CSY1063
Web Development
Week 11

Chris.Rafferty@northampton.ac.uk

Learning Objectives

- This week we will be covering
  - Generating random number
  - Using nextSibling to find elements
  - An introduction to Arrays
  - 2D arrays
  - DOM (Document Object Model) introduction
    - Creating elements
    - Removing elements
  - Closures

Random numbers

- In JavaScript random numbers can be generated using the code:
- This will store a random number between 0 and 1 inside the variable randomNumber
  - e.g. 0.23445466 or 0.93445477

```
let randomNumber = Math.random();
```

Random numbers pt2

- This will always generate a random number between 0 and 1
- In most cases this value will not be useful on its own
- However, it's possible to change the maximum value by multiplying the random number by your chosen maximum

```
let randomNumber = Math.random() * 10;
```

- 8.23344564
- 2.04933531
- 0.12467897

Random numbers pt3

- The chance of a whole number e.g. 5.00000 being generated is tiny
- Most of the time you will want a whole number e.g. 1,2,3 etc
- It's possible to round the generated number by using the Math.round() function:

```
let randomNumber = Math.round(Math.random() * 10);
```

Issue with rounding

- This will round the number to the nearest whole number e.g.
  - 1.65758 becomes 2
  - 9.23581 becomes 9

- However, this is not a very good method of creating a random number:
  - **0.0** - **0.49999** will round to zero (0.5 range)
  - **0.5** - **1.49999** will round to 1.
    - There is twice as much chance of getting a 1 as a zero (1.0 range)
  - **1.5** - **2.49999** will round to 2.
    - There is twice as much chance of getting a 2 as a zero

Random numbers pt4

- Instead of Math.round() there are two other rounding methods:
  - Math.ceil(); which rounds up
  - Math.floor(); which rounds down
- To get a random number between 1 and 10 you can use

```
let randomNumber = Math.ceil(Math.random() * 10);
```

- To get a random number between 0 and 9 you can use

```
let randomNumber = Math.floor(Math.random() * 10);
```

Exercise 1

- Download ex1.zip from NILE
- Using querySelectorAll() and a loop add a click event listener to each dice image (Each dice is a element)
- Each time you click an image, set the class name of the div to “side3”
  - This will display the dice face with 3 dots

- You can use the same event listener function for each dice image and use the “this” variable to set a class on the one that was clicked

```
this.classList = 'side3';
```

Exercise 2

- When you click on one of the dice generate a random number between 1 and 6 and display a message using alert() that says “You rolled a 4”.
- Each time you click on one of the dice it should display the number rolled.

- Adjust exercise 1 so that a random side of the dice is shown when you click on one of the dice
- You will need to set the div's class name to “side1”, “side2”, “side3” etc depending on which number was generated

```
let random = //randomly generated number;
this.classList = 'side' + random;
```

Finding the next element

- It would be better if instead of printing something in an alert box, the paragraph below the dice updated

```html
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Dice Game</title>
    <link rel="stylesheet" href="style.css">
    <script src="ex1.js" defer></script>
</head>
<body>
    <div class="side1">
    </div>
    <p>Roll the dice</p>

    <div class="side1">
    </div>
    <p>Roll the dice</p>
</body>
</html>
```

We need to find the paragraph

- The variable ‘this’ tells us which element was clicked on
- Unfortunately it does not give us an index number
- There's no way to use querySelectorAll('p'); to find the corresponding paragraph tag.
- It's possible to find all the paragraph tags but there is no way to know which one is below the div that was clicked on

nextSibling

- Any element reference (whether found querySelector() or the ‘this’ variable) has a property called nextSibling

```
let element = document.querySelector('#mydiv');
let nextElement = element.nextSibling;
```

```
<div id="mydiv">
</div>

<h2>Heading</h2>
```

nextSibling pt2

- nextSibling finds the next node in the document
- However, not all nodes in the document are elements with tags!
- This will actually match the text between the elements.
- In this case, the whitespace!

```
<div id="mydiv">
</div>

<h2>Heading</h2>
```

nextSibling pt3

- To find the next element you will need to use nextSibling twice
- You can keep using nextSibling to traverse down the entire document

```
let element = document.querySelector('#mydiv');
let nextElement = element.nextSibling.nextSibling;
```

```
<div id="mydiv">
</div>

<h2>Heading</h2>
```

nextSibling pt4

- Once you have selected an element using nextSibling you can make changes to it like any other element

```
let element = document.querySelector('#mydiv');
let nextElement = element.nextSibling.nextSibling;

nextElement.style.backgroundColor = 'red';
nextElement.style.opacity = 0.5;
nextElement.firstChild.nodeValue = 'CHANGED!!!!';
nextElement.addEventListener('click', clicked);

function clicked() {
    alert('clicked');
}
```

nextElementSibling

- To fix this issue the property nextElementSibling was introduced
- nextSibling will find the next available node (including whitespace) while nextElementSibling will only find the next element

```
let element = document.querySelector('#mydiv');
let nextElement = element.nextElementSibling;
```

```
<div id="mydiv">
</div>

<h2>Heading</h2>
```

Arrays

- So far, most variables you have assigned a value to have contained a single value

```
let number = 4;
let string = 'Hello there';
let element = document.querySelector('#h1Tag');
let diceRoll = Math.ceil(Math.random() * 6);
```

- A different type of variable known as an array lets you store more than one value in a single variable

Arrays pt2

- You have already used an array when using
  - querySelector**All**()

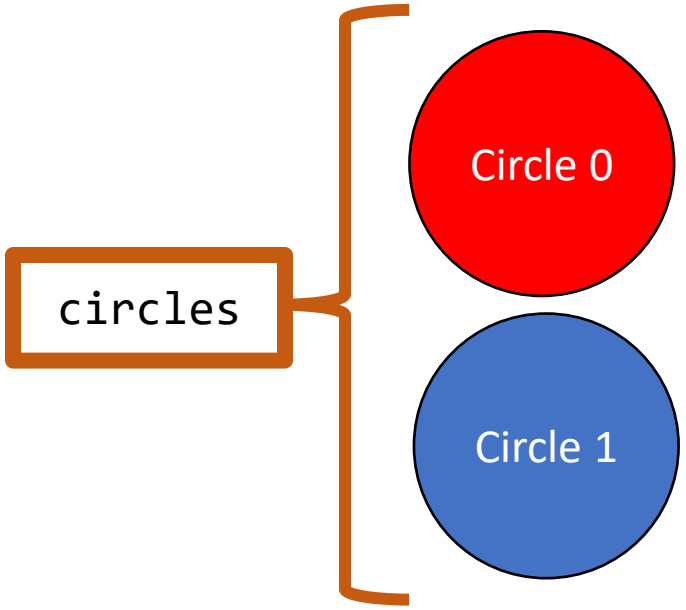
```
<div class="circle">
</div>

<div class="circle">
</div>
```

```
let circles = document.querySelectorAll('.circle');
circles[0].style.backgroundColor = 'red';
circles[1].style.backgroundColor = 'blue';
```

Arrays pt3

- In this example, the circles variable is an array
- The variable stores more than one element
  - circles[0] (The first circle)
  - circles[1] (The second circle)

```
let circles = document.querySelectorAll('.circle');
circles[0].style.backgroundColor = 'red';
circles[1].style.backgroundColor = 'blue';
```



Arrays pt4

- You can also create your own arrays
- Arrays can store any type of variable
- Arrays are used to map one value to another

Creating an array

- To create an array you use two square brackets

```
let myArray = [];
```

- This will create an empty array, you can then write values to individual indexes

```
let myArray = [];
myArray[0] = 123;
myArray[1] = 456;
myArray[2] = 789;
myArray[3] = 10;
```

Creating an array pt2

- An array consists of two parts:
- Indexes - The value you will use to access the value
- Values - The value stored under the named key

```
myArray[INDEX] = VALUE;
```

Creating an array pt3

- You can store any type of variable in the array

```
let myArray = [];
myArray[0] = 'Red';
myArray[1] = 'Green';
myArray[2] = 'Blue';
```

- Once the array is created, you can reference the value like any other variable

```
alert(myArray[1]);
```

Creating an array pt4

- You can also use a variable in place of the numerical index

```
let myArray = [];
myArray[0] = 'Red';
myArray[1] = 'Green';
myArray[2] = 'Blue';

let num = 1;

alert(myArray[num]);
```

2D array

- You can even store an array inside of another array, this is called a 2D array

```
let myArray = [
    [1, 2, 3],
    [4, 5, 6],
    [7, 8, 9]
];
```

- To access the values you will need two index positions
  - The first is for the array (row)
  - Second for the value inside the array (column)

```
myArray[0][1]; //is equal to 2 (first array second value)
```

```
let myArray = [
    [1, 2, 3],
    [4, 5, 6],
    [7, 8, 9]
];

myArray[0][1];
```

```
let myArray = [
    [1, 2, 3],
    [4, 5, 6],
    [7, 8, 9]
];

myArray[1][2];
```

```
let myArray = [
    [1, 2, 3],
    [4, 5, 6],
    [7, 8, 9]
];

myArray[2][0];
```

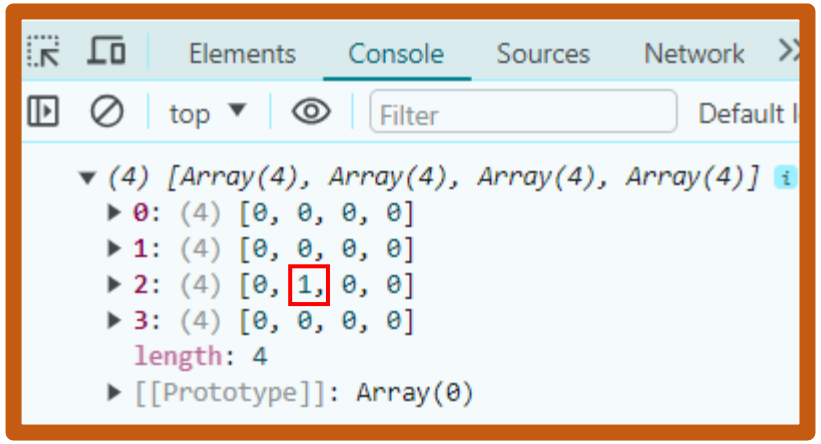
```
let myArray = [
    [1, 2, 3],
    [4, 5, 6],
    [7, 8, 9]
];

myArray[1][1];
```

Random numbers in a 2D array

- You can use two randomly generated numbers to change the values of a 2D array
- Random numbers for
  - row
  - Column

```
let maze = [
    [0, 0, 0, 0],
    [0, 0, 0, 0],
    [0, 0, 0, 0],
    [0, 0, 0, 0]
];
```

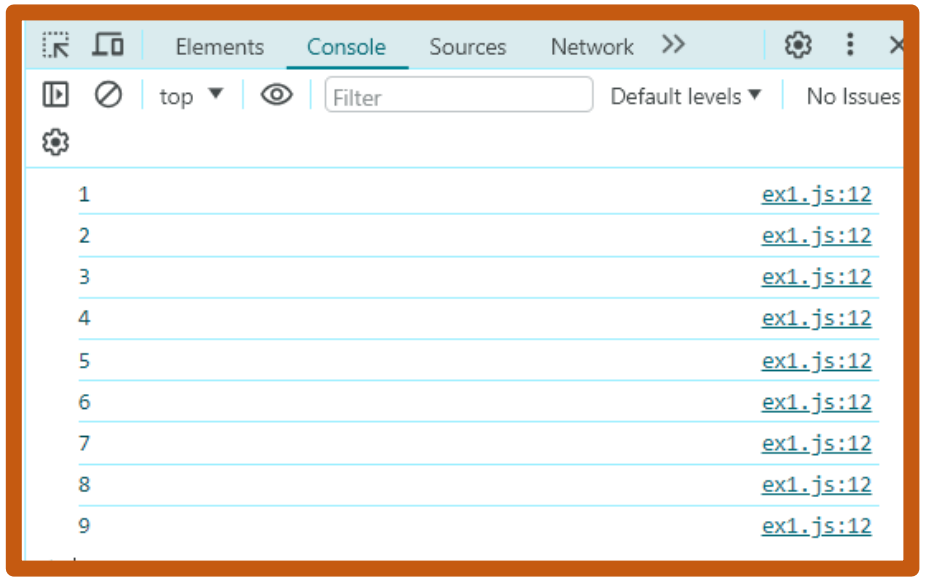
```
let row = Math.floor(Math.random() * maze.length);
let column = Math.floor(Math.random() * maze[row].length);

maze[row][column] = 1;

console.log(maze);
```



Looping through a 2D array

- We can loop through a 2D array by using a loop inside another loop
  - The first loop is for the arrays
  - Second loop for the values in each array

```
let myArray = [
    [1, 2, 3],
    [4, 5, 6],
    [7, 8, 9]
];

for(let i = 0; i < myArray.length; i++) {
    for(let j = 0; j < myArray[i].length; j++) {
        console.log(myArray[i][j]);
    }
}
```



For of loop

- A for of loop can be used to simplify this
  - Both produce the same result

```
let myArray = [
    [1, 2, 3],
    [4, 5, 6],
    [7, 8, 9]
];

for(let i = 0; i < myArray.length; i++) {
    for(let j = 0; j < myArray[i].length; j++) {
        console.log(myArray[i][j]);
    }
}
```

```
let myArray = [
    [1, 2, 3],
    [4, 5, 6],
    [7, 8, 9]
];

for(let row of myArray) {
    for(let value of row) {
        console.log(value);
    }
}
```

AS2 Maze

- The maze for AS2 is populated by using a 2D array and a loop
- Values in the 2D array are used in a switch statement to determine which classes are added to the maze grid (10x10 CSS grid)
  - 1 is a wall (blue square) ■
  - 2 is the player ●
  - 3 is an enemy (green circles) ●
  - 0 are points (white circles) ◦

- Using a switch statement each condition (case) is checked
  - if the value == 1 add the wall class to the CSS grid

AS2 Maze (code)

```
//Player = 2, Wall = 1, Enemy = 3, Point = 0
let maze = [
    [1, 1, 1, 1, 1, 1, 1, 1, 1, 1],
    [1, 2, 0, 1, 0, 0, 0, 0, 3, 1],
    [1, 0, 0, 0, 0, 0, 0, 1, 1, 1],
    [1, 0, 0, 0, 0, 0, 0, 0, 0, 1],
    [1, 0, 1, 1, 0, 0, 0, 0, 0, 1],
    [1, 0, 0, 0, 0, 0, 0, 1, 1, 1],
    [1, 0, 0, 1, 0, 3, 0, 0, 0, 1],
    [1, 0, 0, 0, 0, 0, 0, 1, 0, 1],
    [1, 3, 1, 0, 0, 0, 0, 0, 0, 1],
    [1, 1, 1, 1, 1, 1, 1, 1, 1, 1]
];
```

```
//Populates the maze in the HTML
for (let y of maze) {
    for (let x of y) {
        let block = document.createElement('div');
        block.classList.add('block');

        switch (x) {
            case 1:
                block.classList.add('wall');
                break;
            case 2:
                block.id = 'player';
                let mouth = document.createElement('div');
                mouth.classList.add('mouth');
                block.appendChild(mouth);
                break;
            case 3:
                block.classList.add('enemy');
                break;
            default:
                block.classList.add('point');
                block.style.height = '1vh';
                block.style.width = '1vh';
        }

        main.appendChild(block);
    }
}
```

Exercise 3

- Using nextElementSibling, set the content of the <p> tag to the message “You rolled a 3” (or whatever the random number was) instead of displaying it in an alert popup

- (Optional) There is also a previousSibling property that stores the element above the one selected in the document.
- Adjust the game so clicking the <p> tag rolls the dice instead of the dice image

Document Object Model (DOM)

- HTML files are processed with JavaScript using the DOM (Document Object Model)
- This is a standardised way of accessing information from XML and HTML documents
- Each element can contain other elements or have properties (attributes)

DOM

- There are two main types of node in a HTML document:
  - Element: <p> tags, tags, <h2> tags, etc that represent HTML elements
  - Text nodes: These store the text inside the element

- Elements can contain other elements
- Text nodes only contain text
- Most HTML documents contain a mix of text nodes and elements

Text nodes

- Text nodes have a nodeValue attribute which stores the text that the node contains
- You have already used this when updating the content of a node:

```
let elements = document.querySelectorAll('p');
elements[0].firstChild.nodeValue = 'Paragraph text to set';
```

Text nodes pt2

- To write content to a text node, you first have to select it.
- element.firstChild selects the first child node inside the <p> element. In this case, the first child node is a text node
- The text node can then have its nodeValue set to update the content

```
let elements = document.querySelectorAll('p');
elements[0].firstChild.nodeValue = 'Paragraph text to set';
```

Child nodes

- Every element has zero or more child nodes
- These are text nodes and elements that exist inside the selected element
  - The <ul> element is a child node of the <aside> element
  - The <li> elements are child nodes of the <ul> element

```
<aside>
    <ul>
        <li>item 1</li>
        <li>item 2</li>
        <li>item 3</li>
        <li>item 4</li>
        <li>item 5</li>
    </ul>
</aside>
```

Child nodes pt2

- You can read an element's child nodes as an array
- Once you have the childNodes you can select a specific node using the array syntax

```
const element = document.querySelector('ul');
let childNodes = element.childNodes;

let nodeOne = childNodes[0];
let nodeTwo = childNodes[1];
let nodeThree = childNodes[2];
```

Issue with childNodes

- However, childNodes also contains the text nodes (even whitespace!)

```
<aside>
    <ul>[  childNodess[0] - text  ]
        <li>item 1</li>[  childNodess[2] - text  ]
        <li>item 2</li>[  childNodess[4] - text  ]
        <li>item 3</li>[  childNodess[6] - text  ]
        <li>item 4</li>[  childNodess[8] - text  ]
        <li>item 5</li>[  childNodess[10] - text  ]
    </ul>
</aside>
```

```
const element = document.querySelector('ul');
let childNodes = element.childNodes;

childNodes[0];
childNodes[1];
childNodes[2];
childNodes[3];
childNodes[4];
childNodes[5];
childNodes[6];
childNodes[7];
childNodes[8];
childNodes[9];
childNodes[10];
```

querySelectorAll()

- Because of this, it's usually better to use querySelectorAll instead of childNodes.
- querySelectorAll does not retrieve the text nodes, only the elements

```
const li = document.querySelectorAll('ul li');
li[0].style.backgroundColor = 'red';
li[1].style.backgroundColor = 'green';
li[2].style.backgroundColor = 'blue';
```

- item 1
- item 2
- item 3
- item 4
- item 5

Other properties

- There are other properties available to find elements in the document
- Earlier we looked at nextElementSibling and previousSibling
- This selects the next node (text or element) relative to the element it is called on

parentNode

- There is also a parentNode property which selects the parent element of selected element
- This will always select an element and not a text node

```
const li = document.querySelectorAll('li');
let parentNode = li[0].parentNode;
```

```
<aside>
    <ul>
        <li>item 1</li>
        <li>item 2</li>
        <li>item 3</li>
        <li>item 4</li>
        <li>item 5</li>
    </ul>
</aside>
```

parentNode pt2

- Like nextSibling, you can chain parentNode selectors to climb up the document

```
const li = document.querySelectorAll('li');
let parentNode = li[0].parentNode.parentNode;
```

```
<aside>
    <ul>
        <li>item 1</li>
        <li>item 2</li>
        <li>item 3</li>
        <li>item 4</li>
        <li>item 5</li>
    </ul>
</aside>
```

DOM pt2

- By combining
  - querySelector
  - querySelectorAll
  - childNodes
  - parentNode
  - nextSibling/previousSibling

- It's possible to select any element or text node on any HTML page

Creating Elements

- JavaScript can also create elements
- This is done using

```
const element = document.createElement('tagName');

const divTag = document.createElement('div');
const pTag = document.createElement('p');
const h2Tag = document.createElement('h2');
```

Creating elements pt2

- Once an element has been created you can treat it like any other element
  - Set CSS properties
  - Add event listeners
  - Add CSS Classes/IDs

```
const element = document.createElement('div');
element.style.width = '200px';
element.style.height = '200px';
element.style.backgroundColor = 'red';
```

Creating elements pt3

- Once an element has been created it exists as a variable but is not visible on the page
- Before you can put the element on the page, you need to decide where to put it
- This is done by finding the element you want to add it to and then using the function appendChild

Adding elements to the page

- This code will create an element and add it to the <body> tag
  - You can use document.body to get the body tag

```
const element = document.createElement('div');
element.style.width = '200px';
element.style.height = '200px';
element.style.backgroundColor = 'red';

document.body.appendChild(element);
```

Adding elements to the page pt2

- This code will create an element and add it to the <main> tag

```
const element = document.createElement('div');
element.style.width = '200px';
element.style.height = '200px';
element.style.backgroundColor = 'red';

const main = document.querySelector('main');
main.appendChild(element);
```

Creating text nodes

- To change the text of an element you use the code:

```
element.firstChild.nodeValue = 'text to display';
```

- This won't work with newly created elements because there is no text node to update
- element.firstChild doesn't actually exist
- When an element is created, it does not contain any text nodes

Creating text nodes pt2

- To add text to a newly created element you can use .createTextNode()
- Once the text node has been created, you can use appendChild to add it to the new element
- Now when the element is added to the page it will contain the text “text to create”

```
const textNode = document.createTextNode('text to create');

const element = document.createElement('p');
element.appendChild(textNode);
document.body.appendChild(element);
```

Removing elements

- Along with adding elements to the page, you can also remove them
- This is done using

```
parentElement.removeChild(elementToBeRemoved);
```

Remove child

- Given the following HTML
- The <p> element can be removed with the code:

```html
<!doctype html>
<html>

<head>
    <title>Example page</title>
</head>

<body>
    <main>
        <h1>Example page</h1>
        <p>Some example text</p>
    </main>
</body>

</html>
```

```javascript
const main = document.querySelector('main');
const paragraph = document.querySelector('p');

main.removeChild(paragraph);
```

Removing elements pt2

- This involves finding two elements:
  - The element that contains the element you want to remove
  - The element you want to remove
- This can be extra work

- It's possible to do this by utilising parentNode and only finding the element you want to remove

```
const paragraph = document.querySelector('p');
paragraph.parentNode.removeChild(paragraph);
```

Mouse position

- You can read the X/Y (top and left) mouse position of a click event using the properties:
  - event.clientX
  - event.clientY

```
function myClickEvent(event) {
    alert('You clicked at the position left: ' + event.clientX + ' top: ' + event.clientY);
}

document.addEventListener('click', myClickEvent);
```

Closures

- A closure is a special type of function
  - Closures do not have function names
  - They are usually defined inside other functions
  - You can declare a closure and use it like a normal function

```
setInterval(function() {
    console.log('Hello World');
}, 1000);
```

Closures pt2

- These two pieces of code produce the same result

```
function clicked() {
    alert('clicked');
}

document.addEventListener('click', clicked);
```

```
document.addEventListener('click', function(){
    alert('clicked');
});
```

Closures pt3

- Closures can be used anywhere you reference a function name in an event

```
let num = 0;

setInterval(function(){
    num = num + 1;
    console.log(num);
}, 100);

document.addEventListener('keydown', function(event) {
    alert('Key pressed = ' + event.key);
});
```

Closures pt4

- Closures are useful because they have access to the variables from the function that calls them

```
function myClickFunction() {
    let number = 1;

    setInterval(function(){
        console.log(number); //Can access number
    }, 1000);
}

document.addEventListener('click', myClickFunction);
```

Closures pt5

- When you create a closure, it is a new version of the function and any variables are copied
  - Each time you create a new version of the closure it has access to its own set of variables
  - The closure below will create a new random number each time there is a click

```
document.addEventListener('click', function(){
    let num = Math.floor(Math.random() * 100);
    console.log(num);
});
```

AS2 maze

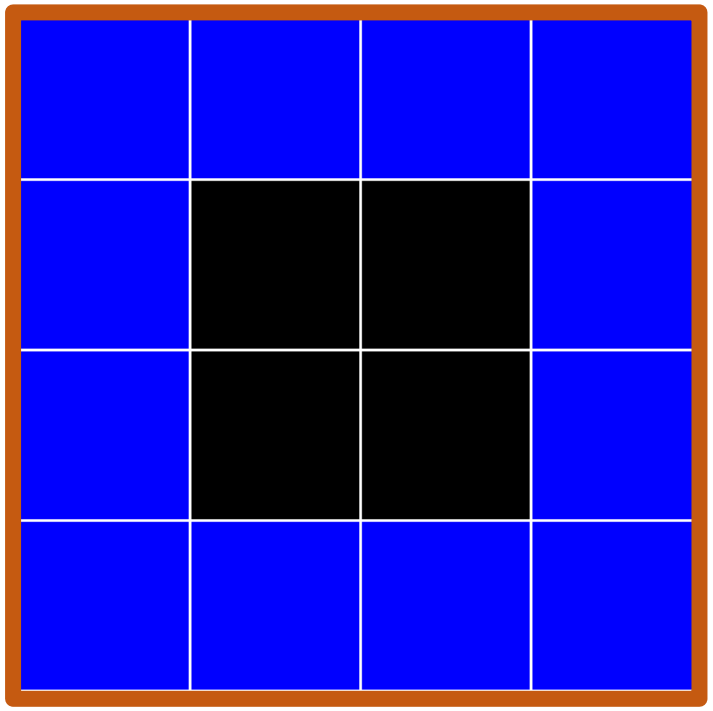
- For the assignment a new div tag is created and appended onto the main for each value in the 2D array maze
  - In the CSS main is a 10 x 10 grid

```
const maze = [
    [0, 0, 0, 0],
    [0, 0, 0, 0],
    [0, 0, 0, 0],
    [0, 0, 0, 0]
]

const main = document.querySelector('main');

for (let row of maze) {
    for (let block of row) {
        let div = document.createElement('div');
        main.appendChild(div);
    }
}
```

AS2 maze (if)

- Using an if statement the value can be checked
  - If the value = 1 set the wall class

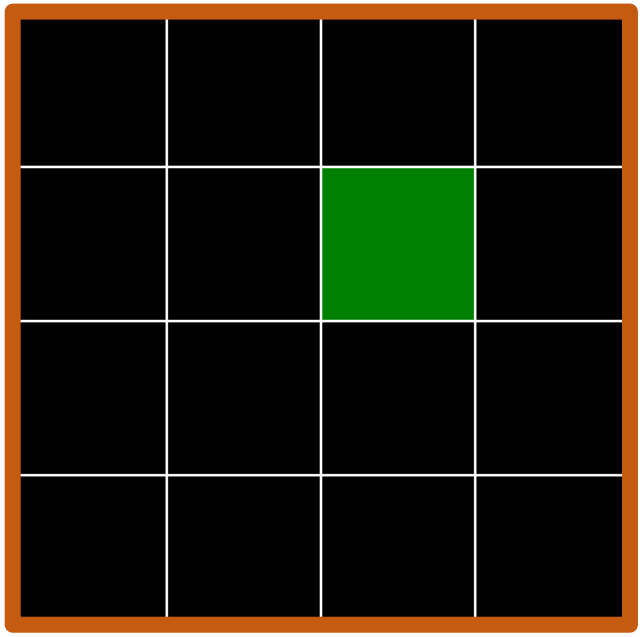
```
const maze = [
    [1, 1, 1, 1],
    [1, 0, 0, 1],
    [1, 0, 0, 1],
    [1, 1, 1, 1]
]

const main = document.querySelector('main');

for (let row of maze) {
    for (let block of row) {
        let div = document.createElement('div');
        main.appendChild(div);

        if(block == 1) {
            div.classList.add('wall');
        }
    }
}
```



AS2 random enemy

- Combing this with random numbers we can randomly generate a div with the enemy class in the maze
  - Random row number
  - Random column number

```
let row = Math.floor(Math.random() * maze.length);
let column = Math.floor(Math.random() * maze[row].length);

maze[row][column] = 3;

for (let row of maze) {
    for (let block of row) {
        let div = document.createElement('div');
        if(block == 3) {
            div.classList.add('enemy');
        }
        main.appendChild(div);
    }
}
```



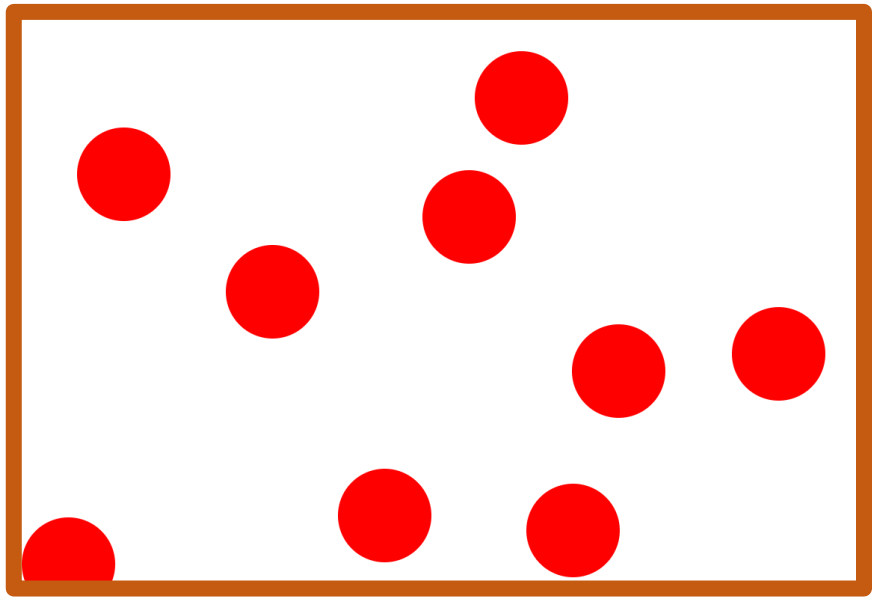
Exercise 4

- Create a HTML page and use a script tag to write some JavaScript
- Using a click event create a new div element, append the new element to the body
  - Each new click should create a new div element
- Add relevant CSS to make the div a 100x100px red circle.

- Using event.clientX & event.clientY set the top and left to the location of your mouse
- **Hint**: You need to set the position to absolute for the divs

Exercise 4 example

```
function createCircle(event) {
    let div = document.createElement('div');
    document.body.appendChild(div);

    //Add the width, height, backgroundColor

    div.style.position = 'absolute';

    div.style.top = event.clientY + 'px';
    div.style.left = event.clientX + 'px';
}

document.addEventListener('click', createCircle);
```



Exercise 5

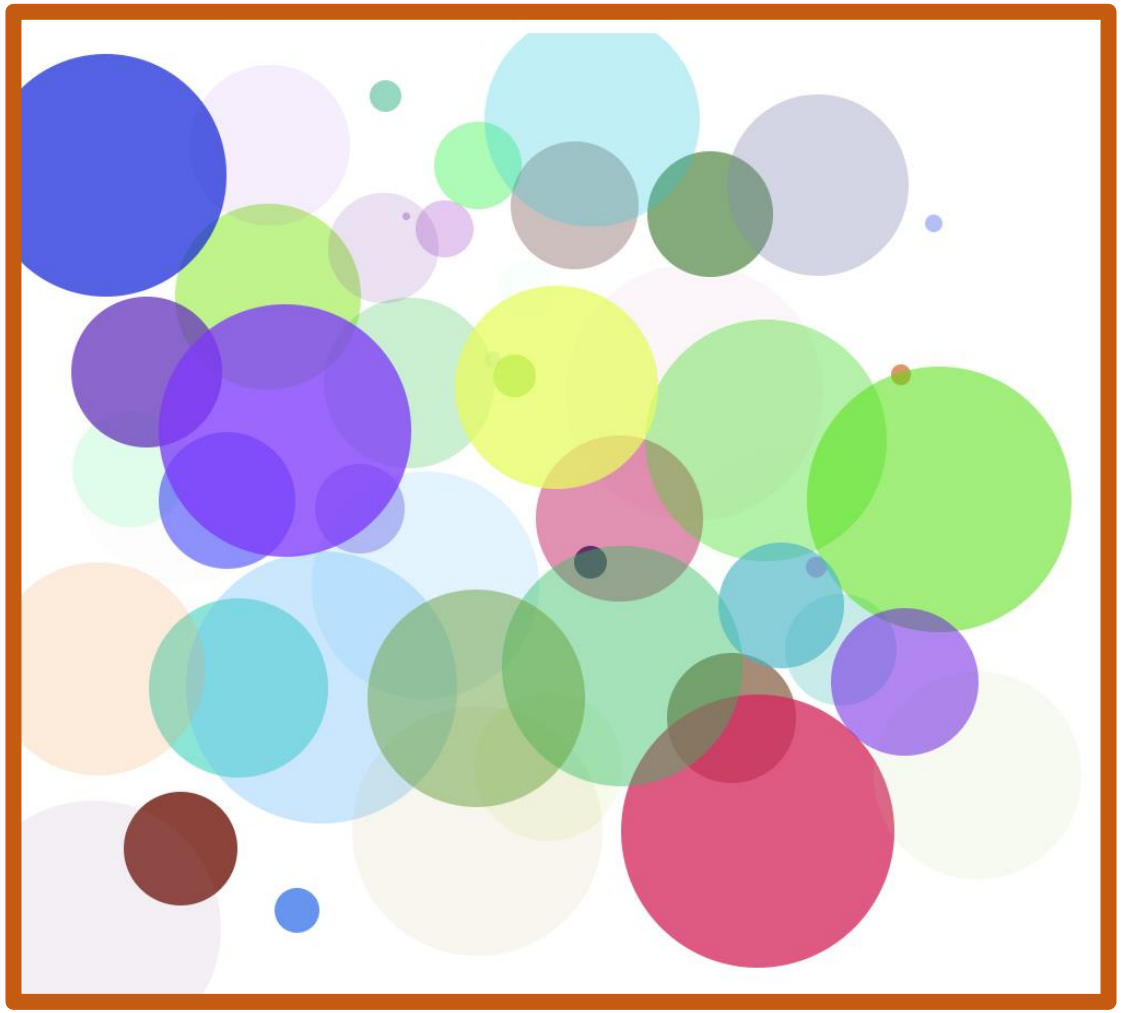
- Create three random numbers between 0 – 255
- Using backgroundColor rgb() randomise the colours of the circles

```
div.style.backgroundColor = 'rgb(' + randomNumber1 + ',' + randomNumber2 + ',' + randomNumber3 + ')';
```

- Randomly set the width/height to a value between 1 and 200
- Change the opacity to a decimal number between 0 - 1

- **Hint**: Look back at the random number slides

Exercise 5 example



Exercise 6

- Add movement to each circle, when the circle is created they should move downwards
  - Clicking multiple times should start multiple circles moving downward
- Randomise the speed of each circle

- When the circle hits the bottom of the screen make start moving upwards
- **Hint**: Week 9 screen collision

- **Optional:** Make the circle move in a random direction and speed and bounce off the edges of the screen

Useful Links

- https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Math/random
- https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Math/ceil
- https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Math/floor
- https://developer.mozilla.org/en-US/docs/Web/API/Element/nextElementSibling
- https://developer.mozilla.org/en-US/docs/Web/API/Element/previousElementSibling
- https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Array
- https://developer.mozilla.org/en-US/docs/Web/API/Document_Object_Model/Introduction
- https://developer.mozilla.org/en-US/docs/Web/API/Document/createElement
- https://developer.mozilla.org/en-US/docs/Web/API/Node/appendChild
- https://developer.mozilla.org/en-US/docs/Web/API/Document/createTextNode
- https://developer.mozilla.org/en-US/docs/Web/API/Node/parentNode
- https://developer.mozilla.org/en-US/docs/Web/API/Node/removeChild
- https://developer.mozilla.org/en-US/docs/Web/JavaScript/Closures