Week 4

Drawing with SVG

+ INTRO TO D3.JS

Review of JavaScript Basics

WHAT IS JAVASCRIPT FOR?

JavaScript

“Behavior”

All the dynamic stuff, such as animation, user interaction, manipulating DOM elements...

HTML

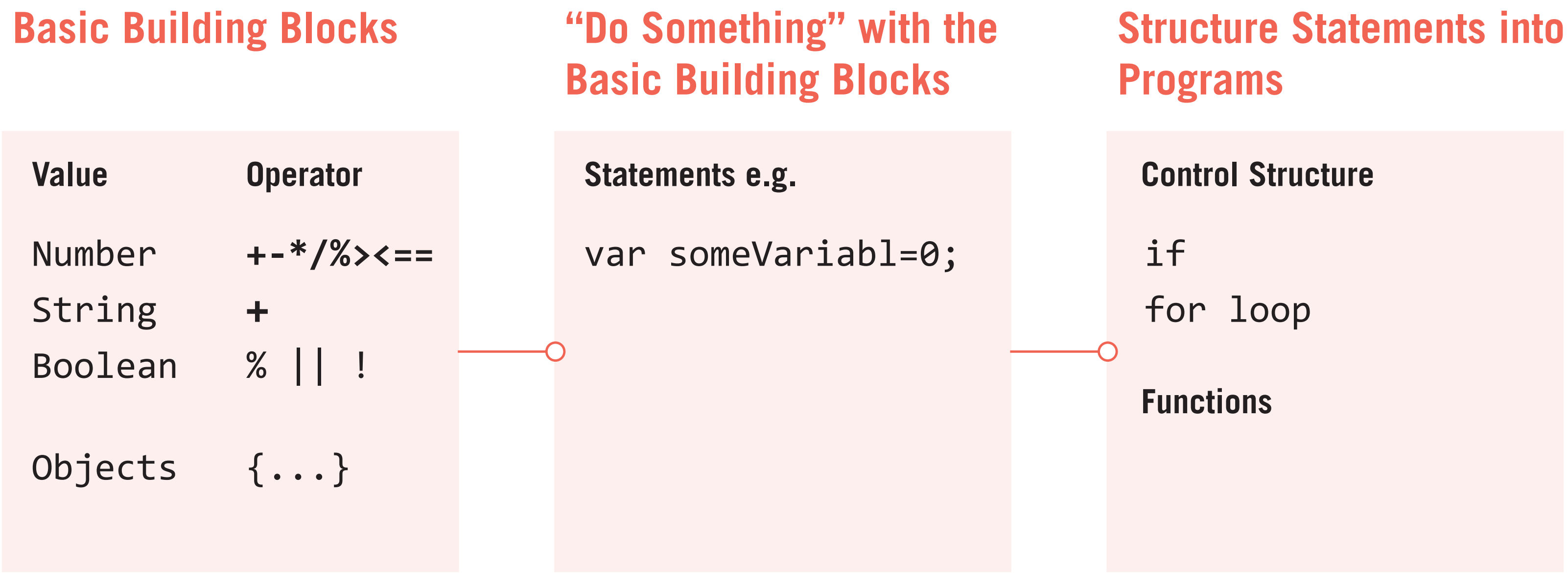
“Content”

CSS

“Style”

Controls the appearance of HTML DOM elements

KEY JAVASCRIPT CONCEPTS



Basic Building Blocks

Value	Operator
`Number`	`+-*/%><==`
`String`	`+`
`Boolean`	`% || !`
`Objects`	`{...}`

“Do Something” with the Basic Building Blocks

Statements e.g.

```
var someVariabl=0;
```

Structure Statements into Programs

Control Structure

```
if
for loop
```

Functions

IF...STATEMENT

If a boolean condition is true, then do something; if not, do something else

```
if( [some boolean expression] ){
   //...do this if boolean expression equals
true
}else{
   //...do this if boolean expression equals
false
}
```

FOR...STATEMENT

1. Create an initial conditions
2. Create a boundary condition (boolean) to stop the loop
3. Update the state the loop at each iteration, checking against the boundary condition; stop once the boundary condition is reached

“tracking vari- 3

```
for(var i=0; i<1000; i++){
    console.log(i);
}
```

Note the space

FUNCTIONS

Functions help to define blocks of sub-program that 1) functionally relate to each other and/or 2) can be re-used.

This is how you create a function

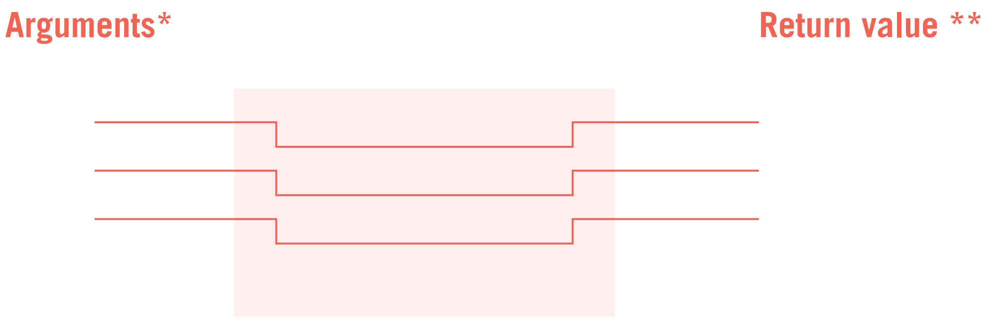
```
function doSomething(){...}

var doSomething = function(){}
```

This is how you then run this function

```
doSomething();
```

FUNCTIONS: PARAMETERS AND RETURN VALUES



```
function doSomething (argument 1, argument 2...){
    //do something
    //do something else
    //...
    //return return value;

}
```

Representing Data Structures: Objects and Arrays

Object

```
var newCar = {

    //these are properties
    make: “Subaru”,
    year: 2009,
    color: “Silver”,

    //these are methods
    start: function(){
        console.log(“Vroom”);
    }
}
```

“Property” and “Method”

Almost all JavaScript entities have them.

Properties are values:

```
newCar.make  // “Subaru”
```

Methods are functions:

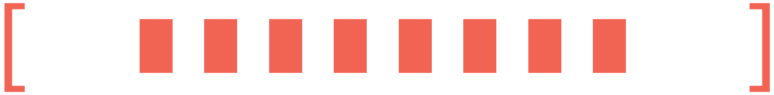
```
newCar.start(); // “Vroom”
```

INTRODUCING ARRAYS

Arrays are a JavaScript object that represents a parallel list of values or variables.

```
var students = [‘Anna’, ‘Brian’, ‘Christina’,
‘Dean’];
```

1. Arrays, like functions and any JavaScript object, can be assigned to a variable;
2. Arrays are enclosed by [ ];



ARRAY INDEX

Arrays, like other JavaScript objects, have properties. One key property is `.length`

```
>> var students = [‘Jessie’, ‘Audrey’,
‘Patrick’, ‘Andrew’];
>> console.log(students.length); //4
```

Individual elements of an array can be access using an index, starting from `0` and ending at `.length-1,` with `array[index]`

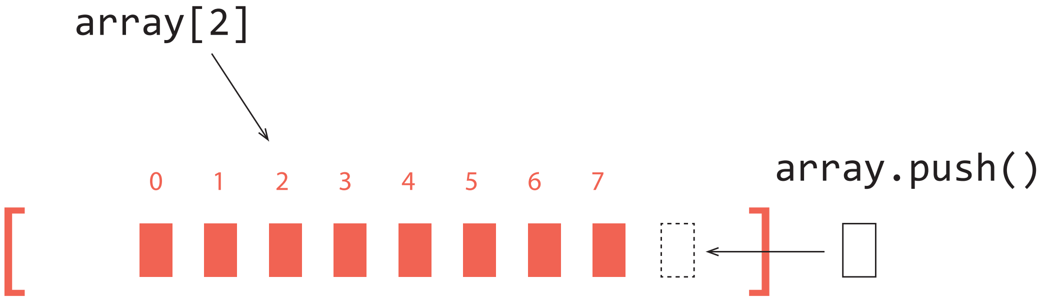
```
>> var students = [‘Jessie’, ‘Audrey’,
‘Patrick’, ‘Andrew’];
>> console.log(students[0]); // ‘Jessie’
>> console.log(students[3]); // ‘Andrew’
```

ARRAY METHODS

Arrays, like other JavaScript objects, have methods. One key property is `.push()`, which adds a value to an array at the end

```
>> var students = [‘Jessie’, ‘Audrey’,
‘Patrick’, ‘Andrew’];
>> students.push(‘Nina’);
>> console.log(students[4]); // ‘Nina’
```

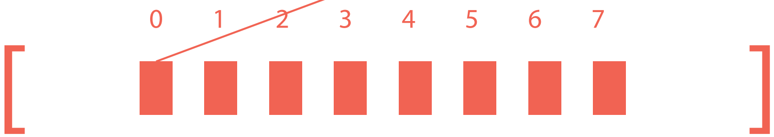
ARRAY INDEX



ARRAY METHODS

What are some other useful methods of array?
https://developer.mozilla.org/en-US/docs/Web/JavaScript/

```
array.forEach(function(element){
   //do something with each element
});
```



MORE ON ARRAYS

Values in the array don’t just have to be numbers, strings or booleans. They can be any JavaScript object:

```
var student1 = {
   program: “MFA”,
   name: “Skye”
};
var student2 = {
   program: “Architecture”,
   name: “Matthew”
}
var students = [];
students.push(student1);
students.push(student2);
```

Arrays represent a data structure--a collection of values.

Any value in an array can be accessed with an index, using the array[index] notation.

Arrays can be easily modified, using methods such as .push()

Become Familiar with Arrays

Open up Exercise 1 and let’s work through arrays.

Intro to D3

D3 IS A “LIBRARY”

d3.js is a JavaScript **library**.

A JavaScript library contains **pre-written functions** and **objects** that you can use off the shelf.

To begin, let’s use D3 to manipulate the DOM.

D3 IS A “LIBRARY”

d3.js is a JavaScript **library**.

A JavaScript library contains **pre-written functions** and **objects** that you can use off the shelf.

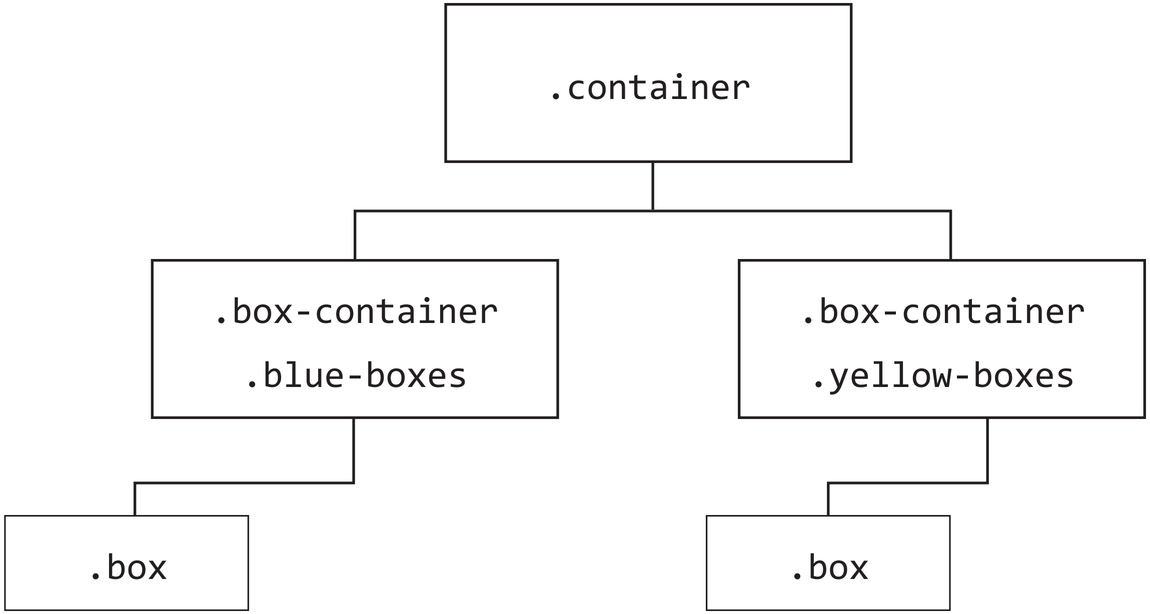
To begin, let’s use D3 to manipulate the DOM.

DRAWING BOXES

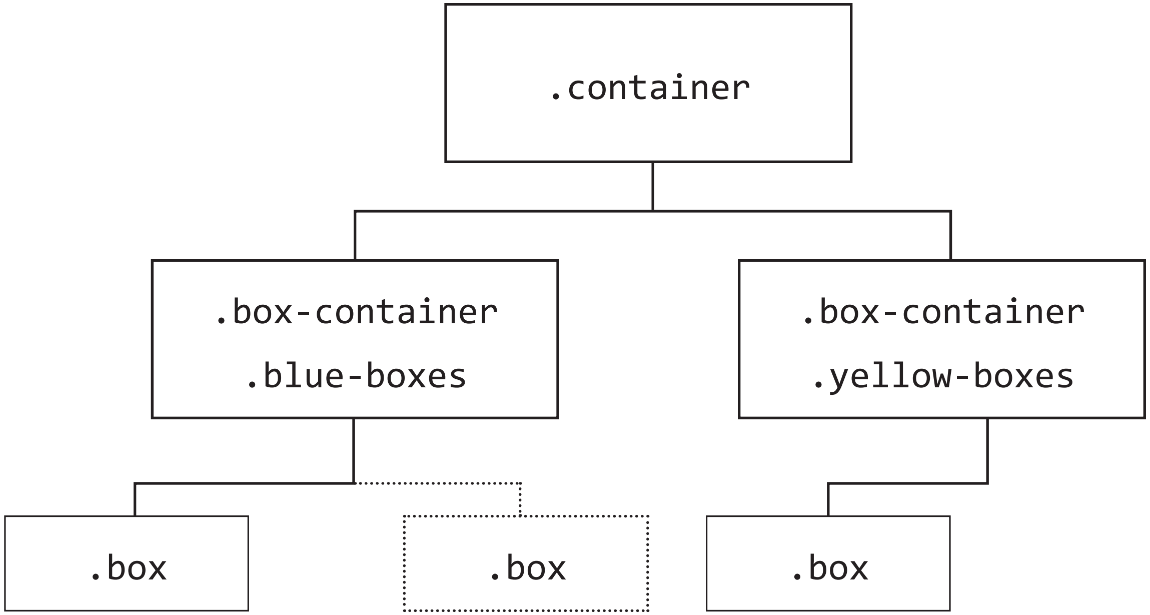
Open Exercise 2, and take a look at “script/script.js”. What does the DOM tree look like?

Also take a look at “style.css” and examine the styles being applied to each DOM element.

DRAWING BOXES



DRAWING BOXES



DIPPING INTO D3

Our first block of d3 code ever

```
d3.select(“.blue-boxes”)
    .append(“div”)
    .attr(“class”, “box”);
```

D3 SELECTION

`d3.select( )`

Using `d3.select()` turns one DOM element on the page into a selection:

```
d3.select(“.blue-boxes”)
```

`selection.append( ) / selection.attr( ) / selection.style( )`

then, you use D3 methods to manipulate this selection:

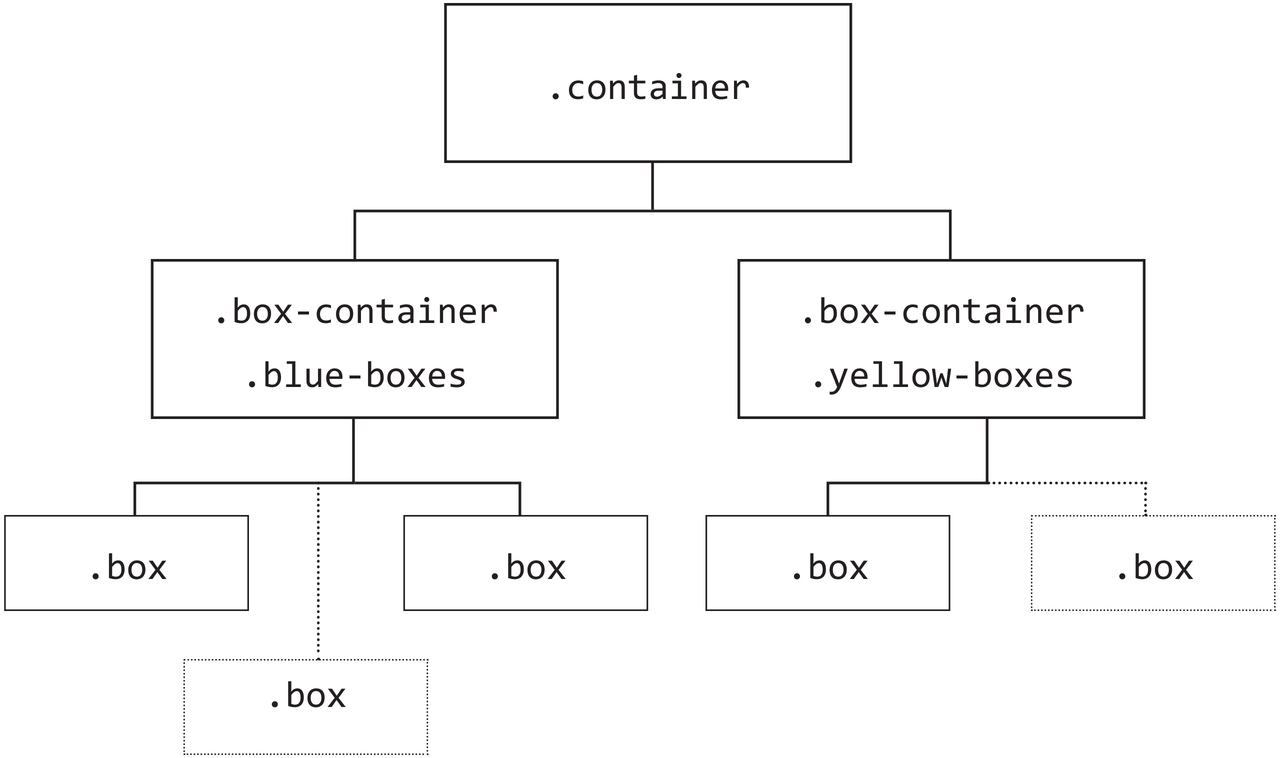
```
d3.select(“.blue-boxes”)
    .append(“div”)
    .attr(“class”, “box”)
```

D3 SELECTION

```
d3.select(“.container”)
    .append(“div”)
    .attr(“class”, “box”)
    .style(“width”, “100px”);
```

- Select one element with class name “blue-boxes”
- Append a new <div> element under it
- Set the “class” attribute of the new <div> to “box”
- Add inline CSS style for the new <div>

DRAWING BOXES

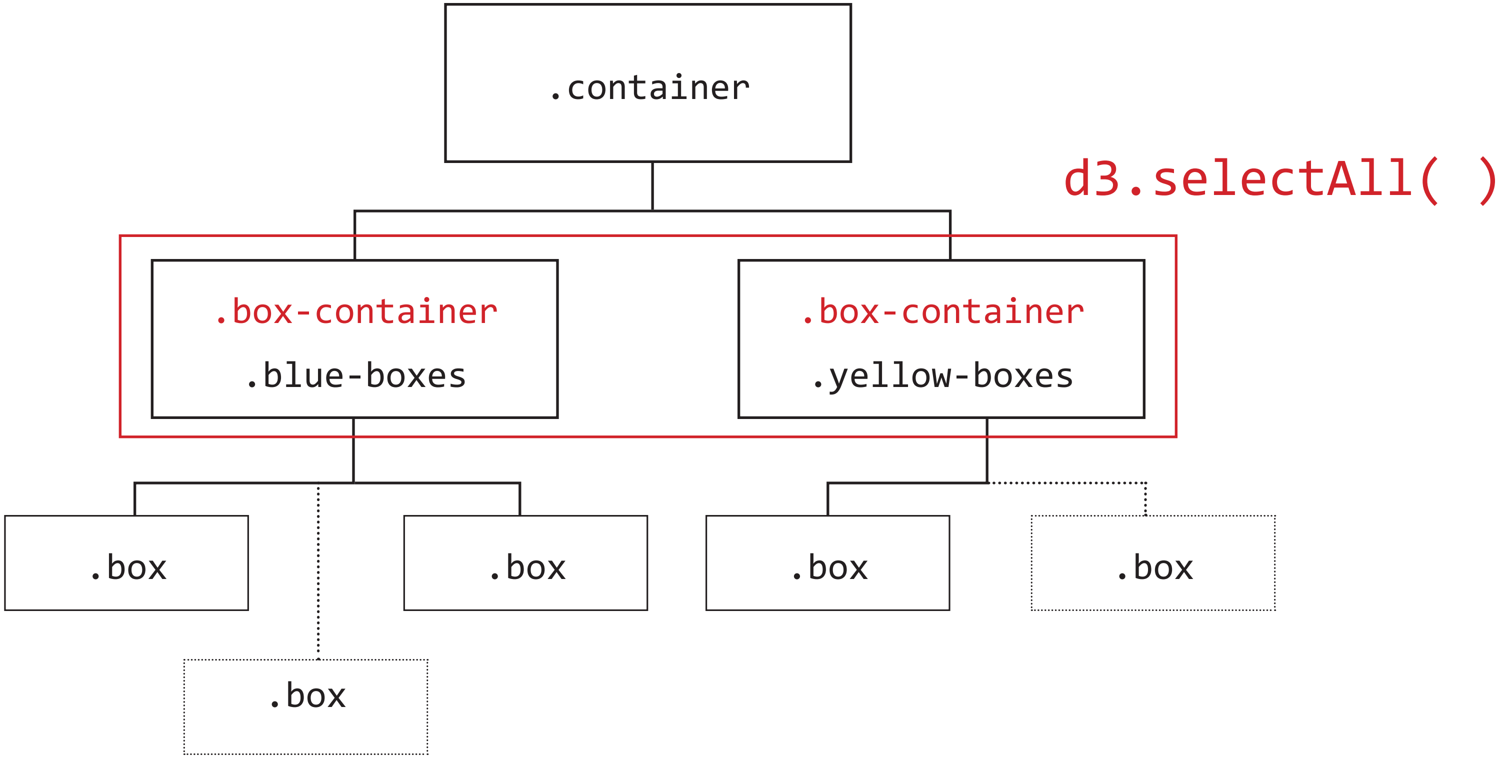


d3.selectAll( )

```
d3.selectAll(“.box-container”)
    .append(“div”)
    .attr(“class”, “box”);
```

- `d3.select( )` creates a selection of a single DOM element
- `d3.selectAll( )` creates a selection of multiple DOM elements

DRAWING BOXES



d3.selectAll( )

BOX WITHIN A BOX

```
d3.select(“.yellow-boxes”)
    .append(“div”)
    .attr(“class”, “box”)
    .append(“div”)
    .attr(“class”, “inner”)
    .style(“width”, “50%”)
    .style(“background”, “red”);
```

Select the <div> element with class “yellow-boxes”

Append an <div> element

Set the attributes on <div>

Append a <div> element under <div.box>

Set the attributes on <div>

BOX WITHIN A BOX

```
d3.select(“.yellow-boxes”)
    .append(“div”)
    .attr(“class”, “box”)
    .append(“div”)
    .attr(“class”, “inner”)
    .style(“width”, “50%”)
    .style(“background”, “red”);
```

One more thing: how come we can keep “chaining” method calls one after another?

CHAINING

```
d3.select(“.yellow-boxes”)
    .append(“div”)
    .attr(“class”, “box”)
    .append(“div”)
    .attr(“class”, “inner”)
    .style(“width”, “50%”)
    .style(“background”, “red”);
```

One more thing: how come we can keep “chaining” method calls one after another?

- Each `.attr()` call returns the old selection, for you to call a new method onto it;
- Each `.append()` call returns the newly appended elements as the new selection, for you to call a new method onto it.

CHAINING

How is this different from the previous example?

```
var container = d3.select(“.yellow-boxes”);
container
    .append(“div”)
    .attr(“class”, “box”);
container
    .append(“div”)
    .attr(“class”, “inner”)
    .style(“width”, “50%”)
    .style(“background”, “red”);
```

DRAWING WITH SVG

Drawing with SVG is just DOM manipulation (selecting, appending, removing).

Open 4-3-Complete and inspect its DOM tree. What do you see?

DRAWING WITH SVG

Drawing with SVG is just DOM manipulation (selecting, appending, removing).

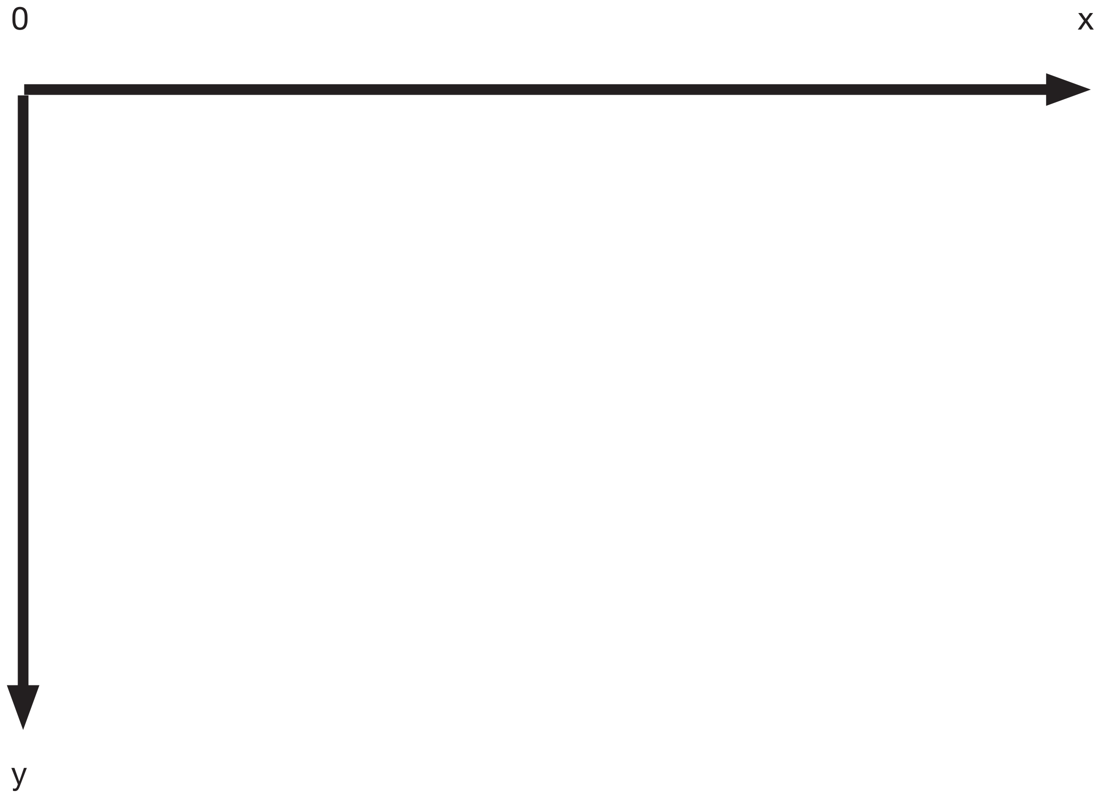
Open 4-3-Complete and inspect its DOM tree. What do you see?

COMMON SVG ELEMENTS

	<circle>	<line>	<rect>	<text>	<path>	<g>
attr	cx cy r	x1 y1 x2 y2	x y width height	x y text	d	
	transform class					
style	fill fill-opacity stroke stroke-width stroke-opacity					

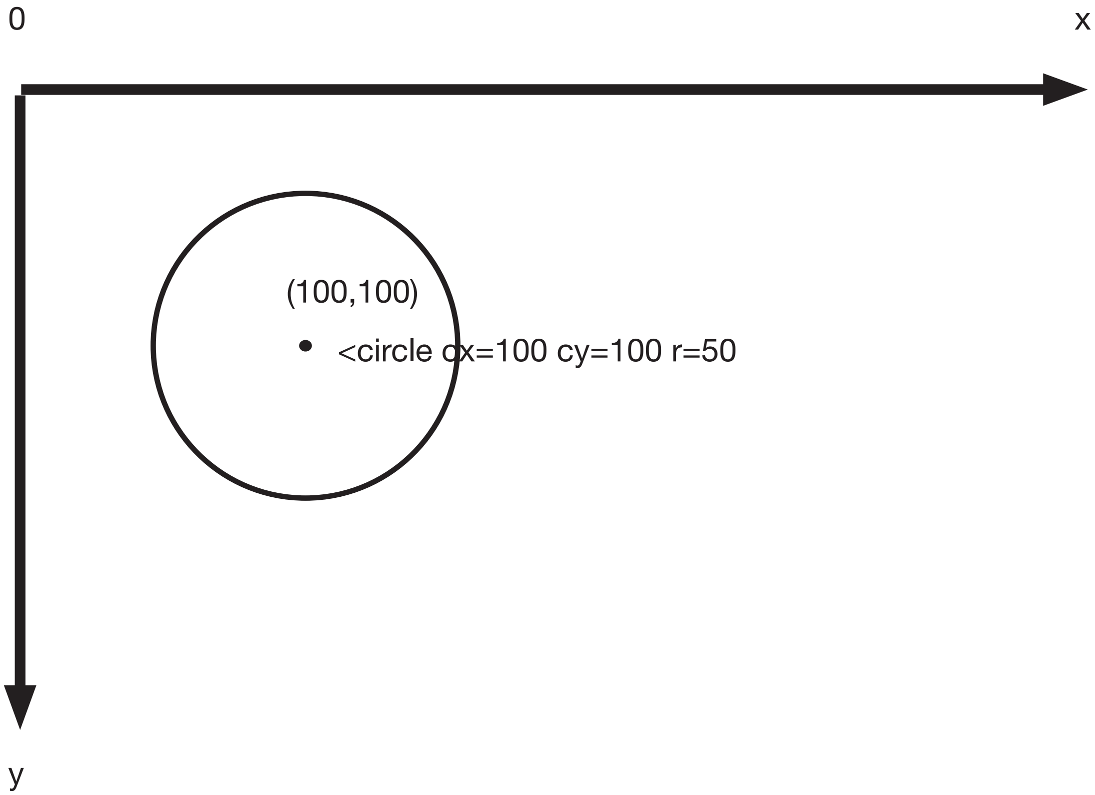
COORDINATES IN SVG

The grid system in <svg> works left to right, top to bottom



COORDINATES IN SVG

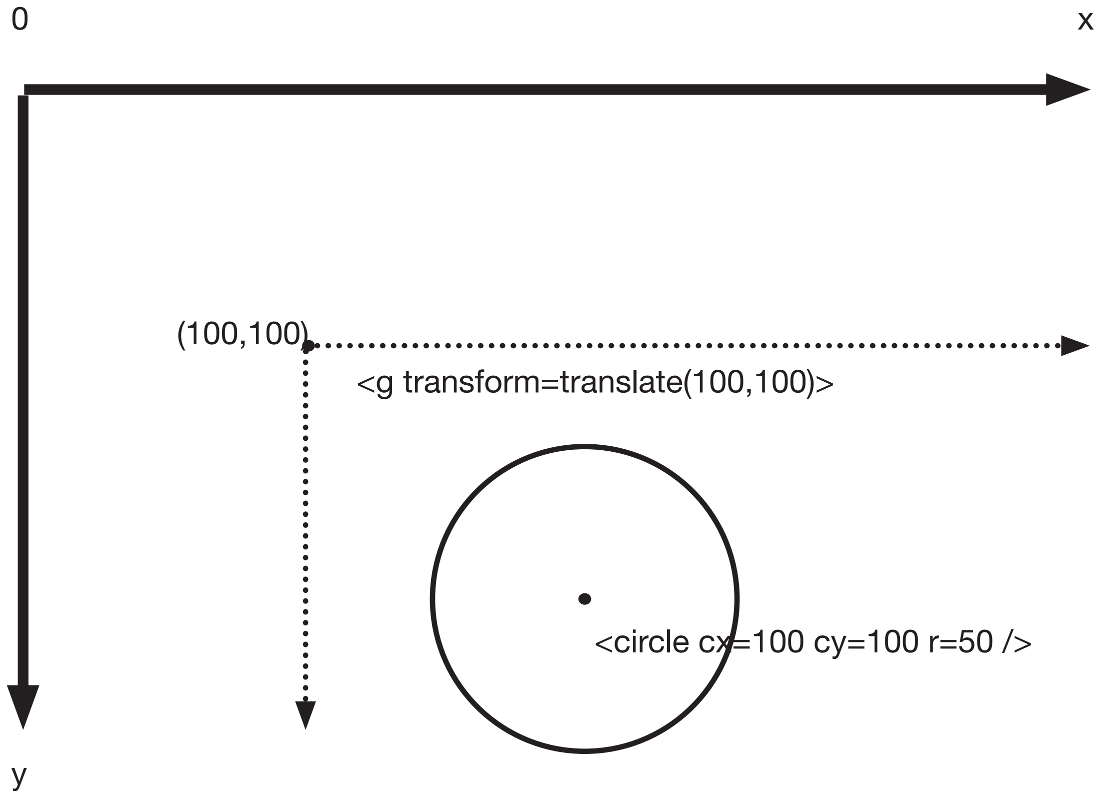
```
<svg>
     <circle ... />
</svg>
```



COORDINATES IN SVG

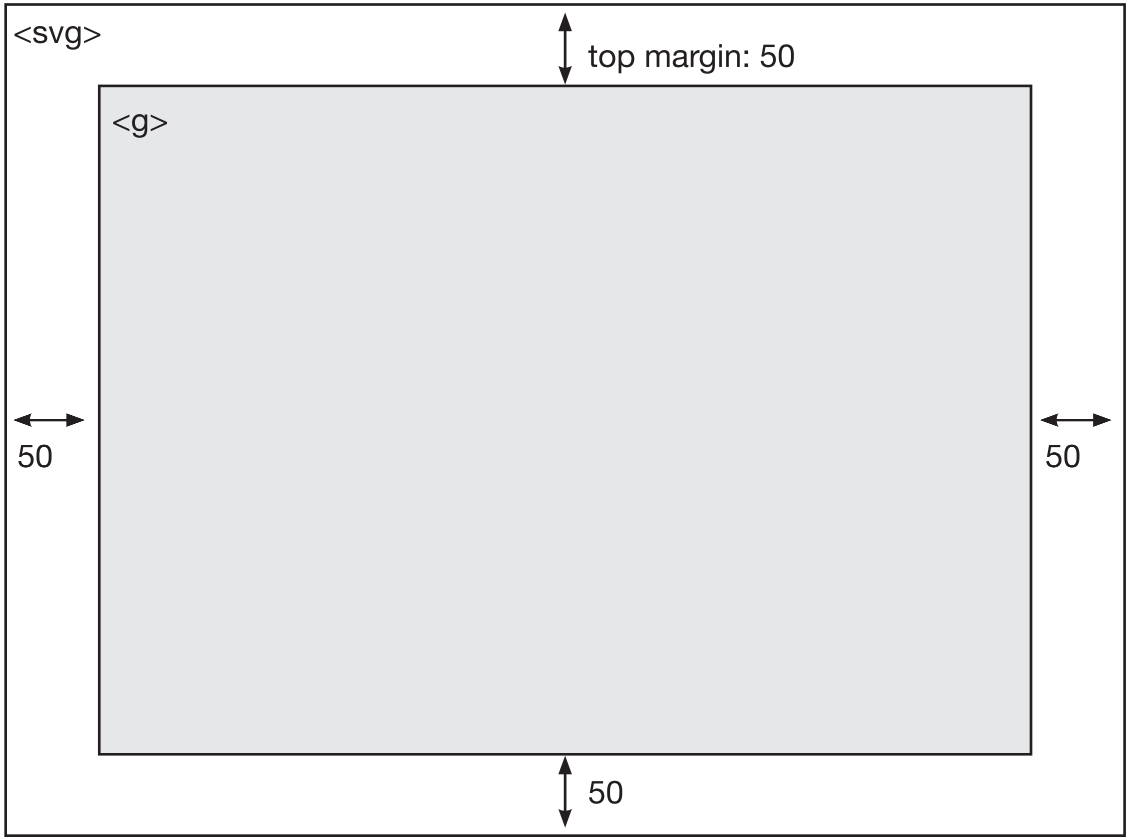
We use <g> to group individual elements; each <g> starts its own coordinate system.

In this example, we "translated" <g> by (100,100), so that the <circle> element is actually at (200,200) relative to the overall <svg>



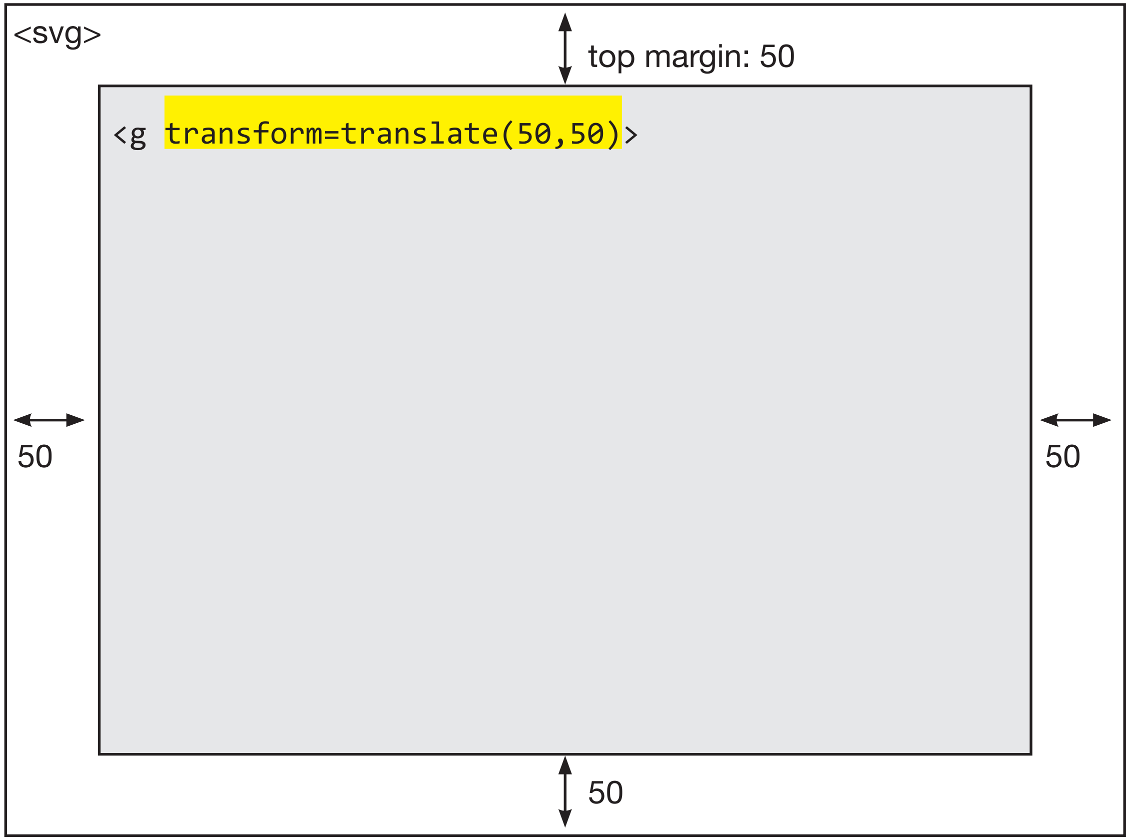
MARGIN CONVENTIONS

We often find it useful NOT to draw from the very edge of <svg>. Instead, we use a <g> to offset everything by a margin, so that we leave some margin between the drawing and the edges.



MARGIN CONVENTIONS

We often find it useful NOT to draw from the very edge of <svg>. Instead, we use a <g> to offset everything by a margin, so that we leave some margin between the drawing and the edges.



DRAWING WITH SVG

Continue with Exercise 3

PUTTING EVERYTHING TOGETHER

In Exercise 4, let’s visualize the workings of Math.random()

Before you start, sketch out what this might look like. What choices are you making?

RECAP

Last class, we studied the concept of Javascript objects and functions

A library is a collection of pre-built objects and functions.

D3 is a library that, among other things, can help us manipulate DOM elements

By manipulating <svg> DOM elements, we can visualize shapes on the screen.

RECAP

In the last exercise we encountered two typical considerations we tend to encounter in data visualization.

Visual encoding: what visual properties (position, shape, size, color) best express what we are trying to show.

Mapping domain to range: how do we effectively map numbers to screen coordinates?

A goal of this course is to help you develop better intuitions about how to address these considerations!