WIDGETS AND DATA COLLECTIONS

CSSE 120—Rose Hulman Institute of Technology

Widgets

- "Widget" is a generic name for a component of a Graphical User interface(such as a button, window, menu, or textbox).
- Our simple Python graphics module only provides one kind of widget.
- We can create our own.
- Create a button widget.

Button Widget Class

- Button object:
  - Displayed as text inside a rectangle.
  - What are its operations (methods)?
  - What information does a Button object need to know about itself (instance variables)?
  - Before answering these questions, let's run a demo.

Constructor for Button class

```
class Button:
    """ A Button widget similar to the one in Zelle 10.6,
        but with the extra property of making sure the
        Button's text fits inside the rectangle """

    FONT_NAME = 'courier'
    FONT_SIZE = 20

    def __init__(self, win, label, center, width=0, height=0):
        """Initialize a Button with the given characteristics. """
        self.__setRect(width, height, center)
        self.__setText(label, center)
        self.enable()
        self.rect.draw(win)
        self.text.draw(win)
```

class variables (used as constants here)

put complex tasks into separate methods, to keep constructor simple

methods (like __setRect) whose names begin with two underscores (and do not also end with two underscores) are intended to be used as "private" helper methods, only called from other methods from the same class.

Private methods used by __init__()

```
def __setRect(self, width, height, center):
    """ Internal method.  Called by the constructor.
        Create the rectangular border of the button """
    centerX, centerY = center.getX(), center.getY()
    self.minX = centerX - width/2
    self.minY = centerY - height/2
    self.maxX = centerX + width/2
    self.maxY = centerY + height/2
    self.rect = Rectangle(Point(self.minX, self.minY),
                          Point(self.maxX, self.maxY))

def __setText(self, label, center):
    """ Internal method.  Called by the constructor.
        Create the Text object for the button's label"""
    self.text = Text(center, label)
    self.text.setFace(Button.FONT_NAME)
    self.text.setSize(Button.FONT_SIZE)
```

Note the use of the class variables

Build a Better Button?

- The problem
  - If width and/or height passed to the constructor are too small, the button's label may extend outside the rectangle that is supposed to bound it.
  - User has to rely on trial and error to get a reasonable button width.
- Solution
  - A better Button constructor that will create a "barely large enough" rectangle if the given width and height are too small.

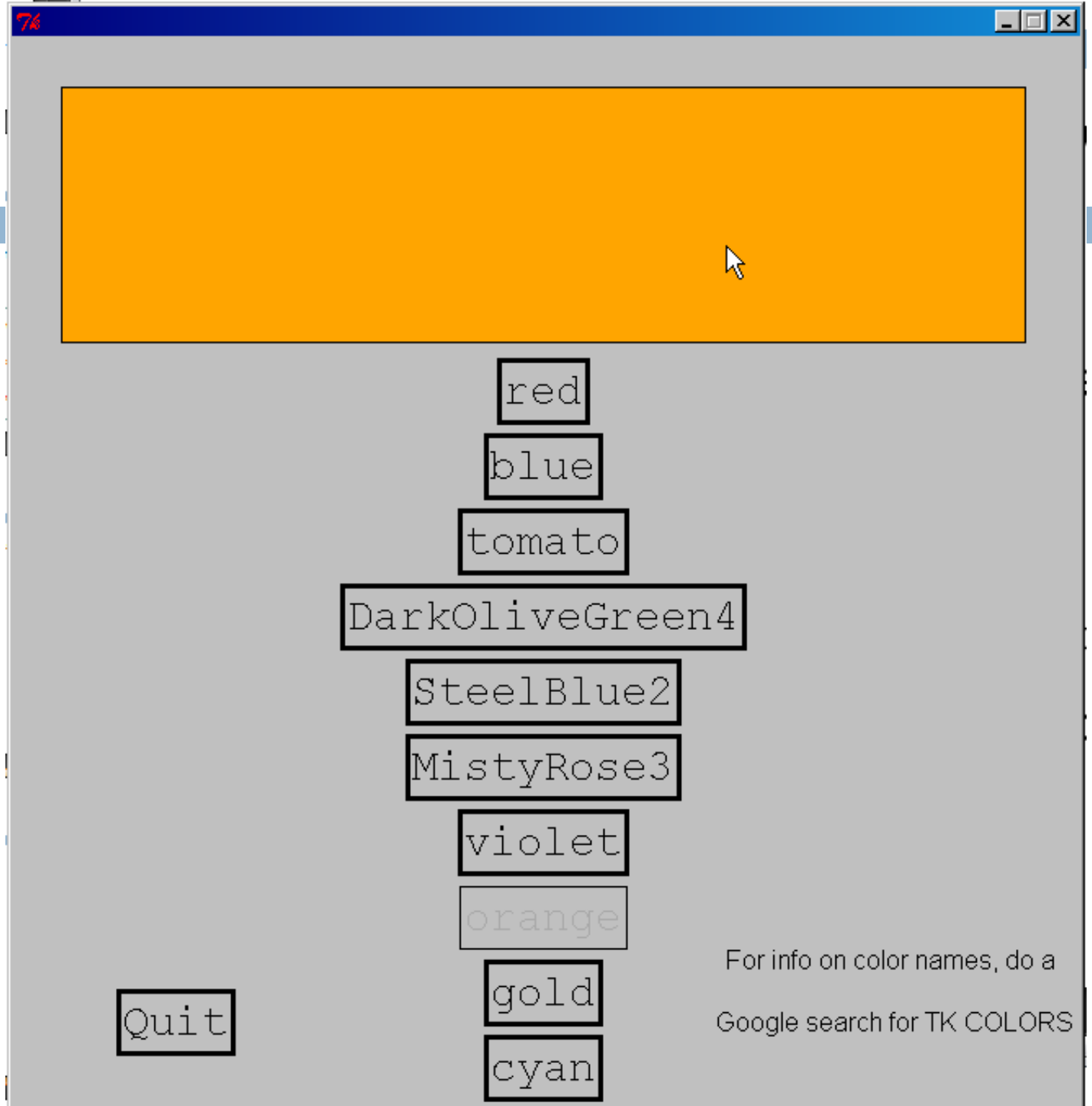
Constraints and research

- Fixed-width font (Courier). Why?
- Fixed font size (I chose 20 point).
- Experiments indicated that each character is about 16 pixels wide and about 34 pixels tall.
- Thus these class variables:

```
MIN_HEIGHT = 38  # A button should be at least this tall,
                 # in order to comfortably surround the text.
FONT_NAME = 'courier'
FONT_SIZE = 20
CHAR_WIDTH = 16     # The width of a character (in pixels) of
                    #   the chosen font face/size.
EXTRA_HORZ_SPACE = 6  # Put in this much extra horizontal
                        # space so that the label doesn't run
                        # into the end of the rectangle.
```

Enhance the Button Class

- From your SVN repository, checkout the ButtonWidget project.
- In Button.py, add code to guarantee that a button's rectangle will be large enough to enclose its text.
- Then go to ColorButtons.py.
  - Study the code that is there (especially the event loop) Add code to create the buttonList.
  - Add the **re-enable all** button and make it work.
- Commit the new version to your repository.


red
blue
tomato
DarkOliveGreen4
SteelBlue2
MistyRose3
violet
orange
gold
cyan
Quit
For info on color names, do a
Google search for TK COLORS

Pipe Dreams – by Animusic

- As you watch, think about how Objects could make the code for this easier to write.
- Each object (ball, string, xylophone key, etc.) knows its own physical characteristics, position, velocity, as well as how it reacts to striking or being struck by another object.
- There could be a loop that calls **timePasses()** for each object in the picture.
- Each object does what it would do in that time, and draws itself in its new position.

Some Basic List Operations

- Do the same thing to each object in a list
- find the largest number in a list of numbers.
- find the second largest element.
- Find the point in a list that is farthest away from a given point.
- Find the point in a list which, when chosen as the center, can enclose all of the points in the smallest circle

Same operation on all list objects

```
colorList = [color_rgb(r,0,255-r) for r in range (0,255,2)] + \
            [color_rgb(255-r,r,0) for r in range (0,255,2)] + \
            [color_rgb(r,255-r,r) for r in range (0,255,2)] + \
            [color_rgb(255,r,255-r) for r in range (0,255,2)]

def moveAllElementsBy(list, dx, dy):
    for obj in list:
        obj.move(dx, dy)

def colorAll(list, color):
    for obj in list:
        obj.setFill(color)

def moveThoseColors():
    win = GraphWin("", 950, 600)
    rectList = []
    for i in range(5):
        rect = Rectangle(Point(i*50, 10), Point(i*50+40, 50))
        rect.draw(win)
        rectList.append(rect)

    for c in colorList:
        time.sleep(.02)
        moveAllElementsBy(rectList, 1, 1)
        colorAll(rectList, c)

    time.sleep(1)
```

Watch the demo

Write the following functions

1.
```
def doubleAll(list):
    """ returns a list of numbers that are twice
        those in the original list. """
```

2.
```
def largestInList(numList):  # A nonempty list of numbers
    """ returns the largest number in the list. """
```

3.
```
def secondLargest(numList):
    # numList contains at least 2 numbers, all different
    """ returns the second largest number in the list """
```

4.
```
def farthestPoint(pointList, p):
    """return the point in pointList that is
      farthest from point p """
```