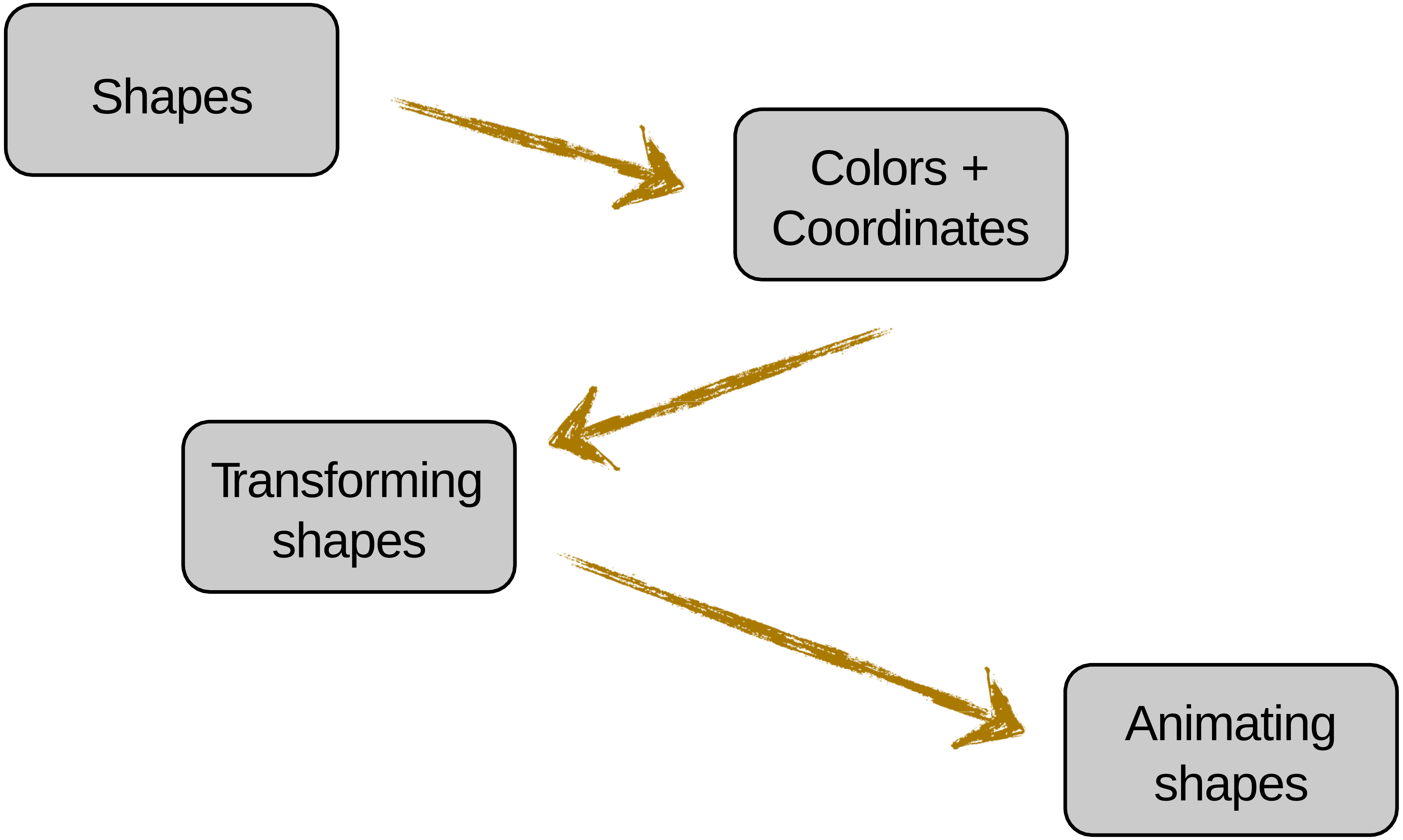
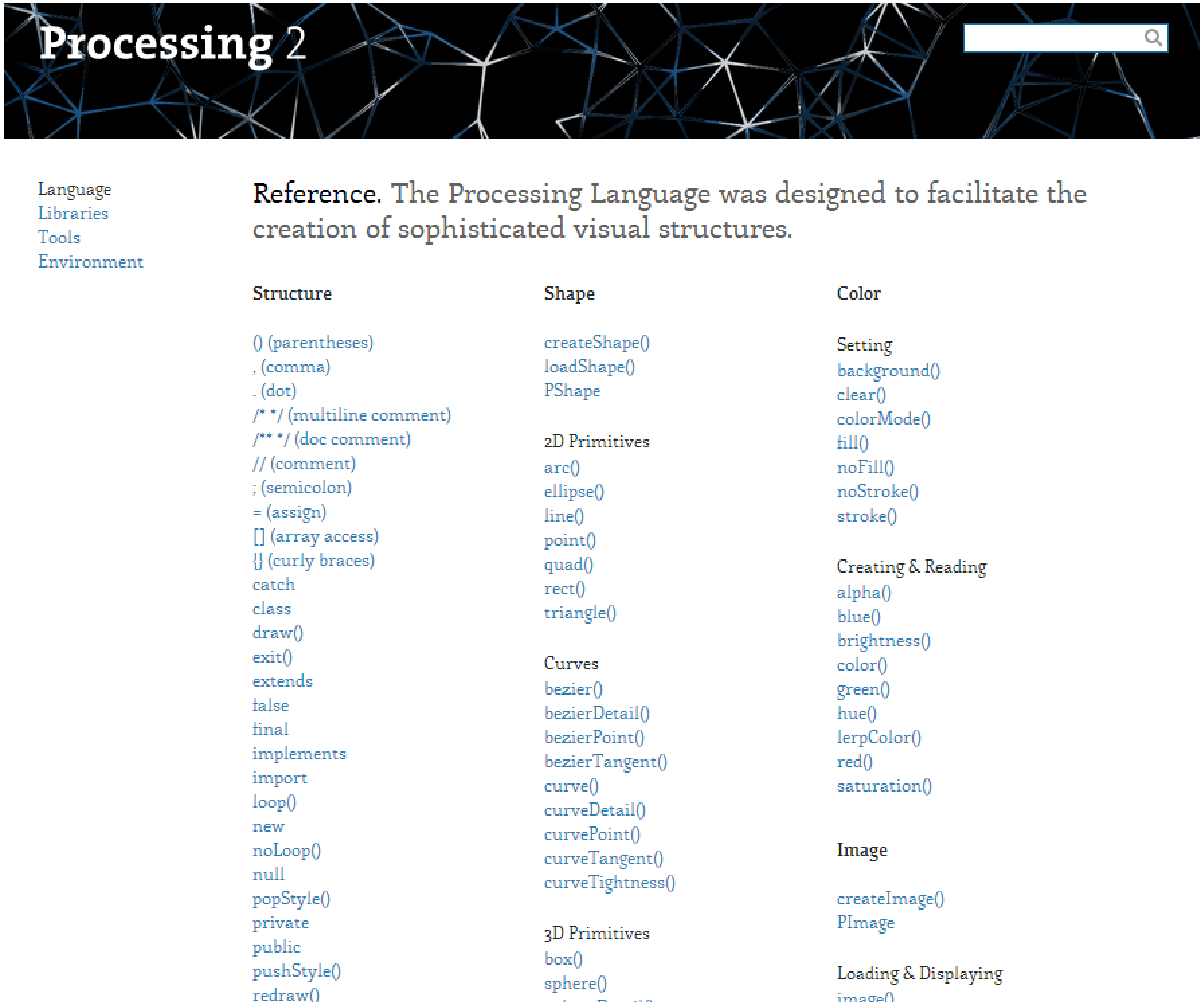
Creative Programming


Shapes
Colors + Coordinates
Transforming shapes
Animating shapes


Processing 2

Language
Libraries
Tools
Environment

Reference. The Processing Language was designed to facilitate the creation of sophisticated visual structures.

Structure

() (parentheses)
, (comma)
. (dot)
/* */ (multiline comment)
/** */ (doc comment)
// (comment)
; (semicolon)
= (assign)
[] (array access)
{} (curly braces)
catch
class
draw()
exit()
extends
false
final
implements
import
loop()
new
noLoop()
null
popStyle()
private
public
pushStyle()
redraw()

Shape

createShape()
loadShape()
PShape

2D Primitives
arc()
ellipse()
line()
point()
quad()
rect()
triangle()

Curves
bezier()
bezierDetail()
bezierPoint()
bezierTangent()
curve()
curveDetail()
curvePoint()
curveTangent()
curveTightness()

3D Primitives
box()
sphere()

Color

Setting
background()
clear()
colorMode()
fill()
noFill()
noStroke()
stroke()

Creating & Reading
alpha()
blue()
brightness()
color()
green()
hue()
lerpColor()
red()
saturation()

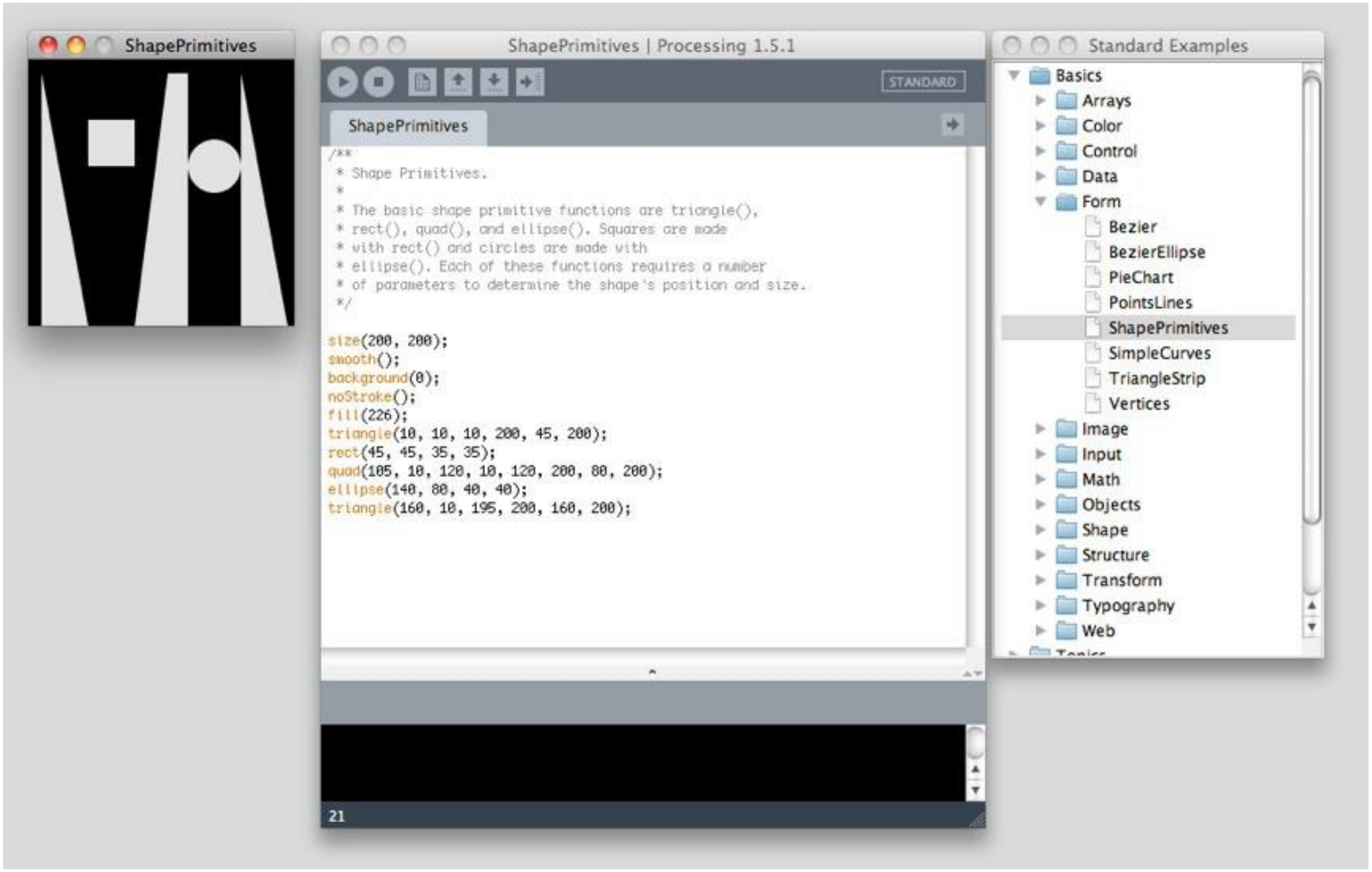
Image

createImage()
PImage

Loading & Displaying

Shapes

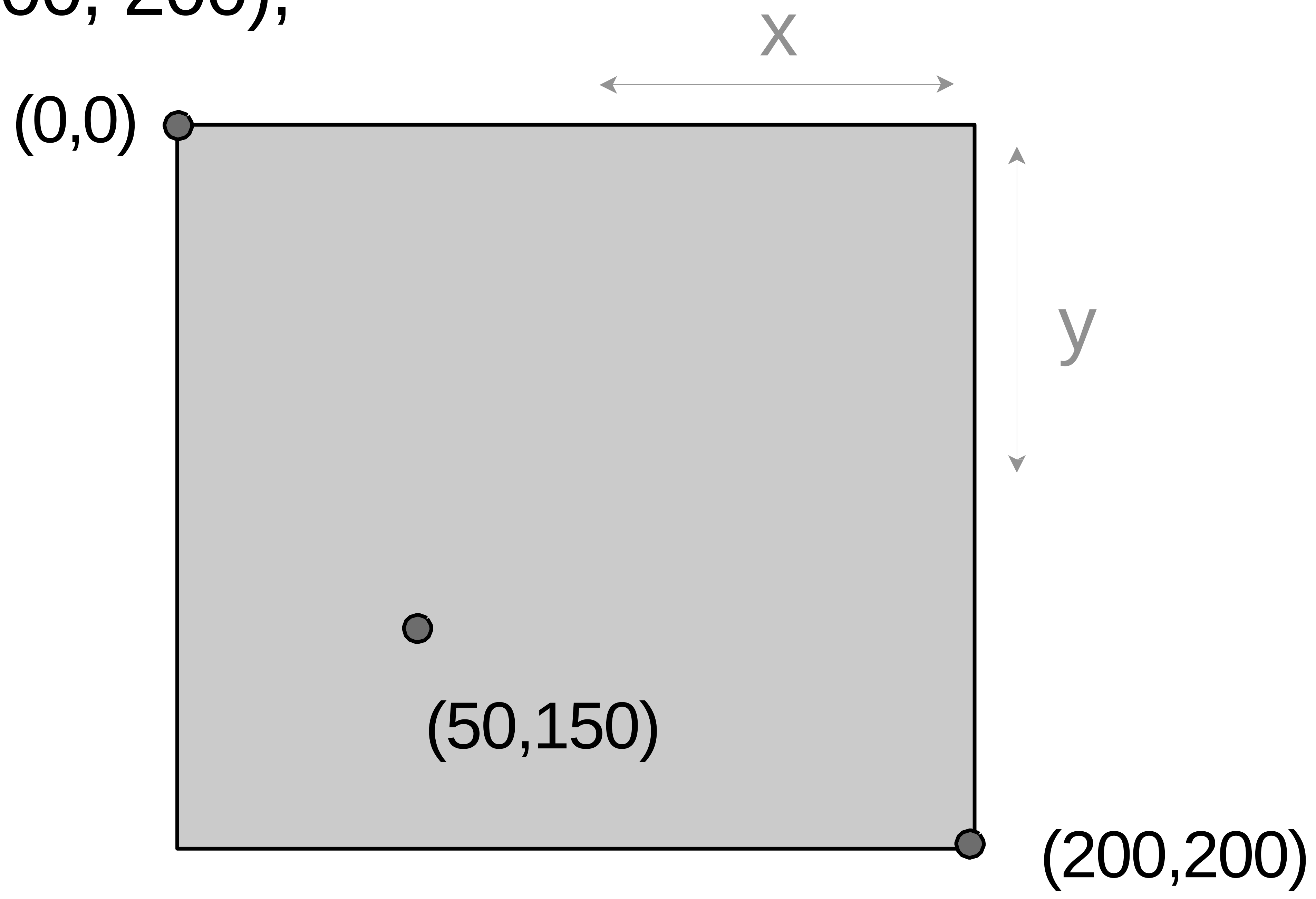
First running sketch



```
/**
 * Shape Primitives.
 *
 * The basic shape primitive functions are triangle(),
 * rect(), quad(), and ellipse(). Squares are made
 * with rect() and circles are made with
 * ellipse(). Each of these functions requires a number
 * of parameters to determine the shape's position and size.
 */

size(200, 200);
smooth();
background(0);
noStroke();
fill(226);
triangle(10, 10, 10, 200, 45, 200);
rect(45, 45, 35, 35);
quad(105, 10, 120, 10, 120, 200, 80, 200);
ellipse(140, 80, 40, 40);
triangle(160, 10, 195, 200, 160, 200);
```

Coordinate system

size(200, 200);



Shapes

- Triangle

```
triangle(x1, y1, x2, y2, x3, y3);
```

- Quad

```
quad(x1, y1, x2, y2, x3, y3, x4, y4);
```

- Rectangle

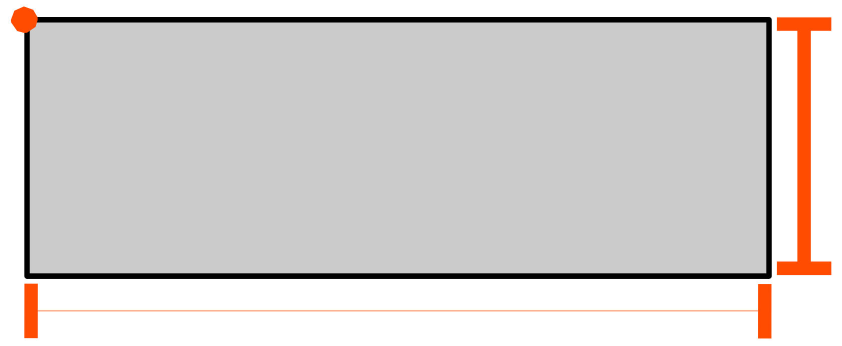
```
rect(x, y, width, height);
```

- Ellipse

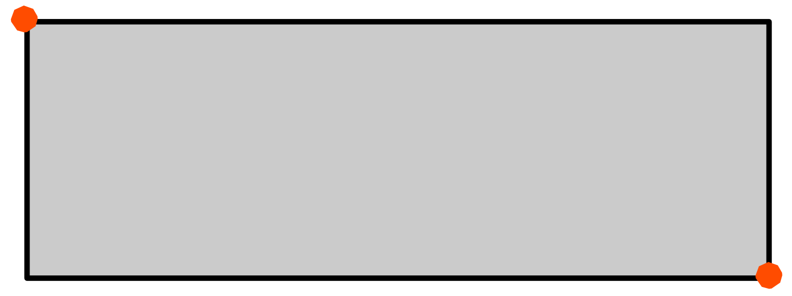
```
ellipse(x, y, width, height);
```

Rectangle Drawing Mode

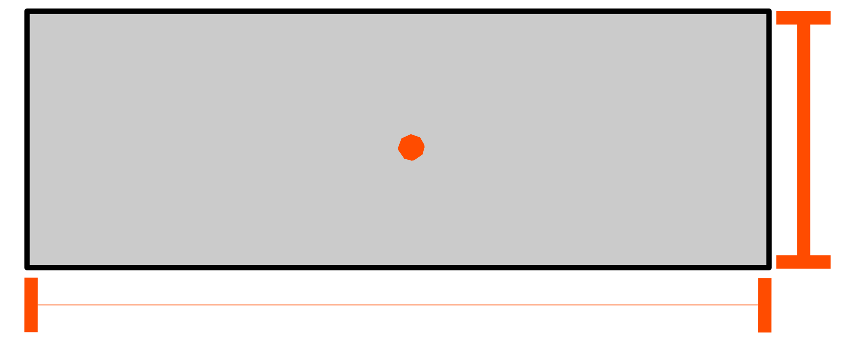
- rectMode(CORNER);



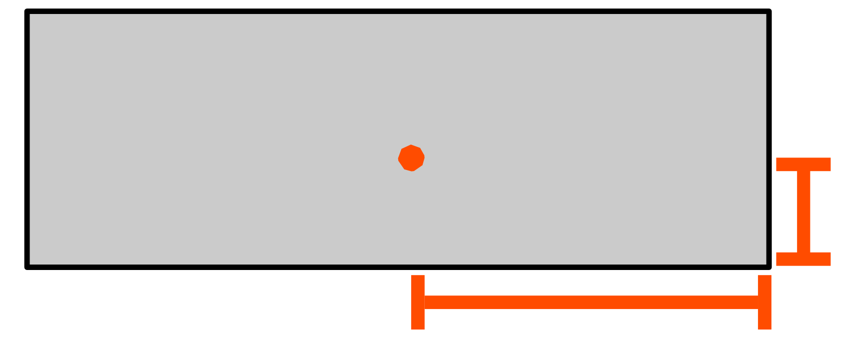
- rectMode(CORNERS);



- rectMode(CENTER);

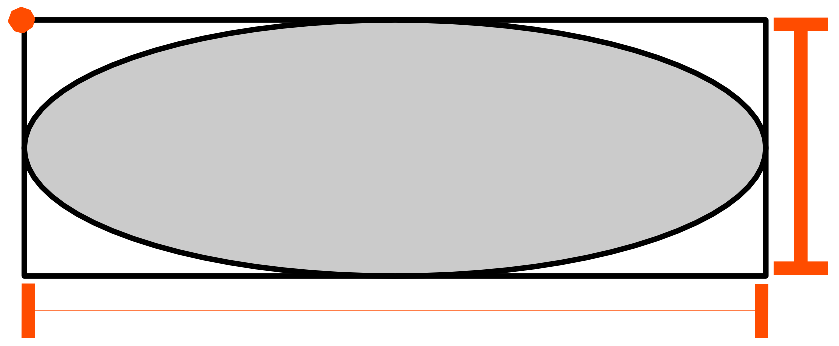


- rectMode(RADIUS);

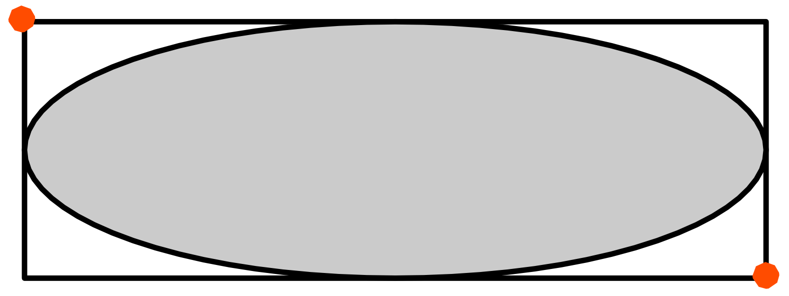


Ellipse Drawing Mode

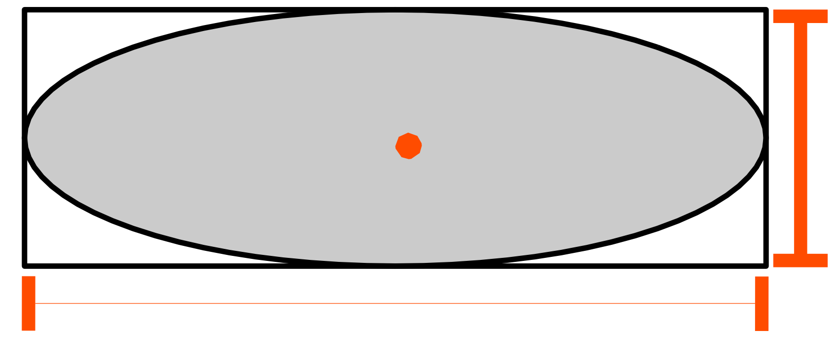
- ellipseMode(CORNER);



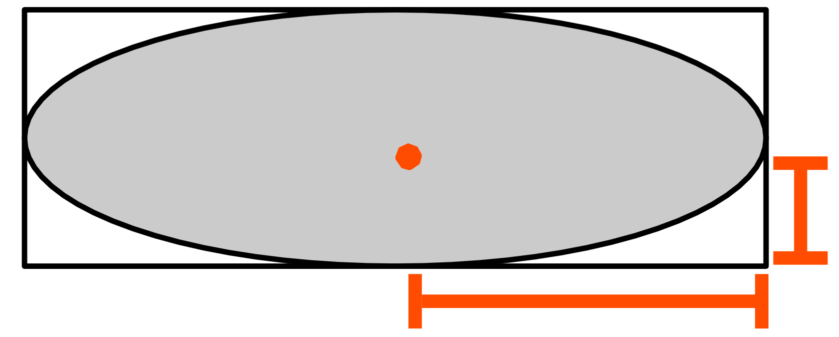
- ellipseMode(CORNERS);



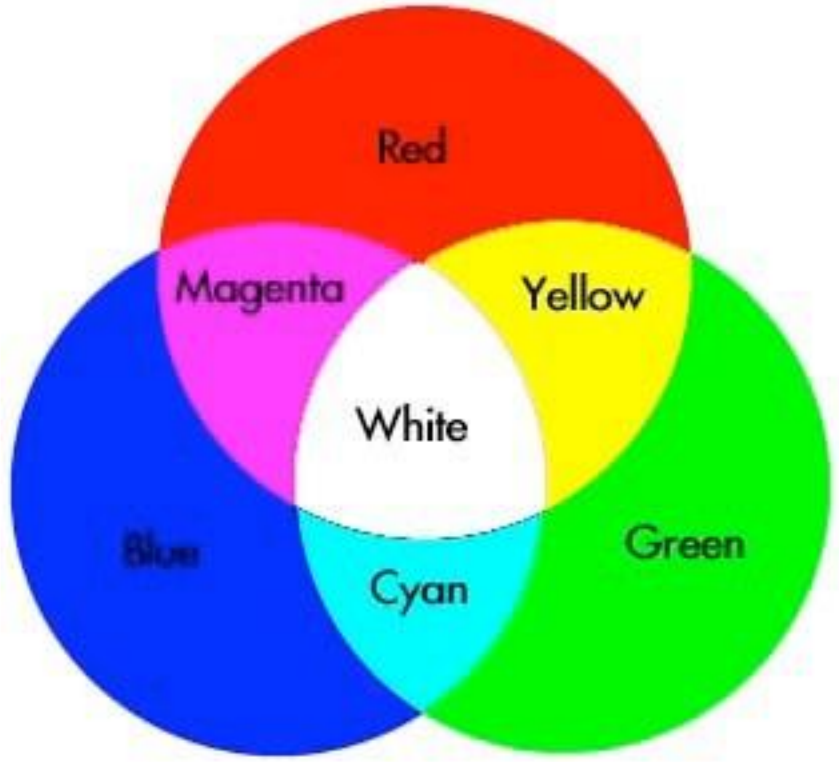
- ellipseMode(CENTER);



- ellipseMode(RADIUS);



Colors



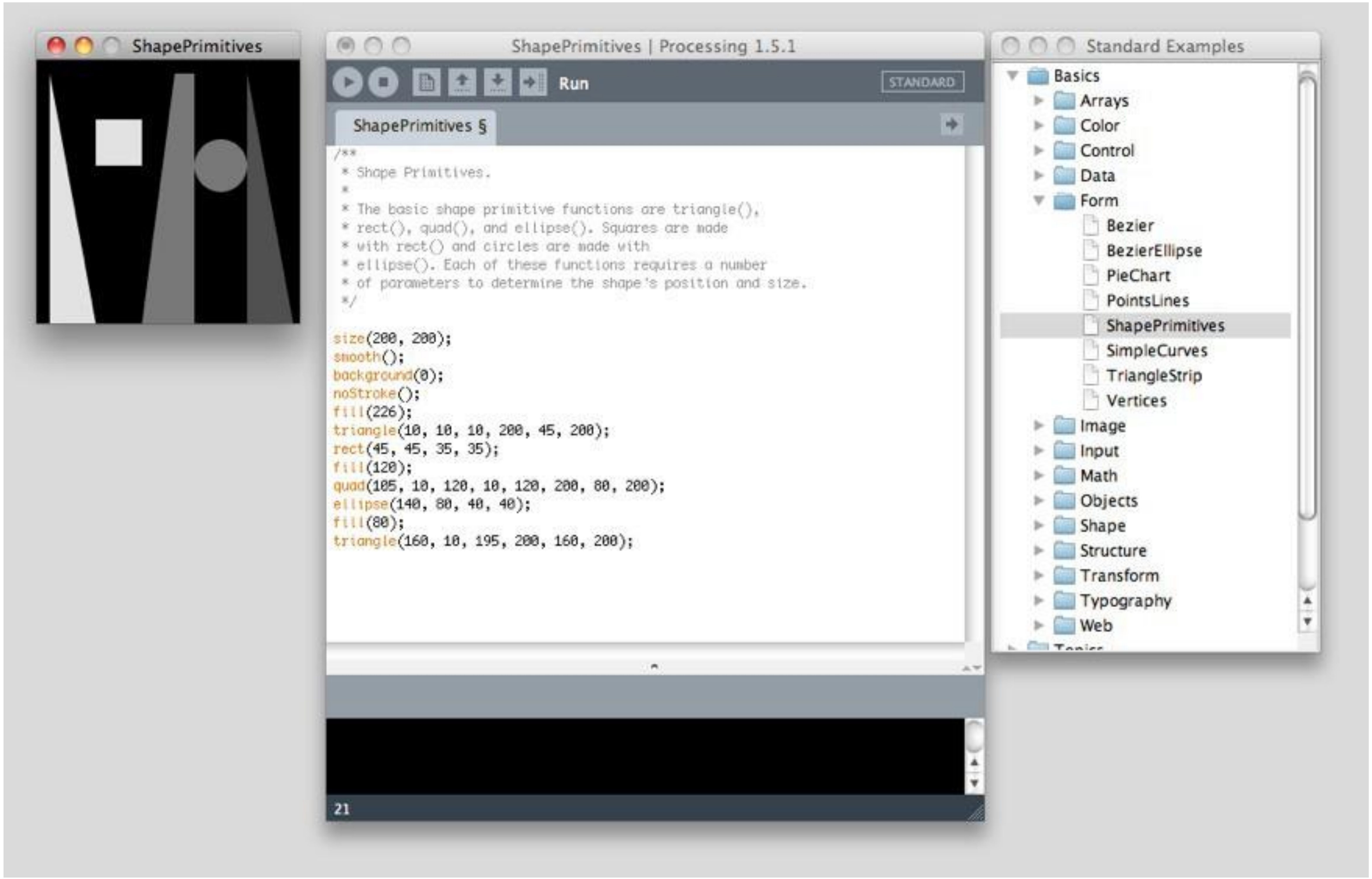
How colors work in Processing

- Color rendering in Processing works in the additive color model: RGB
- fill (<RED>, <GREEN>, <BLUE>); // all values from 0 - 255 possible

- fill(255, 0, 0); // pure red
- fill(0, 0, 130); // dark blue
- How to get yellow?

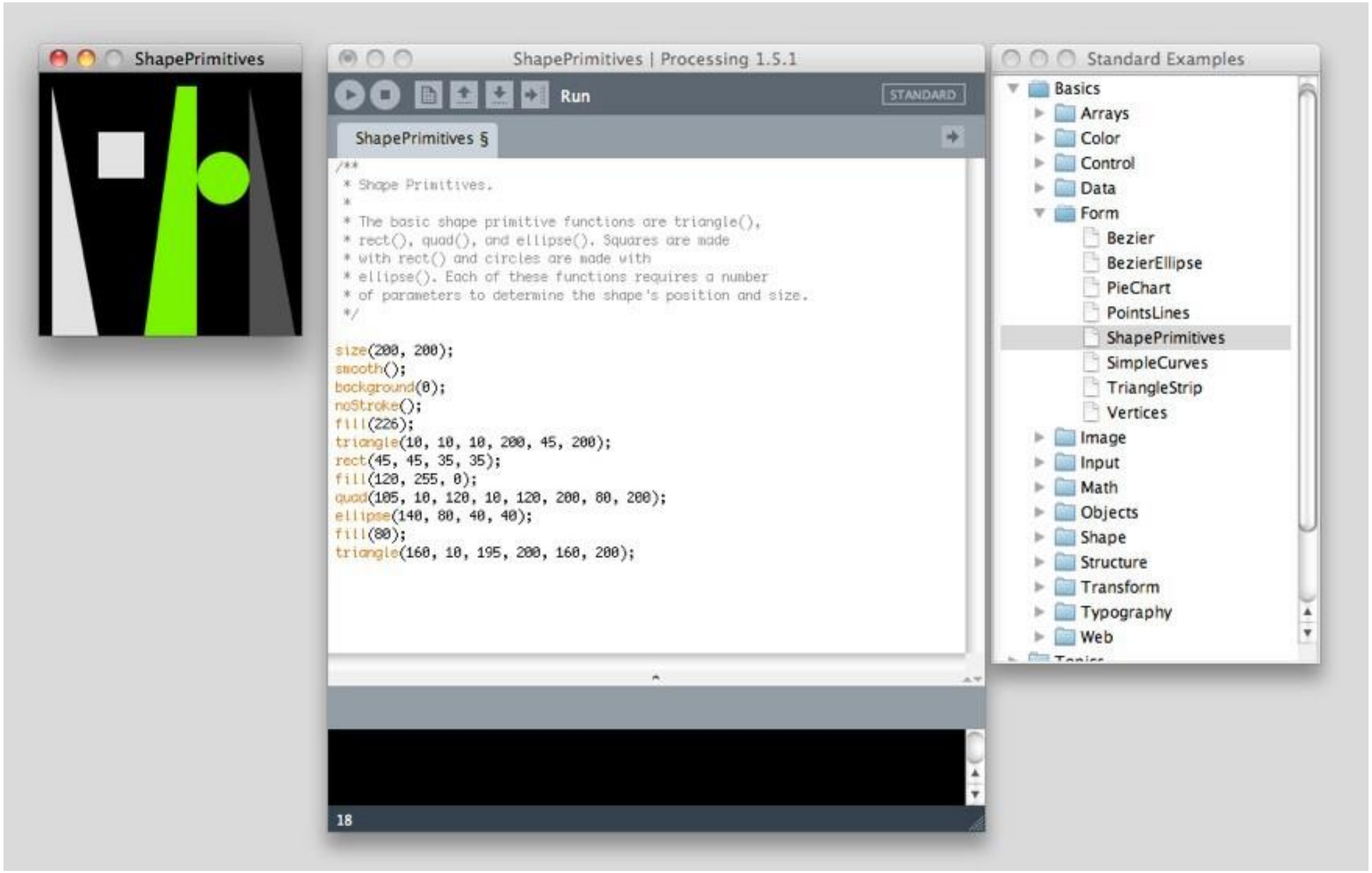
- When all values are same you will get grayscale colors (or white or black).
- “fill(120)” is a shortcut for “fill(120, 120, 120)”

Colors...



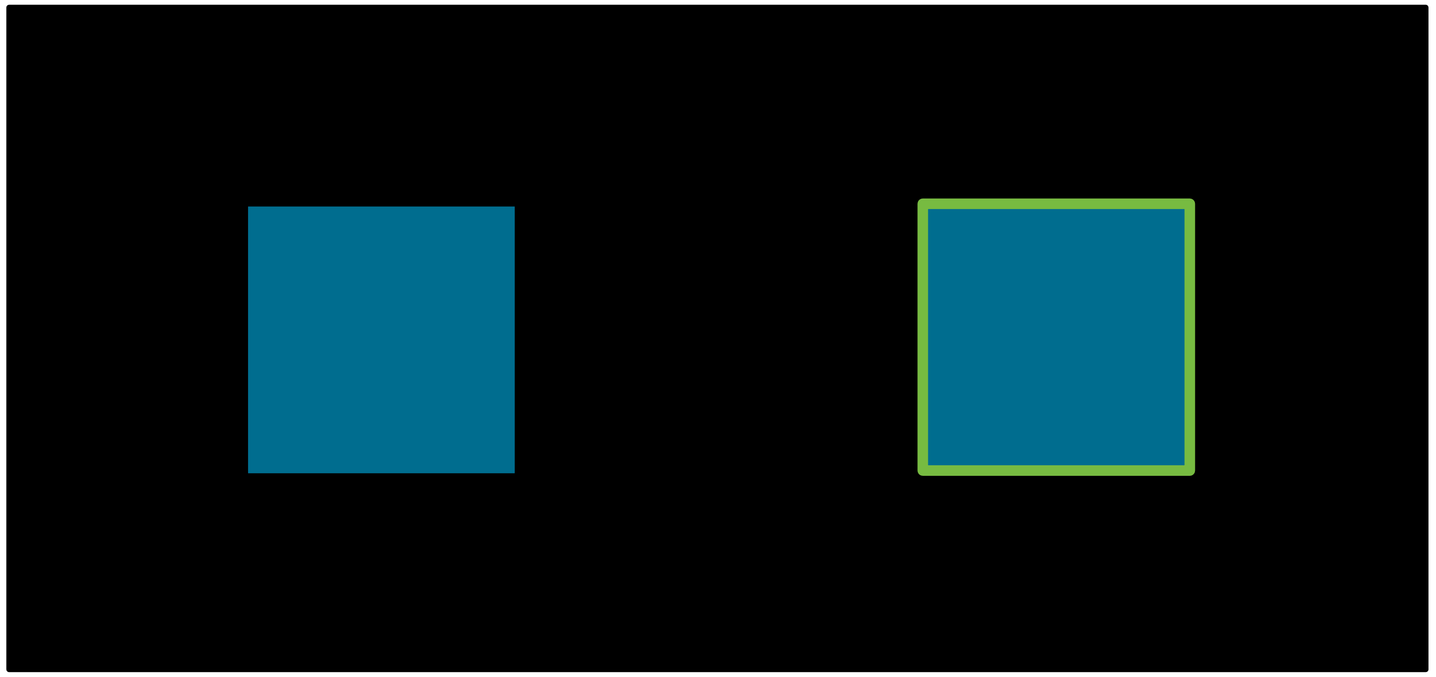
```
/**
 * Shape Primitives.
 *
 * The basic shape primitive functions are triangle(),
 * rect(), quad(), and ellipse(). Squares are made
 * with rect() and circles are made with
 * ellipse(). Each of these functions requires a number
 * of parameters to determine the shape's position and size.
 */

size(200, 200);
smooth();
background(0);
noStroke();
fill(226);
triangle(10, 10, 10, 200, 45, 200);
rect(45, 45, 35, 35);
fill(120);
quad(105, 10, 120, 10, 120, 200, 80, 200);
ellipse(140, 80, 40, 40);
fill(80);
triangle(160, 10, 195, 200, 160, 200);
```

Colors, really



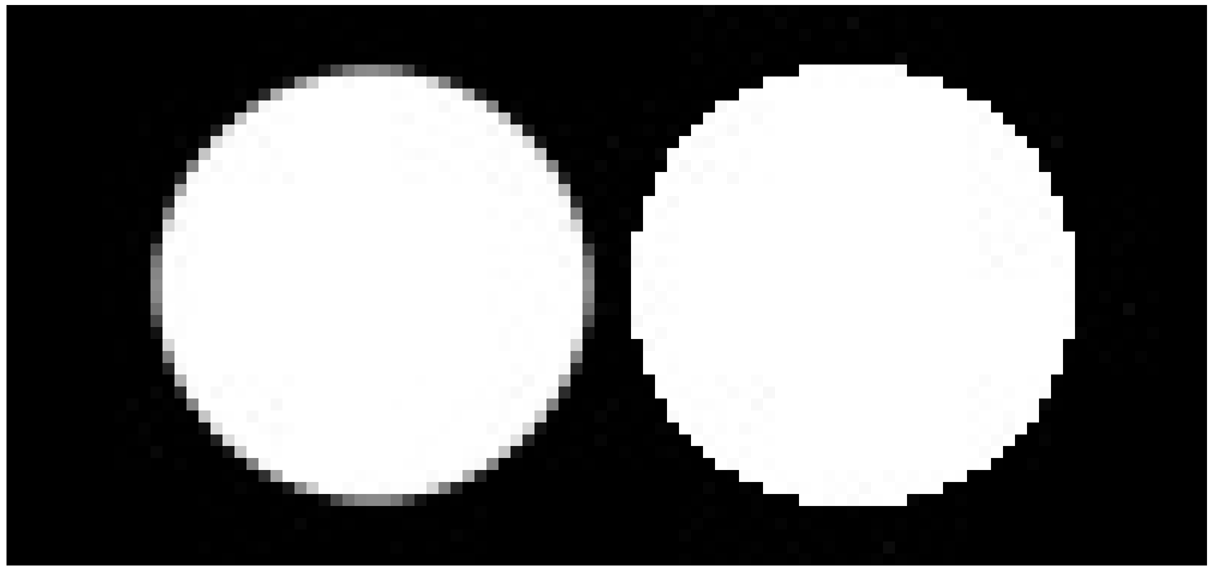
```
/**
 * Shape Primitives.
 *
 * The basic shape primitive functions are triangle(),
 * rect(), quad(), and ellipse(). Squares are made
 * with rect() and circles are made with
 * ellipse(). Each of these functions requires a number
 * of parameters to determine the shape's position and size.
 */

size(200, 200);
smooth();
background(0);
noStroke();
fill(226);
triangle(10, 10, 10, 200, 45, 200);
rect(45, 45, 35, 35);
fill(120, 255, 0);
quad(105, 10, 120, 10, 120, 200, 80, 200);
ellipse(140, 80, 40, 40);
fill(80);
triangle(160, 10, 195, 200, 160, 200);
```

Outline aka Border aka Stroke



```
noStroke();          stroke(0,255,0);
```

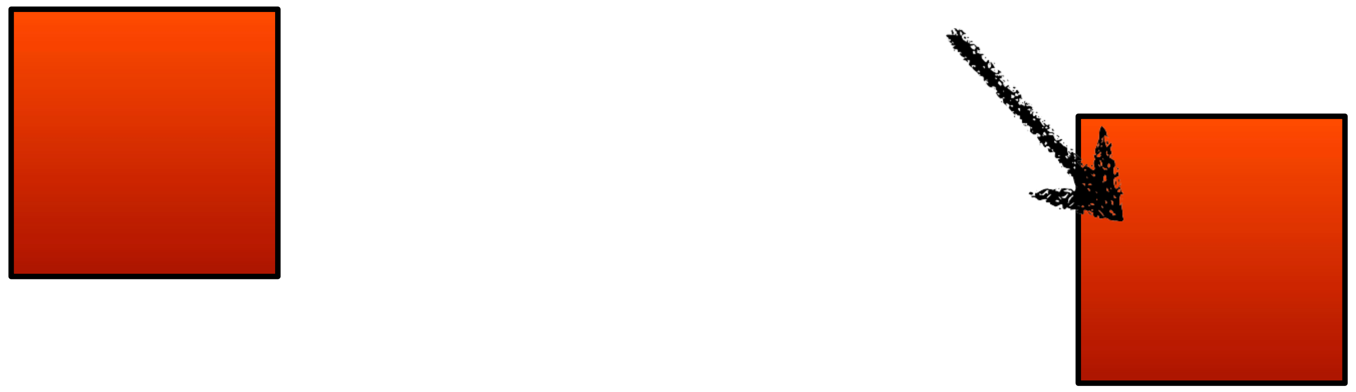
Smoothing aka Anti-Aliasing



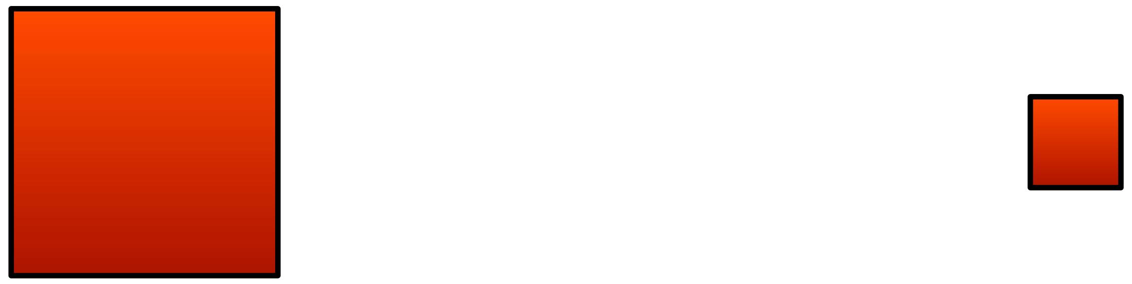
```
smooth();                noSmooth();
```

Transforming shapes

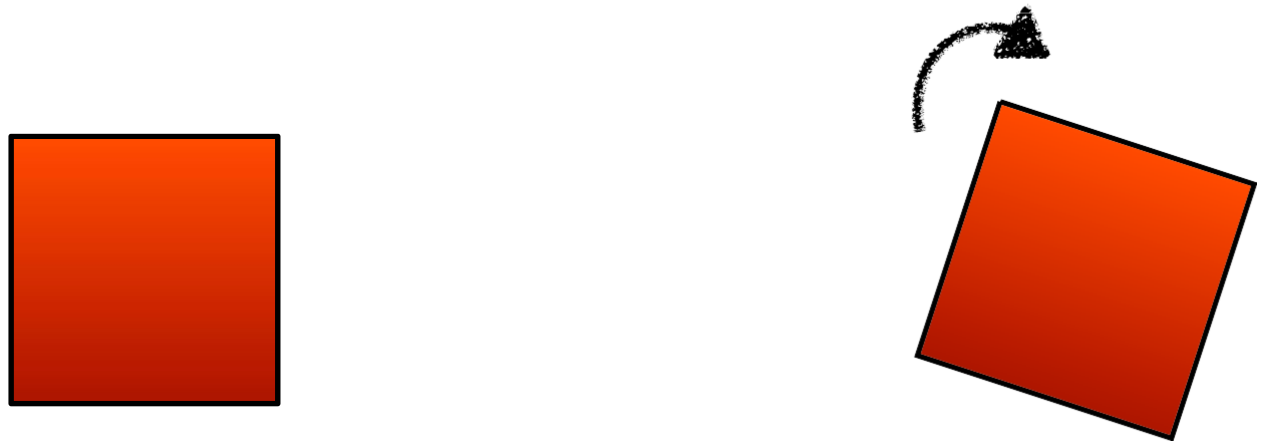
Transformations?



translate

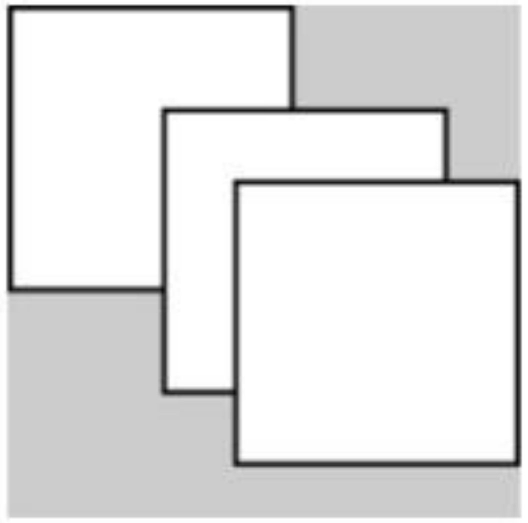


scale



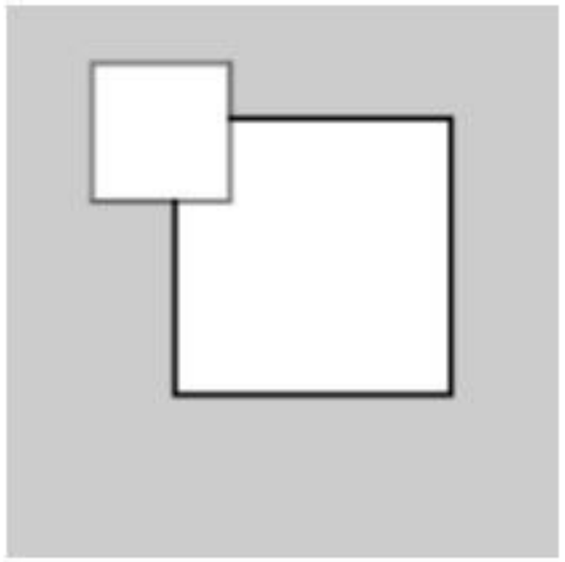
rotate

Translate

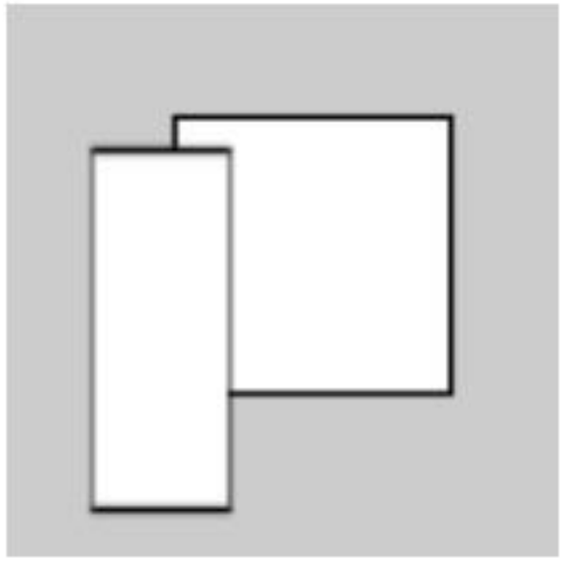


```
rect(0, 0, 55, 55);   // Draw rect at original 0,0
translate(30, 20);
rect(0, 0, 55, 55);   // Draw rect at new 0,0
translate(14, 14);
rect(0, 0, 55, 55);   // Draw rect at new 0,0
```

Scale

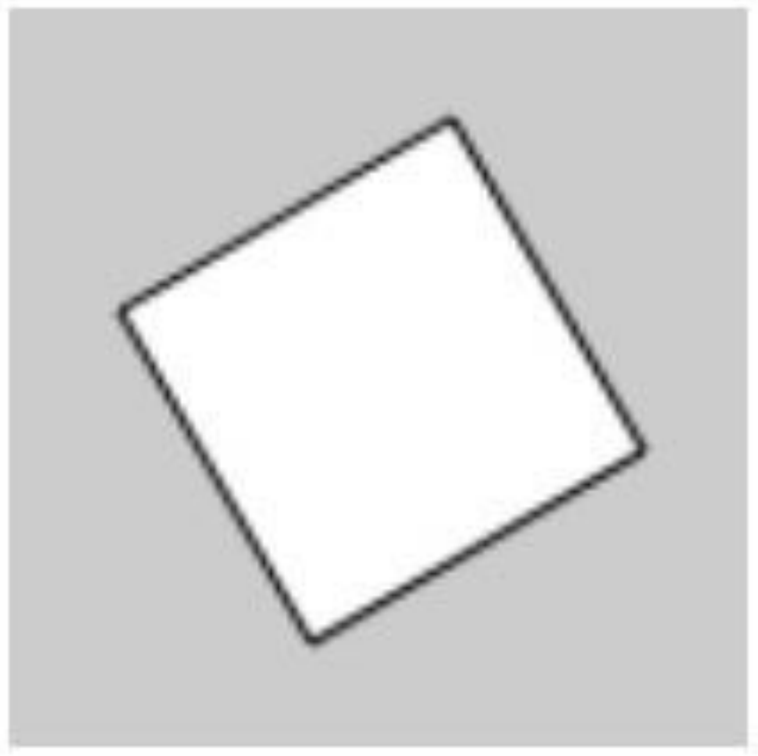


```
rect(30, 20, 50, 50);
scale(0.5);
rect(30, 20, 50, 50);
```



```
rect(30, 20, 50, 50);
scale(0.5, 1.3);
rect(30, 20, 50, 50);
```

Rotate



```
translate(width/2, height/2);
rotate(PI/3.0);
rect(-26, -26, 52, 52);
```

Hint: rotate(radians(30));

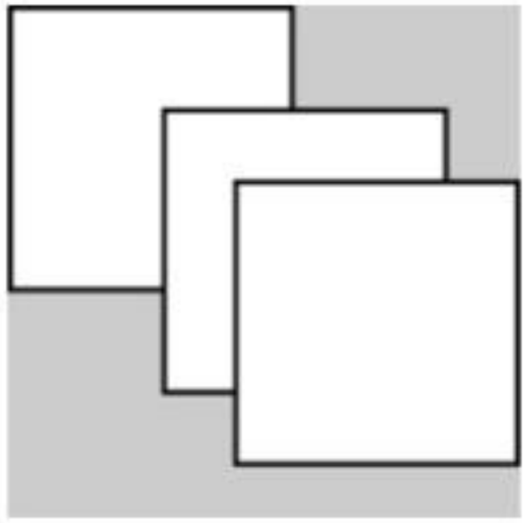
Math !

$$\begin{pmatrix} 1 & 0 & p \\ 0 & 1 & q \\ 0 & 0 & 1 \end{pmatrix} . \begin{pmatrix} x \\ y \\ 1 \end{pmatrix} = \begin{pmatrix} x + p \\ y + q \\ 1 \end{pmatrix}$$

$$\begin{pmatrix} L & 0 & 0 \\ 0 & L & 0 \\ 0 & 0 & 1 \end{pmatrix} . \begin{pmatrix} x \\ y \\ 1 \end{pmatrix} = \begin{pmatrix} x * L \\ y * L \\ 1 \end{pmatrix}$$

$$\begin{pmatrix} \cos\alpha & -\sin\alpha & 0 \\ \sin\alpha & \cos\alpha & 0 \\ 0 & 0 & 1 \end{pmatrix} . \begin{pmatrix} x \\ y \\ 1 \end{pmatrix} = \begin{pmatrix} \cos\alpha\, x - \sin\alpha\, y \\ \sin\alpha\, x + \cos\alpha\, y \\ 1 \end{pmatrix}$$

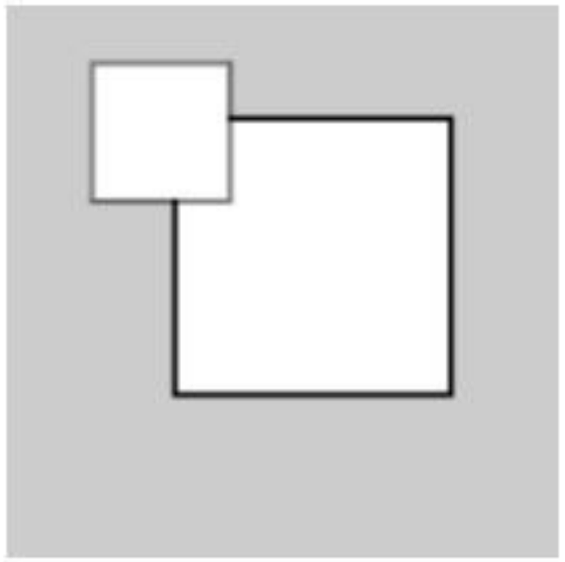
Translate



```
rect(0, 0, 55, 55);   // Draw rect at original 0,0
translate(30, 20);
rect(0, 0, 55, 55);   // Draw rect at new 0,0
translate(14, 14);
rect(0, 0, 55, 55);   // Draw rect at new 0,0
```

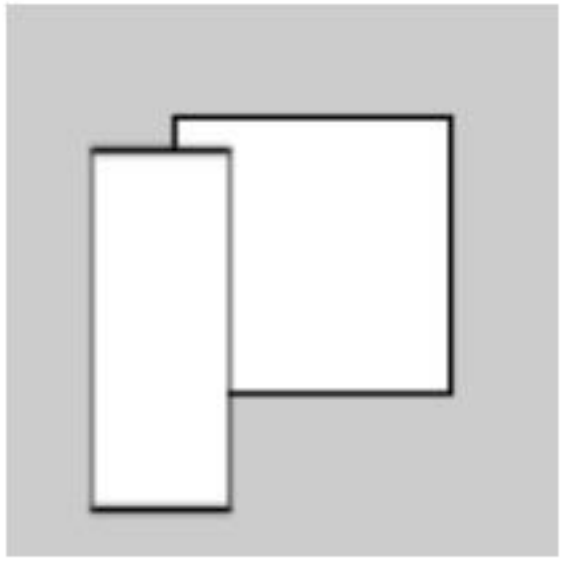
$$\begin{pmatrix} 1 & 0 & p \\ 0 & 1 & q \\ 0 & 0 & 1 \end{pmatrix} \cdot \begin{pmatrix} x \\ y \\ 1 \end{pmatrix} = \begin{pmatrix} x + p \\ y + q \\ 1 \end{pmatrix}$$

Scale



```
rect(30, 20, 50, 50);
scale(0.5);
rect(30, 20, 50, 50);
```

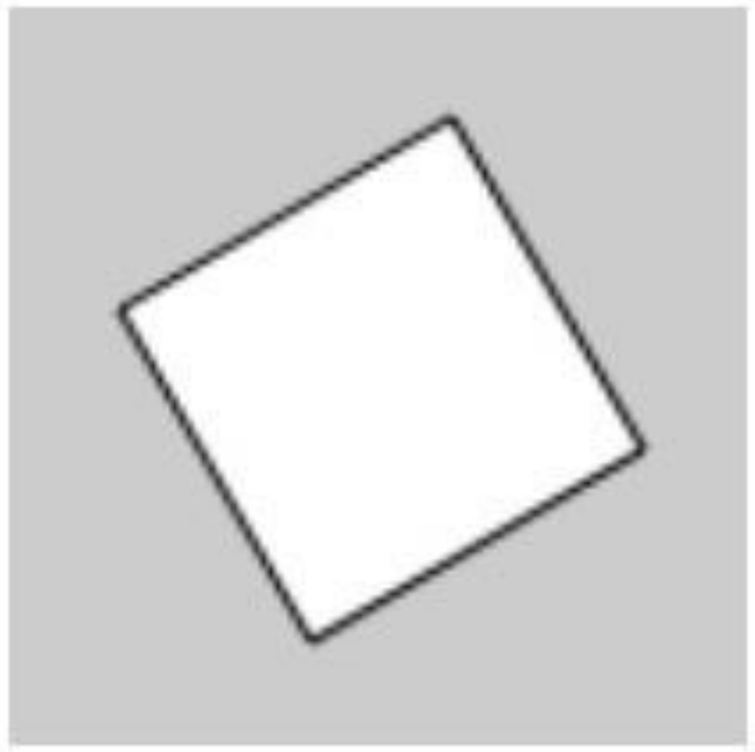
$$\begin{pmatrix} L & 0 & 0 \\ 0 & L & 0 \\ 0 & 0 & 1 \end{pmatrix} . \begin{pmatrix} x \\ y \\ 1 \end{pmatrix} = \begin{pmatrix} x * L \\ y * L \\ 1 \end{pmatrix}$$



```
rect(30, 20, 50, 50);
scale(0.5, 1.3);
rect(30, 20, 50, 50);
```

$$\begin{pmatrix} L & 0 & 0 \\ 0 & M & 0 \\ 0 & 0 & 1 \end{pmatrix} . \begin{pmatrix} x \\ y \\ 1 \end{pmatrix} = \begin{pmatrix} x * L \\ y * M \\ 1 \end{pmatrix}$$

Rotate

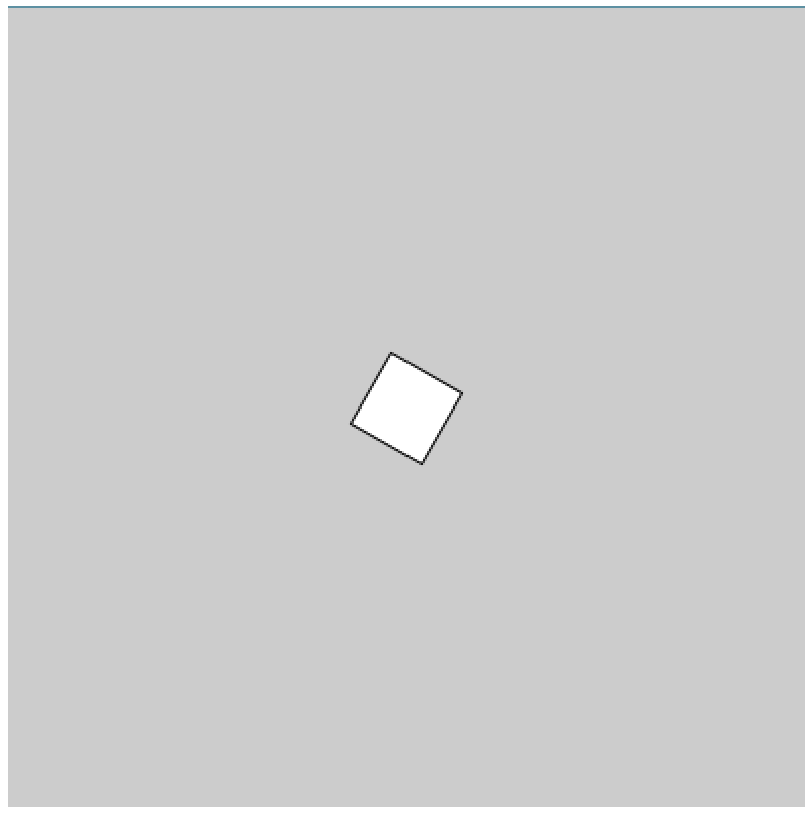


```
translate(width/2, height/2);
rotate(PI/3.0);
rect(-26, -26, 52, 52);
```

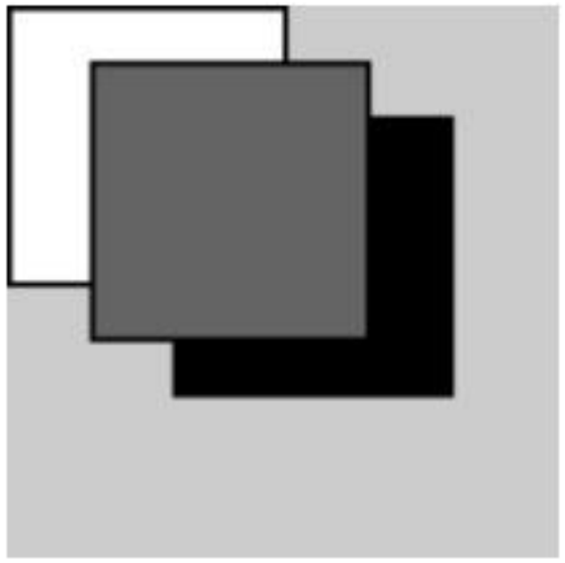
$$\begin{pmatrix} \cos\alpha & -\sin\alpha & 0 \\ \sin\alpha & \cos\alpha & 0 \\ 0 & 0 & 1 \end{pmatrix} . \begin{pmatrix} x \\ y \\ 1 \end{pmatrix} = \begin{pmatrix} \cos\alpha\, x - \sin\alpha\, y \\ \sin\alpha\, x + \cos\alpha\, y \\ 1 \end{pmatrix}$$

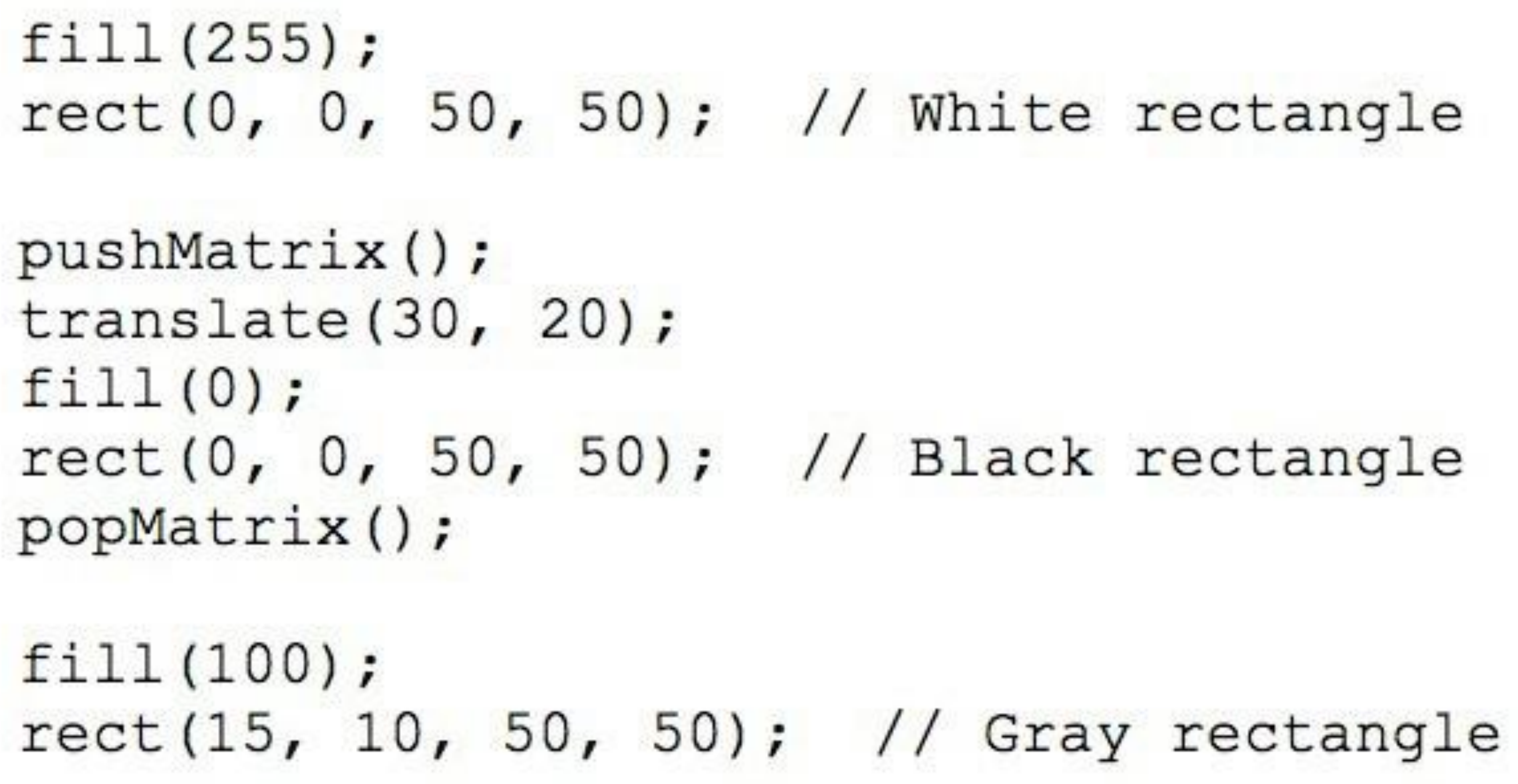
Hint: rotate(radians(30));

```
size(400,400);
translate(200,200);
rotate(radians(30));
rect(-20,-20,40,40);
```



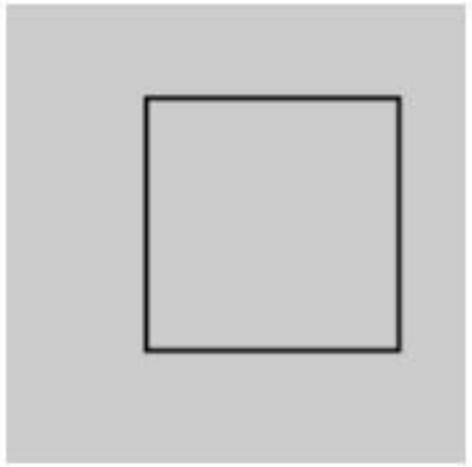
Transformation UNDO



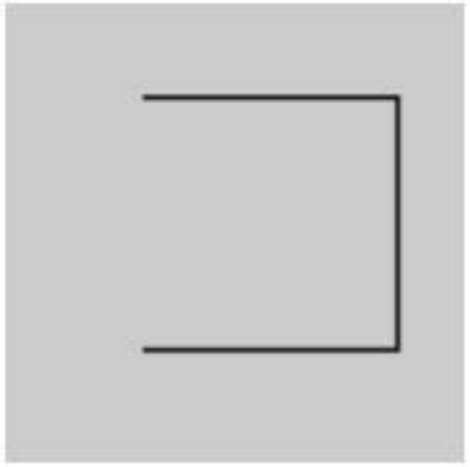


```
fill(255);
rect(0, 0, 50, 50);  // White rectangle

pushMatrix();
translate(30, 20);
fill(0);
rect(0, 0, 50, 50);  // Black rectangle
popMatrix();

fill(100);
rect(15, 10, 50, 50);  // Gray rectangle
```

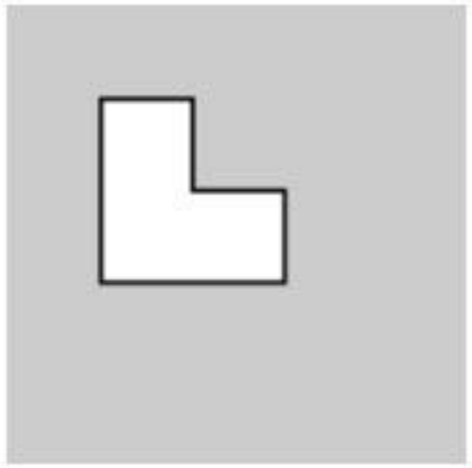
Polygon shapes



```
noFill();
beginShape();
vertex(30, 20);
vertex(85, 20);
vertex(85, 75);
vertex(30, 75);
endShape(CLOSE);
```



```
noFill();
beginShape();
vertex(30, 20);
vertex(85, 20);
vertex(85, 75);
vertex(30, 75);
endShape();
```



```
beginShape();
vertex(20, 20);
vertex(40, 20);
vertex(40, 40);
vertex(60, 40);
vertex(60, 60);
vertex(20, 60);
endShape(CLOSE);
```

Dynamics – animating shapes

```
void setup( ) {
  size(200, 200);
}
```

just once
on start up

```
void draw( ) {
  // erase background
  background(0);
  // draw some stuff
  // …
}
```

every frame,
this happens

by the way: every frame starts without any transformations

```
// declare variable and set start value
int x = 0;
```

```
void setup( ){
    size(400, 400);
}
```

```
void draw( ){
    // erase background
    background(0);
    // add 1 to variable
    x = x + 1;
    // draw a rectangle of 20 by 20 pixels
    rect(x, x, 20, 20);
}
```

just once
on start up

every frame
this happens

Challenge

Rotate a rectangle around the center of the stage.