DEEPer

JavaScript
Week 2 Session 1

JavaScript

01

What is JavaScript?

- **Completely unrelated to Java**
- JIT language run in browsers – no pre-compilation required
- Loosely typed
- Forms the third high-level aspect of web pages;
    - Structure – HTML
    - Styling – CSS
    - **Interaction - JavaScript**
- Can also be run on the server (NodeJS) – transferrable knowledge, but not the topic of this course

Inclusion - Example Code

- Like CSS, can be included within an HTML file directly, or imported from a separate .js file

```html
<!DOCTYPE html>
<html>
    <head>
        <script>
            alert('hello from the head!');
        </script>
        <script src="example.js"></script>
    </head>
    <body>
    </body>
</html>
```

Variables

- A variable is a container for a value
- This value can be defined by the developer in code, or retrieved from other sources, like user input
- Each variable has a unique name
- The value of a variable can be changed at any time
- The value within a variable can then be used for display, calculation, etc
- The data within a variable can be one of many "types"

Variables – Definition

```
// var [variableName] = [value];
var name = 'Joe Bloggs';
```

Variables – Primitive Types

```
/*
 * Primitive types are types that have a single value
 * There are non-objects, and have no methods (more on this to come)
 */

var name = 'Joe Bloggs'; // string
var age = 20; // number
var isMammal = true; // boolean
var cost = undefined; // undefined
```

Variables – Structural Types

```
/*
 * Structural types are more complex types, which can hold multiple values
 * There are technically more, but for now we will review Array and a generic Object
 */

// array
var names = ['Joe', 'Jane', 'Bob', 'Billy'];

// object
var address = {
  lineOne: '123 Fake Street',
};
```

Variables – Arrays

- An array is a flat list-like object capable of holding multiple values within a single variable
- Arrays also include functions to facilitate traversal and mutation operations
- Array elements each have a unique index – starting at 0
- Elements added to the end of an array are assigned the next available index

```
//           0      1       2      3
var names = ['Joe', 'Jane', 'Bob', 'Billy'];
names.push('Simon'); // Adds "Simon" to the array with an index of 4
```

Functions

- A function is a snippet of code which is wrapped in a reusable block, and given a name
- When a function is ”called”, it can optionally be passed parameters that the snippet can then use
- A function can optionally **return** a value for use by the calling code
- Like variables, a function’s name must be unique

Functions

```
function addNumbers(a, b) {
  return a + b;
}

var sum = addNumbers(3, 5);

alert(sum) // 8
```

- This function accepts two parameters – `a` and `b`
- The two parameters are added together, and returned
- Obviously, functions will tend to be more complex than this

Variables – Objects

- Almost everything in JavaScript behaves like an object, including primitives like strings
- Each type provides properties and/or methods we can call
- For example, we can check a string’s length through a provided property

```
var name = 'Joe Bloggs';
alert(name.length); // 10
```

Variables – Objects

- We can also define our own objects, with our own properties and methods (functions)

```javascript
var user = {
  name: 'Joe Bloggs',
  age: 20,
  address: {
    lineOne: '123 Fake Street',
    // ...
  },
  speak: function () {
    alert('Hello!');
  },
};

var name = user.name;
var lineOne = user.address.lineOne;
user.speak();
```

Using Variables

```
// Primitive variables are referenced using their name
var name = 'Joe Bloggs';
alert(name); // "Joe Bloggs"
```

Comparison

```
var a = '123';
var b = 123;

var looseEquality = a == b; // true
var strictEquality = a === b; // false
```

- A double-equals will perform type conversion when testing equality
- A triple-equals will test both value **and** type

Conditionals

- We often want to run code only if certain conditions are met
- For example, if a form is not valid, present an error to the user
- Checking for falsey values can be shortcutted with `!`

```
function submitForm(formData) {
  // Some validation logic...
  var isValid = validateForm(formData);

  if (isValid === false) {
    alert('Form is invalid!');
    return;
  }

  // ...
}
```

```
function submitForm(formData) {
  // Some validation logic...
  var isValid = validateForm(formData);

  if (!isValid) {
    alert('Form is invalid!');
    return;
  }

  // ...
}
```

Conditionals

- Often, we have multiple conditions to test, where only one of them can be true
- We can utilise `else if` and `else` conditions

```
var sum = 8; // Calculated from somewhere...

if (sum < 0) {
  alert('Less than 0');
} else if (sum <= 10) {
  alert('0 - 10');
} else {
  alert('> 10');
}
```

Conditionals

- For cases where there are many options to test, we can use a switch statement instead
- The first matched case is entered. If it concludes with a break, no more cases will be tested
- If no case is matched here, the default will be entered

```
var userRole = 'ADMIN';

switch (userRole) {
  case 'USER':
    alert('User has role User!');
    break;

  case 'ADMIN':
    alert('User has role Admin!');
    break;

  case 'SUPERADMIN':
    alert('User has role Super Admin!');
    break;

  // ...

  default:
    alert('User role was not recognised!');
    break;
}
```

Loops

- We commonly require a piece of code to run multiple times
- As developers, we may not know when writing code how many times the code needs to run
- For example, if running a block of code once for every product a user has in their basket
- We can use loops to handle this dynamically repeated task for us

Loops

```
var i = 0;
var output = '';

while (i < 10) {
  output += 'Iteration ' + i + ' <br>';

  i++;
}

document.write(output);
```

```
Iteration 0
Iteration 1
Iteration 2
Iteration 3
Iteration 4
Iteration 5
Iteration 6
Iteration 7
Iteration 8
Iteration 9
```

Repeats some code until a condition is no longer met – the number of iterations likely isn't known up front

Loops

```
var output = '';

for (var i = 0; i < 10; i++) {
  output += 'Iteration ' + i + ' <br>';
}

document.write(output);
```

```
Iteration 0
Iteration 1
Iteration 2
Iteration 3
Iteration 4
Iteration 5
Iteration 6
Iteration 7
Iteration 8
Iteration 9
```

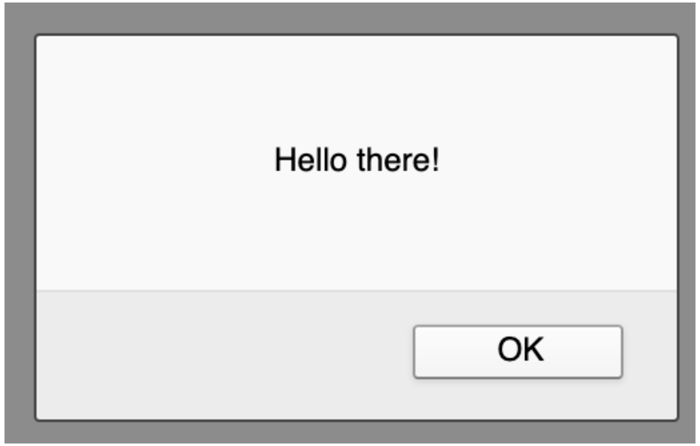
- Repeats some code a set given number of times
- The number of iterations can be dynamic, from a variable

Dialogs

- There are three types of browser-provided dialog
    - `alert` – has a single "OK" button
    - `confirm` – has two buttons: "Confirm" or "Cancel"
    - `prompt` – has an input field with "OK" or "Cancel"
- All default dialogs are synchronous, meaning all following code is blocked until interaction is complete
- These default dialogs are styled by browsers, and cannot be restyled using CSS, or have the buttons relabelled
- Dialogs are great for getting started – going forward we will replace with custom, skinnable components

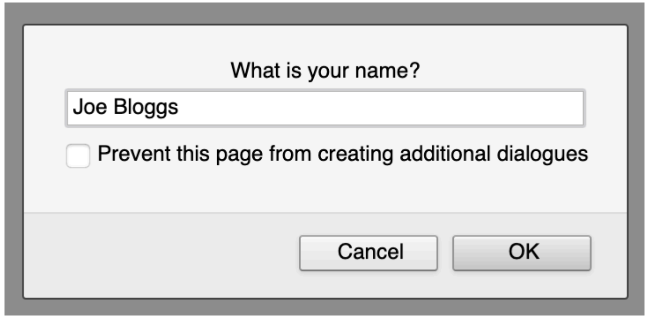
Dialogs

```
<!DOCTYPE html>
<html>
  <head>
    <script>
      alert('Hello there!');
      var name = prompt('What is your name?');
      var likesCats = confirm('Click OK if you like cats');
    </script>
  </head>
  <body>
    <p>
      <script>
        document.write('Hello, ' + name + '!');
      </script>
    </p>
    <p>
      <script>
        if (likesCats) {
          document.write("Let's be friends!");
        } else {
          document.write('Go away.');
        }
      </script>
    </p>
  </body>
</html>
```


Hello there!

Dialogs

```
<!DOCTYPE html>
<html>
  <head>
    <script>
      alert('Hello there!');
      var name = prompt('What is your name?');
      var likesCats = confirm('Click OK if you like cats');
    </script>
  </head>
  <body>
    <p>
      <script>
        document.write('Hello, ' + name + '!');
      </script>
    </p>
    <p>
      <script>
        if (likesCats) {
          document.write("Let's be friends!");
        } else {
          document.write('Go away.');
        }
      </script>
    </p>
  </body>
</html>
```



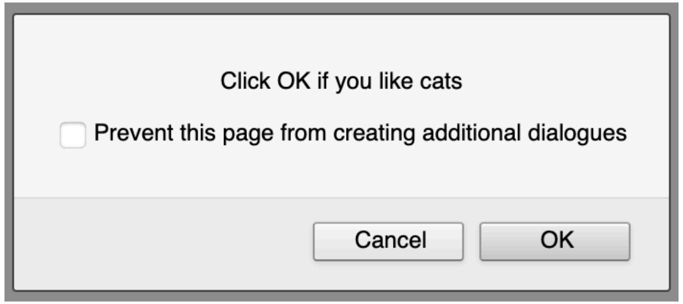
What is your name?

Joe Bloggs

Prevent this page from creating additional dialogues

Dialogs

```
<!DOCTYPE html>
<html>
  <head>
    <script>
      alert('Hello there!');
      var name = prompt('What is your name?');
      var likesCats = confirm('Click OK if you like cats');
    </script>
  </head>
  <body>
    <p>
      <script>
        document.write('Hello, ' + name + '!');
      </script>
    </p>
    <p>
      <script>
        if (likesCats) {
          document.write("Let's be friends!");
        } else {
          document.write('Go away.');
        }
      </script>
    </p>
  </body>
</html>
```


Click OK if you like cats

Prevent this page from creating additional dialogues

Dialogs

```
<!DOCTYPE html>
<html>
  <head>
    <script>
      alert('Hello there!');
      var name = prompt('What is your name?');
      var likesCats = confirm('Click OK if you like cats');
    </script>
  </head>
  <body>
    <p>
      <script>
        document.write('Hello, ' + name + '!');
      </script>
    </p>
    <p>
      <script>
        if (likesCats) {
          document.write("Let's be friends!");
        } else {
          document.write('Go away.');
        }
      </script>
    </p>
  </body>
</html>
```

Hello, Joe Bloggs!

Let's be friends!

DOM Manipulation

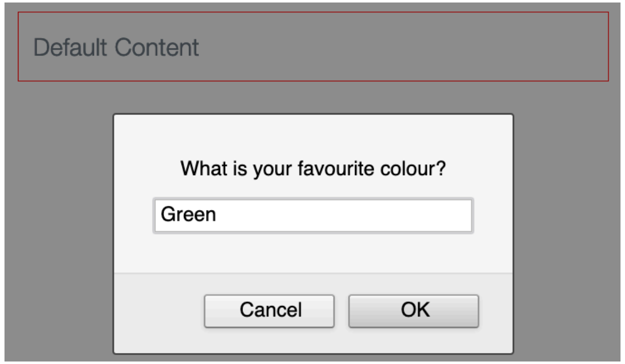
- JavaScript is primarily used to manipulate the DOM, or the HTML elements within a web page
- JavaScript can dynamically add new elements, remove elements or update existing elements within the DOM
- In order to update the content of an element, we must first obtain a reference to it
- We generally store an element reference in a variable to be manipulated

Selecting Elements

```
<p id="first-paragraph" class="paragraph"></p>
<p id="second-paragraph" class="paragraph"></p>

<script type="text/javascript">
  // "document" represents the whole DOM tree

  // Obtain reference by the ID attribute
  var firstParagraph = document.getElementById('first-paragraph');

  // Obtain ALL elements which match the given CSS selector
  // (all elements with a class of "paragraph")
  var allParagraphs = document.querySelectorAll('.paragraph');
</script>
```

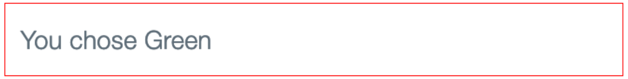
Replacing Element Content

```
<div id="output" class="my-box">
  Default Content
</div>

<script type="text/javascript">
  var outputBox = document.getElementById('output');
  var favouriteColour = prompt('What is your favourite colour?');

  outputBox.innerHTML = 'You chose ' + favouriteColour;
</script>
```


Default Content

What is your favourite colour?

Green


You chose Green

1. Obtain an element reference, for example using **`document.getElementById()`**
2. Set the innerHTML property to the desired output, which replaces any existing content

Appending Element Content

```html
<p id="output">Default Content</p>

<script type="text/javascript">
  var output = document.getElementById('output');
  output.append(' (and some dynamic content)');
</script>
```

Default Content (and some dynamic content)

1. Obtain an element reference, for example using **`document.getElementById()`**
2. Use the **`element.append()`** function to add output to the end of the current content within

Creating DOM Elements

```
var dynamicElement = document.createElement('p');
dynamicElement.innerHTML = 'My dynamic content';
dynamicElement.id = 'my-id';
dynamicElement.className = 'my-class';
```

1. Create an empty DOM element of the provided tag name
2. Use the standard element API to manipulate the element (this approach can also be used to edit existing elements after selection)

Appending DOM Elements

```
<ul id="shopping-list">
  <li>Milk</li>
</ul>

<script type="text/javascript">
  var listItem = document.createElement('li');
  listItem.innerHTML = 'Bread';

  var list = document.getElementById('shopping-list');
  list.appendChild(listItem);
</script>
```

- Milk
- Bread

1. Create and populate a new DOM element
2. Use **`element.appendChild()`** to add the new element to a selected parent. (note **`append()`** does also support DOM elements)

Dialogs - innerHTML

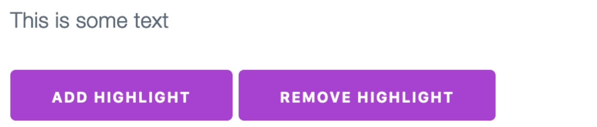
```
<body>
  <p id="greeting"></p>
  <p id="cats-message"></p>

  <script>
    alert('Hello there!');
    var name = prompt('What is your name?');
    var likesCats = confirm('Click OK if you like cats');

    var greetingElement = document.getElementById('greeting');
    var catsElement = document.getElementById('cats-message');
    var greetingMessage = 'Hello, ' + name + '!';
    var catsMessage = 'Go away.';

    if (likesCats) {
      catsMessage = "Let's be friends!";
    }

    greetingElement.innerHTML = greetingMessage;
    catsElement.innerHTML = catsMessage;
  </script>
</body>
```

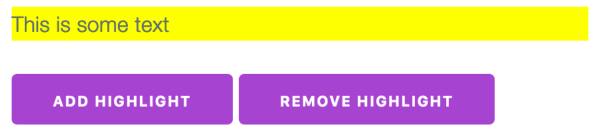
Manipulating Styling

- JavaScript can add and remove attributes from DOM elements, such as classes
- JavaScript can also be used to directly update CSS properties of elements

Manipulating Element Classes

```
<style>
  .highlight {
    background: yellow;
  }
</style>

<p id="demo-paragraph">This is some text</p>

<button onclick="addHighlightClass()" type="button">
  Add Highlight
</button>
<button onclick="removeHighlightClass()" type="button">
  Remove Highlight
</button>

<script>
  function addHighlightClass() {
    var paragraph = document.getElementById('demo-paragraph');
    paragraph.classList.add('highlight');
  }

  function removeHighlightClass() {
    var paragraph = document.getElementById('demo-paragraph');
    paragraph.classList.remove('highlight');
  }
</script>
```

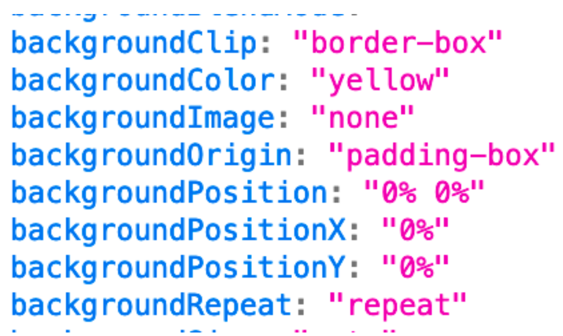

This is some text


This is some text

Manipulating CSS Properties

```
<p id="demo-paragraph">This is some text</p>

<button onclick="addHighlight()" type="button">
  Add Highlight
</button>
<button onclick="removeHighlight()" type="button">
  Remove Highlight
</button>

<script>
  function addHighlight() {
    var paragraph = document.getElementById('demo-paragraph');
    paragraph.style.background = 'yellow';
  }

  function removeHighlight() {
    var paragraph = document.getElementById('demo-paragraph');
    paragraph.style.background = 'none';
  }
</script>
```

- [element].style is an object containing the current CSS properties
- Properties can be updated directly and dynamically using JavaScript
- All default values are included and can be updated, e.g…



```
backgroundClip: "border-box"
backgroundColor: "yellow"
backgroundImage: "none"
backgroundOrigin: "padding-box"
backgroundPosition: "0% 0%"
backgroundPositionX: "0%"
backgroundPositionY: "0%"
backgroundRepeat: "repeat"
```

Detecting Interactions

- Whenever the browser parses JavaScript code, it runs it from top to bottom
- This process runs immediately – on first page load, when the JavaScript code is first loaded
- Most of the JS code we will write however we do not want to run immediately
- We want to be able to react to how the user interacts with our application
- We want to detect interactions, through **events**

Basic Events - Attributes

```
<!-- On element click -->
<button onclick="myFunction()">Click Me</button>

<!-- When the input gains focus -->
<input onfocus="myFunction()">

<!-- When the input loses focus -->
<input onblur="myFunction()">

<!-- When the value changes -->
<input onchange="myFunction()">

<!-- When the user presses down a key when focused on the element -->
<input onkeydown="myFunction()">

<script>
  function myFunction() {
    // Some logic...
  }
</script>
```

Basic Events - Dynamic

```
<button id="my-button">Click Me</button>

<script>
  var button = document.getElementById('my-button');
  button.onclick = function() {
    // Some logic...
  };

  // The same other events are available
</script>
```

Dialogs - onclick

```
<body>
  <button onclick="greetMe()">Greet Me!</button>
  <p id="greeting"></p>
  <p id="cats-message"></p>

  <script>
    function greetMe() {
      alert('Hello there!');
      var name = prompt('What is your name?');
      var likesCats = confirm('Click OK if you like cats');

      var greetingElement = document.getElementById('greeting');
      var catsElement = document.getElementById('cats-message');
      var greetingMessage = 'Hello, ' + name + '!';
      var catsMessage = 'Go away.';

      if (likesCats) {
        catsMessage = "Let's be friends!";
      }

      greetingElement.innerHTML = greetingMessage;
      catsElement.innerHTML = catsMessage;
    }
  </script>
</body>
```