Attributes, Modes, and Color

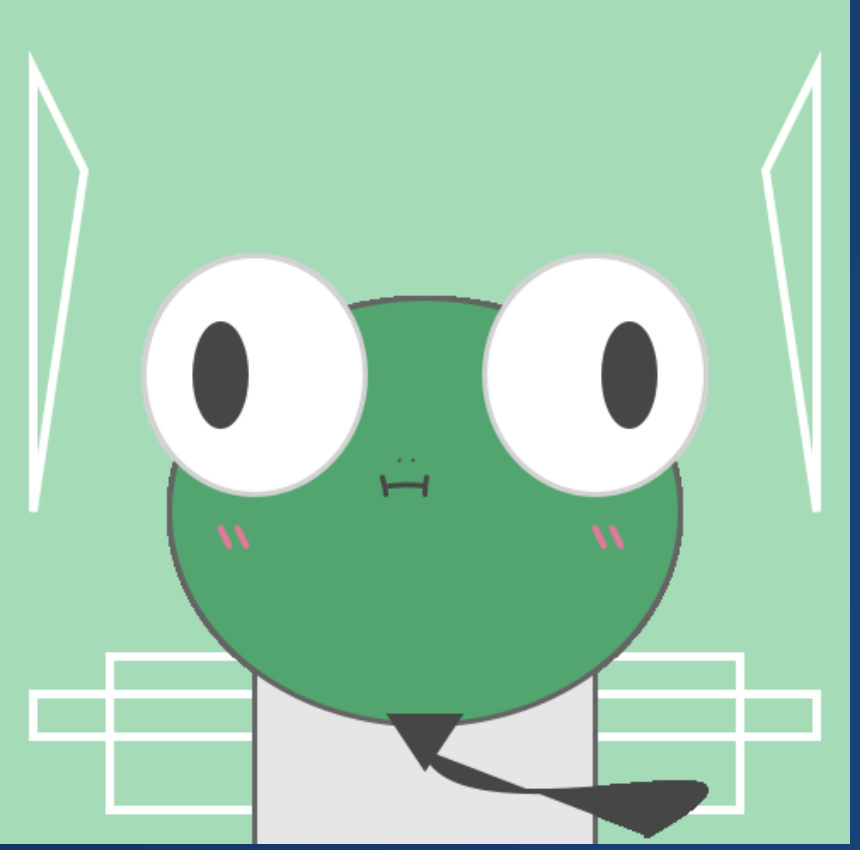
Last Time

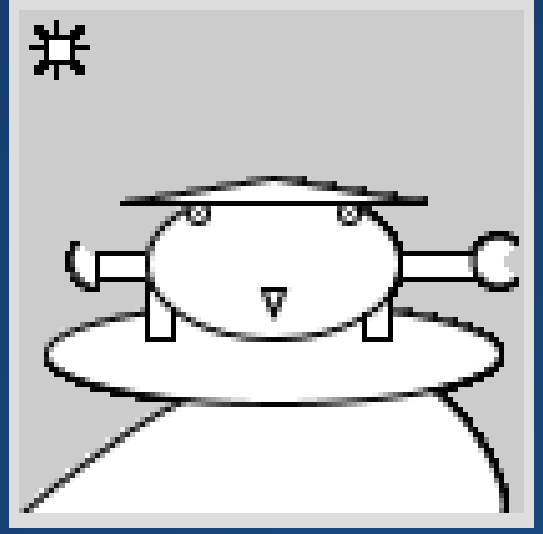
Shape Primitives: Shapes included in Processing as a building block for more complex shapes.

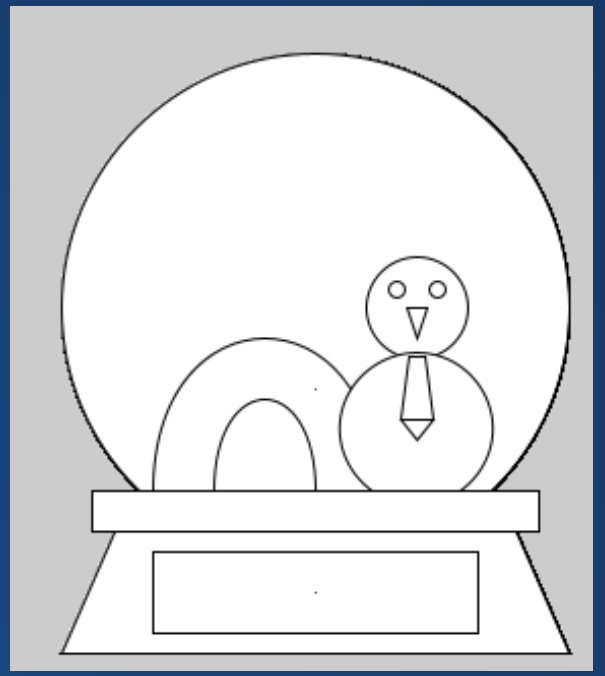
- point
- triangle
- ellipse
- rect
- arc
- bezier

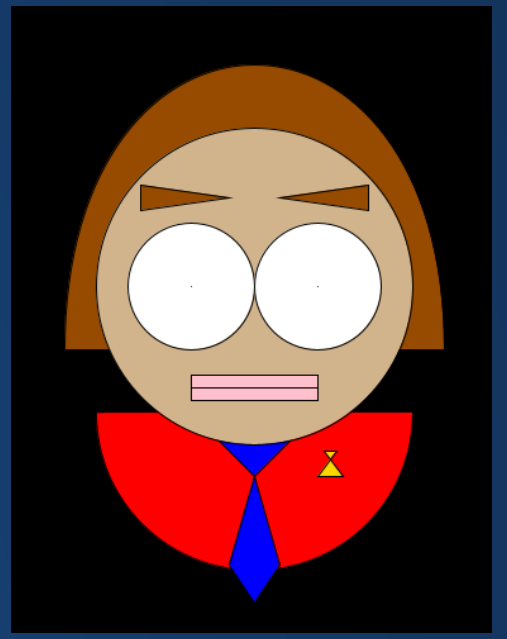
Locations described in *screen space* (a.k.a. the screen coordinate system).

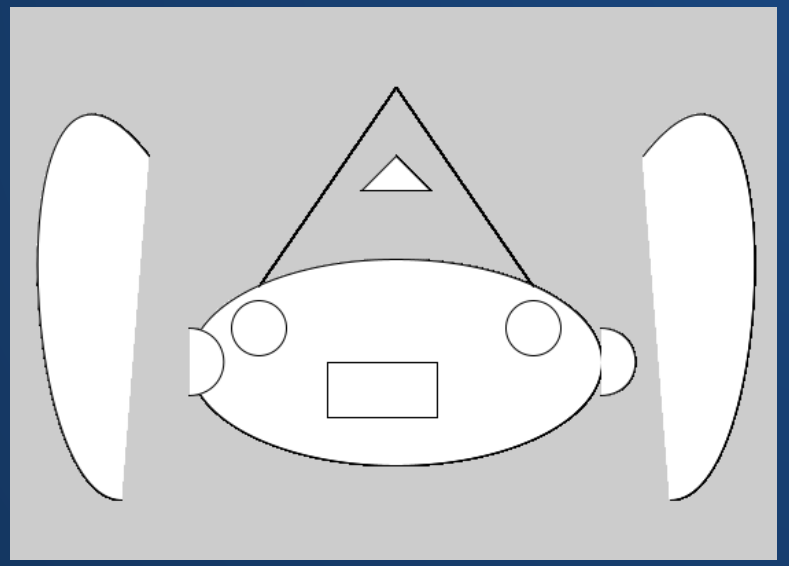
Code Review

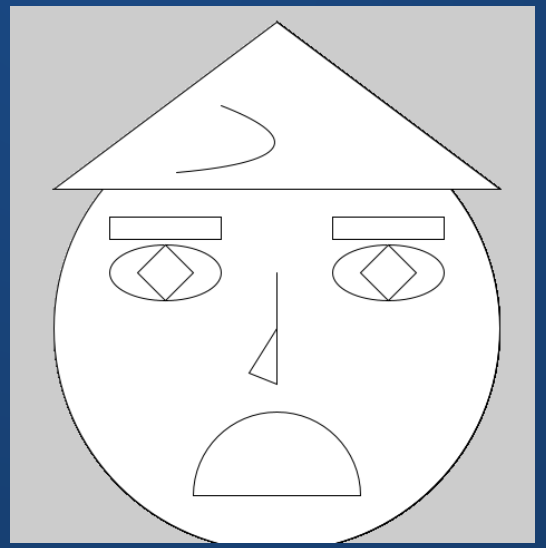


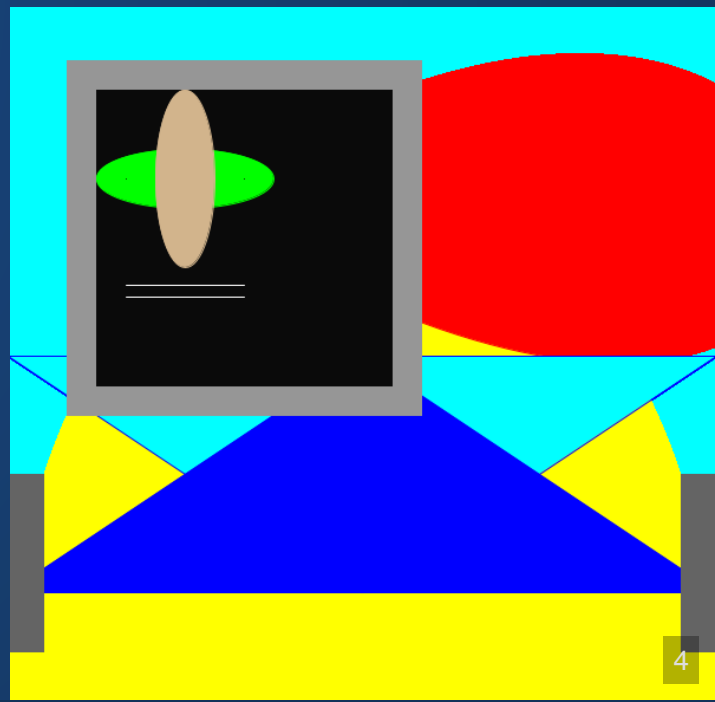












There are two kinds of code:

- Code that "makes a thing available".
- Code that "actually does a thing".

```
1  float hello;
2  PImage myImg;
3  float myDouble(float x){
4    return 2*x;
5  }
6
7  hello = 3.0;
8  ellipse(2, 5, 10, 50);
9  fill(20, 30, 50);
10 myDouble(hello);
```

If your code "does something", it needs to live within setup() or draw().

If you're interested in why, find me during office hours.

Q: Why can't we write code outside of setup() and draw()?

Well....you *kinda* can. Processing offers a "static" mode where we can write all our code at the top level. But once we do this, we can no longer write *any* functions, and we lose the ability to run code repeatedly.

```
1 // This Processing program compiles and runs
2 String s = "The size is ";
3 int w = 1920;
4 int h = 1080;
5 println(s);
6 println(w, "x", h);
```

The `setup()`/`draw()` separation is initially confusing, but much more powerful in the long run.

Processing?

Will this language be easy to learn in?

Yes*.

What will we be able to do by the end of this class?

Work with 2D and 3D scenes which move, are interactive, and use data stored on the computer. Basic games, data visualization, and short animations are all on the table.

Why is Processing considered a simple language?

*terms and conditions apply

```
import { CanvasAnimatio
import { GUI } from "./
import { Mat4, Vec4, Ve
import { Easel } from "
import { Scene } from "
import { Sphere } from
import { Transform } fr
import { Material } fro

export interface Raytra
  reset(): void;
  draw(): void;
}

export class Raytracer
  private gui: GUI;
  private gl: WebGLRend
  private easel: Easel;
  private scene: Scene;
  private sceneLoaded:

  public constructor(ca
    super(canvas);
    this.gl = this.ctx;
    this.gui = new GUI(
    this.scene = new Sc
    this.sceneLoaded =
    this.reset();
  }

  public draw(): void {
    this.easel.render()
  }

  public reset(): void
    const t = Transform
    this.scene.shapes.p

    this.easel = new Ea
    this.easel.setDefau
  }
}

export function initial
  const canvas = docume
  /* Start drawing */
  const canvasAnimation
  canvasAnimation.start
}
```

```
import {
  CanvasAnimation,
  WebGLUtilities
} from "../lib/webglutils/CanvasAnimatio
import {
  vsSource,
  fsSource
} from "./Shaders.js";
import { Vec3 } from "../lib/TSM.js";

// A class for storing image data and dr
export class Easel {
  public render(): void {
    this.gl.clearColor(0.0, 0.0, 0.0, 1.
    this.gl.clearDepth(1.0); // Clear ev
    this.gl.clear(this.gl.COLOR_BUFFER_B

    this.gl.bindBuffer(this.gl.ARRAY_BUF
    this.gl.vertexAttribPointer(this.ver
    this.gl.enableVertexAttribArray(this

    this.gl.bindBuffer(this.gl.ARRAY_BUF
    this.gl.vertexAttribPointer(this.ver
    this.gl.enableVertexAttribArray(this

    this.gl.activeTexture(this.gl.TEXTUR
    this.gl.bindTexture(this.gl.TEXTURE_
    this.gl.uniform1i(this.uSamplerUnifo

    this.gl.useProgram(this.program);
    this.gl.drawArrays(this.gl.TRIANGLE_
  }

  public reset(): void {
    const nbytes = 3 * this.width * this
    this.data = new Uint8Array(nbytes);
    this.texture = this.gl.createTexture

    this.program = WebGLUtilities.create

    const vertices = new Float32Array([
      -1.0, 1.0,   // Top left
      1.0, 1.0,   // Top right
      -1.0, -1.0,   // Bottom left
      1.0, -1.0,   // Bottom right
    ]);
```

```
    const textureCoordinates = new Flo
      0.0, 0.0,
      1.0, 0.0,
      0.0, 1.0,
      1.0, 1.0,
    ]);

    this.vertexBuffer = this.gl.create
    this.gl.bindBuffer(this.gl.ARRAY_B
    this.gl.bufferData(this.gl.ARRAY_B

    this.textureBuffer = this.gl.creat
    this.gl.bindBuffer(this.gl.ARRAY_B
    this.gl.bufferData(this.gl.ARRAY_B

    this.vertexPosAttribLoc = this.gl.
    this.vertexTexCoordAttribLoc = thi
    this.uSamplerUniformLoc = this.gl.
  }

  /// Set appearance to a polictical c
  public setDefaultAppearance(): void {
    const lightRed = new Vec3([1.0, 0.5, 0.5]);
    const lightBlue = new Vec3([0.5, 0.5, 1.0]);
    const lightGreen = new Vec3([0.5, 1.0, 0.5]);
    const lightPurple = new Vec3([1.0, 0.5, 1.0]);
    for (let y = 0; y < 512; y++) {
      for (let x = 0; x < 512; x++) {
        if (x < 256 && y < 256) {
          this.setColor(x, y, lightRed);
        } else if (x >= 256 && y < 256) {
          this.setColor(x, y, lightBlue);
        } else if (x < 256 && y >= 256) {
          this.setColor(x, y, lightGreen);
        } else {
          this.setColor(x, y, lightPurple);
        }
        if (x == 256 || y == 256) {
          this.setColor(x, y, new Vec3([0.0, 0.0, 0.0]));
        }
      }
    }
  }
```

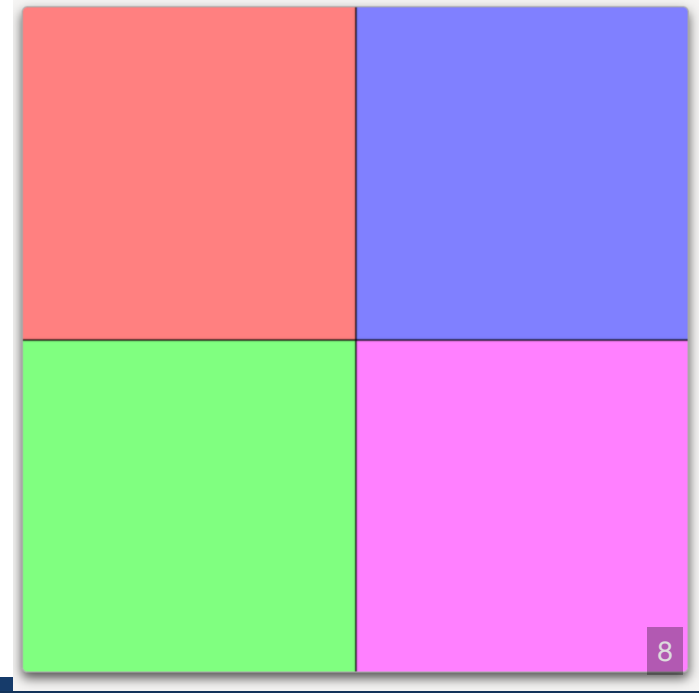
```
export const vsSource = `
  attribute vec4 aVertexPosition;
  attribute vec2 aTextureCoord;
  varying highp vec2 vTextureCoord;

  void main(void) {
    vTextureCoord = aTextureCoord;
    gl_Position = aVertexPosition;
  }
`;

export const fsSource = `
  varying highp vec2 vTextureCoord;

  uniform sampler2D uSampler;

  void main(void) {
    gl_FragColor = texture2D(uSampler, vTextureCoord);
    // gl_FragColor = vec4(vTextureCoord.x, vTextureCoord.y, 0.0, 1.0);
    // gl_FragColor = vec4(1.0, 0.6, 0.4, 1.0);
  }
`;
```



Many of the basic ideas that we find in Processing exist in these "more complex" languages!

One example: the above code has a function called `draw()`.
This function is called 60 times per second by the JavaScript library responsible for the screen layout.

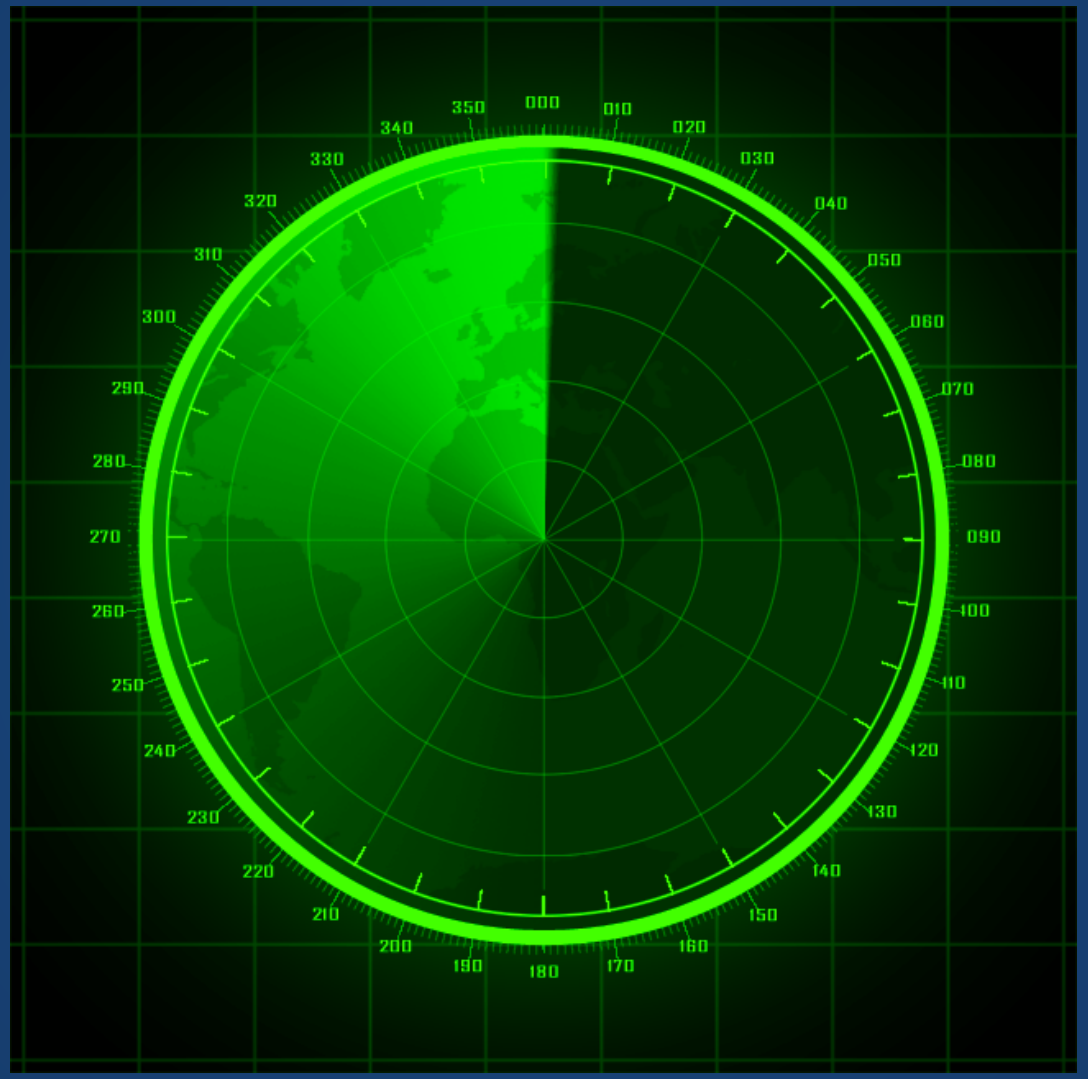
Béziers are Silly!

Okay, yes. Working with Bézier curves the way Processing has you do it is an exercise in pain and futility.

But pair it with a graphical interface or a way to programmatically generate these control points...

```
1  class ControlPoint {
2    PVector location;
3    color c;
4    float r = 30;
5
6    ControlPoint(float x, float y, color c_) {
7      location = new PVector(x, y);
8      c = c_;
9    }
10
```

Do circular screens use polar coordinates?



It depends....but usually no.

Lightning Round!

How can I make two different files with draw() and setup() methods?

Processing distinguishes between "projects" which consist of multiple files in the same directory, and actual files. Files within the same project can interact with each other.

If you want new code to not interact with anything you've already written, you need a new project (File > New). This will open a new Processing window.

If we're absent, are hands-on assignments still due at 7:30?

Yes. Contact me if this will be a problem.

Can we work on the hands-on assignments ahead of time?

I would prefer you didn't, but I can't stop you.

All lecture slides that I have not covered yet are considered non-final. This includes the hands-on assignments.

Attributes

Attributes

- Function calls which modify the appearance of shape primitives
- Apply to all primitives displayed ***after*** the attribute

Attributes: Fill and Stroke

- `background()` sets the background color of the screen
- `fill()` sets the fill color for a shape
- `stroke()` sets the outline color for a shape
- `noFill()` and `noStroke()` prevent shape fill or shape stroke respectively

```
1  int WHITE = 255;
2  int BLACK = 0;
3
4  fill(WHITE);
5  stroke(BLACK);
6  rect(0, 0, 50, 50);
7
8  fill(BLACK);
9  stroke(WHITE);
10 rect(50, 50, 50, 50);
```

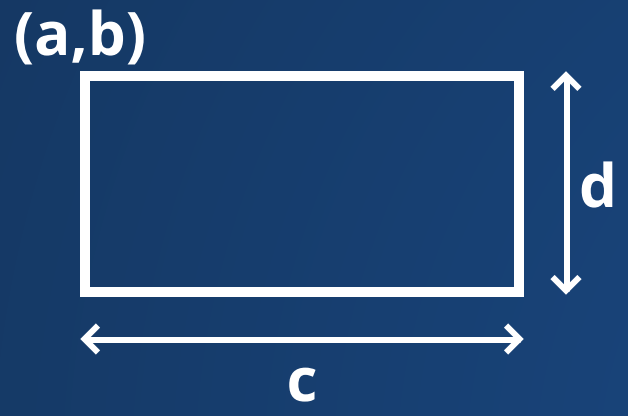
Modes

- Function calls which modify the interpretation of shape primitive functions
- Apply to all primitives displayed ***after*** the mode

Modes for rect and ellipse

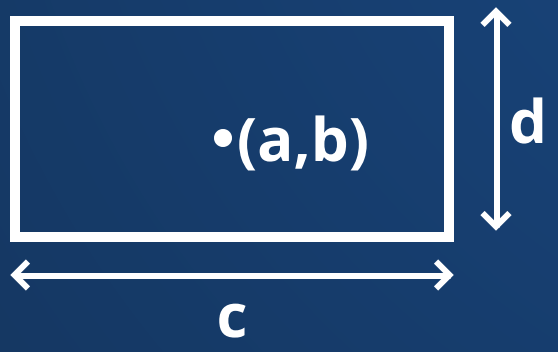
- `rectMode()` and `ellipseMode()` take a parameter:
  - `CORNER`, `CORNERS`, `CENTER`, `RADIUS`
- These parameters change how the parameters to calls to `rect()` and `ellipse()` are interpreted.

```
rectMode(CORNER);
rect(a, b, c, d);
```

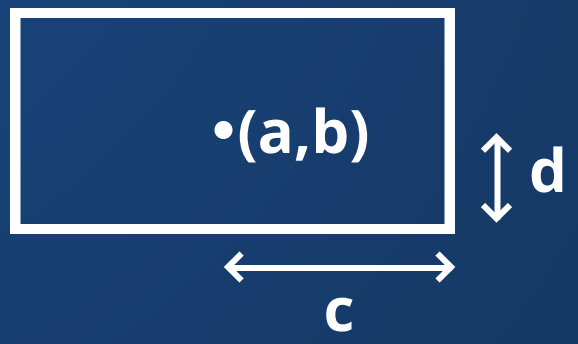



```
rectMode(CORNERS);
rect(a, b, c, d);
```


```
rectMode(CENTER);
rect(a, b, c, d);
```




```
rectMode(RADIUS);
rect(a, b, c, d);
```



Other Modes

Modes allow for different models within the context of the same method.

Other modes in Processing:

- colorMode
- textureMode
- imageMode
- shapeMode
- blendMode
- textMode

Order Matters!

Attribute and mode commands only affect the commands that come *after* them. Ordering of statements is important!

```
1  stroke(100);
2  rect(80, 120, 150, 40);
3  stroke(200);
4  rect(50, 100, 150, 40);
```

Order Matters!

The order of draw commands can also affect output!

```
1  rect(50, 100, 150, 40);
2  rect(80, 120, 150, 40);
```

vs

```
1  rect(80, 120, 150, 40);
2  rect(50, 100, 150, 40);
```

Order Matters *Globally*

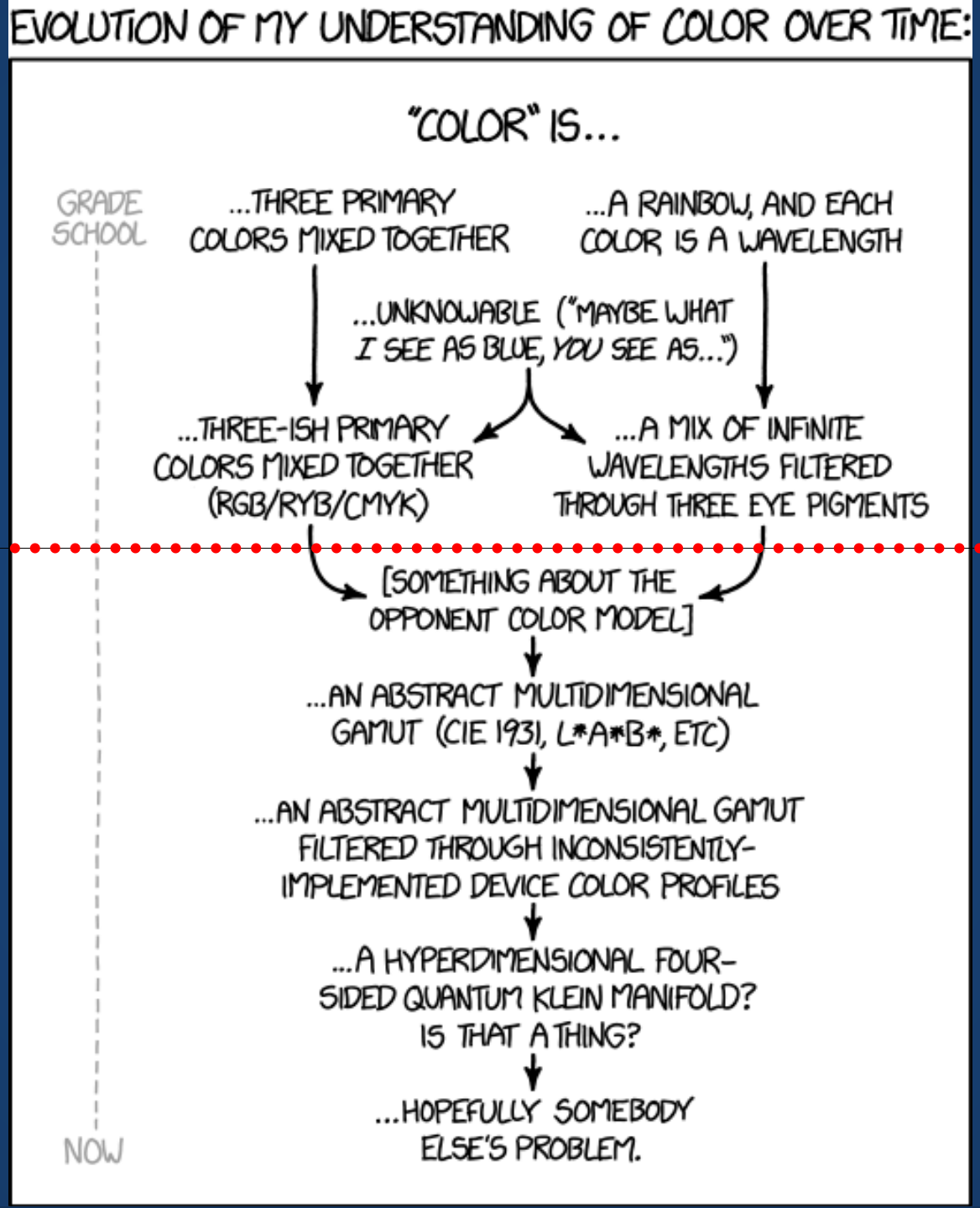
```
1  int WHITE = 255;
2  int BLACK = 0;
3
4  fill(WHITE);
5  stroke(BLACK);
6  rect(0, 0, 50, 50);
7
8  drawRect(0, 0, 100, 100);
9
10 // What color does this draw in?
11 rect(100, 100, 100, 100);
```

```
1 void drawRect(int a, int b,
2                int c, int d){
3   fill(BLACK);
4   stroke(BLACK);
5   rect(a,b,c,d);
6 }
```

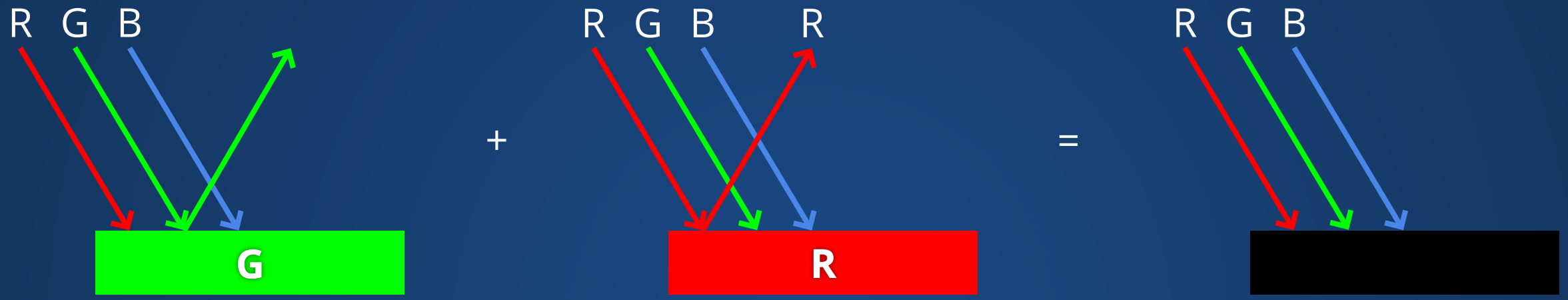
Hands-on: Using Attributes

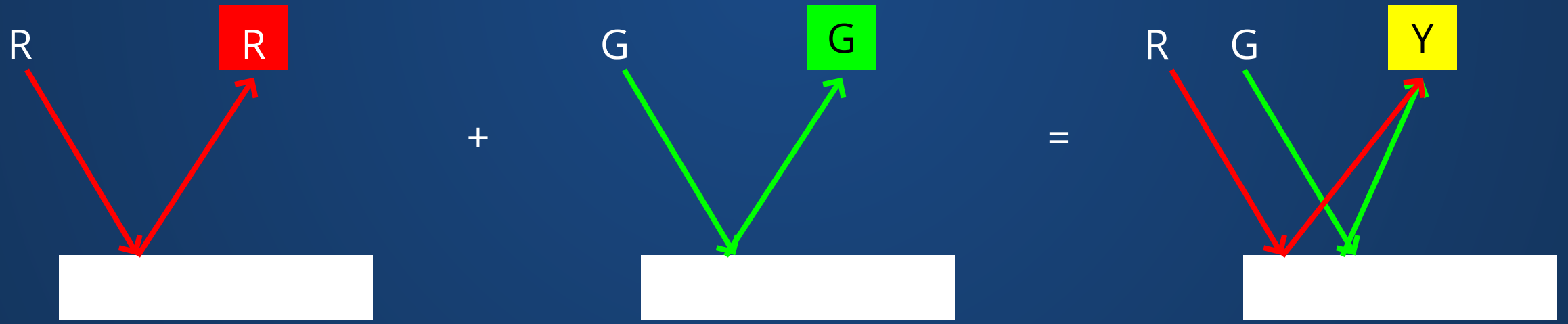
1. Experiment with `stroke()`, `fill()`, `noStroke()`, and `noFill()`.
2. Draw a rectangle and an ellipse and then try out at least one alternate mode on these shapes.
3. Experiment with the order of attribute/mode/draw calls. Create at least two shape clusters which demonstrate a difference in order.
4. Answer the following questions in a block comment:
    1. How can order create the illusion of depth?
    2. How can primitive modes help us build images?

Color


EVOLUTION OF MY UNDERSTANDING OF COLOR OVER TIME:
"COLOR" IS...
GRADE SCHOOL
...THREE PRIMARY COLORS MIXED TOGETHER
...A RAINBOW, AND EACH COLOR IS A WAVELENGTH
...UNKNOWABLE ("MAYBE WHAT I SEE AS BLUE, YOU SEE AS...")
...THREE-ISH PRIMARY COLORS MIXED TOGETHER (RGB/RYB/CMYK)
...A MIX OF INFINITE WAVELENGTHS FILTERED THROUGH THREE EYE PIGMENTS
[SOMETHING ABOUT THE OPPONENT COLOR MODEL]
...AN ABSTRACT MULTIDIMENSIONAL GAMUT (CIE 1931, L*A*B*, ETC)
...AN ABSTRACT MULTIDIMENSIONAL GAMUT FILTERED THROUGH INCONSISTENTLY-IMPLEMENTED DEVICE COLOR PROFILES
...A HYPERDIMENSIONAL FOUR-SIDED QUANTUM KLEIN MANIFOLD? IS THAT A THING?
...HOPEFULLY SOMEBODY ELSE'S PROBLEM.
NOW

Additive vs Subtractive



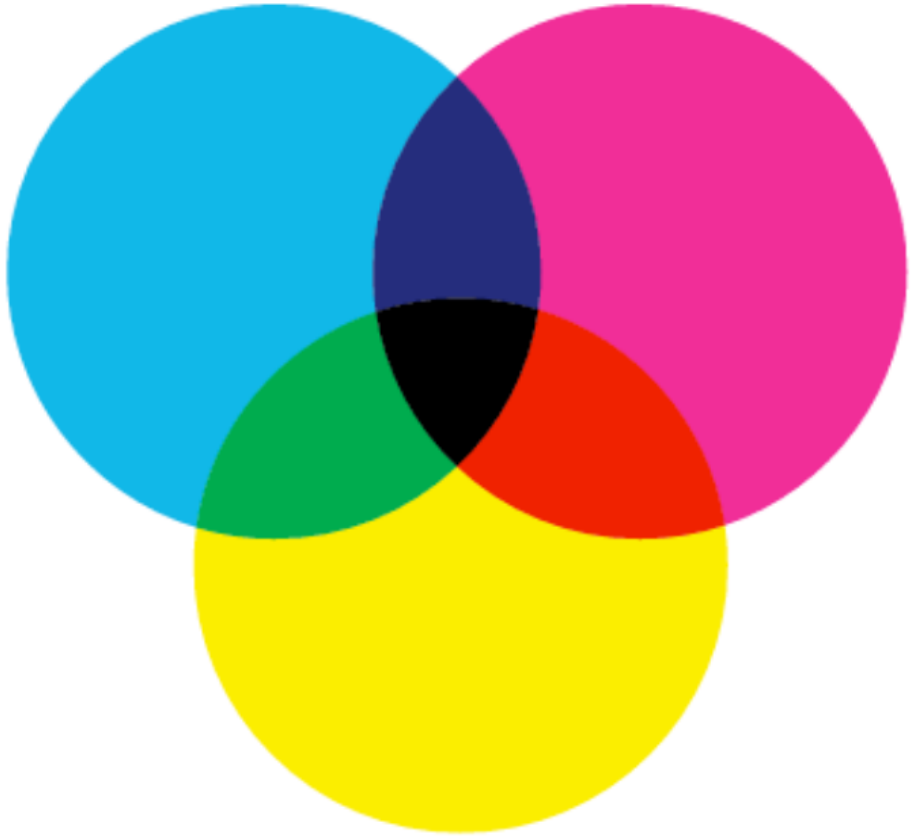


Note: this isn't quite true! c.f. "Color is a mess"

Color Models

RGB

Additive color



CYM(K)

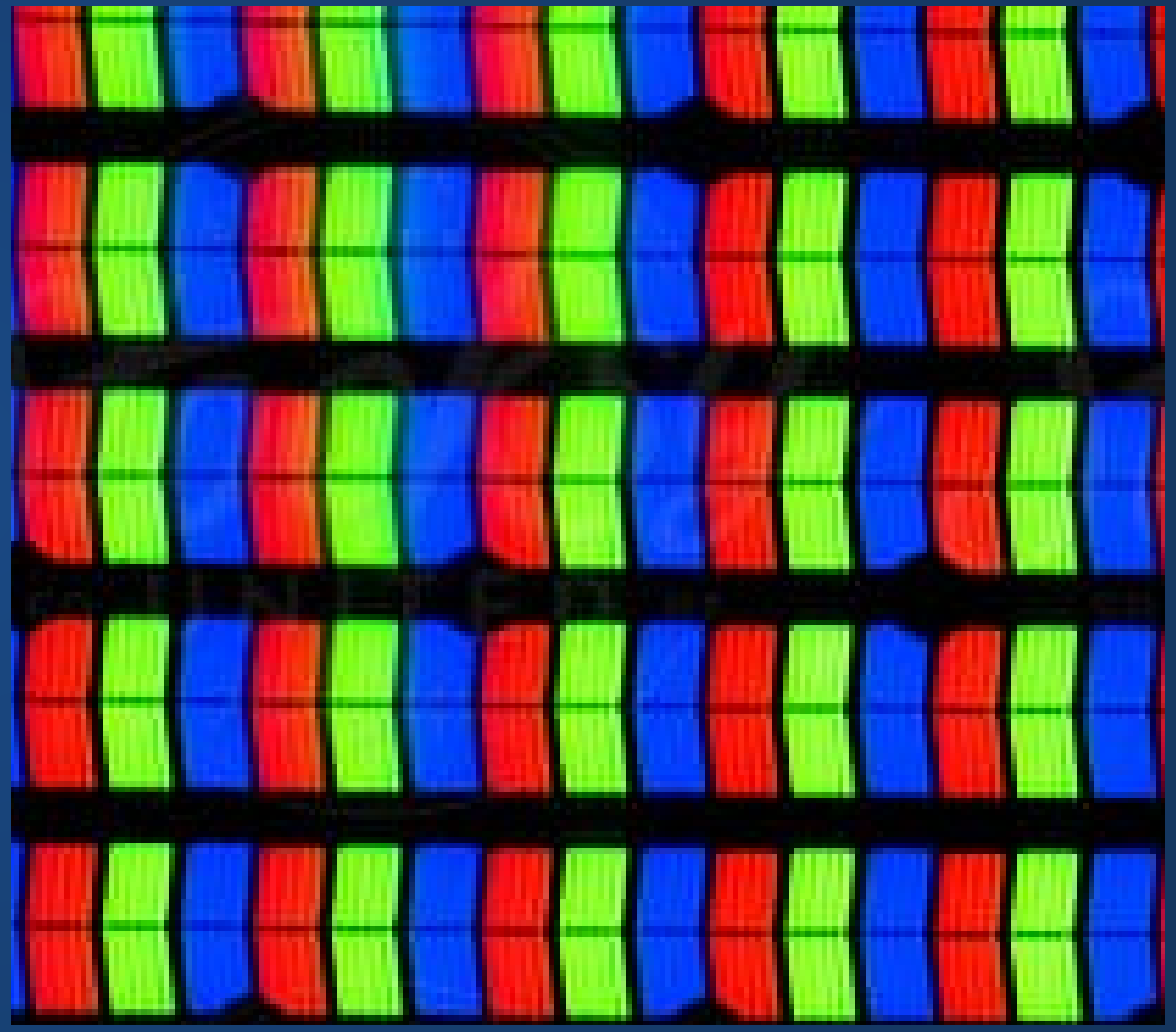
Subtractive color

Digital Color

Each pixel has three light elements: red, green, and blue.

Light element intensity ranges from 0 to 255

- 0 means element is off
- 255 means element is max



What color model is this?

Colors!

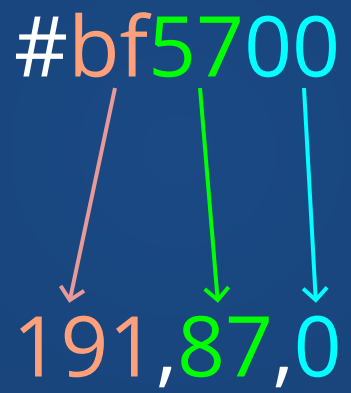
- **Red: 255, 0, 0**
- **Green: 0, 255, 0**
- **Blue: 0, 0, 255**

Colors at full intensity can be a little much!

Processing includes a color selector for more intuitive color selection if you don't have access to a digital paint program (Tools > Color Selector)

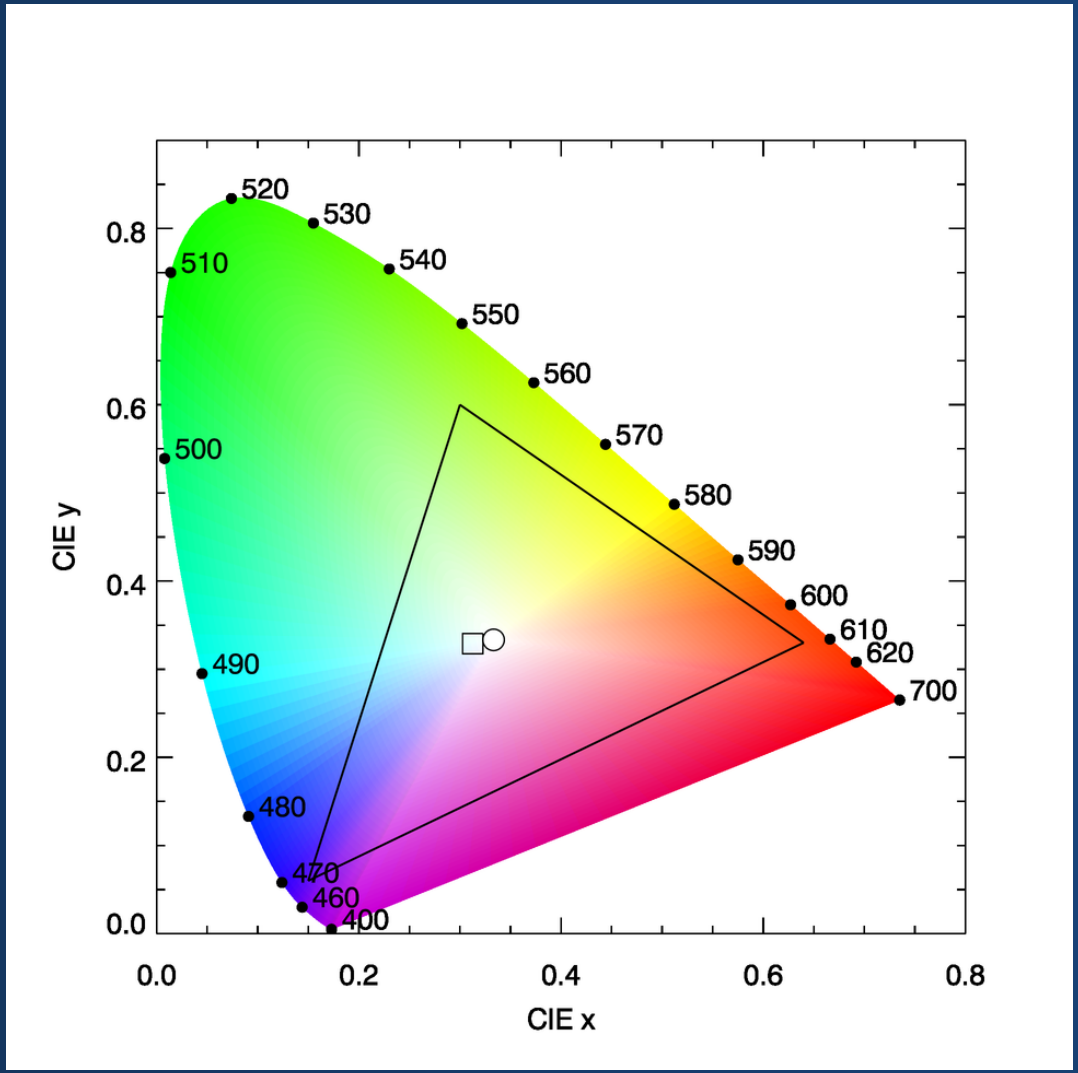
Hexadecimal

A popular notation for writing down RGB colors. Each number 0-255 is encoded as a two-digit hex.



Hey, what color is this anyways?

"True Color"



Three 8-bit channels (RGB) makes for a 24-bit color space. Sometimes add a 4th channel for transparency (RGBA).

Allows us to represent 16,777,216 different colors. Humans can see around 10 million colors.

Does not actually cover all colors humans can see!

Color and Space Costs

- How many values can a bit store?
- How many values can two bits store?
- How many values can three bits store?
- How many bits do we need to store 255 values?

Images are expensive!

For true color, we store 8 bits of information per color (0-255), or 24 bits per pixel.

How much data do we need for a 1080p60 video?

$$24\frac{\text{bits}}{\text{pixel}} \times \left(1920 \times 1080\frac{\text{pixel}}{\text{frame}}\right) \times 60\frac{\text{frames}}{\text{second}} = \quad 2{,}985{,}984{,}000 \text{ bits/second}$$

370 megabytes per second!

How can we tame the amount of data we need to store images?

Color Depth

Use different number of bits to store color data! If we only use 8 bits per pixel, we can store 1/3rd the data relative to true color.

True Color (24 bits)

12-bit Color

9-bit Color

Image Formats

Various image formats have different color and transparency depths.

- GIF
  - Color depth: 1-bit to 8-bit
  - Transparency: 1-bit
- JPEG
  - Color depth: 24-bit
  - Transparency: None
- PNG
  - Color depth: 1-bit to 24-bit
  - Transparency: 8-bit

Processing and Blending

There are functions in Processing that allow us to set colors.

- `background(int red, int green, int blue)` sets the color of the window in terms of RGB
- `fill(int red, int green, int blue)` sets the color for any subsequent shape primitives
- `fill(int red, int green, int blue, int alpha)` includes a transparency channel to modify opacity

The `color` primitive

Processing has a special type for color:

`color(float red, float green, float blue);`

We can store it and use it in functions that expect color.

```
1 color yellow = color(255.0, 255.0, 0.0);
2 fill(yellow);
3 rect(0, 0, 200, 200);
4
5 // Processing also accepts color hexes!
6 color burnt_orange = #bf5700;
```

An aside

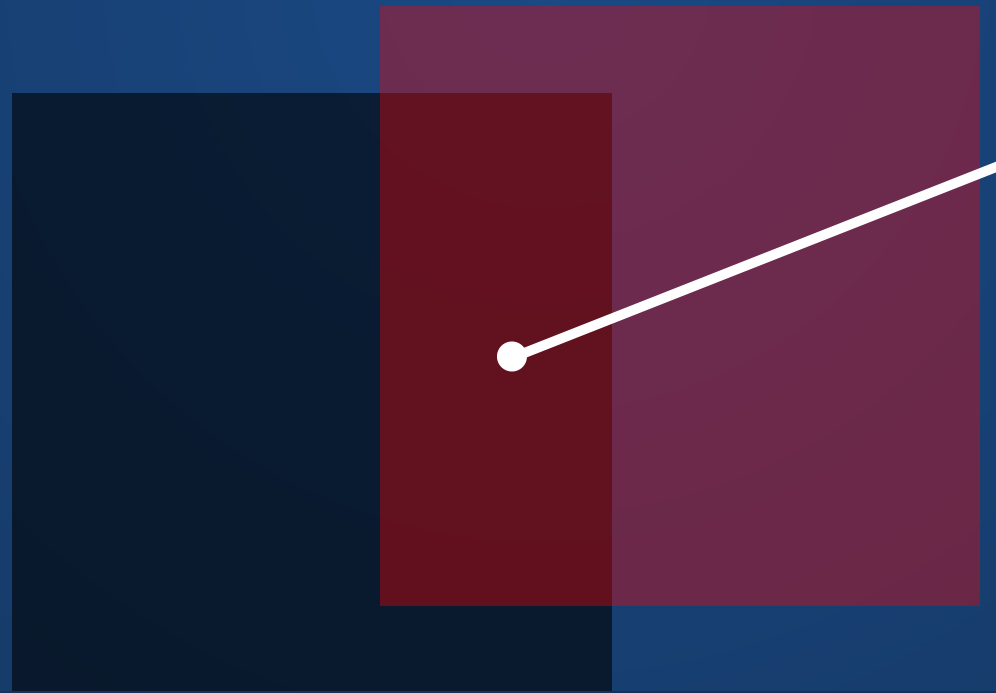
What is this code trying to do?

```
1  void draw(){
2     background(255,0,0);
3     background(0,255,0);
4     background(0,0,255);
5  }
```

Why do we *never* do this?

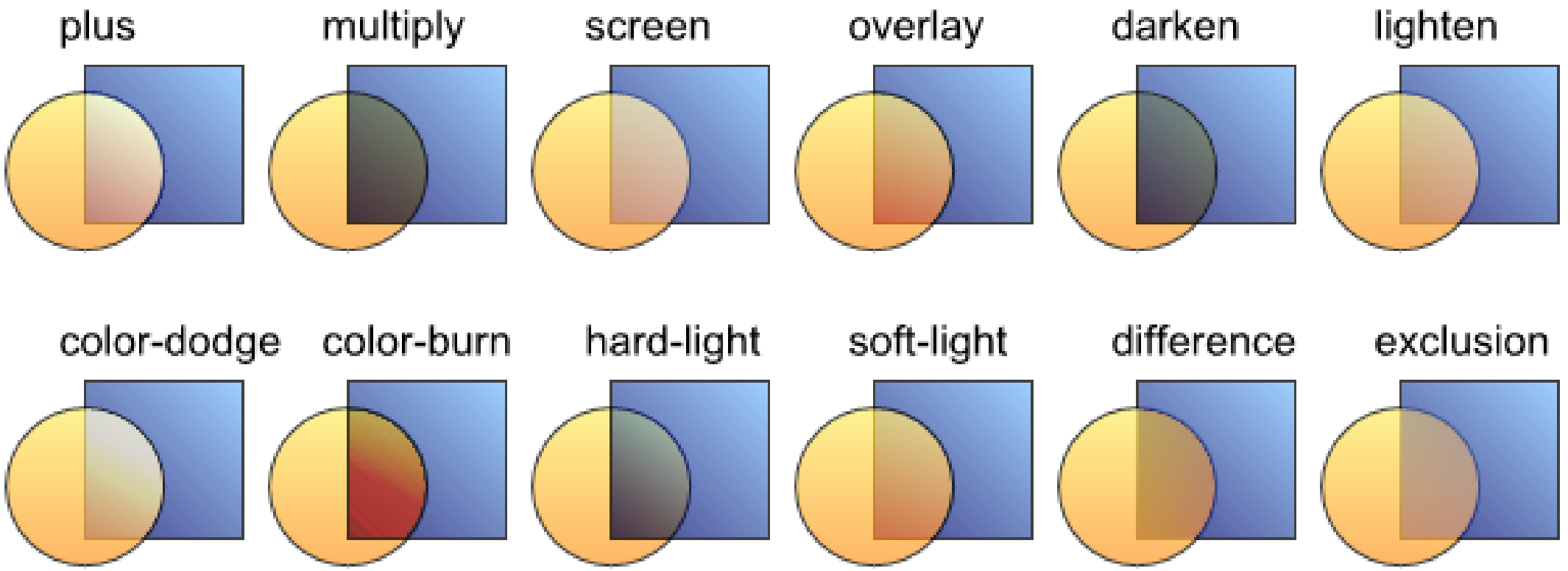
Transparency

- Transparency (alpha) also ranges from 0 to 255
- Allows for on-screen color mixing based on the blend mode.
- Default mode is `blendMode(BLEND)`



What operation is happening here? It isn't an add...

Partially Transparent



Hands-on: Using Color

1. Store some `color` primitives in an array for re-use (make at least 4 colors). Use the color picker (Tools > Color Selector) if you don't have any ideas off the top of your head.

2. Incorporate several of your color primitives into shapes using `fill()` and `stroke()`.

3. Use `blendMode()` to affect color interactions with transparent colors. Show at least one example of how a different `blendMode()` can result in a different behavior. Remember that mode calls only affect the draws that occur **after** them!

Index Cards!

1. Your name and EID.
2. One thing that you learned from class today.
3. One question you have about something covered in class today.
4. (Optional) Any other comments/questions/thoughts about today's class.

It's okay if you didn't learn anything, but you must ask a question.