CSY1063
Web Development
Week 10

Chris.Rafferty@northampton.ac.uk

Learning Objectives

- This week we will be covering
  - Collision detection
    - elementFromPoint()
    - Multiple points of collision
    - Precise collision
  - Removing repetition
    - For loops
    - .length
    - The **this** variable

Week 8 Recap - Script

- Using the <script> tag to link JavaScript to HTML
- Defer can be used to specify that the code in our script should be executed after the HTML
  - <script> tags should go inside the pages <head> tag

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Document</title>
    <script src="script.js" defer></script>
</head>
```

Week 8 Recap - Variables

- In modern JavaScript there are two different types of variables
  - Let
  - Const

```
let variableName = 'value';
```

- By declaring a variable with let we can reassign the value of that variable at any time

```
const variableName = 'value';
```

- Const variables cannot be reassigned, they cannot be changed once they have been declared
- If you try changing the value of a const you will get an error

Week 8 Recap - Functions

- You can label a block of code using a function
- To run the code in the addition function, it must be called using the code
  - addition();

```
function addition() {
    let num1 = 5;
    let num2 = 10;

    let total = num1 + num2;

    alert(total);
}

addition();
```

Week 8 Recap - Selecting elements

- JavaScript contains inbuilt functions for selecting HTML elements
- Once an element on the page has an ID, we can use the JavaScript function document.querySelector() to select it and store the element in a variable

```
<body>

    <h1 id="heading">Heading</h1>

    <p id="content">Content</p>

</body>
```

```
let heading = document.querySelector('#heading');

heading.firstChild.nodeValue = 'This has been changed!';
```

Week 8 Recap - Event Listeners

- We can use event listeners to run a function when a specific event occurs
- There is a ‘click’ event which triggers whenever an element is clicked on
  - The heading will only change when the user clicks on the page

```
function changeHeading() {
    let heading = document.querySelector('#heading');
    heading.firstChild.nodeValue = 'This has been changed!';
}

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

Week 9 Recap - CSS

- You can set CSS properties on an element using JavaScript
- Once you have a reference to the element in a variable you can change the CSS on it using

```
element.style.propertyName = 'propertyValue';
```

```
element.style.backgroundColor = 'green';
element.style.borderRadius = '50px';
element.style.fontFamily = 'Verdana, Helvetica, Sans-serif';
element.style.height = '50px';
```

Week 9 Recap - Intervals

- There are events that are based on timers
- You can get the browser to repeatedly run a function based on a time interval

```
function myInterval() {
    console.log('myInterval called');
}

setInterval(myInterval, 1000);
```

- This will run the function myInterval every second
- Intervals are measured in 1000ths of a second

Week 9 Recap – Elements position

- JavaScript provides a simple way of retrieving data on an element's position on the page regardless of how it has been positioned on the page
- Using getBoundingClientRect()
  - You can also retrieve the height and width of the element

```
const circle = document.querySelector('#circle');
let position = circle.getBoundingClientRect();

let positionLeft = position.left;
let positionRight = position.right;
let positionTop = position.top;
let positionBottom = position.bottom;

let height = position.height;
let width = position.width;
```

Week 9 Recap – Moving elements

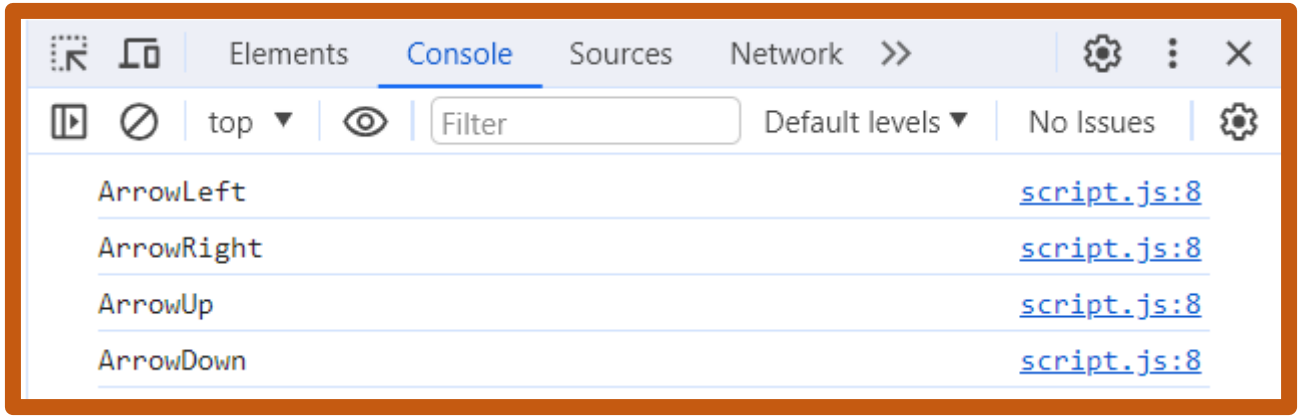
- This will store the position of the element on the page in the variables
- Which allows you to move an element around the screen regardless of its starting position.
- To move an element 10px to the left of where it currently is you can use
  - Note that 'px' is also added to create the valid CSS unit

```
const circle = document.querySelector('#circle');
let position = circle.getBoundingClientRect();
let positionLeft = position.left;

circle.style.left = positionLeft - 10 + 'px';
```

Week 9 Recap – Key events

- We can use event.key to see what key was pressed

```
function moveLeft(event) {
    console.log(event.key)
}

document.addEventListener('keydown', moveLeft);
```



Week 9 Recap – if statements

- An if statement allows you to test so see if two values are equal
- To check to see if the variable event.key is equal to the ArrowLeft

```
function moveLeft(event) {
    if (event.key == 'ArrowLeft') {
        let position = circle.getBoundingClientRect();
        let positionLeft = position.left;
        circle.style.left = positionLeft - 10 + 'px';
    }

    console.log(event.key)
}

document.addEventListener('keydown', moveLeft);
```

Week 9 Recap – Global variables

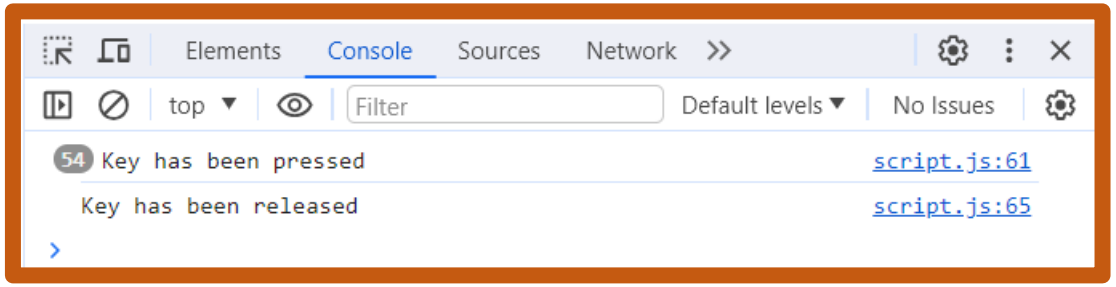
- When a variable is created inside a function it is only accessible inside the function it is created in, and it is recreated each time the function is called
- However, you can create a variable that is available in every function and retains its value when the function is called again
- This is done by declaring the variable outside of any functions

```
let myVariable = 0;

function myClickEvent() {
    myVariable = myVariable + 1;
    console.log(myVariable);
}

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

Week 9 Recap – Key up/down

- By using both keyup and keydown events you can track whether or not a key is being held down

```
function myKeyDown() {
    console.log('Key has been pressed');
}

function myKeyUp() {
    console.log('Key has been released');
}

document.addEventListener('keydown', myKeyDown);
document.addEventListener('keyup', myKeyUp);
```



Week 9 Recap – elements by class name

- You can use querySelectorAll() to retrieve all the elements of that class name
- Because more than one element is retrieved, if you want to make changes to one, you have to specify which element that was matched you'd like to change
- This is done using an index position [number]
  - [0] is the first element

```
const circles = document.querySelectorAll('.circle');

circles[0].style.backgroundColor = 'blue';
circles[1].style.backgroundColor = 'green';
```

Collision detection

- The logic of collision detection is:
- If the player is about to move into something solid, don’t move the player.

- The first step is getting the coordinates of where the player is about to move to without actually updating the player’s position

Current movement function

- The current movement code for the player

```
if(downPressed) {
    playerTop++;
    player.style.top = playerTop + 'px';
    playerMouth.classList = 'down';
}
```

- The players position will always be updated when the left key is pressed
  - playerTop++ is short for playerTop = playerTop + 1;

Where is the player going?

- The first step is storing the position the player is about to move to
  - You can use getBoundingClientRect() for the current bottom position
  - newBottom is where the player is about to move to

```
if(downPressed) {
    let position = player.getBoundingClientRect()
    let newBottom = position.bottom + 1;

    /* playerTop++; */
    /* player.style.top = playerTop + 'px'; */
    playerMouth.classList = 'down';
}
```

Collision detection pt2

- The next step is checking to see if there is something in the way at the new position
- If there's nothing in the way (such as a wall) move the player

```
if (downPressed) {
    let position = player.getBoundingClientRect()
    let newBottom = position.bottom + 1;

    if (/* There is no wall at the new position */) {
        playerTop++;
        player.style.top = playerTop + 'px';
    }
    playerMouth.classList = 'down';
}
```

document.elementFromPoint(x,y)

- You can find the element at a specific X/Y position on the screen using pixel coordinates using
  - document.elementFromPoint(x,y)

```
if (downPressed) {
    let position = player.getBoundingClientRect()
    let newBottom = position.bottom + 1;

    let element = document.elementFromPoint(position.left, newBottom);
    if (/* There is no wall at the new position */) {
        playerTop++;
        player.style.top = playerTop + 'px';
    }
    playerMouth.classList = 'down';
}
```

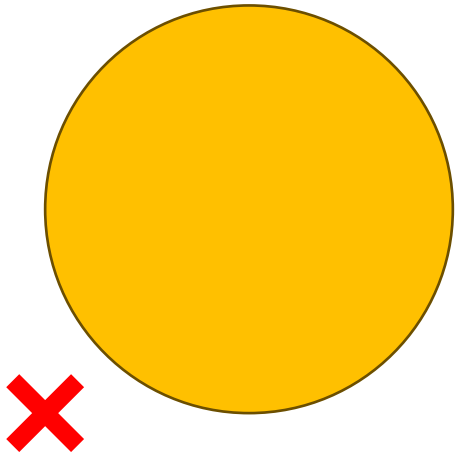
classList.contains()

- You can find out if any element has a particular CSS class using element.classList.contains('nameOfClass');

```
if (downPressed) {
    let position = player.getBoundingClientRect()
    let newBottom = position.bottom + 1;

    let element = document.elementFromPoint(position.left, newBottom);

    if (element.classList.contains('wall') == false) {
        playerTop++;
        player.style.top = playerTop + 'px';
    }

    playerMouth.classList = 'down';
}
```

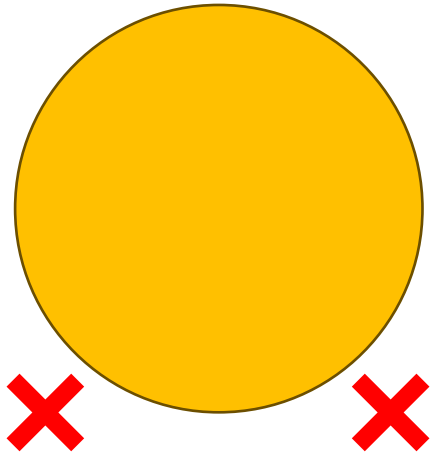
Only one side has detection

- Only the bottom left position of the player has collision detection
  - This means that there is no collision on the right side when moving down



Multiple points of collision

- It would be better to test for the bottom right of the player as well as the bottom left



- Checking for multiple points would increase the accuracy of the collision detection (player won't move through the wall)

Multiple points of collision pt2

- We can store the bottom left and the bottom right positions in two variables
- Using the AND operator (&&) you can check to see if two conditions are met
  - Only move the player downwards while there are no collision (bottom left and right)

```
if (downPressed) {
    let position = player.getBoundingClientRect()
    let newBottom = position.bottom + 1;

    let btmL = document.elementFromPoint(position.left, newBottom);
    let btmR = document.elementFromPoint(position.right, newBottom);

    if (btmL.classList.contains('wall') == false && btmR.classList.contains('wall') == false) {
        playerTop++;
        player.style.top = playerTop + 'px';
    }

    playerMouth.classList = 'down';
}
```

Up collision

- By changing a few of the variables you can add collision to the upwards direction too

```
else if (upPressed) {
    let position = player.getBoundingClientRect();
    let newTop = position.top - 1;

    let topL = document.elementFromPoint(position.left, newTop);
    let topR = document.elementFromPoint(position.right, newTop);

    if (topL.classList.contains('wall') == false && topR.classList.contains('wall') == false) {
        playerTop--;
        player.style.top = playerTop + 'px';
    }
    playerMouth.classList = 'up';
}
```

Exercise

- Download the ex1.zip (extract it too)
- Implement collision detection for the down arrow key using the code from the previous slides
  - The player should not be able to go through the walls of the maze
- Add collision detection for the rest of the arrow keys
  - Up
  - Right
  - Left
- **Hint**: You will need to change the variables depending on the direction

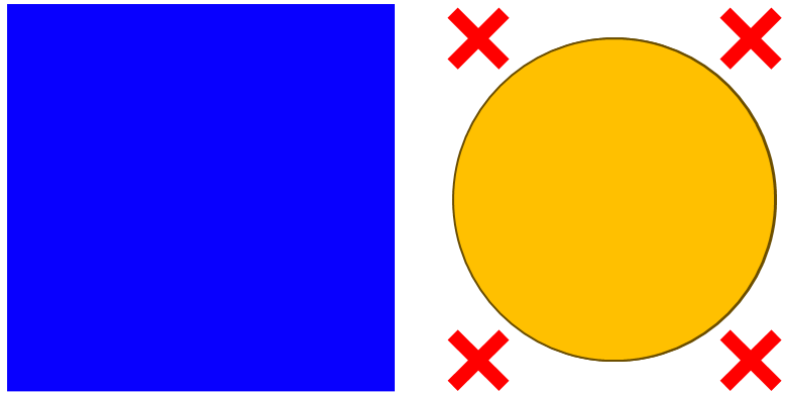
Exercise 2

- Prevent the player from colliding with the walls and the enemies inside the maze (enemy class)
- **Hint**: Rather than checking for multiple CSS classes, give the wall and enemy divs another class like solid and check for that

```
<body>
    <div class="wall top solid"></div>
    <div class="wall left solid"></div>
    <div class="wall right solid"></div>
    <div class="wall bottom solid"></div>

    <div class="enemy solid"></div>
    <div class="enemy solid"></div>
    <div class="enemy solid"></div>

    <div id="player"></div>
</body>
```

Multiple points of collision pt3

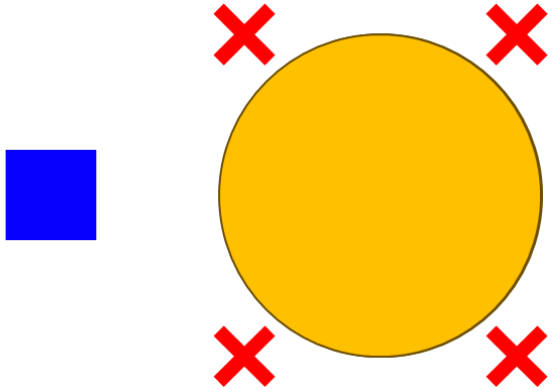
- By adding a point of collision to each corner of the player you can ensure the player never moves through the walls of the maze



- This works well due to the walls being a similar size to the player, one of those positions will always interact with the other

Collision issue

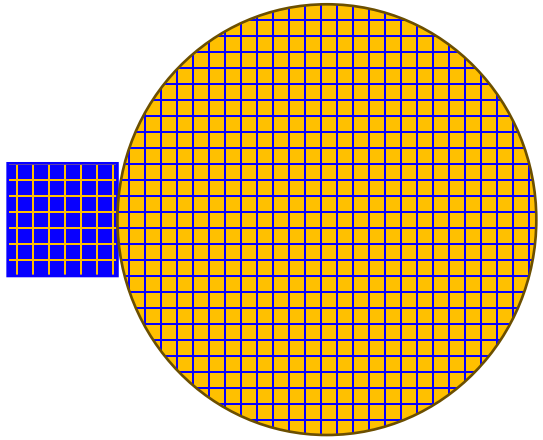
- But what if the element was smaller



- None of the collision variables would react to the wall element
  - This means that the collision is not 100% accurate with smaller elements like the points in the AS2 game

A different type of collision

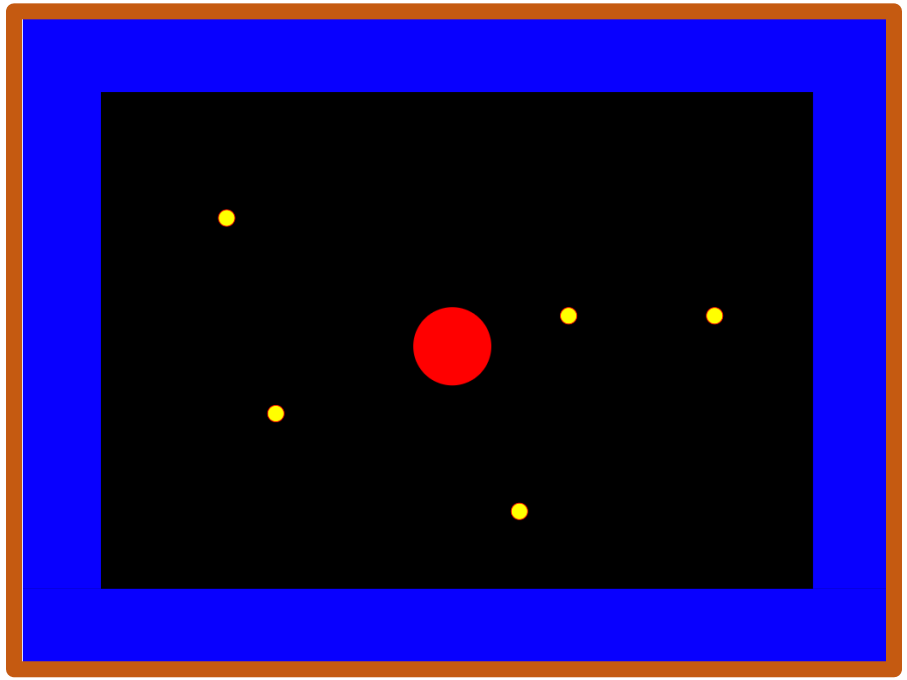
- Instead of specific coordinates what if we check if two elements are overlapping, this would use the area of each elements



- This could be achieved using multiple if statements and the **<** or **>** operators

Ex3 scenario

- For the next exercise the elements are much smaller than the player
- You need a way of finding out if the player has collided with the point elements
  - yellow circles



Find each of the points

- Firstly we need to find the elements
- We can use querySelectorAll and then an index position
  - Assuming we have five point elements

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

```
const point1 = document.querySelectorAll('.point')[0];
const point2 = document.querySelectorAll('.point')[1];
const point3 = document.querySelectorAll('.point')[2];
const point4 = document.querySelectorAll('.point')[3];
const point5 = document.querySelectorAll('.point')[4];
```

The positions for each point

- We then need to use getBoundingClientRect() to find the positions for each element

```
const point1 = document.querySelectorAll('.point')[0];
const point2 = document.querySelectorAll('.point')[1];
const point3 = document.querySelectorAll('.point')[2];
const point4 = document.querySelectorAll('.point')[3];
const point5 = document.querySelectorAll('.point')[4];

const p1Position = point1.getBoundingClientRect();
const p2Position = point2.getBoundingClientRect();
const p3Position = point3.getBoundingClientRect();
const p4Position = point4.getBoundingClientRect();
const p5Position = point5.getBoundingClientRect();
```

Creating the if statement

- We can then use an if statement to check is the elements overlap
  - If all four conditions are true, it means that the two elements are colliding with each other

```
if (
    position.right > p1Position.left &&
    position.left < p1Position.right &&
    position.bottom > p1Position.top &&
    position.top < p1Position.bottom
)
```

Tidying up

- It would be good to have this code in a separate function and then call it when we need it

```
function pointCheck() {
    const position = player.getBoundingClientRect();

    const point1 = document.querySelectorAll('.point')[0];
    const point2 = document.querySelectorAll('.point')[1];
    const point3 = document.querySelectorAll('.point')[2];
    const point4 = document.querySelectorAll('.point')[3];
    const point5 = document.querySelectorAll('.point')[4];

    const p1Position = point1.getBoundingClientRect();
    const p2Position = point2.getBoundingClientRect();
    const p3Position = point3.getBoundingClientRect();
    const p4Position = point4.getBoundingClientRect();
    const p5Position = point5.getBoundingClientRect();

    if (
        position.right > p1Position.left &&
        position.left < p1Position.right &&
        position.bottom > p1Position.top &&
        position.top < p1Position.bottom
    ) {}
```

```
    if (
        position.right > p2Position.left &&
        position.left < p2Position.right &&
        position.bottom > p2Position.top &&
        position.top < p2Position.bottom
    ) {}

    if (
        position.right > p3Position.left &&
        position.left < p3Position.right &&
        position.bottom > p3Position.top &&
        position.top < p3Position.bottom
    ) {}

    if (
        position.right > p4Position.left &&
        position.left < p4Position.right &&
        position.bottom > p4Position.top &&
        position.top < p4Position.bottom
    ) {}
```

```
    if (
        position.right > p5Position.left &&
        position.left < p5Position.right &&
        position.bottom > p5Position.top &&
        position.top < p5Position.bottom
    ) {}
}
```

Hiding the point

- To hide the point upon collision with the player use display: none
  - Making it look as if the player is collecting the points

```
if (
    position.right > p1Position.left &&
    position.left < p1Position.right &&
    position.bottom > p1Position.top &&
    position.top < p1Position.bottom
) {
    point1.style.display = 'none';
}
```

Calling the function

- Remember to call the function

```
function move() {

    pointCheck();

    const position = player.getBoundingClientRect();

    if (downPressed) {
```

Exercise 3

- Download ex3.zip
- Using the new collision detection method add collision between the player and the points
- Hide the points when the player collides with them
- Add your code from Exercise 2
- You should now have collision detection for the walls and points

- Try adding the wall collision for the assignment

Repetition

- The point collision involves a lot of repeated code
- You have to make a new variable and if statement for each point
- It shares 99% of the same code
- All that actually changes is a single number

Repetition pt2

- It would be better if a single function could be written that worked for every point
- For this to work we would need a way of finding all the point elements and looping through each one
- All the points have the point class

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

QuerySelectorAll()

- We can use querySelectorAll('.point') to find all the point elements

```
const points = document.querySelectorAll('.point');
```

- All five point elements are in the points variable
- Using the points variable it is possible to loop over each variable inside of points

For loops

- All programming languages provide a way to create a counter
- This is done using a loop
- In JavaScript you can create a loop like this

```
for(let i = 0; i < 10; i++) {
    console.log(i);
}
```

For loop

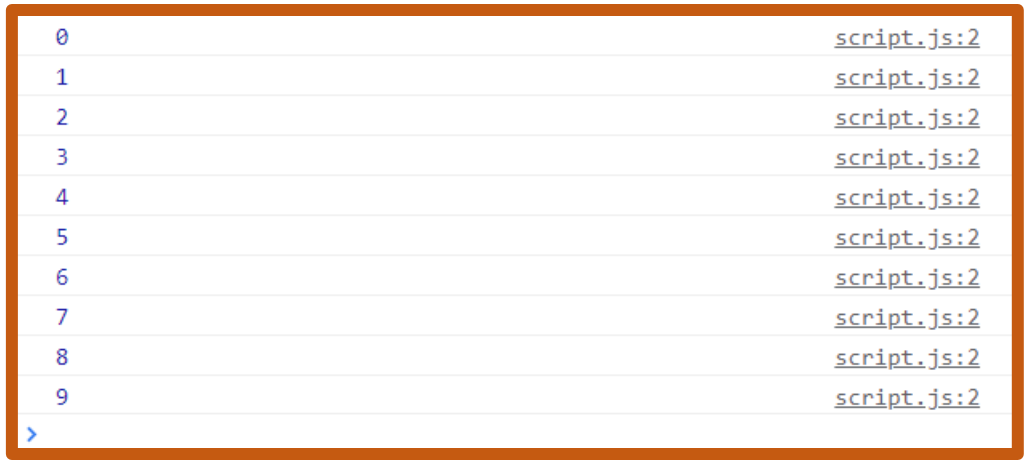
- A loop has 3 parts
  - A starting value
  - A condition
  - A counter

```
for(
        let i = 0; /* Starting value */
        i < 10; /* Condition */
        i++ /* Counter */
    )
{
    console.log(i);
}
```

For loop - starting value

- Changing the starting value affects the first value in the loop

```
for (let i = 0; i < 10; i++) {
  console.log(i);
}
```




```
for (let i = 5; i < 10; i++) {
  console.log(i);
}
```

For loop - condition

- Changing the condition affects when the loop stops
  - How many times to loop the code

```
for (let i = 0; i < 10; i++) {
  console.log(i);
}
```

```
0 script.js:2
1 script.js:2
2 script.js:2
3 script.js:2
4 script.js:2
5 script.js:2
6 script.js:2
7 script.js:2
8 script.js:2
9 script.js:2
>
```

```
for (let i = 0; i < 5; i++) {
  console.log(i);
}
```

```
0 script.js:4
1 script.js:4
2 script.js:4
3 script.js:4
4 script.js:4
>
```

For loop - counter

- Changing the counter changes how much is incremented each time (++ means add 1)

```
for (let i = 0; i < 10; i++) {
  console.log(i);
}
```

```
0 script.js:2
1 script.js:2
2 script.js:2
3 script.js:2
4 script.js:2
5 script.js:2
6 script.js:2
7 script.js:2
8 script.js:2
9 script.js:2
>
```

```
for (let i = 0; i < 10; i = i + 2) {
  console.log(i);
}
```

```
0 script.js:2
2 script.js:2
4 script.js:2
6 script.js:2
8 script.js:2
>
```

Loops and elements

- It's possible to use a variable in place of the number when selecting an element by an index
  - Selecting the third point element

```
const points = document.querySelectorAll('.point');

let num = 3;

points[num].style.display = 'none';
```

Loops and elements pt2

- By combining a loop with querySelectorAll its possible to apply the same thing to each element that was matched

```
const points = document.querySelectorAll('.point');

for(let i = 0; i < 10; i++) {
    let pointPosition = points[i].getBoundingClientRect();
}
```

- You can create the position variable for each of the points

Loops and elements pt2

- This is a lot shorter than creating each one manually

```
const points = document.querySelectorAll('.point');

for(let i = 0; i < 10; i++) {
    let pointPosition = points[i].getBoundingClientRect();
}
```

```
const point1 = document.querySelectorAll('.point')[0];
const point2 = document.querySelectorAll('.point')[1];
const point3 = document.querySelectorAll('.point')[2];
const point4 = document.querySelectorAll('.point')[3];
const point5 = document.querySelectorAll('.point')[4];

const p1Position = point1.getBoundingClientRect();
const p2Position = point2.getBoundingClientRect();
const p3Position = point3.getBoundingClientRect();
const p4Position = point4.getBoundingClientRect();
const p5Position = point5.getBoundingClientRect();
```

Error

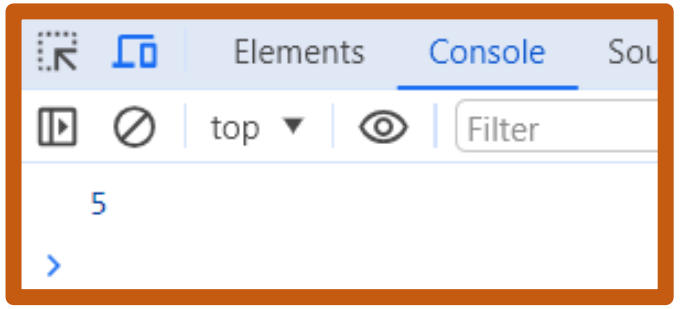
- Running this code could cause an error
- The number of point elements may not be 10
- If there were fewer than 10 point elements this would cause an error because the element does not exist
- There are only 5 points not 10

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

.length

- It's possible to find out how many elements were retrieved from querySelectorAll using the code
  - elements.length
- elements.length stores the number of elements that were retrieved from the page

```
const points = document.querySelectorAll('.point');
console.log(points.length);
```



Amending the loop

- Adding points.length to the condition will remove the error from the code as the number of loops will not surpass the number of point elements

```
const points = document.querySelectorAll('.point');

for(let i = 0; i < points.length; i++) {
    let pointPosition = points[i].getBoundingClientRect();
}
```

Adding the collision back

- By adding the collision back we have now reduced the amount of code
  - The function now works with any number of point elements as well

```
for (let i = 0; i < points.length; i++) {
    let pointPosition = points[i].getBoundingClientRect();

    if (
        position.right > pointPosition.left &&
        position.left < pointPosition.right &&
        position.bottom > pointPosition.top &&
        position.top < pointPosition.bottom
    ) {
        points[i].style.display = 'none';
    }
}
```

```
function pointCheck() {
    const position = player.getBoundingClientRect();

    const point1 = document.querySelectorAll('.point')[0];
    const point2 = document.querySelectorAll('.point')[1];
    const point3 = document.querySelectorAll('.point')[2];
    const point4 = document.querySelectorAll('.point')[3];
    const point5 = document.querySelectorAll('.point')[4];

    const p1Position = point1.getBoundingClientRect();
    const p2Position = point2.getBoundingClientRect();
    const p3Position = point3.getBoundingClientRect();
    const p4Position = point4.getBoundingClientRect();
    const p5Position = point5.getBoundingClientRect();

    if (
        position.right > p1Position.left &&
        position.left < p1Position.right &&
        position.bottom > p1Position.top &&
        position.top < p1Position.bottom
    ) {
        point1.style.display = 'none';
    }

    if (
        position.right > p2Position.left &&
        position.left < p2Position.right &&
        position.bottom > p2Position.top &&
        position.top < p2Position.bottom
    ) {
        point2.style.display = 'none';
    }

    if (
        position.right > p3Position.left &&
        position.left < p3Position.right &&
        position.bottom > p3Position.top &&
        position.top < p3Position.bottom
    ) {
        point3.style.display = 'none';
    }

    if (
        position.right > p4Position.left &&
        position.left < p4Position.right &&
        position.bottom > p4Position.top &&
        position.top < p4Position.bottom
    ) {
        point4.style.display = 'none';
    }

    if (
        position.right > p5Position.left &&
        position.left < p5Position.right &&
        position.bottom > p5Position.top &&
        position.top < p5Position.bottom
    ) {
        point5.style.display = 'none';
    }
}
```

Before

```
function pointCheck() {
    const position = player.getBoundingClientRect();
    const points = document.querySelectorAll('.point');

    for (let i = 0; i < 5; i++) {
        let pointPosition = points[i].getBoundingClientRect();

        if (
            position.right > pointPosition.left &&
            position.left < pointPosition.right &&
            position.bottom > pointPosition.top &&
            position.top < pointPosition.bottom
        ) {
            points[i].style.display = 'none';
        }
    }
}
```

After

Exercise 4

- Amend the previous exercise to be more efficient
- Utilise the for loop from the previous slide and loop through all the points

- Try adding more points to the game to see if the collision still works

- **Hint**: Remember .length

Character customizer

- The code in the exercise zip includes a character colour selector
- Making this functional will require a click event on each of the

```
<p>Select Colour</p>
<ul class="colours">
    <li id="#ff0000"></li>
    <li id="#0000ff"></li>
    <li id="#ffff00"></li>
    <li id="#008000"></li>
    <li id="#8b008b"></li>
    <li id="#ffa500"></li>
    <li id="#000000"></li>
    <li id="#a52a2a"></li>
    <li id="#ffc0cb"></li>
    <li id="#ffffff"></li>
</ul>
```

Repetition again

- We need to manually find all of the list items and add a click event to them

```
const color1 = document.querySelectorAll('li')[0];
const color2 = document.querySelectorAll('li')[1];
const color3 = document.querySelectorAll('li')[2];
const color4 = document.querySelectorAll('li')[3];
const color5 = document.querySelectorAll('li')[4];
const color6 = document.querySelectorAll('li')[5];
const color7 = document.querySelectorAll('li')[6];
const color8 = document.querySelectorAll('li')[7];
const color9 = document.querySelectorAll('li')[8];
const color10 = document.querySelectorAll('li')[9];
```

```
color1.addEventListener('click', setColor1);
color2.addEventListener('click', setColor2);
color3.addEventListener('click', setColor3);
color4.addEventListener('click', setColor4);
color5.addEventListener('click', setColor5);
color6.addEventListener('click', setColor6);
color7.addEventListener('click', setColor7);
color8.addEventListener('click', setColor8);
color9.addEventListener('click', setColor9);
color10.addEventListener('click', setColor10);
```

Repetition again pt2

- For this to work we also need to create 10 separate functions for each of the colours

```
function setColor1() { player.style.backgroundColor = '#ff0000'; }
function setColor2() { player.style.backgroundColor = '#0000ff'; }
function setColor3() { player.style.backgroundColor = '#ffff00'; }
function setColor4() { player.style.backgroundColor = '#008000'; }
function setColor5() { player.style.backgroundColor = '#8b008b'; }
function setColor6() { player.style.backgroundColor = '#ffa500'; }
function setColor7() { player.style.backgroundColor = '#000000'; }
function setColor8() { player.style.backgroundColor = '#a52a2a'; }
function setColor9() { player.style.backgroundColor = '#ffc0cb'; }
function setColor10() { player.style.backgroundColor = '#ffffff'; }
```

Repetition pt3

- This solution involves a lot of repeated code
- You have to make a function for each click event
- Each function shares 99% of the same code

- A good rule in programming is if you find yourself doing the same thing over and over again you are probably doing it wrong

- How can we fix it?

Loop the click events

- Using a for loop we could reduce a lot of the duplication

```
const color1 = document.querySelectorAll('li')[0];
const color2 = document.querySelectorAll('li')[1];
const color3 = document.querySelectorAll('li')[2];
const color4 = document.querySelectorAll('li')[3];
const color5 = document.querySelectorAll('li')[4];
const color6 = document.querySelectorAll('li')[5];
const color7 = document.querySelectorAll('li')[6];
const color8 = document.querySelectorAll('li')[7];
const color9 = document.querySelectorAll('li')[8];
const color10 = document.querySelectorAll('li')[9];
color1.addEventListener('click', setColor1);
color2.addEventListener('click', setColor2);
color3.addEventListener('click', setColor3);
color4.addEventListener('click', setColor4);
color5.addEventListener('click', setColor5);
color6.addEventListener('click', setColor6);
color7.addEventListener('click', setColor7);
color8.addEventListener('click', setColor8);
color9.addEventListener('click', setColor9);
color10.addEventListener('click', setColor10);
```

Before

```
const colours = document.querySelectorAll('li');

for(let i = 0; i < 10; i++) {
    colours[i].addEventListener('click', setColor1);
}
```

After

Problem

- We are now adding the click event automatically to each of the <li> but it's the setColor1() function each time
- It would be better if a single function could be written that worked for every colour

```
const colours = document.querySelectorAll('li');

for(let i = 0; i < 10; i++) {
    colours[i].addEventListener('click', setColor);
}

function setColor() {
    player.style.backgroundColor = '';
}
```

Finding the colour

- This will work, each time any of the colour <li> is clicked on
- This isn't quite what we want, we need to find the colour value to set the player to.

```
<li id="#ff0000"></li>
<li id="#0000ff"></li>
<li id="#ffff00"></li>
<li id="#008000"></li>
<li id="#8b008b"></li>
<li id="#ffa500"></li>
<li id="#000000"></li>
<li id="#a52a2a"></li>
<li id="#ffc0cb"></li>
<li id="#ffffff"></li>
```

- However, the element's ID contains the hex value we want to set

this

- It is possible to know which element was clicked on inside the event listener
- There is a special variable that always exists when an event listener is fired
- The variable is **this**
- The “this” variable is available inside every event listener function automatically
- And stores a reference to the element that was clicked on

this pt2

- You can read an elements ID using this.id
- This can be done either when an element is found using querySelector() or the **this** reference

```
function setColor() {
    alert(this.id);
}
```

this pt3

- Using **this** our code has now been optimised to a couple of lines

```
const colours = document.querySelectorAll('li');

for(let i = 0; i < 10; i++) {
    colours[i].addEventListener('click', setColor);
}

function setColor() {
    player.style.backgroundColor = this.id;
}
```

Comparison pt2

Before

```
const color1 = document.querySelectorAll('li')[0];
const color2 = document.querySelectorAll('li')[1];
const color3 = document.querySelectorAll('li')[2];
const color4 = document.querySelectorAll('li')[3];
const color5 = document.querySelectorAll('li')[4];
const color6 = document.querySelectorAll('li')[5];
const color7 = document.querySelectorAll('li')[6];
const color8 = document.querySelectorAll('li')[7];
const color9 = document.querySelectorAll('li')[8];
const color10 = document.querySelectorAll('li')[9];
color1.addEventListener('click', setColor1);
color2.addEventListener('click', setColor2);
color3.addEventListener('click', setColor3);
color4.addEventListener('click', setColor4);
color5.addEventListener('click', setColor5);
color6.addEventListener('click', setColor6);
color7.addEventListener('click', setColor7);
color8.addEventListener('click', setColor8);
color9.addEventListener('click', setColor9);
color10.addEventListener('click', setColor10);
function setColor1() { player.style.backgroundColor = '#ff0000'; }
function setColor2() { player.style.backgroundColor = '#0000ff'; }
function setColor3() { player.style.backgroundColor = '#ffff00'; }
function setColor4() { player.style.backgroundColor = '#008000'; }
function setColor5() { player.style.backgroundColor = '#8b008b'; }
function setColor6() { player.style.backgroundColor = '#ffa500'; }
function setColor7() { player.style.backgroundColor = '#000000'; }
function setColor8() { player.style.backgroundColor = '#a52a2a'; }
function setColor9() { player.style.backgroundColor = '#ffc0cb'; }
function setColor10() { player.style.backgroundColor = '#ffffff'; }
```

After

```
const colours = document.querySelectorAll('li');

for(let i = 0; i < 10; i++) {
    colours[i].addEventListener('click', setColor);
}

function setColor() {
    player.style.backgroundColor = this.id;
}
```

Other types of loops

- JavaScript offers a few different loops we can use in our code
  - For loops
  - While Loops
  - Do-While loops
  - For-Of loop

- Each serving a similar purpose of looping a certain number of times, the main difference is how it is achieved

While loop

- A while loop executes a block of code as long as a specified condition evaluates is true.
- It's used when the number of loops is not known beforehand
  - Useful for random numbers (next week)

```
const colours = document.querySelectorAll('li');

let i = 0;
while (i < 10) {
    colours[i].addEventListener('click', setColor);
    i++;
}

function setColor() {
    player.style.backgroundColor = this.id;
}
```

Do-While loop

- Similar to a while loop, but it always executes the block of code at least once before checking the condition

```
const colours = document.querySelectorAll('li');

let i = 0;
do {
    colours[i].addEventListener('click', setColor);
    i++;
} while (i < 10);

function setColor() {
    player.style.backgroundColor = this.id;
}
```

For-Of loop

- A for-of loop assigns each iteration of the data to a variable
- It provides a simpler way of looping without using an index position
  - You can use a variable like colour instead of colours[i]

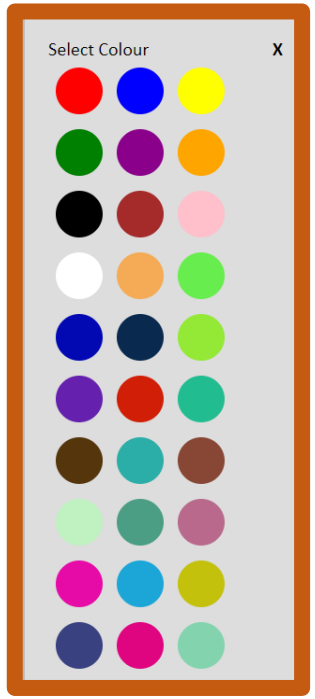
```
const colours = document.querySelectorAll('li');

for(const colour of colours) {
    colour.addEventListener('click', setColor);
}

function setColor() {
    player.style.backgroundColor = this.id;
}
```

Exercise 5

- Add the functionality for the character colour customizer
- Use a for loop and "**this**", you should only need one function for all the colours

- Try adding some more colours to the HTML
  - Does your code still work?

- Create a click event for the "X"
- When the user clicks the X hide aside (character customizer)
- If the user clicks the player show aside



Useful links

- https://developer.mozilla.org/en-US/docs/Web/API/Document/elementFromPoint
- https://developer.mozilla.org/en-US/docs/Web/API/Document/querySelector
- https://developer.mozilla.org/en-US/docs/Web/API/Document/querySelectorAll
- https://developer.mozilla.org/en-US/docs/Web/JavaScript/Guide/Expressions_and_operators
- https://developer.mozilla.org/en-US/docs/Web/JavaScript/Guide/Loops_and_iteration
- https://developer.mozilla.org/en-US/docs/Web/JavaScript/Guide/Functions