Week 4

DRAWING WITH SCRIPT: AN INTRO

+ 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

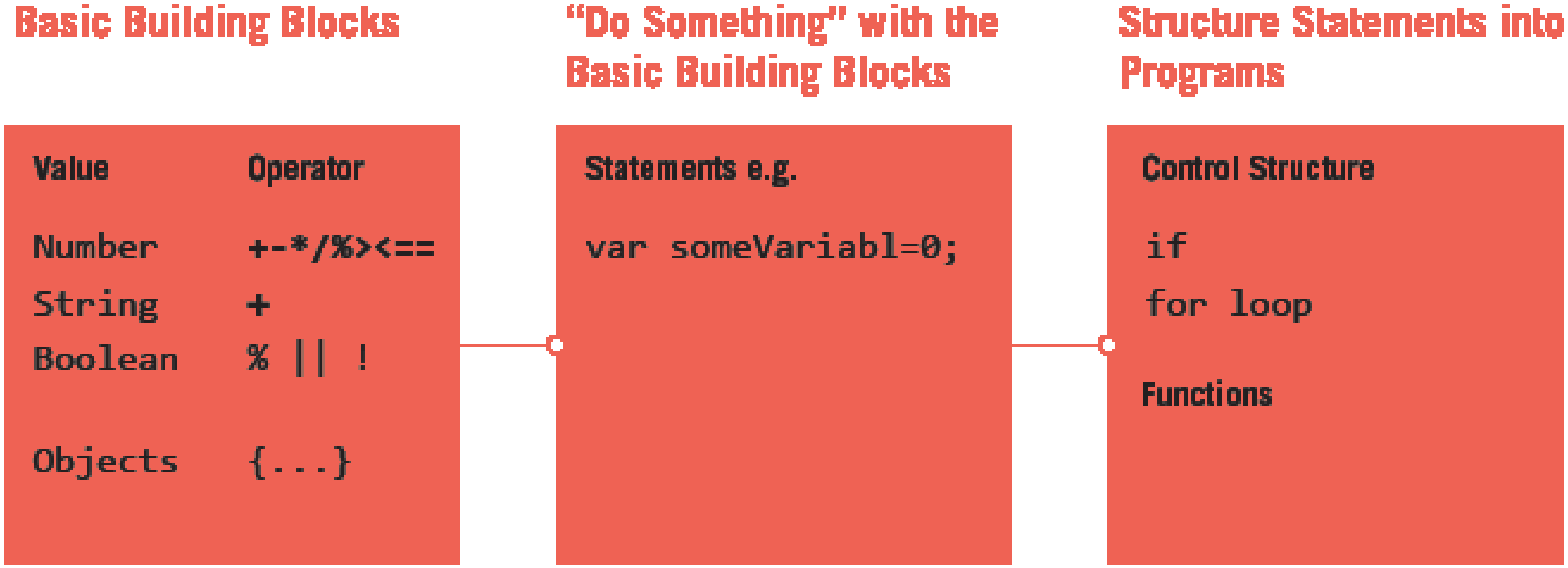
“Content”

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.

First, we can **define a function**:

```
var someFunc = function(){...};
```

which is exactly the same as:

```
function someFunc(){
    ...
}
```

Defining a function will NOT run the statements inside it.

FUNCTIONS

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

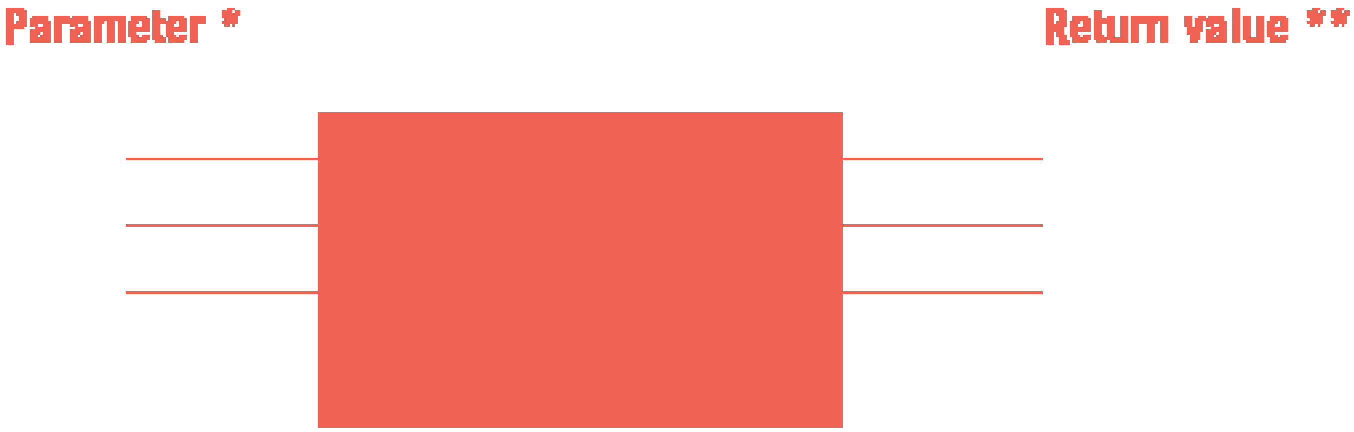
Two ways to create a function:

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

var doSomething = function(){}
```

```
doSomething(); //this will run someFunc
```

FUNCTIONS: PARAMETERS AND RETURN VALUES



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

}
```

FUNCTION SCOPE

```
var v3;

function func1 (parameter 1, parameter 2...){
    var v1;
}
function func2 (parameter3){
    var v2;
}
```

Representing Data Structures: Objects and Arrays

Objects and Arrays as Data Structures

Values (number, string, boolean) are inadequate for representing more complex data structures.

For example, what if I want to store a long list of numbers (like your student IDs)?

Or what if I want to group a number of related values into a single entity?

We've seen examples of a JavaScript **object**.

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 = [‘Jessie’, ‘Audrey’, ‘Patrick’,
‘Andrew’];
```

1. The above example is an array (“a collection”) of string values;
2. Arrays, like functions and any JavaScript object, can be assigned to a variable;
3. Arrays are enclosed by [];

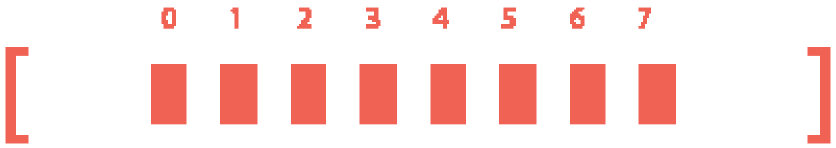
ARRAY INDEX

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]`

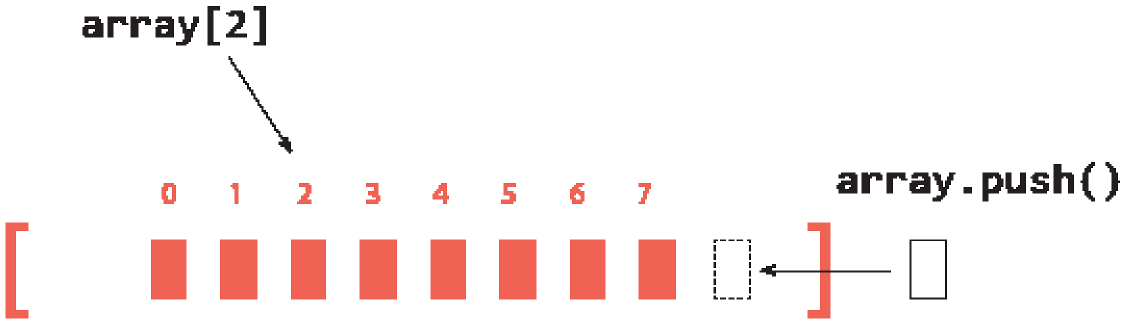
```
>> var students = [‘Jessie’, ‘Audrey’,
‘Patrick’, ‘Andrew’];
>> console.log(students[0]); // ‘Jessie’
>> console.log(students[3]); // ‘Andrew’
```


0
1
2
3
4
5
6
7

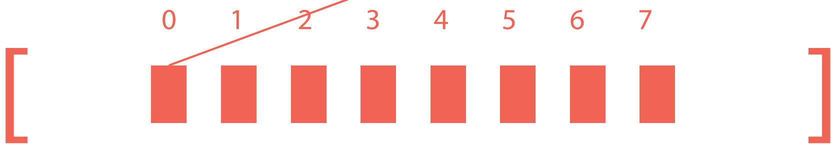
ARRAY METHODS

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[2]
0
1
2
3
4
5
6
7
array.push()

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



ARRAY METHODS

Knowing these properties and methods of arrays, and the for... loop, we can quickly generate a large array of values

```
var numbers = []; //empty array

for(var i = 0; i < 100; i++){
   numbers.push(Math.random());
}
```

The code above 1) creates an empty array and 2) adds 100 random numbers between 0 and 1

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

DIPPING INTO D3

Our first block of d3 code ever

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

DIPPING INTO D3

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

```
d3.select(“.container”)
```

then, you use D3 methods to manipulate this selection:

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

DIPPING INTO D3

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

- Select element with class name “container”
- 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>

LET’S DRAW A CIRCLE

Open Exercise 2, and take a look at “script/script.js”

DRAWING A CIRCLE

```
d3.select(“.canvas”)
    .append(“svg”)
    .attr(“width”,width)
    .attr(“height”,height)
    .append(“circle”)
    .attr(“cx”,100)
    .attr(“cy”,100)
    .attr(“r”,50);
```

Select the <div> element with class “canvas”

Add an <svg> element

Set the attributes on <svg>

Add a <circle> element under <svg>

Set the attributes on <circle>

DRAWING A CIRCLE

```
d3.select(“.canvas”)
    .append(“svg”)
    .attr(“width”,width)
    .attr(“height”,height)
    .append(“circle”)
    .attr(“cx”,100)
    .attr(“cy”,100)
    .attr(“r”,50);
```

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

DRAWING A CIRCLE: “CHAINING” IN D3

```
d3.select(“.canvas”)
    .append(“svg”)
    .attr(“width”,width)
    .attr(“height”,height)
    .append(“circle”)
    .attr(“cx”,100)
    .attr(“cy”,100)
    .attr(“r”,50);
```

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 element as the new selection, for you to call a new method onto it.

DRAWING A CIRCLE: “CHAINING” IN D3

```
d3.select(“.canvas”)             Returns “.canvas”
    .append(“svg”)               Returns <svg>
    .attr(“width”,width)         Returns <svg>
    .attr(“height”,height)       Returns <svg>
    .append(“circle”)            Returns <circle>
    .attr(“cx”,100)              Returns <circle>
    .attr(“cy”,100)              Returns <circle>
    .attr(“r”,50);               Returns <circle>
```

DRAWING A CIRCLE: “CHAINING” IN D3

```
d3.select(“.canvas”)               Returns “.canvas”
    .append(“svg”)                 Returns <svg>
    .attr(“width”,width)           Returns <svg>
    .attr(“height”,height)         Returns <svg>
    .append(“circle”)              Returns <circle>
    .attr(“cx”,100)                Returns <circle>
    .attr(“cy”,100)                Returns <circle>
    .attr(“r”,50)                  Returns <circle>
    .append(“circle”) //??
    .attr(“cx”,200)
    .attr(“cy”,200)
    .attr(“r”,50);
```

DRAWING A CIRCLE: “CHAINING” IN D3

```
var elem = d3.select(“.canvas”)
    .append(“svg”)
    .attr(“width”,width)
    .attr(“height”,height)
    .append(“rect”)
    .attr(“width”,100)
    .attr(“height”,100);
```

What is the variable `elem`?

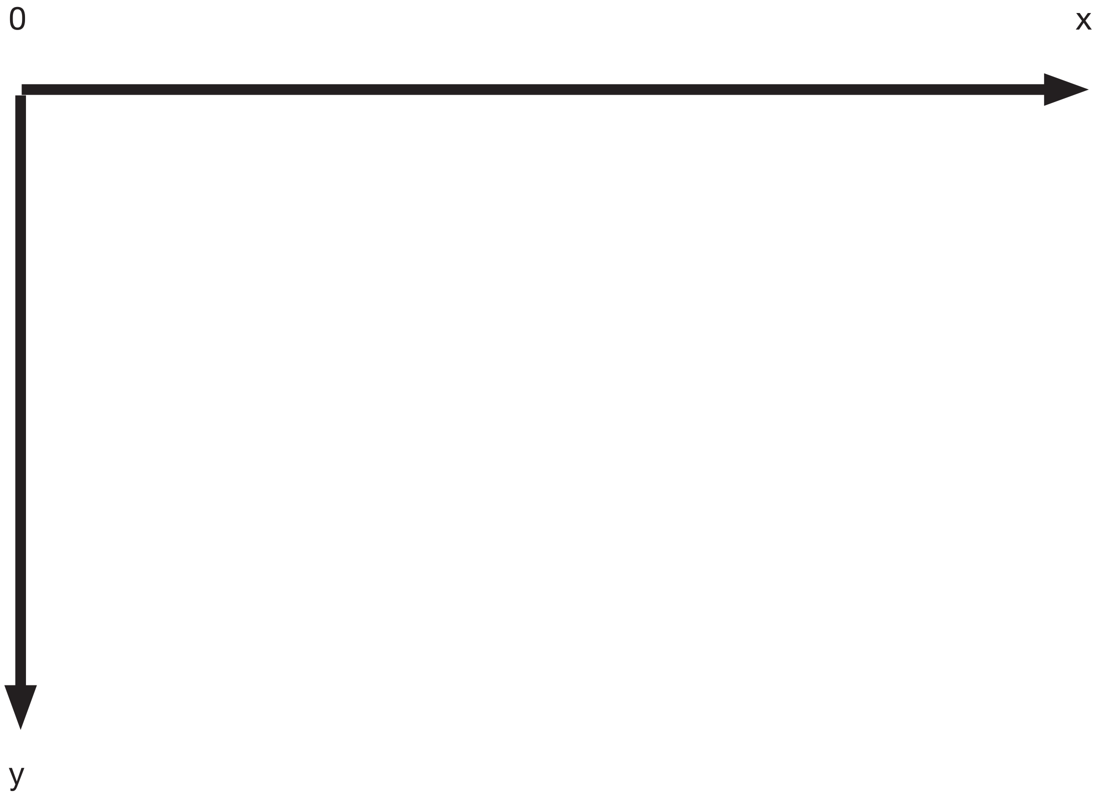
How would use this to our advantage?

HOW DO WE DRAW COMMON SVG ELEMENTS?

Open Exercise 3, and take a look at "script/script.js"; also take a look at "style.css"

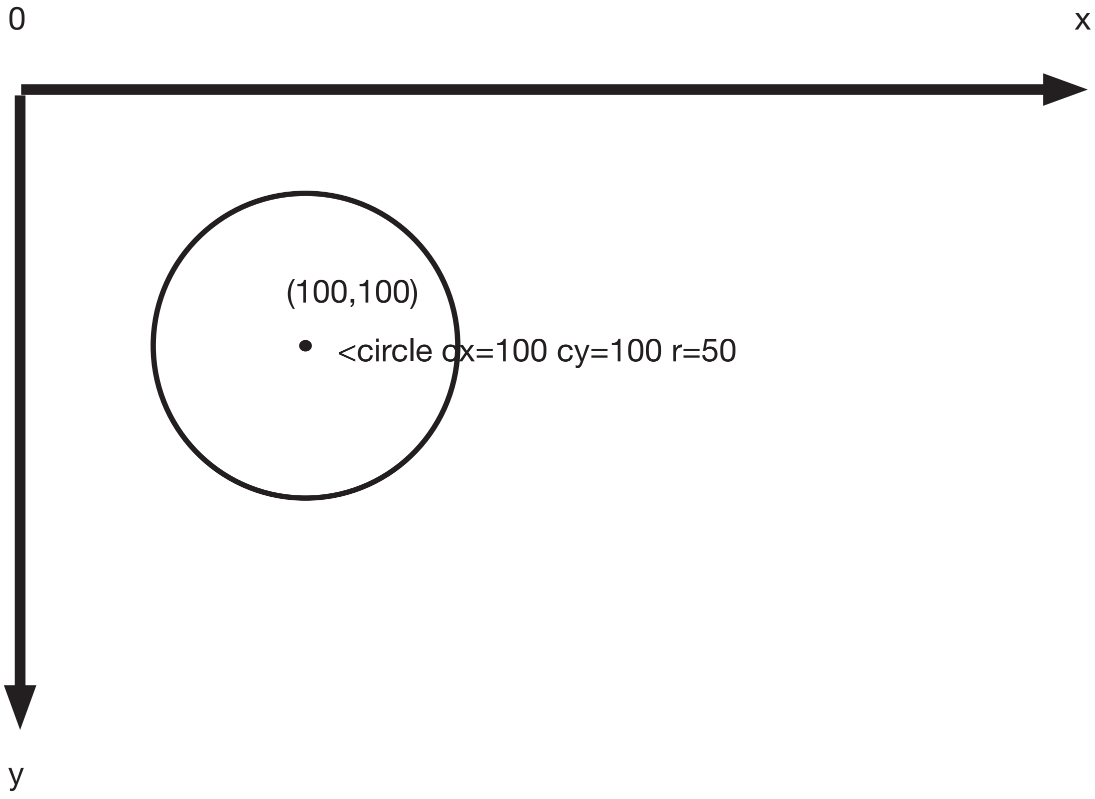
COORDINATES IN SVG

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



COORDINATES IN SVG

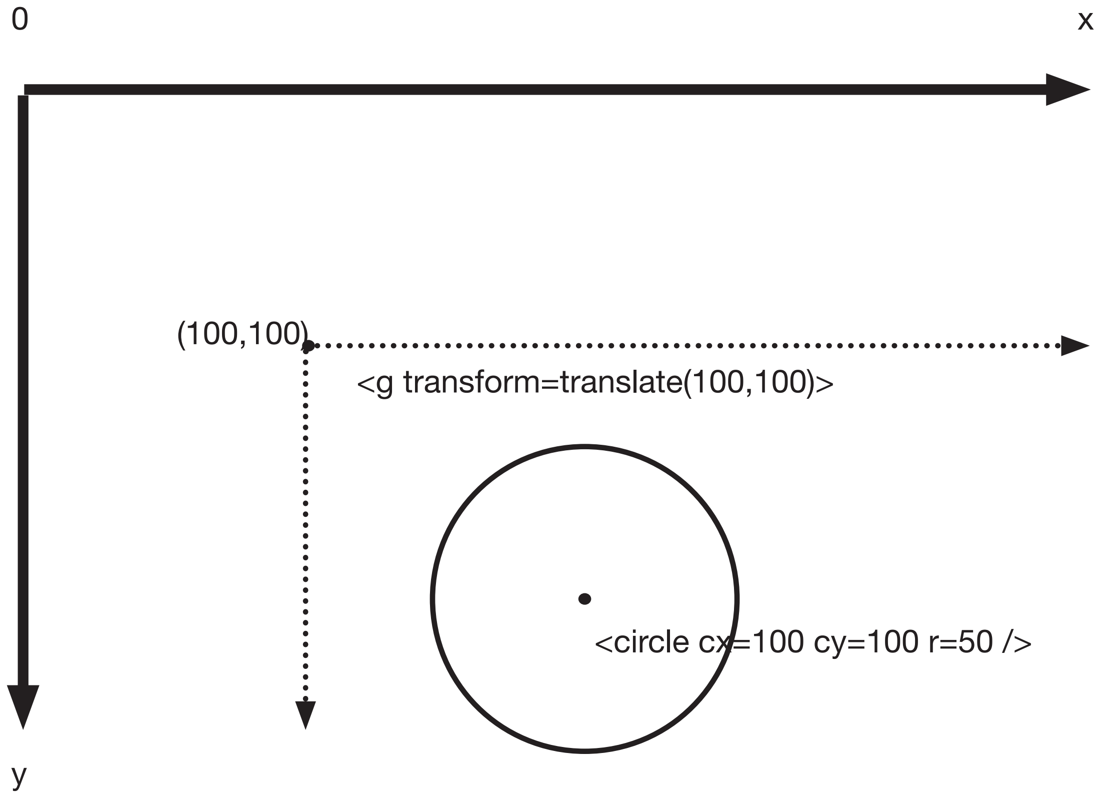
```
<svg>
     <circle ... />
</svg>
```



COORDINATES IN SVG

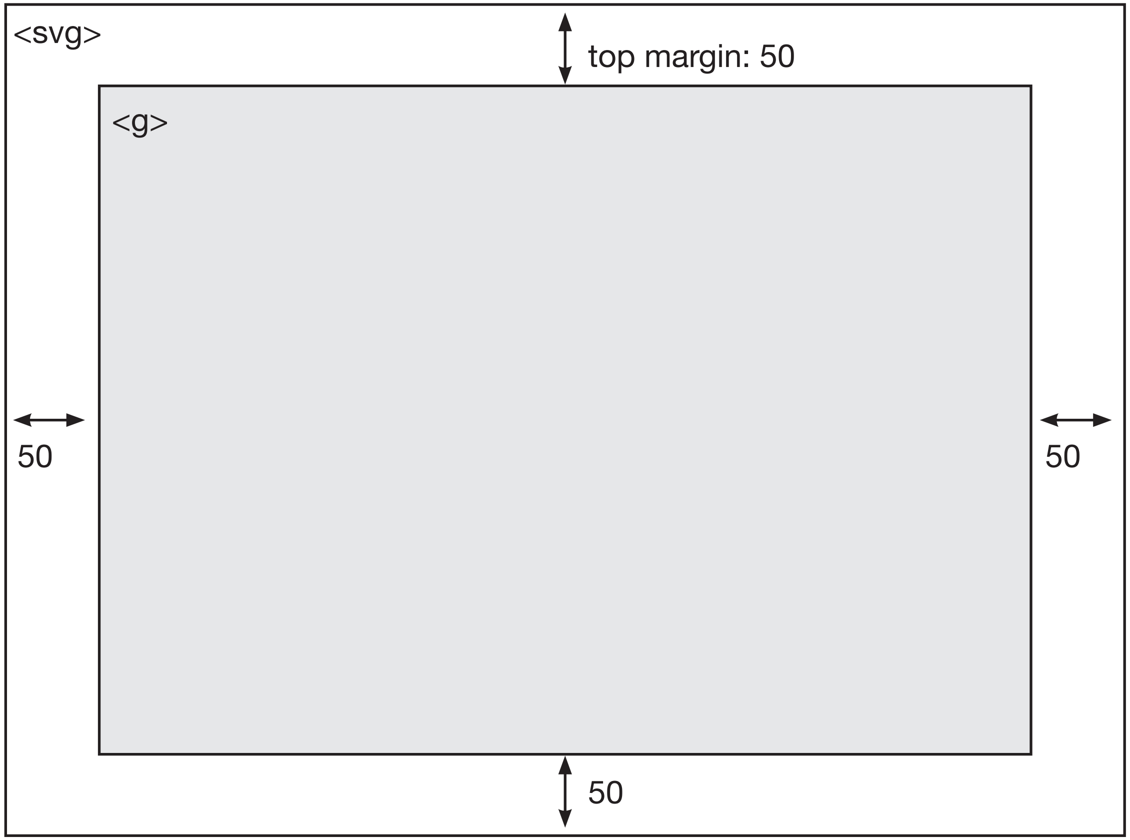
We use <g> to group individual elements; each <g> starts its own coordinate system.

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



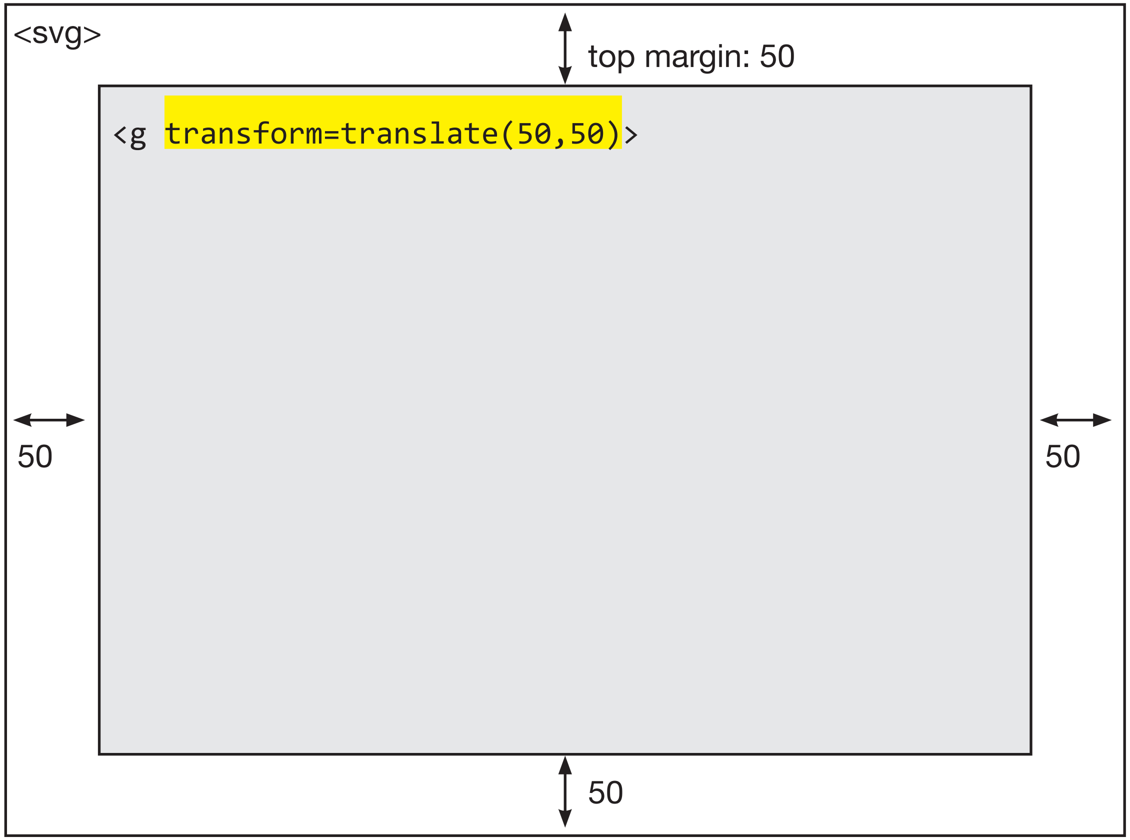
MARGIN CONVENTIONS

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.



MARGIN CONVENTIONS

Let’s continue with Exercise 3 and incorporate the margin conventions.

PUTTING EVERYTHING TOGETHER

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