15729: Creating Generative Art with Code

Splash Fall 2023

Lauren Chua, Samuel Figueroa,
Nicholas Sbalbi, Xiaoqi Sun

What is Generative Art?

"**Generative art** refers to art that in whole or in part has been created with the use of an autonomous system"
- Wikipedia

Jonathan Chaffer, 2021

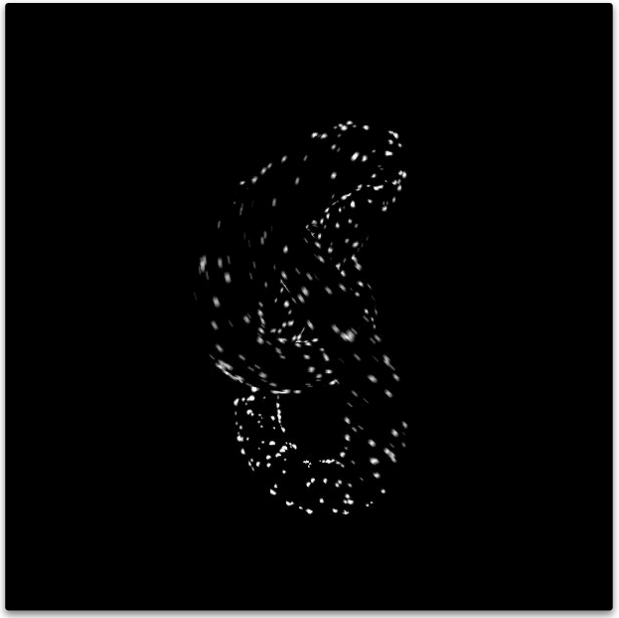
What is Processing?

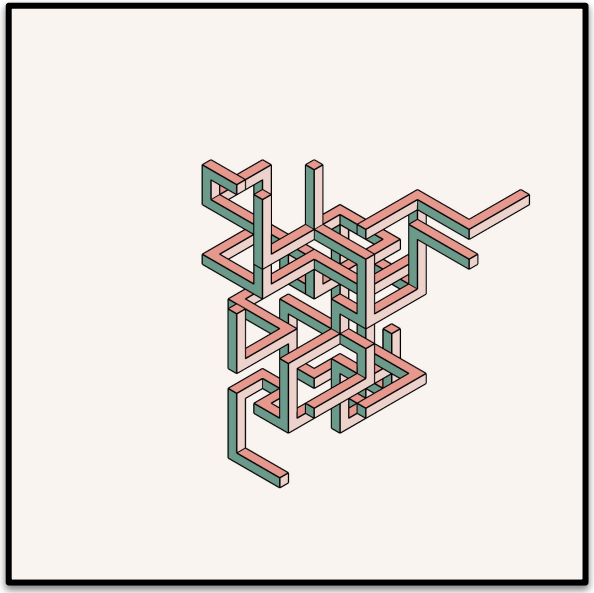
Processing was released by two graduate students in the MIT Media Lab in 2001

"…a flexible software sketchbook and a language for learning how to code"

processing.org hosts a number of examples and detailed, accessible documentation (Java-based)

What You Can Make?

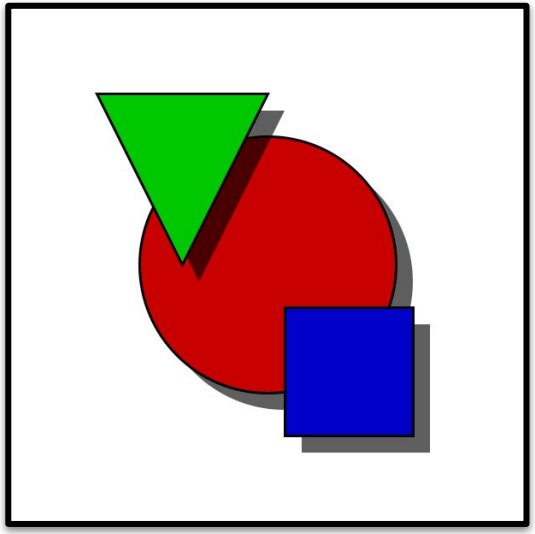




What We'll Be Making

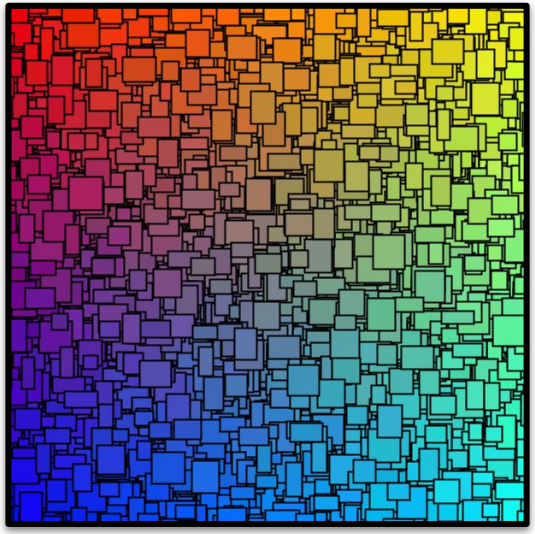
Section 1

- Shape Primitives
- Borders and Fill
- Colors



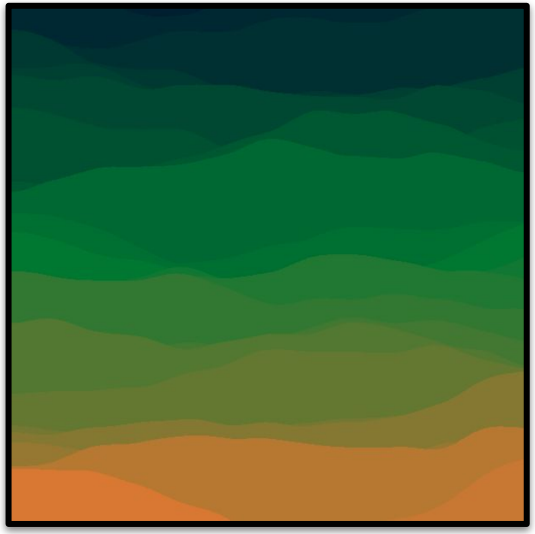
Section 2

- Randomness
- Repetition
- Map()



Section 3

- Noise
- Nesting Loops
- If/Else Logic



What We'll Be Making

Section 1

- Shape Primitives
- Borders and Fill
- Colors

Section 2

- Randomness
- Repetition
- Map()

Section 3

- Noise
- Nesting Loops
- If/Else Logic

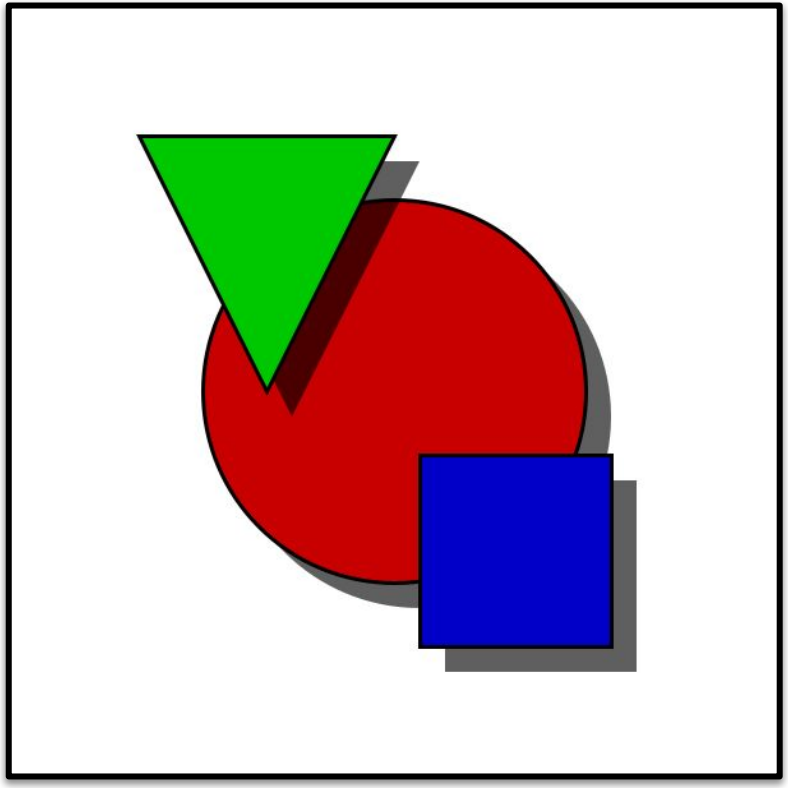
Some Logistics

1. **Class Files** - Emailed out but also available at: https://github.com/nsbalbi/Splash-2023-C15729
   a. template.pde - Good starting point, make copies!
   b. splash_section1.pde - Finished example for each section
2. **Processing Install** - https://processing.org/download
3. **Class Structure Overview** - 3 Sections

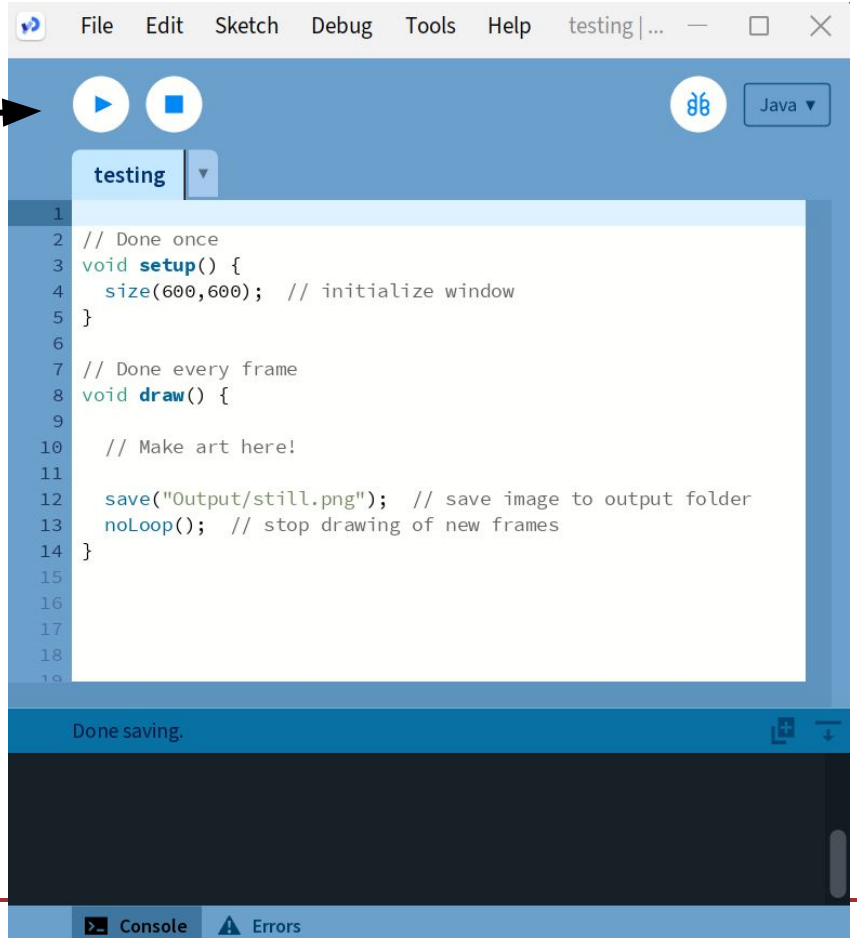
~15 min "Lecture"	~15 min "Free Coding"
Walkthrough of concepts and an example	Time to make whatever you'd like!

What We'll Cover

- Shape Primitives
- Coordinates
  - `width` and `height`
- Colors
- `stroke()` and `fill()`
- Drawing Order
- Alpha



Section 1: The Interface



Run script

Code/Script

Console/Errors

Initialization

Make Art Here

Section 1: `circle()`

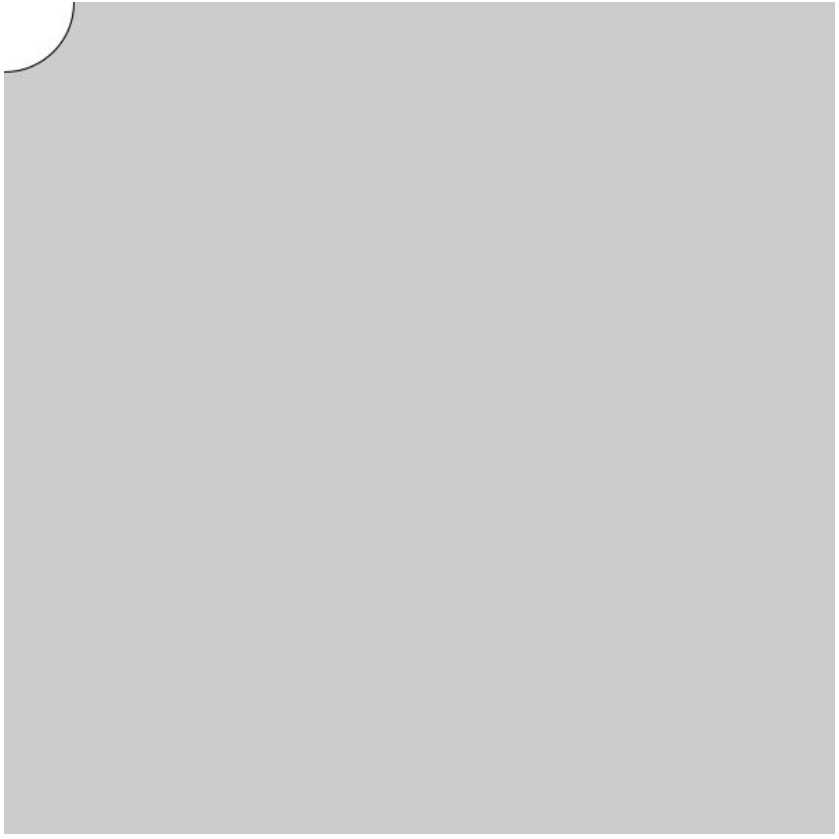
Syntax `circle(x, y, extent)`

Parameters

x	(float)	x-coordinate of the ellipse
y	(float)	y-coordinate of the ellipse
extent	(float)	width and height of the ellipse by default

```
1
2  // Done once
3  void setup() {
4    size(600,600);  // initialize window
5  }
6
7  // Done every frame
8  void draw() {
9
10   circle(0, 0, 100);
11
12   save("Output/still.png");  // save image
13   noLoop();  // stop drawing of new frames
14 }
15
```

In Processing, every line of code must end with a semicolon



Section 1: Coordinates

Syntax `circle(x, y, extent)`

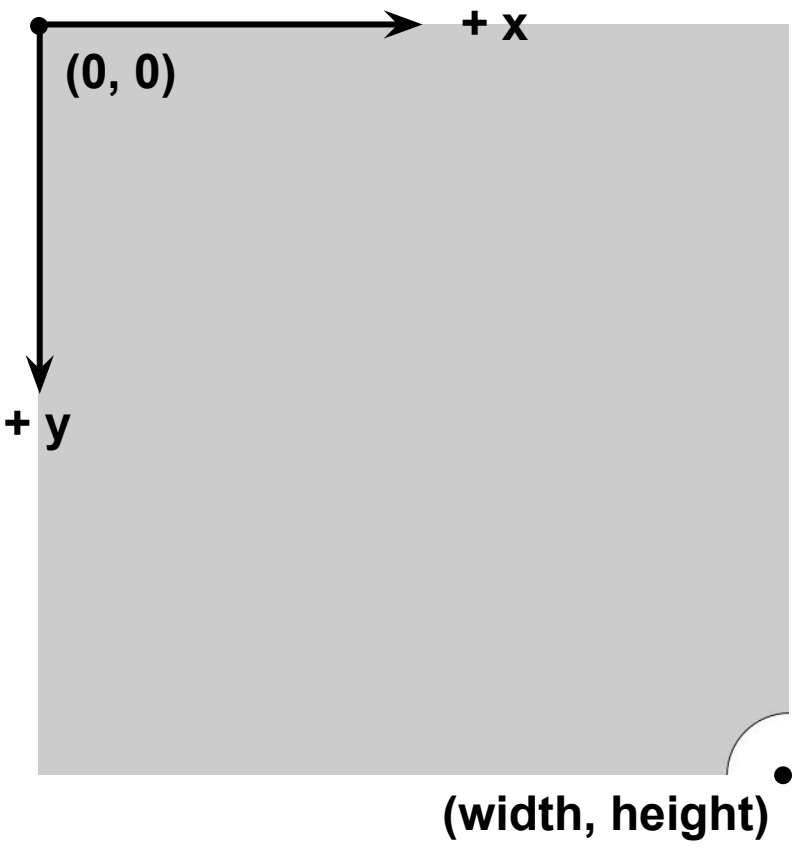
Parameters

x	(float)	x-coordinate of the ellipse
y	(float)	y-coordinate of the ellipse
extent	(float)	width and height of the ellipse by default

```
1
2  // Done once
3  void setup() {
4    size(600,600);  // initialize window
5  }
6
7  // Done every frame
8  void draw() {
9
10   circle(600, 600, 100);
11
12   save("Output/still.png");  // save image
13   noLoop();  // stop drawing of new frames
14 }
15
```

width

height



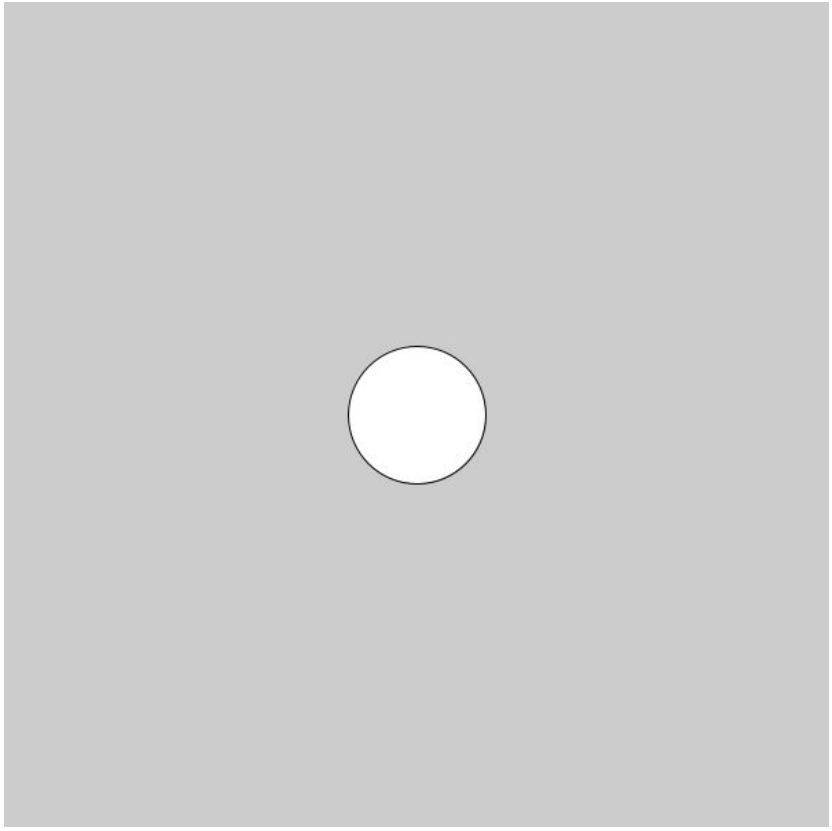
Section 1: `Width` and `Height`

Syntax `circle(x, y, extent)`

Parameters

x	(float)	x-coordinate of the ellipse
y	(float)	y-coordinate of the ellipse
extent	(float)	width and height of the ellipse by default

```
1
2  // Done once
3  void setup() {
4    size(600,600);  // initialize window
5  }
6
7  // Done every frame
8  void draw() {
9
10   circle(width/2, height/2, 100);
11
12   save("Output/still.png");  // save image
13   noLoop();  // stop drawing of new frames
14 }
15
```



Processing uses RGB colors by default

(R, G, B)

R/G/B varies from 0 to 255

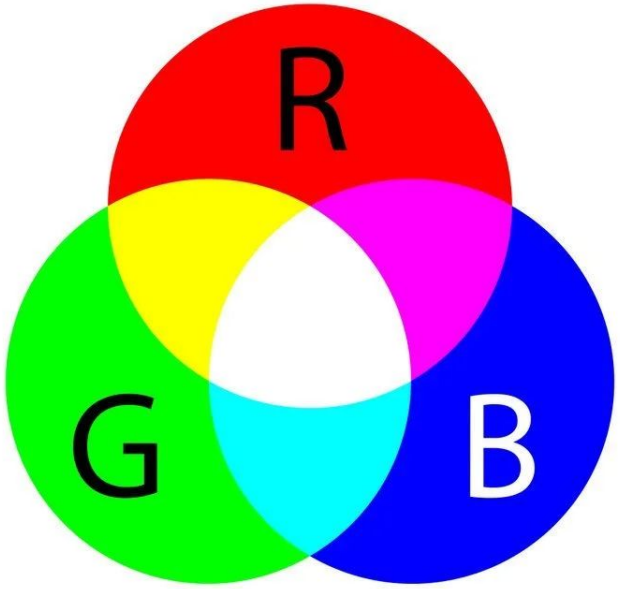
(255, 0, 0)

(0, 255, 0)

(0, 0, 255)

(0, 0, 0)

(255, 255, 255)



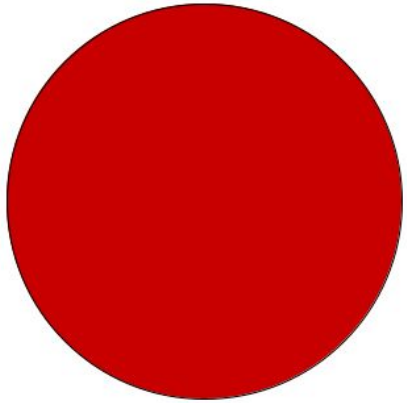
Google - “RGB Color Picker”

Section 1: `background()` and `fill()`

`background()` sets the color of the background

`fill()` sets the color that fills the shape

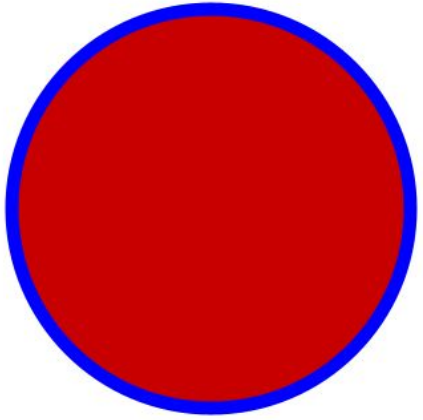
```
7  // Done every frame
8  void draw() {
9
10   background(255, 255, 255);
11
12   fill(200, 0, 0);
13   circle(width/2, height/2, 300);
14
15   save("Output/still.png");  // save image
16   noLoop();  // stop drawing of new frames
17 }
18
```



Section 1: `stroke()` and `strokeWeight()`

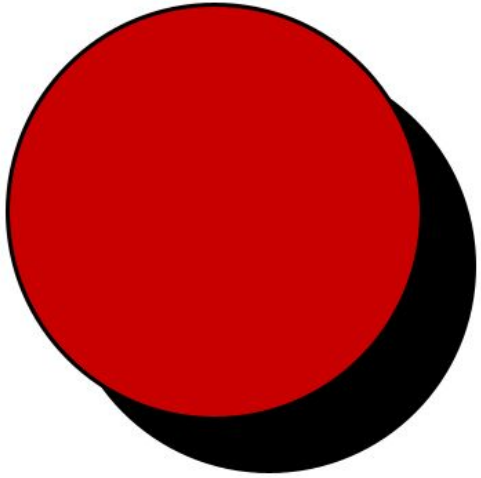
`stroke()` sets the color of the border of the shape

```
7  // Done every frame
8  void draw() {
9
10   background(255, 255, 255);
11
12   fill(200, 0, 0);
13   stroke(0, 0, 255);
14   strokeWeight(10);
15   circle(width/2, height/2, 300);
16
17   save("Output/still.png");  // save image
18   noLoop();  // stop drawing of new frames
19 }
20
```



Section 1: Drawing Order

```
7  // Done every frame
8  void draw() {
9
10   background(255, 255, 255);
11
12   // Draw circle shadow
13   fill(0, 0, 0);
14   noStroke();
15   circle(width/2 + 40, height/2 + 40, 300);
16
17   // Draw circle
18   fill(200, 0, 0);
19   stroke(0, 0, 0);
20   strokeWeight(3);
21   circle(width/2, height/2, 300);
22
23   save("Output/still.png");  // save image
24   noLoop();  // stop drawing of new frames
25 }
26
```



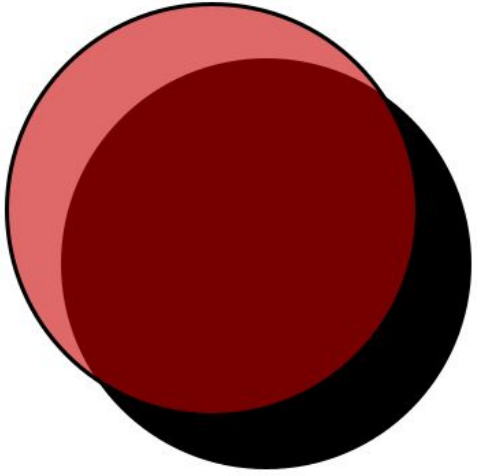
Section 1: Alpha Value

Syntax

```
stroke(r, g, b)
stroke(r, g, b, alpha)
```

```
7  // Done every frame
8  void draw() {
9
10   background(255, 255, 255);
11
12   // Draw circle shadow
13   fill(0, 0, 0);
14   noStroke();
15   circle(width/2 + 40, height/2 + 40, 300);
16
17   // Draw circle
18   fill(200, 0, 0, 150);
19   stroke(0, 0, 0);
20   strokeWeight(3);
21   circle(width/2, height/2, 300);
22
23   save("Output/still.png");  // save image
24   noLoop();  // stop drawing of new frames
25 }
26
```

The *alpha* sets the opacity of the color
0 = clear, 255 = opaque (solid)

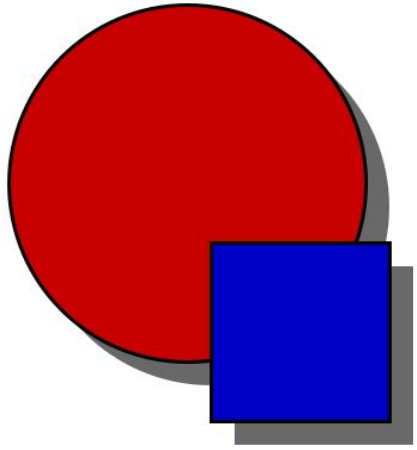


Section 1: Rectangles (And Squares)

`rect(a, b, c, d)`

a	`(float)`	x-coordinate of the rectangle by default
b	`(float)`	y-coordinate of the rectangle by default
c	`(float)`	width of the rectangle by default
d	`(float)`	height of the rectangle by default

```
// Draw square shadow
fill(0, 0, 0, 150);
noStroke();
rect(320 + 20, 350 + 20, 150, 150);

// Draw square
fill(0, 0, 200);
stroke(0, 0, 0);
rect(320, 350, 150, 150);
```



Section 1: Triangles

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

x1 (float) x-coordinate of the first point

y1 (float) y-coordinate of the first point

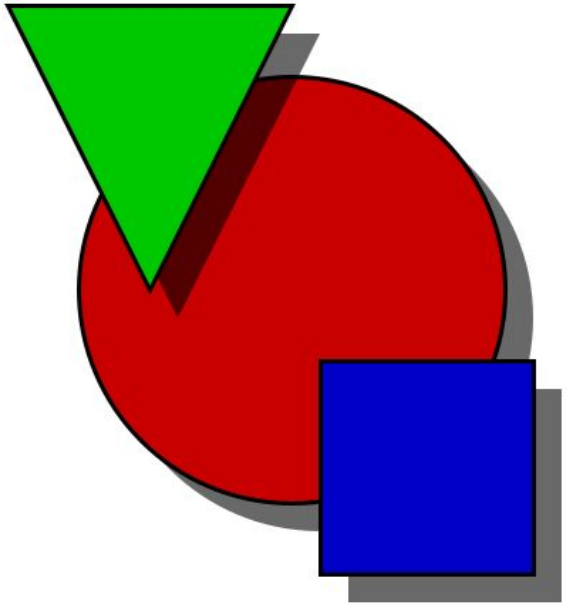
x2 (float) x-coordinate of the second point

y2 (float) y-coordinate of the second point

x3 (float) x-coordinate of the third point

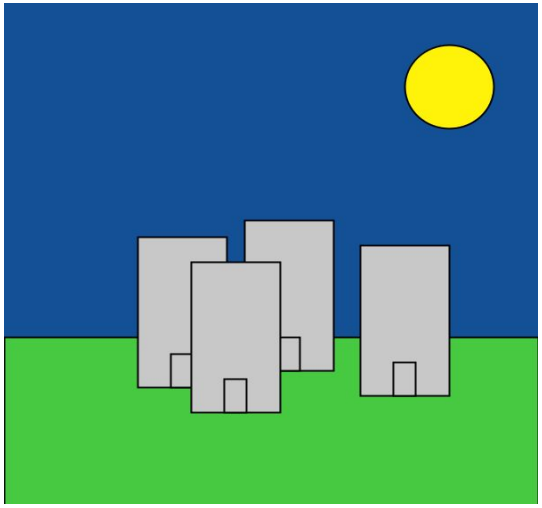
y3 (float) y-coordinate of the third point

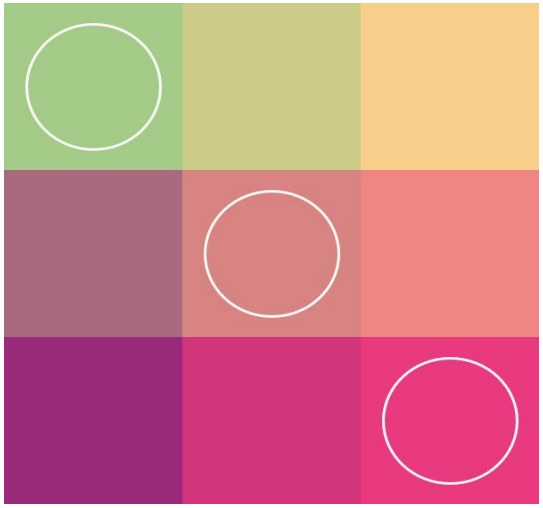
```
// Draw triangle shadow
fill(0, 0, 0, 150);
noStroke();
triangle(100 + 20, 100 + 20,
         300 + 20, 100 + 20,
         200 + 20, 300 + 20);

// Draw triangle
fill(0, 200, 0);
stroke(0, 0, 0);
triangle(100, 100,
         300, 100,
         200, 300);
```



Free Coding Section!

Now you're free to code whatever you'd like! We'll be walking around to help and answer questions.

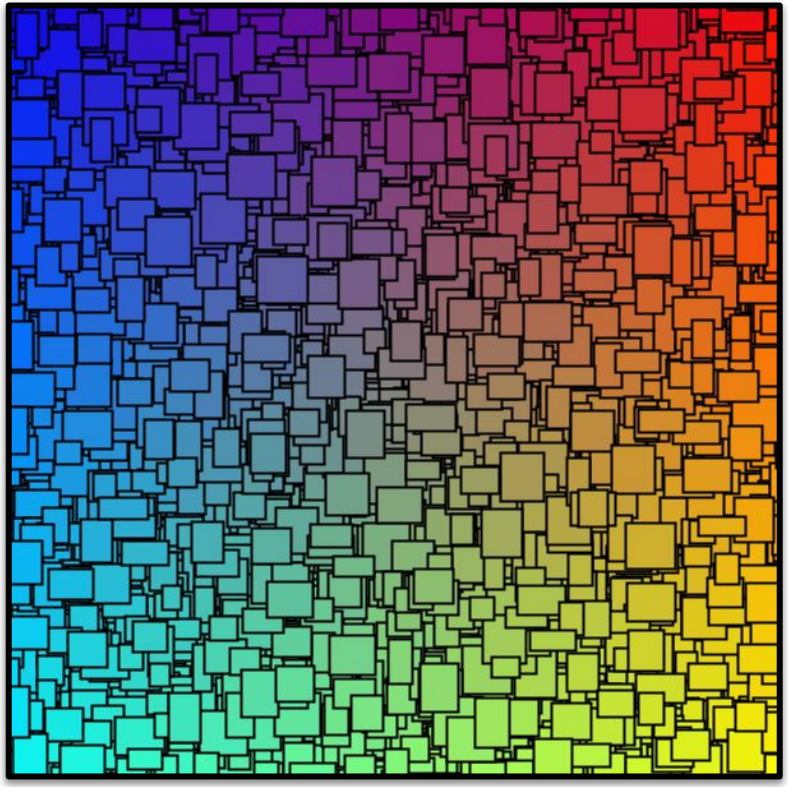




No need to rush! You can pick up where you left off during the next free section or even at home

What We'll Cover

- Randomness
- Variables
- `for` loops
- `map()`
- `println`

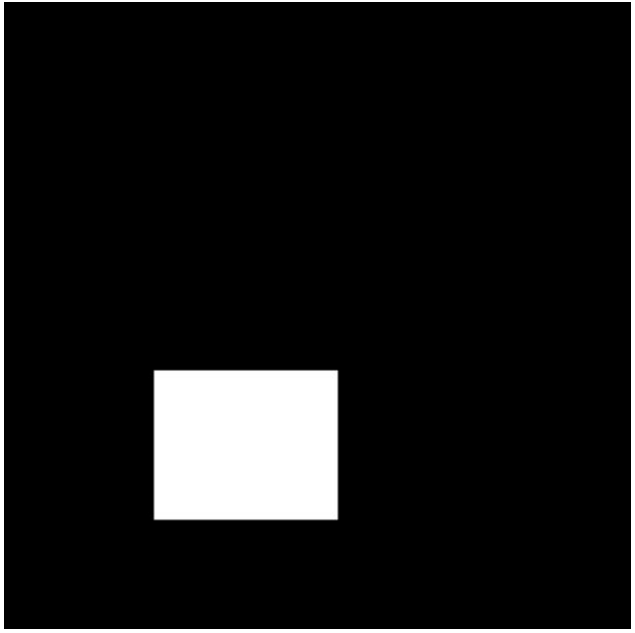


`random()` allows you to generate random numbers in a range

Syntax

```
random(high)
random(low, high)
```

```
7  // Done every frame
8  void draw() {
9
10   background(0, 0, 0);
11   strokeWeight(2);
12
13   rect(random(width),
14        random(height),
15        random(100, 200),
16        random(100, 200));
17
18   save("Output/still.png");  // save image to output folder
19   noLoop();  // stop drawing of new frames
20 }
21
```

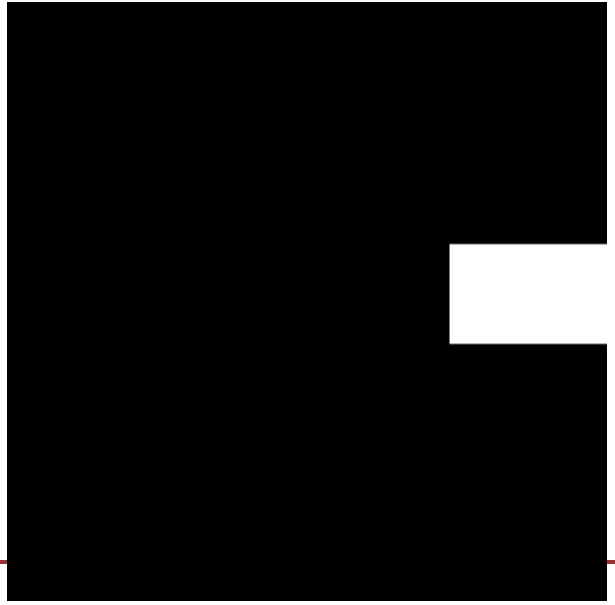


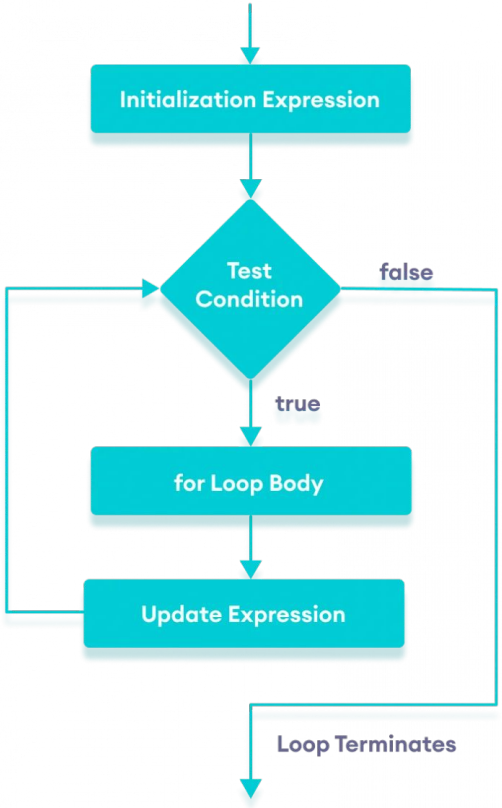
Variables allow you to save data and use it later.

Decimal (floating point) numbers `float var = value`

Integers (whole numbers) `int var = value`

```
7  // Done every frame
8  void draw() {
9
10   background(0, 0, 0);
11   strokeWeight(2);
12
13   float x = random(width);  // x coordinate
14   float y = random(height);  // y coordinate
15   float rWidth = random(100, 200);  // rect width
16   float rHeight = random(100, 200);  // rect height
17
18   rect(x, y, rWidth, rHeight);  // draw rect
19
20   save("Output/still.png");  // save image to output folder
21   noLoop();  // stop drawing of new frames
22 }
```





`for` loops let you repeat a command multiple times.

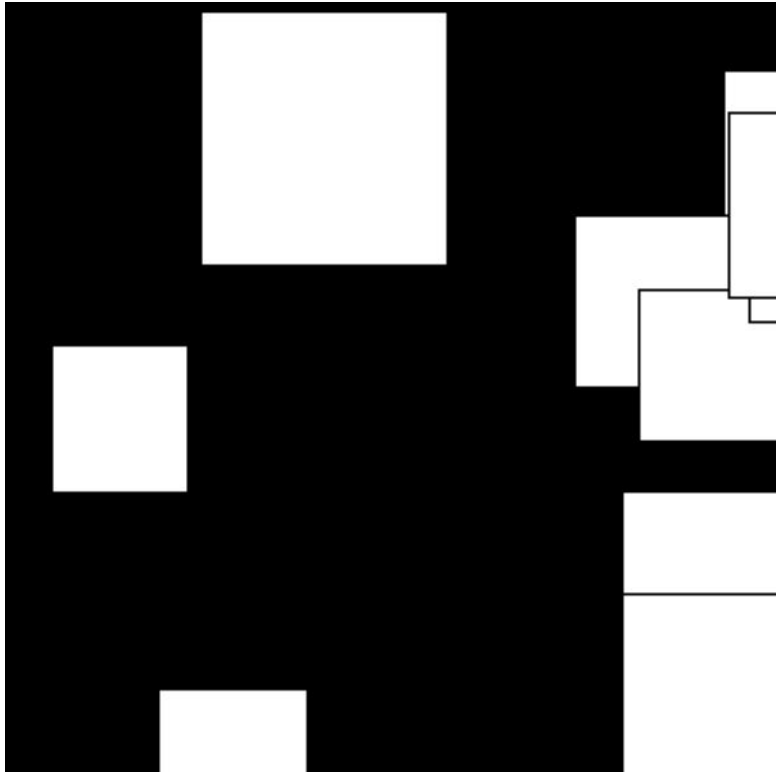
Examples

```
for (int i = 0; i < 5; i++){
    // do something
}
```

```
for (int x = 0; x < width; x += width/10){
    // do something with x
}
```

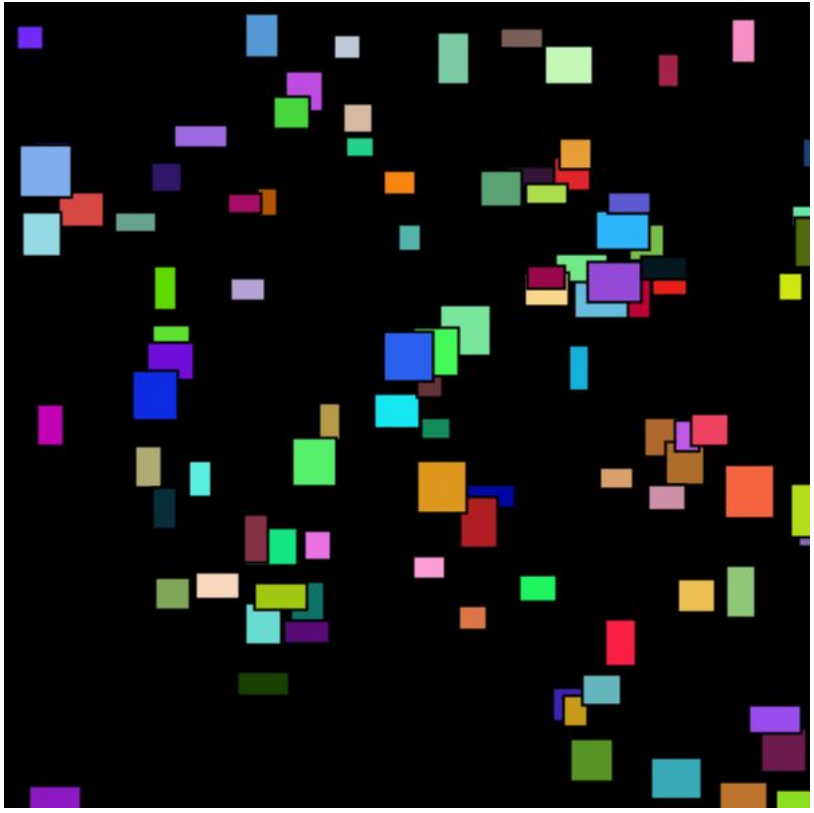
Section 2: `for` Loops

```
7  // Done every frame
8  void draw() {
9
10   background(0, 0, 0);
11   strokeWeight(2);
12
13   for (int i = 0; i < 10; i++ ) {
14
15     float x = random(width);  // x coordinate
16     float y = random(height);  // y coordinate
17     float rWidth = random(100, 200);  // rect width
18     float rHeight = random(100, 200);  // rect height
19
20     rect(x, y, rWidth, rHeight);  // draw rect
21
22   }
23
24   save("Output/still.png");  // save image to output folder
25   noLoop();  // stop drawing of new frames
26 }
```



Section 2: Random Colors

```
// Done every frame
void draw() {

  background(0, 0, 0);
  strokeWeight(2);

  for (int i = 0; i < 100; i++ ) {

    float x = random(width);  // x coordinate
    float y = random(height);  // y coordinate
    float rWidth = random(15, 40);  // rect width
    float rHeight = random(15, 40);  // rect height

    float r = random(0, 255);  // random red component
    float g = random(0, 255);  // random green component
    float b = random(0, 255);  // random blue component

    fill(r, g, b);  // set color
    rect(x, y, rWidth, rHeight);  // draw rect

  }

  save("Output/still.png");  // save image to output folder
  noLoop();  // stop drawing of new frames
}
```

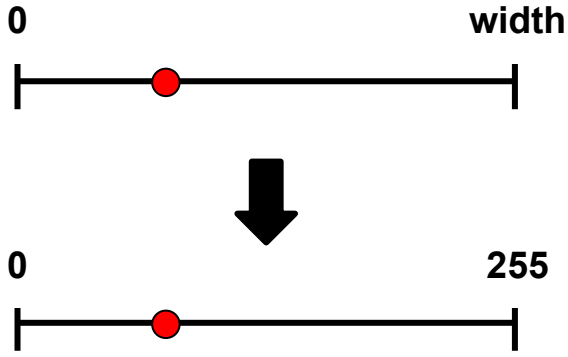


`map()` allows you to remap a number from one range to another

Syntax `map(value, start1, stop1, start2, stop2)`

Parameters

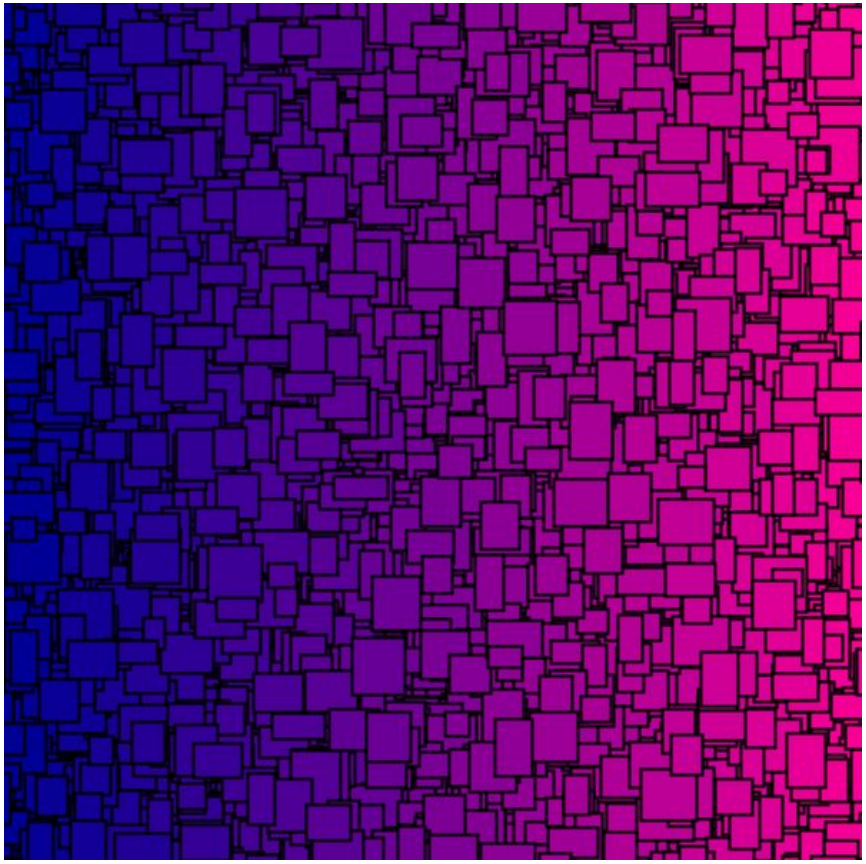
value	(float)	the incoming value to be converted
start1	(float)	lower bound of the value's current range
stop1	(float)	upper bound of the value's current range
start2	(float)	lower bound of the value's target range
stop2	(float)	upper bound of the value's target range



Ex. `float r = map(x, 0, width, 0, 255);`

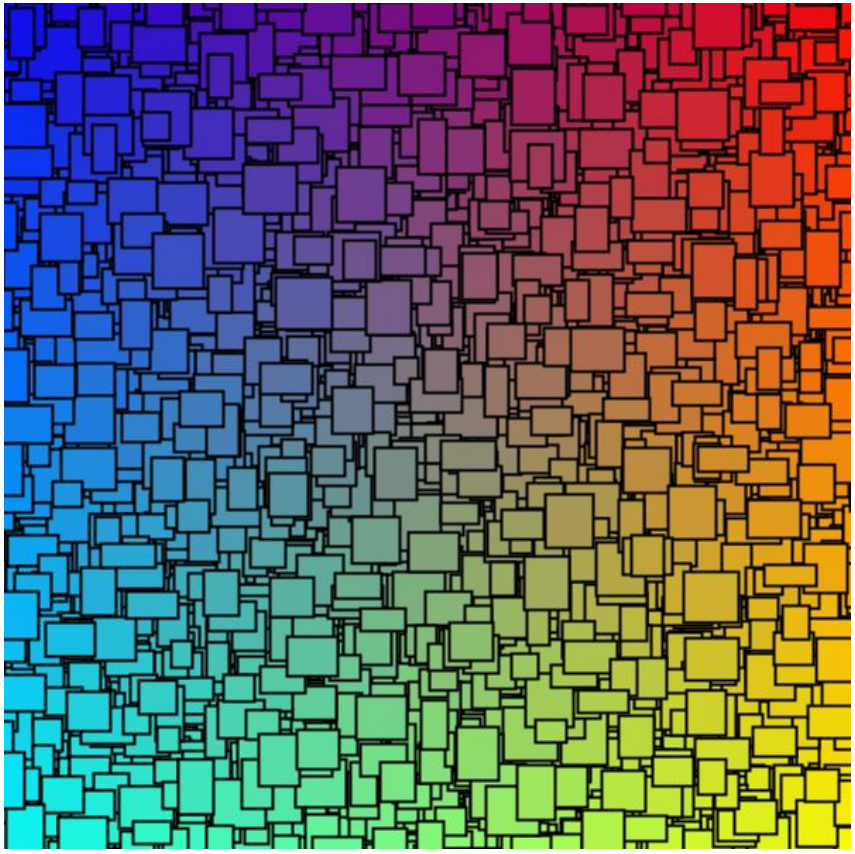
Section 2: `map()`

```
// Done every frame
void draw() {

  background(0, 0, 0);
  strokeWeight(2);

  for (int i = 0; i < 10000; i++ ) {

    float x = random(-40, width);  // x coordinate
    float y = random(-40, height);  // y coordinate
    float rWidth = random(15, 40);  // rect width
    float rHeight = random(15, 40);  // rect height

    float r = map(x, 0, width, 0, 255);  // red component
    float g = 0;  // green component
    float b = 150;  // blue component

    fill(r, g, b);  // set color
    rect(x, y, rWidth, rHeight);  // draw rect

  }

  save("Output/still.png");  // save image to output folder
  noLoop();  // stop drawing of new frames
}
```



Section 2: Finishing Touches

```
// Done every frame
void draw() {

  background(0, 0, 0);
  strokeWeight(2);

  for (int i = 0; i < 10000; i++ ) {

    float x = random(-40, width);  // x coordinate
    float y = random(-40, height);  // y coordinate
    float rWidth = random(15, 40);  // rect width
    float rHeight = random(15, 40);  // rect height

    float r = map(x, -40, width, 0, 255);  // red component
    float g = map(y, -40, height, 0, 255);  // green component
    float b = map(x, -40, width, 255, 0);  // blue component

    fill(r, g, b);  // set color
    rect(x, y, rWidth, rHeight);  // draw rect

  }

  save("Output/still.png");  // save image to output folder
  noLoop();  // stop drawing of new frames
}
```

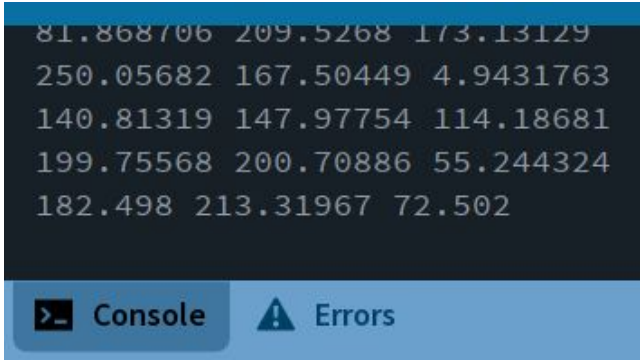


If your code isn't behaving as you expect, you might have a bug!

`println()` can be used to output a variable's value to the console

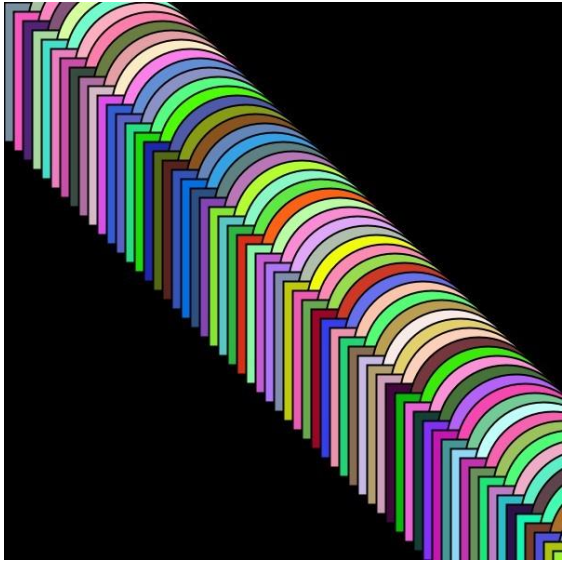
```
println(variables)
```

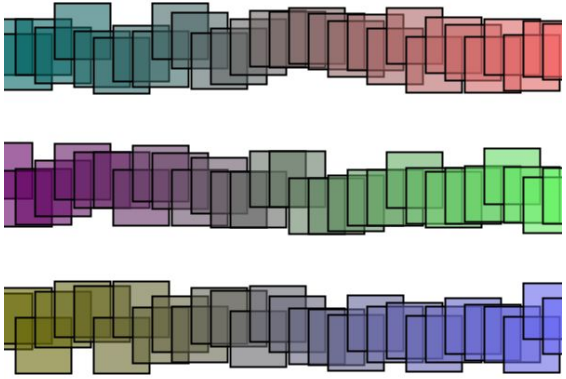
```
float r = map(x, -40, width, 0, 255);  // red component
float g = map(y, -40, height, 0, 255);  // green component
float b = map(x, -40, width, 255, 0);  // blue component

println(r, g, b);
```

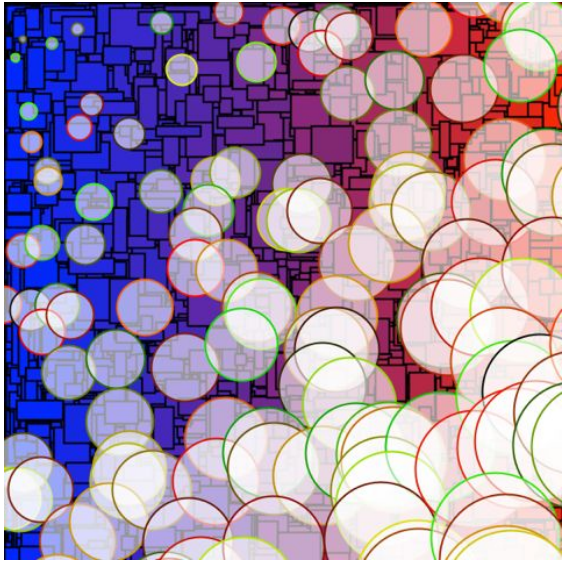


Free Coding Section!

Don’t hesitate to ask us questions! And no need to use everything we learned in this section

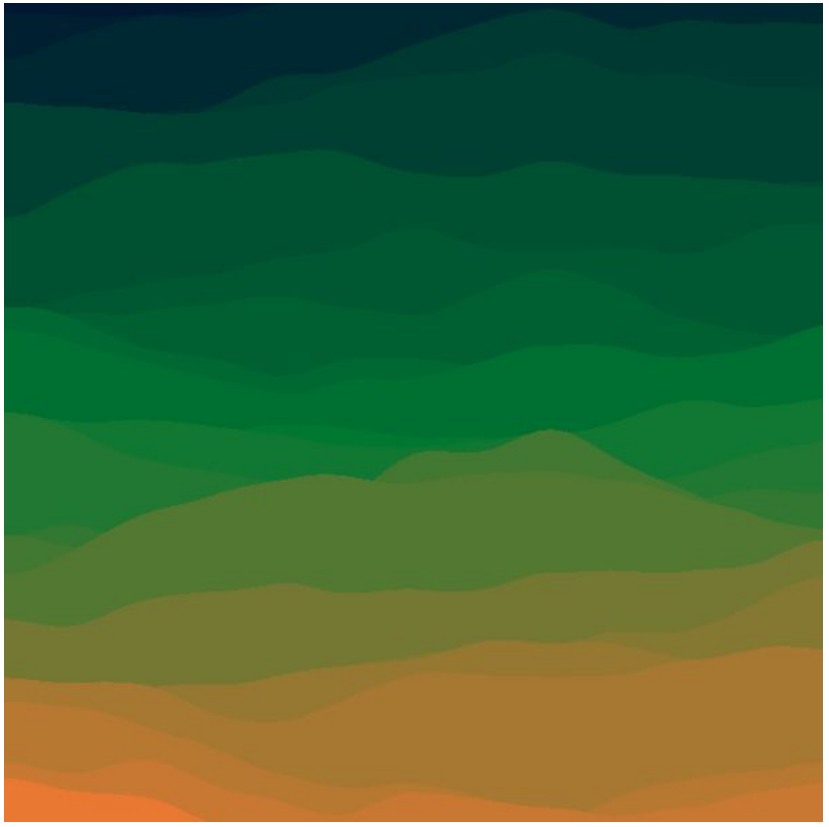






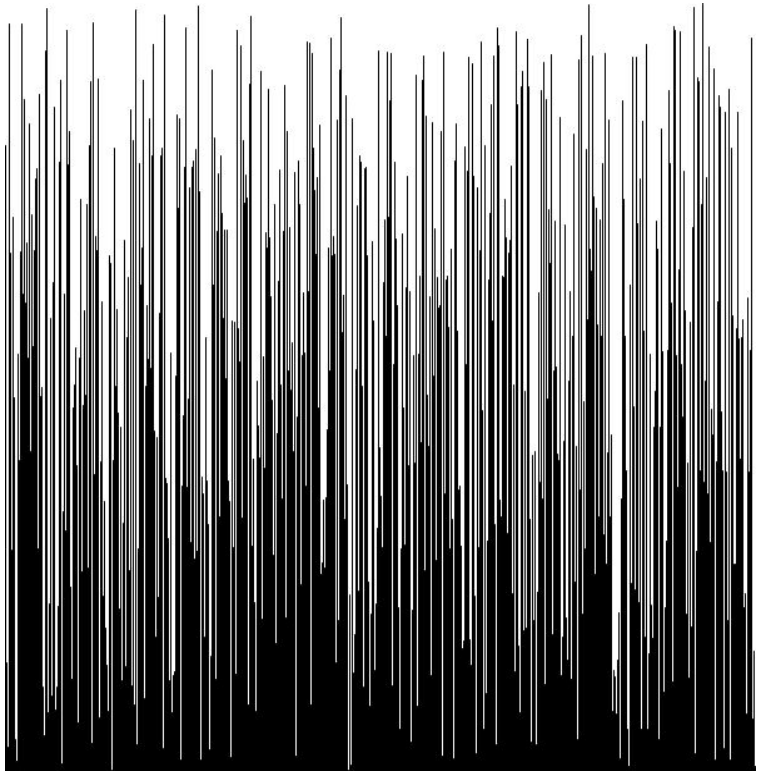
What We'll Cover

- Perlin noise
- Nesting loops

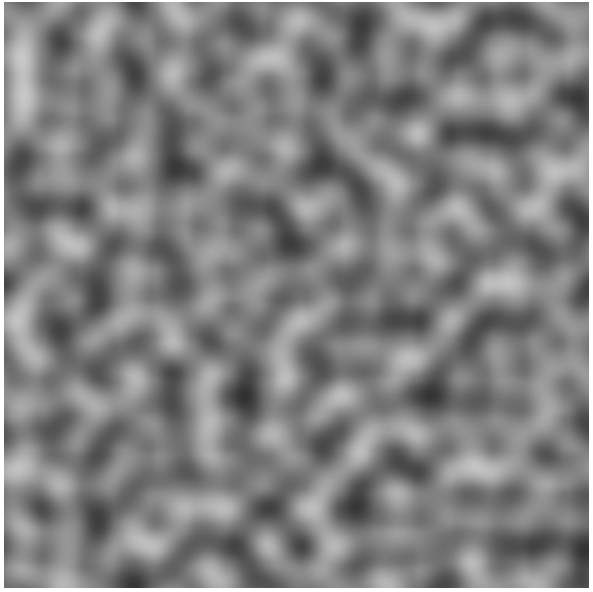


What do you think this will generate?

```
7  // Done every frame
8  void draw() {
9
10   background(255, 255, 255);
11   stroke(0, 0, 0);
12
13   for (float x = 0; x <= width; x++) {
14
15     float lineHeight = random(height);
16
17     line(x, height, x, lineHeight);
18
19   }
20
21   save("Output/still.png");  // save image to output folder
22   noLoop();  // stop drawing of new frames
23 }
24
```



Perlin noise is random but continuous



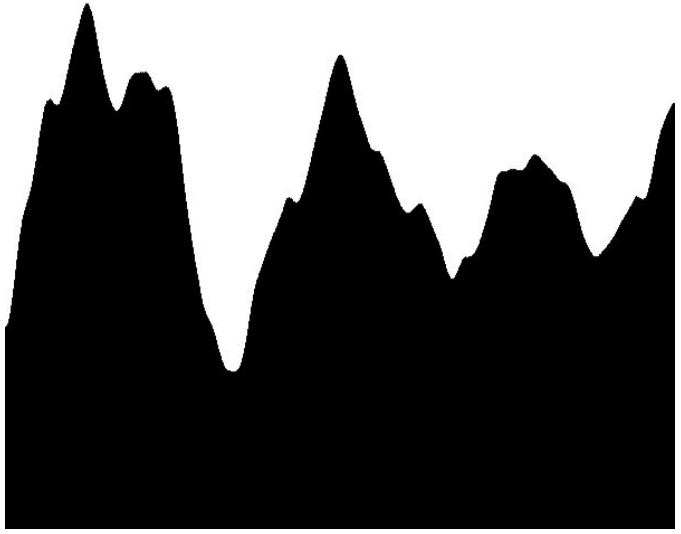
2D Perlin Noise

Minecraft terrain is generated using continuous noise

Section 3: 1D Perlin Noise

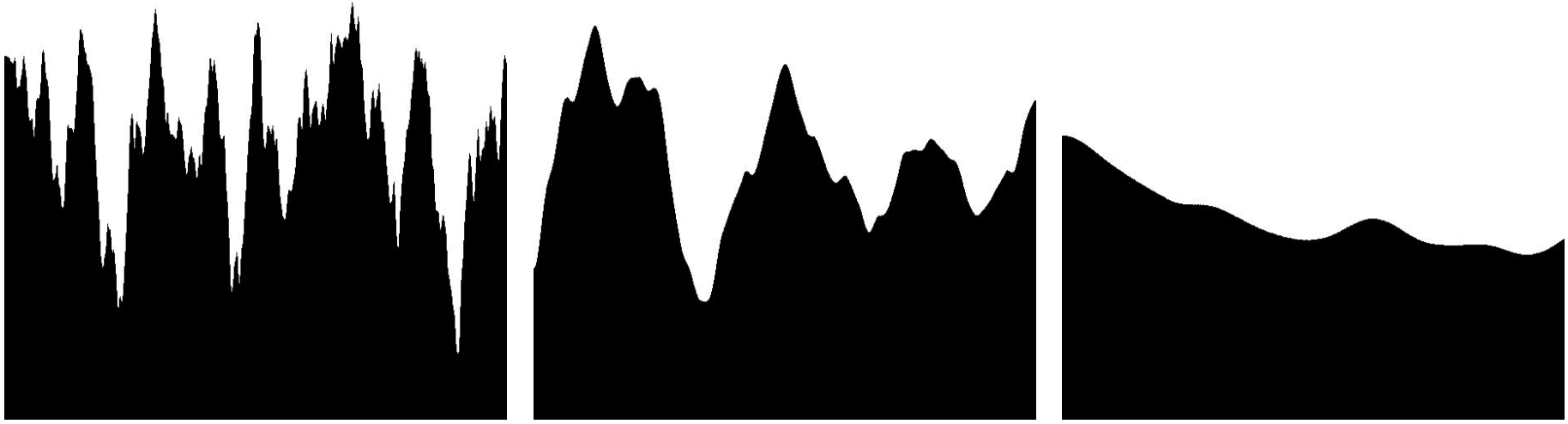
Let's choose our line height based on 1D Perlin noise using the `noise()` function. It always outputs between 0 and 1

```
// Done every frame
void draw() {

  background(255, 255, 255);
  stroke(0, 0, 0);

  for (float x = 0; x <= width; x++) {

    float noiseVal = noise(x/100);
    float lineHeight = height * noiseVal;

    line(x, height, x, lineHeight);

  }

  save("Output/still.png");  // save image to output folder
  noLoop();  // stop drawing of new frames
}

```



We can vary the scale of the noise to make the surface smoother

`noise(x/20)` `noise(x/100)` `noise(x/500)`

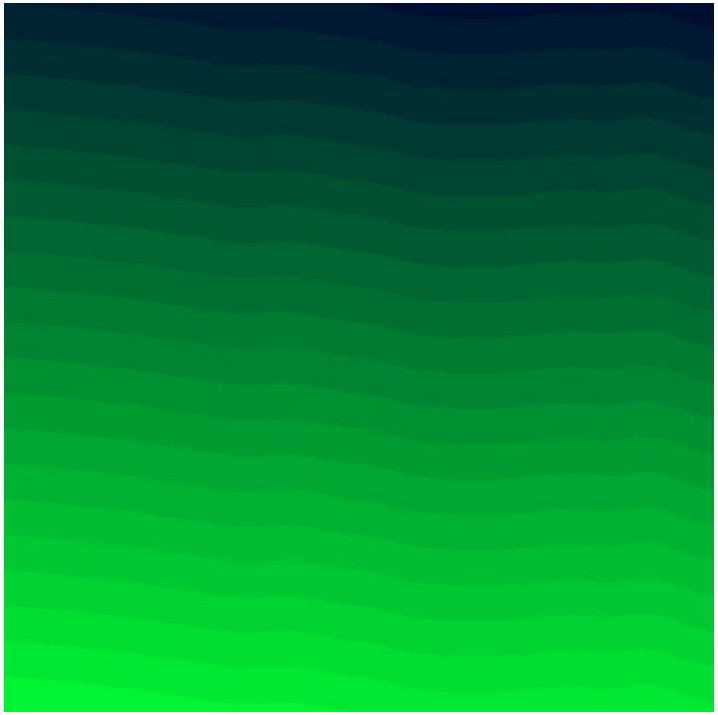


We can offset a flat line using our noise value

```
7  // Done every frame
8  void draw() {
9
10   background(255, 255, 255);
11   stroke(0, 0, 0);
12
13   for (float x = 0; x <= width; x++) {
14
15     float noiseVal = noise(x/300);
16     float offset = 100 * noiseVal;
17
18     float y = width/2 + offset;
19
20     line(x, height, x, y);
21
22   }
23
24   save("Output/still.png");  // save image to output folder
25   noLoop();  // stop drawing of new frames
26 }
27
```

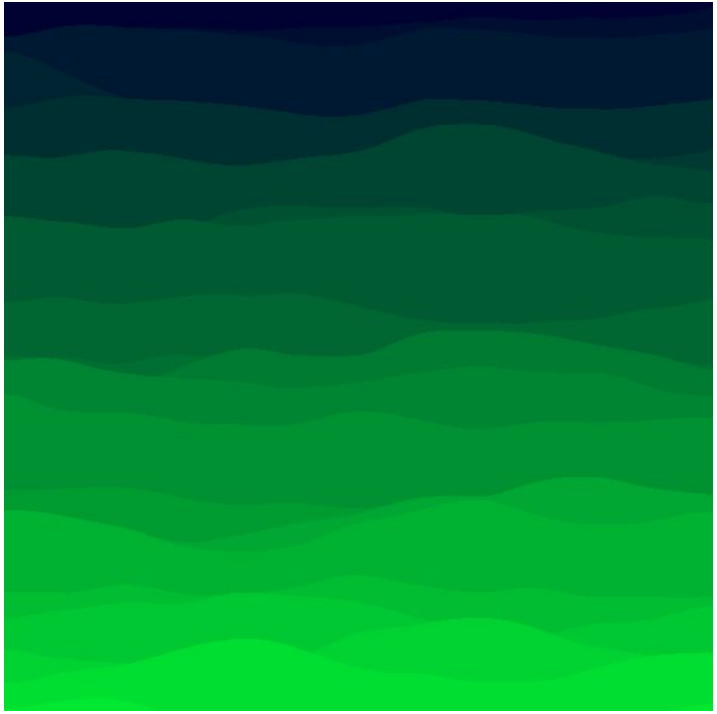
We can use a nested loop to repeat this pattern

```
9   background(255, 255, 255);
10  stroke(0, 0, 0);
11
12  for (float y = -150; y <= height; y += 30) {
13
14    stroke(0, map(y, -100, height, 0, 255), 50);
15
16    for (float x = 0; x <= width; x++) {
17
18      float noiseVal = noise(x/400);
19      float offset = 150 * noiseVal;
20
21      float yLine = y + offset;
22
23      line(x, height, x, yLine);
24
25    }
26
27  }
```



We can get the pattern to vary in y if we use 2D Perlin noise

```
12  for (float y = -150; y <= height; y += 30) {
13
14    stroke(0, map(y, -100, height, 0, 255), 50);
15
16    for (float x = 0; x <= width; x++) {
17
18      float noiseVal = noise(x/400, y);
19      float offset = 150 * noiseVal;
20
21      float yLine = y + offset;
22
23      line(x, height, x, yLine);
24
25    }
26
27  }
```



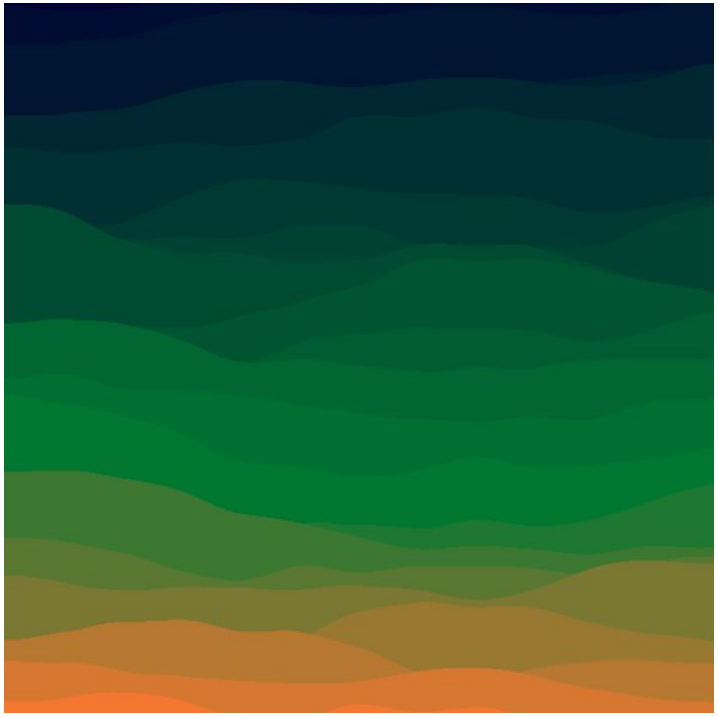
If/Else statements can be used to conditionally execute a command

```
if (i <= 5) {
  // Do something if condition 1 is met
} else if (i < 10) {
  // Do something if condition 2 but not 1 is met
} else {
  // Do somthing if no conditions are met
}
```

```
if (y < height/3) {
  fill(255, 0, 0);
} else if (y < 2*height/3) {
  fill(0, 255, 0);
} else {
  fill(0, 0, 255);
}
```

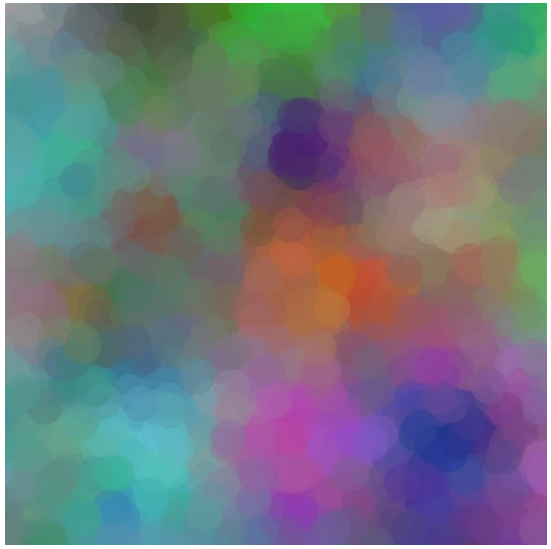
Using if/else statements, we can create a three color gradient

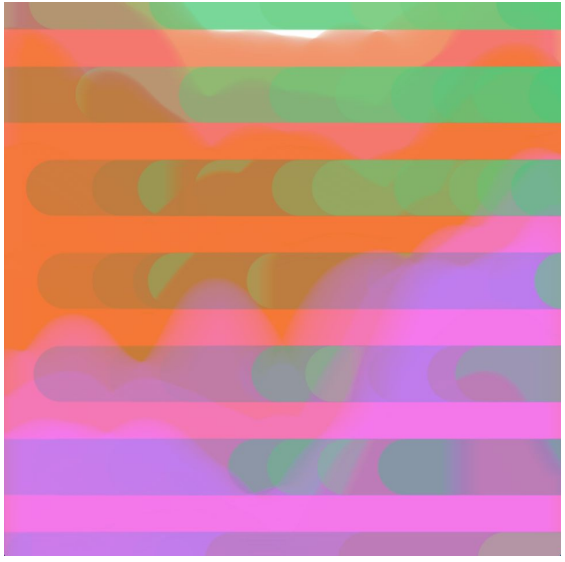
```
for (float y = -150; y <= height; y += 30) {

  if (y < height/2) {
     stroke(0,
            map(y, -100, height/2, 0, 120),
            50);
  } else {
     stroke(map(y, height/2, height-50, 0, 255),
            120,
            50);
  }
```

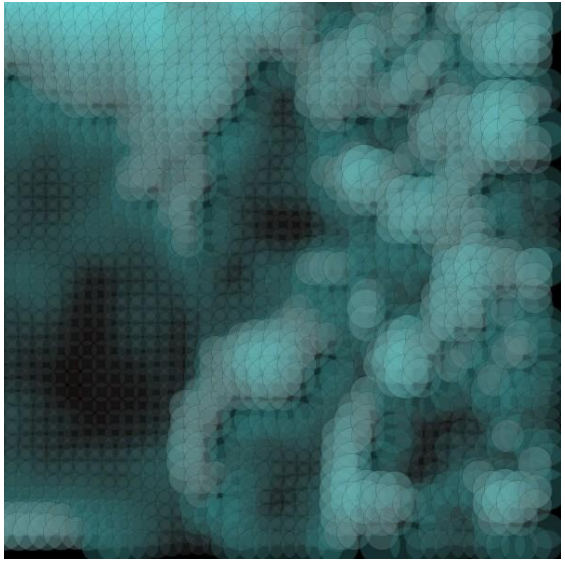


Free Coding Section 3

Don’t hesitate to ask us questions! And no need to use everything we learned in this section







We’ll stop ~5-10 mins before the end of class for wrap-up