selected	`activeforeground="white"`
`activebackground`	Sets background color when button is selected	`activebackground="white"`
`height`	Specify height in **lines**	`height=5`
`width`	Specify width in **pixels**	`width=40`
`font`	Specify font	`font="times 16 bold"`, `font="verdana 10 italic"`
`anchor`	Position text within widget	`anchor=N`, `anchor=NW`, `anchor=SE`

Specifying Color as a String

- Color can be specified as a string or an RGB hex code

- Color strings are limited to "standard colors"
  - Ex. white, black, red, blue, etc.

```
fg="white"
```

Specifying Color in RGB

- Using hexadecimal code we can specify over 16 million distinct colors!

- To describe a color, we say how much **red**, how much **green**, and how much **blue** are in the color

- The amount of each **R**, **G**, **B** are measured on a scale of **0** *(min)* to **255** *(max)*

- In *hexadecimal, t*he amount of each **R**, **G**, **B** are measured on a scale of **00** *(min)* to **FF** *(max)*
  - *Hexadecimal* is a **base-16** number system instead of *decimal* which is **base-10**

Specifying Color in RGB

- Format

  #RRGGBB

- **RR**, **GG**, and **BB** are hexadecimal values **00-FF** specifying how much **red**, **green**, and **blue**

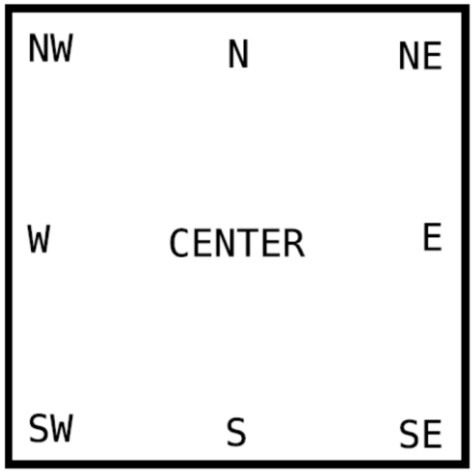
Common RGB Values

Hexadecimal Value	Color
#000000	**Black** (none)
#FFFFFF	**White** (red + green + blue)
#0000FF	**Blue**
#FF0000	**Red**
#00FF00	**Green**
#FF00FF	**Magenta** (red + blue)
#7F7F7F	**Medium Gray**
#CF640B	**Orange**

Anchor

- Used to position text horizontally and vertically

  anchor=NW



- Exercise 2

Creating an Application Class

- Rather than making a **`Frame`** object, create a your own class (**`Application`**)

- **`Application`** *inherits* all the properties and methods from **`Frame`**

- Inside the new **`__init__`**, you can add any necessary widgets as member attributes (**`self`**)

Example

```
class Application(Frame):
  def __init__(self, rootWindow):
    super().__init__(rootWindow)
    self.grid()

    self.lbl1 = Label(self, text="I'm a Label!")
    self.lbl1.grid()
```

Example, cont'd

```
root = Tk()
root.title("Welcome!")
root.geometry("200x200")

myApplication = Application(root)

root.mainloop()
```

- Exercise 3

Event-Driven Programming

- GUI programs are traditionally event-driven
  - The programs respond to actions regardless of the order in which they occur
  - The user determines when actions will occur
- When you write an event-driven program, you **bind** *(associate)* **events** *(things that can happen involving the program's objects)*

Event-Driven Programming

- By defining all of your objects, events, and event handlers, you establish how your program works
- You kick off the program by entering an **event loop**, where the program waits for events
  - When any of those events do occur, the program handles them

Event-Driven Programming

- What is an event?
  - Whenever a user takes an action on a component
- Examples
  - Clicks mouse pointer on a button
  - Enters text in a text box
  - Scrolls in a text box
  - Clicks on a menu

Imperative Programming

Start Program

```
import math
def main():
  num = 16
  result = 0

  result = math.sqrt(num)



  print("Sq Rt = ", result)
```

End Program

input: 16

Math.sqrt

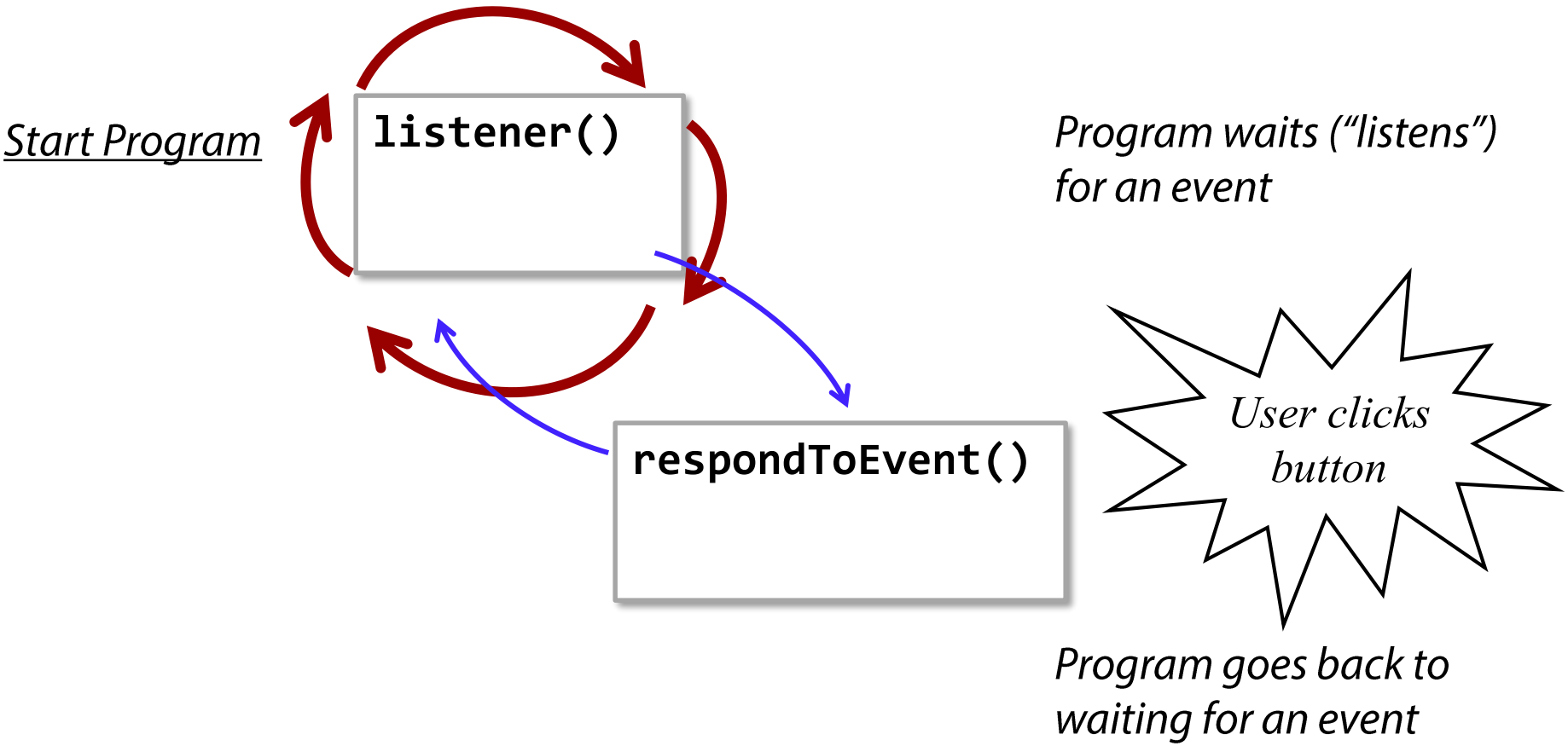
output: 4

input: **"Sq rt = 16"**

print

No return output

Event-Driven Programming



Steps Enable Button Actions

1. Create a method in your frame class
   – This will be triggered when the button is clicked

2. Connect the method to button with **`command=…`**

Example

```
class Application(Frame):
  def __init__(self, rootWindow):
    ...
    self.btn = Button(self, …, command=myAction)

  def myAction(self):
    #something awesome happens!
```

- Exercise 4

Message Boxes

Message box	Description
`showinfo(title, message)`	Shows information
`showerror(title, message)`	Shows error message
`showwarning(title, message)`	Shows warning
`askyesno(title, message)`	Asks yes/no question; returns `True` if **yes** selected, or `False` otherwise
`askokcancel(title, message)`	Asks **ok/cancel** question; returns `True` if **ok** selected, or `False` otherwise

Message Box Examples

```
from tkinter import *
from tkinter import messagebox

messagebox.showwarning("Invalid Move", "Beware")

messagebox.showinfo("Message", "Today is Thursday")

messagebox.showerror("Error", "You can't do that")
```

- Exercise 5

Grid

- All widgets are laid out in a grid format
- The grid is variably sized; rows and columns will fit themselves to their contents
- The **`grid()`** command sets an object's position; it also tells Tk to render that object

Grid Usage

```
newWidget = Button(self, text="Sample")
newWidget.grid(row = x, column = y, rowspan
   = 1, columnspan = 1, sticky = "")
```

- **row**: beginning row (vertical) coordinate
- **column**: beginning column (horizontal) coordinate

Grid Usage

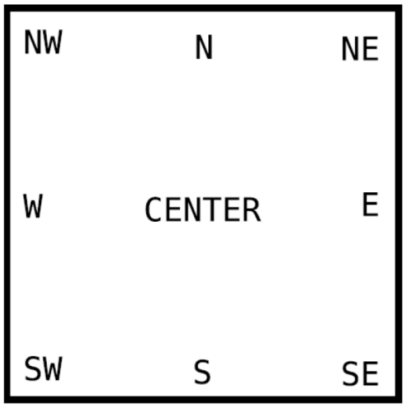
```
newWidget = Button(self, text="Sample")
newWidget.grid(row = x, column = y, rowspan
   = 1, columnspan = 1, sticky = "")
```

- **rowspan**: number of rows this widget occupies; vertical size
- **columnspan**: number of columns; horizontal size

Grid Usage

```
newWidget = Button(self, text="Sample")
newWidget.grid(row = x, column = y, rowspan
   = 1, columnspan = 1, sticky = "")
```

- **sticky**: how a widget is placed in its cell (e.g. if it is in a column with a wider object)



- Exercises 6 and 7

Checkbutton

- Check and uncheck options
- Not supposed to be mutually exclusive

Checkbutton

```
Checkbutton(self, text="…", variable=var)
```

- Can be checked (**`true`**) or unchecked (**`false`**)
- **`text`** is the text the user sees
- Automatically checks/unchecks the box according to **`variable`** value
- **`variable`** may be a **`BooleanVar`** or **`IntVar`** *(more in a moment)*

Radiobutton

- Select one options from multiple choices

- **Should** be mutually exclusive
  - "There can be only one!"

Radiobutton

```
Radiobutton(self, text="…", variable=var, value= "")
```

- **text** is the text the user sees
- **variable** is set to **value** when clicked on
- **variable** may be a **IntVar** or **StringVar**
- **variable** will automatically change the appearance of buttons
- Mutually exclusive **Radiobuttons** should have the same **variable**

Reading Values

- **`BooleanVar`**
  - Use with **`Checkbutton`**
- **`StringVar`**
  - Use with **`Radiobutton`** or **`Entry`**
- **`IntVar`**
  - Use with **`Checkbutton`** or **`Radiobutton`**

Special UI Variables

- Special variable classes that can be *associated* with input widgets (e.g. check boxes, sliders)
- Methods
  - `.get()` method to return the value
  - `.set()` to update it
- Widgets will automatically handle updating their control variables

Example #1: Radiobutton

```
class Application(Frame):
  def __init__(self, rootWindow):
    self.svYesNo = StringVar()
    self.svYesNo.set("y")
    self.radioYes = Radiobutton(self, text="Yes", value="y",
                                variable=self.svYesNo)
    self.radioNo = Radiobutton(self, text="No", value="n",
                                variable=self.svYesNo)
    self.btnShow = Button(self, text="Show", command=self.show)
    self.btnShow.grid(row=1, column=0)

  def show(self):
      messagebox.showinfo("Show", self.svYesNo.get())
```

Example #1: Radiobutton

```
class Application(Frame):
  def __init__(self, rootWindow):
    self.svYesNo = StringVar()
    self.svYesNo.set("y")
    self.radioYes = Radiobutton(self, text="Yes", value="y",
                                variable=self.svYesNo)
    self.radioNo = Radiobutton(self, text="No", value="n",
                                variable=self.svYesNo)
    self.btnShow = Button(self, text="Show", command=self.show)
    self.btnShow.grid(row=1, column=0)

  def show(self):
      messagebox.showinfo("Show", self.svYesNo.get())
```

Example #1: Radiobutton

```
class Application(Frame):
  def __init__(self, rootWindow):
    self.svYesNo = StringVar()
    self.svYesNo.set("y")
    self.radioYes = Radiobutton(self, text="Yes", value="y",
                                variable=self.svYesNo)
    self.radioNo = Radiobutton(self, text="No", value="n",
                                variable=self.svYesNo)
    self.btnShow = Button(self, text="Show", command=self.show)
    self.btnShow.grid(row=1, column=0)

  def show(self):
      messagebox.showinfo("Show", self.svYesNo.get())
```

Example #1: Radiobutton

```
class Application(Frame):
  def __init__(self, rootWindow):
    self.svYesNo = StringVar()
    self.svYesNo.set("y")
    self.radioYes = Radiobutton(self, text="Yes", value="y",
                                variable=self.svYesNo)
    self.radioYes = Radiobutton(self, text="No", value="n",
                                variable=self.svYesNo)
    self.btnShow = Button(self, text="Show", command=self.show)
    self.btnShow.grid(row=1, column=0)

  def show(self):
      messagebox.showinfo("Show", self.svYesNo.get())
```

Example #2: Checkbutton

```
class Application(Frame):
  def __init__(self, rootWindow):
    self.bvCheck = BooleanVar()
    self.checkTerms = Checkbutton(self, text="Accept?",
                                  variable=self.bvCheck)
    self.btnTest = Button(self, text="Show", command=self.test)
    self.btnTest.grid(row=1, column=0)

  def test(self):
    if self.bvCheck.get() == True:
      messagebox.showinfo("Show", "You checked!")
```

Example #2: Checkbutton

```
class Application(Frame):
  def __init__(self, rootWindow):
    self.bvCheck = BooleanVar()
    self.checkTerms = Checkbutton(self, text="Accept?",
                                  variable=self.bvCheck)
    self.btnTest = Button(self, text="Show", command=self.test)
    self.btnTest.grid(row=1, column=0)

  def test(self):
    if self.bvCheck.get() == True:
      messagebox.showinfo("Show", "You checked!")
```

Example #2: Checkbutton

```
class Application(Frame):
  def __init__(self, rootWindow):
    self.bvCheck = BooleanVar()
    self.checkTerms = Checkbutton(self, text="Accept?",
                                  variable=self.bvCheck)
    self.btnTest = Button(self, text="Show", command=self.test)
    self.btnTest.grid(row=1, column=0)

  def test(self):
    if self.bvCheck.get() == True:
      messagebox.showinfo("Show", "You checked!")
```

Example #2: Checkbutton

```
class Application(Frame):
  def __init__(self, rootWindow):
    self.bvCheck = BooleanVar()
    self.checkTerms = Checkbutton(self, text="Accept?",
                                  variable=self.bvCheck)
    self.btnTest = Button(self, text="Show", command=self.test)
    self.btnTest.grid(row=1, column=0)

  def test(self):
    if self.bvCheck.get() == True:
      messagebox.showinfo("Show", "You checked!")
```

- Exercise 8