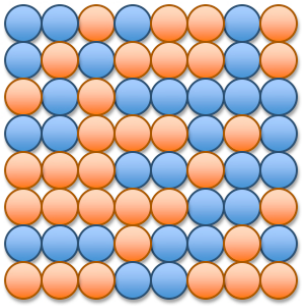


UNIT 11A
Visualizing Data: Graphics in Python

Drawing using PythonLabs

- The `PythonLabs.Canvas` is based on Python's interface to Tcl/Tk, a cross-platform graphics library.
  - To use this, you should be logged in directly into the Andrew machines or logged in remotely (using ssh) with an X client running (we have installed a pre-release version of PythonLabs on Andrew machines)
  - Start with:

```
from PythonLabs.Canvas import Canvas
```

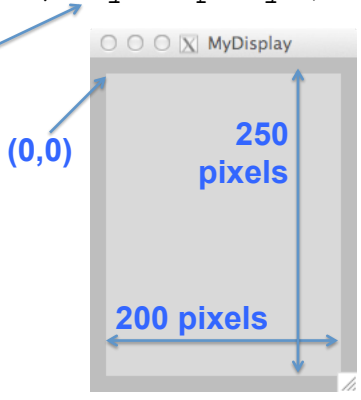
Coordinate System

- When drawing on a canvas, the location of the origin is at the TOP LEFT, not bottom left.
  - x increases left to right
  - y increases top to bottom
- Coordinates are based on PIXELS, not other units like inches or millimeters.

Coordinate System

```
>>> window = Canvas()
>>> window.init(200,250,"MyDisplay")
```

Do not name your window using spaces! Note the example above!



Drawing Rectangles

`Rectangle(x0, y0, x1, y1, optional_params)`

- Draw a rectangle from top left (x_0,y_0) to bottom right (x_1,y_1) in units of pixels.
- Optional parameters:
  fill="*color*" *(default: none)*
  outline="*color*" *(default: "BLACK")*
  color can also be specified in hex as "#RRGGBB"
  width=*numpixels* *(default: 1)*

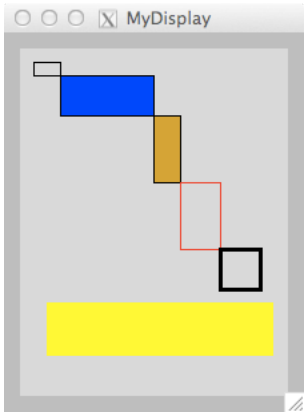
Available Color Names

http://www.cs.cmu.edu/~tcortina/15110m14/colorchart.png

Drawing Rectangles

```
>>> window = Canvas()
>>> window.init(200,250,"MyDisplay")
>>> window.Rectangle(10,10,30,20)
>>> window.Rectangle(30,20,100,50,fill="BLUE")
>>> window.Rectangle(100,50,120,100,fill="#DAA520")
>>> window.Rectangle(120,100,150,150,outline="RED")
>>> window.Rectangle(150,150,180,180,width=3)
>>> window.Rectangle(20,190,190,230,fill="#FFFF00",width=0)
```


Drawing Rectangles



Circles

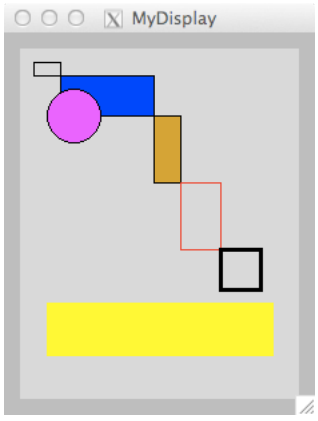
`Circle(x0, y0, radius, optional_params)`

- Draw a Circle with center at coordinate (x_0, y_0) and the given radius in pixels.

```
>>> window.Circle(40, 50, 20, fill="#FF00FF")
```

Note how the window acts like a painter's canvas.

Circles



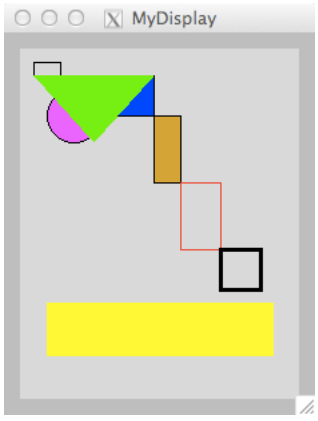
Polygons

`Polygon(point_list, optional_params)`

- Draw a Polygon with vertices taken from the list of points as follows:
  $[x_0, y_0, x_1, y_1, \ldots x_{n-1}, y_{n-1}]$.

```
>>> window.Polygon([10, 20, 100, 20,
55, 70], fill="GREEN")
```


Polygons



Closing the Canvas

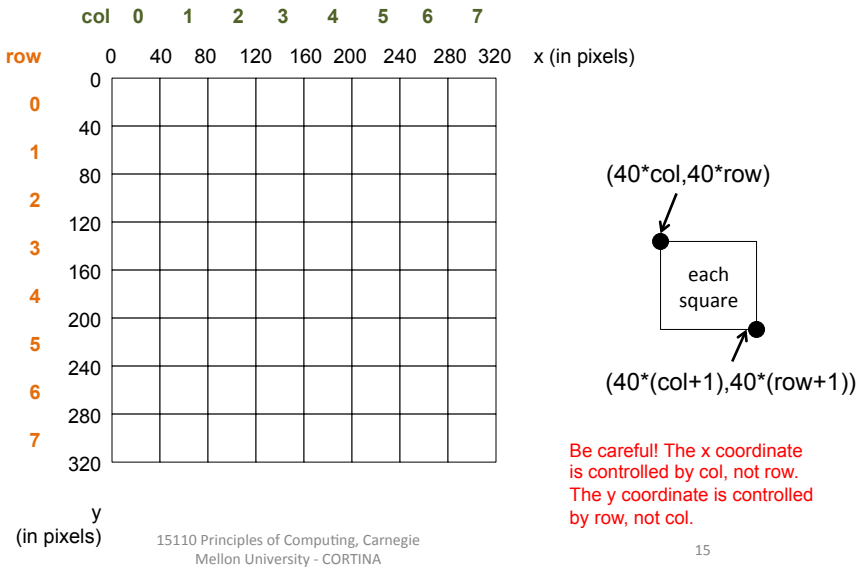
- Don't click the X (red) button to delete the window.
- Instead, when you're done, you can do this:
  `>>> window.close()`
- The Canvas is automatically deleted when Python3 exits, so you should run Python3 with the -i (interactive) switch to prevent Python3 from exiting until you have seen your graphical output.

Example

In `graphicsdemo.py`, write a Python function `demo()` that creates a window of size 360 by 360 and draws a grid of 8 by 8 squares, each of size 40 by 40 pixels, and colored a random color of red, green or blue for each square.

The random number generator is seeded with the number 15110 to generate the same sequence of pseudorandom numbers each time the function is called.

Coordinates for the Squares



Be careful! The x coordinate is controlled by col, not row. The y coordinate is controlled by row, not col.

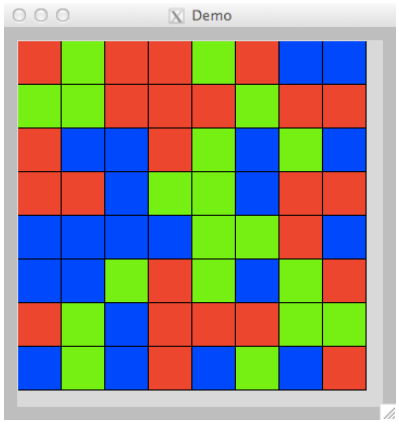
Programming Example

```
from PythonLabs.Canvas import Canvas
from random import randint, seed

def demo():
    Canvas.init(360,360,"Demo")
    colors = ["red", "green", "blue"]
    seed(15110)
    for row in range(0,8):
        for col in range(0,8):
            randcolor = colors[randint(0,2)]
            Canvas.Rectangle(40*col, 40*row,
              40*(col+1), 40*(row+1), fill=randcolor)
    return None
```

Results

```
python3 -i graphicsdemo.py
>>> demo()
>>>
```



Note that the size of the window is about 40 pixels wider than 8*40, due to an implementation issue in PythonLabs.