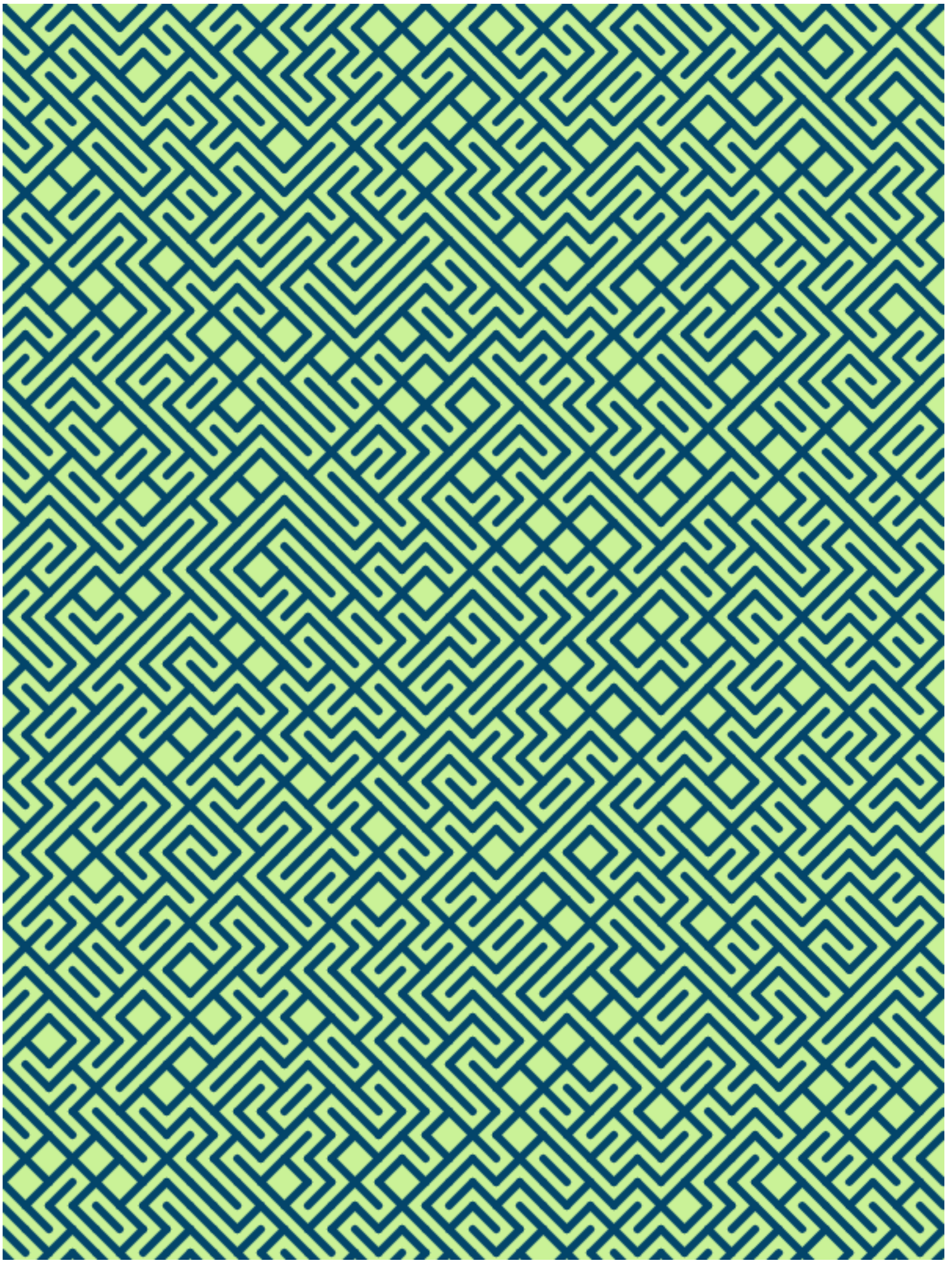


Generative Art

https://editor.p5js.org/

http://www.galaxykate.com/apps/Prototypes/LTrees/

http://math.hws.edu/eck/js/edge-of-chaos/CA.html

What goes into a piece of generative art?

- **Randomness** is crucial for creating generative art. The art should be different each time you run the generation script, so randomness is usually a large part of that.
- **Algorithms** — Implementing an algorithm visually can often generate awesome art, for example, the binary tree above.
- **Geometry** — Most generative art incorporates shapes, and the math from high school geometry class can aid in some really cool effects.

sketch.js

```
function setup() {
  createCanvas(400, 400);
}

function draw() {
  background(220);
}
```

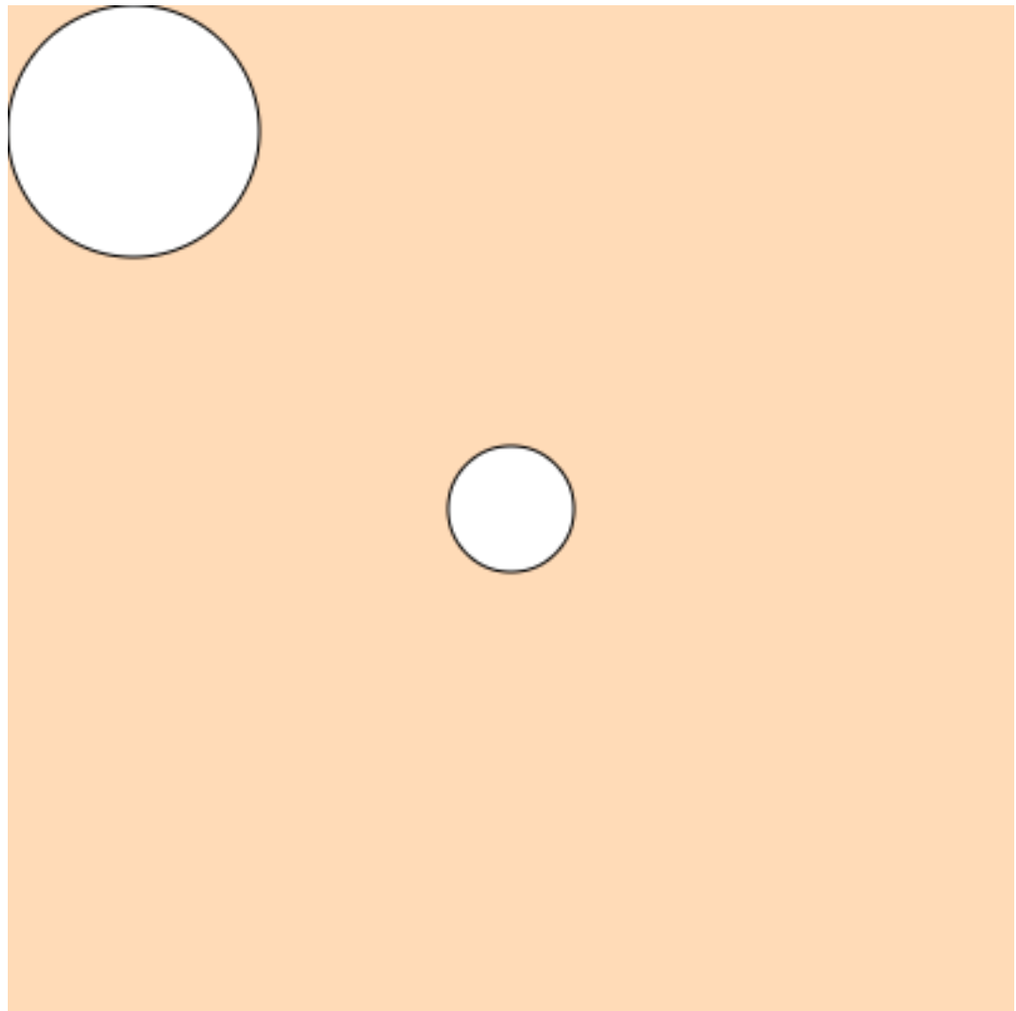
Code is a precise way of explaining something and it is written in a programming language.

Preview

```
function draw() {
  background('PEACHPUFF');
  ellipse(50,50,100);
  ellipse(200,200,50);
}
```

PEACHPUFF is from CSS colours. You can check out some of these colours:

http://colours.neilorangepeel.com/



in your JavaScript file, you will add two functions that will be used in pretty much ever p5 script: `setup` and `draw`. These names are necessary for p5 to call them.

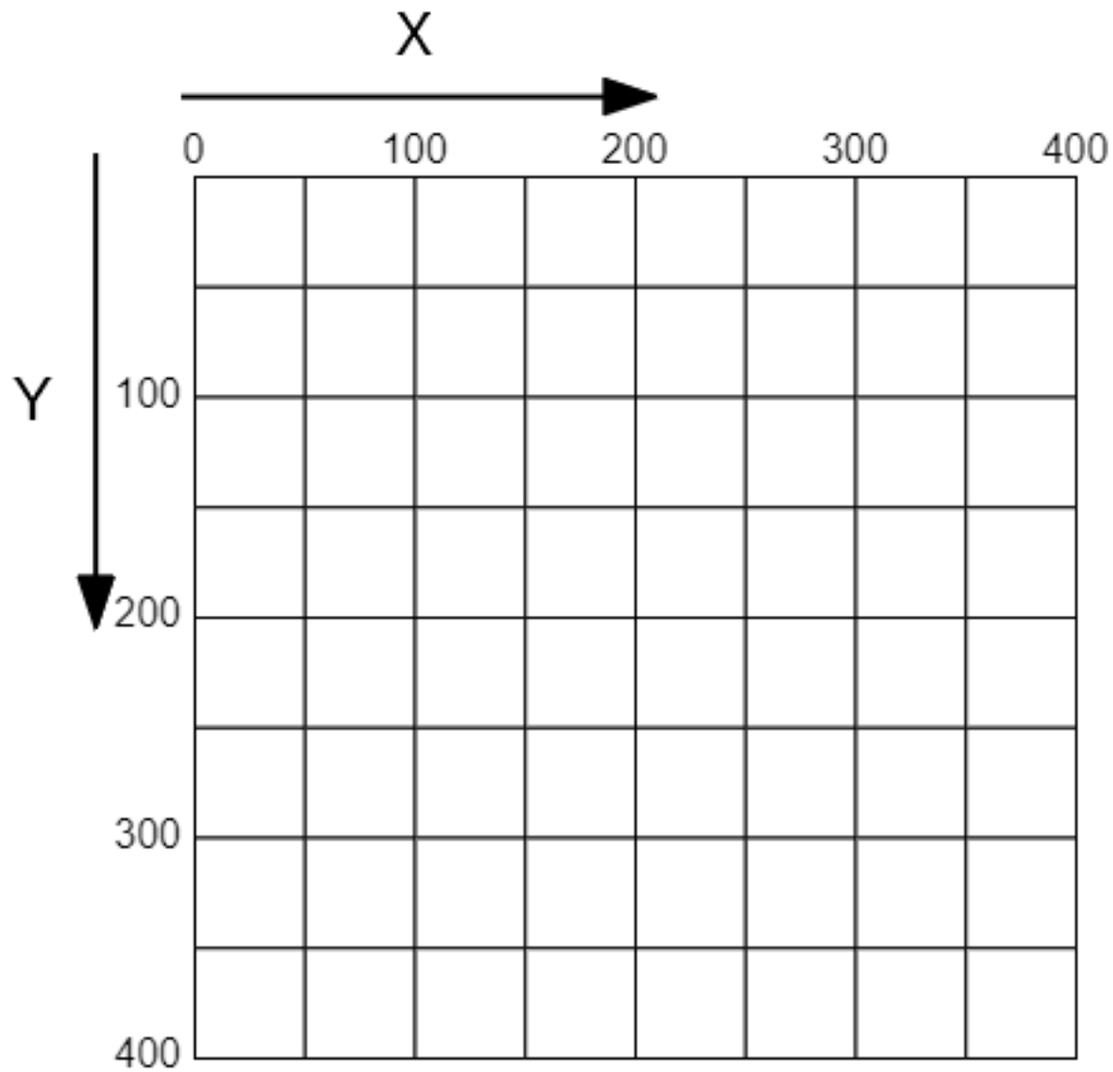
`setup` is called when the program starts. You will probably create a canvas to draw on within it using the `createCanvas` function, and you may set some defaults there. It is only called once!

`draw` is called after setup, and is executed constantly until you call `noLoop`, which will stop it from running again. You can control how many times `draw` runs each second with the `frameRate` function.

```
line(startX, startY, endX, endY)

//circle
ellipse(xCoordinate, yCoordinate, width, height)
//square
rect(xCoordinate, yCoordinate, width, height)

//triangle
triangle(xCoordinate1, yCoordinate1, x2, y2, x3, y3)
```

```
ellipse(50,50,100);
```

```
line(0,0,400,400);
```



This means the line goes from (0,0) to (400, 400) ->
the top left corner to the bottom right.

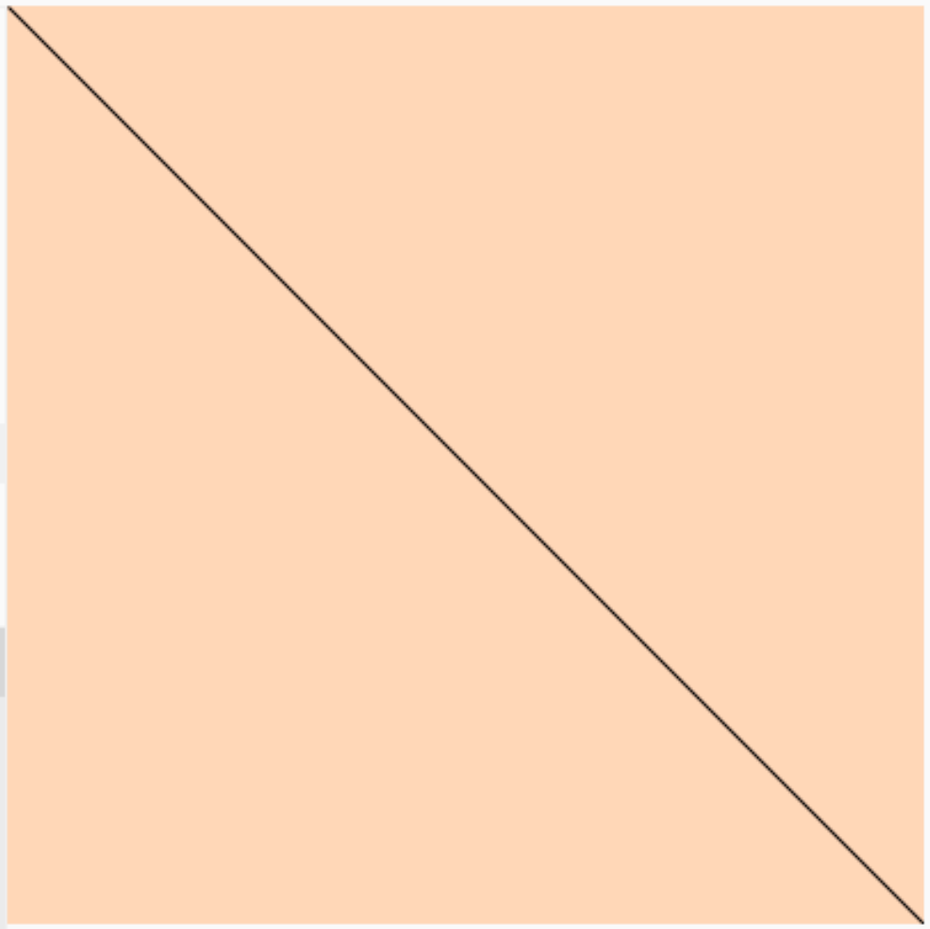
Auto-refresh Primo stick

sketch.js

```
function setup() {
  createCanvas(400, 400);
}

function draw() {
  background('PEACHPUFF');
  line(0,0,400,400);
}
```

Console Clear

Preview



Remember to click play to update your sketch

`random()`

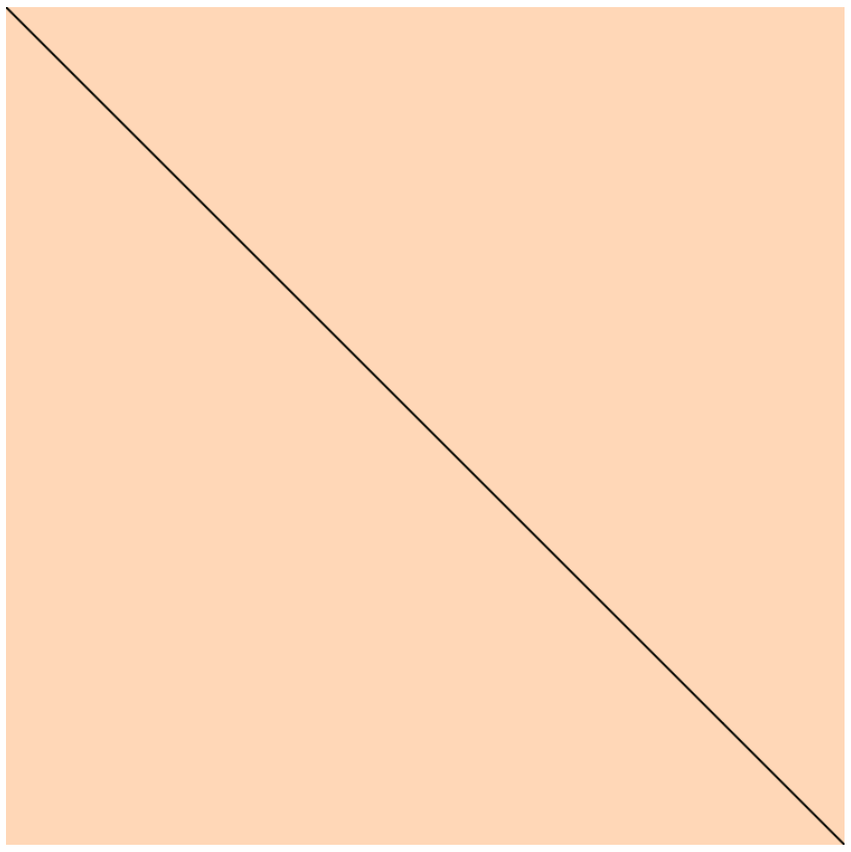
This command will result in a random value between **0 and 1**. That means around half the time the value will be bigger than 0.5 and around half the time the value will be less than 0.5.

So what we could do is use the `random()` command and draw a left line if the value is bigger than 0.5, otherwise, we draw the right-to-left line.

```
function setup() {
  createCanvas(400, 400);
  noLoop();
}
```

```
if (random()>0.5) {
  doAnAction
}
else {
  doSomeOtherAction
}
```

```
function setup() {
  createCanvas(400, 400);
  noLoop();
}
function draw() {
    background('PEACHPUFF');
  if (random()>0.5) {
    line(0,0,400,400);
  }
  else {
    line(400, 0, 0, 400);
  }
}
```



Make sure to add `noLoop();` - that makes sure the line is not flashing.

For Loops

```
for (var count=0; count<400; count = count + 100) {
  doAnAction;
}
```

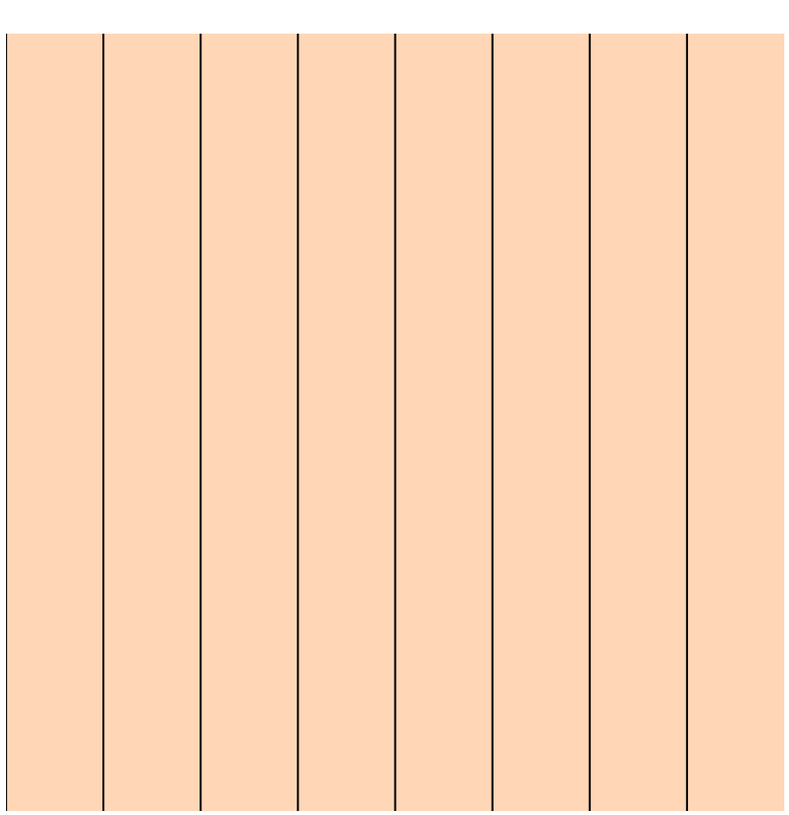
1) it declares a variable `var count = 0`

2) it then adds one hundred *after* running the loop: `count = count + 100`

3) it then checks if count is < 400 with `count < 400`

`for (var x=0; x<10; x=x+1)` goes from 0, 1, 2, 3..., 8, 9! (10 is not less than 10, so we exit without running the code.

```
function setup() {
  createCanvas(400, 400);
  noLoop();
}
function draw() {
  background('PEACHPUFF');
  for (var x=0; x<400; x=x+50) {
    line(x,0,x,400);
  }
}
```



We can use the for loop in our drawing!
This draws multiple lines... but not quite random.

What does this draw?

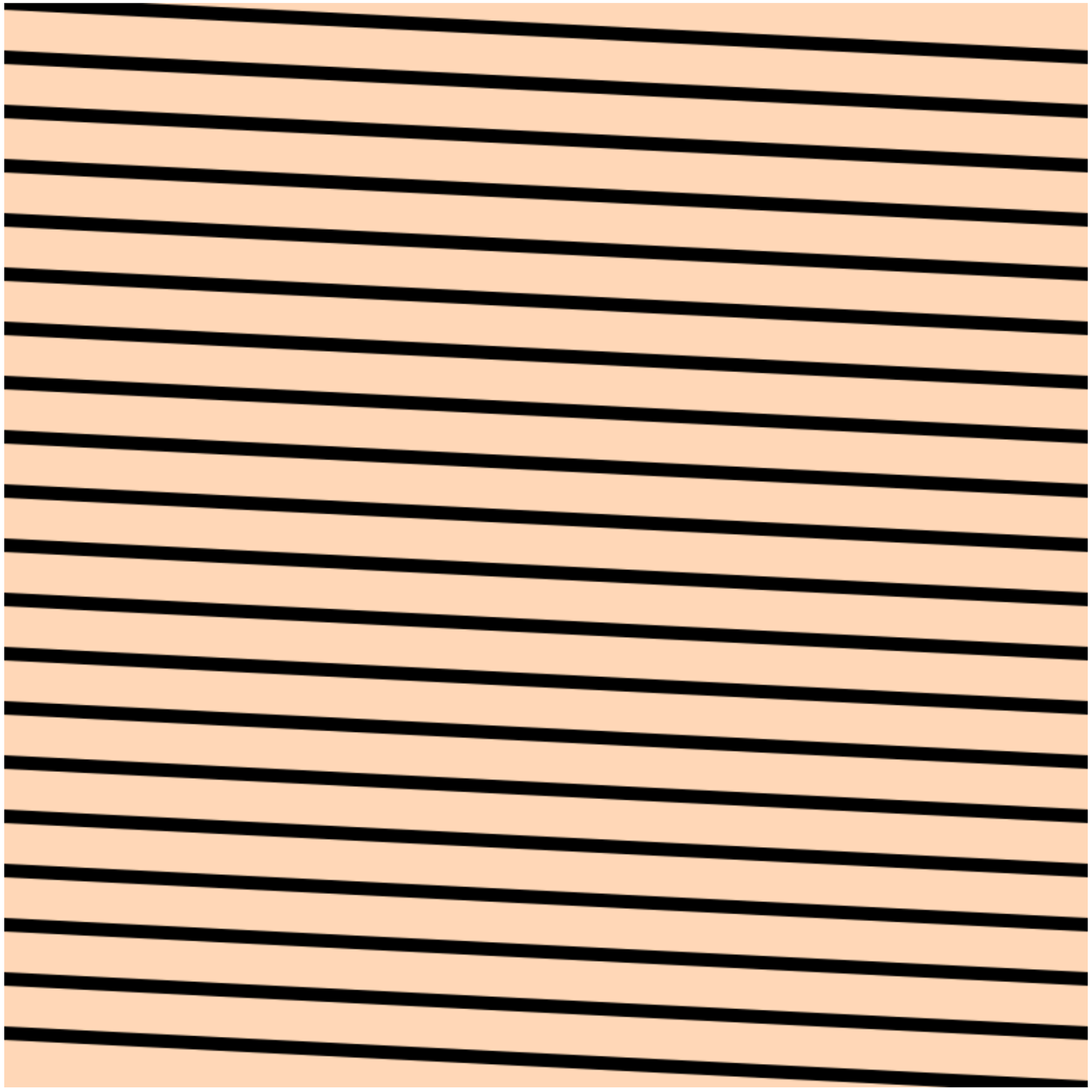
The gap between the lines

```
function setup() {
  createCanvas(400, 400);
  noLoop();
}
function draw() {
  background('PEACHPUFF');
  var step = 20;
  for (var y = 0; y < 400; y = y + step) {
    strokeWeight(5);
    line(0, y, 400, y+step);
  }
}
```

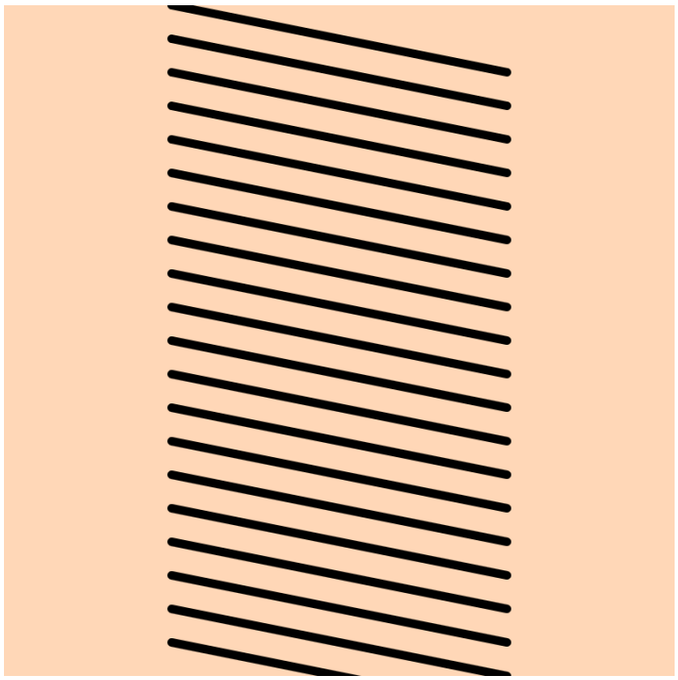
Sets the lines' thickness

Starts are same horizontal position, and moves up

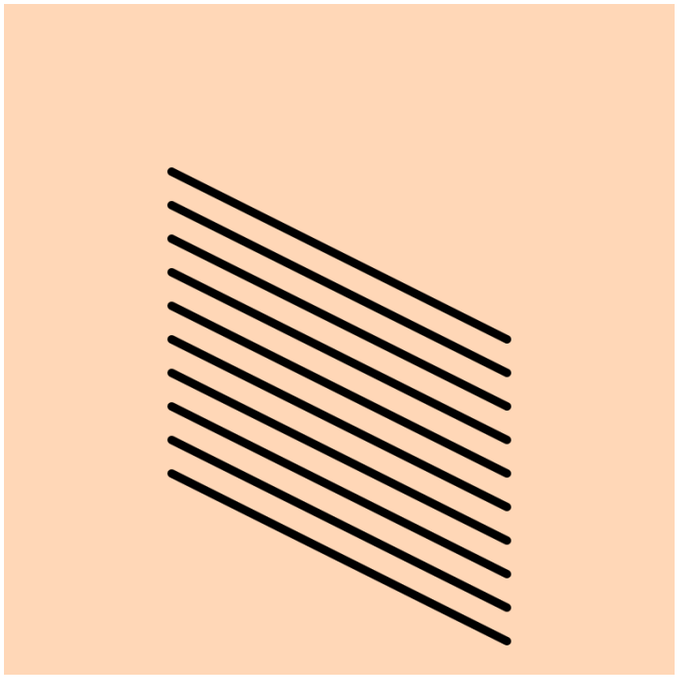
Ends at bottom and also moves up



```
function setup() {
  createCanvas(400, 400);
  noLoop();
}
function draw() {
  background('PEACHPUFF');
  var step = 20;
  for (var y = 0; y < 400; y = y + step) {
    strokeWeight(5);
    line(100, y, 300, y+step*2);
  }
}
```



```
function setup() {
  createCanvas(400, 400);
  noLoop();
}
function draw() {
  background('PEACHPUFF');
  var step = 20;
  for (var y = 100; y < 300; y = y + step) {
    strokeWeight(5);
    line(100, y, 300, y+step*5);
  }
}
```



For loops inside for loops!

```
for (var x=0;x<5;x=x+1) {
  for (var y=0;y<5;y=y+1) {
    drawTheLines;
  }
}
```

```
x=0, y=0; x=0, y=1; x=0, y=2; x=0, y=3; x=0, y=4;
x=1, y=0; x=1, y=1; x=1, y=2; x=1, y=3; x=1, y=4;
x=2, y=0; x=2, y=1; x=2, y=2; x=2, y=3; x=2, y=4;
x=3, y=0; x=3, y=1; x=3, y=2; x=3, y=3; x=3, y=4;
x=4, y=0; x=4, y=1; x=4, y=2; x=4, y=3; x=4, y=4;
```

Here’s what it looks like, in actuality...

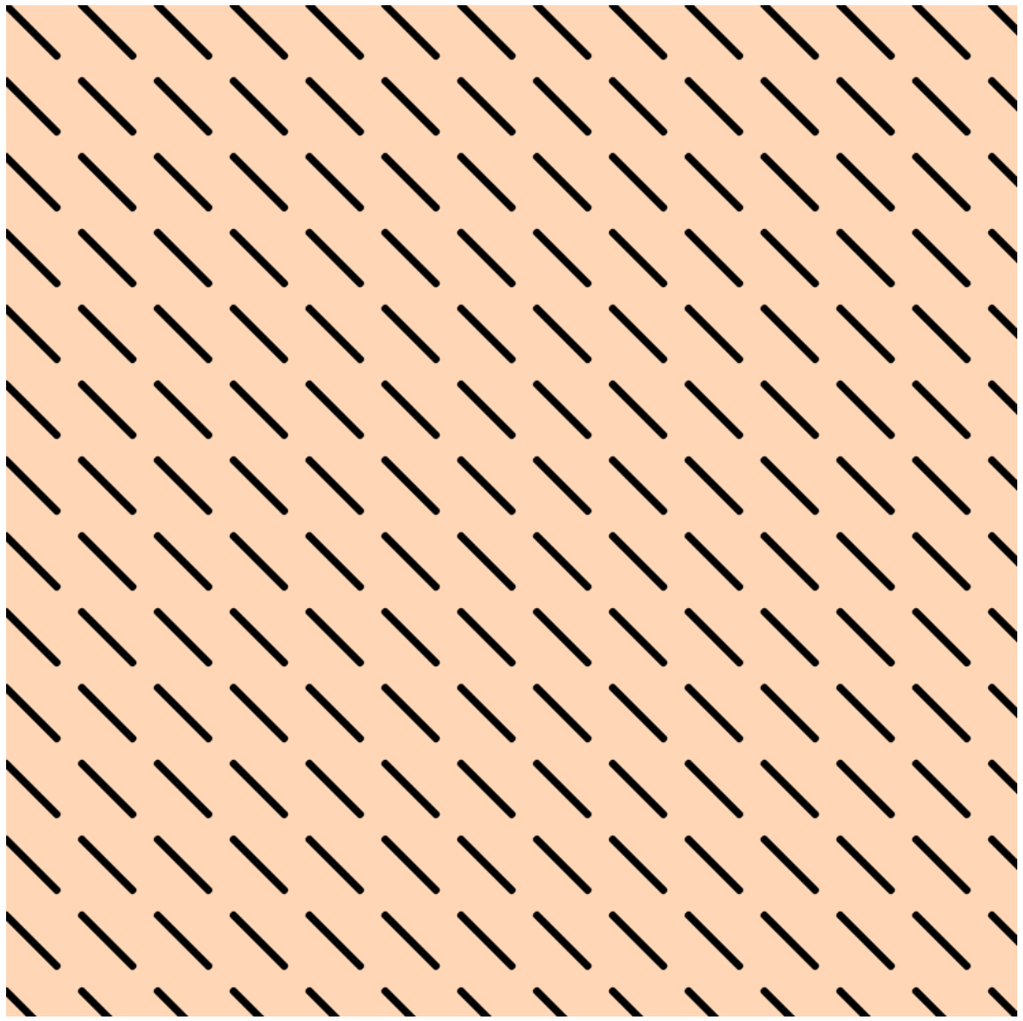
```
function setup() {
  createCanvas(400, 400);
  noLoop();
}
function draw() {
  background('PEACHPUFF');

  for (var x=0;x<400;x=x+1) {
    for (var y=0;y<400;y=y+1) {
       strokeWeight(3);
        line(x,y,x+20, y+20);
    }
  }
}
```

We go from (0,0) to (400,400)!

What do you think this outputs?

Ha! It’s all black, because the changes are just x=x+1 and y=y+1.

Too small of a change so we just fill it all.



```
function setup() {
  createCanvas(400, 400);
  noLoop();
}
function draw() {
  background('PEACHPUFF');
  var step = 30;
  for (var x=0;x<400;x=x+step) {
    for (var y=0;y<400;y=y+step) {
       strokeWeight(3);
        line(x,y,x+20, y+20);
    }
  }
}
```

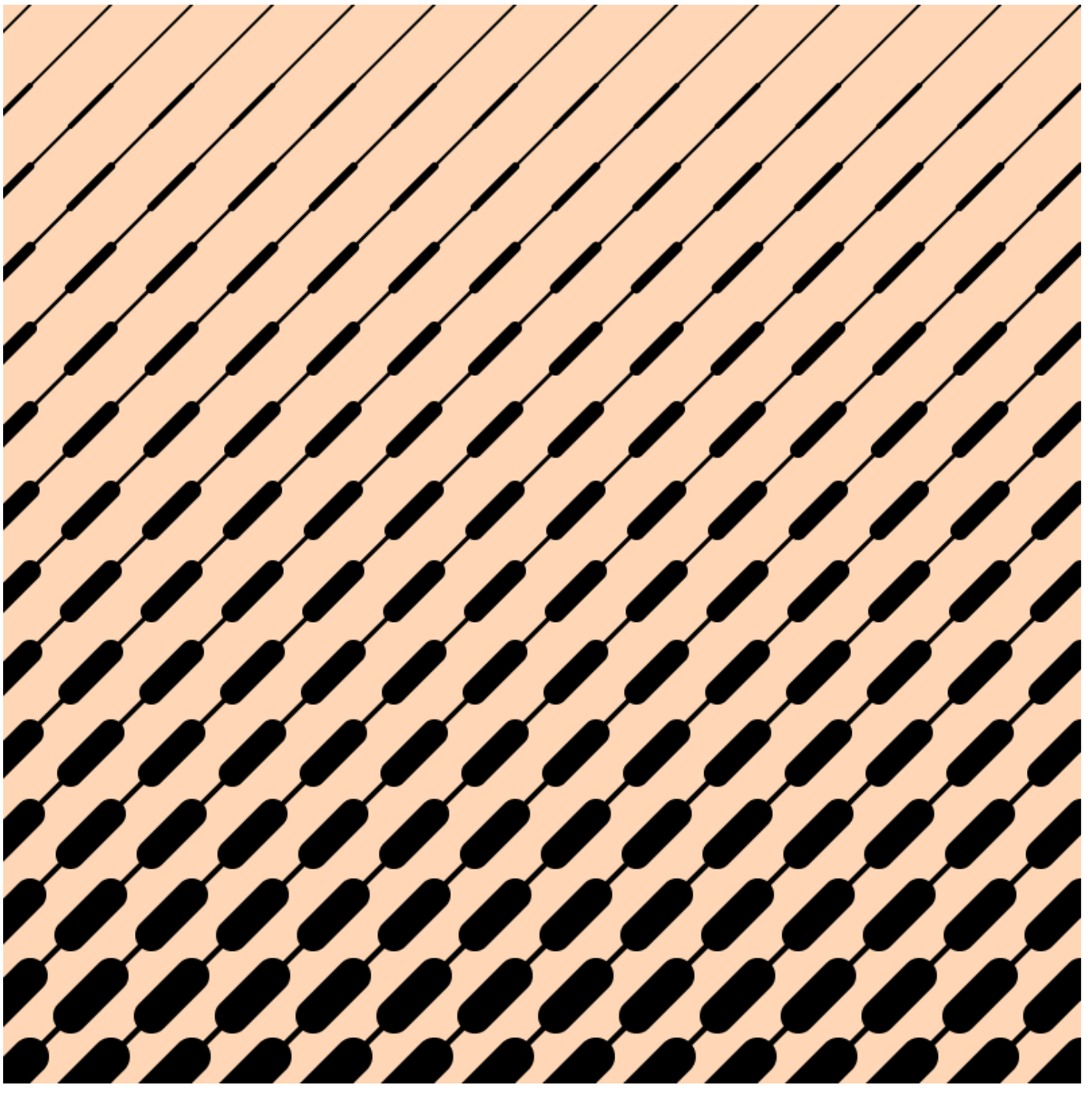
Notice how step is > 20, which is the jump we have in the lines.

This makes the gap effect!

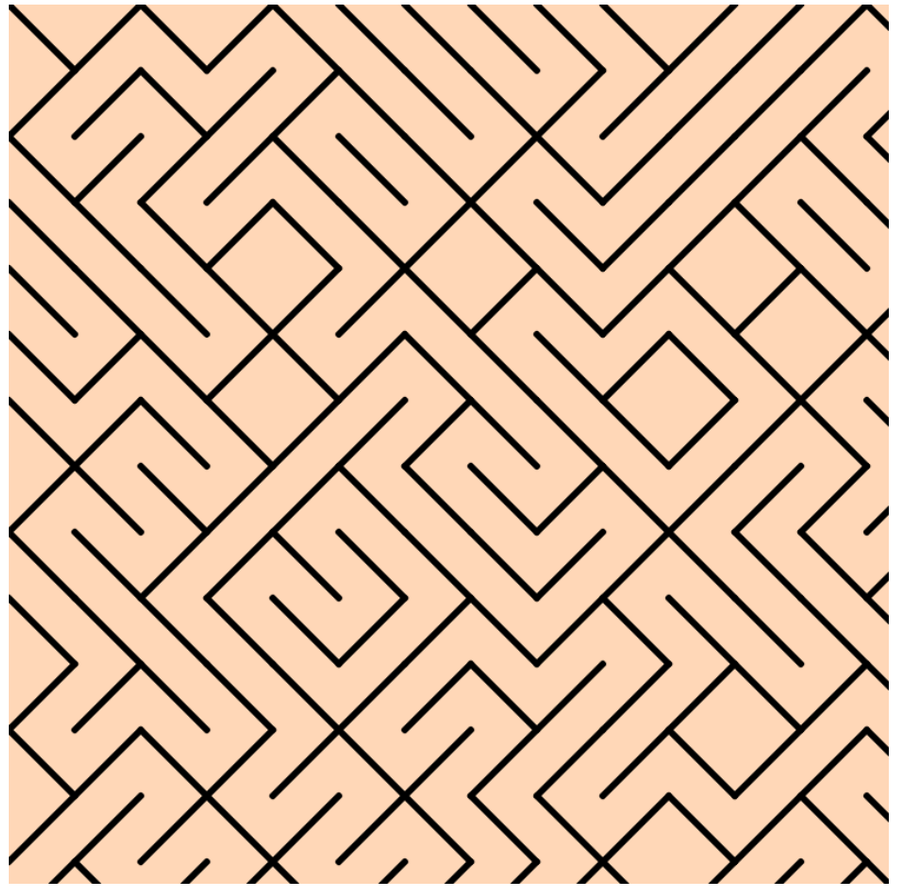
```
function setup() {
  createCanvas(400, 400);
  noLoop();
}
function draw() {
  background('PEACHPUFF');
  var step = 30;
  for (var x=0;x<400;x=x+step) {
    for (var y=0;y<400;y=y+step) {
      strokeWeight(1+x/step);
      line(x,y,x+(step/2),y+(step/2));
      strokeWeight(1+x/step/10);
      line(x+(step/2),y+(step/2),x+step,y+step);
    }
  }
}
```

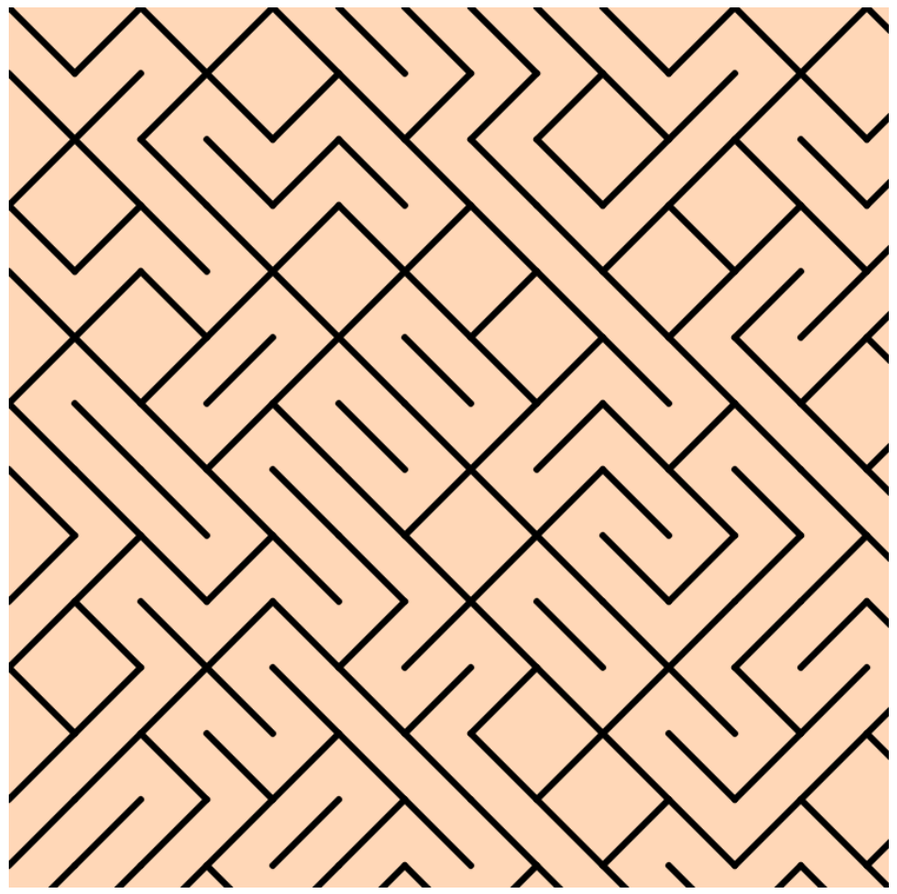
We change the `strokeWeight` to be heavier as we near (400,400)

The lines also have some that jump = step and
some < step (more precisely, step/2)

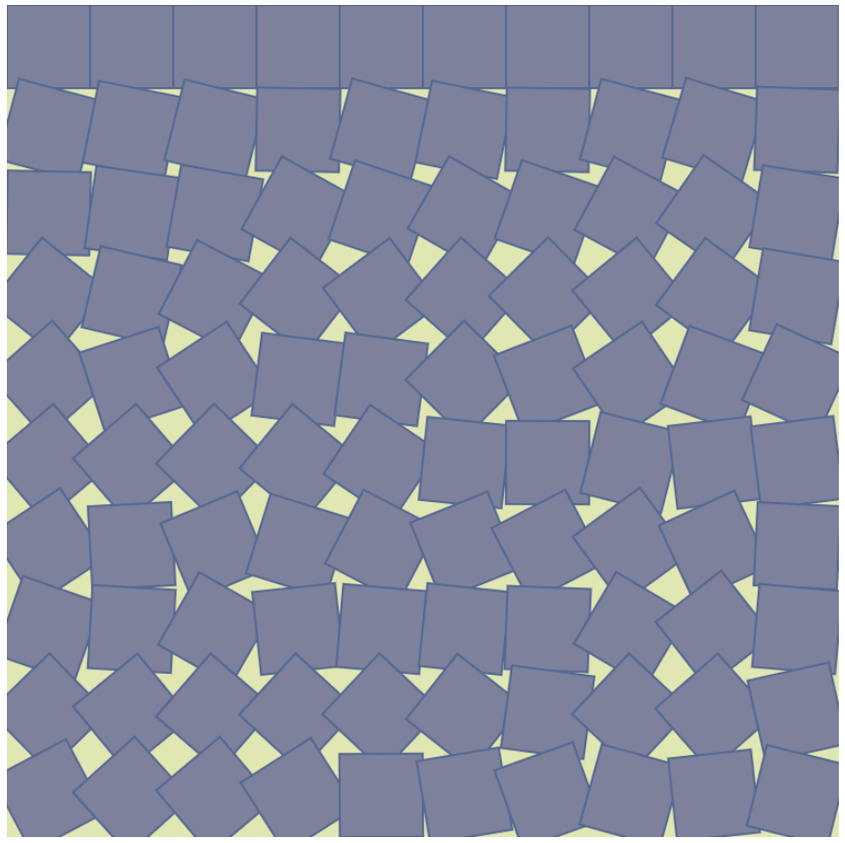


```
function setup() {
  createCanvas(400, 400);
  noLoop();
}
function draw() {
  background('PEACHPUFF');
  var step = 30;
  for (var x=0;x<400;x=x+step) {
    for (var y=0;y<400;y=y+step) {
      strokeWeight(3);
      if (random()>0.5) {
        line(x,y,x+step,y+step);
      }
      else {
        line(x+step,y,x,y+step);
      }
    }
  }
}
```

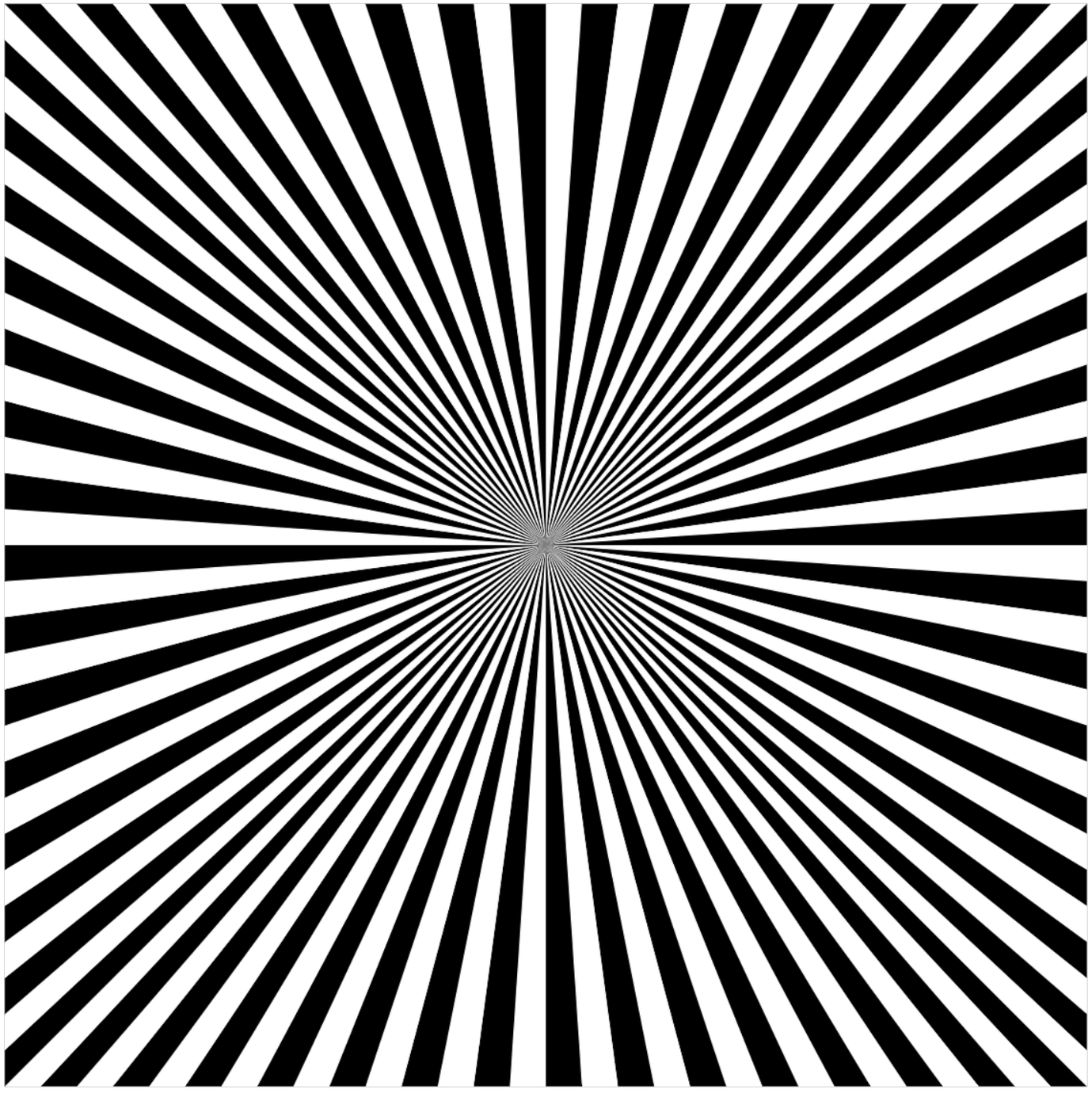




```
function setup() {
  createCanvas(400, 400);
  noLoop();
  rectMode(CENTER);
}

function draw() {
  background(229, 232, 182);
  var step = 40;
  for (var x = 0; x < 400; x = x + step) {
    for (var y = 0; y < 400; y = y + step) {
      drawRandomSquare(x, y, step);
    }
  }
}

function drawRandomSquare(x,y,step) {
    fill(125, 134, 156);
  stroke(88, 105, 148);
  push();
  translate(x+step/2,y+step/2);
  var maxRotation = map(y,0,400,0,PI);
  rotate( random(0,maxRotation) );
  rect(0,0,step,step);
  pop();
}
```

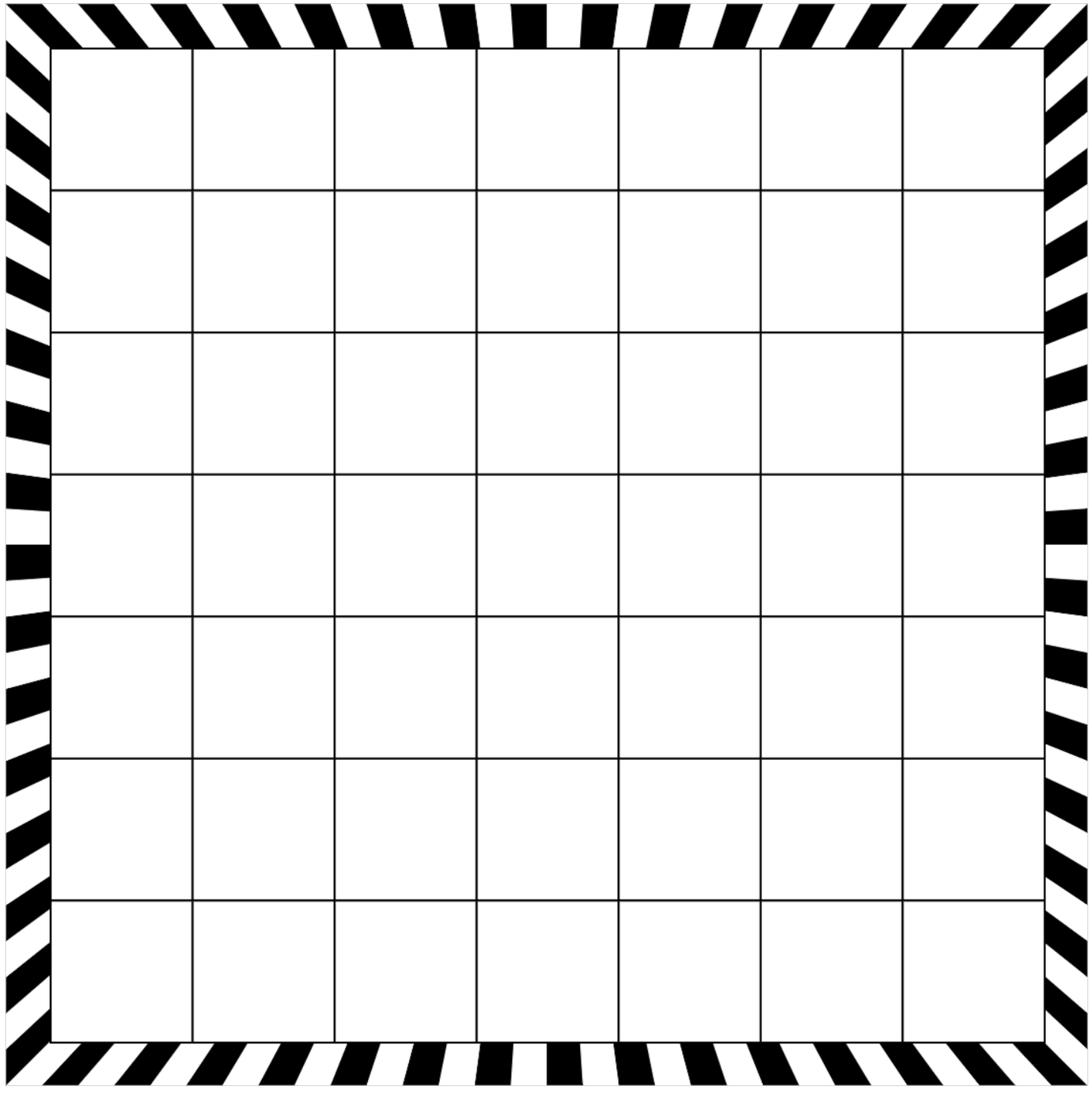


```
function setup() {
  createCanvas(800, 800);
  angleMode(DEGREES);
  rectMode(CENTER);
  const ctx = drawingContext;
  const x = width / 2;
  const y = height / 2;
  const squareSideDotsCount = 30;

  const squareVertices = [];
  let startAngle = 45;
  for (let i = 0; i < 4; i += 1) {
    squareVertices.push({
      x: 400 * cos(startAngle),
      y: 400 * sin(startAngle),
    });
    startAngle += 360 / 4;
  }

  const square = [];
  for (let i = 0; i < 4; i += 1) {
    for (let j = 0; j < squareSideDotsCount; j += 1) {
      const x = lerp(
        squareVertices[i].x,
        squareVertices[(i + 1) % squareVertices.length].x,
        j / squareSideDotsCount,
      );
      const y = lerp(
        squareVertices[i].y,
        squareVertices[(i + 1) % squareVertices.length].y,
        j / squareSideDotsCount,
      );
      square.push({ x, y });
    }
  }

  push();
  translate(x, y);
   for (let i = 0; i < square.length; i += 1) {
     push();
     noStroke();
     if (i % 2 === 0) fill(0);
     else fill(255);
     beginShape();
     vertex(square[i].x, square[i].y);
     vertex(0, 0);
     vertex(
        square[(i + 1) % square.length].x,
        square[(i + 1) % square.length].y,
      );
     endShape(CLOSE);
     pop();
   }
  pop();
}
```



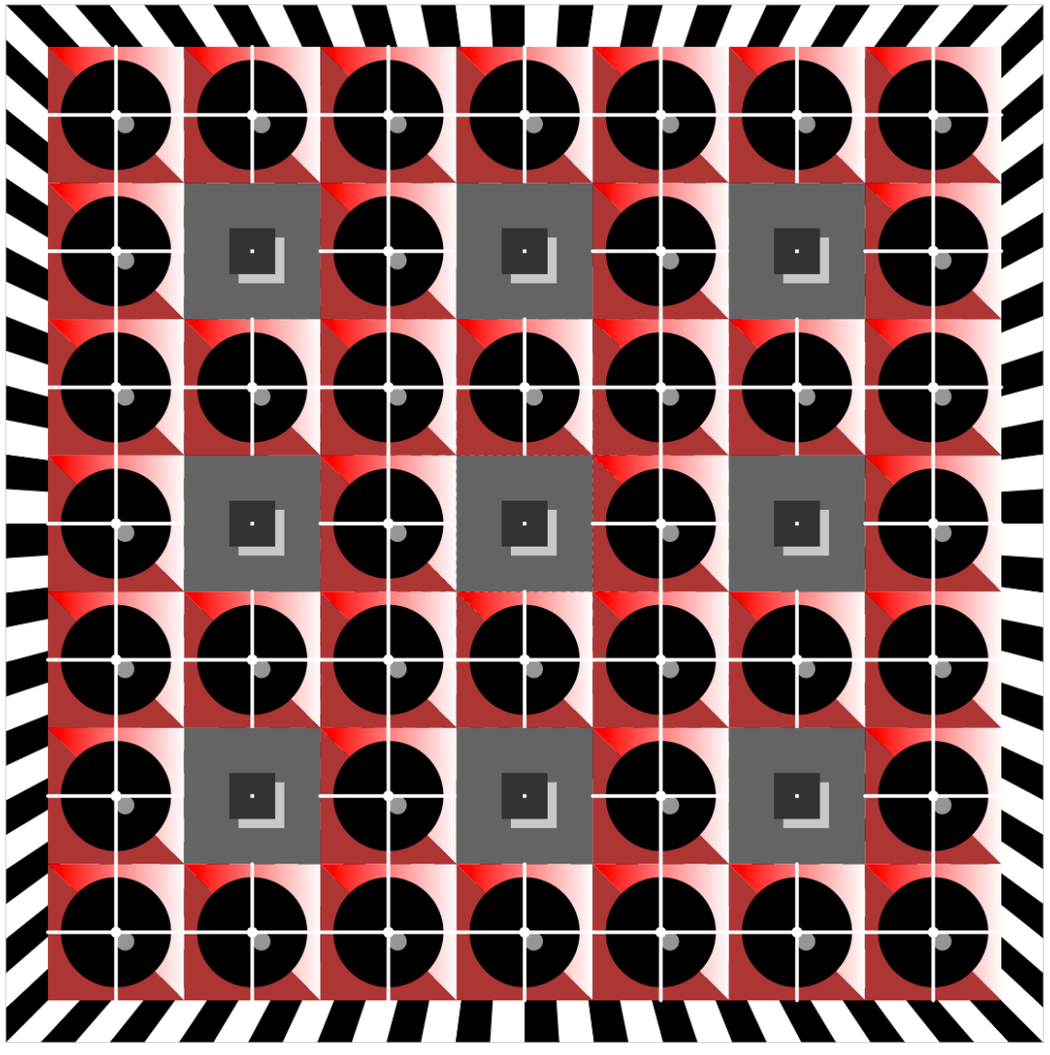
https://gist.github.com/ClaireBookworm/5df4783f15f018d6f83c5aac32c6b0a0

```
function setup() {
  createCanvas(800, 800);
  angleMode(DEGREES);
  rectMode(CENTER);
  const ctx = drawingContext;
  const x = width / 2;
  const y = height / 2;
  const squareSideDotsCount = 30;
  stroke(0);

  const squareVertices = [];
  let startAngle = 45;
  for (let i = 0; i < 4; i += 1) {
    squareVertices.push({
      x: 400 * cos(startAngle),
      y: 400 * sin(startAngle),
    });
    startAngle += 360 / 4;
  }

  const square = [];
  for (let i = 0; i < 4; i += 1) {
    for (let j = 0; j < squareSideDotsCount; j += 1) {
      const x = lerp(
        squareVertices[i].x,
        squareVertices[(i + 1) % squareVertices.length].x,
        j / squareSideDotsCount,
      );
      const y = lerp(
        squareVertices[i].y,
        squareVertices[(i + 1) % squareVertices.length].y,
        j / squareSideDotsCount,
      );
      square.push({ x, y });
    }
  }

  push();
  translate(x, y);
   for (let i = 0; i < square.length; i += 1) {
     push();
     noStroke();
     if (i % 2 === 0) {
        fill(0);
      } else {
        fill(255);
      }
     beginShape();
     vertex(square[i].x, square[i].y);
     vertex(0, 0);
     vertex(
        square[(i + 1) % square.length].x,
        square[(i + 1) % square.length].y,
      );
     endShape(CLOSE);
     pop();
   }

  const innerRectSide = 520;
  const cellCount = 7;
  const grid = [];
  const pointCount = cellCount ** 2;
  const cellSide = innerRectSide / cellCount;
  const startPoint = -(cellSide * (cellCount - 1)) / 2;
  for (let rowIndex = 0; rowIndex < cellCount; rowIndex += 1) {
        for (let colIndex = 0; colIndex < cellCount; colIndex += 1) {
            grid.push({
              x: startPoint + colIndex * cellSide,
              y: startPoint + rowIndex * cellSide,
            });
        }
    }
  for (let rowIndex = 0; rowIndex < cellCount; rowIndex += 1) {
        for (let colIndex = 0; colIndex < cellCount; colIndex += 1) {
            const x = grid[rowIndex * cellCount + colIndex].x;
            const y = grid[rowIndex * cellCount + colIndex].y;
            rect(x, y, cellSide, cellSide)
        }
  }
  pop();
}
```

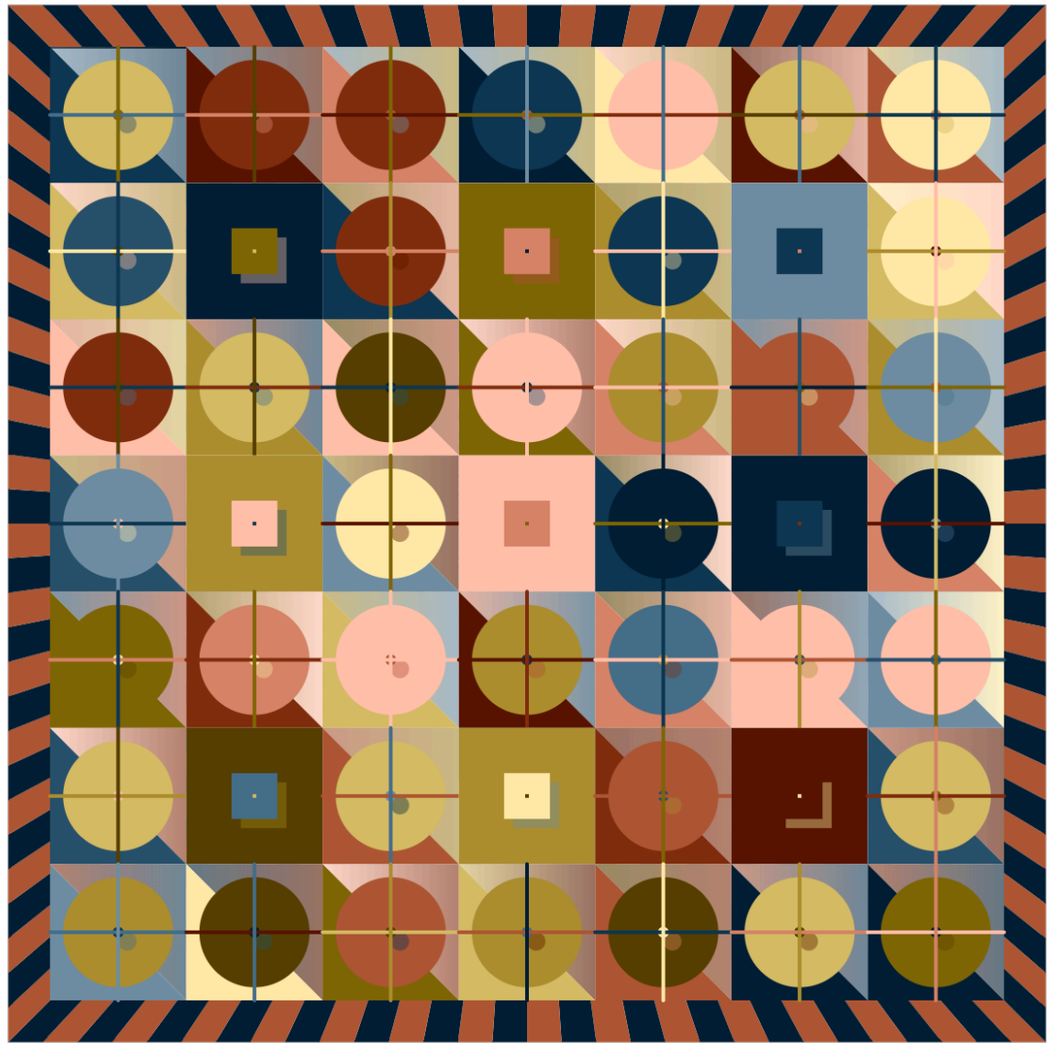


https://gist.github.com/ClaireBookworm/ebf3379ab6fb6840d5e6dfd70b05d62b

https://codepen.io/Shvembldr/pen/zbqpBp



https://codepen.io/Shvembldr/pen/pYypqd



openprocessing.org

Creative coding — find and share!

Beginner's Generative Art in P5.js CheatSheet

Lines


```
line(startX, startY, endX, endY)
strokeWeight(weight)
```

Shapes


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


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


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

Structure

```
function setup(){
  // called when the script starts running
}

function draw(){
  // called infinitely until noLoop() is called
}
```

Helpful Functions

```
createCanvas(height, width)
random(min, max)
random(choices)
frameRate(rate)
noLoop()
```

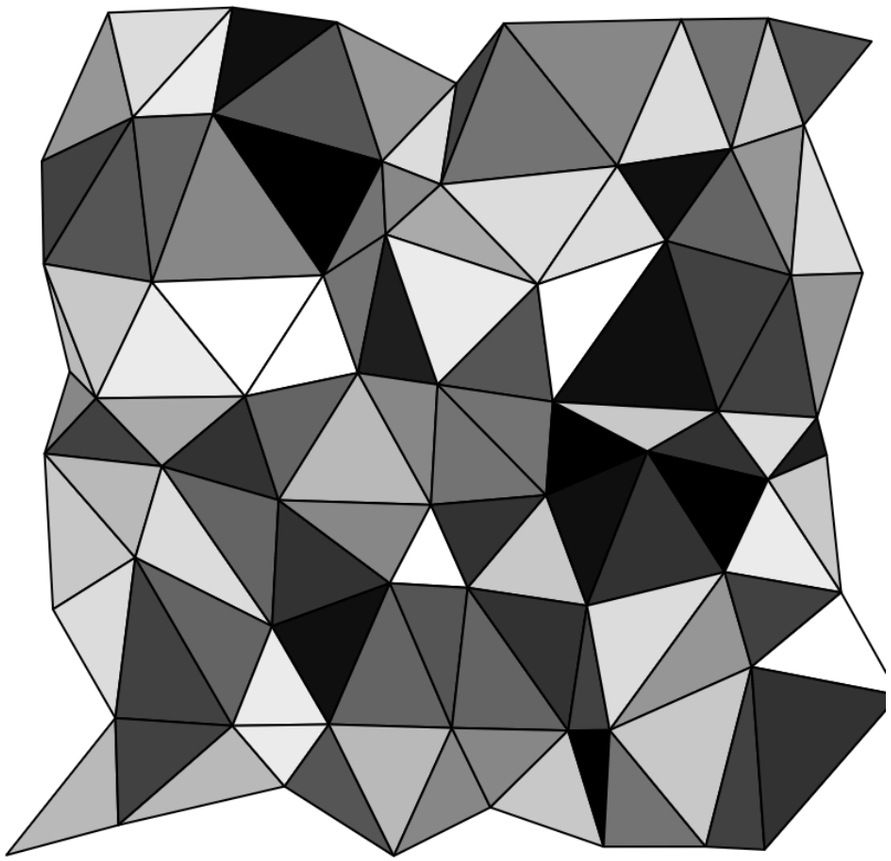
Helpful Variables

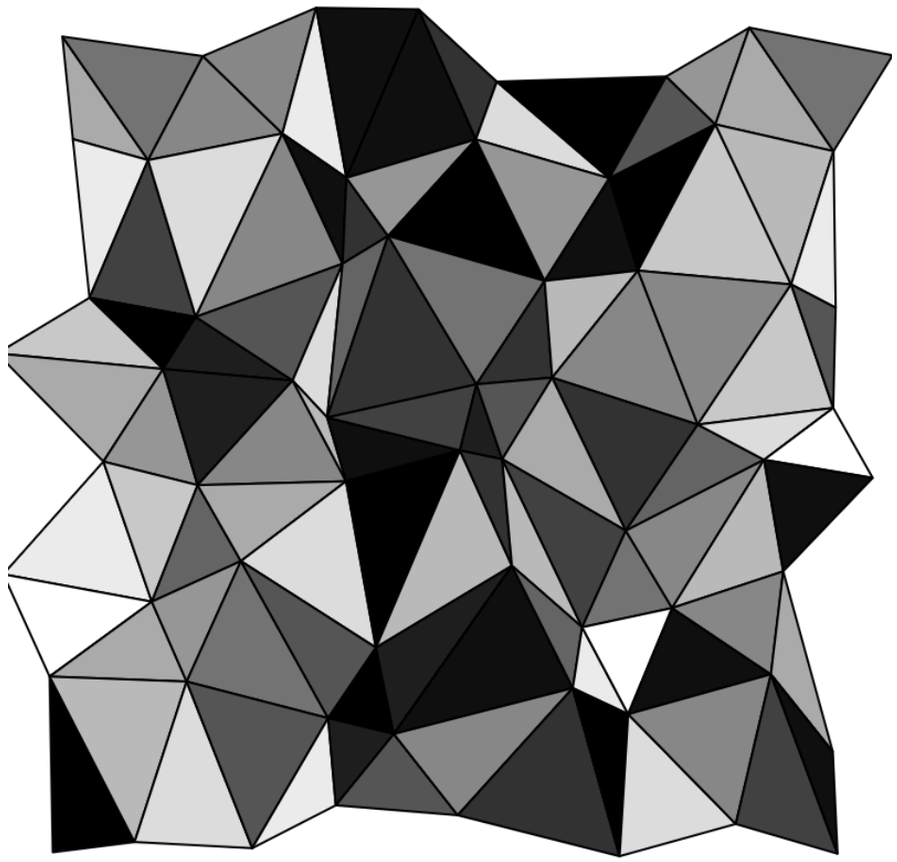
```
windowWidth
windowHeight
keyIsPressed
keyCode
```

Color

```
fill(color)
fill(red, green, blue, opacity)
stroke(red, green, blue, opacity)
```

Full Reference p5js.org/reference/





https://gist.github.com/ClaireBookworm/04b139695749f53ad896df0c67668f3c

```
var canvas = document.querySelector('canvas');
var context = canvas.getContext('2d');

var size = window.innerWidth;
var dpr = window.devicePixelRatio;
canvas.width = size * dpr;
canvas.height = size * dpr;
context.scale(dpr, dpr);
context.lineJoin = 'bevel';
```

Now let’s make a **grid of dots**.

For every dot coordinate we will draw it on the canvas, but also store the coordinate in an array for future use.

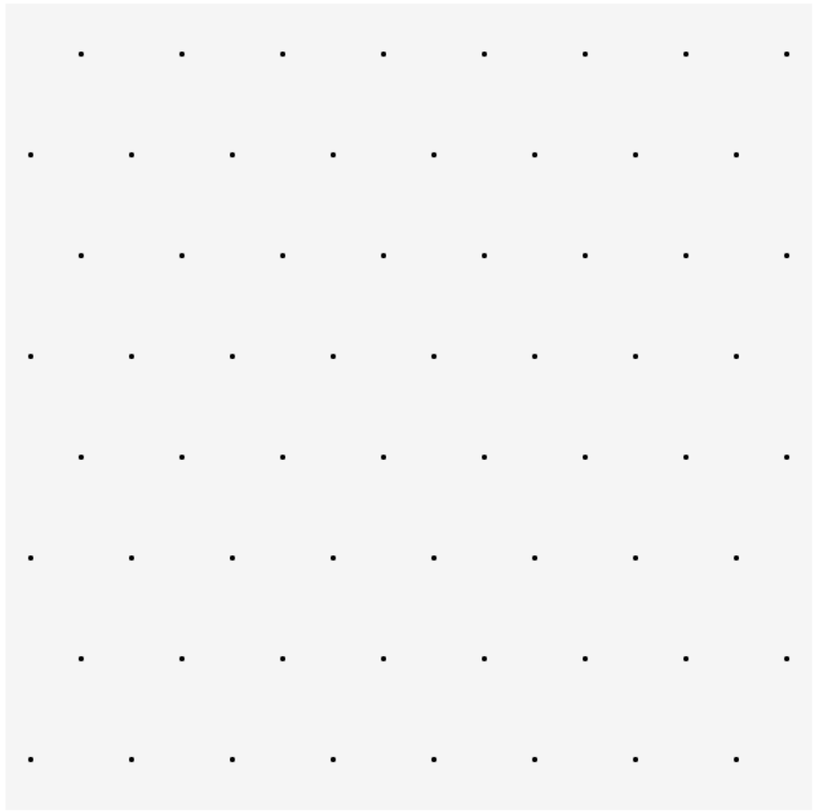
Every coordinate will be represented by an object with 2 properties: x and y.

The space between lines and columns is defined by the variable gap, we’ll draw these circles so we can see how our grid is placed out on the canvas.

```
var canvas = document.querySelector('canvas');
var context = canvas.getContext('2d');

var size = window.innerWidth;
var dpr = window.devicePixelRatio;
canvas.width = size * dpr;
canvas.height = size * dpr;
context.scale(dpr, dpr);
context.lineJoin = 'bevel';

var line,
    lines = [],
    gap = size / 7;

for(var y = gap / 2; y <= size; y+= gap) {
  line = [];
  for (var x = gap / 2; x <= size; x+= gap) {
    line.push({x: x, y: y});
    context.beginPath();
    context.arc(x, y, 1, 0, 2 * Math.PI, true);
    context.fill();
  }
  lines.push(line);
}
```

Now, we're going to displace every other line on the x axis. We do this by alternating the variable called odd between true and false. We can see that the new pattern is shaping up to be a mesh of regular triangles.

```
var line, dot,
    odd = false,
    lines = [],
    gap = size / 8;

for(var y = gap / 2; y <= size; y+= gap) {
  odd = !odd;
  line = [];
  for(var x = gap / 4; x <= size; x+= gap) {
    dot = {x: x + (odd ? gap/2 : 0), y: y};
    line.push(dot);
    context.beginPath();
    context.arc(dot.x, dot.y, 1, 0, 2 * Math.PI, true);
    context.fill();
  }
  lines.push(line);
}
```



The next step will be using the dots to draw the triangles. To make our life easier let's make a function that take the 3 coordinates of a triangle and draw them together.

```
function drawTriangle(pointA, pointB, pointC) {
  context.beginPath();
  context.moveTo(pointA.x, pointA.y);
  context.lineTo(pointB.x, pointB.y);
  context.lineTo(pointC.x, pointC.y);
  context.lineTo(pointA.x, pointA.y);
  context.closePath();
  context.stroke();
}
```

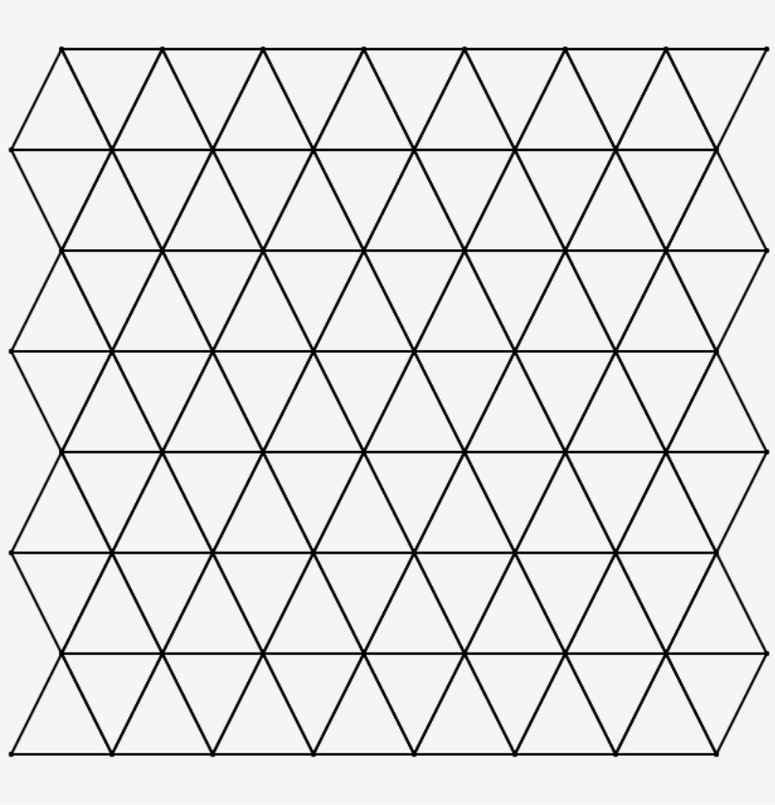
This is called a ***helper function***.

Now we'll string up our `drawTriangle` function, and use the dots we generated earlier to draw all the triangles.

This part might be a bit complex to understand. The script is going to go through all the lines and combining the two dots of the sibling line, forming triangles. The concatenation of two lines, let's say line `a` and line `b`, and merge the dots into one array to make it look like a zig-zag: `a1`, `b1`, `a2`, `b2`, `a3` etc.

This will give us an array, containing each triangles specific coordinates. Looking something like [`a1`, `b1`, `a2`], [`b1`, `a2`, `b2`], [`a2`, `b2`, `a3`]...

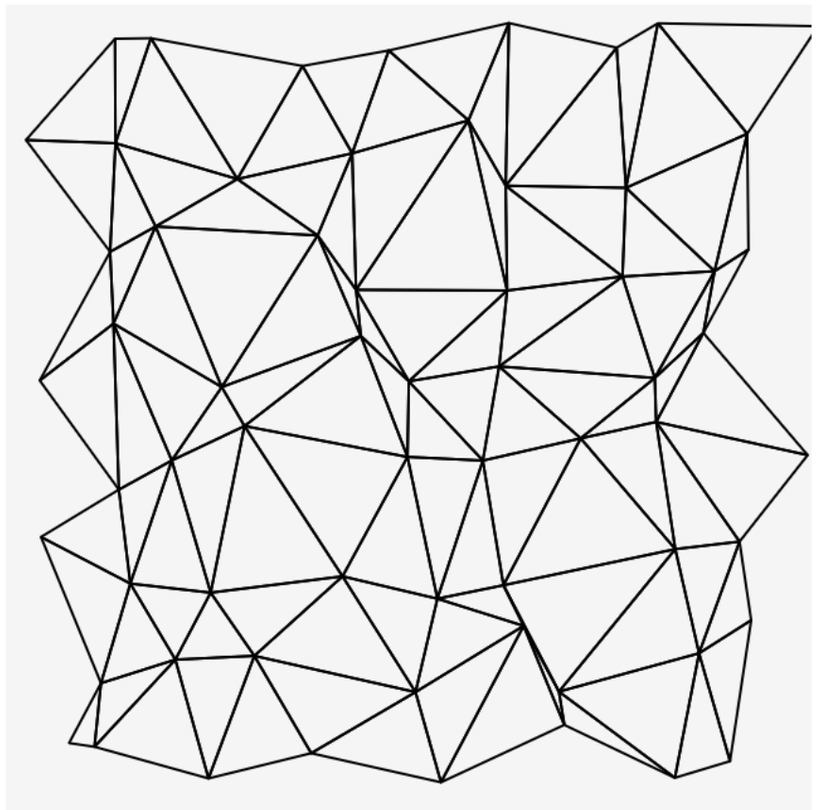
```
var dotLine;
odd = true;

for(var y = 0; y < lines.length - 1; y++) {
  odd = !odd;
  dotLine = [];
  for(var i = 0; i < lines[y].length; i++) {
    dotLine.push(odd ? lines[y][i]   : lines[y+1][i]);
    dotLine.push(odd ? lines[y+1][i] : lines[y][i]);
  }
  for(var i = 0; i < dotLine.length - 2; i++) {
    drawTriangle(dotLine[i], dotLine[i+1], dotLine[i+2]);
  }
}
```



Now that we have a regular triangle mesh, we are one detail away from getting the magic to happen. Every dot is a gap away from the surrounding dots. So a dot can be moved in this area without overlapping with other dots.

Let’s use a bit of `Math.random()` to get a random position in this area.

```
//setup code here

var line, dot,
    odd = false,
    lines = [],
    gap = size / 8;

for(var y = gap / 2; y <= size; y+= gap) {
  odd = !odd;
  line = [];
  for(var x = gap / 4; x <= size; x+= gap) {
    dot = {x: x + (odd ? gap/2 : 0), y: y};
    line.push({
      x: x + (Math.random()*.8 - .4) * gap  + (odd ? gap/2 : 0),
      y: y + (Math.random()*.8 - .4) * gap
    });
    context.fill();
  }
  lines.push(line);
}

function drawTriangle(pointA, pointB, pointC) {
// helper function code here
}

var dotLine;
odd = true;

for(var y = 0; y < lines.length - 1; y++) {
  odd = !odd;
  dotLine = [];
  for(var i = 0; i < lines[y].length; i++) {
    dotLine.push(odd ? lines[y][i]   : lines[y+1][i]);
    dotLine.push(odd ? lines[y+1][i] : lines[y][i]);
  }
  for(var i = 0; i < dotLine.length - 2; i++) {
    drawTriangle(dotLine[i], dotLine[i+1], dotLine[i+2]);
  }
}
```



```
function drawTriangle(pointA, pointB, pointC) {
    context.beginPath();
    context.moveTo(pointA.x, pointA.y);
    context.lineTo(pointB.x, pointB.y);
    context.lineTo(pointC.x, pointC.y);
    context.lineTo(pointA.x, pointA.y);
    context.closePath();
    var gray = Math.floor(Math.random()*45).toString(16);
    context.fillStyle = '#' + gray + gray + gray;
    context.fill();
    context.stroke();
  }
```

What you multiply random() by is the number of shades.

45 is a lot, 4 is too little — experiment!

I ended up using 16.