Last Time

• Library graphics.py
  – built on top of Tkinter (Python interface to Tk)
  – built on top of Tk (graphics library)
  – built on top of X11 (graphics library) on Ubuntu
class GraphWin(tk.Canvas):

    """A GraphWin is a toplevel window for displaying graphics."""
    def __init__(self, title="Graphics Window",
                 width=200, height=200, autoflush=True):
        master = tk.Toplevel(_root)
        master.protocol("WM_DELETE_WINDOW", self.close)
        tk.Canvas.__init__(self, master, width=width, height=height)
        self.master.title(title)
        self.pack()
        master.resizable(0,0)
        self.foreground = "black"
        self.items = []
Object inheritance (see graphics.py code):
- GraphicsObject
  - Point: draw, move, clone
  - Text: clone, move, draw, setText, getText, setFace
  - Polygon: clone, getPoints, move, draw
  - _Bbox: move
    - Rectangle: draw
    - Oval: clone, draw
      - Circle: clone, getRadius
    - Line: clone, draw, setArrow
Directly in wish tcl/tk

Stylistic variation...

- Use primitive object `Point` in `graphics.py`
- `tk` uses plain x,y coordinates
Interactive Graphics

• textbook 5.7:
  \[
  p = \text{win.getMouse}() \quad \text{returns a Point}
  \]
Interactive Graphics

# Program: triangle.pyw
# Description: Interactive graphics program to draw a triangle

from graphics import *

def main():
    win = GraphWin("Draw a Triangle")
    win.setCoords(0.0, 0.0, 10.0, 10.0)
    message = Text(Point(5, 0.5), "Click on three points")
    message.draw(win)

    # Get and draw three vertices of triangle
    p1 = win.getMouse()
    p1.draw(win)
    p2 = win.getMouse()
    p2.draw(win)
    p3 = win.getMouse()
    p3.draw(win)
Interactive Graphics

```python
# Use Polygon object to draw the triangle
triangle = Polygon(p1,p2,p3)
triangle.setFill("peachpuff")
triangle.setOutline("cyan")
triangle.draw(win)

# Wait for another click to exit
message.setText("Click anywhere to quit.")
win.getMouse()
win.close()

main()
```
Interactive Graphics

- textbook 5.7:
  - `input = Entry(center, width)` creates a user-editable text box
from graphics import *

def main():
    win = GraphWin("Celsius Converter", 400, 300)
    win.setCoords(0.0, 0.0, 3.0, 4.0)

    # Draw the interface
    Text(Point(1,3), "  Celsius Temperature:" ).draw(win)
    Text(Point(1,1), "Fahrenheit Temperature:" ).draw(win)
    input = Entry(Point(2,3), 5)
    input.setText("0.0")
    input.draw(win)
    output = Text(Point(2,1),"")
    output.draw(win)
    button = Text(Point(1.5,2.0),"Convert It")
    button.draw(win)
    Rectangle(Point(1,1.5), Point(2,2.5)).draw(win)
Interactive Graphics

```python
# wait for a mouse click
win.getMouse()

# convert input
celsius = eval(input.getText())
fahrenheit = 9.0/5.0 * celsius + 32

# display output and change button
output.setText("%.1f" % fahrenheit)
button.setText("Quit")

# wait for click and then quit
win.getMouse()
win.close()

main()
```
graphics.py API

5.8.1 GraphWin Objects

A GraphWin object represents a window on the screen where graphical images may be drawn. A program may define any number of GraphWins. A GraphWin understands the following methods:

GraphWin(title, width, height) Constructs a new graphics window for drawing on the screen. The parameters are optional, the default title is “Graphics Window,” and the default size is 200 x 200.

plot(x, y, color) Draws the pixel at (x, y) in the window. Color is optional, black is the default.
graphics.py API

plotPixel(x, y, Color) Draws the pixel at the “raw” position \((x, y)\) ignoring any coordinate transformations set up by setCoords.

setBackground(color) Sets the window background to the given color. The initial background is gray. See Section 5.8.5 for information on specifying colors.

close() Closes the on-screen window.

getMouse() Pauses for the user to click a mouse in the window and returns where the mouse was clicked as a Point object.

setCoords(xll, yll, xur, yur) Sets the coordinate system of the window. The lower left corner is \((xll, yll)\) and the upper right corner is \((xur, yur)\). All subsequent drawing will be done with respect to the altered coordinate system (except for plotPixel).
5.8.2 Graphics Objects

The module provides the following classes of drawable objects: Point, Line, Circle, Oval, Rectangle, Polygon, and Text. All objects are initially created unfilled with a black outline. All graphics objects support the following generic set of methods:

`setFill(color)` Sets the interior of the object to the given color.

`setOutline(color)` Sets the outline of the object to the given color.

`setWidth(pixels)` Sets the width of the outline of the object to this many pixels. (Does not work for Point.)

`draw(aGraphWin)` Draws the object into the given GraphWin.

`undraw()` Undraws the object from a graphics window.

`move(dx,dy)` Moves the object dx units in the \( x \) direction and dy units in the \( y \) direction. If the object is currently drawn, the image is adjusted to the new position.

`clone()` Returns a duplicate of the object. Clones are always created in an undrawn state. Other than that, they are identical to the cloned object.
The graphics module also provides a function for mixing your own colors numerically. The function `color.rgb(red, green, blue)` will return a string representing a color that is a mixture of the intensities of red, green and blue specified. These should be ints in the range 0–255. Thus `color.rgb(255, 0, 0)` is a bright red, while `color.rgb(130, 0, 130)` is a medium magenta.

- see [http://wiki.tcl.tk/37701](http://wiki.tcl.tk/37701)
Point Methods

Point(x, y) Constructs a point having the given coordinates.

getX() Returns the x coordinate of a point.

getY() Returns the y coordinate of a point.

Line Methods

Line(point1, point2) Constructs a line segment from point1 to point2.

setArrow(string) Sets the arrowhead status of a line. Arrows may be drawn at either the first point, the last point, or both. Possible values of string are 'first', 'last', 'both', and 'none'. The default setting is 'none'.

getCenter() Returns a clone of the midpoint of the line segment.

getP1(), getP2() Returns a clone of the corresponding endpoint of the segment.
graphics.py API

Circle Methods

Circle(centerPoint, radius) Constructs a circle with given center point and radius.

getCenter() Returns a clone of the center point of the circle.

getRadius() Returns the radius of the circle.

getP1(), getP2() Returns a clone of the corresponding corner of the circle's bounding box. These are opposite corner points of a square that circumscribes the circle.

Rectangle Methods

Rectangle(point1, point2) Constructs a rectangle having opposite corners at point1 and point2.

getCenter() Returns a clone of the center point of the rectangle.

getP1(), getP2() Returns a clone of corner points originally used to construct the rectangle.
graphics.py API

Oval Methods

Oval(point1, point2) Constructs an oval in the bounding box determined by point1 and point2.

gCenter() Returns a clone of the point at the center of the oval.

getP1(), getP2() Returns a clone of the corresponding point used to construct the oval.

Polygon Methods

Polygon(point1, point2, point3, ...) Constructs a polygon having the given points as vertices. Also accepts a single parameter that is a list of the vertices.

gPoints() Returns a list containing clones of the points used to construct the polygon.
**graphics.py API**

**Text Methods**

`Text(anchopPoint, string)` Constructs a text object that displays the given string centered at anchorPoint. The text is displayed horizontally.

`setText(string)` Sets the text of the object to string.

`getText()` Returns the current string.

`getAnchor()` Returns a clone of the anchor point.

`setFace(family)` Changes the font face to the given family. Possible values are: 'helvetica', 'courier', 'times roman', and 'arial'.

`setSize(point)` Changes the font size to the given point size. Sizes from 5 to 36 points are legal.

`setStyle(style)` Changes font to the given style. Possible values are 'normal', 'bold', 'italic', and 'bold italic'.

`setTextColor(color)` Sets the color of the text to color. Note: `setFill` has the same effect.
5.8.3 Entry Objects

Objects of type Entry are displayed as text entry boxes that can be edited by the user of the program. Entry objects support the generic graphics methods move(), draw(graphwin), undraw(), setFill(color), and clone(). The Entry specific methods are given below.

Entry(centerPoint, width) Constructs an Entry having the given center point and width. The width is specified in number of characters of text that can be displayed.

getAnchor() Returns a clone of the point where the entry box is centered.

getText() Returns the string of text that is currently in the entry box.

setText(string) Sets the text in the entry box to the given string. Changes the font face to the given family. Possible values are: 'helvetica', 'courier', 'times roman', and 'arial'.

setSize(point) Changes the font size to the given point size. Sizes from 5 to 36 points are legal.

setStyle(style) Changes font to the given style. Possible values are: 'normal', 'bold', 'italic', and 'bold italic'.

setTextColor(color) Sets the color of the text to color.
Section 6.2-6.4

• Happy birthday example

```python
def sing(person):
    happy()
    happy()
    print "Happy Birthday, dear", person
    happy()
```

• nested function calls:

**Note**: parameters are passed by value (copying)
- not by reference (pointer)
- (some other programming languages allow both options)
Section 6.5

• To send a value back, use
  – return <Expression>

• Multiple values can be returned:

```python
def sumDiff(x, y):
    sum = x + y
    diff = x - y
    return sum, diff
```

```
num1, num2 = input("Please enter two numbers (num1, num2) ")
s, d = sumDiff(num1, num2)
print "The sum is", s, "and the difference is", d
```

**Note**: None is returned when there is no return statement
triangle2.py

• Calculates the perimeter of a triangle you draw.

![Image of a triangle with perimeter calculation]
triangle2.py

```python
import math
from graphics import *

def square(x):
    return x * x

def distance(p1, p2):
    dist = math.sqrt(square(p2.getX() - p1.getX())
                     + square(p2.getY() - p1.getY()))
    return dist
```

**Note:** parameters are passed by value (copying)
- mutable objects (Point)
```python
# Calculate the perimeter of the triangle
perim = distance(p1,p2) + distance(p2,p3) + distance(p3,p1)
message.setText("The perimeter is: %.2f" % perim)

# Wait for another click to exit
win.getMouse()
win.close()

main()
```