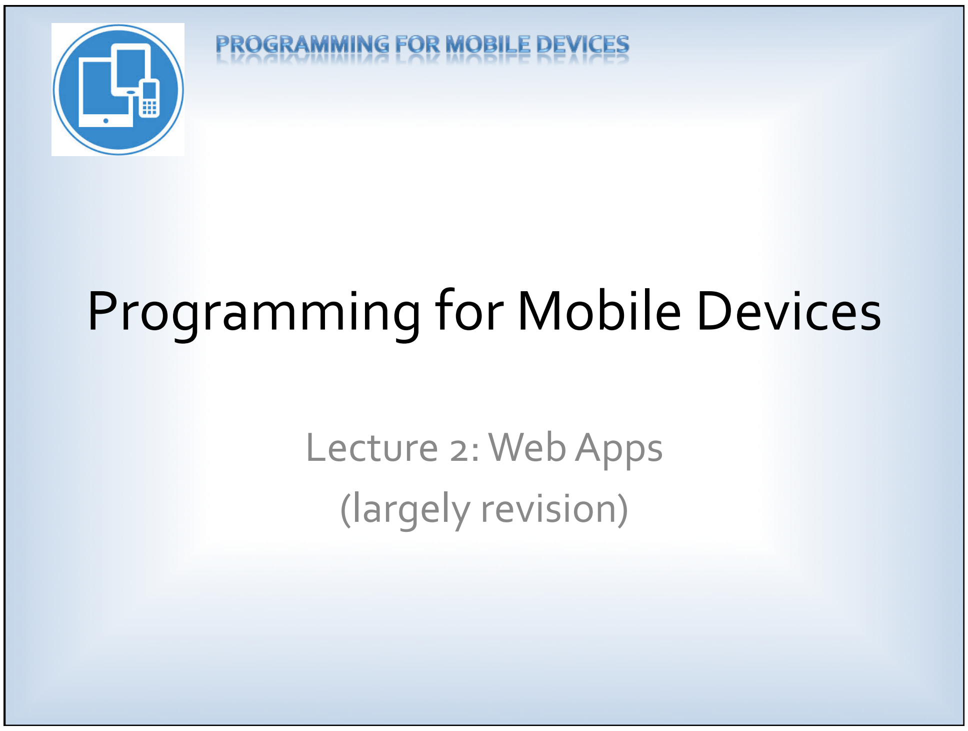
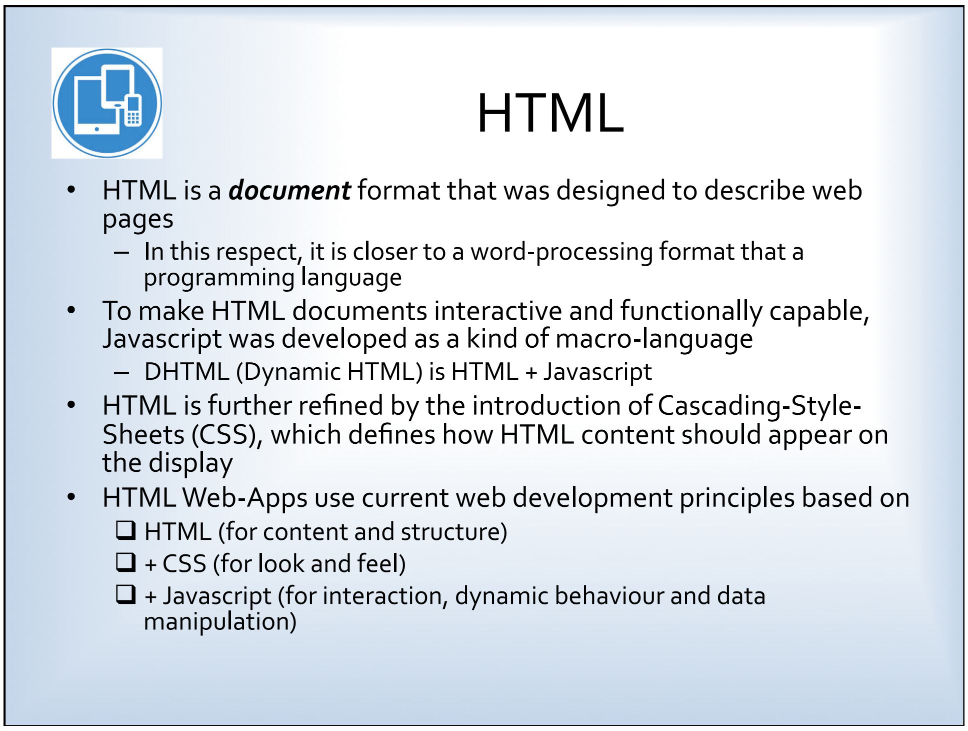


PROGRAMMING FOR MOBILE DEVICES

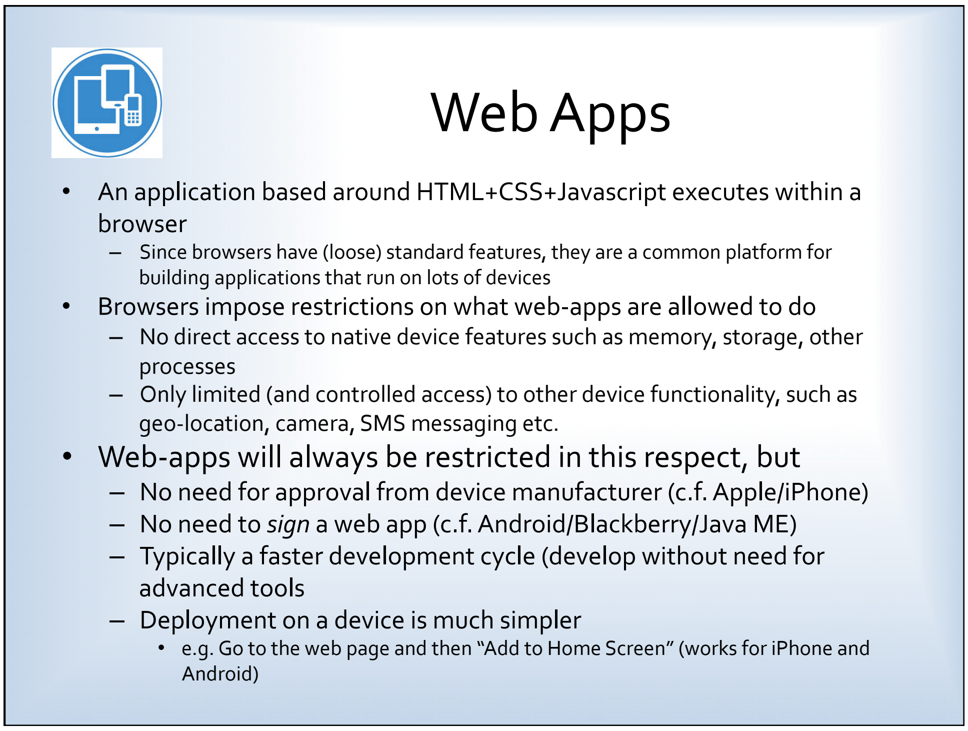
Programming for Mobile Devices

Lecture 2: Web Apps
(largely revision)



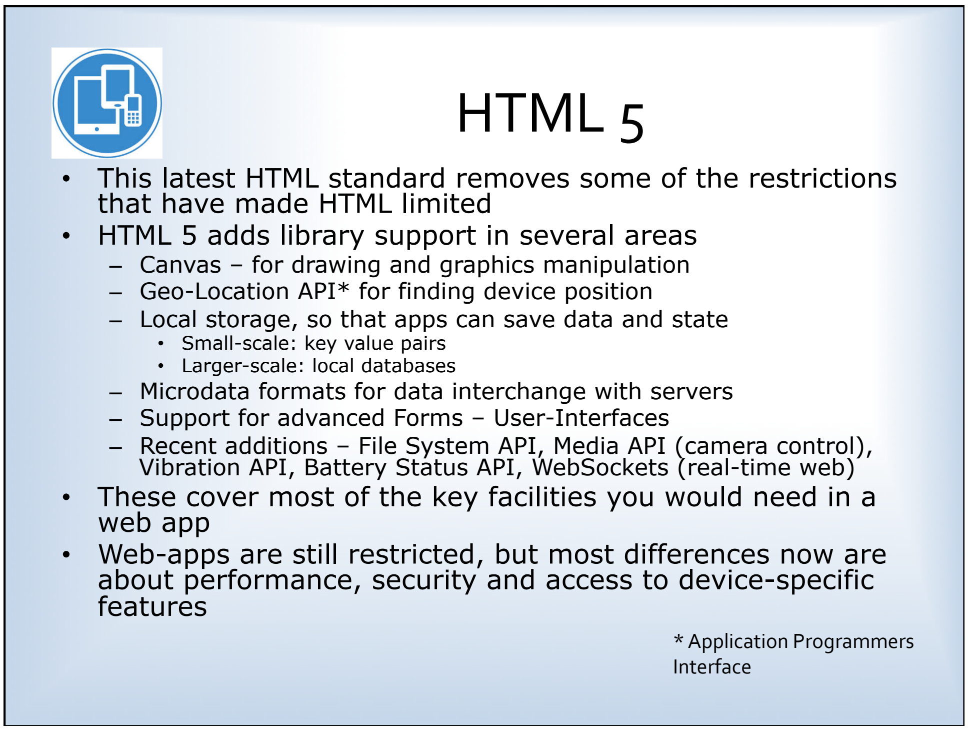
HTML

- HTML is a ***document*** format that was designed to describe web pages
  - In this respect, it is closer to a word-processing format that a programming language
- To make HTML documents interactive and functionally capable, Javascript was developed as a kind of macro-language
  - DHTML (Dynamic HTML) is HTML + Javascript
- HTML is further refined by the introduction of Cascading-Style-Sheets (CSS), which defines how HTML content should appear on the display
- HTML Web-Apps use current web development principles based on
  - ❑ HTML (for content and structure)
  - ❑ + CSS (for look and feel)
  - ❑ + Javascript (for interaction, dynamic behaviour and data manipulation)



Web Apps

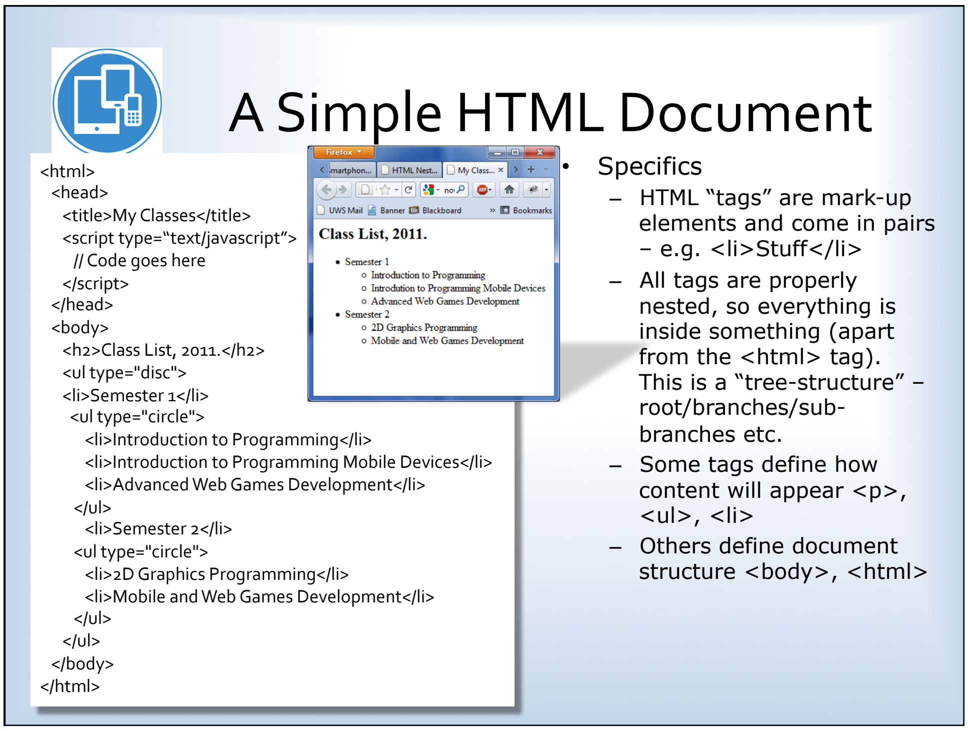
- An application based around HTML+CSS+Javascript executes within a browser
  - Since browsers have (loose) standard features, they are a common platform for building applications that run on lots of devices
- Browsers impose restrictions on what web-apps are allowed to do
  - No direct access to native device features such as memory, storage, other processes
  - Only limited (and controlled access) to other device functionality, such as geo-location, camera, SMS messaging etc.
- Web-apps will always be restricted in this respect, but
  - No need for approval from device manufacturer (c.f. Apple/iPhone)
  - No need to *sign* a web app (c.f. Android/Blackberry/Java ME)
  - Typically a faster development cycle (develop without need for advanced tools
  - Deployment on a device is much simpler
    - e.g. Go to the web page and then "Add to Home Screen" (works for iPhone and Android)



HTML 5

- This latest HTML standard removes some of the restrictions that have made HTML limited
- HTML 5 adds library support in several areas
  - Canvas – for drawing and graphics manipulation
  - Geo-Location API* for finding device position
  - Local storage, so that apps can save data and state
    - Small-scale: key value pairs
    - Larger-scale: local databases
  - Microdata formats for data interchange with servers
  - Support for advanced Forms – User-Interfaces
  - Recent additions – File System API, Media API (camera control), Vibration API, Battery Status API, WebSockets (real-time web)
- These cover most of the key facilities you would need in a web app
- Web-apps are still restricted, but most differences now are about performance, security and access to device-specific features

* Application Programmers Interface



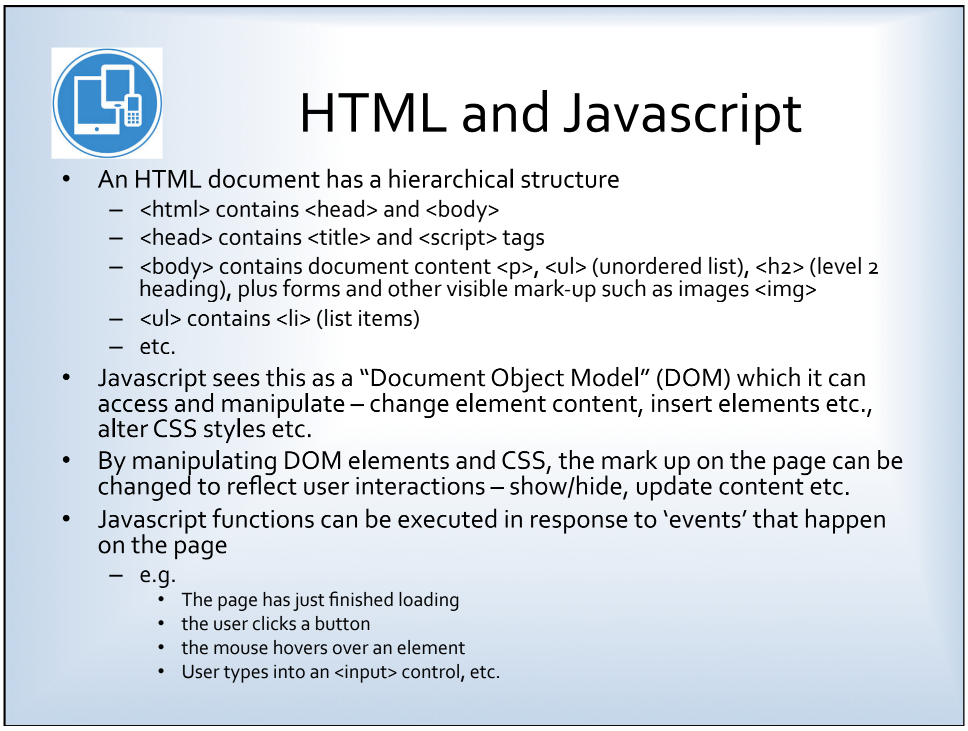
A Simple HTML Document

```
<html>
  <head>
    <title>My Classes</title>
    <script type="text/javascript">
      // Code goes here
    </script>
  </head>
  <body>
    <h2>Class List, 2011.</h2>
    <ul type="disc">
    <li>Semester 1</li>
      <ul type="circle">
        <li>Introduction to Programming</li>
        <li>Introduction to Programming Mobile Devices</li>
        <li>Advanced Web Games Development</li>
      </ul>
        <li>Semester 2</li>
      <ul type="circle">
        <li>2D Graphics Programming</li>
        <li>Mobile and Web Games Development</li>
      </ul>
    </ul>
  </body>
</html>
```

Class List, 2011.

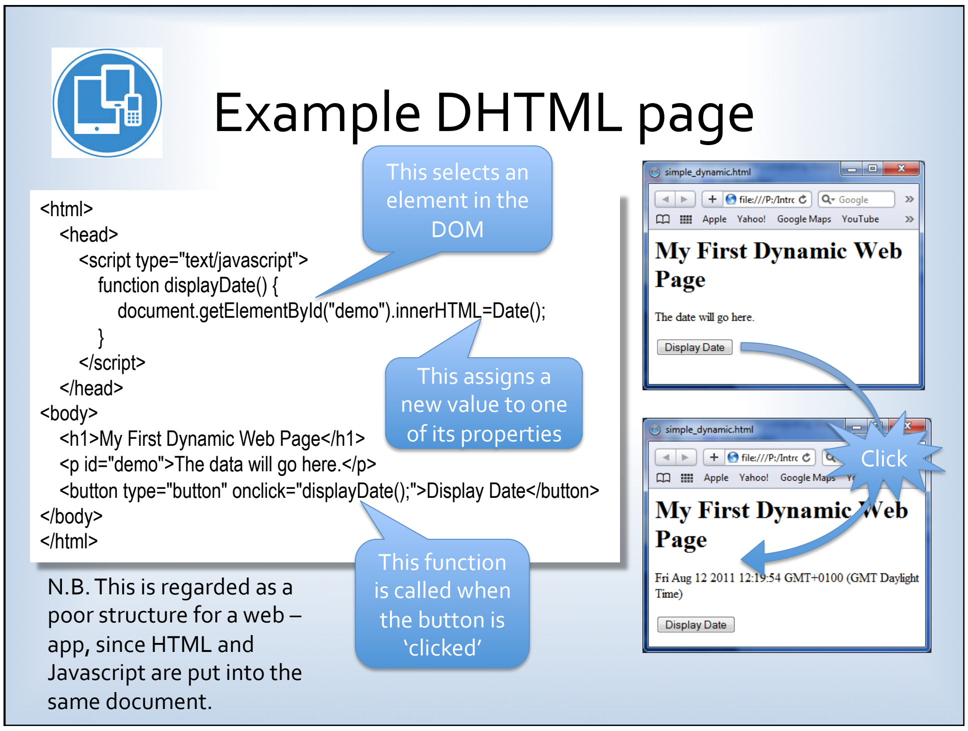
- Semester 1
  - Introduction to Programming
  - Introduction to Programming Mobile Devices
  - Advanced Web Games Development
- Semester 2
  - 2D Graphics Programming
  - Mobile and Web Games Development

- Specifics
  - HTML "tags" are mark-up elements and come in pairs – e.g. <li>Stuff</li>
  - All tags are properly nested, so everything is inside something (apart from the <html> tag). This is a "tree-structure" – root/branches/sub-branches etc.
  - Some tags define how content will appear <p>, <ul>, <li>
  - Others define document structure <body>, <html>



HTML and Javascript

- An HTML document has a hierarchical structure
  - <html> contains <head> and <body>
  - <head> contains <title> and <script> tags
  - <body> contains document content <p>, <ul> (unordered list), <h2> (level 2 heading), plus forms and other visible mark-up such as images <img>
  - <ul> contains <li> (list items)
  - etc.
- Javascript sees this as a "Document Object Model" (DOM) which it can access and manipulate – change element content, insert elements etc., alter CSS styles etc.
- By manipulating DOM elements and CSS, the mark up on the page can be changed to reflect user interactions – show/hide, update content etc.
- Javascript functions can be executed in response to 'events' that happen on the page
  - e.g.
    - The page has just finished loading
    - the user clicks a button
    - the mouse hovers over an element
    - User types into an <input> control, etc.



Example DHTML page

```
<html>
  <head>
    <script type="text/javascript">
      function displayDate() {
        document.getElementById("demo").innerHTML=Date();
      }
    </script>
  </head>
<body>
  <h1>My First Dynamic Web Page</h1>
  <p id="demo">The data will go here.</p>
  <button type="button" onclick="displayDate();">Display Date</button>
</body>
</html>
```

This selects an element in the DOM

This assigns a new value to one of its properties

This function is called when the button is 'clicked'

N.B. This is regarded as a poor structure for a web – app, since HTML and Javascript are put into the same document.

simple_dynamic.html

My First Dynamic Web Page

The date will go here.

Display Date

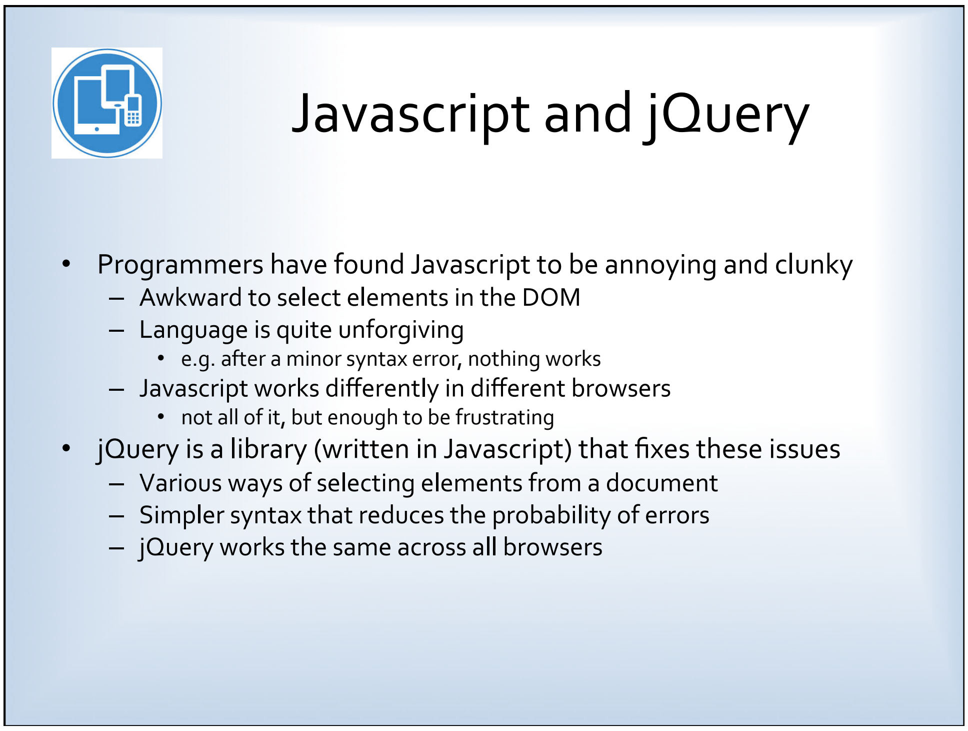
Click

simple_dynamic.html

My First Dynamic Web Page

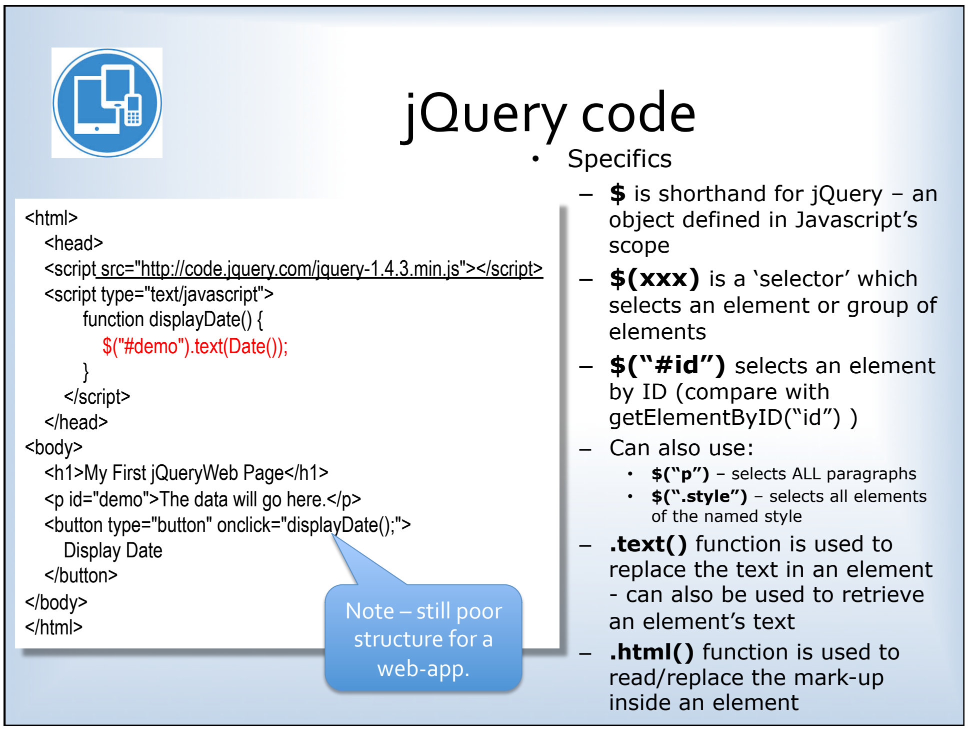
Fri Aug 12 2011 12:19:54 GMT+0100 (GMT Daylight Time)

Display Date



Javascript and jQuery

- Programmers have found Javascript to be annoying and clunky
  - Awkward to select elements in the DOM
  - Language is quite unforgiving
    - e.g. after a minor syntax error, nothing works
  - Javascript works differently in different browsers
    - not all of it, but enough to be frustrating
- jQuery is a library (written in Javascript) that fixes these issues
  - Various ways of selecting elements from a document
  - Simpler syntax that reduces the probability of errors
  - jQuery works the same across all browsers

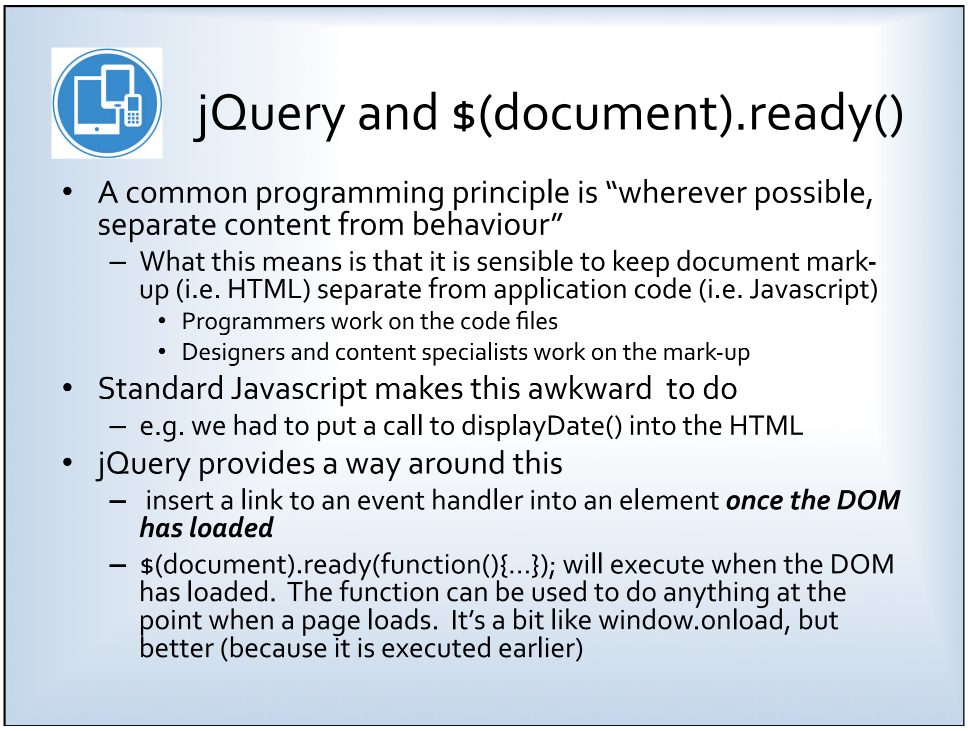


jQuery code

```
<html>
  <head>
  <script src="http://code.jquery.com/jquery-1.4.3.min.js"></script>
  <script type="text/javascript">
      function displayDate() {
        $("#demo").text(Date());
      }
    </script>
  </head>
<body>
  <h1>My First jQueryWeb Page</h1>
  <p id="demo">The data will go here.</p>
  <button type="button" onclick="displayDate();">
    Display Date
  </button>
</body>
</html>
```

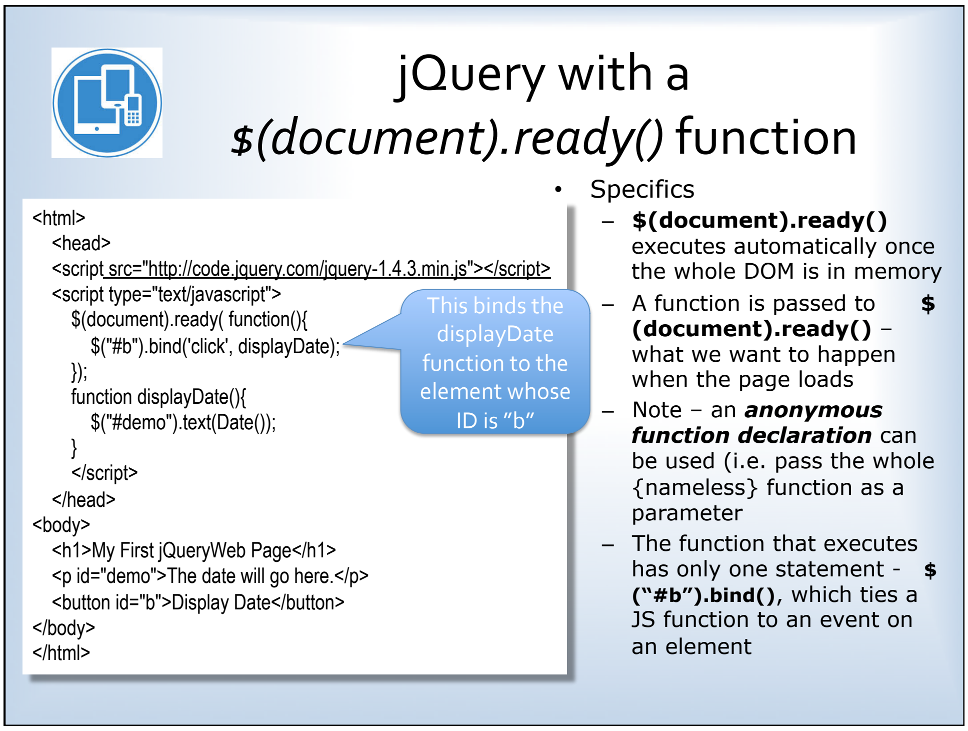
Note – still poor structure for a web-app.

- Specifics
  - **$** is shorthand for jQuery – an object defined in Javascript's scope
  - **$(xxx)** is a 'selector' which selects an element or group of elements
  - **$("#id")** selects an element by ID (compare with getElementByID("id"))
  - Can also use:
    - **$("p")** – selects ALL paragraphs
    - **$(".style")** – selects all elements of the named style
  - **.text()** function is used to replace the text in an element - can also be used to retrieve an element's text
  - **.html()** function is used to read/replace the mark-up inside an element



jQuery and $(document).ready()

- A common programming principle is "wherever possible, separate content from behaviour"
  - What this means is that it is sensible to keep document mark-up (i.e. HTML) separate from application code (i.e. Javascript)
    - Programmers work on the code files
    - Designers and content specialists work on the mark-up
- Standard Javascript makes this awkward to do
  - e.g. we had to put a call to displayDate() into the HTML
- jQuery provides a way around this
  - insert a link to an event handler into an element ***once the DOM has loaded***
  - $(document).ready(function(){…}); will execute when the DOM has loaded. The function can be used to do anything at the point when a page loads. It's a bit like window.onload, but better (because it is executed earlier)

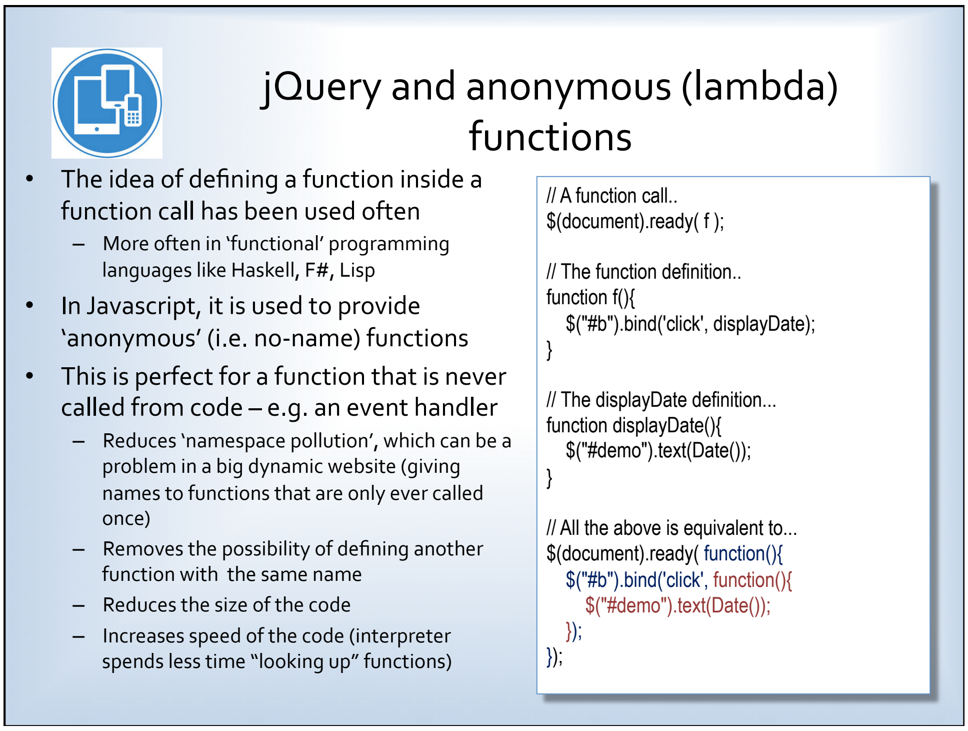


jQuery with a *$(document).ready()* function

```
<html>
   <head>
   <script src="http://code.jquery.com/jquery-1.4.3.min.js"></script>
   <script type="text/javascript">
      $(document).ready( function(){
         $("#b").bind('click', displayDate);
      });
      function displayDate(){
         $("#demo").text(Date());
      }
      </script>
   </head>
<body>
   <h1>My First jQueryWeb Page</h1>
   <p id="demo">The date will go here.</p>
   <button id="b">Display Date</button>
</body>
</html>
```

This binds the displayDate function to the element whose ID is "b"

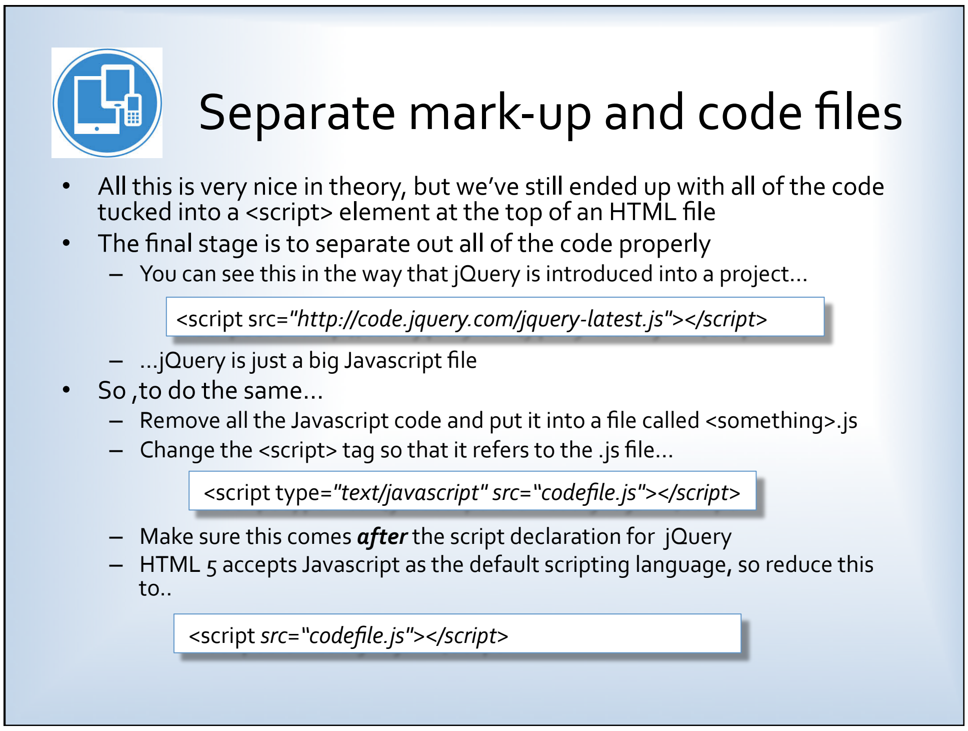
- Specifics
  - **$(document).ready()** executes automatically once the whole DOM is in memory
  - A function is passed to **$(document).ready()** – what we want to happen when the page loads
  - Note – an ***anonymous function declaration*** can be used (i.e. pass the whole {nameless} function as a parameter
  - The function that executes has only one statement - **$("#b").bind()**, which ties a JS function to an event on an element



jQuery and anonymous (lambda) functions

- The idea of defining a function inside a function call has been used often
  - More often in 'functional' programming languages like Haskell, F#, Lisp
- In Javascript, it is used to provide 'anonymous' (i.e. no-name) functions
- This is perfect for a function that is never called from code – e.g. an event handler
  - Reduces 'namespace pollution', which can be a problem in a big dynamic website (giving names to functions that are only ever called once)
  - Removes the possibility of defining another function with the same name
  - Reduces the size of the code
  - Increases speed of the code (interpreter spends less time "looking up" functions)

```
// A function call..
$(document).ready( f );

// The function definition..
function f(){
   $("#b").bind('click', displayDate);
}

// The displayDate definition...
function displayDate(){
   $("#demo").text(Date());
}

// All the above is equivalent to...
$(document).ready( function(){
   $("#b").bind('click', function(){
      $("#demo").text(Date());
   });
});
```



Separate mark-up and code files

- All this is very nice in theory, but we've still ended up with all of the code tucked into a <script> element at the top of an HTML file
- The final stage is to separate out all of the code properly
  - You can see this in the way that jQuery is introduced into a project…

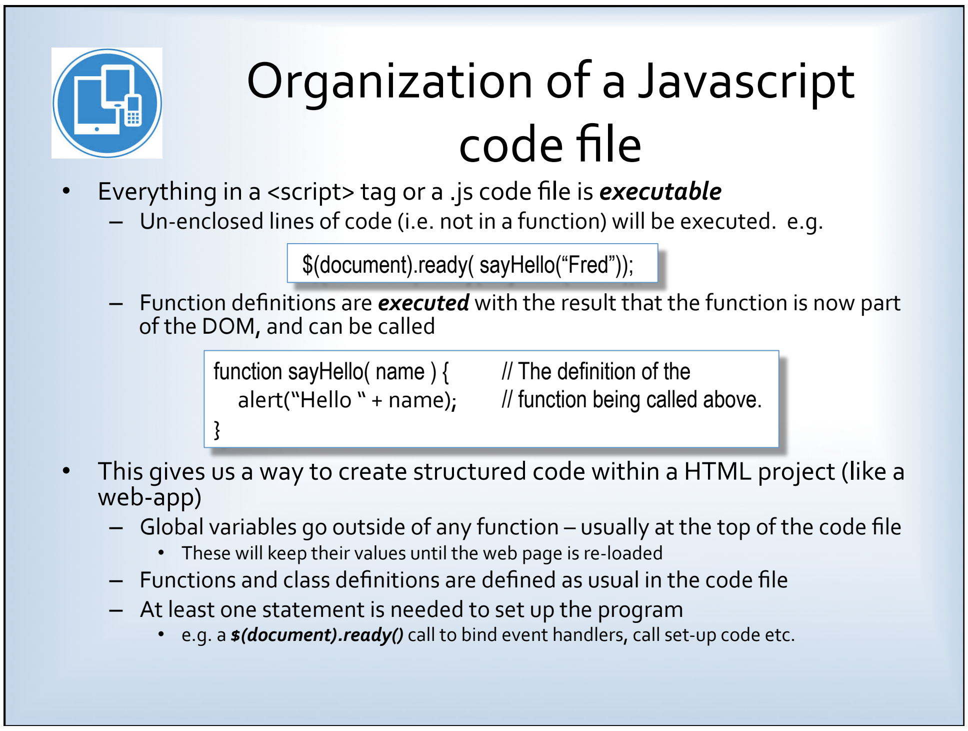
```
<script src="http://code.jquery.com/jquery-latest.js"></script>
```

  - …jQuery is just a big Javascript file
- So ,to do the same…
  - Remove all the Javascript code and put it into a file called <something>.js
  - Change the <script> tag so that it refers to the .js file…

```
<script type="text/javascript" src="codefile.js"></script>
```

  - Make sure this comes ***after*** the script declaration for jQuery
  - HTML 5 accepts Javascript as the default scripting language, so reduce this to..

```
<script src="codefile.js"></script>
```



Organization of a Javascript code file

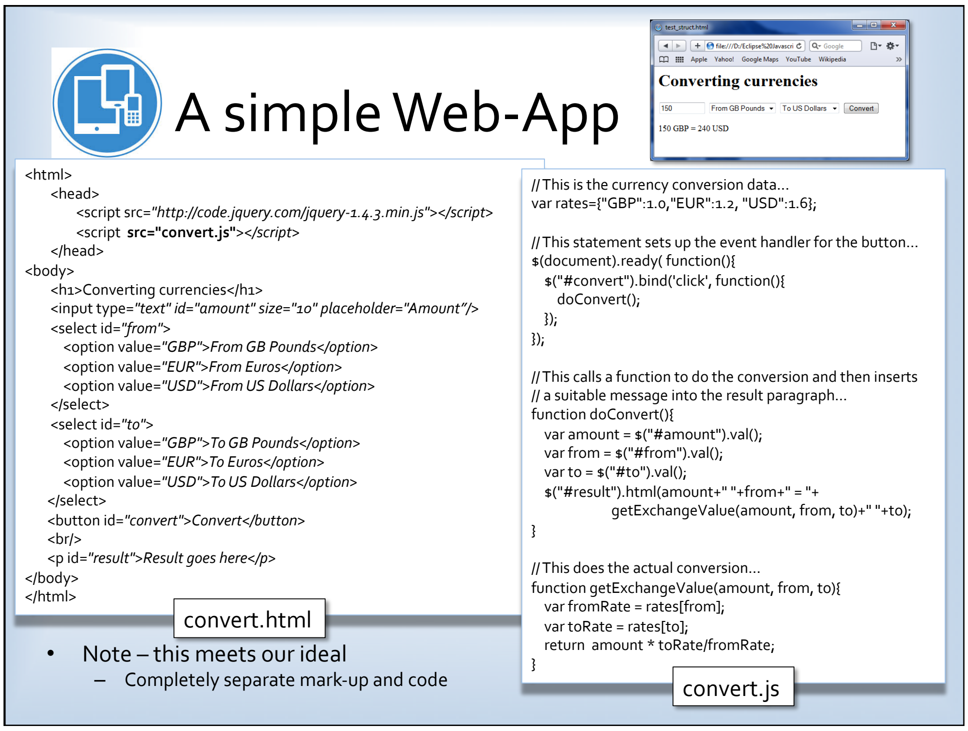
- Everything in a <script> tag or a .js code file is ***executable***
  - Un-enclosed lines of code (i.e. not in a function) will be executed. e.g.

```
$(document).ready( sayHello("Fred"));
```

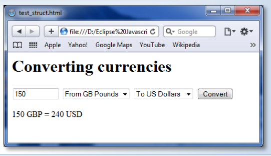
  - Function definitions are ***executed*** with the result that the function is now part of the DOM, and can be called

```
function sayHello( name ) {        // The definition of the
    alert("Hello " + name);        // function being called above.
}
```

- This gives us a way to create structured code within a HTML project (like a web-app)
  - Global variables go outside of any function – usually at the top of the code file
    - These will keep their values until the web page is re-loaded
  - Functions and class definitions are defined as usual in the code file
  - At least one statement is needed to set up the program
    - e.g. a ***$(document).ready()*** call to bind event handlers, call set-up code etc.



A simple Web-App



Converting currencies

150 From GB Pounds To US Dollars Convert

150 GBP = 240 USD

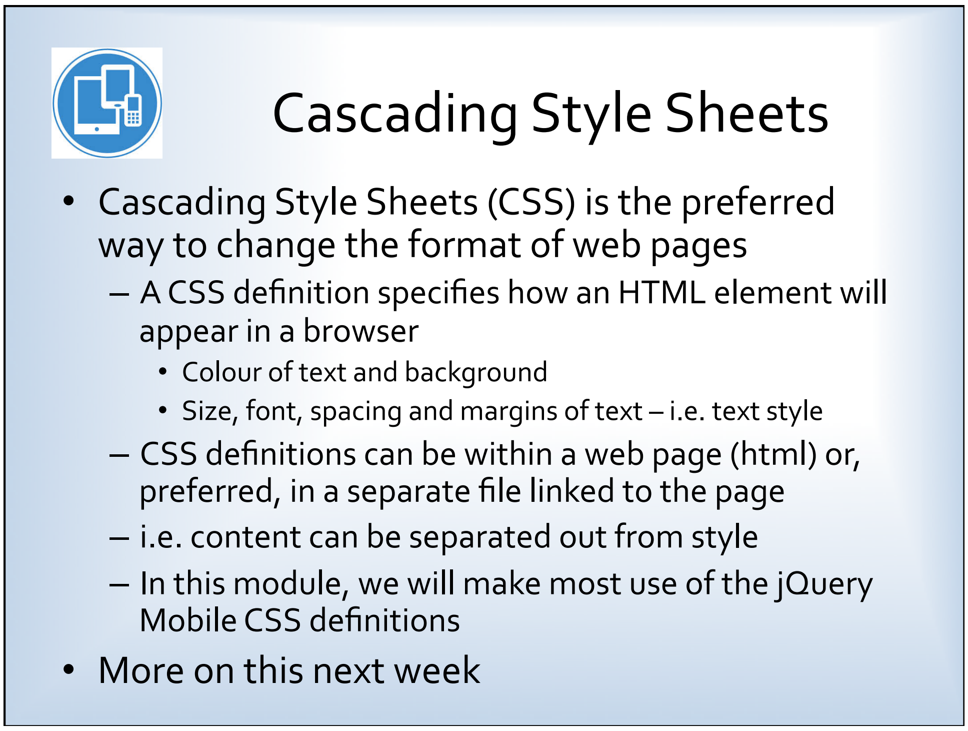
```
<html>
    <head>
        <script src="http://code.jquery.com/jquery-1.4.3.min.js"></script>
        <script  src="convert.js"></script>
    </head>
<body>
    <h1>Converting currencies</h1>
    <input type="text" id="amount" size="10" placeholder="Amount"/>
    <select id="from">
        <option value="GBP">From GB Pounds</option>
        <option value="EUR">From Euros</option>
        <option value="USD">From US Dollars</option>
    </select>
    <select id="to">
        <option value="GBP">To GB Pounds</option>
        <option value="EUR">To Euros</option>
        <option value="USD">To US Dollars</option>
    </select>
    <button id="convert">Convert</button>
    <br/>
    <p id="result">Result goes here</p>
</body>
</html>
```

convert.html

```
//This is the currency conversion data...
var rates={"GBP":1.0,"EUR":1.2, "USD":1.6};

//This statement sets up the event handler for the button...
$(document).ready( function(){
   $("#convert").bind('click', function(){
      doConvert();
   });
});

//This calls a function to do the conversion and then inserts
// a suitable message into the result paragraph...
function doConvert(){
   var amount = $("#amount").val();
   var from = $("#from").val();
   var to = $("#to").val();
   $("#result").html(amount+" "+from+" = "+
                getExchangeValue(amount, from, to)+" "+to);
}

//This does the actual conversion...
function getExchangeValue(amount, from, to){
   var fromRate = rates[from];
   var toRate = rates[to];
   return  amount * toRate/fromRate;
}
```

convert.js

- Note – this meets our ideal
  - Completely separate mark-up and code



Cascading Style Sheets

- Cascading Style Sheets (CSS) is the preferred way to change the format of web pages
  - A CSS definition specifies how an HTML element will appear in a browser
    - Colour of text and background
    - Size, font, spacing and margins of text – i.e. text style
  - CSS definitions can be within a web page (html) or, preferred, in a separate file linked to the page
  - i.e. content can be separated out from style
  - In this module, we will make most use of the jQuery Mobile CSS definitions
- More on this next week