CE212

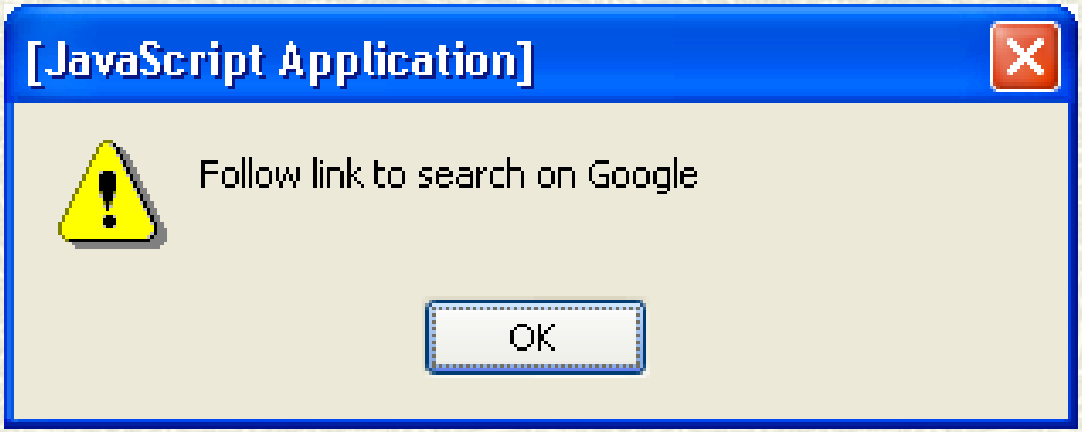
Web Application Programming

Part 2

JavaScript Event-Handlers 1

JavaScript may be invoked to handle input events on HTML pages, e.g.

```
<html>
  <body>
    <a href = "http://www.google.co.uk"
       onmouseover = "alert(
        'Follow link to search on Google')">
       Search on Google
    </a>
  </body>
</html>
```



JavaScript Event-Handlers 2

The general syntax for JavaScript event-handling in HTML tags is

```
<tag attribute1 ..... attributen
    oneventname="JavaScriptCode">
```

Events that can be detected and handled include the entry of a form element (using **onselect**), the leaving of a form element after changing it (using **onchange**) and the clicking of a button (using **onsubmit**).

The JavaScript code would usually involve a call to a function written in JavaScript in the document header (or in an external file.)

JavaScript Event-Handlers – An Example 1

On the next slides we present a factorial function in JavaScript and a form in which the user should enter a value whose factorial is to be calculated. When the user has completed entry of the value the code for the `onchange` event is used to call the factorial function and display its result.

Note the use of `this.value` to get the value of the current item and `document.getElementById` to access the element in which the result is to be displayed; these features will be further explained in part 3.

An example of the form in use can be seen on the subsequent slide.

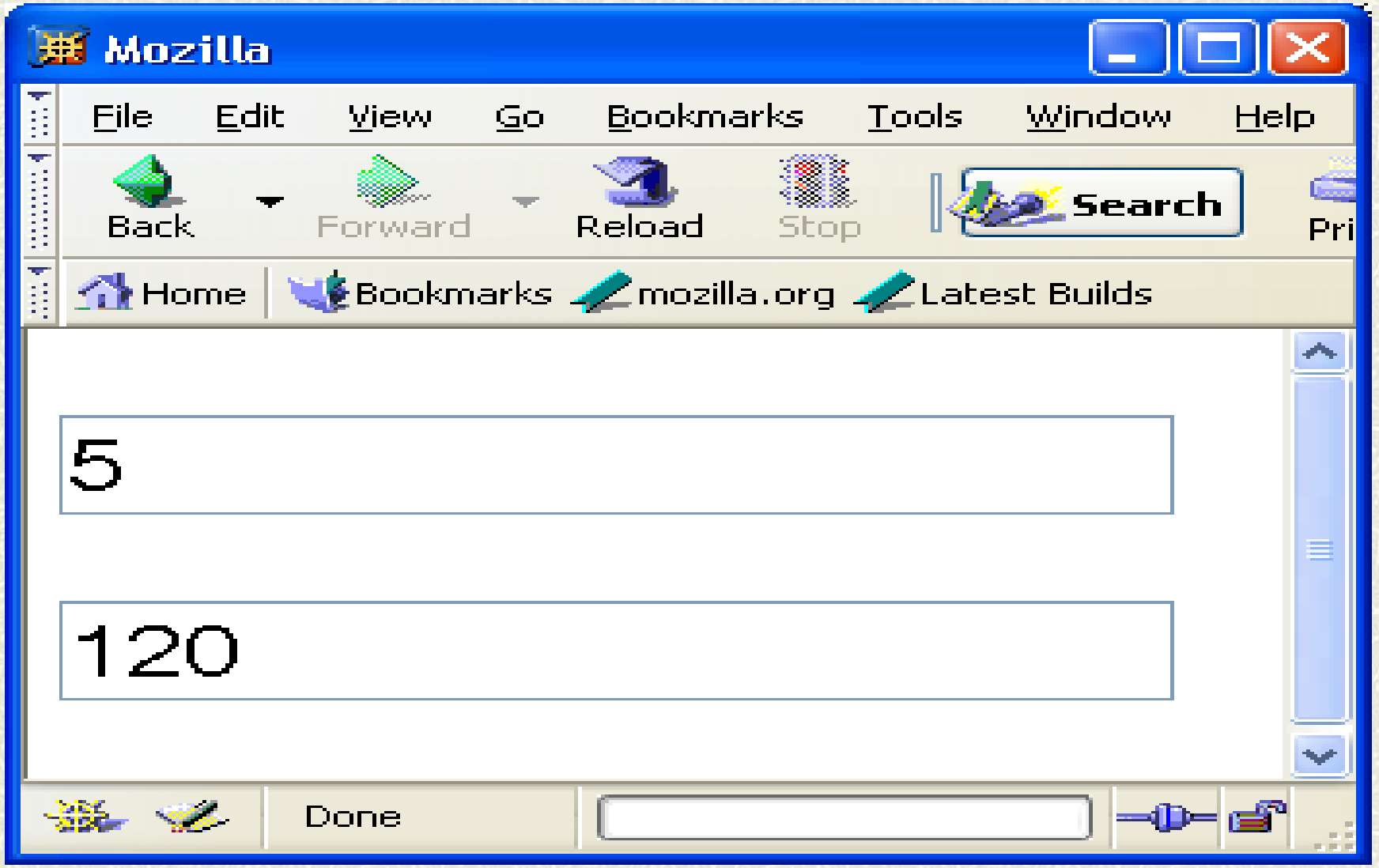
JavaScript Event-Handlers – An Example 2

```
<html>
  <head>
    <script language="JavaScript"
            type="text/javascript">
      function fac(n)
      { if (n < 2)
          return 1;
        else
          return n * fac(n-1);
      }
    </script>
  </head>
```

JavaScript Event-Handlers – An Example 3

```
  <body>
    <form>
      <p>
        <input id = "facArg" type = "text"
         onchange = " result = fac(this.value);
              document.getElementById(
              'facRes').value = result; " />
      </p>
      <p>
        <input id = "facRes" type = "text" />
      </p>
    </form>
  </body>
</html>
```

JavaScript Event-Handlers – An Example 4



Invoking JavaScript When a Page is Loaded

JavaScript can be invoked when the page is loaded using the **onload** attribute of the **body** tag, e.g.

```
<html>
  <head>
    <link rel="stylesheet" type="text/css"
          href="../css/converter.css"/>
    <title>Celsius to Fahrenheit Converter</title>
    <script language="JavaScript" src="../js/c2f.js"
            type="text/javascript">
    </script>
  </head>
  <body onload="conversionTable('conversion', 0, 30);">
    <h2>Celsius to Fahrenheit Converter</h2>
      <div id="conversion"> </div>
  </body>
</html>
```

Invoking JavaScript from a Hyperlink 1

JavaScript code can also be the destination of a hyperlink – in this case a function is invoked when the user clicks on the link.

In this case the syntax is

```
<a href="javascript: myFunc(args) ">
Click here to invoke myFunc(args) </a>
```

The example on the following slides shows the use of hyperlinks to change the style of a list. To keep the example concise the list is simply the list of hyperlinks.

Invoking JavaScript from a Hyperlink 2

```
<html>
  <head>
    <style> …… </style>
    <script language="JavaScript"
            type="text/javascript">
      function listStyle(style)
      { var ml = document.getElementById("myList");
        ml.setAttribute("class", style)
      }
    </script>
  </head>
```

Invoking JavaScript from a Hyperlink 3

```
<body>
 <p>Click a link to choose a style</p>
   <div id="myList" class="navcontainer">
    <ul> <li>
      <a href="javascript:listStyle('')"> default </a>
     </li>
     <li>
      <a href="javascript:listStyle('navcontainer')">
         inline </a>
     </li>
     <li>
      <a href="javascript:listStyle('nobullet')">
         no-bullet vertical </a>
     </li> </ul>
  </div> </body> <html>
```

Using the `eval` Function 1

The **`eval`** function allows any string containing JavaScript code to be evaluated at runtime. The result of this function is the value (if any) of the last expression or statement in the code.

Using this function it is possible to generate code at runtime and immediately run it. We present an example that allows the user to type JavaScript into a form and run it.

If the evaluation fails and an exception is thrown the exception message will be displayed (its **`toString`** function will be invoked implicitly) instead of the result of the evaluation. Note that the contents of the text area are obtained using its **`value`** property; the contents of the pre tags are written by assigning to their **`textcontent`** property.

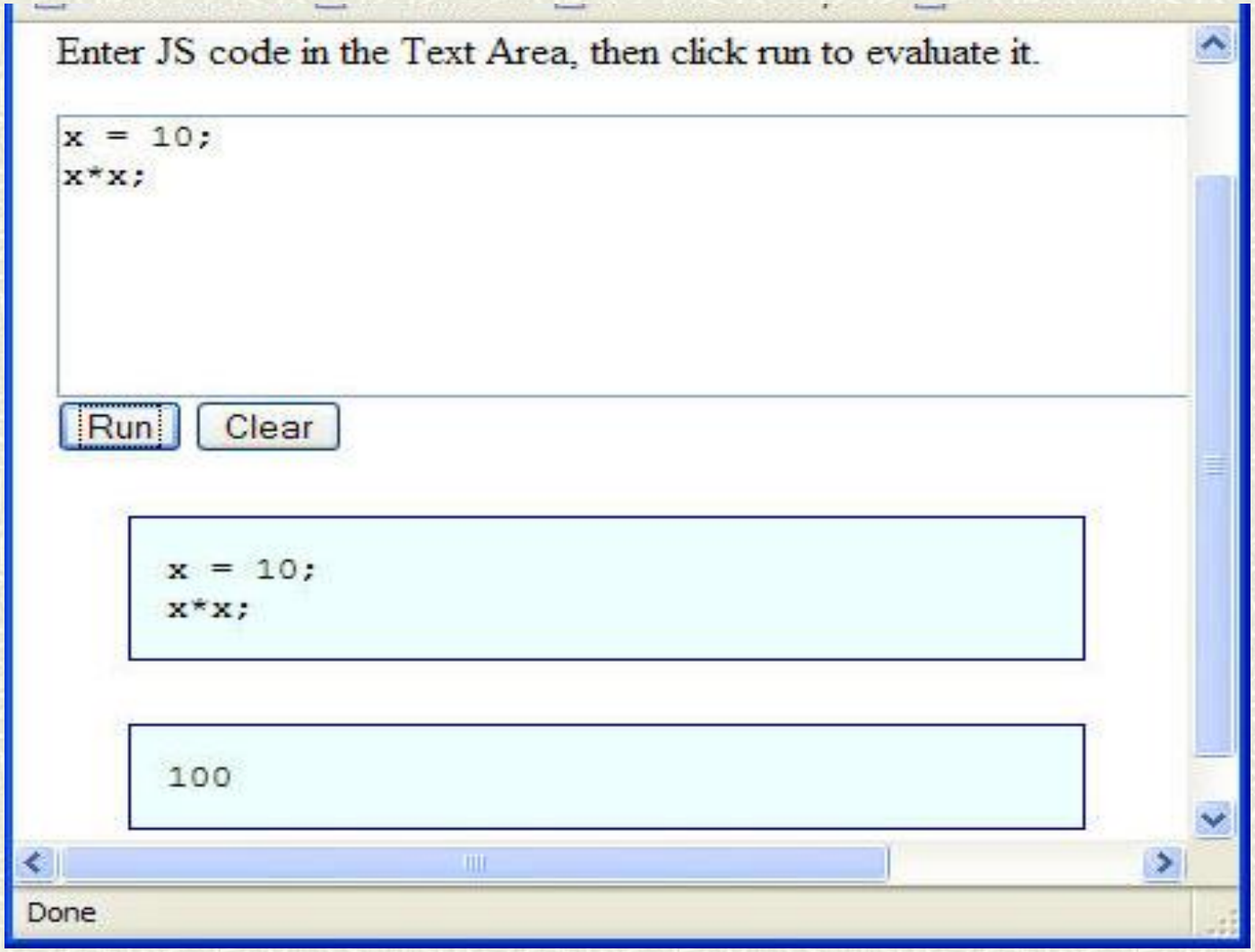
Using the `eval` Function 2

```
<html>
 <head>
  <script language="JavaScript" type="text/javascript">
    function run()
    { var ipNode = document.getElementById("input");
      var opNode = document.getElementById("output");
      opNode.textContent = ipNode.value;
      var evNode = document.getElementById("eval");
      try
      { evNode.textContent = eval(ipNode.value);
      }
      catch (Exception e)
      { evNode.textContent = e;
      }
    }
  </script>
 </head>
```

Using the `eval` Function 3

```
<body>
 <div>
  <h3> JavaScript test </h3>
  <p> Enter JS code in the text area,
      then click Run to evaluate it </p>
 </div>
 <form action = "">
  <p> <textarea rows = 6 cols = 61 id = "input" /> </p>
  <input type = "button" value = "Run"
         onClick = "run()" />
  <input type = "reset" value = "Clear" />
 </form>
 <pre id = "output">   </p>
 <pre id = "eval">   </p>
</body>
</html>
```

Using the `eval` Function 4



Document Object Models 1

When using JavaScript for web application programming elements of the HTML page must be accessed using a ***document object model***.

There are two such models available, the legacy DOM and the W3Schools DOM.

These do similar things but in different ways. The former is more concise but more awkward to use due to inconsistent naming conventions, whereas the latter has consistent naming conventions but requires the writing of more code.

Document Object Models 2

In the legacy DOM various elements of the HTML page are accessed using properties of the document object. For example `document.anchors` is an array of objects representing all of the `<a>` tags in the document. Other properties include `links`, `forms` and `images`.

The names of properties do ***not*** correspond to their HTML names so until you become familiar with them it is necessary to keep a reference guide constantly at hand to find the correct names to use.

Document Object Models 3

In the W3Schools DOM the HTML page elements are accessed by applying functions to the document object. The number of such functions is small and their arguments are strings which correspond precisely with the HTML element names.

For example

```
document.getElementById("uniqueId");
```

and

```
document.getElementsByTagName("a");
```

Document Object Models 4

The content of an HTML element can be accessed as a JavaScript string using the attribute **`innerHTML`**, e.g.

For example

```
document.getElementById("myDiv").innerHTML =
    "<p>Hello</p>";
```

Unless JavaScript is being used to generate simple HTML it is preferable to build the HTML as JavaScript objects which are then added to the DOM representation of the element.

New elements can be created using the **`createElement`** function which is applied to the document object; text caontent can be created using **`createTextNode`**.

Document Object Models 5

A newly-created element can be added to the document as a child of an existing element using **appendChild**:

```
var p = document.createElement("p");
document.getElementById("myDiv").appendChild(p);
p.appendChild(
        document.createTextNode("Hello"));
```

Attribute values of an HTML element can be retrieved using **getAttribute** and can be set using **setAttribute**:

```
p.setAttribute("class", "even");
```

A Sorting Example 1

We now present an example involving the sorting of arrays. Since the no-argument `sort` function of the `Array` class treats the elements of the array as strings and sorts them lexicographically, we need to use a version which takes a comparator function as an argument when sorting numbers. The example demonstrates the sorting of strings and both the lexicographical and numeric sorting of numbers.

To display the sorted and unsorted arrays results we set the text content of the presentation tags using the `setTextContent` function.

The screenshot on the slide following the code shows the display after the "buggy" (lexicographical) link was clicked.

A Sorting Example 2

```
<html>
 <head>
  <script language="JavaScript" type="text/javascript">
    function test(args, func)
    { var us = document.getElementById("unsorted");
      var s = document.getElementById("sorted");
      us.textContent = args.toString();
      if (func==null) args.sort();
      else args.sort(func);
      s.textContent = args.toString();
    }
    function numComp(a, b)
    { return a-b;
    }
  </script>
 </head>
```

A Sorting Example 3

```
<body>
 <ul> <li>
   <a href = "javascript:test(['Cherry', 'Apple',
     'Pear'])"> Strings </a>
  </li>
  <li>
   <a href = "javascript:test([10, 5, 20])">
      Numbers (buggy) </a>
  </li>
  <li>
   <a href = "javascript:test([10, 5, 20], numComp)">
       Numbers </a>
  </li> </ul>
 <pre id = "unsorted">   </pre>
 <pre id = "sorted">   </pre>
</body>
</html>
```

A Sorting Example 4

- Strings
- Integers (buggy)
- Integers (correct)

```
10,5,20
```

```
10,20,5
```

Using Multiple Windows 1

Our next example illustrates multi-window communication; a child window is created, filled with content, and talks back to the parent. Some of the content of the child window is included in the HTML for the parent but made invisible by setting the value of the `display` attribute to `none`. This content is then copied to the child window which is not invisible.

Note use of `window.open()` to create a new (child) window and the child window's use of the `opener` property to obtain the identity of its parent.

The `write` function is used to write HTML content to the new document; this cannot be used to modify the HTML of an existing page after it has been loaded.

Using Multiple Windows 2

```
<html>
 <head>
  <script language="JavaScript" type="text/javascript">
    function run()
    { var w = window.open();
      var d = w.document();;
      d.open();
      d.write("<h2> Hello World </h2>");
      var popup = document.getElementbyId("popup");
      d.write(popup.innerHTML);
      d.close();
    }
    function reply(a)
    { alert(a);
    }
  </script>
 </head>
```

Using Multiple Windows 3

```
 <body>
  <a href = "javascript:run()"> Open window 4</a>
  <div id = "popup" class = "invis"
       style = "{display:none}">
   <p> Is this invisible? </p>
   <a href = "javascript:opener.reply('yes'); close();">
    Yes </a>
   <a href = "javascript:opener.reply('no'); close();">
    No </a>
  </div>
  <div id = "popup2" style = "{display:block}>
   <p> Is this a block? </p>
  </div>
 </body>
</html>
```

jQuery

jQuery is a JavaScript framework that allows the document object model to be accessed and manipulated more concisely than using the methods already introduced.

To use jQuery a JavaScript file has to be accessed – this script is available from several sources:

```
<head>
  <script type="text/javascript"
          src="…/jquery/1.5.1/jquery.min.js"/>
  …
</head>
```

jQuery Selectors 1

A jQuery object represents one or more HTML elements from the document object model. There is a constructor that takes a string argument using notation based on the CSS selector syntax, so `jQuery("p")` would refer to the paragraph elements in the document, `jQuery("#x")` would refer to the element with `id` attribute `x` and `jQuery("tr.odd")` would refer to all table row elements with `class` attribute `odd`.

It is also possible to retrieve all elements that have a particular attribute: `jQuery("[name]")` would refer to all elements with a `name` attribute.

jQuery Selectors 2

It is possible to select elements that based on the value of an attribute : `jQuery("[name='x']")` would refer to any elements with a `name` attribute with value `x` and `jQuery("[name!='y']")` would refer to all elements with a `name` attribute whose value is not `y`.

The operators `^=`, `$=` and `*=` can be used to select elements with attribute values that begin with, end with or contain a specific string. For example `jQuery("a[href^='https']")` would refer to all `a` elements with a `href` attribute what begins with `https`.

jQuery Selectors 3

As in CSS it is possible to use nested tag names, so for example `jQuery("b i")` would refer to all `i` elements within `b` elements. It is also possible to search for elements that are immediate descendants: `jQuery("p>i")` would refer to all `i` elements that are direct children of `p` elements so if the document contained the following HTML fragment the first `i` tag would be found but the second would not (since it is nested within another tag).

```
<p>
This is <i>italic </i><br/>
This is <b>bold <i>italic</i></b>
</p>.
```

jQuery Selectors 4

To make code even more concise it is normally possible to use `$` instead of `jQuery`, (e.g. `$("tr.odd").css(…)`), but this is not guaranteed to work if other JavaScript is imported.

There is also a `jQuery` constructor that takes a document object as an argument; this would normally be called using the JavaScript variable `document` as this argument.

Invoking jQuery Code

JavaScript code written outside any functions in scripts included in HTML documents is invoked as the document is being loaded; when using jQuery we would normally wish to wait until the loading of the document has been completed as the selectors would otherwise find only elements that have already been loaded. We could do this using code in a function invoked using `<body onload = …>`, but jQuery provides another mechanism.

The `ready` method can be applied to the jQuery document object: it takes as an argument a JavaScript function to be called as soon as loading of the document has been completed. An example is shown on the next slide; in this case we supply the function inline but we could just supply the name of a function.

Invoking jQuery Code – an Example

```
<html>
  <head>
    <script type="text/javascript"
            src="…/jquery/1.5.1/jquery.min.js"/>
    <script type="text/javascript">
      $(document).ready(function()
      { $("tr.odd").css("color", "blue");
      } );
    </script>
  </head>
  <body> … </body>
</html>
```

Manipulating DOM Elements using JQuery 1

Many methods can be applied to jQuery objects to access and manipulate the DOM elements to which they refer. An example was seen on the previous slide – the `css` method can be used to change the style of elements.

There are three functions for accessing and changing the content of an element: `text` is used for textual content, `val` is used for values of form elements and `html` is used for textual representation of inner HTML. When called with no argument these methods retrieve the content, but when called with a string argument they change the content.

Manipulating DOM Elements using JQuery 2

Consider the following HTML fragment.

```
<p id = "i1" />
<div id = "i2" />
<form>
  <input type = "text" id = "i" name = "in"
         value = "input field" />
</form>
```

We could retrieve the contents of the input field using `$("#i").val()`, or set its contents using `$("#i").val("x")`.

We can set the textual content of the `p` element using `$("#i1").text("hello")`.

Manipulating DOM Elements using JQuery 3

We can insert HTML content into the `div` element on the previous slide using

```
$(#i2).html("<p> Hello <b>world</b>.</p>").
```

Note that if we used

```
$(#i2).text("<p> Hello <b>world</b>.</p>")
```

the string would be stored in the element as textual content so it would be equivalent to writing something like

```
<div id = "i2">
  &lt;p&gt; Hello &lt;b&gt;world
  &lt;/b&gt;.&lt;/p&gt;
</div>
```

Manipulating DOM Elements using JQuery 4

When called with string arguments the three functions for manipulating HMTL content return a reference to the jQuery object to which they are applied so we can chain them or apply other methods, e.g.

```
$("#i").val("x").css("color", "red");
```

The `attr` method can be used to get or set HTML attribute values. To retrieve the value of an attribute we use single a string argument, but to set the value we use two arguments, e.g.

```
alert($("#myid").attr("href"));
$("#myid").attr("href", "http://essex.ac.uk");
```

Manipulating DOM Elements using JQuery 5

The three methods `text`, `val` and `html` can also be called with a function as an argument and `attr` can be called with a function as its second argument. The value returned by the function will be used to set the content or attribute value of the element. For example `$("#i1").text(myfun)` will set the text content of the element to be the value returned by the function `myfun`. This function will be called with two arguments, the index of the element within the jQuery object (which may of course represent a list of DOM elements) and the current value. Since the number of arguments supplied to a JavaScript function does not have to match the number of arguments in the function declaration we can use a function with no arguments if we do not wish to use this information.

Manipulating DOM Elements using JQuery 6

There are methods `append` and `prepend` to add content to the document as a child of the selected element. The former will add the contents after any existing children, whereas the latter will add it before any existing children. The argument may be an HTML element created using `document.createElement`, a new HTML element created as a jQuery object using a constructor call of the form `$("<li></li>")` or a string representation of HTML. Multiple arguments may be supplied:

```
var x = $("<li></li>").text("Item 1");
$("#myList").append(x, "<li>Item 2</li>");
$("#myList").prepend(
                document.createElement("li"));
```

Manipulating DOM Elements using JQuery 7

There are methods `before` and `after` to add content to the document before or after the selected element. These take arguments in the same way as the methods on the previous slide.

The `remove` method will remove the selected element(s) from the document, e.g. `$("#x").remove()` will remove the element with `id` attribute `x`, and `$("i").remove()` will all `i` elements from the document. Any elements within the removed element(s) will also be removed.

The `empty` method will remove from the document all content within the selected element(s) but will not remove the element itself, or its attributes, e.g. `$("#conv").empty()` would remove any previous-generated conversion tables from the `div` element in the lab 2 exercise.