CSY1018
Web Development

Tom Butler
thomas.butler@northampton.ac.uk

Topic 5

- Generating random numbers
- Finding the next element in the document with nextSibling
- Introduction to Arrays

Exercise 1

- 10 – 15 minutes
- 1) Download the Exercise1-Exercise branch from Topic5 on the github page
- 2) Add a script tag to the page to run some javascript
- 3) Using getElementsByTagName() and a loop add a click event listener to each dice image (Each dice is a element)
- 4) Each time you click an image, set the class name of the div to "side3" - This will display the dice face with 3 dots
  - Hint: 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!

Exercise 1 - Solution

```
function diceClick() {
        this.className = 'side3';
}

function myLoadEvent() {
        var elements = document.getElementsByTagName('div');
        for (var i = 0; i < elements.length; i++) {
                elements[i].addEventListener('click', diceClick);
        }
}

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

Random Numbers

- In Javascript random numbers can be generated using the code:

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

- This will store a random number between 0 and 1 inside the variable randomNumber
- e.g. 0.23445466 or 0.93445477

Random Numbers

- 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:

Random Numbers

- This will create a random number between 0 and 10

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

- e.g.
  - 8.23344564
  - 2.04933531
  - 0.12467897
  - 9.46589342

Random Numbers

- 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:

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

Random Numbers

- 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

- 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 (inclusive) you can use

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

- Or for 0-9 you can use:

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

Exercise 2

- 15 minutes
- 1) 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.
- 2) 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

  - Hint: You should be able to do this without 6 if statements!

Exercise 2 - Solution

```
function diceClick() {
        var diceRoll = Math.ceil(Math.random() * 6);
        this.className = 'side' + diceRoll;
        alert('You rolled a ' + diceRoll);
}

function myLoadEvent() {
        var elements = document.getElementsByTagName('div');
        for (var i = 0; i < elements.length; i++) {
                elements[i].addEventListener('click', diceClick);
        }
}

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

Finding the next element on the page

- The HTML for the dice game looks like this:

```
<!doctype html>
<html>
        <head>
                <title>Dice Game</title>
                <link rel="stylesheet" href="dice.css" />
                <script src="script.js"></script>
        </head>
        <body>

        <div class="side1">
        </div>

        <p>Roll the dice</p>

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

Finding the next element

- It would be better if instead of printing “You rolled a 5” in an alert box, the paragraph below the relevant dice image was updated:

```
<div class="side1">

</div>

<p>Roll the dice</p>

<div class="side1">

</div>
<p>Roll the dice</p>
```

Finding the paragraph

- When a dice image is clicked on this function is called:

```
function diceClick() {
        var diceRoll = Math.ceil(Math.random() * 6);
        this.className = 'side' + diceRoll;
        alert('You rolled a ' + diceRoll);
}
```

- The variable *this* tells us which dice image was clicked on
  - Unfortunately it does not give us an index number
  - There's no way to use getElementsByTagName('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 via getElementById(), getElementsByTagName() or the this variable) has a property called *nextSibling* e.g.

```
var element = document.getElementById('mydiv');
var nextElement = element.nextSibling;
```

```
<div id="mydiv">

</div>

<h2>Heading</h2>
```

nextSibling

- 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!

```
        var element = document.getElementById('mydiv');
        var nextElement = element.nextSibling;
```

```
        <div class="myelement">

        </div>
        <h2>Heading</h2>
```

nextSibling

- To find the next *element* you will need to use nextSibling twice

```
var element = document.getElementById('mydiv');
var nextElement = element.nextSibling.nextSibling;
```

```
<div class="myelement">

</div>

<h2>Heading</h2>
```

- You can keep using nextSibling to traverse down the entire document

nextSibling

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

```
element.nextSibling.nextSibling.style.backgroundColor = 'red';
element.nextSibling.nextSibling.style.className = 'red';
element.nextSibling.nextSibling.firstChild.nodeValue = 'new text content';
element.nextSibling.nextSibling.addEventListener('new text content', myClickEvent);
```

Exercise 3

- 15 minutes
- 1) Using nextSibling, 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
- 2) **(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

Exercise 3 - Solution

```
function diceClick() {
        var diceRoll = Math.ceil(Math.random() * 6);
        this.className = 'side' + diceRoll;
        this.nextSibling.nextSibling.firstChild.nodeValue = 'You rolled a ' + diceRoll;
}

function myLoadEvent() {
        var elements = document.getElementsByTagName('div');
        for (var i = 0; i < elements.length; i++) {
                elements[i].addEventListener('click', diceClick);
        }
}

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

Arrays

- So far, most variables you have assigned a value to has contained a *single value* e.g.

```
var number = 4;
var string = 'hello world';
var element = document.getElementById('mydiv');
var 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

- You have already used an array when using getElementsByTagName() and getElementsByClassName:

```
<div class="circle">

  </div>

  <div class="circle">

  </div>
```

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

  elements[1].style.backgroundColor = 'green';
```

Arrays

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

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

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

  elements[1].style.backgroundColor = 'green';
```

- In this example, the elements variable is an *array*
- The variable stores more than one element

Arrays

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

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

elements[1].style.backgroundColor = 'green';
```

- Here, 0 and 1 are mapped to *elements* on the page

Creating an array

- To create an array you use two square brackets:

```
var myArray = [];
```

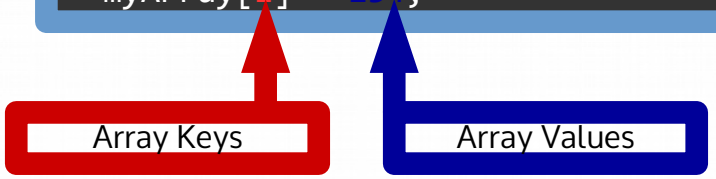
- This will create an empty array, you can then write values to individual indexes. e.g.

```
var myArray = [];
myArray[0] = 121;
myArray[1] = 234;
```

Creating an array

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

```
var myArray = [];
myArray[0] = 121;
myArray[1] = 234;
```



Creating an array

- You can store any type of variable in the array e.g. strings

```
var myArray = [];
myArray[0] = 'green';
myArray[1] = 'blue';
```

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

```
alert(myArray[1]); // prints “blue” in an alert box
```

Creating an array

```
var myArray = [];
myArray[0] = 'green';
myArray[1] = 'blue';
```

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

```
var num = 1;
alert(myArray[num]); // prints “blue” in an alert box
```

```
var num = 0;
alert(myArray[num]); // prints “green” in an alert box
```

Extending the game

- Instead of displaying “You rolled a 5” it would be nicer to display the english equivalent of the numeral e.g.
  - “You rolled a five”
  - “You rolled a three”
  - etc

Extending the game

- This could be done with a series of if statements:

```
function diceClick() {
        var diceRoll = Math.ceil(Math.random() * 6);
        this.className = 'side' + diceRoll;

        if (diceRoll == 1) {
                this.nextSibling.nextSibling.firstChild.nodeValue = 'You rolled a one';
        }
        if (diceRoll == 2) {
                this.nextSibling.nextSibling.firstChild.nodeValue = 'You rolled a two';
        }
        if (diceRoll == 3) {
                this.nextSibling.nextSibling.firstChild.nodeValue = 'You rolled a three';
        }
        if (diceRoll == 4) {
                this.nextSibling.nextSibling.firstChild.nodeValue = 'You rolled a four';
        }
        if (diceRoll == 5) {
                this.nextSibling.nextSibling.firstChild.nodeValue = 'You rolled a five';
        }
        if (diceRoll == 6) {
                this.nextSibling.nextSibling.firstChild.nodeValue = 'You rolled a six';
        }
}
```

Exercise 4

- 1) Use an array instead of if statements to display the relevant text when the dice is clicked:
  - Hint: You should not need any if statements
  - Hint: You should not need any loops in the click event
  - Hint: You only need one array
- 2) Add a timer to the game to make the dice look like it's being rolled for a few seconds when the dice is clicked. It should display various numbers before settling on one
- 3) **Optional** Continue to exercise 5

Exercise 4 – Solution (1)

```
function diceClick() {
        var numbers = [];
        numbers[1] = 'One';
        numbers[2] = 'Two';
        numbers[3] = 'Three';
        numbers[4] = 'Four';
        numbers[5] = 'Five';
        numbers[6] = 'Six';

        var diceRoll = Math.ceil(Math.random() * 6);
        this.className = 'side' + diceRoll;

        this.nextSibling.nextSibling.firstChild.nodeValue = 'You rolled a ' + numbers[diceRoll];
}

function myLoadEvent() {
        var elements = document.getElementsByTagName('div');
        for (var i = 0; i < elements.length; i++) {
                elements[i].addEventListener('click', diceClick);
        }
}

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

Exercise 4 – Solution (2)

```
var interval = 0;
var stopInterval = 0;
var currentDice = 0;

function animateDice() {
        var diceRoll = Math.ceil(Math.random() * 6);
        currentDice.className = 'side' + diceRoll;
}
function stopAnimation() {
        clearInterval(interval);
        clearInterval(stopInterval);

        var numbers = [];
        numbers[1] = 'One';
        numbers[2] = 'Two';
        numbers[3] = 'Three';
        numbers[4] = 'Four';
        numbers[5] = 'Five';
        numbers[6] = 'Six';

        var diceRoll = Math.ceil(Math.random() * 6);
        currentDice.className = 'side' + diceRoll;
        currentDice.nextSibling.nextSibling.firstChild.nodeValue = 'You rolled a ' + numbers[diceRoll];
}
function diceClick() {
        currentDice = this;
        interval = setInterval(animateDice, 100);
        stopInterval = setInterval(stopAnimation, 2000);
}
function myLoadEvent() {
        var elements = document.getElementsByTagName('div');
        for (var i = 0; i < elements.length; i++) {
                elements[i].addEventListener('click', diceClick);
        }
}
document.addEventListener('DOMContentLoaded', myLoadEvent);
```

Exercise 5

- **Optional (no solution available)**
- The dice game will break if you click more than one dice at a time. Fix this by storing timers and dice in arrays. Hint: You can create an array of boolean values to track whether each dice should be animating or not.
- Extend last week's game so that the opponent character walks around the screen randomly e.g.
  - Walk left for 3 seconds
  - Walk up for 2 seconds
  - Walk right for 6 seconds
- Add more characters to the game using the available heads/bodies and have them all walk randomly around the screen