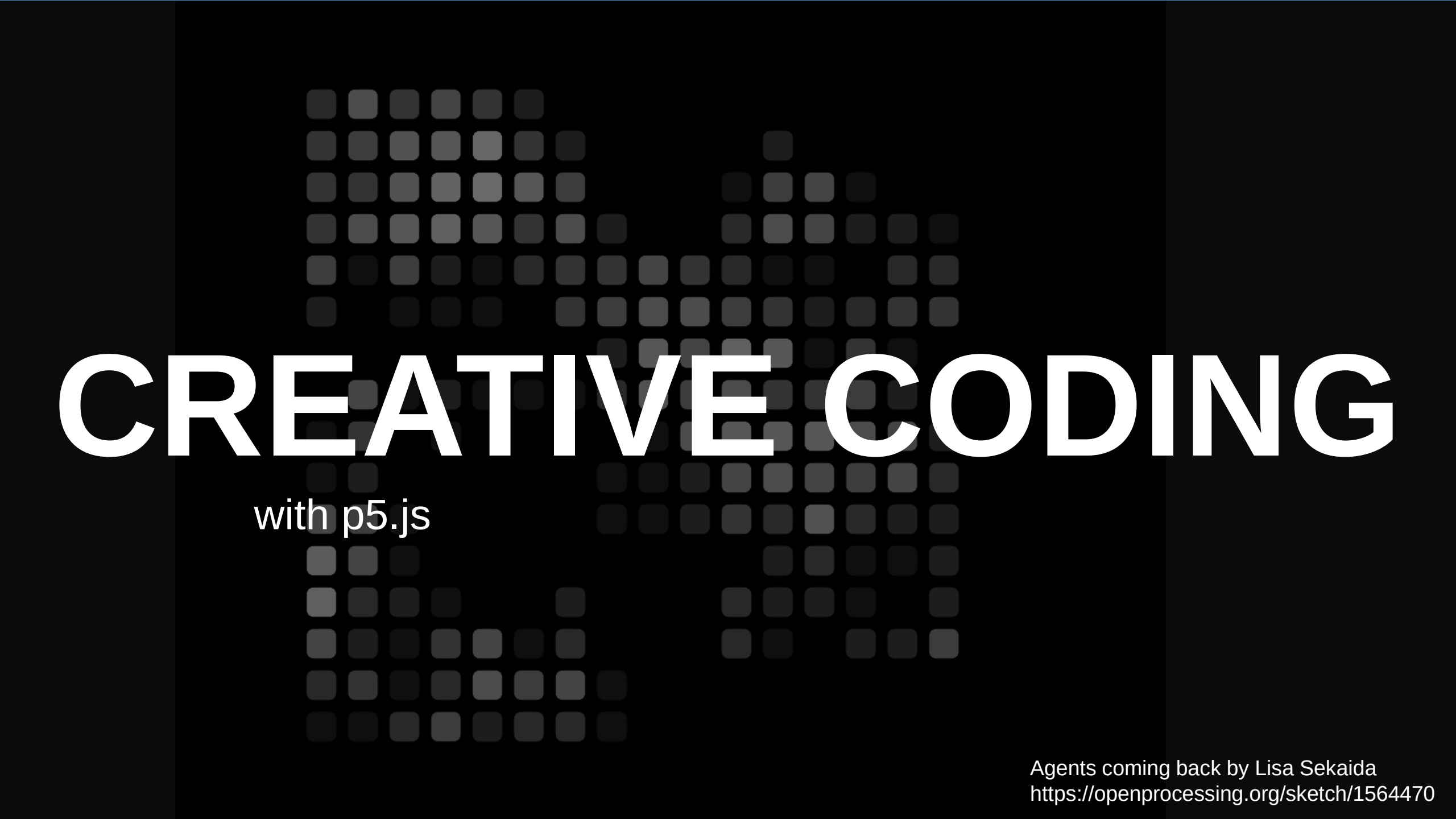
CREATIVE CODING

with p5.js



Agents coming back by Lisa Sekaida
https://openprocessing.org/sketch/1564470

Agents…

… are individual and autonomous actors or entities in a system

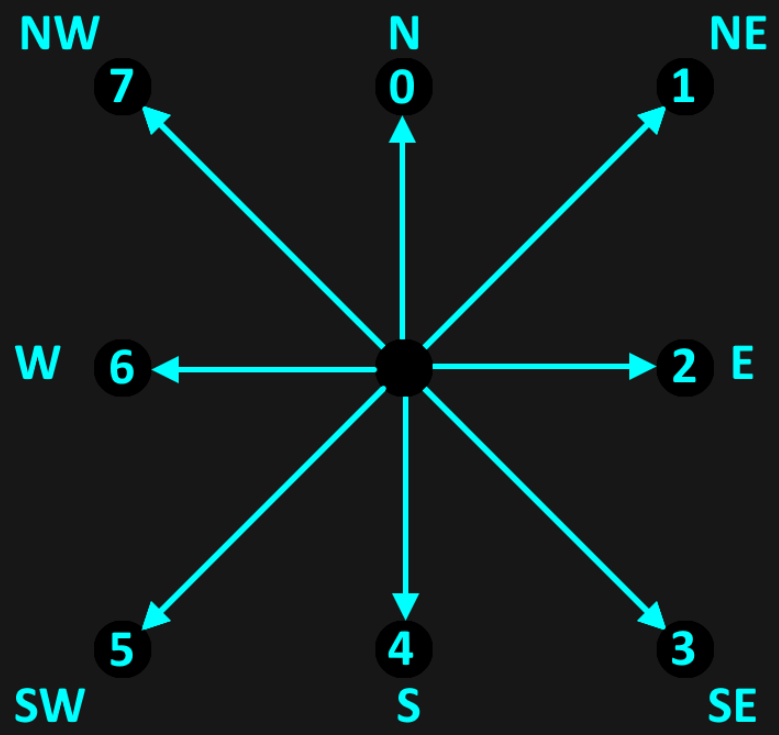
… can simulate organisms, particles, life simulation etc.

… don’t require interaction of user

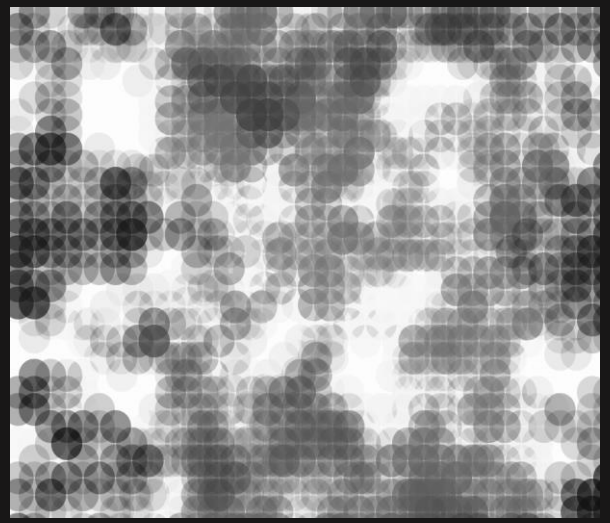
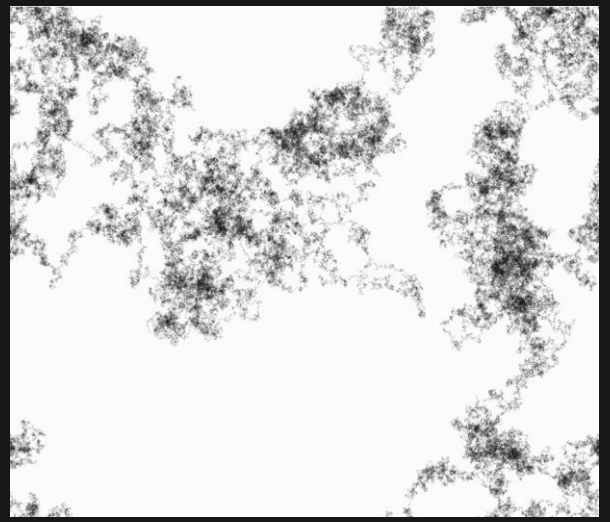
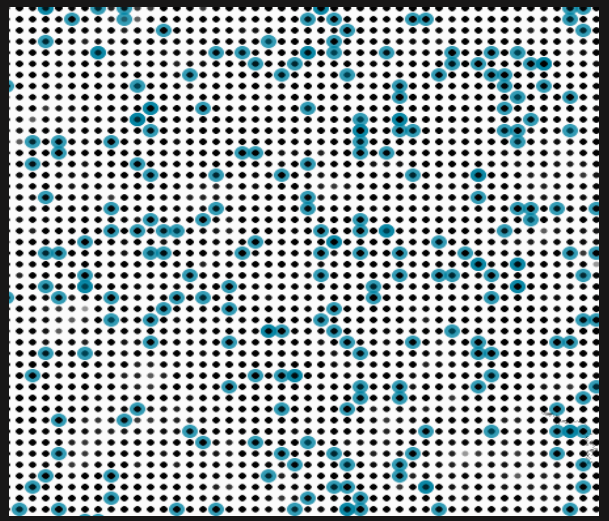
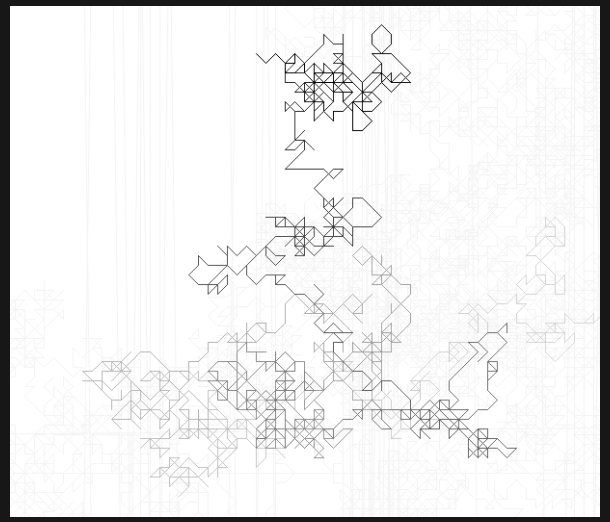
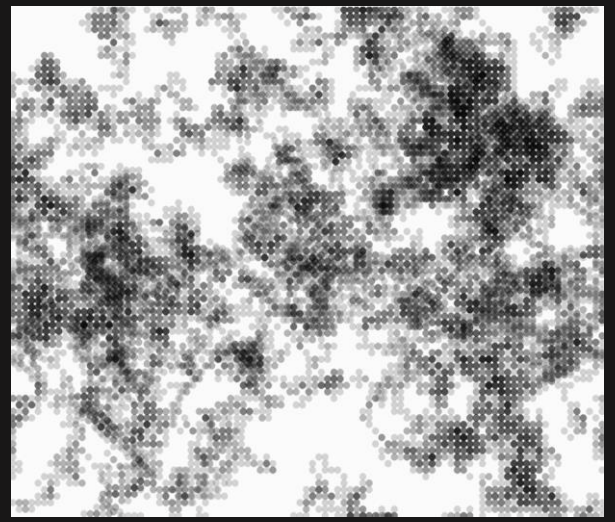
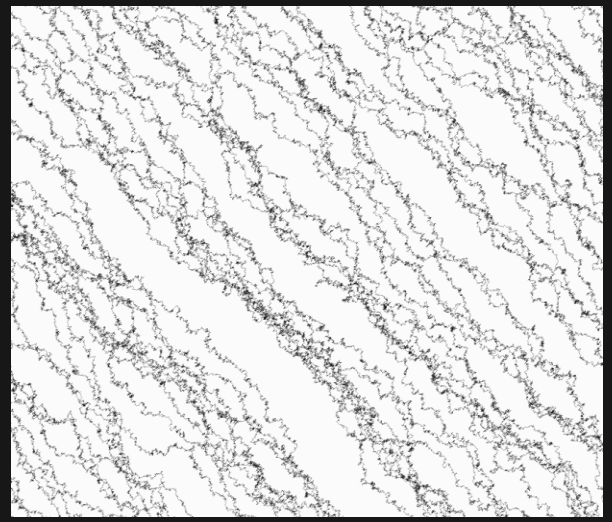
… can be controlled by user input or by algorithms that adjust their behavior based on external factors

Random Directions

- Declare directions (NORTH = 0, NORTHEAST = 1, …
- Select a random number (0-7)
- Change x/y position based on direction
- Handle edge cases (width < xPos || xPos < 0), same for y
- Draw something based on xPos yPos



Random Directions

Reference tips

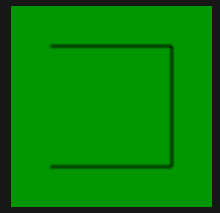
- clear()
  - clear() makes every pixel 100% transparent.
- get(x, y, w, h) || get(x,y)
  - Returns a region of pixels or a single pixel
  - Don’t forget to call LoadPixels() before
- dist(x1, y1, x2, y2)
  - Returns a distance between 2 points
- map(value, inMin, inMax, outMin, outMax)
  - Example: map(mouseX, 0, width, 0, 1)
- noLoop(), loop(), isLooping(),
  - Handling of loops
- floor(n), ceil(n)
  - floor(5.7) // 5
  - ceil(2.1) // 3
- pow(n, e)
  - Exponential expressions
  - pow(2,3) = $2^3 = 8$

Demonstrate clear()

```
let graphics;

function setup() {
  createCanvas(400, 400);
  graphics = createGraphics(200, 200); // Create an off-screen graphics
buffer
}

function draw() {
  background(220);

  // Draw on the main canvas
  fill(255, 0, 0); // Red
  rect(50, 50, 100, 100);

  // Draw on the graphics buffer
  graphics.fill(0, 0, 255); // Blue
  graphics.noStroke();
  graphics.ellipse(random(graphics.width), random(graphics.height), 20,
20);

  // Display the graphics buffer on the main canvas
  image(graphics, 100, 100);
}

function mousePressed() {
  // Clear the graphics buffer when the mouse is clicked
  graphics.clear();
}
```

Vertex

- beginShape([kind]), endShape()
  - either POINTS, LINES, TRIANGLES, TRIANGLE_FAN TRIANGLE_STRIP, QUADS, QUAD_STRIP or TESS
- vertex, curveVertex



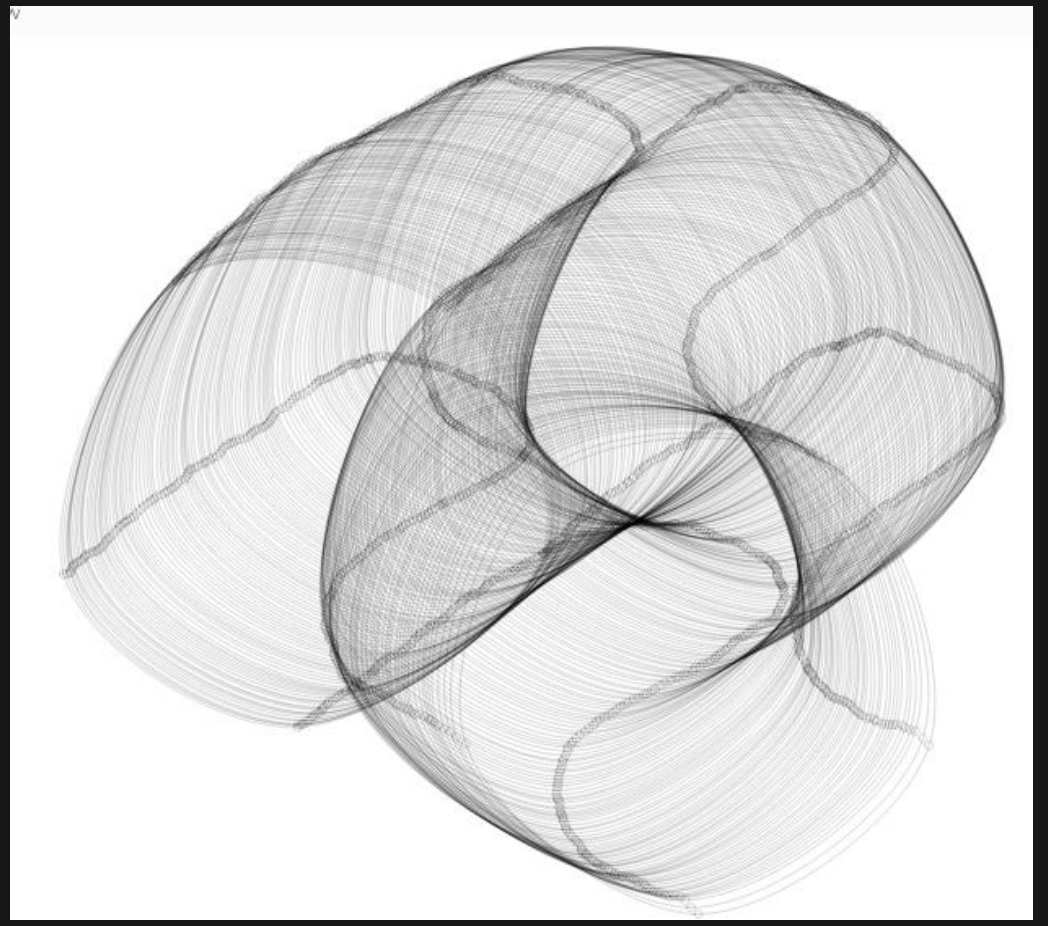
```
function setup() {
  createCanvas(100, 100)
  background(0, 150, 0);
  noFill();
  beginShape();
  vertex(20, 20);
  vertex(80,20);
  vertex(80, 80);
  vertex(20, 80);
  endShape(); // endShape(CLOSE)
}

```

```
function setup() {
  createCanvas(100, 100);
  background(0, 150, 0);
  noFill();
}

function draw() {
  clear();
  beginShape();
  curveVertex(20, 20);
  curveVertex(20, 20);

  curveVertex(mouseX, mouseY);
  curveVertex(80, 80);

  curveVertex(20, 80);
  curveVertex(20, 80);
  endShape();
}

```

The first and last points in a series of curveVertex() lines will be used to guide the beginning and end of the curve

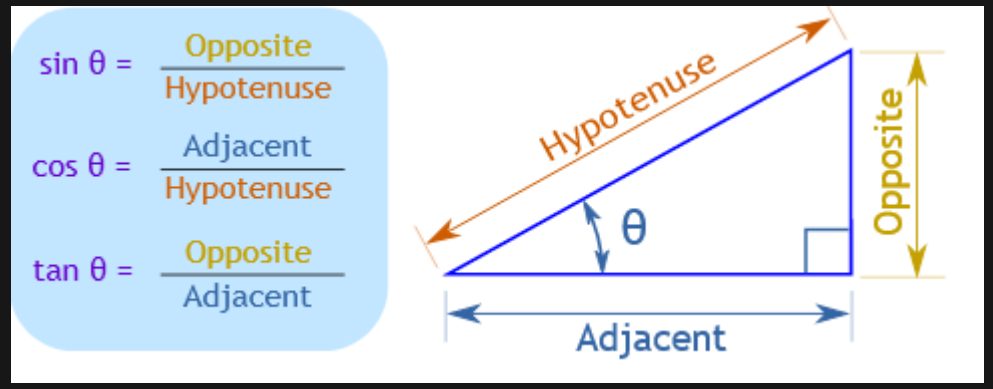
Shapes

```
1  const amountOfFormPoints = 5;
2  const stepSize = 2;
3  const initRadius = 150;
4  const mouseAttraction = 0.01;
5  let centerX, centerY;
6  let x = [];
7  let y = [];
```

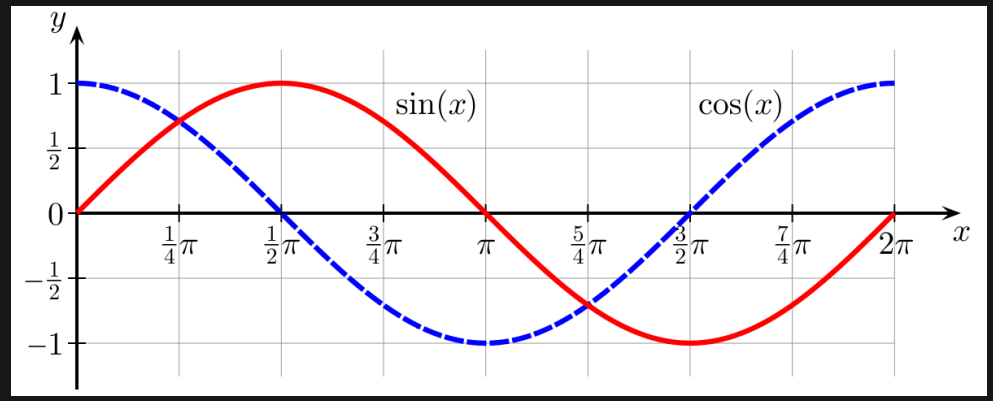


Shapes

```
1  function setup() {
2    createCanvas(windowWidth, windowHeight);
3
4    // initial shape
5    centerX = width / 2;
6    centerY = height / 2;
7    const angle = radians(360 / amountOfFormPoints);
8    for (let i = 0; i < amountOfFormPoints; i++) {
9      x.push(cos(angle * i) * initRadius);
10     y.push(sin(angle * i) * initRadius);
11   }
12
13   // styling
14   stroke(0, 75);
15   strokeWeight(0.5);
16   background(255);
17   noFill();
18 }
```



https://www.mathsisfun.com/sine-cosine-tangent.html



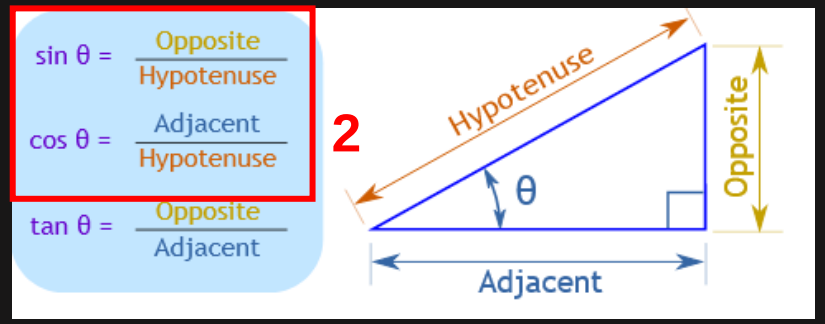
https://de.wikipedia.org/wiki/Sinus_und_Kosinus

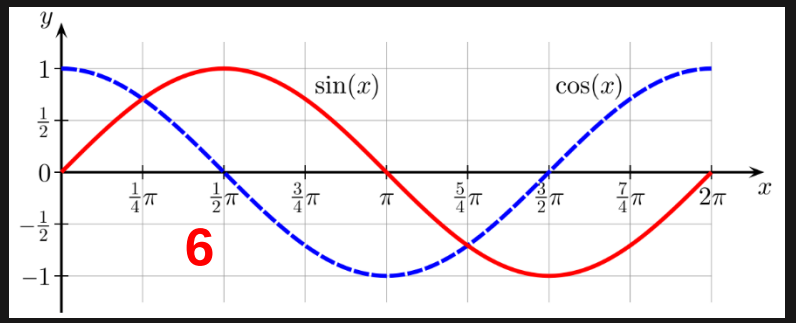
Shapes – in Detail

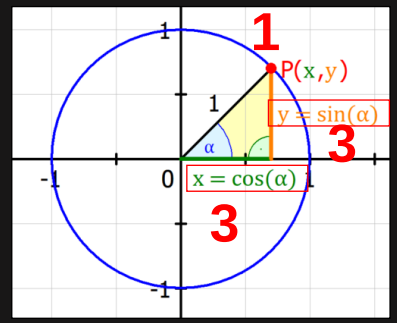
```
1  function setup() {
2    createCanvas(windowWidth, windowHeight);
3
4    // initial shape
5    centerX = width / 2;
6    centerY = height / 2;
7    const angle = radians(360 / amountOfFormPoints);
8    for (let i = 0; i < amountOfFormPoints; i++) {
9      x.push(cos(angle * i) * initRadius);
10     y.push(sin(angle * i) * initRadius);
11   }
12
13   // styling
14   stroke(0, 75);
15   strokeWeight(0.5);
16   background(255);
17   noFill();
18 }
```

6 4 5

1. A point P in a unit circle is defined as P(x,y).
2. By converting the formulas, we get …
3. … the x-value and the y-value of the point P
    1. x = cos(α), y = sin(α)
    2. Not in the image visible due to hypotenuse = 1
4. Radius = Hypotenuse; We want to have a different radius than 1, so we need to add that to the formula as follows:
    1. x = cos(α) * hypotenuse , y = sin(α) * hypotenuse
5. The formulas in 4.1 only calculates one point. We want to have multiple points, so we need to put that in a for-loop and add it to the angle
6. Why are we calculating into radians?
    1. By default p5.js will expect angles to be in radians.
    2. https://p5js.org/learn/getting-started-in-webgl-coords-and-transform.html

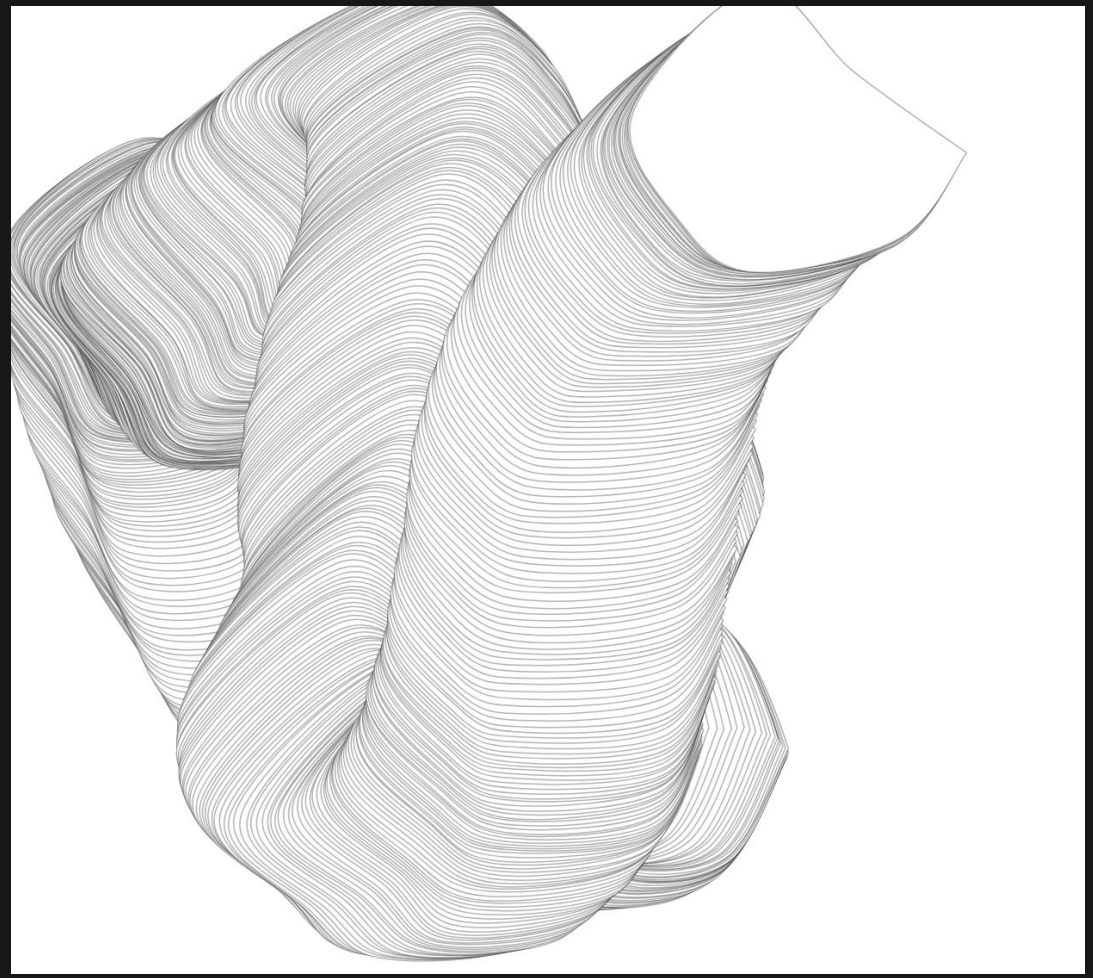






Sources: Slide before and https://studyflix.de/mathematik/einheitskreis-2061

Shapes

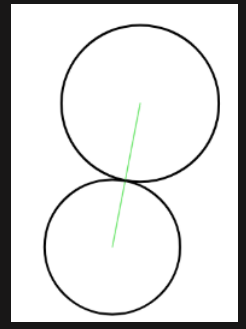


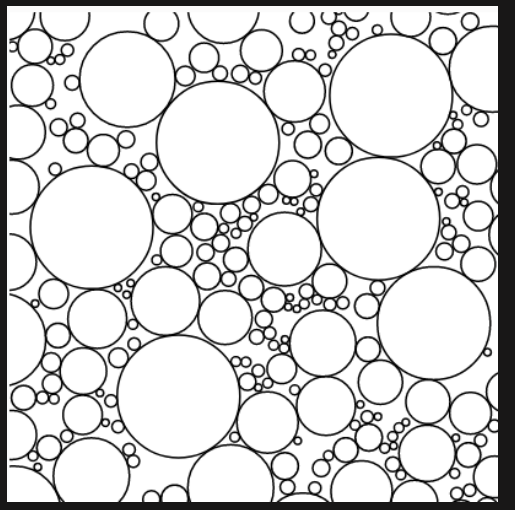
```
function draw() {
  // float towards mouse position
  centerX += (mouseX - centerX) * mouseAttraction;
  centerY += (mouseY - centerY) * mouseAttraction;

  // calculate new points
  for (let i = 0; i < amountOfFormPoints; i++) {
    x[i] += random(-stepSize, stepSize);
    y[i] += random(-stepSize, stepSize);
    ellipse(x[i] + centerX, y[i] + centerY, 5, 5); // show points
  }

  beginShape();
  // first controlPoint
  curveVertex(x[0] + centerX, y[0] + centerY);

  // only these points are drawn
  for (let i = 0; i < amountOfFormPoints; i++) {
    curveVertex(x[i] + centerX, y[i] + centerY);
  }

  // Connect to the first poing again
  // or use endShape(CLOSE); but result is different
  curveVertex(x[0] + centerX, y[0] + centerY);

  // end controlPoint
  curveVertex(
    x[amountOfFormPoints - 1] + centerX,
    y[amountOfFormPoints - 1] + centerY
  );
  endShape();
}
```

Abstract algorithm

Draw circles at random position (not overlapping), with random radius

- Once: Draw a starting circle (random pos, random radius). Push it to the circles array
- Generate a random point and check, that the point is not on or inside of any circle
  - (Distance from point to any circle) – (radius of any circle) must be > 0
- Get the shortest distance to every other circle already drawn
- Check if the resulting distance is larger than the max allowed radius defined by you
- If not, change the random radius to (distance – radius to other circle)
- Draw the circle
- Push the circle to the other circles array
- Repeat ☺





Tasks

1. Implement the given code for «Shapes»
    1. Play around with it and try to manipulate / change the code

2. Create an agent, which makes use of
    - Randomness
    - State changes (position based on directions/angles, colors etc)

3. Implement the abstract algorithm

4. Experiment with
    - «collision» (get(x,y) pixels, check for color) or
    - shapes and forms
    - And make a sketch based on those experiments.