Python Object-Oriented Program with Libraries

Unit 4: PyGame Tutorial

CHAPTER 3: DRAWING

DR. ERIC CHOU

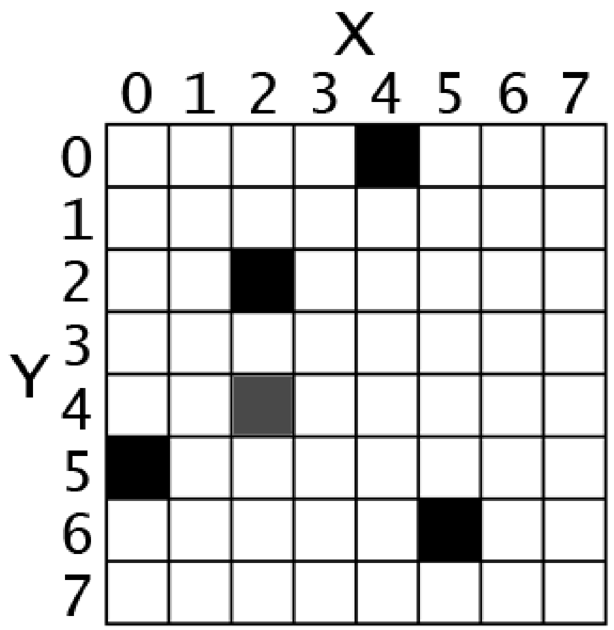
IEEE SENIOR MEMBER

PyGame Coordinates

LECTURE 1

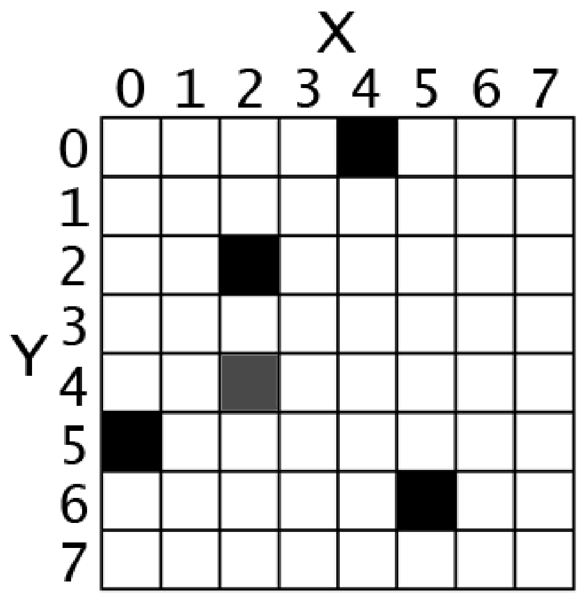
Pixel Coordinates

- The window that the “Hello World” program created was just composed of little square dots on your screen called **pixel**s.
- Each pixel starts off as black but can be set to a different color. Imagine that instead of a Surface object that is 400 pixels wide and 300 pixels tall, we just had a Surface object that was 8 pixels by 8 pixels.
- If that tiny 8x8 Surface was enlarged so that each pixel looks like a square in a grid, then a good representation of it could look something like this:



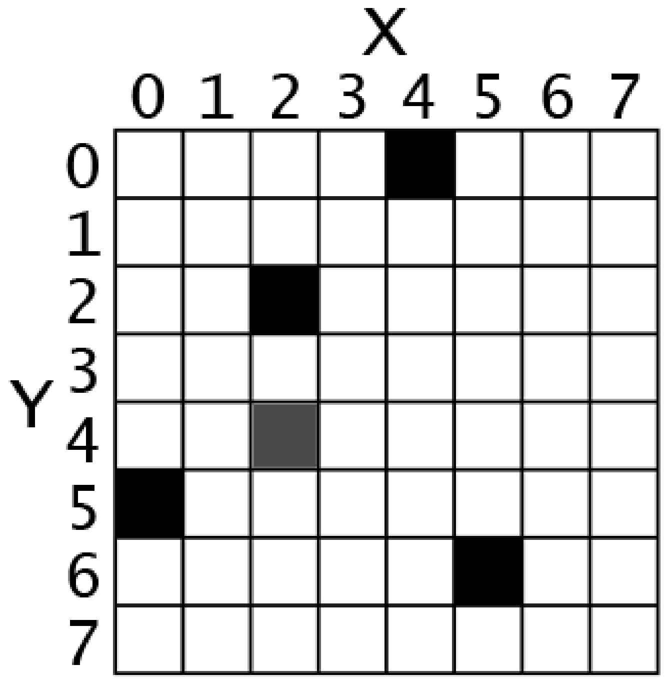
Pixel Coordinates

- If you’ve taken a math class and learned about Cartesian Coordinates, you might notice that the Y-axis starts at 0 at the top and then increases going down, rather than increasing as it goes up. This is just how Cartesian Coordinates work in Pygame (and almost every programming language).
- Each column of the X-axis and each row of the Y-axis will have an “address” that is an integer from 0 to 7 so that we can locate any pixel by specifying the X and Y axis integers.



Pixel Coordinates

- For example, in the above 8x8 image, we can see that the pixels at the XY coordinates (4, 0), (2, 2), (0, 5), and (5, 6) have been painted black, the pixel at (2, 4) has been painted gray, while all the other pixels are painted white. XY coordinates are also called points.



Functions Vs. Methods

```
import whammy
fizzy()                  // function call
egg = Wombat()           // Constructor  function -returns an object
egg.blahblah()           // method call (associated w/ object)
whammy.spam()            // function – see import
```

PyGame Surface(Canvas)

LECTURE 2

A Bit about Surface Objects

- Surface objects are objects that represent a rectangular **2D image**.
- The pixels of the Surface object can be changed by calling the PyGame drawing functions (more on these later…) and then displayed on the screen.
- The window border, title bar, and buttons are not part of the display Surface object.

A Bit about Surface Objects

- The Surface object returned by **pygame.display.set_mode()** is called the **display Surface.**
- Anything that is drawn on the display Surface object will be displayed on the window when the **pygame.display.update()** function is called.
- It is a lot faster to draw on a Surface object (which only exists in the computer's memory) than it is to draw a Surface object to the computer screen. Computer memory is much faster to change than pixels on a monitor.

A Bit about Surface Objects

- Often your program will draw several different things to a Surface object.
- Once you are done drawing everything on the display Surface object for this iteration of the game loop (called a **frame**), it can be drawn to the screen.
- The computer can draw frames very quickly, and our programs will often run around 30 frames per second (that is, **30 FPS**). This is called the "frame rate"

PyGame Colors

LECTURE 2

Colors

How is the Color of a PIXEL recorded?

- It is represented using the RGB (**Red**, **Green**, **Blue**) color model.
- In Python, we use tuples of 3 integers to represent the values of **red**, **green**, **and** **blue** each range from 0 to 255.

256 x 256 x 256 =
16,777,216 COLORS!
Error if <0 or >255

Color PIXELS

- Combining color is not the same as mixing paint to make a color.
- You can make **yellow** by combining **red** and **green**!
- The computer uses light to display color, not paint.

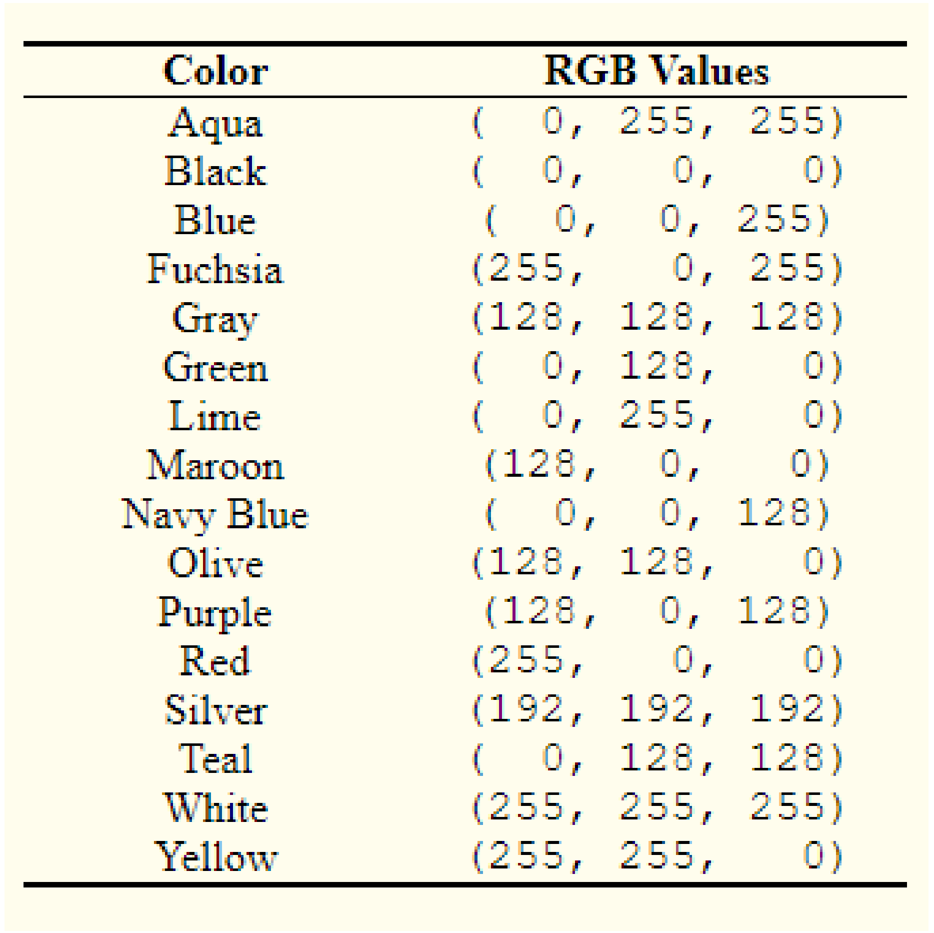
PyGame Colors

- There are three primary colors of light: **red**, **green** and **blue**.
- By combining different amounts of these three colors you can form any other color. In Pygame, we represent colors with tuples of three integers.
  - The first value in the tuple is how much red is in the color. An integer value of 0 means there is no red in this color, and a value of 255 means there is the maximum amount of red in the color.
  - The second value is for green and the third value is for blue.
  - These tuples of three integers used to represent a color are often called RGB values.
- Because you can use any combination of 0 to 255 for each of the three primary colors, this means Pygame can draw **16,777,216** different colors (that is, 256 x 256 x 256 colors).
- However, if try to use a number larger than 255 or a negative number, you will get an error that looks like “ValueError: invalid color argument”.

PyGame Colors

- For example, we will create the tuple (0, 0, 0) and store it in a variable named BLACK. With no amount of red, green, or blue, the resulting color is completely black.
- The color black is the absence of any color. The tuple (255, 255, 255) for a maximum amount of red, green, and blue to result in white.
- The color white is the full combination of red, green, and blue. The tuple (255, 0, 0) represents the maximum amount of red but no amount of green and blue, so the resulting color is red. Similarly, (0, 255, 0) is green and (0, 0, 255) is blue.

Color PIXELS



Color	RGB Values
Aqua	(0, 255, 255)
Black	(0, 0, 0)
Blue	(0, 0, 255)
Fuchsia	(255, 0, 255)
Gray	(128, 128, 128)
Green	(0, 128, 0)
Lime	(0, 255, 0)
Maroon	(128, 0, 0)
Navy Blue	(0, 0, 128)
Olive	(128, 128, 0)
Purple	(128, 0, 128)
Red	(255, 0, 0)
Silver	(192, 192, 192)
Teal	(0, 128, 128)
White	(255, 255, 255)
Yellow	(255, 255, 0)

- The colors in your CSS system.
  https://trinket.io/docs/colors
  https://www.w3schools.com/colors/colors_picker.asp

- ColorPicker.py, ColorChooser
  - pygame_color.py
  - pygame_color_simple.py

Transparent Colors

- This value is known as the alpha value.
- It is a measure of how opaque (that is, not transparent) a color is.
- For example, this tuple of three integers is for the color green: (0, 255, 0). But if we add a fourth integer for the alpha value, we can make this a half transparent green color: (0, 255, 0, 128). An alpha value of 255 is completely opaque (that is, not transparency at all). The colors (0, 255, 0) and (0, 255, 0, 255) look exactly the same.
- An alpha value of 0 means the color is completely transparent. If you draw any color that has an alpha value of 0 to a surface object, it will have no effect, because this color is completely transparent and invisible.

Color - Opacity

- The RGB model sometimes includes an alpha value as well that indicates how transparent or opaque the color is. A color that is transparent will let you see some of the color beneath it.
- Think OPACITY in Alice
- Alpha value of 0 - completely transparent
- Alpha value of 255 – not transparent

Color – Opacity

Demo Program: happy.py/happy1.py

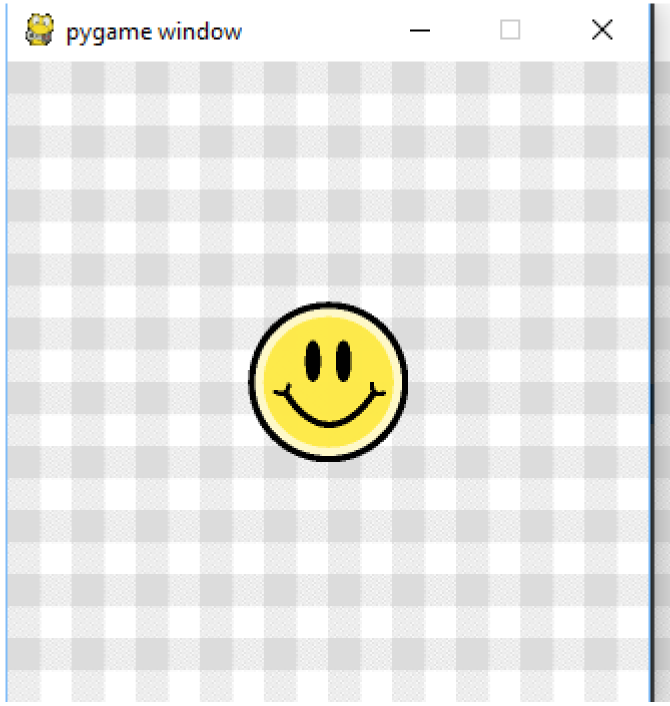
- In order to draw using transparent colors, you must create a Surface object with the **convert_alpha()** method. For example, the following code creates a Surface object that transparent colors can be drawn on:

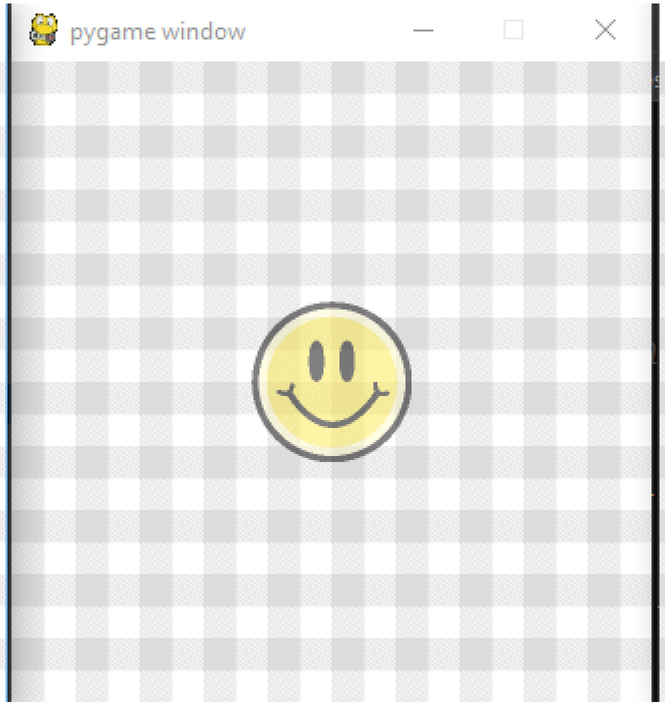
```
anotherSurface = DISPLAYSURF.convert_alpha()
```

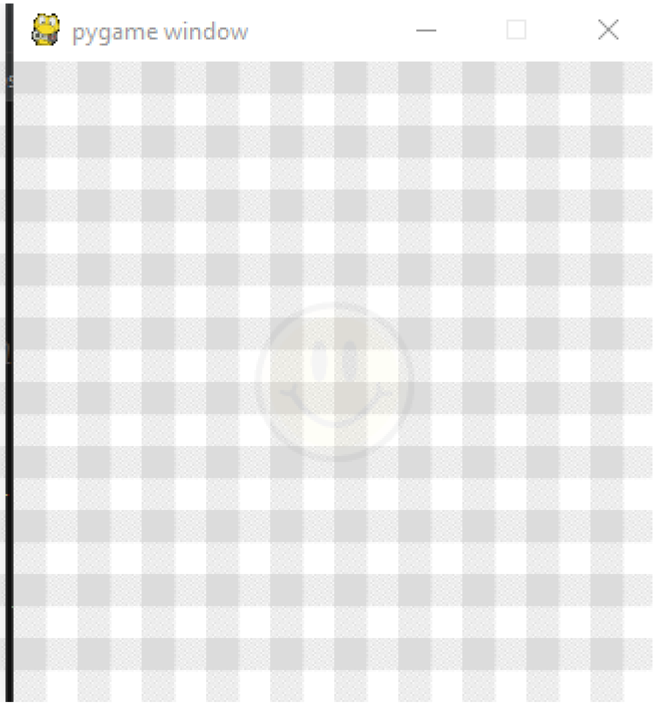
```
import pygame                                                        # happy1.py
import time

def blit_alpha(target, source, location, opacity):
    x = location[0]
    y = location[1]
    temp = pygame.Surface((source.get_width(), source.get_height())).convert()
    temp.blit(target, (-x, -y))
    temp.blit(source, (0, 0))
    temp.set_alpha(opacity)
    target.blit(temp, location)

pygame.init()
screen = pygame.display.set_mode((320, 320))
done = False

happy = pygame.image.load('happy80.png')   # our happy blue protagonist
checkers = pygame.image.load('background32.png')   # 32x32 repeating checkered image
```

```
while not done:                                          # happy1.py (part 2)
    start = time.time()
    for e in pygame.event.get():
        if e.type == pygame.QUIT:
            done = True
    # checker the background
    x = 0
    while x < 300:
        y = 0
        while y < 300:
            screen.blit(checkers, (x, y))
            y += 32
        x += 32
    # here comes the protagonist
    blit_alpha(screen, happy, (120, 120), 10)
    pygame.display.flip()
    end = time.time()
    diff = end - start
    framerate = 30
    delay = 1.0 / framerate - diff
    if delay > 0:
        time.sleep(delay)
```

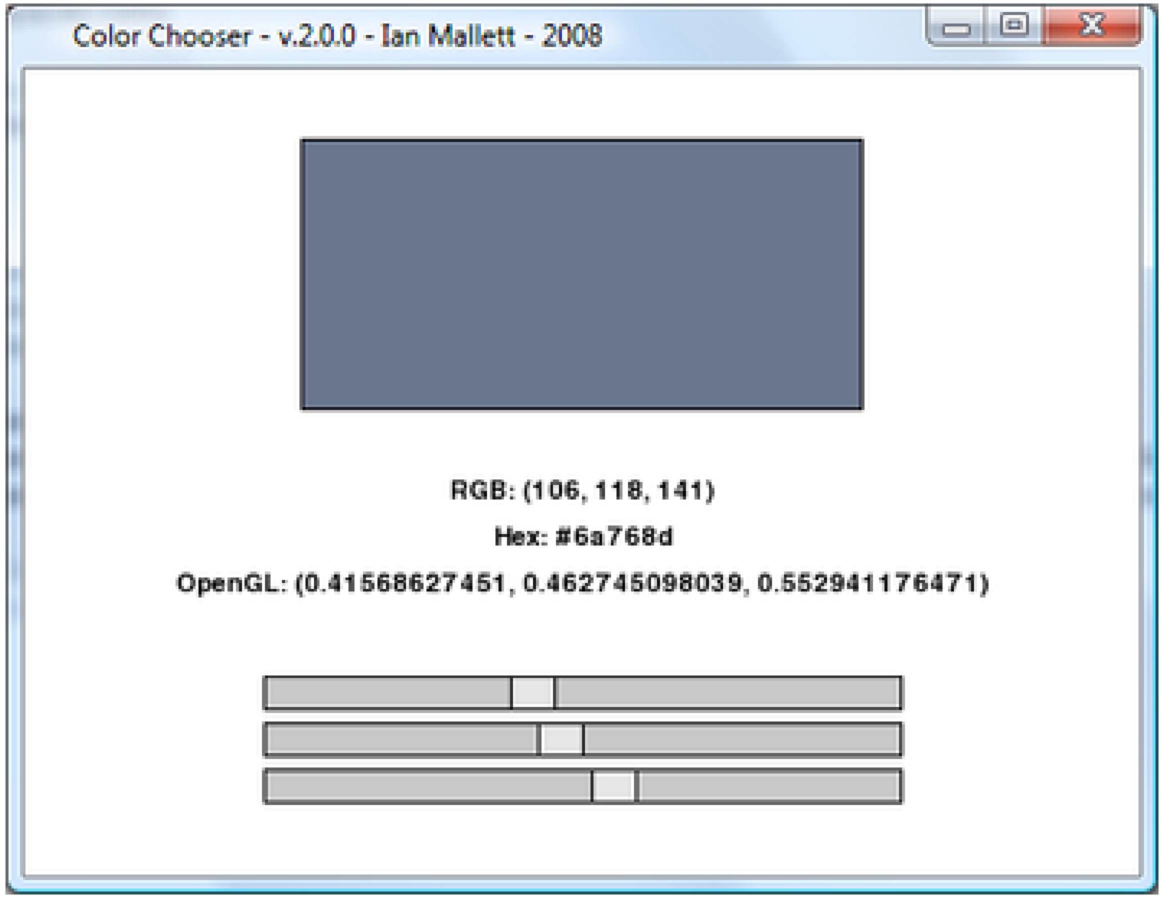
Different Opacity (255, 128, 10)







Color Chooser


Color Chooser - v.2.0.0 - Ian Mallett - 2008

RGB: (106, 118, 141)

Hex: #6a768d

OpenGL: (0.41568627451, 0.462745098039, 0.552941176471)

Demo Program: ColorChooser0.py

Go PyCharm!!!

PyGame Objects

LECTURE 2

Rectangle Objects

Also works as Bounding Box

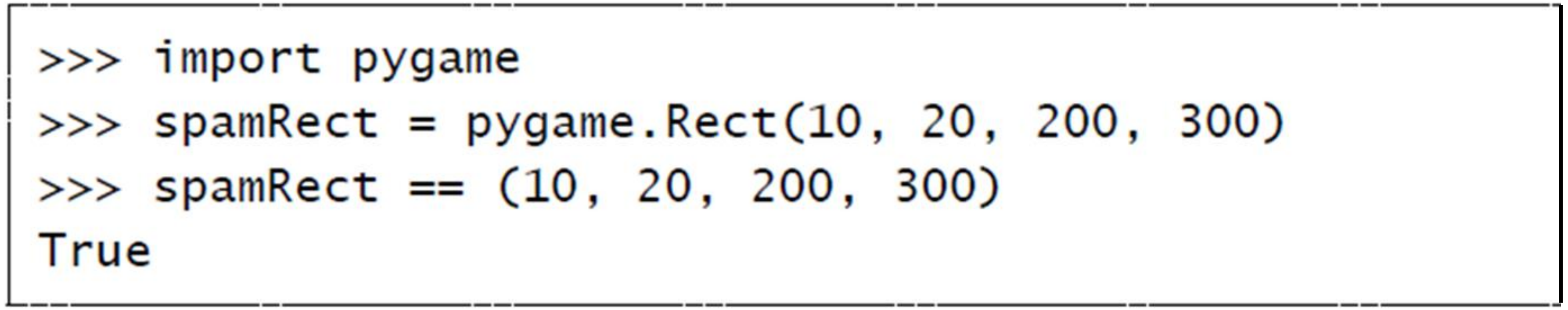
Pygame has two ways to represent rectangular areas (just like there are two ways to represent colors).

The first is a tuple of four integers:

1. **The X coordinate of the top left corner.**
2. **The Y coordinate of the top left corner.**
3. **The width (in pixels) of the rectangle.**
4. **Then height (in pixels) of the rectangle.**

Rectangle Objects

The second way is as a **pygame.Rect** object, which we will call Rect objects for short. For example, the code below creates a Rect object with a top left corner at (10, 20) that is 200 pixels wide and 300 pixels tall:

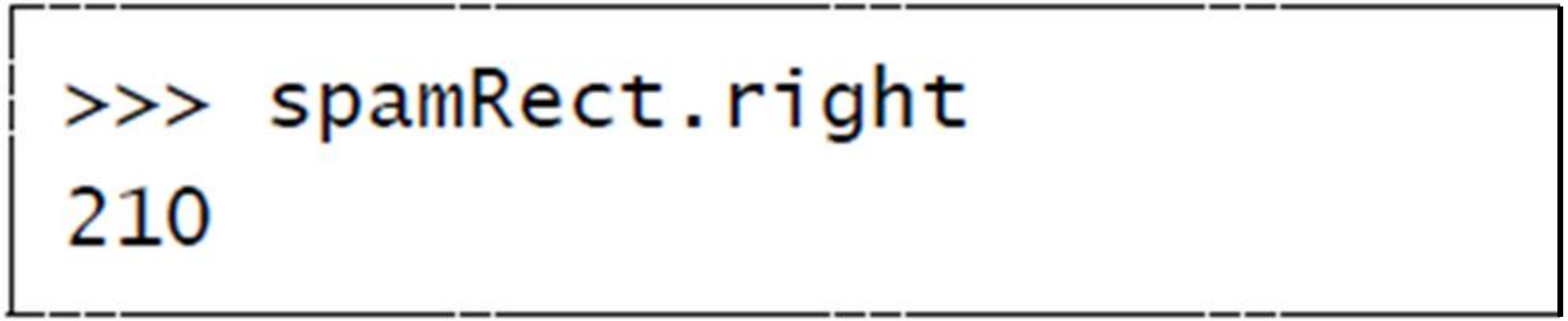


```
>>> import pygame
>>> spamRect = pygame.Rect(10, 20, 200, 300)
>>> spamRect == (10, 20, 200, 300)
True
```

Rectangle Objects

The handy thing about this is that the **Rect** object automatically calculates the coordinates for other features of the rectangle.

For example, if you need to know the X coordinate of the right edge of the **pygame.Rect** object we stored in the **spamRect** variable, you can just access the **Rect** object's right attribute:

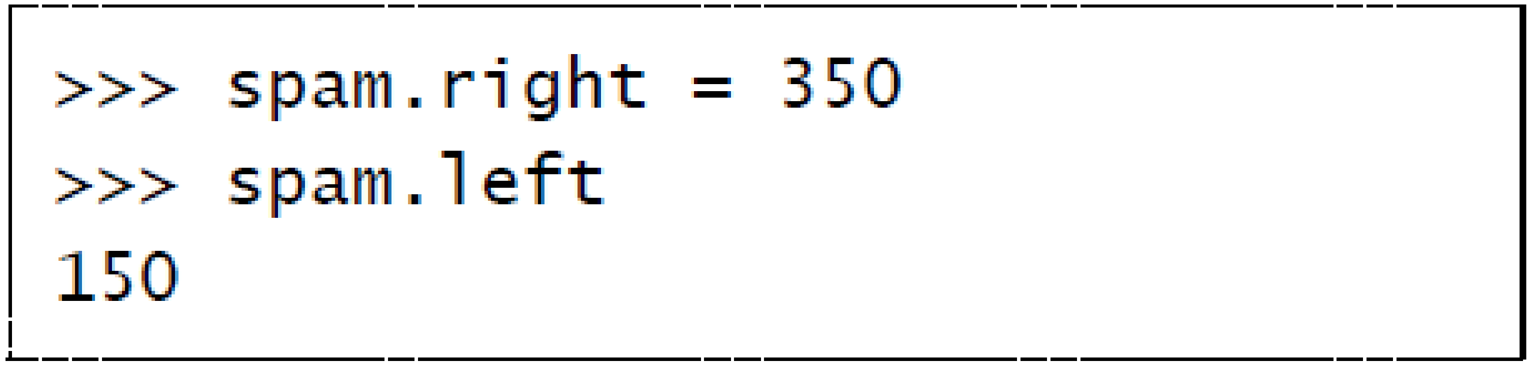


```
>>> spamRect.right
210
```

Rectangle Objects

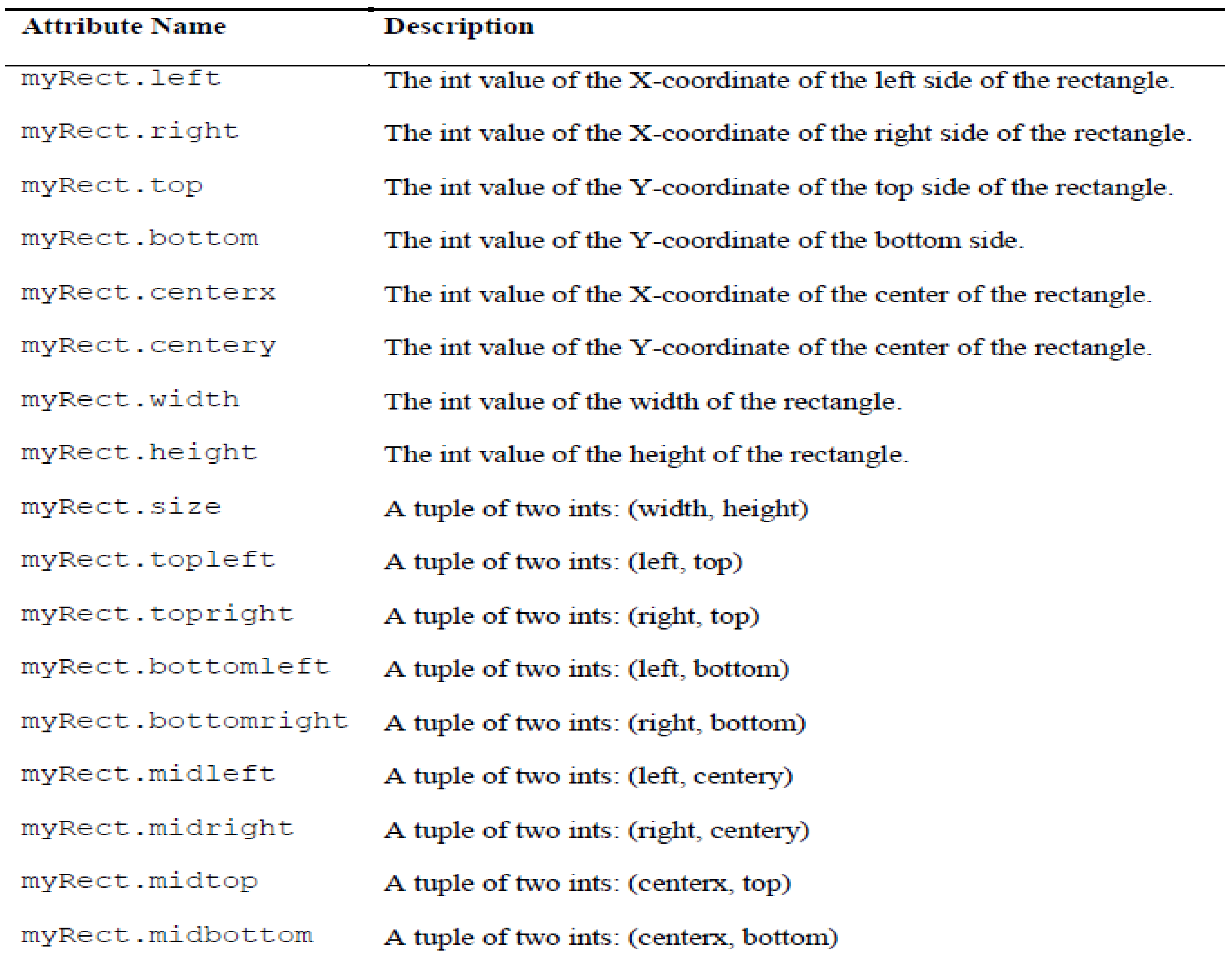
The Pygame code for the Rect object automatically calculated that if the left edge is at the X coordinate 10 and the rectangle is 200 pixels wide, then the right edge must be at the X coordinate 210.

If you reassign the right attribute, all the other attributes are automatically recalculated:



```
>>> spam.right = 350
>>> spam.left
150
```

Rectangle Objects



Attribute Name	Description
`myRect.left`	The int value of the X-coordinate of the left side of the rectangle.
`myRect.right`	The int value of the X-coordinate of the right side of the rectangle.
`myRect.top`	The int value of the Y-coordinate of the top side of the rectangle.
`myRect.bottom`	The int value of the Y-coordinate of the bottom side.
`myRect.centerx`	The int value of the X-coordinate of the center of the rectangle.
`myRect.centery`	The int value of the Y-coordinate of the center of the rectangle.
`myRect.width`	The int value of the width of the rectangle.
`myRect.height`	The int value of the height of the rectangle.
`myRect.size`	A tuple of two ints: (width, height)
`myRect.topleft`	A tuple of two ints: (left, top)
`myRect.topright`	A tuple of two ints: (right, top)
`myRect.bottomleft`	A tuple of two ints: (left, bottom)
`myRect.bottomright`	A tuple of two ints: (right, bottom)
`myRect.midleft`	A tuple of two ints: (left, centery)
`myRect.midright`	A tuple of two ints: (right, centery)
`myRect.midtop`	A tuple of two ints: (centerx, top)
`myRect.midbottom`	A tuple of two ints: (centerx, bottom)

Primitive Drawing Function Example

LECTURE 2

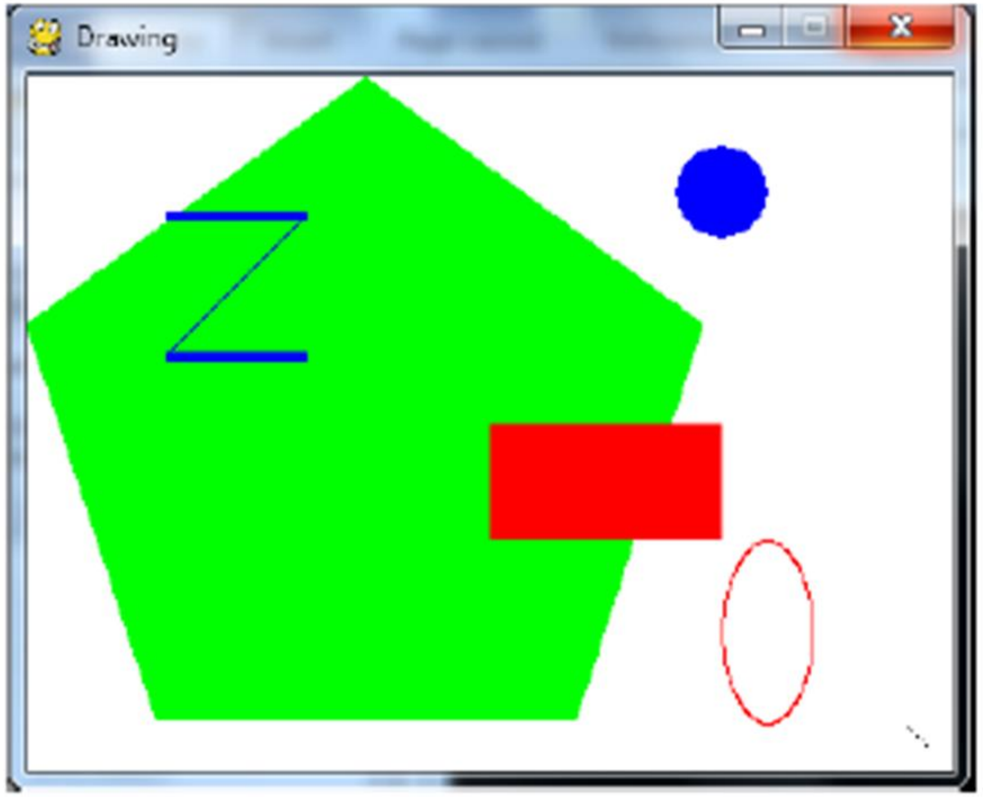
Drawing Primitives

Pygame provides several different functions for drawing different shapes onto a **surface object**. These shapes such as:

- Rectangles
- Circles
- Ellipses
- Lines
- Individual pixels

are called **drawing primitives**.

Our First Drawing Program



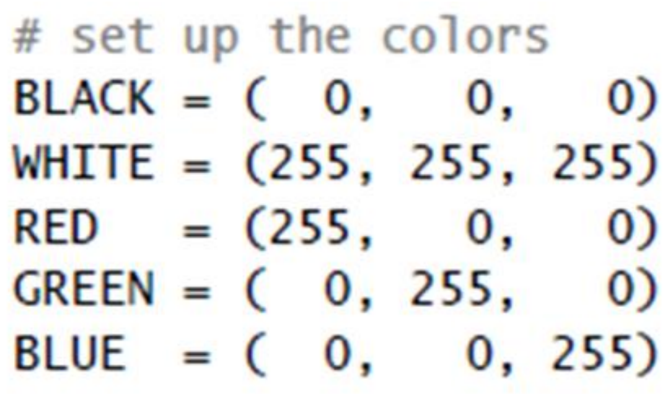
Drawing.py

```
# Drawing.py
import pygame, sys
from pygame.locals import *

pygame.init()
# set up the window
DISPLAYSURF = pygame.display.set_mode((500, 400), 0, 32)
pygame.display.set_caption('Drawing')

# set up the colors
BLACK = (  0,   0,   0)
WHITE = (255, 255, 255)
RED = (255,   0,   0)
GREEN = (  0, 255,   0)
BLUE = (  0,   0, 255)
```

```python
# draw on the surface object                                    # Drawing.py
DISPLAYSURF.fill(WHITE)
pygame.draw.polygon(DISPLAYSURF, GREEN, ((146, 0), (291, 106), (236, 277), (56, 277), (0, 106)))
pygame.draw.line(DISPLAYSURF, BLUE, (60, 60), (120, 60), 4)
pygame.draw.line(DISPLAYSURF, BLUE, (120, 60), (60, 120))
pygame.draw.line(DISPLAYSURF, BLUE, (60, 120), (120, 120), 4)
pygame.draw.circle(DISPLAYSURF, BLUE, (300, 50), 20, 0)
pygame.draw.ellipse(DISPLAYSURF, RED, (300, 250, 40, 80), 1)
pygame.draw.rect(DISPLAYSURF, RED, (200, 150, 100, 50))

pixObj = pygame.PixelArray(DISPLAYSURF)
pixObj[480][380] = BLACK
pixObj[482][382] = BLACK
pixObj[484][384] = BLACK
pixObj[486][386] = BLACK
pixObj[488][388] = BLACK
del pixObj

# run the game loop
while True:
    for event in pygame.event.get():
        if event.type == QUIT:
            pygame.quit()
            sys.exit()
    pygame.display.update()
```

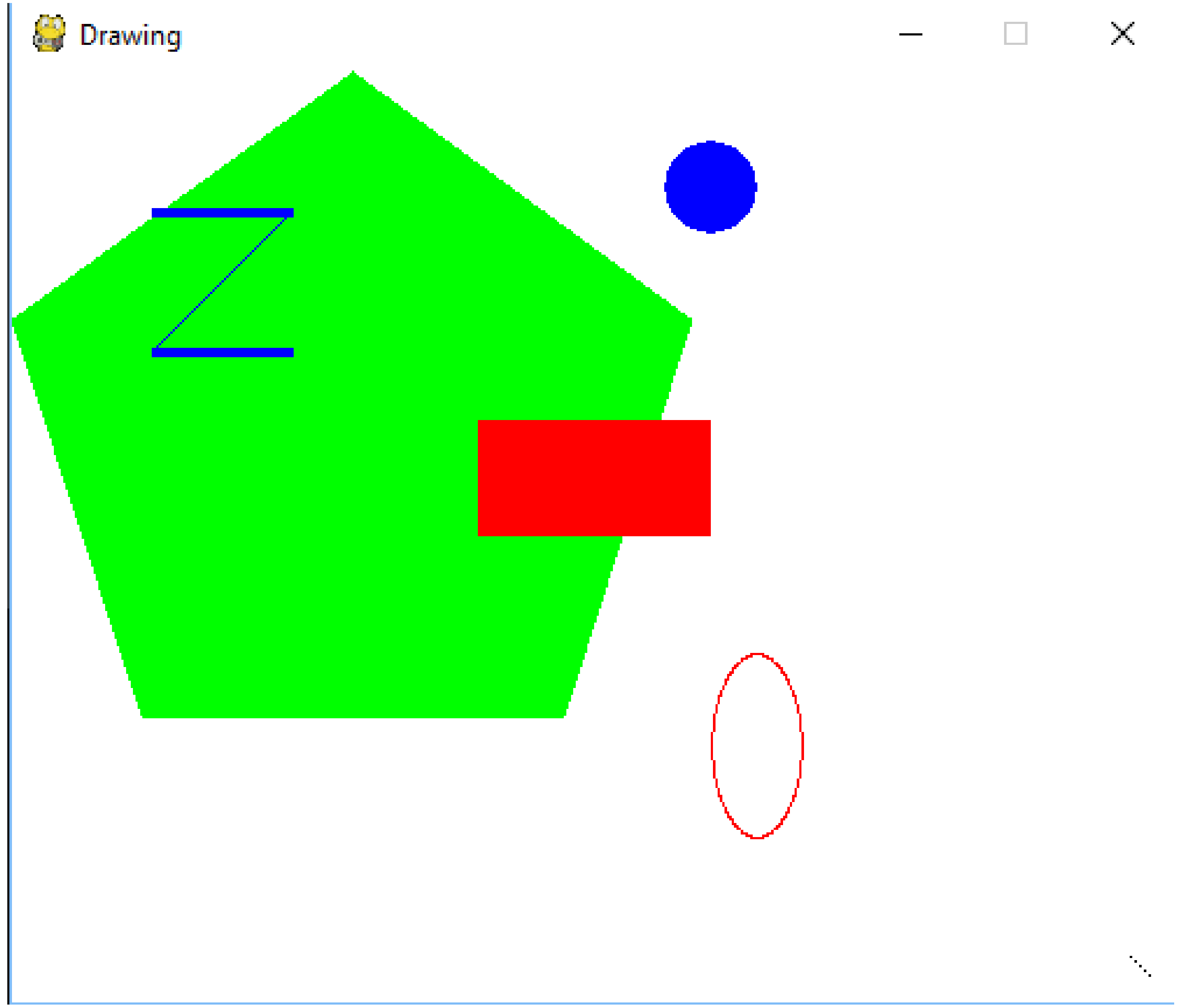
Drawing.py

- Notice how we make constant variables for each of the colors?
- Doing this makes our code more readable.
- **GREEN** in the source code is easier to understand than **(0, 255, 0)** !



```
# set up the colors
BLACK = (  0,   0,   0)
WHITE = (255, 255, 255)
RED   = (255,   0,   0)
GREEN = (  0, 255,   0)
BLUE  = (  0,   0, 255)
```

Drawing.py



Drawing.py

- The drawing functions are named after the shapes they draw.
- The parameters you pass these functions tell them:
  1. Which Surface object to draw on
  2. Where to draw the shape (and what size)
  3. What color to use
  4. And how wide to make the lines.

Primitive Drawing Functions

LECTURE 2

fill(color)

The fill() method is not a function but a method of **pygame.Surface** objects. It will completely fill in the entire **Surface** object with whatever color value you pass as for the color parameter.

pygame.draw.polygon(surface, color, pointlist, width)

- A polygon is shape made up of only flat sides. The surface and color parameters tell the function on what surface to draw the polygon, and what color to make it.
- The **pointlist** parameter is a tuple or list of points (that is, tuple or list of two-integer tuples for XY coordinates). The polygon is drawn by drawing lines between each point and the point that comes after it in the tuple. Then a line is drawn from the last point to the first point. You can also pass a list of points instead of a tuple of points.

pygame.draw.polygon(surface, color, pointlist, width)

- The **width** parameter is optional. If you leave it out, the polygon that is drawn will be filled in, just like our green polygon on the screen is filled in with color. If you do pass an integer value for the width parameter, only the outline of the polygon will be drawn.
- The integer represents how many pixels width the polygon's outline will be. Passing 1 for the width parameter will make a skinny polygon, while passing 4 or 10 or 20 will make thicker polygons. If you pass the integer 0 for the width parameter, the polygon will be filled in (just like if you left the width parameter out entirely).

pygame.draw.polygon(surface, color, pointlist, width)

- All of the **pygame.draw** drawing functions have optional width parameters at the end, and they work the same way as **pygame.draw.polygon()**’s width parameter. Probably a better name for the width parameter would have been **thickness**, since that parameter controls how thick the lines you draw are.

pygame.draw.line(surface, color, start_point, end_point, width)

This function draws a line between the **start_point** and **end_point** parameters.

pygame.draw.lines(surface, color, closed, pointlist, width)

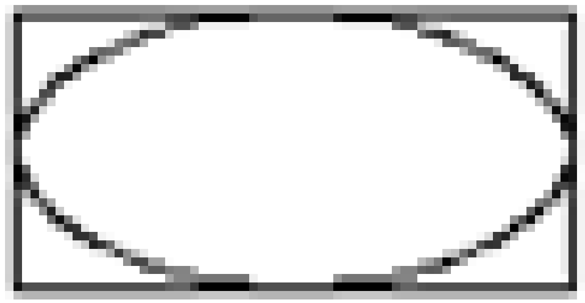
- This function draws a series of lines from one point to the next, much like **pygame.draw.polygon()**.
- The only difference is that if you pass False for the closed parameter, there will not be a line from the last point in the pointlist parameter to the first point. If you pass True, then it will draw a line from the last point to the first.

pygame.draw.circle(surface, color, center_point, radius, width)

- This function draws a circle. The center of the circle is at the **center_point** parameter. The integer passed for the radius parameter sets the size of the circle.
- The radius of a circle is the distance from the center to the edge. (The radius of a circle is always half of the diameter.) Passing 20 for the radius parameter will draw a circle that has a radius of 20 pixels.

pygame.draw.ellipse(surface, color, bounding_rectangle, width)

- This function draws an ellipse (which is like a squashed or stretched circle). This function has all the usual parameters, but in order to tell the function how large and where to draw the ellipse, you must specify the bounding rectangle of the ellipse. A bounding rectangle is the smallest rectangle that can be drawn around a shape. Here's an example of an ellipse and its bounding rectangle:



pygame.draw.ellipse(surface, color, bounding_rectangle, width)

- The bounding_rectangle parameter can be a pygame.Rect object or a tuple of four integers. Note that you do not specify the center point for the ellipse like you do for the pygame.draw.circle() function.

pygame.draw.rect(surface, color, rectangle_tuple, width)

- This function draws a rectangle. The rectangle_tuple is either a tuple of four integers (for the XY coordinates of the top left corner, and the width and height) or a pygame.Rect object can be passed instead. If the rectangle_tuple has the same size for the width and height, a square will be drawn.

Pixel Array Object

LECTURE 2

Pixel Array Objects

- Unfortunately, there isn’t a single function you can call that will set a single pixel to a color (unless you call **pygame.draw.line()** with the same start and end point).
- The **pygame** framework needs to run some code behind the scenes before and after drawing to a Surface object.
- If it had to do this for every single pixel you wanted to set, your program would run much slower. (Almost 2 to 3 times slower).

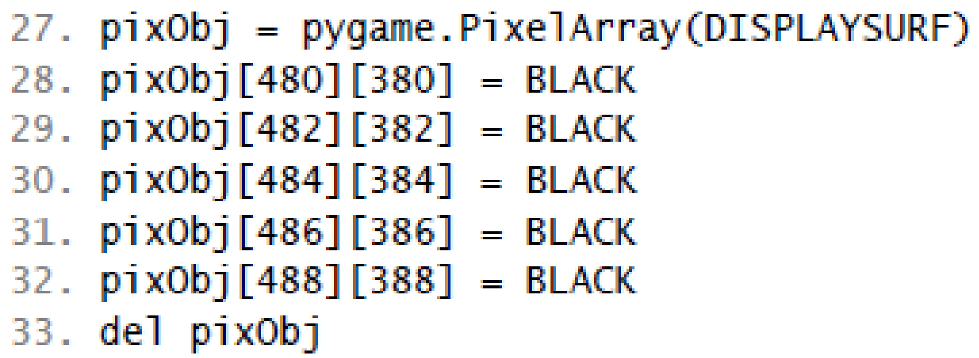
Pixel Array Objects

- Instead, you should create a
  pygame.PixelArray object
- (we'll call them **PixelArray** objects for short) of a **Surface** object and then set individual pixels.
- Creating a **PixelArray** object of a **Surface** object will "lock" the Surface object.
- While a **Surface** object is locked, the drawing functions can still be called on it, but it cannot have images like **PNG** or **JPG** images drawn on it.

Pixel Array Objects

set then delete

- The PixelArray object that is returned from pygame.PixelArray() can have individual pixels set by accessing them with two indexes.
- For example, line 28’s pixObj[480][380] = BLACK will set the pixel at X coordinate 480 and Y coordinate 380 to be black.



```
27. pixObj = pygame.PixelArray(DISPLAYSURF)
28. pixObj[480][380] = BLACK
29. pixObj[482][382] = BLACK
30. pixObj[484][384] = BLACK
31. pixObj[486][386] = BLACK
32. pixObj[488][388] = BLACK
33. del pixObj
```

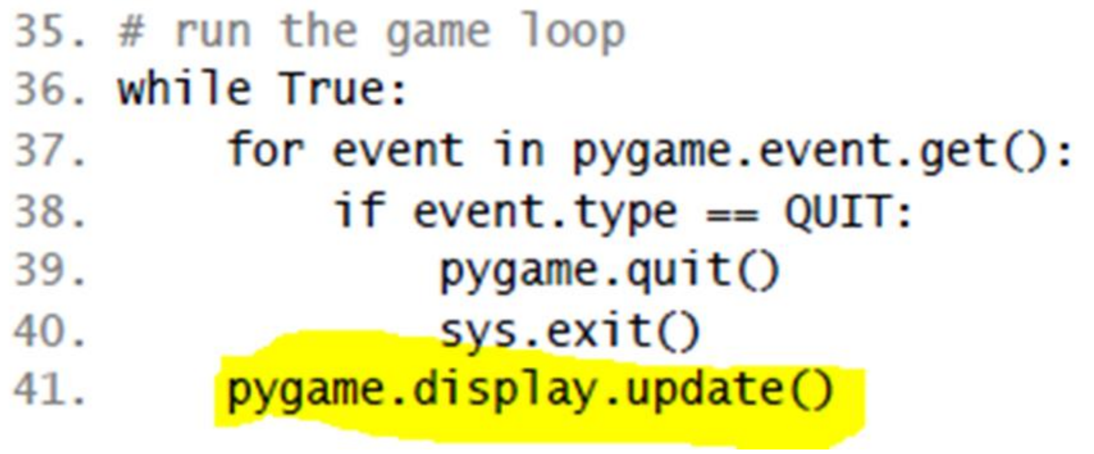
Pixel Array Objects

- To tell Pygame that you are finished drawing individual pixels, delete the PixelArray object with a del statement.   See line 33
- Deleting the **PixelArray** object will “unlock” the Surface object so that you can once again draw images on it.
- If you forget to delete the **PixelArray** object, the next time you try to blit (that is, draw) an image to the Surface, you will get an ERROR.

```
27. pixObj = pygame.PixelArray(DISPLAYSURF)
28. pixObj[480][380] = BLACK
29. pixObj[482][382] = BLACK
30. pixObj[484][384] = BLACK
31. pixObj[486][386] = BLACK
32. pixObj[488][388] = BLACK
33. del pixObj
```

Pygame.display. Update() function

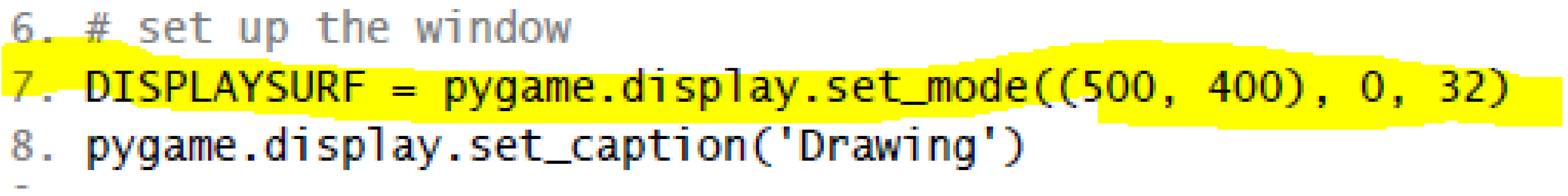
After you are done calling the drawing functions to make the display Surface object look the way you want, you must call **pygame.display.update()** to make the display Surface actually appear on the user's monitor.



```
# run the game loop
while True:
    for event in pygame.event.get():
        if event.type == QUIT:
            pygame.quit()
            sys.exit()
    pygame.display.update()
```

Pygame.display. Update() function

- The one thing that you must remember is that **pygame.display.update()** will only make the display Surface (that is, the Surface object that was returned from the call to **pygame.display.set_mode()**) appear on the screen.
- If you want the images on other Surface objects to appear on the screen, you must “blit” them (that is, copy them) to the display Surface object with the **blit()** method. We will discuss this later…



```
# set up the window
DISPLAYSURF = pygame.display.set_mode((500, 400), 0, 32)
pygame.display.set_caption('Drawing')
```

Learning Channel

PROJECT TIME