Lecture 06: Graphics and Objects

COSC 225: Algorithms and Visualization

Spring, 2023

Outline

1. Assignment 4 Notes
2. Scalable Vector Graphics
3. Activity: Draw a Cat
4. JavaScript Objects

Steps for CA Simulation

Given:

135

1. Rule = # from 0 to 255
2. Configuration = 0/1 array

[0, 1, 1, 0, 1, 0]

Compute: updated configuration

[? ? ?]

How?

1. convert rule number to binary to get update rules
2. apply update rule to each 3 consecutive entries of configuration

Another CA Example

Pick a random number: 11

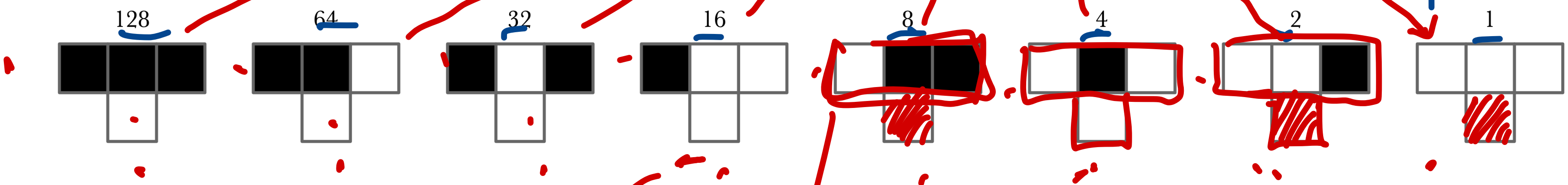
8 + 3

8 4 2 1

1000 + 11

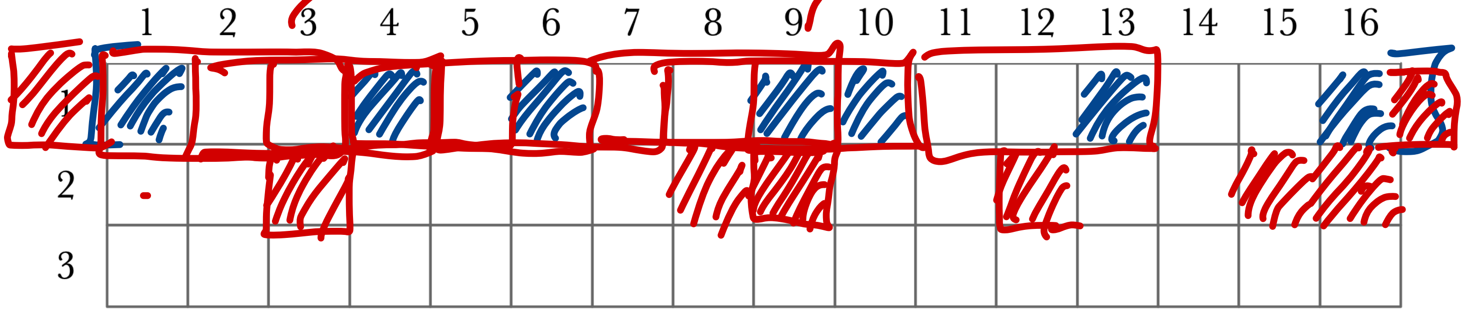
Draw rules: 0 0 0 0 1 0 1 1

0 0 0 0 1 0 1 1



Simulate execution:

init config



integer

floating pt #
(fractional value)

JavaScript Typing

Numerical values are `Number`s

- does not distinguish floating point vs integer formats

Example: `4` `==` `4.0`?

So `4 / 2` is the same is `4.0 / 2`

What about `5 / 2`?

Java → 2

Javascript 2.5

JavaScript Typing

Numerical values are `Numbers`

- does not distinguish floating point vs integer formats

Example: `4 == 4.0`?

So `4 / 2` is the same is `4.0 / 2`

What about `5 / 2`?

- to do integer division, use `Math.floor()`
- E.g., `Math.floor(5 / 2)` gives 2

· arr[size/2] ~ arr[Math.floor(size/2)]

T = text

So Far

HTML + CSS + JavaScript

- HTML for document structure, content, semantics
- CSS for styling based on semantics
- JavaScript to generate/interact with elements

This is all good for *text-based* documents

- simple graphics for including images, drawing boxes

Next Steps

Course Goal: Visualizations of algorithmic processes

1. Graphics
   - want to depict things other than static images and interactive boxes
   - **Tool:** Scalable Vector Graphics (SVG)
   - **Another Tool:** Canvas API
2. Objects
   - algorithms/processes have intermediate states that we want to visualize
   - **Tool:** JavaScript Objects

Connection: visualize the states of objects as computation progresses

Scalable Vector Graphics (SVG)

What is SVG?

Scalable **V**ector **G**raphics

- format for representing graphical objects
- *vector* graphics: image defines instructions for how to draw
  - not just pixels (e.g., `png`, `jpg`, `tiff`)
- specify shapes, shapes
- XML-based—structured like HTML:
  - elements and attributes
  - can be styled with CSS
  - can be manipulated with JavaScript
- standalone file `.svg` or embedded in HTML

extensible
markdown
language

tag

pixels

Structure of SVG

Create an SVG element with <svg> tag:

```
<svg width="600" height="400" xmlns="http://www.w3.org/2000/svg">
    ...
</svg>
```

attributes

Must specify `width`, `height`, and `xmlns`

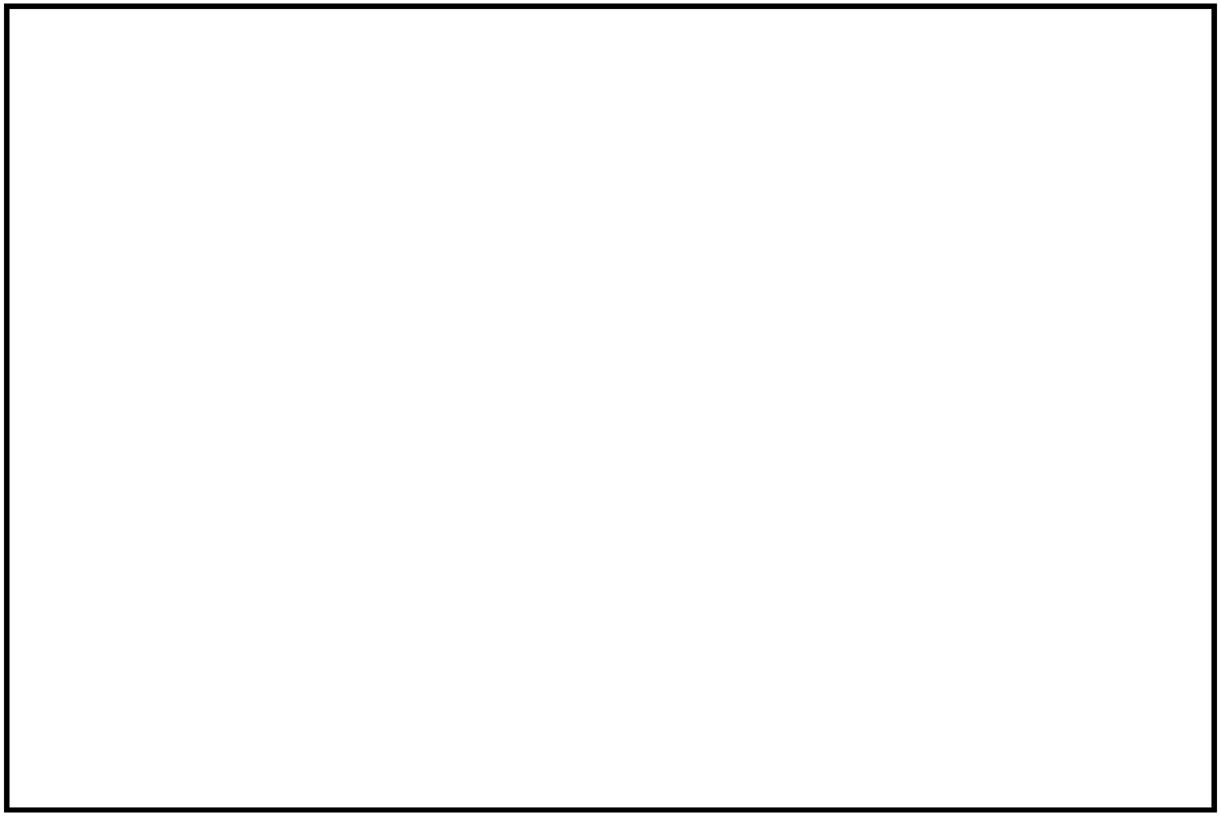
- `xmlns` is “xml namespace”, used to avoid naming conflicts with other types of XML (e.g., HTML)
- don’t worry about this

closing tags

What Can SVG Do?

Rectangles

tag

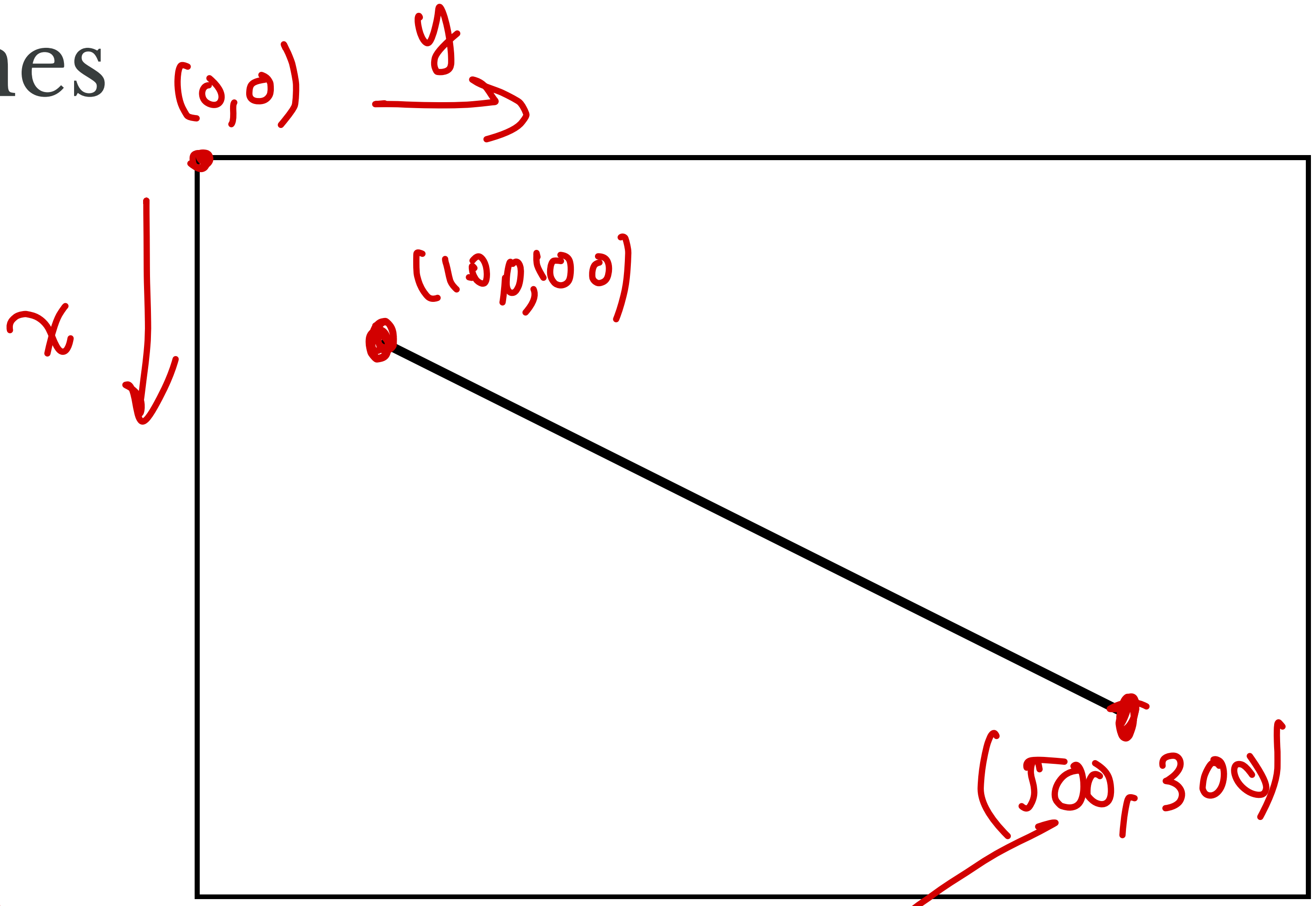


```
<rect width="100%" height="100%"
fill="white" stroke="black" stroke-width="5"/>
```

background

boarder color

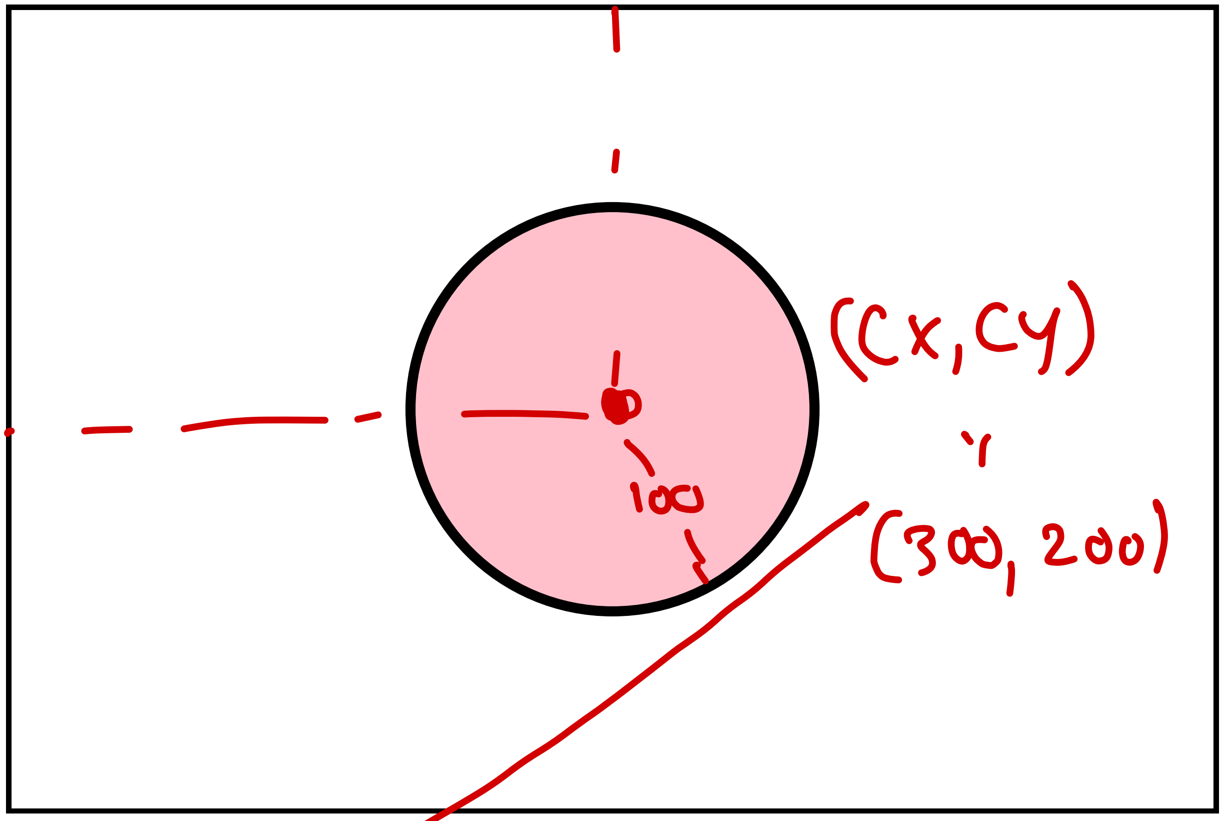
Lines



tag

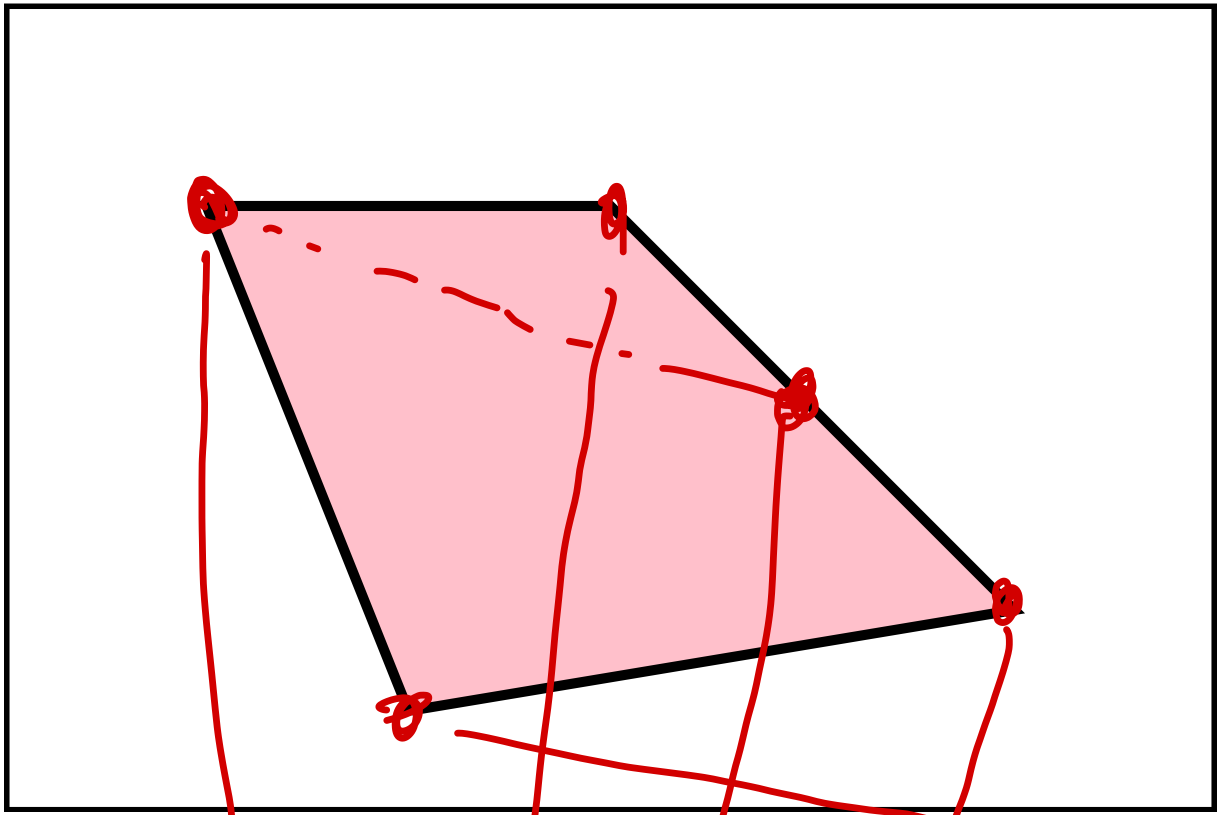
```
<line x1="100" y1="100" x2="500" y2="300"
stroke="black" stroke-width="5"/>
```

Circles



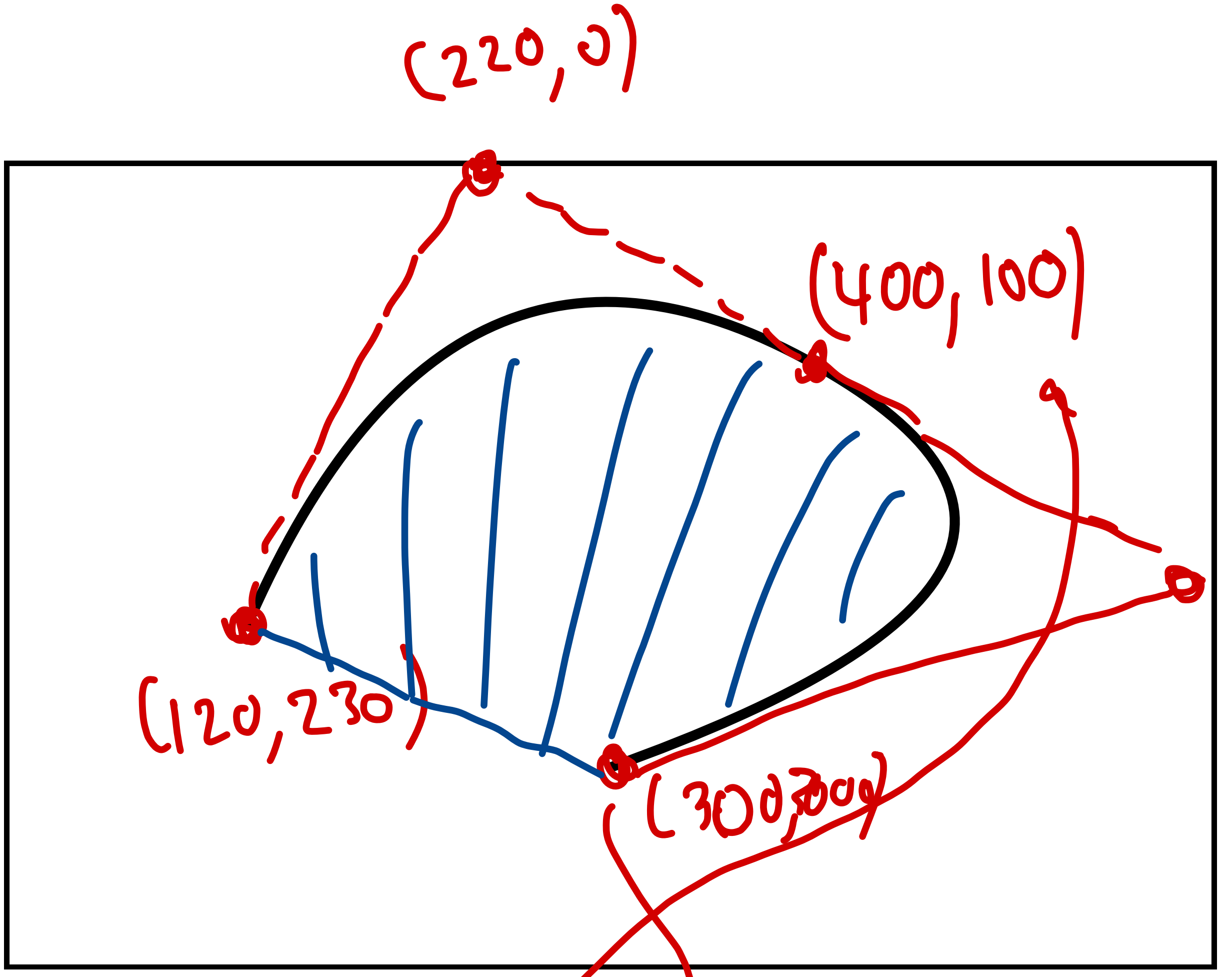
```
<circle cx="300" cy="200" r="100" stroke="black"
fill="pink" stroke-width="5"/>
```

Polygons



```
<polygon points="100 100 300 100 400 200 500 300 200 350"
stroke="black" stroke-width="5" fill="pink"/>
```

Paths

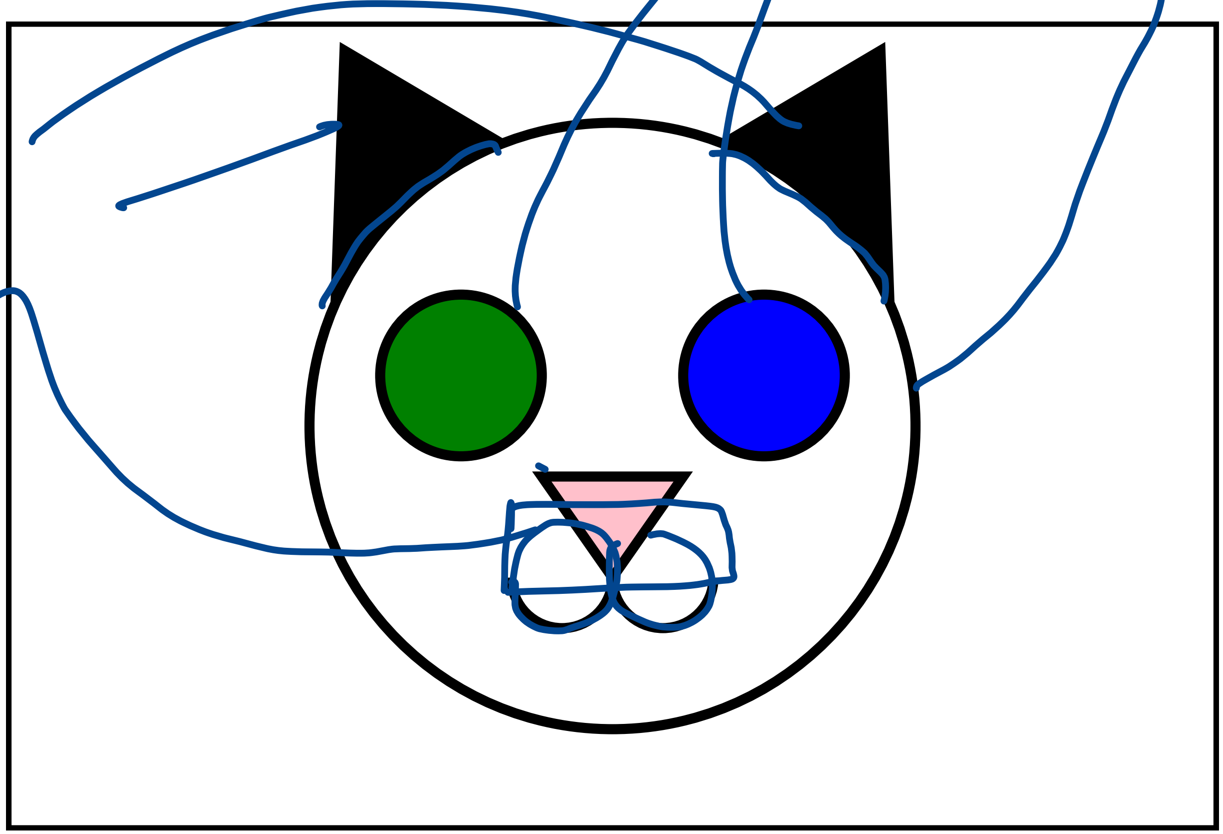


```
<path d="M120,230 Q220,0 400,100 T300,300"
stroke="black" stroke-width="5" fill="transparent"/>
```

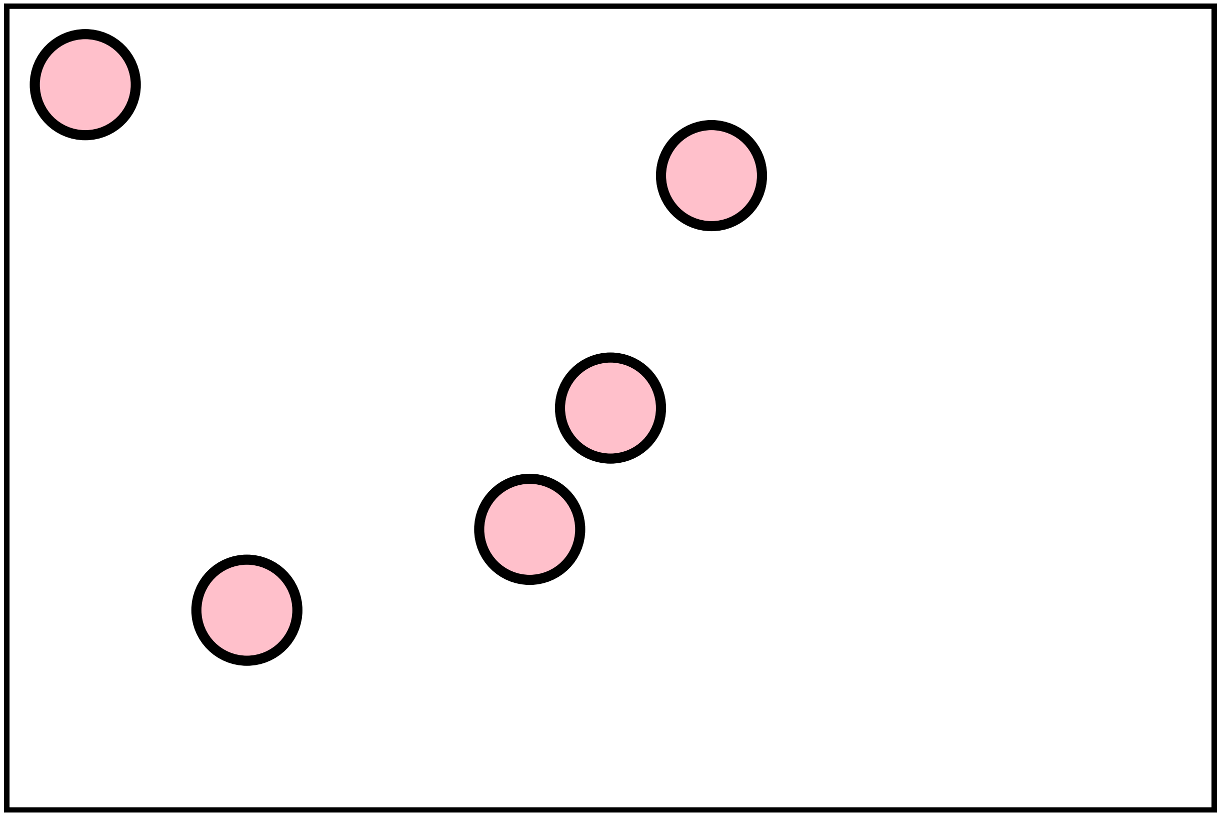
Quadric Bezier Curve

Activity

Draw This (or something better)

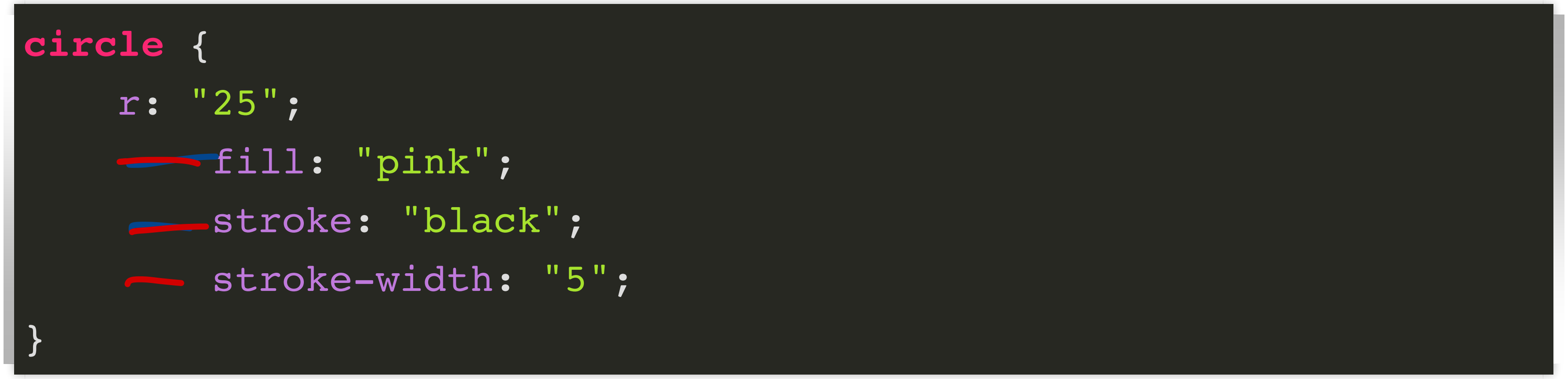


Dealing with Repetition



All `circles` have same `radius`, `stroke`, `stroke-width`, `fill`

SVG elements can be styled using CSS!



```
circle {
    r: "25";
    fill: "pink";
    stroke: "black";
    stroke-width: "5";
}
```

Now must only specify the location of each circle!

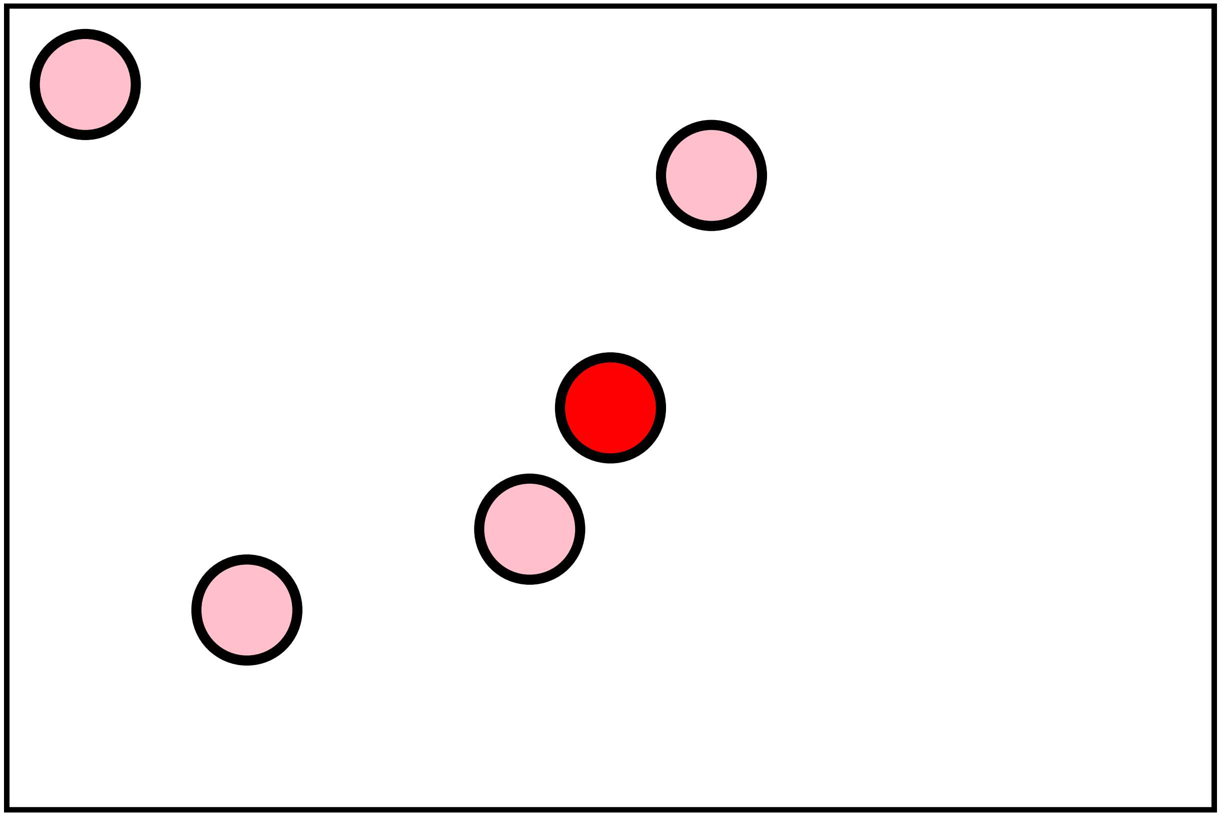
Styling by class, id

SVG elements can be given `class` and `id` just like HTML elements!

```
<circle cx="300" cy="200" class="dot" id="special-dot"/>
<circle cx="120" cy="300" class="dot"/>
<circle cx="40" cy="40" class="dot"/>
<circle cx="260" cy="260" class="dot"/>
<circle cx="350" cy="85" class="dot"/>
```

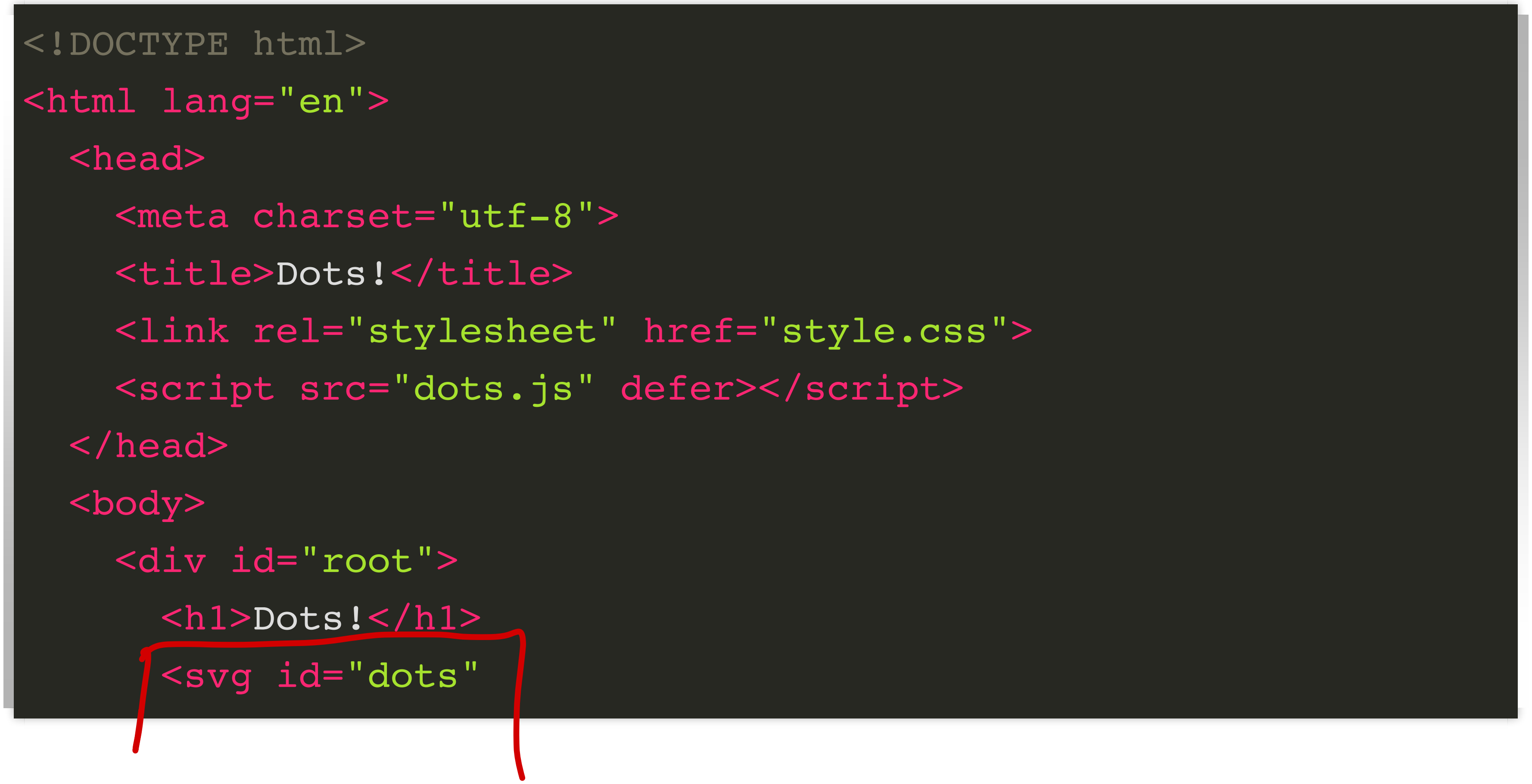
```
.dot {
    r: 25;
        fill: pink;
        stroke: black;
        stroke-width: 5;
}
#special-dot {
    fill: red;
}
```

Result



SVG + JavaScript

SVG can be accessed and modified with JavaScript!



```html
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="utf-8">
    <title>Dots!</title>
    <link rel="stylesheet" href="style.css">
    <script src="dots.js" defer></script>
  </head>
  <body>
    <div id="root">
      <h1>Dots!</h1>
      <svg id="dots"
```

Creating Elements

HTML contains:

```
<svg id="dots"
     width="600" height="400"
     xmlns="http://www.w3.org/2000/svg">
  <rect id="dot-background" width="100%" height="100%"/>
</svg>
```

Slightly different that html elements

- must include namespace

```
const ns = 'http://www.w3.org/2000/svg';
const svg = document.querySelector('#dots');

let circle = document.createElementNS(ns, 'circle');
svg.appendChild(circle);
```

type

Modifying Elements

Again, different from HTML

attribute name

value to assign to attribute

```
let circle = document.createElementNS(ns, 'circle');
circle.setAttributeNS(null, 'cx', this.cx);
circle.setAttributeNS(null, 'cy', this.cy);
circle.setAttributeNS(null, 'class', 'dot');
svg.appendChild(circle);
```

```
.dot {
    r: 10px;
    fill: rgb(50, 120, 255);
    stroke: black;
    stroke-width: 2;
}
```

Objects in JavaScript

What are Objects?

Collection of

- attributes and associated values
- methods

Example `dot` class

- attributes:
  - `cx` x position of center
  - `cy` y position of center
- methods:
  - `updateLocation(cx, cy)` moves dot to a new location

Object Constructors

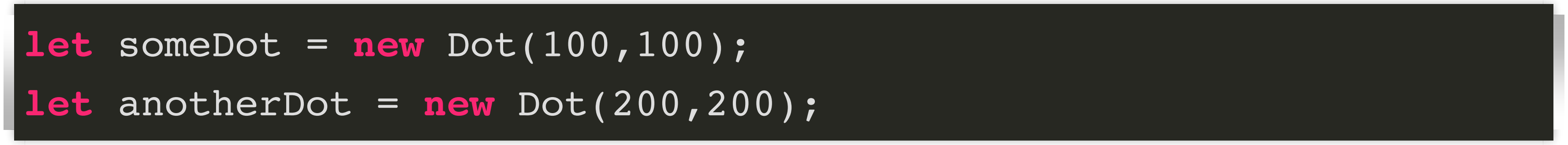
In JS, object types can be defined by defining a **constructor**

- function that creates the object
- keyword `this` defines attributes and methods

By convention, constructor names are Capitalized:

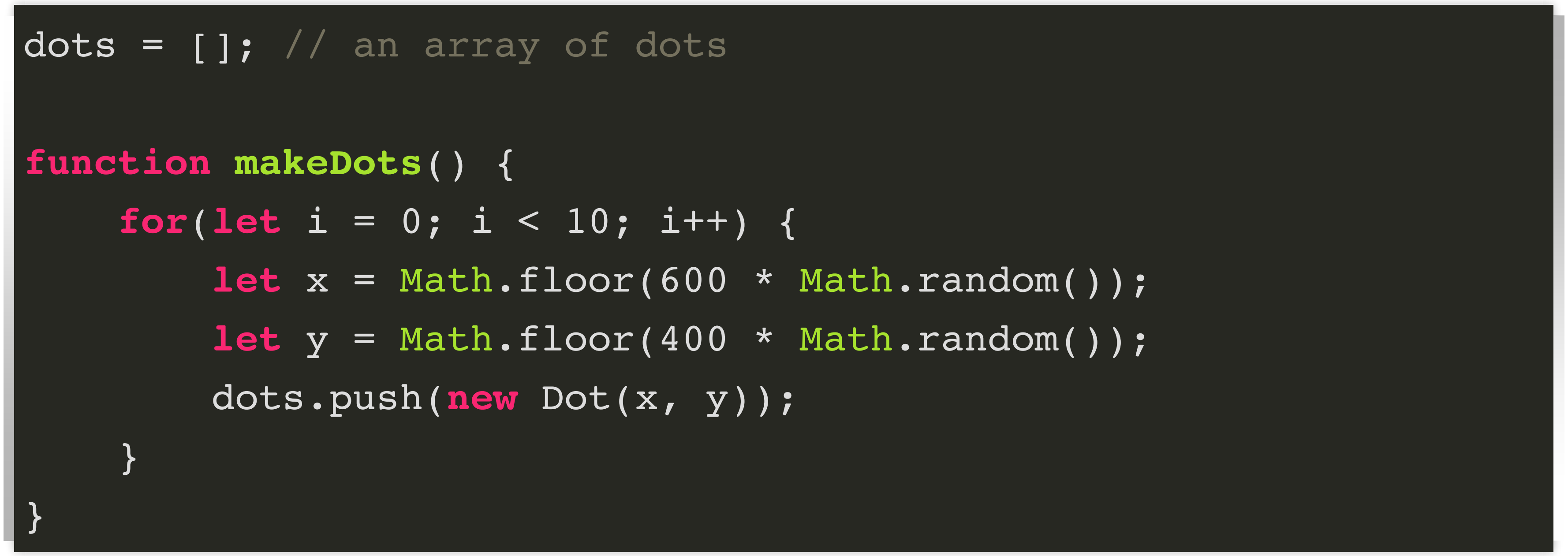
```
function Dot(cx, cy) {
    this.cx = cx;
    this.cy = cy;
    this.circle = document.createElementNS(ns, 'circle');
    this.circle.setAttributeNS(null, 'cx', this.cx);
    this.circle.setAttributeNS(null, 'cy', this.cy);
    this.circle.setAttributeNS(null, 'class', 'dot');
    svg.appendChild(this.circle);
}
```

To make one dot



```
let someDot = new Dot(100,100);
let anotherDot = new Dot(200,200);
```

Now to make some dots...



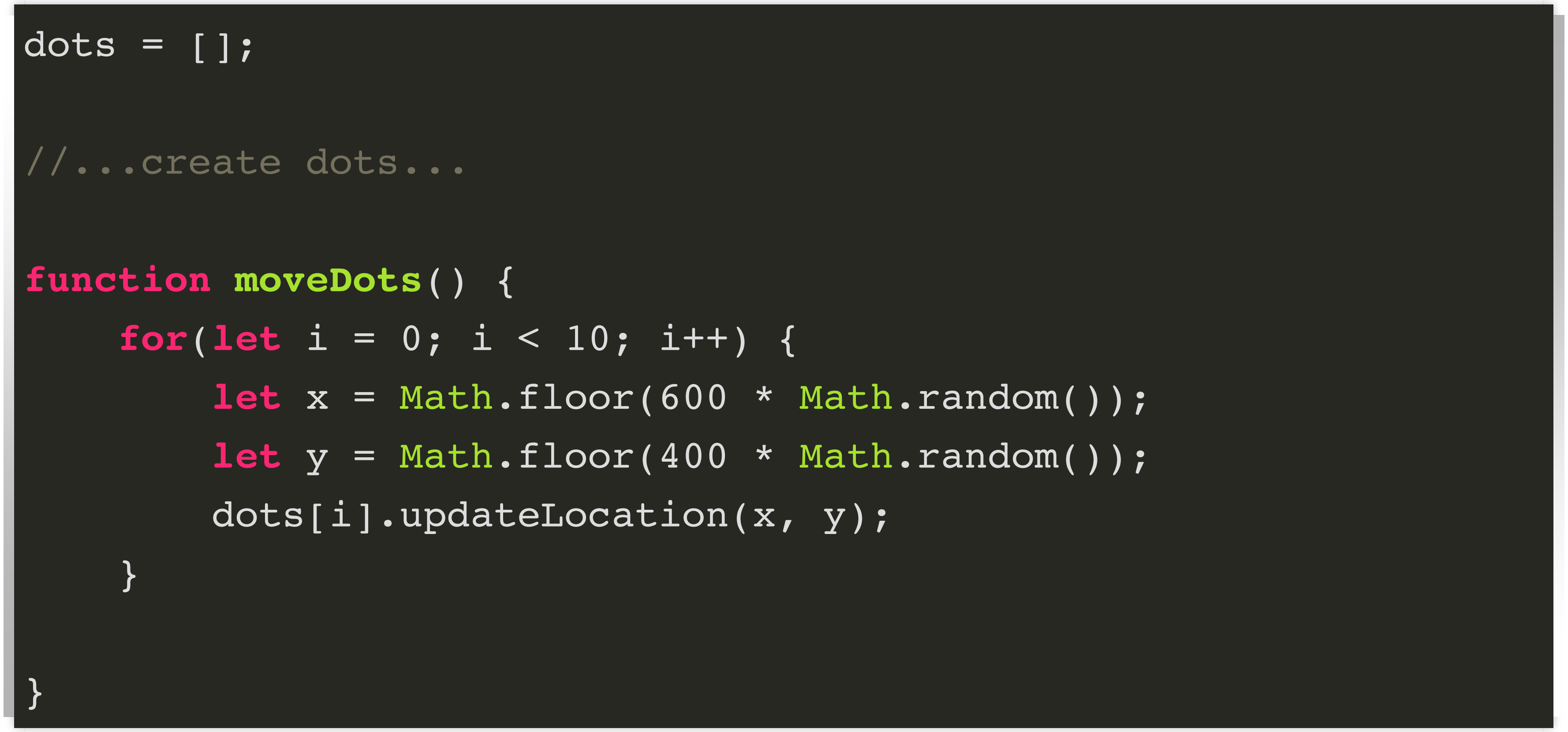
```
dots = []; // an array of dots

function makeDots() {
    for(let i = 0; i < 10; i++) {
        let x = Math.floor(600 * Math.random());
        let y = Math.floor(400 * Math.random());
        dots.push(new Dot(x, y));
    }
}
```

Defining Methods

You can include method definitions in the constructor as well!

```
function Dot(cx, cy) {
    ...
    this.updateLocation = function (cx, cy) {
        this.cx = cx;
        this.cy = cy;
        this.circle.setAttributeNS(null, 'cx', this.cx);
        this.circle.setAttributeNS(null, 'cy', this.cy);
    };
}
```

Now we can move dots around



```
dots = [];

//...create dots...

function moveDots() {
    for(let i = 0; i < 10; i++) {
        let x = Math.floor(600 * Math.random());
        let y = Math.floor(400 * Math.random());
        dots[i].updateLocation(x, y);
    }

}
```

Dots Demo

Next Time

1. Representing more interesting data types
2. Visualizing algorithms