

The background features a decorative pattern of hexagons. The hexagons on the right side contain blurred images of code snippets, likely from a web development context, with visible text such as 'coun', 'post', 'has_post', and 'col-xs'. The hexagons on the left are solid light grey.

CSY1063

Web Development

Week 9

Chris.Rafferty@northampton.ac.uk



Learning Objectives

- This week we will be covering
 - JavaScript events
 - CSS properties & classes
 - Strings to numbers
 - Intervals
 - Moving elements & detecting key presses (keydown, keyup)
 - If statements (else & switch)
 - Global variables
 - Finding elements by class name



Event listeners

- addEventListener is a very useful function
- It allows you to run a function when a specific event occurs
 - When this happens run that function
- A few useful events we can use are
 - click
 - mouseover/mouseout
 - keyboard events (more later)
 - DOMContentLoaded (like defer)
 - resize
 - scroll

Click events

- In last week's exercise, if you click anywhere on the document, the contents of both elements will be updated
- It's possible to assign a click event to a particular element

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

- You can call `element.addEventListener` to add an event to a specific element
- This works exactly the same way as `document.addEventListener` however it will only call your function when that particular element is clicked on

Click event pt2

- Add a click event to the h1 element so that function is run when the h1 is clicked on
- The click updates the contents of the h1 element

```
let heading = document.querySelector('#pageHeading');  
  
function changeHeading() {  
    heading.firstChild.nodeValue = 'Heading has changed!!!';  
}  
  
heading.addEventListener('click', changeHeading);
```

Click events pt 2

- We can add a click event to both the h1 and p tags that change the contents only when that specific tag is clicked on

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

function changeHeading() {
    heading.firstChild.nodeValue = 'Heading has changed!!!';
}

heading.addEventListener('click', changeHeading);

let paragraph = document.querySelector('#pageParagraph');

function changeParagraph() {
    paragraph.firstChild.nodeValue = 'Paragraph has changed!!!'
}

paragraph.addEventListener('click', changeParagraph);
```

Setting CSS properties

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

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

Setting CSS Properties pt2

- To set the width and height of an element

```
element.style.width = '50px';  
element.style.height = '50px';
```

- Or text colour:

```
element.style.color = 'red';
```

- Any CSS property can be used and set
- However, the names are slightly different for some properties

CSS Properties

- Some CSS properties contain hyphens, e.g.
 - background-color
 - border-radius
 - font-family
- In JavaScript these are written by removing the hyphen and making the first letter of the second word uppercase

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

CSS properties comparison

- The value is placed in quotes as it as a string, however the outcome is the same

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

- Will have the same effect as the CSS

```
#paragraph {  
    background-color: green;  
    border-radius: 50px;  
    font-family: Verdana, Helvetica, Sans-serif;  
    height: 50px;  
}
```



CSS in JavaScript

- The JavaScript code is longer but it can be run at any time
- CSS is applied once, when the page loads
- JavaScript can be used to change the look of an element after the page has loaded
- Usually this is useful when an event occurs



Other properties

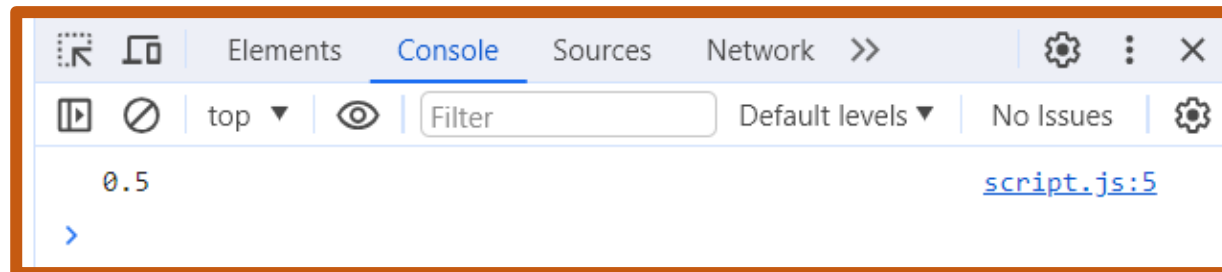
- There's a CSS property called opacity
 - week 5 animation lecture
- This acts as transparency and takes a value from 0 – 1. 1 is completely opaque
- 0 is completely transparent
- 0.5 is half transparent

Reading from properties

- You can read a CSS property using the same syntax provided it has been set via JavaScript originally (this doesn't work if the original property was set via a CSS stylesheet)

```
const circle = document.querySelector('#circle');
circle.style.opacity = 0.5;

let opacity = circle.style.opacity
console.log(opacity);
```



Reading from properties pt2

- Once the value is stored in a variable it can be modified and the original value changed

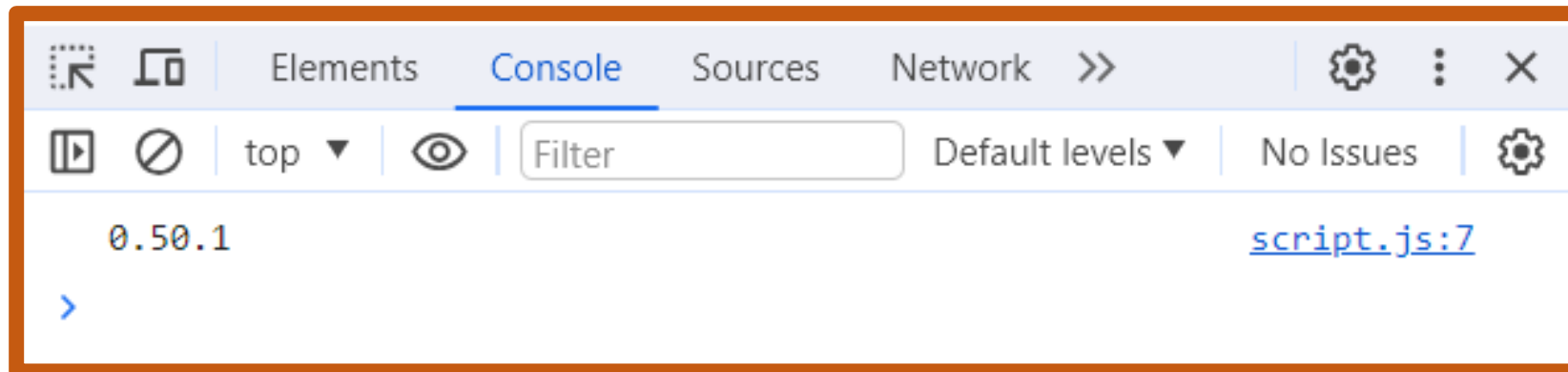
```
const circle = document.querySelector('#circle');
circle.style.opacity = 0.5;
let circleOpacity = circle.style.opacity;

circle.style.opacity = circleOpacity + 0.1;
console.log(circleOpacity + 0.1);
```

- However, this doesn't quite work!
- This is because CSS values are stored as text
- You need to convert the text into a number before doing any calculations

Reading from properties pt3

- The code will treat both values as strings and the opacity value will be the joined strings something like '0.50.1' rather than adding them to get 0.6
 - '0.5' + '0.1' = '0.50.1'
 - 0.5 + 0.1 = 0.6



Converting strings to numbers

- There are two functions for converting strings to numbers
- `parseInt()` - converts a string into the closest whole number
 - 1, 2, 3
- `parseFloat()` - converts a string into the closest decimal number
 - 1.1, 1.2, 1.3

```
const circle = document.querySelector('#circle');  
circle.style.opacity = 0;  
let circleOpacity = parseFloat(circle.style.opacity);  
  
circle.style.opacity = circleOpacity + 0.1;
```




Intervals

- Along with events that are triggered by the user doing something
 - click
 - mouseenter
 - mouseleave
 - keyup
 - Keydown
- There are also other events that are based on timers

setInterval

- You can get the browser to repeatedly run a function based on a time interval
 - run the function every 10 seconds
 - Every second
 - Every 100th of a second
 - Etc
- This is done with setInterval

```
setInterval(functionName, interval);
```

setInterval pt2

- This will run the function myInterval every second
- Intervals are measured in 1000ths of a second
 - 1000 = 1 second
 - 5000 = 5 seconds
 - 10000 = 10 seconds

```
function myInterval() {  
    console.log('myInterval called');  
}  
  
setInterval(myInterval, 1000);
```

Cancelling timers

- You can store a timer inside a variable
- Once a timer has been stored in a variable it can be stopped
- This is done using clearInterval

```
function myInterval() {  
    console.log('myInterval called');  
}  
  
let timer = setInterval(myInterval, 1000);  
  
clearInterval(timer);
```

setTimeout

- You can use setTimeout to call a function after a delay
 - The function will only be called once unlike setInterval which will continue until its cancelled

```
function myInterval() {  
    console.log('myInterval called');  
}  
  
setTimeout(myInterval, 3000);
```



Moving elements

- There are many different ways of positioning an element:
 - Margin properties
 - Top/Left/Right/Bottom properties
 - By positioning int inside another element with a margin
- Because of this, unless you know exactly how the element has been positioned it's difficult to read the position of an element from a CSS property as you do not know whether to read margin, padding, etc

getBoundingClientRect()

- 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
- This means you don't have to work out which CSS properties have been used to position it
- 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;
```

Storing an elements position

- This will store the position of the element on the page in the variables
- This 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';
```


Moving elements

- By using the key down function we can move an element when a key has been pressed

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

function moveLeft() {
  let position = circle.getBoundingClientRect();
  let positionLeft = position.left;

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

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



Detecting which key was pressed

- When you press any key on the keyboard, the button will move left
- It would be better if only a specific key (for example the left arrow key) moved the circle on the page
- To do this you need to work out which key was pressed

Event parameter

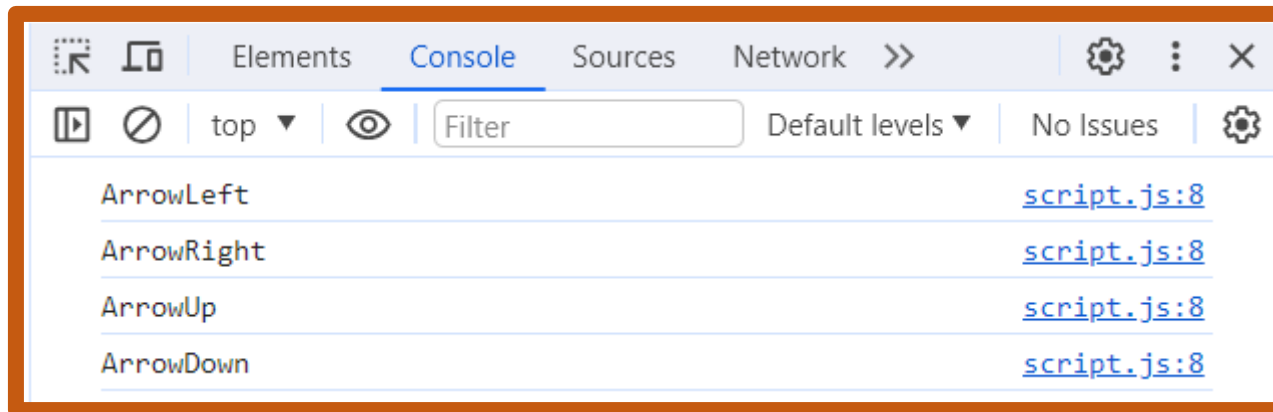
- This is done in the event listener
- Each time an event is triggered by the browser (clicks, mouse movement, key presses, etc) you can ask the browser to give you some information about what triggered the event

```
function moveLeft(event) {  
}
```

event.key

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

```
function moveLeft(event) {  
    console.log(event.key)  
}  
  
document.addEventListener('keydown', moveLeft);
```





Conditional statement

- We still need a way to programme if the left key is pressed move left
- In most languages we can use an if statement to see if something is true, this could be used to check if the left key was pressed, if so run the code in the moveLeft function

If statements

- An if statement allows you to test so see if two values are equal
- If statements take the format

```
if (condition) {  
    //Any code here  
    //Will only run  
    //If the condition  
    //is met  
}
```

If statements pt2

- If statements are often used to check if a variable is equal to a specific value
 - If the condition evaluates to true, then the code between the braces will run
 - If the condition is not met, the code will not run

```
if (variablename == value) {  
    //Any code here  
    //Will only run  
    //If the condition  
    //is met  
}
```



If statements pt3

- If statements allow you to run blocks of code in certain circumstances
- This is very useful if you only want some code to run some of the time
- We could use if statements for each of the arrow keys, move left when the left arrow key is pressed

If statements pt4

- To check to see if the variable event.key is equal to the ArrowLeft you can use the code

```
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);
```

== not =

- Note: This is == (Two equals signs!) not =
- In JavaScript == is used for comparison
- = is only used to set a variable to a specific value

```
if (event.key == 'ArrowLeft') {}
```

Multiple if statements

- One if statement can follow another
- This will let you perform a different action for each key that's pressed
 - ArrowLeft, ArrowRight, ArrowUp & ArrowDown
- It would be better to have a movement function that ran specific move functions when those keys are pressed

```
function move(event) {  
  if(event.key == 'ArrowLeft') {  
    moveLeft();  
  }  
  if(event.key == 'ArrowRight') {  
    moveRight();  
  }  
  //ect  
}  
  
document.addEventListener('keydown', move);
```

else

- You can use else to run a piece of code if the if statements condition is not met

```
function move(event) {  
    if (event.key == 'ArrowLeft') {  
        //If the left arrow is pressed do this  
    }  
    else {  
        //Will run if the left arrow is not pressed  
    }  
}
```

else if

- You can use else if to check for multiple conditions in a sequence, they allow us to specify additional considerations

```
function move(event) {  
  if (event.key == 'ArrowLeft') {  
    //If the left arrow is pressed do this  
  }  
  else if (event.key == 'ArrowRight') {  
    //If the left arrow is not pressed but the right arrow is  
  }  
  else if (event.key == 'ArrowUp') {  
    //If the left or right arrows are not pressed but the up arrow is  
  }  
  else if (event.key == 'ArrowDown') {  
    //If the left or right or up arrows are not pressed but the down arrow is  
  }  
}
```

else if and else

- You can use else with else if for when none of the conditions are met

```
function move(event) {  
    if (event.key == 'ArrowLeft') {  
        //If the left arrow is pressed do this  
    }  
    else if (event.key == 'ArrowRight') {  
        //If the left arrow is not pressed but the right arrow is  
    }  
    else if (event.key == 'ArrowUp') {  
        //If the left or right arrows are not pressed but the up arrow is  
    }  
    else if (event.key == 'ArrowDown') {  
        //If the left or right or up arrows are not pressed but the down arrow is  
    }  
    else {  
        //If left, right, up & down arrows not pressed, do this  
    }  
}
```

Switch statements

- If you have multiple else if statements it can get difficult to read and manage, good alternative is using a switch statement instead
- It's very similar to the else if except we use case for each condition
- default is the same as else, the code will run if no other conditions are met (value1, value2 or value3)

```
switch (condition) {  
    case value1:  
        //code  
        break;  
    case value2:  
        //code  
        break;  
    case value3:  
        //code  
        break;  
    default:  
        //code  
}
```

Switch example

```
function move(event) {  
    switch (event.key) {  
        case 'ArrowLeft':  
            //If the left arrow is pressed do this  
            break;  
        case 'ArrowRight':  
            //If the left arrow is not pressed but the right arrow is  
            break;  
        case 'ArrowUp':  
            //If the left or right arrows are not pressed but the up arrow is  
            break;  
        case 'ArrowDown':  
            //If the left or right or up arrows are not pressed but the down arrow is  
            break;  
        default:  
            //If left, right, up & down arrows not pressed, do this  
    }  
}
```




Variable Scope

- 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

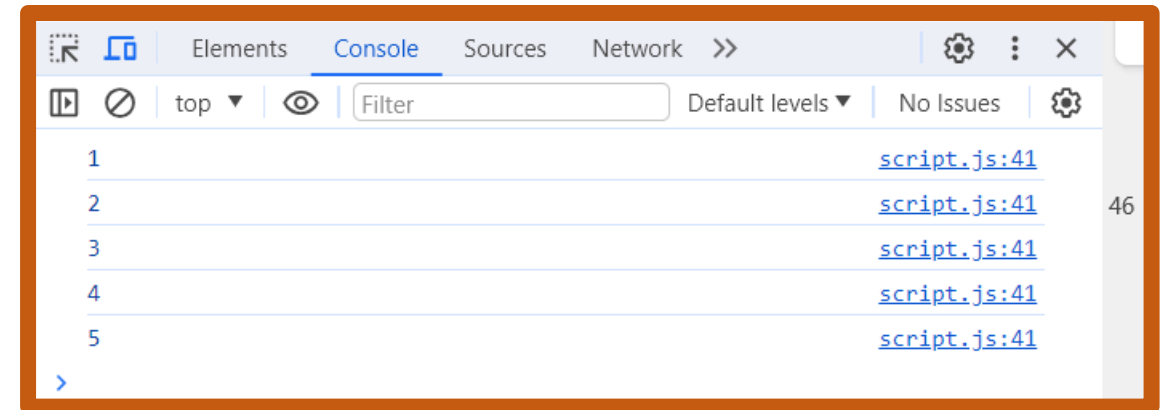
Global variable

- The variable myVariable is declared outside the function
- Each time myClickEvent is called the value of the variable is retained from last time
 - This is called a “global variable”

```
let myVariable = 0;

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

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



Global variable pt2

- Note: When global variables are used they have already been defined so don't need the variable keyword
 - let
 - const

```
let myVariable = 0;

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

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

- If the variable was declared inside the function this would not happen

Exercise

- Download exercise1.zip and extract it
- Add a `<script>` tag to the page so you can run some JavaScript
- Add a click event to the element with the ID ``circle`` so that when the circle is clicked an alert box appears and says “The circle was pressed”
- Make it so when the circle is clicked on, it moves 10px to the left
- Use a keydown event and make it so when a key is pressed, move the circle to the left
- Each time you press a key it should move further and further left, moving 10px each time



Exercise 2

- Add the if statement so that the circle moves left when the left arrow key is pressed
- Add an if statement and relevant code to allow the
 - Up arrow to move the circle up 10px (ArrowUp)
 - Down arrow to move the circle down 10px (ArrowDown)
 - Right arrow to move the circle right 10px (ArrowRight)
- Hint: One if statement can directly follow another
- Hint: You can only style the left and top in CSS!



Exercise 3

- Instead of moving the circle 10px, use an interval when the relevant key is pressed and have the circle slide off the screen in the direction of the arrow pressed
- The animation should look smooth
- Hint: Each arrow should trigger a different timer and each timer will need its own function, you'll end up with at least 5 functions!
 - One for the keydown event and one for each of the four direction timers
 - Up, Down, Left & Right
 - Remember to clear the timer otherwise the circle will get faster with each key press

Exercise 3 example

- You can cancel the previous timer by setting it as a global variable

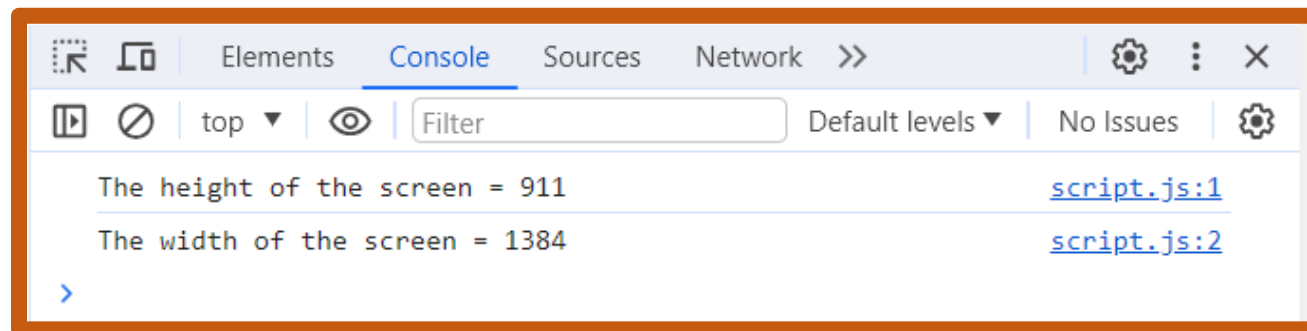
```
const circle = document.querySelector('#circle');
let timer = 0;

function circleMove(event) {
  clearInterval(timer);
  if (event.key == 'ArrowLeft') {
    timer = setInterval(moveLeft, 10);
  }
  if (event.key == 'ArrowRight') {
    timer = setInterval(moveRight, 10);
  }
  if (event.key == 'ArrowUp') {
    timer = setInterval(moveUp, 10);
  }
  if (event.key == 'ArrowDown') {
    timer = setInterval(moveDown, 10);
  }
}
//Rest of code (move functions)
```

Screen size

- You can measure the width and height of the browser window using:
 - `window.innerWidth`
 - `window.innerHeight`

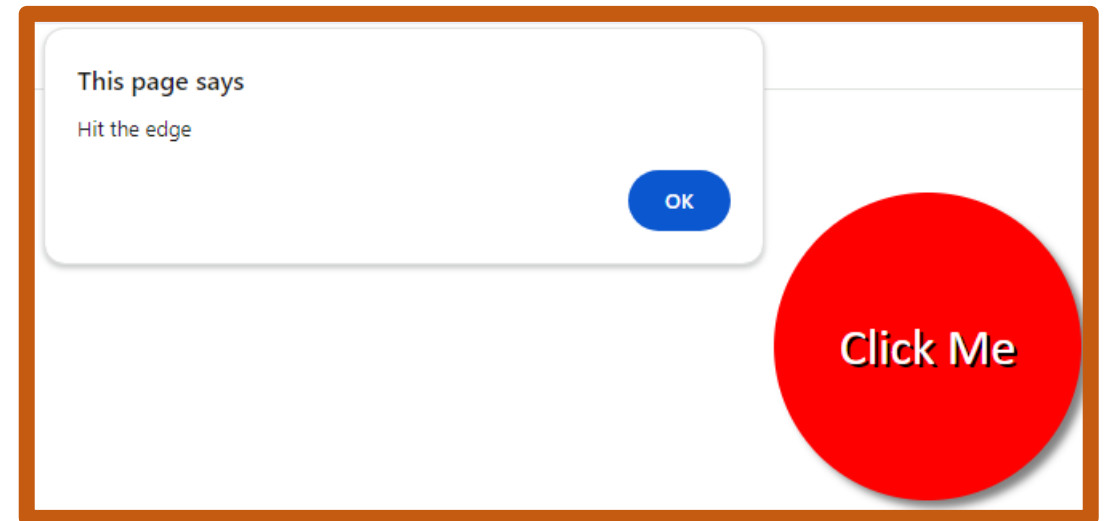
```
console.log('The height of the screen = ' + window.innerHeight);  
console.log('The width of the screen = ' + window.innerWidth);
```



Screen collision

- You can compare an element's position (left, top, ect) to the window height/width to determine whether it is at the bottom of the window

```
function moveRight() {  
  let position = circle.getBoundingClientRect();  
  let circleLeft = position.left;  
  let circleRight = position.right;  
  
  if (circleRight < window.innerWidth) {  
    circle.style.left = (circleLeft + 1) + 'px';  
  }  
  else {  
    alert('Hit the edge');  
  }  
}
```





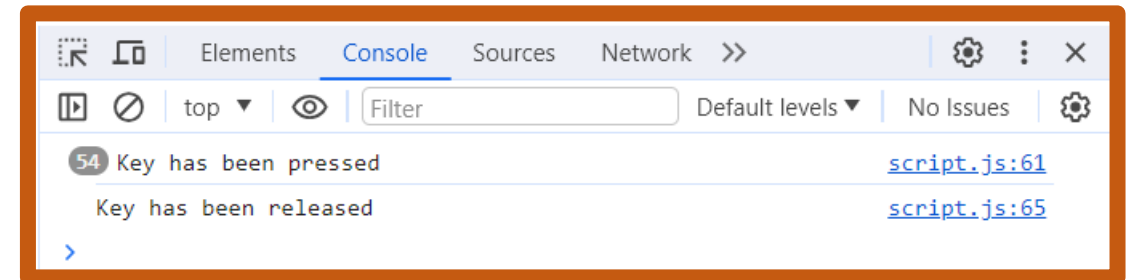
keyup

- You used keydown to detect when a key was pressed and move the circle in the corresponding direction
- Along with keydown there is also keyup which determines whether a key has been released
- Keyup also allows you to detect which key was pressed

Combining keyup and keydown

- 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);
```



Keyup and keydown

- This will allow you to set the timer when the key is pressed and clear it when the key is released

```
let timer = 0;

function myKeyDown(event) {
  if (event.key == 'ArrowLeft') {
    timer = setInterval(moveLeft, 10);
  }
  if (event.key == 'ArrowRight') {
    timer = setInterval(moveRight, 10);
  }
}

function myKeyUp() {
  clearInterval(timer);
}

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



Keyup and keydown pt2

- This will not have quite the effect you expect!
- The code looks like it should start the interval when the key is pressed
- And clear the interval when the key is released
- However, you'll notice that as you hold the key down the circle moves faster and faster
- This is because multiple intervals are being created



Key presses

- This happens because when you hold a key down, the keydown event gets triggered over and over
- You can try this yourself by clicking into a text field and holding a letter key down. The letter doesn't just appear once, it will be repeated



Tracking key presses

- Instead, to do this we can track whether a key is being held down or not using a global variable
- JavaScript supports Boolean variables which can store true or false
- A global Boolean variable can be used to track whether or not a key is being held down

Tracking key presses pt2

- Create variables to store whether a key is being pressed or not.
- Start it as “not being pressed” (false)

```
let leftPressed = false;  
let rightPressed = false;  
let upPressed = false;  
let downPressed = false;
```

- Using global variables we can access these in any of our functions

Tracking key presses pt3

- When the key is pressed, set it to true (keydown)
- When the key is released, set it back to false (keyup)

```
let leftPressed = false;

function myKeyDown(event) {
  if (event.key == 'ArrowLeft') {
    leftPressed = true;
  }
}

function myKeyUp(event) {
  if (event.key == 'ArrowLeft') {
    leftPressed = false;
  }
}

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



Adding the movement

- Once you have a Boolean variable that stores the key presses you can check to see whether the key is pressed in a timer that is always running (moveInterval)
- The move timer will always be checking to see if any of the global variables (leftPressed, rightPressed etc) are true, if they are true the circle will move in that direction

Adding the movement pt2

```
let leftPressed = false;

function myKeyDown(event) {
  if (event.key == 'ArrowLeft') {
    leftPressed = true;
  }
}

function myKeyUp(event) {
  if (event.key == 'ArrowLeft') {
    leftPressed = false;
  }
}

function moveInterval() {
  if (leftPressed == true) {
    let position = circle.getBoundingClientRect();
    let circleLeft = position.left;
    circle.style.left = (circleLeft - 1) + 'px';
  }
}

setInterval(moveInterval, 10);

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

Assignment movement

- This might look familiar it's how your assignments movement has been set

```
let upPressed = false;
let downPressed = false;
let leftPressed = false;
let rightPressed = false;
```

```
function keyDown(event) {
  if (event.key === 'ArrowUp') {
    upPressed = true;
  } else if (event.key === 'ArrowDown') {
    downPressed = true;
  } else if (event.key === 'ArrowLeft') {
    leftPressed = true;
  } else if (event.key === 'ArrowRight') {
    rightPressed = true;
  }
}
```

```
//Player movement
function keyUp(event) {
  if (event.key === 'ArrowUp') {
    upPressed = false;
  } else if (event.key === 'ArrowDown') {
    downPressed = false;
  } else if (event.key === 'ArrowLeft') {
    leftPressed = false;
  } else if (event.key === 'ArrowRight') {
    rightPressed = false;
  }
}
```

- In the as2.zip

Setting CSS classes using JavaScript

- You can set a CSS class on a HTML Element using JavaScript
- This is similar to setting a style, however it uses the syntax

```
const circle = document.querySelector('#circle');  
circle.className = 'nameOfClass';
```

Setting CSS classes using JavaScript pt2

- This allows you to define a set of CSS rules in the stylesheet that can be applied with a single line of JavaScript

```
.semiTransparentAndBlue {  
  background-color: blue;  
  opacity: 0.5;  
}
```

```
const circle = document.querySelector('#circle');  
circle.classList.add('semiTransparentAndBlue');
```

Setting CSS classes using JavaScript pt3

- You can apply more than one CSS class by adding more than one class to the classList

```
.semiTransparent {  
    opacity: 0.5;  
}  
  
.blue {  
    background-color: blue;  
}
```

```
const circle = document.querySelector('#circle');  
circle.classList.add('semiTransparent');  
circle.classList.add('blue');
```

Removing CSS classes

- You can remove a CSS class from an element using `element.classList.remove()`

```
const circle = document.querySelector('#circle');  
circle.classList.remove('blue');
```


Finding Elements using CSS Classes

- So far to find elements on the page, we have been using the code
 - `document.querySelector('#ID')`

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

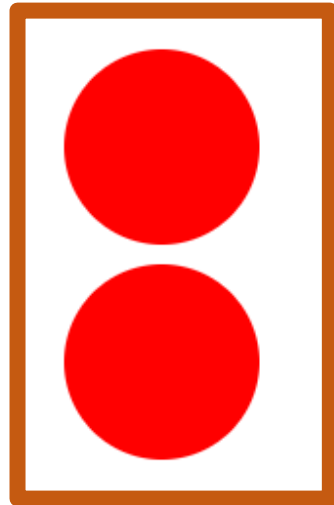
- This finds an element on the page using the HTML ID attribute
- It is also possible to find elements on the page using the CSS class name

Finding Elements using CSS Classes pt2

- There is one major difference between IDs and classes:
 - An ID should be unique. There will only be one element on the page with each ID
 - Classes are not, multiple elements can have the same class
- `document.querySelector('#ID')` will always retrieve a single element from the page
- However, this is not possible with classes because there may be more than one element on the page with the same class

Finding Elements using CSS Classes pt3

- Which element would you expect the following code to find?



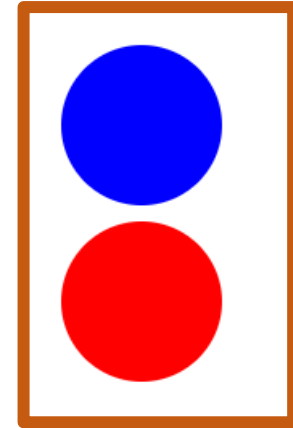
```
<div class="circle">  
  
</div>  
  
<div class="circle">  
  
</div>
```

```
const circle = document.querySelector('.circle');
```

Finding Elements using CSS Classes pt4

- `querySelector()` will only retrieve the first circle element
 - We use a `.` for classes `#` for IDs

```
const circle = document.querySelector('.circle');  
circle.style.backgroundColor = 'blue';
```



querySelectorAll()

- You can use querySelectorAll() to retrieve all the elements of that class (both circle divs)
- 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';
```

Exercise 3 solution

```
const circle = document.querySelector('#circle');
let timer = 0;

function circleMove(event) {
  clearInterval(timer);
  if (event.key == 'ArrowLeft') {
    timer = setInterval(moveLeft, 10);
  }
  else if (event.key == 'ArrowRight') {
    timer = setInterval(moveRight, 10);
  }
  else if (event.key == 'ArrowUp') {
    timer = setInterval(moveUp, 10);
  }
  else if (event.key == 'ArrowDown') {
    timer = setInterval(moveDown, 10);
  }
}
```

```
function moveLeft() {
  let position = circle.getBoundingClientRect();
  let circleLeft = position.left;
  circle.style.left = (circleLeft - 1) + 'px';
}

function moveRight() {
  let position = circle.getBoundingClientRect();
  let circleLeft = position.left;
  circle.style.left = (circleLeft + 1) + 'px';
}

function moveUp() {
  let position = circle.getBoundingClientRect();
  let circleTop = position.top;
  circle.style.top = (circleTop - 1) + 'px';
}


function moveDown() {
  let position = circle.getBoundingClientRect();
  let circleTop = position.top;
  circle.style.top = (circleTop + 1) + 'px';
}

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



Exercise 4

- Finish the previous exercises (ex1 – 3)
- Change the code for exercise 3 so that when the left key is held down, the circle moves left.
 - Once the key is released, it should stop moving
- Add the other directions
 - Right
 - Up
 - Down
- Hint: You should only need one timer that is always running!



Exercise 4 (bonus)

- (Optional) Make the 1-9 keys on the keyboard set the element's speed with 1 being slow and 9 being fastest
- Stop the circle from leaving the screen
- Hint: `window.innerWidth` & `window.innerHeight`

Useful links

- https://developer.mozilla.org/en-US/docs/Learn/JavaScript/Building_blocks/Events
- https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/parseFloat
- https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/parseInt
- <https://developer.mozilla.org/en-US/docs/Web/API/setInterval>
- <https://developer.mozilla.org/en-US/docs/Web/API/Element/getBoundingClientRect>
- https://developer.mozilla.org/en-US/docs/Web/API/UI_Events/Keyboard_event_key_values
- <https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Statements/if...else>
- <https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Statements/switch>
- <https://developer.mozilla.org/en-US/docs/Web/API/Document/querySelectorAll>