CSY1018
Web Development

Tom Butler
thomas.butler@northampton.ac.uk

Topic 3

- Quick recap of last week
- Keyup events and monitoring whether a key is pressed
- Setting CSS classes with Javascript

Setting CSS Properties

- You can set CSS properties on an element using javascript
- Firstly you have to select the element by using document.getElementById
- Once you have a reference to the element in a variable you can change the CSS on it using

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

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';
```

Reading from properties

- You can read a CSS property using the same syntax provided it has been set via javascript originally:

```
var element = document.getElementById('circle');
element.addEventListener('click', myClickEvent);
element.style.opacity = '0.5';

var circleOpacity = element.style.opacity;
console.log(circleOpacity);
```

Prints "0.5" to the console

Reading from properties

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

```
var element = document.getElementById('circle');
var circleOpacity = element.style.opacity;
element.style.opacity = 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

Converting strings to numbers

- There are two functions for converting strings to numbers
  - parseInt() - converts a string into the closest whole number e.g 123
  - parseFloat() - converts a string into the closest decimal number e.g. 1.23

```
var element = document.getElementById('circle');
var circleOpacity = parseFloat(element.style.opacity);
element.style.opacity = circleOpacity + 0.1;
```

Interval

- Along with events that are triggered by the user doing something e.g.
  - click
  - mouseenter
  - mouseleave
  - keyup
  - keydown
- There are also other events that are based on *timers*

Intervals

- You can get the browser to automatically run a function based on a time interval
  - e.g. run the function every 10 seconds
  - Every second
  - Every 100th of a second
  - Etc
- This is done with the code:

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

Intervals

```
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 so 1000 is equal to 1 second

Changing an element's position

- 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

Getting an element's current position

- Javascript provides a simple way of retrieving the 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
- This can be done with:

```
var element = document.getElementById('circle');
var positionTop = element.offsetTop;
var positionLeft = element.offsetLeft;
```

Moving elements

- 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:

```
var element = document.getElementById('circle');
var positionLeft = element.offsetLeft;

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

- This reads the current offsetLeft, adds 10 to it
- Note that 'px' is also added to create the valid CSS unit

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

Detecting which key was pressed

- 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
- This is done using the code:

```
function myKeyDown(event) {

}
```

event.keyCode

- The keyCode for the left arrow key is 37
- (You can find this out by pressing the left arrow key and looking at the number that appears in the console)
- To make it so that only the left arrow key causes the circle to move on the screen we need to make it so that the code that moves the circle only runs when the keyCode is 37.

If statements

- To check to see if the variable event.keyCode is equal to 37 you can use the code:

```
if (event.keyCode == 37) {
  //Any code here
  //Will only run
  //If the condition
  //is met
}
```

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

Cancelling timers

- You can store a timer inside a variable
- This is done using the code:

```
var timer =  setInterval(moveLeft, 10);
```

- Once a timer has been stored in a variable it can be stopped using the code:

```
var timer =  setInterval(moveLeft, 10);
clearInterval(timer);
```

Topic 3

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:

Variable scope

- The variable myVariable is declared outside the function
- Each time myClickEvent is called the value of the variable is retained from last time (If the variable was declared inside the function this would not happen!)
- This is called a "global variable"

Note: When global
Variables are used
They have already been
Defined so don't need the
var keyword

```
var myVariable = 0;

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

function myLoadEvent() {
  document.addEventListener('click', myClickEvent);
}

document.addEventListener('DOMContentLoaded', myLoadEvent);
```

Last Week's Exercise 10

```
var interval = 0;
function myKeyDown(event) {

  clearInterval(interval);

  if (event.keyCode == 37) {
    interval = setInterval(moveLeft, 10);
  }
  if (event.keyCode == 38) {
    interval =setInterval(moveUp, 10);
  }
  if (event.keyCode == 40) {
    interval =setInterval(moveDown, 10);
  }
  if (event.keyCode == 39) {
    interval =setInterval(moveRight, 10);
  }
}

function myLoadEvent() {
  document.addEventListener('keydown', myKeyDown);
}

document.addEventListener('DOMContentLoaded', myLoadEvent);
```

```
function moveUp() {
  var element = document.getElementById('circle');
  var positionTop = element.offsetTop;
  element.style.top = positionTop - 1 + 'px';
}

function moveDown() {
  var element = document.getElementById('circle');
  var positionTop = element.offsetTop;
  element.style.top = positionTop + 1 + 'px';
}

function moveLeft() {
  var element = document.getElementById('circle');
  var positionLeft = element.offsetLeft;
  element.style.left = positionLeft - 1 + 'px';
}

function moveRight() {
  var element = document.getElementById('circle');
  var positionLeft = element.offsetLeft;
  element.style.left = positionLeft + 1 + 'px';
}
```

Exercise 1

- 10 minites
- 1) Run last week's exercise 10 (Download it from GitHub if you don’t have it completed)
- 2) Change it so that if the circle reaches the top or the left of the screen it stops moving
  - Hint: You will need an if statement inside the interval functions
- 3) **Optional** the variables window.innerWidth and window.innerHeight can be used to retrieve the width of the browser window. Stop the circle moving if it reaches the right or bottom of the window
- 3a) **Optional** Prevent all of the circle from leaving the bottom or right of the screen
  - Hint: the circle is 200x200 pixels
- 3b) **Optional** When the circle hits the edge of the screen, make it bounce off so it starts travelling in the opposite direction.
- 3c) **Optional** When the circle has left one side of the screen make it appear at the opposite edge

Exercise 1 - Solution

```
var interval = 0;

function moveUp() {
  var element = document.getElementById('circle');
  var positionTop = element.offsetTop;
  element.style.top = positionTop - 1 + 'px';

  if (positionTop == 0) {
        clearInterval(interval);
        interval = setInterval(moveDown, 10);
  }
}

function myKeyDown(event) {
        //Clear existing intervals
        clearInterval(interval);
        if (event.keyCode == 37) {
                interval = setInterval(moveLeft, 10);
        }
        if (event.keyCode == 38) {
                interval = setInterval(moveUp, 10);
        }
        if (event.keyCode == 40) {
                interval = setInterval(moveDown, 10);
        }
        if (event.keyCode == 39) {
                interval = setInterval(moveRight, 10);
        }
}


function myLoadEvent() {
  document.addEventListener('keydown', myKeyDown);
}

document.addEventListener('DOMContentLoaded', myLoadEvent);
```

```
function moveDown() {
  var element = document.getElementById('circle');
  var positionTop = element.offsetTop;
  element.style.top = positionTop + 1 + 'px';

  if (positionTop+200 == window.innerHeight) {
        clearInterval(interval);
        interval = setInterval(moveUp, 10);
  }
}

function moveLeft() {
  var element = document.getElementById('circle');
  var positionLeft = element.offsetLeft;
  element.style.left = positionLeft - 1 + 'px';

  if (positionLeft == 0) {
        clearInterval(interval);
        interval = setInterval(moveRight, 10);
  }
}

function moveRight() {
  var element = document.getElementById('circle');
  var positionLeft = element.offsetLeft;
  element.style.left = positionLeft + 1 + 'px';

  if (positionLeft == window.innerWidth-200) {
        clearInterval(interval);
        interval = setInterval(moveLeft, 10);
  }
}
```

Javascript Keyup

- Last week you used keydown to detect when a key was pressed and move a shape 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 using event.keyCode
  - The keyCodes are the same for both keyup and keydown events

Combining keyup and keydown

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

```
function myKeyUp(event) {
        if (event.keyCode == 37) {
                console.log('Left arrow has been released');
        }
}

function myKeyDown(event) {
        if (event.keyCode == 37) {
                console.log('Left arrow has been pressed');
        }
}

function myLoadEvent() {
  document.addEventListener('keydown', myKeyDown);
  document.addEventListener('keyup', myKeyUp);
}

document.addEventListener('DOMContentLoaded', myLoadEvent);
```

Keyup and keydown

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

```
var interval = 0;
function myKeyUp(event) {
        if (event.keyCode == 37) {
                clearInterval(interval);
        }
}

function myKeyDown(event) {
        if (event.keyCode == 37) {
                interval = setInterval(moveLeft, 10);
        }
}


function myLoadEvent() {
  document.addEventListener('keydown', myKeyDown);
  document.addEventListener('keyup', myKeyDown);
}

document.addEventListener('DOMContentLoaded', myLoadEvent);
```

Keyup and keydown

- This will not have quite the effect you expect!
- The code looks like it should start the timeout when the key is pressed
- And clear the timeout when the key is released
- However, you'll notice that as you hold the key down the circle moves faster and faster
- Like last week, 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*
- Java 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:

```
var leftPressed = false;

function myKeyDown(event) {
        if (event.keyCode == 37) {
                leftPressed = true;
        }
}

function myKeyUp(event) {
        if (event.keyCode == 37) {
                leftPressed = false;
        }
}



function myLoadEvent() {
  document.addEventListener('keydown', myKeyDown);
  document.addEventListener('keyUp', myKeyDown);
}

document.addEventListener('DOMContentLoaded', myLoadEvent);
```

Create a variable to store whether the left key is being pressed or not. Start it as "not pressed" (false)

When the key is pressed, set it to true

When the key is released, set it to true

Tracking key presses

- 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

```
var leftPressed = false;

function myKeyDown(event) {
        if (event.keyCode == 37) {
                leftPressed = true;
        }
}

function myKeyUp(event) {
        if (event.keyCode == 37) {
                leftPressed = false;
        }
}


function moveInterval() {
  if (leftPressed == true) {
    var element = document.getElementById('circle');
    var positionLeft = parseFloat(element.offsetLeft);
    element.style.left = positionLeft - 1 + 'px';
  }
}


function myLoadEvent() {
  document.addEventListener('keydown', myKeyDown);
  document.addEventListener('keyUp', myKeyDown);
  setInterval(moveInterval, 10);
}

document.addEventListener('DOMContentLoaded', myLoadEvent);
```

Exercise 2

- 10 minutes
- 1) Change the code so that when the left key is held down, the circle moves left. Once the key is released, it should stop moving
- 2) Add the other directions
- 3) If you've done this well, you should be able to press up and left to move the circle diagonally!

Exercise 2 - Solution

```
function myKeyDown(event) {
    if (event.keyCode == 37) {
        leftPressed = true;
    }
    if (event.keyCode == 38) {
        upPressed = true;
    }
    if (event.keyCode == 40) {
        downPressed = true;
    }
    if (event.keyCode == 39) {
        rightPressed = true;
    }
}

function myKeyUp(event) {
    if (event.keyCode == 37) {
        leftPressed = false;
    }
    if (event.keyCode == 38) {
        upPressed = false;
    }
    if (event.keyCode == 40) {
        downPressed = false;
    }
    if (event.keyCode == 39) {
        rightPressed = false;
    }
}
```

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


function moveInterval() {
  if (leftPressed == true) {
    moveLeft();
  }
  if (rightPressed == true) {
    moveRight();
  }
  if (upPressed == true) {
    moveUp();
  }
  if (downPressed == true) {
    moveDown();
  }
}
```

```
function moveUp() {
  var element = document.getElementById('circle');
  var positionTop = element.offsetTop;
  element.style.top = positionTop - 1 + 'px';

}
function moveDown() {
  var element = document.getElementById('circle');
  var positionTop = element.offsetTop;
  element.style.top = positionTop + 1 + 'px';
}

function moveLeft() {
  var element = document.getElementById('circle');
  var positionLeft = element.offsetLeft;
  element.style.left = positionLeft - 1 + 'px';
}

function moveRight() {
  var element = document.getElementById('circle');
  var positionLeft = element.offsetLeft;
  element.style.left = positionLeft + 1 + 'px';
}


function myLoadEvent() {
  document.addEventListener('keydown', myKeyDown);
  document.addEventListener('keyup', myKeyUp);
  setInterval(moveInterval, 10);
}

document.addEventListener('DOMContentLoaded', myLoadEvent);
```

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:

```
var element = document.getElementById('circle');
element.className = 'nameOfClass';
```

Setting CSS classes using javascript

- 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;
}
```

```
  var element = document.getElementById('circle');
  element.className = 'semiTransparentAndBlue';
```

Setting CSS classes using javascript

- You can apply more than one CSS class to an element by separating the class names with a space:

```
.semiTransparent {
  opacity: 0.5;
}

.blue {
   background-color: blue;
}
```

```
  var element = document.getElementById('circle');
  element.className = 'semiTransparent blue';
```

Finding Elements using CSS Classes

- So far to find elements on the page, we have been using the code
  - document.getElementById()
- 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

- 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

Finding Elements using CSS Classes

- document.getElementById() 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

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

Finding Elements using CSS Classes

-
  ```html
  <div class="circle">
    </div>
    <div class="circle">
    </div>
  ```
- Which element would you expect the following code to find?

```javascript
  var element = document.getElementByClassName('circle');
```

Finding Elements using CSS Classes

- Because it's not clear, there is no function
- getElementByClassName()
- Instead, the function is
- getElementsByClassName()
- That is getElementS instead of element (no S)
- This will retrieve more than one element

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

  </div>

  <div class="circle">

  </div>
```

```javascript
var elements = document.getElementsByClassName('circle');
```

- 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:

```javascript
var elements = document.getElementsByClassName('circle');
elements[0].style.backgroundColor = 'blue';
```

- elements[0] is the first found element
- elements[1] is the second found element
- elements[2] is the third
- etc

```
<div class="circle">

  </div>

  <div class="circle">

  </div>
```

```
  var elements = document.getElementsByClassName('circle');
  elements[0].style.backgroundColor = 'blue';
  elements[1].style.backgroundColor = 'green';
```

Exercise 3

- Until the end of the session
- 1) Complete all other exercises from this term so far
- 2) Download the branch Exercise3-Exercise from Github
- 3) The exercise contains the beginning of a game. There are several CSS classes that can be applied to the element with the ID `player`:
  - standLeft
  - standRight
  - standDown
  - StandUp
- 3a) Try setting these in the HTML. You will need to set the class to "character standLeft", "character standRight", etc
- 3b) When the arrow keys are pressed, set the relevant direction on the player element

Exercise 3 - Continued

- There are other classes:
  - walkUp
  - walkDown
  - walkLeft
  - walkRight
- 4) When the arrow keys are pressed, the character should walk around the screen and the relevant walking animation should be displayed.
- 5) When the arrow keys are not being pressed the player should be still and facing the direction they were last movement
- 6) You should be able to walk diagonally by pressing two keys at once

Exercise 3 - Continued

- There is a side bar on the right hand of the screen
- 1) When you click on one of the heads, change the player's head to the one that was clicked on
  - Hint: Change the background image of the element with the class name 'head'
- 2) When you click on one of the bodies, change the player's body to the one that was clicked on
- 3) When clicking the “X” at the top of the sidebar, slide it off the screen to the right
- 4) When the player is clicked on, re-open the sidebar by sliding it in

Exercise 3 - Continued

- Extra exercises:
  - Stop the player leaving the screen
  - Prevent the player from walking over the tree
  - Difficulty: Hard: Make the opponent walk around the screen applying appropriate animations
  - Difficulty: Hard Add more trees that block the player, make a more interesting level e.g. a maze