The University Of Sheffield.

COM6517

Web Technologies

HTML and Javascript

Prof. Fabio Ciravegna
Department of Computer Science
f.ciravegna@shef.ac.uk

From the COM1004 slides by Dr. Steve Maddock and Dr. Mike Stannett

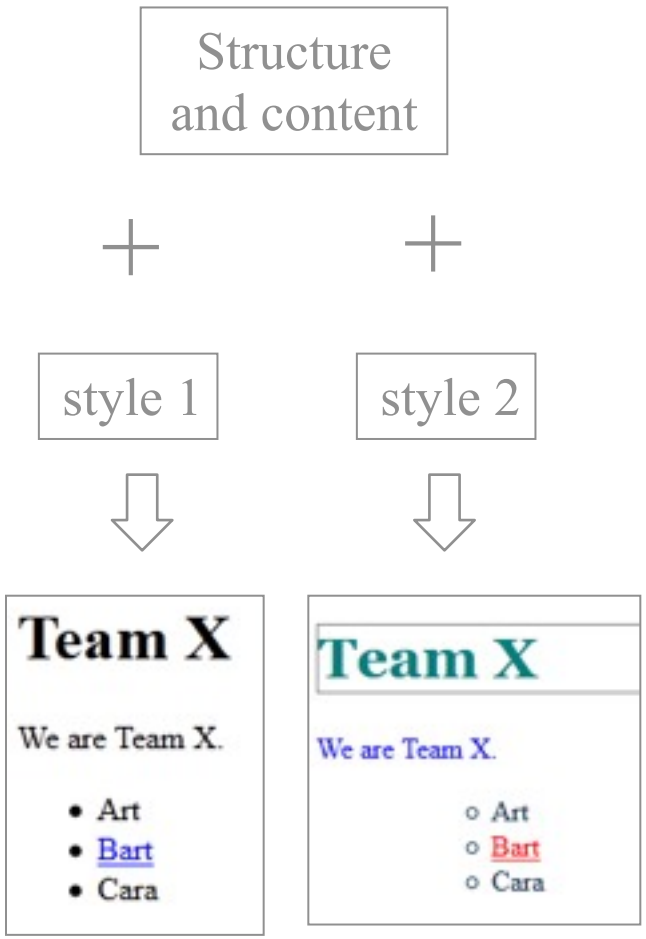
1. Introduction

- When creating a Web page, separate the structure and the appearance
- Structure is indicated using HTML
- HyperText Markup Language

```
<h1>Team X</h1>
```

- Appearance is controlled using CSS
- Cascading Style Sheet

```
h1 {
  color: teal;
}
```



3. Elements, attributes and values

- A general document is made up of elements
- An element:
  - **<el> content </el>**

start tag	mix of text and elements	matching end tag

  - Empty element: **<el />**
- Attributes are named properties of elements
- Attributes are assigned values in elements' start tags, using an = sign

```
<h1>Team X</h1>
  <p>We are Team X.</
p>
  <ul>
    <li>Art</li>
    <li>Bart</li>
    <li>Cara</li>
  </ul>
</body>

</html>
```

```
<li><a href="http://www.thesimpsons.com/">Bart</a></li>
```

element attribute value

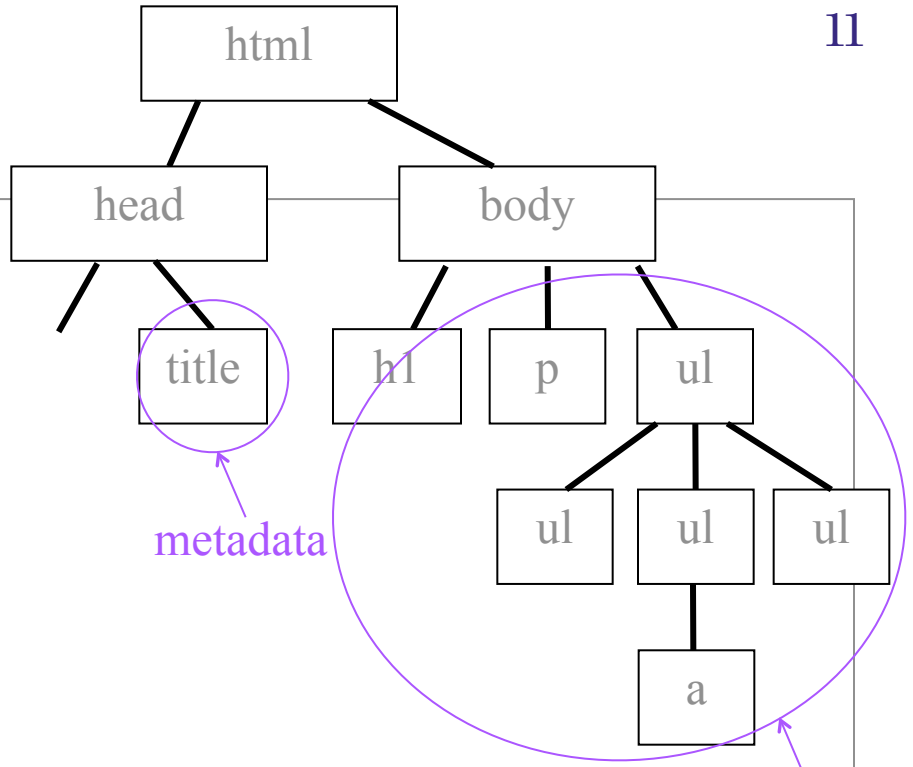
4.1 A sample document

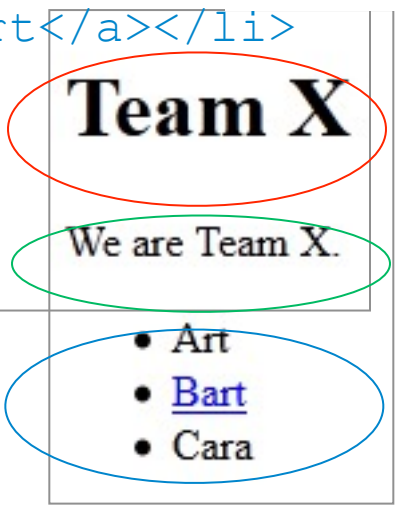
(notice the indentation)

```
<!DOCTYPE html>
<html lang="en">

<head>
  <meta charset="utf-8" />
  <title>Team X</title>
</head>

<body>
  <h1>Team X</h1>
  <p>We are Team X.</p>
  <ul>
    <li>Art</li>
    <li><a href="http://www.thesimpsons.com/">Bart</a></li>
    <li>Cara</li>
  </ul>
</body>

</html>
```

4.3 Document structure

```
<!DOCTYPE html>
<html lang="en">

<head>
  <meta charset="utf-8" />
  <title>Team X</title>
</head>

<body>
  <h1>Team X</h1>
  <p>We are Team X.</p>
  <ul>
    <li>Art</li>
    <li><a href="http://www.thesimpsons.com/">Bart</a></li>
    <li>Cara</li>
  </ul>
</body>

</html>
```





4.4 The doctype and the head

- Specifying the doctype triggers browsers that need it to operate in html standards mode
- The root level of the document is the html element
- The html element has a language attribute
  - en = English

More complex for XHTML

```
<!DOCTYPE html>
<html lang="en">

<head>
  <meta charset="utf-8" />
  <title>Team X</title>
</head>

<body>
<!-- content -->
</body>

</html>
```

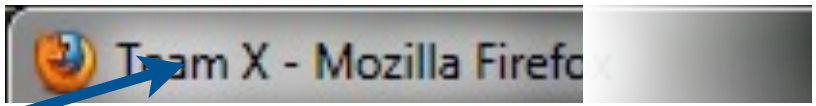
A comment

```
<HTML lang="fr">
<HEAD>
<TITLE>Un document multilingue</TITLE>
</HEAD>
```

www.w3.org/TR/html4/struct/dirlang.html

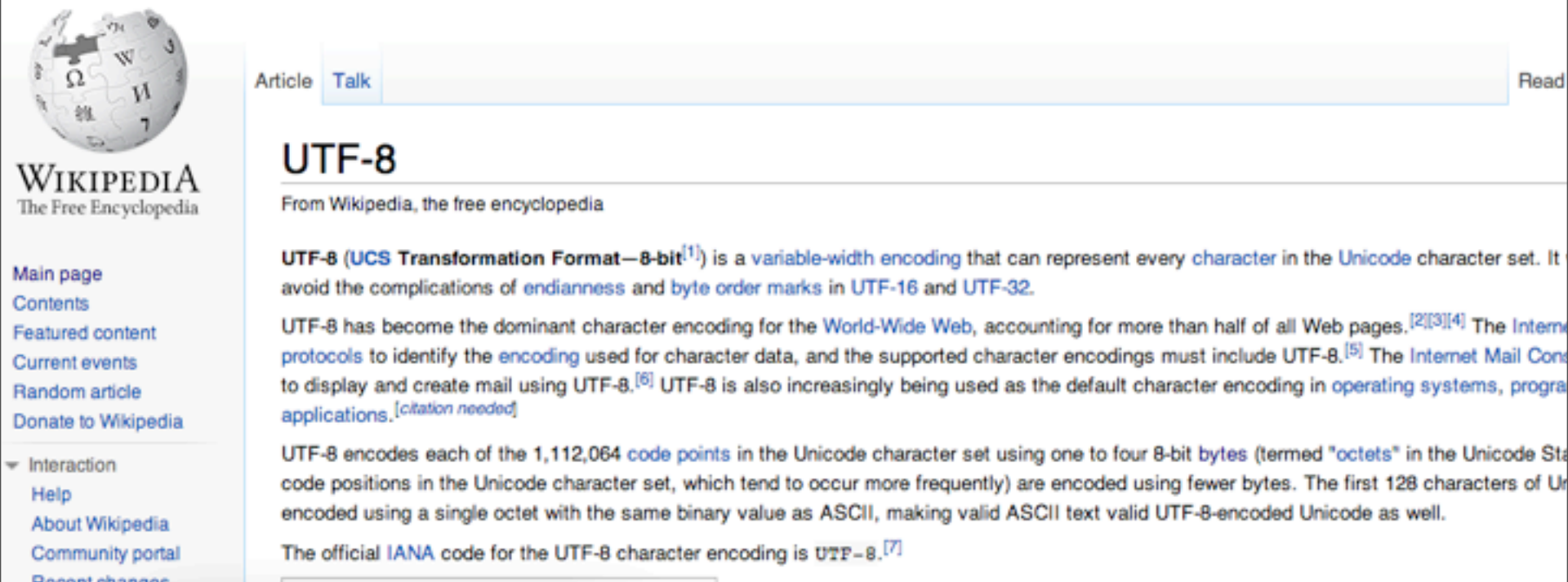
4.5 The document head

- The content of the head element is not rendered in the browser window
- The title element is compulsory and is displayed in the title bar
- The meta element provides a general-purpose mechanism for adding metadata to HTML documents
- charset defines the document's character encoding
  - Security risk of not setting it
  - Must be in first 512 bytes
  - Multibyte character encoding for Unicode.




```
<!DOCTYPE html>
<html lang="en">

<head>
  <meta charset="utf-8" />
  <title>Team X</title>
</head>

<body>
<!-- content -->
</body>

</html>
```

UTF-8: definition



WIKIPEDIA
The Free Encyclopedia

Main page
Contents
Featured content
Current events
Random article
Donate to Wikipedia

Interaction
Help
About Wikipedia
Community portal

Article Talk Read

UTF-8

From Wikipedia, the free encyclopedia

UTF-8 (**UCS Transformation Format—8-bit**[1]) is a variable-width encoding that can represent every character in the Unicode character set. It avoid the complications of endianness and byte order marks in UTF-16 and UTF-32.

UTF-8 has become the dominant character encoding for the World-Wide Web, accounting for more than half of all Web pages.[2][3][4] The Interne protocols to identify the encoding used for character data, and the supported character encodings must include UTF-8.[5] The Internet Mail Cons to display and create mail using UTF-8.[6] UTF-8 is also increasingly being used as the default character encoding in operating systems, progra applications.[*citation needed*]

UTF-8 encodes each of the 1,112,064 code points in the Unicode character set using one to four 8-bit bytes (termed "octets" in the Unicode Sta code positions in the Unicode character set, which tend to occur more frequently) are encoded using fewer bytes. The first 128 characters of Un encoded using a single octet with the same binary value as ASCII, making valid ASCII text valid UTF-8-encoded Unicode as well.

The official IANA code for the UTF-8 character encoding is `UTF-8`.[7]

http://en.wikipedia.org/wiki/UTF-8

4.5 The document head

- Other metadata elements use name and content attributes
- Other elements
  - link – stylesheets (see later in this lecture)
  - script – JavaScript (see a later lecture)

```
<head>
  <meta charset="utf-8" />
  <title>Team X</title>
  <link rel="stylesheet" href="teamx1.css" />
  <meta name="author" content="Steve Maddock" />
  <meta name="description" content="Team X web site for COM1004" />
  <meta name="keywords" content="Team X, sports" />
</head>
```

4.5 The document head

```
<meta name="robots" content="index, nofollow" />
```

Not very popular with spiders who tend to ignore it

Keyword	Meaning
index	This document may be indexed.
noindex	This document may be not indexed.
follow	Links from this document may be followed.
nofollow	Links from this document may not be followed.
all	This document may be indexed and links from it may be followed.
none	This document may not be indexed and links from it may not be followed.

Chapman, N and J. Chapman, Web Design: A complete introduction, John Wiley & Sons, 2006.

4.6 The body and some other elements

```
<body>
  <h1>Team X</h1>
  <p>We are Team X.</p>
  <ul>
    <li>Art</li>
    <li><a href="http://www.thesimpsons.com/">Bart</a></li>
    <li>Cara</li>
  </ul>
</body>
```

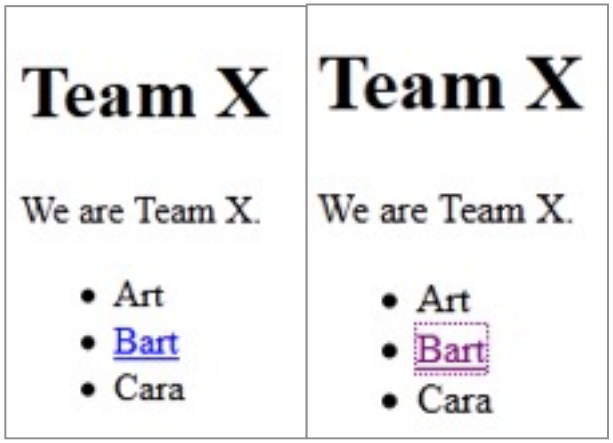
- Six heading elements: **h1**, h2, h3, h4, h5, h6
- List types:
  - Unordered lists (**ul**); items (**li**)
  - Ordered lists (ol); items (li)
  - Definition lists (dl); items (dt term and dd definition)
- The hyperlink (anchor) element is **a**

4.7 Hyperlinks

- The ends of hyperlinks are called *anchors*
  - *They link a source* and a *destination*

```
<a href="URL">link text</a>
```

- Default display: blue and underlined
- Once visited: purple and underlined




```
<body>
  <h1>Team X</h1>
  <p>We are Team X.</p>
  <ul>
    <li>Art</li>
    <li><a href="http://www.thesimpsons.com/">Bart</a></li>
    <li>Cara</li>
  </ul>
</body>
```

4.7.1 Relative URLs

- Orange gives the relative filenames from the start position: index.html
- From within index.html :

```
<a href="../feedback.html">feedback</a>
```

The '..' refers to the next folder level up in the hierarchy

Absolute URL : feedback

A Different starting position can be given with the BASE element
Put the <base> tag as the first element inside the <head> element, so that other elements in the head section uses the information from the <base> element.
<base href="http://www.someurl.org/" target="_blank">

4.7.1 Relative URLs

- Useful because can easily move whole Web site to a different host machine, as the links are relative
- Fragment identifier can be used to link to a location within a document:

```
<a href="../feedback.html#Comments">Send a comment</a>
```

- The location in the document feedback.html is identified with an id:

```
<h1 id="Comments">Your comments</h1>
```

- Implicit destination anchor at the start of every document

```
<a href="#">Jump to top of page</a>
```

4.7 Block and inline elements

- How does the browser know when to start a new line?


```
<body>
  <h1>Team X</h1>
  <p>We are Team X.</p>
  <p>We welcome new members.</p>
  <ul>
    <li>Art</li>
    <li><a href="http://www.thesimpsons.com/">Bart</a> -
Club Captain</li>
    <li>Cara</li>
  </ul>
  <p>Designed by <em>a web designer</em>, 2011</p>
</body>
```

4.7 Block and inline elements

- *Block* elements
  - Begin on new lines
  - Examples: p, div, ul, li, table, h1, h2, h3, h4, h5, h6, hr
- *Inline* elements
  - Displayed within blocks
  - Examples: a, img, span, em, strong, code, b, i, big, small, br, cite,
  - (physical appearance, e.g. i, tt; logical appearance, e.g. em)
  -
 - force a line break

- Note:
- HTML5 uses more content categories
  - Metadata, Flow (similar to block), Sectioning, Heading, Phrasing (similar to inline), Embedded, Interactive, Form-associated, Transparent
  - See https://developer.mozilla.org/en/HTML/Content_categories

5.1 "Gluing" a stylesheet to a document


```
<!DOCTYPE html>
<html lang="en">

<head>
  <meta charset="utf-8" />
  <title>Team X</title>
  <link rel="stylesheet" href="teamx1.css" />
</head>

<body>
  <h1>Team X</h1>
  <p>We are Team X.</p>
  <ul>
    <li>Art</li>
    <li><a href="http://www.thesimpsons.com/">Bart</
a></li>
    <li>Cara</li>
  </ul>
</body>

</html>
```

5.3 CSS rules

- A stylesheet is a set of *rules*
- Example rule: change h1 text colour to teal

```
Selector { Declaration; }
```

```
h1 {  color: teal; }
```

```
Selector { Property: Value; }
```

- This applies to all occurrences of the h1 element

- Multiple declarations separated by semicolons
- If property value has a space, use quotes:

```
h1 {
  font-family: "Lucida Handwriting", Papyrus, serif;
}
```

5.2 The CSS


```
<!DOCTYPE html>
<html lang="en">

<head>
  <meta charset="utf-8" />
  <title>Team X</title>
  <link rel="stylesheet" href="teamx1.cs
</head>

<body>
  <h1>Team X</h1>
  <p>We are Team X.</p>
  <ul>
    <li>Art</li>
    <li><a href="http://
www.thesimpsons.com/">Bart</a></li>
    <li>Cara</li>
  </ul>
</body>

</html>
```

teamx1.css

```
h1 {
  color: teal;
  font-family: Georgia, serif;
  font-size: 200%;
}

p {
  color: blue;
}

ul {
  padding-left: 100px;
  list-style-type: circle;
}

li {
  color: #123456; /* hexadecimal
}

a {
  color: red;
}
```

A comment

5.4 Typography

15.3 Font family: the 'font-family' property

'font-family'

Value:	[[<family-name> | <generic-family>] [, <family-name>| <generic-family>]*] | inherit
Initial:	depends on user agent
Applies to:	all elements
Inherited:	yes
Percentages:	N/A

Always include
a generic family

```
h1 {
  font-family: Papyrus, serif;
}
```

<generic-family>

In the example above, the last value is a generic family name.

- 'serif' (e.g., Times)
- 'sans-serif' (e.g., Helvetica)
- 'cursive' (e.g., Zapf-Chancery)
- 'fantasy' (e.g., Western)
- 'monospace' (e.g., Courier)

http://www.w3.org/TR/CSS2/fonts.html

5.2 The CSS

```
<!DOCTYPE html>
<html lang="en">

<head>
  <meta charset="utf-8" />
  <title>Team X</title>
  <link rel="stylesheet"
href="teamx1.css" />
</head>

<body>
  <h1>Team X</h1>
  <p>We are Team X.</p>
  <ul>
    <li>Art</li>
    <li><a href="http://
www.thesimpsons.com/">Bart</a></li>
    <li>Cara</li>
  </ul>
</body>

</html>
```

teamx1.css

```
h1 {
  color: teal;
  font-family: Georgia,
serif;
  font-size: 200%;
}

p {
  color: blue;
}

ul {
  padding-left: 100px;
  list-style-type: circle;
}

li {
  color: #123456; /*
hexadecimal */
}

a {
  color: red;
}
```

A comment

5.6 Alternative ways to "glue" a stylesheet to a document

- Link to a stylesheet which is in its own file: ← Preferred option

```
<head> ...
  <link rel="stylesheet" href="teamx1.css" />
</head>
```

- 'Old style':

```
<head> ...
  <link rel="stylesheet" type="text/css" href="teamx1.css" />
</head>
```

- Embed it in the head element:

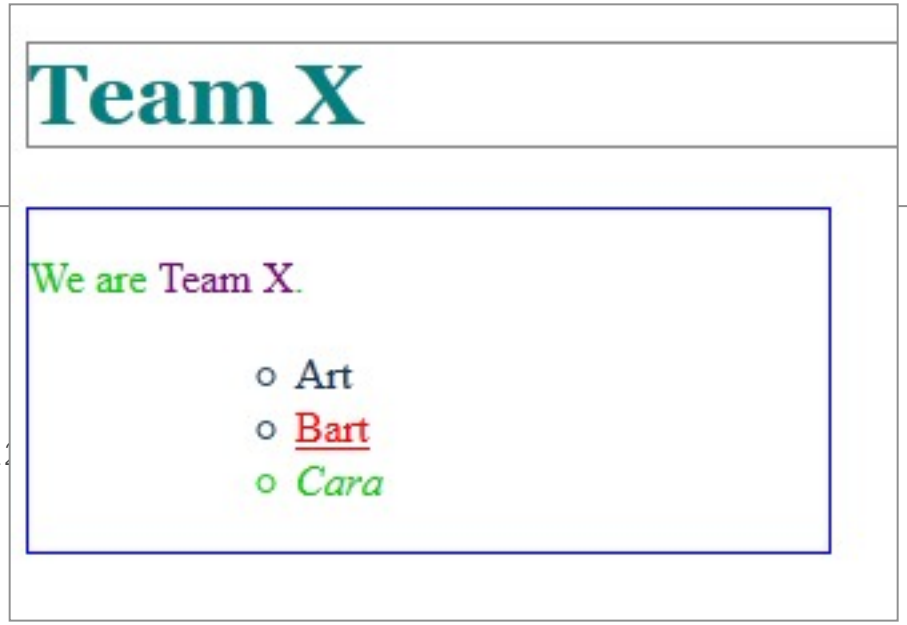
```
<head> ...
  <style> ...style information goes here... </style>
</head>
```

- Inline it:

```
<h1 style="color: teal;">My heading</h1>
```

  - This will only change the current h1, not all of them!

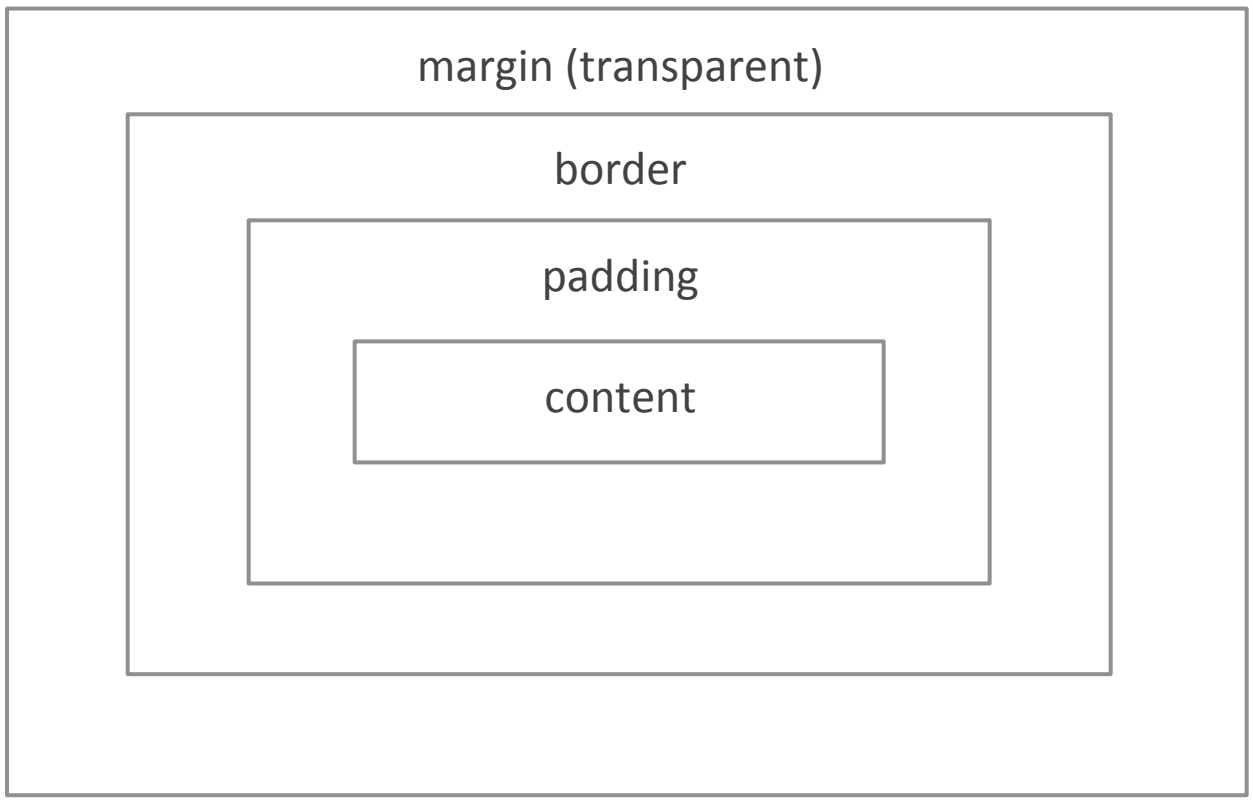
6. The mighty div




```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="utf-8" />
  <title>Team X</title>
  <link rel="stylesheet" href="teamx
</head>

<body>
  <h1>Team X</h1>
  <div id="main">
    <p class="green">We are <span class="purple">Team X</
span>.</p>
    <ul>
      <li>Art</li>
      <li><a href="http://www.thesimpsons.com/">Bart</a></li>
      <li class="italic green">Cara</li>
    </ul>
  </div> <!-- main -->
</body>
</html>
```

6.2 The Box Model

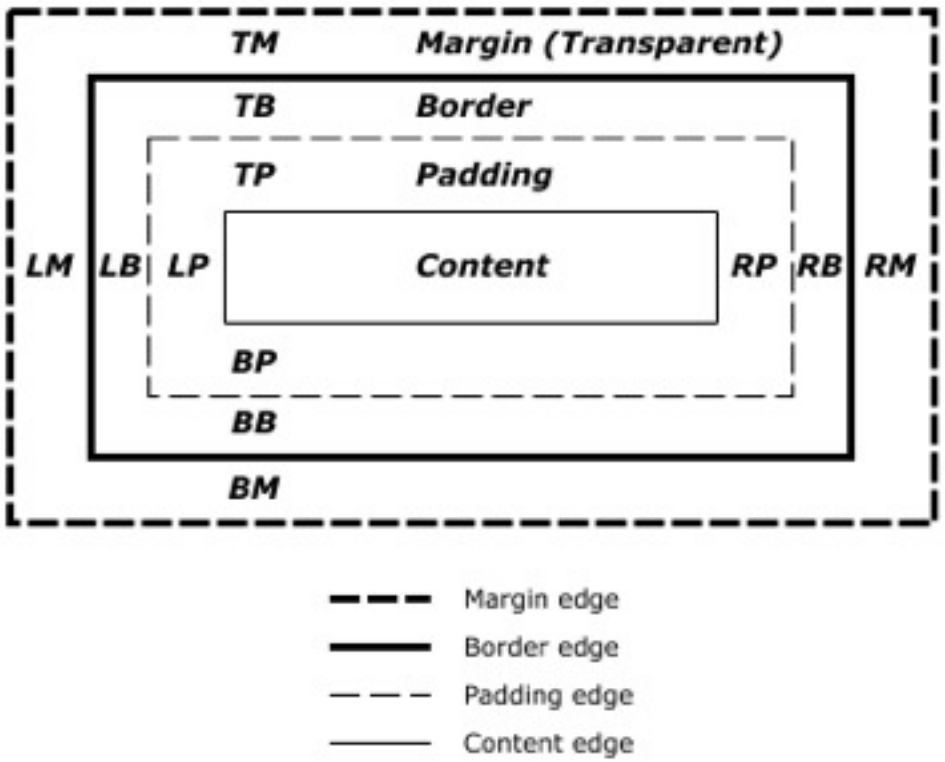
- Every element (content) is placed inside a box
- Each box region may have a thickness of zero



6.2.2 Space around boxes – margin, border, padding

- Margins are transparent
  - Beware collapsing margins – depends on padding and border values
- Padding
  - Takes on same appearance as an element's background
- Border
  - Draws a border of finite thickness around an element
- An element has width and height attributes

http://www.w3.org/TR/CSS2/box.html



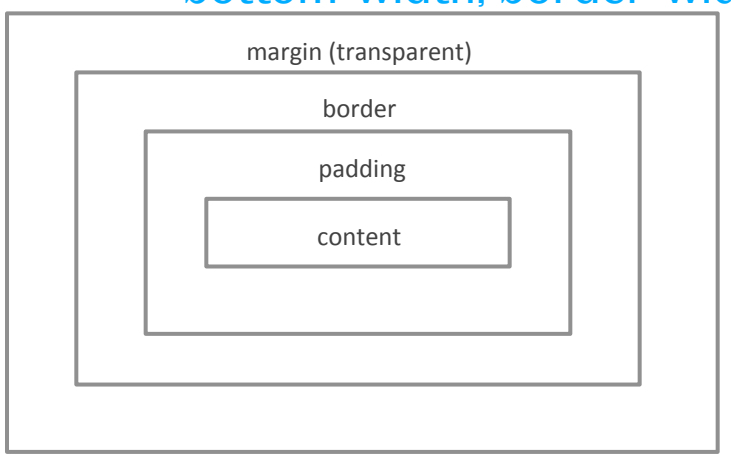
Total width = margin-left + border-left-width + padding-left + 'element width'
+ padding-right + border-right-width + margin-right

6.2.3 Properties

- margin-top, margin-right, margin-bottom, margin-left
  - *length* | *percentage* | auto
  - `p { margin-top: 2em; }`
- Set all at once with:
  - `body { margin: 1em 2em 3em 2em; }`

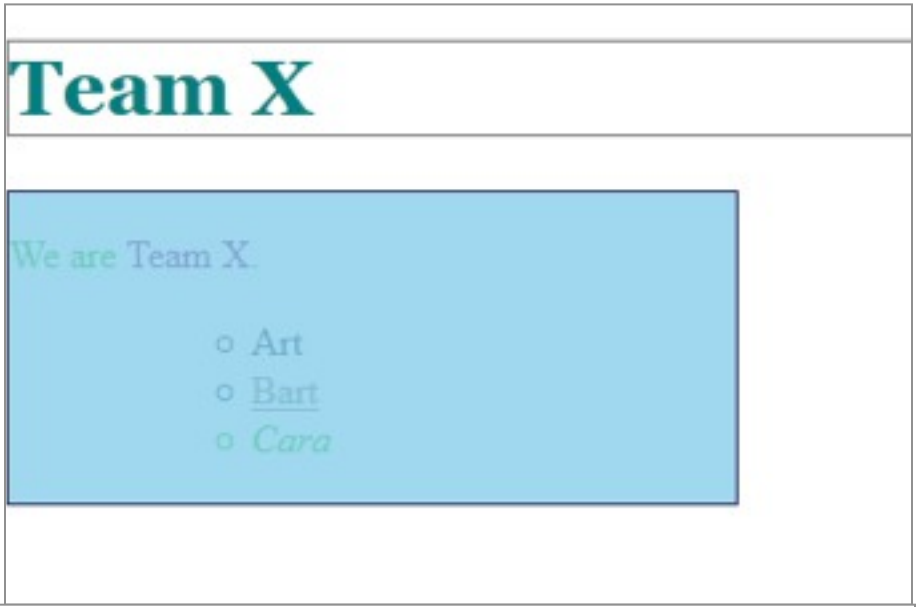
- padding-top, padding-right, padding-bottom, padding-left
  - *length* | *percentage*
- Set all at once with:
  - `body { padding: 2em; }`

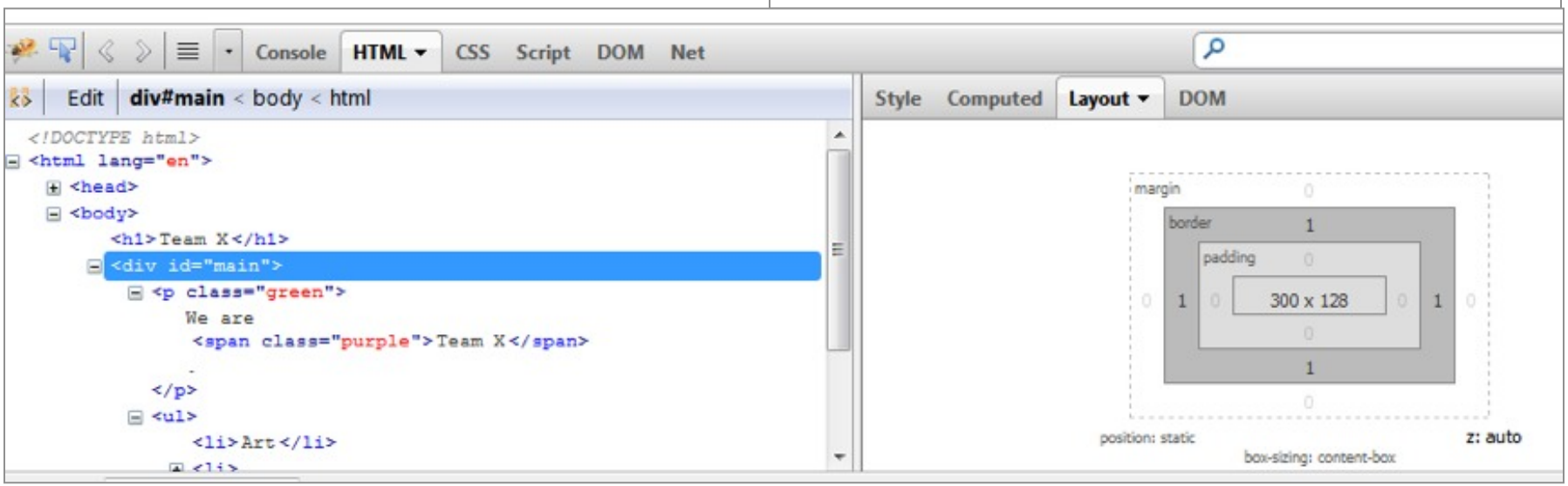
- border-top, border-right, border-bottom, border-left
  - *length* | *percentage*
- Other properties:
  - border-left-color, border-right-color, border-top-color, border-bottom-colour, border-color, border-top-style, border-bottom-style, border-style, border-left-width, border-right-width, border-top-width, border-bottom-width, border-width



6.2.4 Firebug

- Firefox add-on:
- addons.mozilla.org/en-US/firefox/addon/firebug/
- getfirebug.com





6.3 CSS Selectors: class and id

- For a class, e.g. `<p class="italic">`, use . operator:

```
p.italic { font-style: italic; }
*.italic { font-style: italic; }
.italic { font-style: italic; }
```

Matches paragraphs
Matches all elements
Matches all elements

- For a unique id, used only once in the HTML document, e.g. `<div id="main">`, use # operator:

```
#main {
  width: 300px;
  border: solid 1px blue;
}
#main, h1, h3 {
  font-style: italic;
}
```

Matches an element with id "main"

Can add extra declarations within css file

6.4 More CSS selectors

- Contextual selector: E1 E2
  - E2 is a descendant of E1
- Contextual selector: E1>E2
  - E2 is a child of E1
- Contextual selector: E1+E2
  - E2 is the immediate sibling of E1

```
ul li { color: blue; }
/* any li nested to any level in a ul */
```

```
ul>li { color: blue; }
```

```
h1+p { color: blue; }
```

http://www.w3.org/TR/selectors/

6.5 Pre-defined pseudo-classes

- Classes that depend on properties of the document rather than on the presence of a name in the class attribute
- The pseudo-class `:first-child` matches the first child of an element

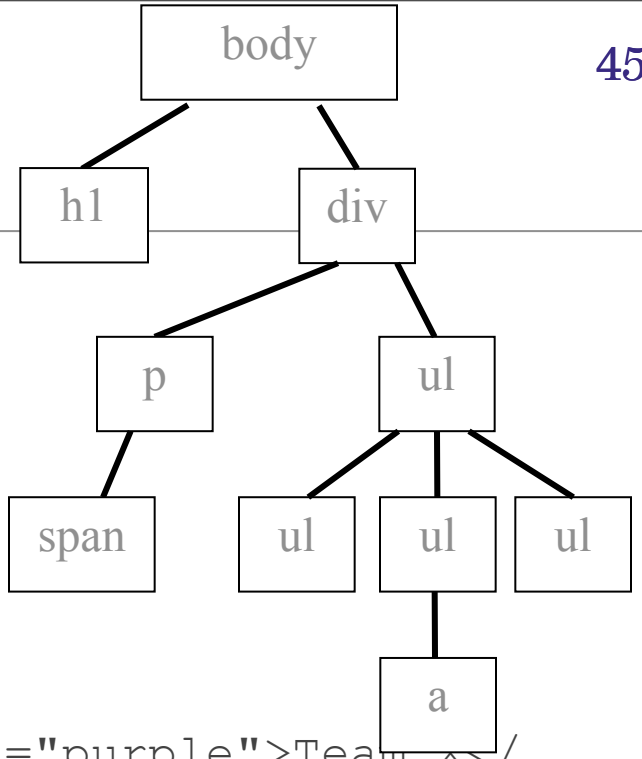
```
h1:first-child { color: red; }
```

- From the Team X example:

```
a:link { color: red; }      /* unvisited link */
a:visited { color: gray; }  /* visited link */
a:hover { color: lime; }    /* mouse over link */
a:active { color: lime; }   /* selected link */
```

http://www.w3.org/TR/selectors/

6.6 Inheritance




```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="utf-8" />
  <title>Team X</title>
  <link rel="stylesheet" href="teamx2.css" />
</head>

<body>
  <h1>Team X</h1>
  <div id="main">
    <p class="green">We are <span class="purple">Team X</
span>.</p>
    <ul>
      <li>Art</li>
      <li><a href="http://www.thesimpsons.com/">Bart</a></li>
      <li class="italic green">Cara</li>
    </ul>
  </div> <!-- main -->
</body>
</html>
```

6.6 Inheritance

- "Some values are inherited by the children of an element in the document tree" (http://www.w3.org/TR/CSS2/cascade.html)
- Unless a rule causes a different value to be explicitly assigned

```
.red { color: red; }
.purple { color: purple; }
```

```
<p class="red">Hello <em>World</em></p>
```

Hello World

em element inherits red colour

```
<p class="red">
Hello <em class="purple">World</em>
</p>
```

Hello World

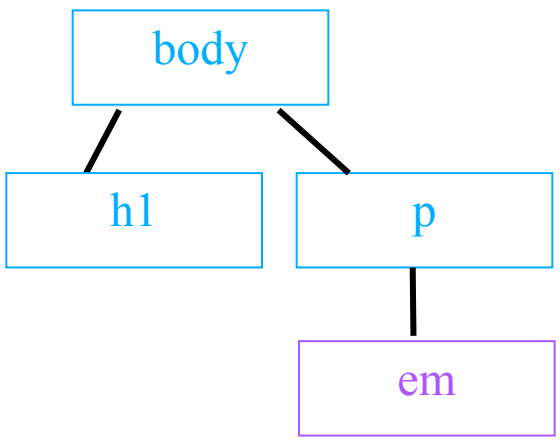
em is now of class purple which overrides the red colour

6.6 Inheritance

- Can be used to create efficient code
  - e.g. set text properties in body element, then override if necessary

```
<body>
  <h1>text</h1>
  <p>text <em>text</em> text</p>
</body>
```

```
body { color: blue; }
em { color: purple; }
```



Templates

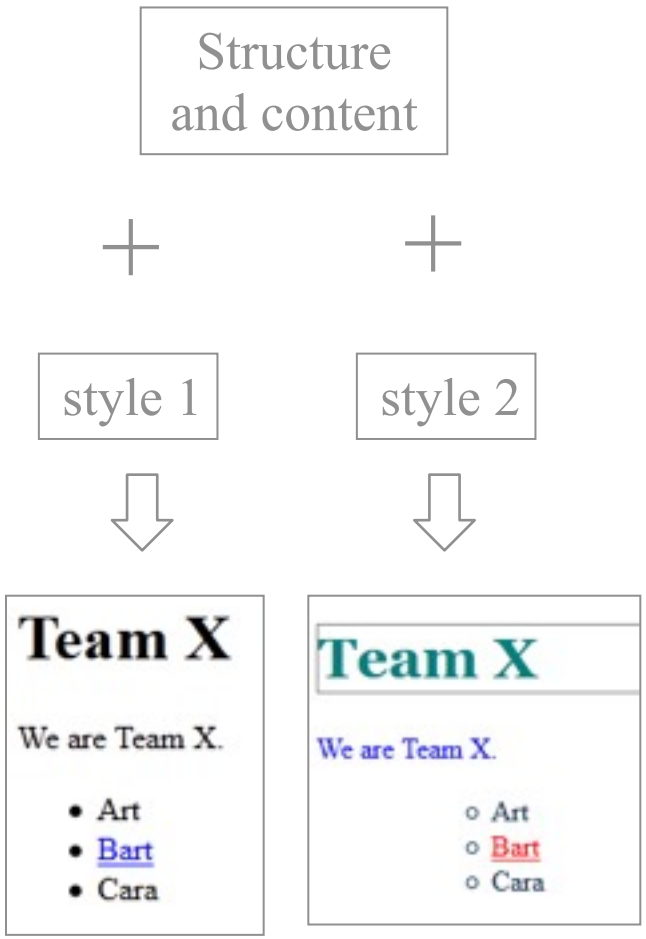
1. Introduction

- When creating a Web page, separate the structure and the appearance
- Structure is indicated using HTML
- HyperText Markup Language

```
<h1>Team X</h1>
```

- Appearance is controlled using CSS
- Cascading Style Sheet

```
h1 {
  color: teal;
}
```



2. Suppose you must create a web site like

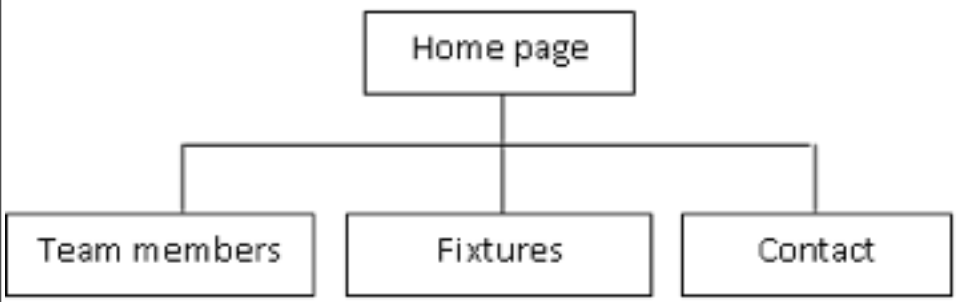


Figure 3.1: The site map for Team X

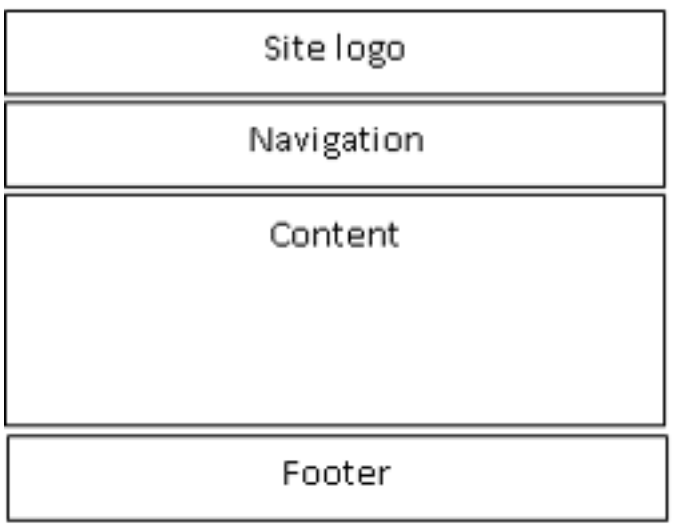


Figure 3.2: Page layout for Team X

Figure 3.3: Screen shots for Team X web site

2.1 Page layout

- Use of div elements to partition areas of the page

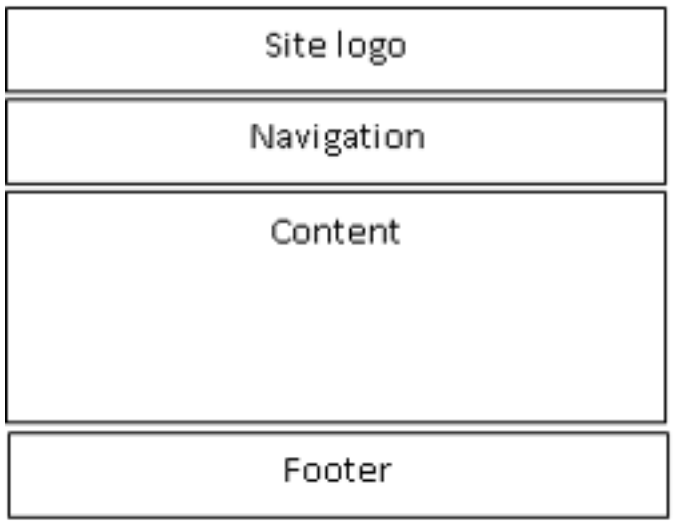


Figure 3.2: Page layout for Team X

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="utf-8" />
  <title>Team X</title>
  <link rel="stylesheet" href="teamx.css" />
</head>

<body>
  <div id="banner">
  </div> <!-- banner -->
  <div id="navigation">
  </div> <!-- navigation -->
  <div id="main">
  </div> <!-- main -->
  <div id="footer">
  </div> <!-- footer -->
</body>

</html>
```

2.2 Template

- Same starting point for each page
- `teamx.html`
- `members.html`
- `fixtures.html`
- `contact.html`

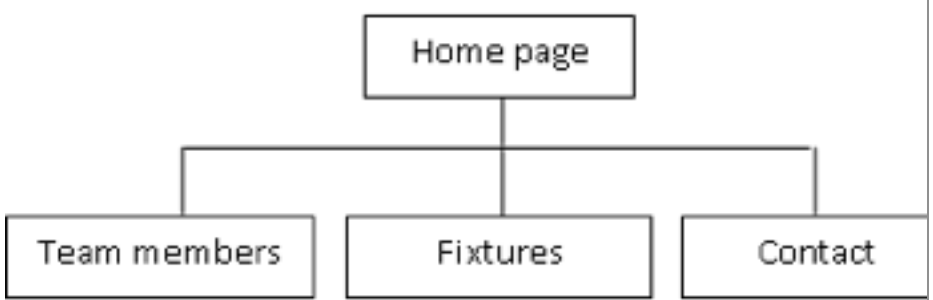


Figure 3.1: The site map for Team X

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="utf-8" />
  <title>Team X</title>
  <link rel="stylesheet" href="teamx.css" />
</head>

<body>
  <div id="banner">
  </div> <!-- banner -->
  <div id="navigation">
  </div> <!-- navigation -->
  <div id="main">
  </div> <!-- main -->
  <div id="footer">
  </div> <!-- footer -->
</body>

</html>
```

2.2 Template

- Fill in common details for each page

Footer:

- ©
- Character entity reference
- http://www.w3.org/TR/html4/sgml/entities.html

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="utf-8" />
  <title>Team X</title>
  <link rel="stylesheet" href="teamx.css" />
</head>
<body>
<div id="banner">
  <h1>Team X</h1>
</div> <!-- banner -->

<div id="navigation">
 <ul>
   <li><a href="teamx.html">Home</a></li>
   <li><a href="members.html">Members</a></li>
   <li><a href="fixtures.html">Fixtures</a></li
   <li><a href="contact.html">Contact</a></li>
  </ul>
</div> <!-- navigation -->

<div id="main">
</div> <!-- main -->

<div id="footer">
  <p>&copy; a web designer, 2011</p>
</div> <!-- footer -->
</body>

</html>
```

2.3 main area for each page

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="utf-8" />
  <title>Team X</title>
  <link rel="stylesheet"
href="teamx.css" />
</head>

<body>
  <div id="banner">
  </div> <!-- banner -->
  <div id="navigation">
  </div> <!-- navigation
-->
  <div id="main">
  </div> <!-- main -->
  <div id="footer">
  </div> <!-- footer -->
</body>

</html>
```

teamx.html

```
<div id="main">
 <h1>Welcome</h1>
 <p>We are <span class="purple">Team X</
span>.</p>
</div> <!-- main -->
```

members.html

```
<div id="main">
  <h1>Members</h1>
  <ul>
    <li>Art</li>
    <li><a href="http://www.thesimpsons.com/">Bar
    <li>Cara</li>
  </ul>
</div> <!-- main -->
```

fixtures.html

```
<div id="main">
  <h1>Fixtures</h1>
  <p>None yet.</p>
</div> <!-- main -->
```

contact.html

```
<div id="main">
  <h1>Contact</h1>
  <p>We can be contacted via <a
href="name@address.com">e-mail</a></p>
</div> <!-- main -->
```

11. Tables

- How to create the following table?

	Estimate	Measured	Error
Height (cm)	40	43	+3
Width (cm)	26	25	-1

Table 1. Widget production error

- Steps:
  - The whole table
  - A row
  - A cell
  - A heading cell
  - A caption

	Estimate	Measured	Error
Height (cm)	40	43	+3
Width (cm)	26	25	-1

Table 1. Widget production error

11. Tables

- The whole table
- A row
- A cell
- A heading cell
- A caption

table1.css

```
caption {
  caption-side: bottom;
}
```

- Further styling: http://www.w3.org/TR/CSS2/tables.html

```
<table>
  <caption>Table 1. Widget production data</caption>
  <tr>
    <td></td>
    <th>Estimate</th>
    <th>Measured</th>
    <th>Error</th>
  </tr>
  <tr>
    <th>Height (cm)</th>
    <td>40</td>
    <td>43</td>
    <td>+3</td>
  </tr>
  <tr>
    <th>Width (cm)</th>
    <td>26</td>
    <td>25</td>
    <td>-1</td>
  </tr>
</table>
```

13. Debugging

- Test after each alteration
  - Don't make lots of changes before testing
- Make use of the W3C tools
  - Markup validation: http://validator.w3.org/
  - CSS validation: http://jigsaw.w3.org/css-validator/
- Useful plug-ins for Firefox:
  - Firebug: http://getfirebug.com/
- Learn from experience
  - CSS tips & tricks: http://www.w3.org/Style/Examples/007/

14. Summary

- When creating a Web page, separate the structure and the appearance.
  - The structure is indicated using HTML
  - The appearance is controlled using CSS
- The hierarchical structure of a document can be visualised as a set of nested boxes – the box model
-

Javascript

1. Introduction

- For a Web site:
  - The structure is indicated using HTML
  - The appearance is controlled using CSS
  - The behaviour is controlled using JavaScript
    - Although see HTML5 forms
    - Although see CSS3 features, e.g. animation
- JavaScript can be used to:
  - Interact with the user
  - Control the web browser
  - Alter the document content
  - Examples: Gmail, Twitter, Firefox

1.1 Standards

- Wikipedia:
  - "JavaScript is a prototype-based scripting language that is dynamic, weakly typed and has first-class functions…
  - As of 2011, the latest version of the language is JavaScript 1.8.5. It is a superset of ECMAScript (ECMA-262) Edition 3. "
- Parts of JavaScript are:
  - formally standardized / de facto standards / browser-specific extensions
- Solutions:
  - Detect browser and change script accordingly
  - Use a library or toolkit (e.g. jQuery) which handles browser differences
- Development
  - Degrade gracefully – Web page should remain usable
  - Progressive enhancement – start basic for all, then enhance

1.2 JavaScript is not Java

- From wikipedia:

Java	**JavaScript**
Static typing	Dynamic typing – a variable can hold an object of any type
Loaded from compiled bytecode	Loaded as human-readable text; Interpreted programming language

- However:
  - Both have a structured C-like syntax (e.g. if, while, switch)
  - Both are case-sensitive
  - JavaScript copies many names and naming conventions from Java

1.3 Lectures

- Our focus will be on client-side JavaScript
  - Scripts are run by the client computer, not the Web server
- We'll start with the basics of the JavaScript language

> "First learn stand, then learn fly."
>
> *(Karate Kid, 1984)*

http request

Client

Server

2. A first program

- Write a program to calculate the area of a room
- We need:
  - Variables
  - Input
  - Calculation
  - Output

2. A first program

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="utf-8" />
  <title>JavaScript examples</title>
</head>
<body>
<h1>Example</h1>
<script>
  var length = prompt("Rectangle length in cm?");
  var width = prompt("Rectangle width in cm?");
  document.write("Area = "+length*width);
  alert("Area = "+length*width);
</script>
</body>
</html>
```

The **script** element identifies and contains the JavaScript

2. A first program

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="utf-8" />
  <title>JavaScript examples</title>
</head>
<body>
<h1>Example</h1>
<script>
  var length = prompt("Rectangle length in cm?");
  var width = prompt("Rectangle width in cm?");
  document.write("Area = "+length*width);
  alert("Area = "+length*width);
</script>
</body>
</html>
```

The variables are named `length` and `width`

prompt() produces a popup box to get input

document.write() writes to the Web page

alert() produces a popup box containing a message

2.1 Variables

- Data types
  - Numbers: 3, -3.1, 2.456
  - Strings: “hello”, “That’s all folks”, ‘world’
  - Booleans: true, false
- Variables: JavaScript

```
var score = 3;
score = 4.2;
score = 'not enough';
```

Variable is NOT associated to a type

- Variables: Java

```
int score = 3;
double scoreB = 4.2;
String message = "not enough";
```

Variable is associated to a type. This cannot be changed

Duck typing

- **A style of dynamic typing in which current properties determine valid semantics.**

- *"when I see a bird that walks like a duck and swims like a duck and quacks like a duck, I call that bird a duck.“ (attributed to James Whitcomb Riley – see http://en.wikipedia.org/wiki/Duck_typing)*

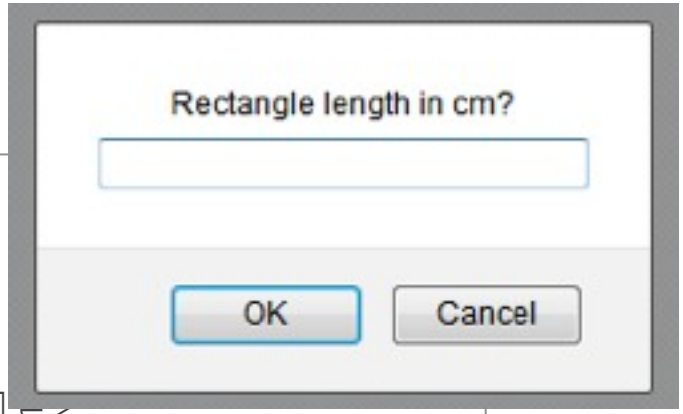
2.1 Variables

- Rules for identifiers are same as Java
  - First character must be a letter, an underscore (_) or a dollar sign ($)
  - Can't be keywords
- Further details:
  - http://en.wikipedia.org/wiki/JavaScript_syntax

JavaScript keywords	Reserved for future use	Reserved for browser
break	abstract	alert
case	boolean	blur
catch	byte	closed
continue	char	document
default	class	focus
delete	const	frames
do	debugger	history
else	double	innerHeight
finally	enum	innerWidth
for	export	length
function	extends	location
if	final	navigator
in	float	open
instanceof	goto	outerHeight
new	implements	outerWidth
return	import	parent
switch	int	screen
this	interface	screenX
throw	long	screenY
try	native	statusbar
typeof	package	window
var	private	
void	protected	
while	public	
with	short	
	static	
	super	
	synchronized	
	throws	
	transient	
	volatile	

McFarland, D.S, JavaScript: the missing manual, 2nd edition, O'Reilly, 2009

2.2 Getting input

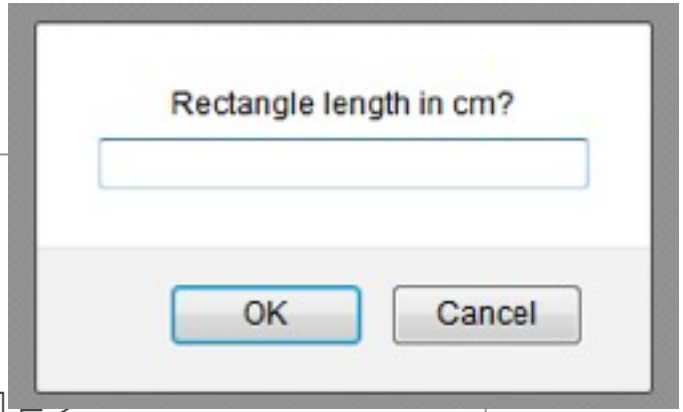



```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="utf-8" />
  <title>JavaScript examples</title>
</head>
<body>
<h1>Example</h1>
<script>
  var length = prompt("Rectangle length in cm?");
  var width = prompt("Rectangle width in cm?");
  document.write("Area = "+length*width);
  alert("Area = "+length*width);
</script>
</body>
</html>
```

prompt() produces a popup box to get input

Alternative: create a form with a series of fields – see later

2.2 Getting input




```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="utf-8" />
  <title>JavaScript examples</title>
</head>
<body>
<h1>Example</h1>
<script>
  var length = prompt("Rectangle length in cm?");
  var width = prompt("Rectangle width in cm?");
  document.write("Area = " + length*width);
  alert("Area = "+length*width);
</script>
</body>
</html>
```

prompt() produces a popup box to get input

Return type of a call to prompt is a string

String concatenation

2.4 Output

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="utf-8" />
  <title>JavaScript examples</title>
</head>
<body>
<h1>Example</h1>
<script>
  var length = prompt("Rectangle length in cm?");
  var width = prompt("Rectangle width in cm?");
  document.write("Area = "+length*width);
  alert("Area = "+length*width);
</script>
</body>
</html>
```

document.write() writes to the Web page

alert() produces a popup box containing a message

2.4 Output

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="utf-8" />
  <title>JavaScript examples</title>
</head>
<body>
<h1>Example</h1>
<script>
  var length = prompt("Rectangle length in cm?");
  var width = prompt("Rectangle width in cm?");
  document.write("<p>Area = " + length*width + "<\/p>");
  alert("Area = "+length*width);
</script>
</body>
</html>
```

document.write() writes to the Web page – should include html elements

\/p – the '\' is required by the specification and validators, but browsers will understand /p without the '\'

2.5 Comments

```
<script>
  var length = prompt("Rectangle length in cm?");
  var width = prompt("Rectangle width in cm?");
  document.write("Area = "+length*width);
  alert("Area = "+length*width);

  // single line comment
  /* multiple-line comment
     multiple-line comment */
</script>
```

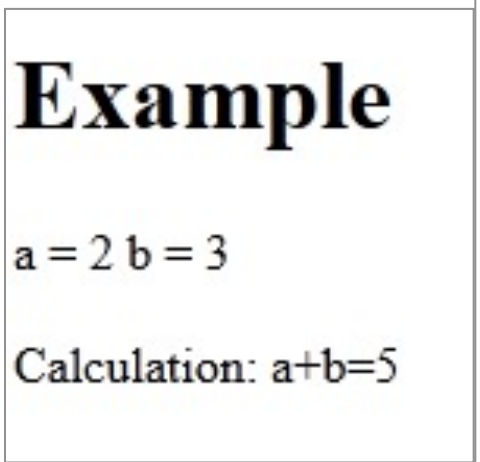
Comments similar to other programming languages

3. More on scripts

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="utf-8" />
  <title>JavaScript examples</title>
</head>
<body>
<h1>Example</h1>
<script>
  var a = 2, b = 3;
  document.write("a = " + a);
  document.write(" b = " + b);
  document.write("<br \/>");
  document.write("<p>Calculation: a+b=" + (a+b) + "<\/
p>");
</script>
</body>
</html>
```

The **script** element identifies and contains the JavaScript

String concatenation



3. More on scripts

- Script can be in a separate file
- Script can be in the head element
  - Usual approach
  - Executed before the body loads, unless controlled
- Multiple scripts can be included

```
<!DOCTYPE html>
<html lang="en">

<head>
  <meta charset="utf-8" />
  <title>JavaScript examples</title>
  <script src="write1_script.js"></script>
</head>

<body>
<h1>Example</h1>
<script src="write1_script.js"></script>
</body>

</html>
```

write1_script.js

```
var a = 2, b = 3;
document.write("a = " + a);
document.write(" b = " + b);
document.write("<br \/>");
document.write("<p>Calculation: a+b=" + (a
+b) + "<\/p>");
```

3. More on scripts

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="utf-8" />
  <title>JavaScript examples</title>
</head>
<body>
<h1>Example</h1>
<script type="text/javascript">
  var a = 2, b = 3;
  document.write("a = " + a);
  document.write(" b = " + b);
  document.write("<br \/>");
  document.write("<p>Calculation: a+b=" + (a+b) + "<\/
p>");
</script>
</body>
</html>
```

HTML 4.x requires type="text/javascript" in the script element, whereas HTML5 does not

4. Debugging

- Firefox
  - Tools, Web Developer, Error Console
  - Type JavaScript commands into the Code window and evaluate them
  - The error console will show where errors are
- Use Firebug add-on for Firefox

5. Expressions and operators: Arithmetic operators

- The most common ones...
- Given y=5, then:

Operator	Description	Example	Result
+	Addition	x=y+2	x=7
-	Subtraction	x=y-2	x=3
*	Multiplication	x=y*2	x=10
/	Division	x=y/2	x=2.5
%	Modulus (division remainder)	x=y%2	x=1

- The most common ones...
- Given x=10 and y=5, then:

Operator	Example	Same As	Result
=	x=y		x=5
+=	x+=y	x=x+y	x=15
-=	x-=y	x=x-y	x=5
*=	x*=y	x=x*y	x=50
/=	x/=y	x=x/y	x=2
%=	x%=y	x=x%y	x=0

https://developer.mozilla.org/en/JavaScript/Reference/Operators/Arithmetic_Operators

5.1 The '+' operator

- "The '+' operator is overloaded. It is used...
  - string concatenation...
  - arithmetic addition...
  - to convert numbers to strings.
  - It also has special meaning when used in a regular expression."

http://en.wikipedia.org/wiki/JavaScript_syntax

```
var vatRate = 0.15;
var costWithoutVat = 3;
var vat = costWithoutVat*vatRate;
var costWithVat = costWithoutVat + vat;
var part1 = "Hello";
var part2 = "world!!";
var message = part1 + " " + part2;
```

5.1 The '+' operator

- Automatic type conversion

```
var score = 8;
var message = "Score out of 10: "+ score;
```

- Can cause problems (https://developer.mozilla.org/en/Core_JavaScript_1.5_Guide/Core_Language_Features#Values)

```
var value1 = "37";
var value2 = 7;
var unexpectedResult = value1 + value2;  // returns
"377"
var expectedResult = value1 - value2;    // returns 30
```

- Conversion

```
var result = +value1 + value2;          // returns 44
var result = Number(value1) + value2; // returns 44
```

- If value1 is just letters rather than numbers, the result is NaN.

5.2 Precedence rules

- Same as Java
- Use brackets to clarify meaning of complicated expressions

```
a  +=   b  *  2  +  ++c  *  6
```

++ performed 1st (group 1)

* performed 2nd (group 2)

\+ performed 3rd (group 3)

+= performed last (group 4)

Group	Operators
1	++ --
2	* / %
3	+ -
4	= += -= *= /=

```
a  +=  ((b  *  2)  +  ((++c)  *  6))
```

5.3 Exercise

- Write a program that calculates the minimum number of coins required to make up a required amount of money given in pence, e.g. 457 pence is two £2 coins, no £1 coins, one 50p coin, etc.
- (Hint: 50/8=6.25; Math.floor(50/8) = 6; 50%8 = 2)
- Solution

6. Arrays

- Important differences from Java
- Creating an array

```
var days = ['Mon','Tue','Wed','Thu','Fri','Sat','Sun'];
var playList = [];
var playList2 = new Array();
var colours = new Array('red', 'green', 'blue');
```

- An array can also contain unrelated items

```
var preferences = [1, 42, 'www.dcs.shef.ac.uk', true];
```

- Arrays can be nested

```
var questions = [
  ['How many moons does Earth have?', 1],
  ['How many moons does Saturn have?', 61]
];
```

McFarland, D.S, JavaScript: the missing manual, 2nd edition, O'Reilly, 2009

6.1 Accessing items in an array

- Arrays are zero-indexed:

```
var days = ['Mon','Tue','Wed','Thu','Fri','Sat','Sun'];
alert(days[0]);  // Mon
```

- Change the value in an array:

```
days[0] = 'Monday';
```

- days.length returns the length of the array:

```
var days = ['Mon','Tue','Wed','Thu','Fri','Sat','Sun'];
var i = days.length-1;
alert(days[i]);  // Sun
```

6.2 Adding and removing items from an array

- push – add items on the end

```
var properties = ['red','14px','Arial'];
properties[properties.length] = 'bold';
                              //  red, 14px, Arial, bold
properties.push('italic', 'underlined');
                              //  red, 14px, Arial, bold, italic, underlined
```

- unshift – add items to beginning

```
var properties = ['red','14px','Arial'];
properties.unshift('bold');     // bold, red, 14px, Arial
```

- push and unshift return number of items in resulting array

```
var p = [0,1,2,3];
var numItems = p.push(4,5);     // numItems = 6
```

6.2 Adding and removing items from an array

- pop() removes the last item from the array
- shift() removes the first item from the array

```
var p = [0,1,2,3];
var removedItem = p.pop();      // removedItem = 3
                                // p = [0,1,2];
```

- splice() can be used to both add and delete anywhere in an array

```
var fruits = ["Banana", "Orange", "Apple", "Mango"];
fruits.splice(2,0,"Lemon","Kiwi");
```

Position to operate at

How many items to remove. 0 means do not remove anything

Elements to be added (optional)

7. Logic and control

- As with Java, there is:
  - if..else
  - switch
  - for
  - while
  - do..while

7.1 The if statement

```
var x = prompt('x?');
if (isNaN(x)) {
  document.write('this is not a number');
}
else if (x<0) {
  document.write('x is negative');
}
else if (x>0) {
  document.write('x is positive');
}
else {
  document.write('x is zero');
}
```

7.1 The if statement

```
var x = prompt('x?');
if (isNaN(x))
  document.write('this is not a number');
else if (x<0)
  document.write('x is negative');
else if (x>0)
  document.write('x is positive');
else
  document.write('x is zero');
```

7.3 Complex conditions

- Similar to Java

```
if (a>1 && a<10) {
  // a is between 1 and 10
  if (a>5) {
    // a is between 5 and 10
  }
  // and so on...
}

if (key=='a' || key=='b') {
  // do something
}

if (!valid) {
  // display errors
}
```

7.5 for loop

- Example: Write 6 heading styles

```
for (i = 1; i <= 6; i++) {
  document.write("<h" + i + ">This is heading " + i);
  document.write("</h" + i + ">");
}
```

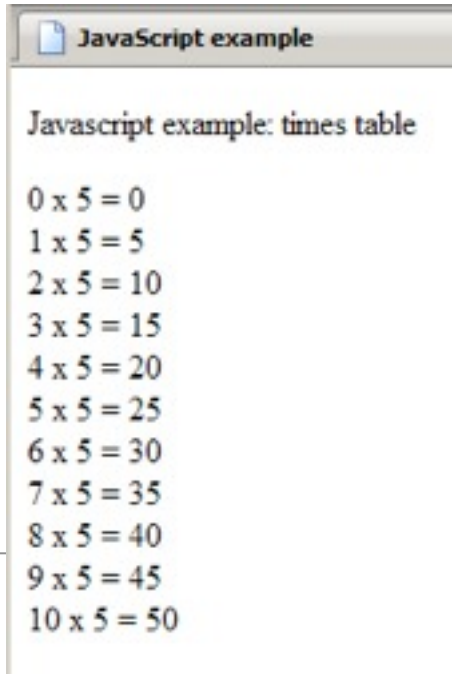

7.6 while loop

- While loop: repeat get a number until !isNaN

```
var x = prompt('x?');
while (isNaN(x)) {
  document.write('<p>not a number, try again</p>');
  x = prompt('x?');
}
document.write('<p>Number is '+x+'</p>');
```

7.7 Exercise

- Write a program to display a times table. The particular times table to display is given by the user.




```
<body>
<p>Javascript example: times table</p>
<script type="text/javascript">
  var number = prompt('Which times table (e.g. 2)? ');
  number = Number(number);
  for (var i=0; i<12; i++) {
    document.write(i + " x " + number + " = " + i*number);
    document.write('<br />');
  }
</script>
</body>
```

Here we should first check if the element is a number and if not, we should be asking again

7.8 Exercise

- Write a JavaScript program that stores 10 random floating point numbers in the range 0.0 to 10.0 in an array – see Figure 1 below – and then produces a display such as that shown in Figure 2.
-

```
4.6
7.7
5.6
1.5
2.2
2.43
3.1
5
0.2
9.0
```

Figure 1. 10 random numbers

```
4: ****
7: *******
5: *****
1: *
2: **
2: **
3: ***
5: *****
0:
9: *********
```

Figure 2. A 'graphical' representation of the data

8. Functions

- Turn useful code into reusable commands

```
<script type="text/javascript">

  function printStars() {
    document.write('<p>**********</p>');
  }

  printStars();

</script>
```

8.1 Functions and parameters

```
<script type="text/javascript">
  function print(message) {
    document.write('<p>'+message+'</p>');
  }
  print('Hello world');
</script>
```

```
<script type="text/javascript">
  function printStars(n) {
    document.write('<p>');
    for (var i=0; i<n; i++)
      document.write('*');
    document.write('</p>');
  }
  for (var i=1; i<=10; i++)
    printStars(i);
</script>
```

2.2 return value for a function

```
const VAT_RATE = 0.2;

function getCost() {
  return parseFloat(prompt("Cost before VAT in GBP? "));
}

function calculateVat(x) {
  return x*VAT_RATE;
}

var cost = getCost();
document.write("<p>Cost before VAT: "+cost+"<\/p>");
var vat = calculateVat(cost);
var costWithVat = (cost + vat).toFixed(2);
document.write("<p>Cost with VAT added:"+costWithVat+"<\/
p>");
```

`const` not in IE.
Typically, use upper-case for constants

Functions can return a value

example

2.2 return value for a function

http://getfirebug.com/logging

```
const VAT_RATE = 0.2;
console.log("vat:"+VAT_RATE);

function getCost() {
  return parseInt(prompt("Cost before VAT in GBP? "));
}

function inputCost() {
  var cost = getCost();
  document.write(cost);
  while (isNaN(cost)) {
    document.write("<p>Not a number, try again<\/p>");
    getCost();
  }
  return cost;
}

function calculateVat(x) {
  return x*VAT_RATE;
}

var cost = inputCost();
// and so on
```

example

Variable Scope

- **Variable Scope**

In JavaScript, variable scope can be global or local. Scope is determined by where a variable is declared, and to some extent whether the *var* keyword is used. Compared to programming languages like C or Java, this is a very simplistic approach.

Global

A variable that is declared outside any functions is *global*. This means it can be referenced anywhere in the current document.

- Declared outside any functions, **with or without** the **var** keyword.
- Declared inside a function **without** using the **var** keyword, but only once the function is called.

Local

A variable that is declared inside a function is *local*. This means it can only be referenced within the function it is declared.

- Declared in a function **with** the **var** keyword.

Other notes

If a variable is declared inside a conditional statement, it is still available anywhere following the declaration in the containing function (or globally if the conditional is not in a function). However, it will equal *undefined* if the condition evaluates to false, unless the variable is later assigned a value.

http://www.mredkj.com/tutorials/reference_js_intro.html#scope

2.3 Scope

```
var i = 1;                              //  global scope
j = 2;                                  //  global scope

if (i>j) {
  var k = 3;                            //  global scope
}

function a() {
  var i = 0;                            //  local scope, i.e. just this function
  document.write(i+"<br \/>");  //  display 0
  m = 4;                                //  global scope, once the function is called
  if (m > i)
    var n = 4;                          //  local scope anywhere in the function after
                                        //this line
}

document.write(i+"<br \/>");  //  display 1
a();
document.write(i+", "+m);       //  display 1, 4
```

JavaScript has function-level scoping, not Java's block-level scoping

2.3 Scope

- Be careful when attaching multiple scripts to the same web page.
- Possibility of a scope conflict
- Perhaps both scripts are trying to define the same global variable
- Perhaps one script has left the var off the front of the statement to define a variable
  - Always use var to define a variable

2.4 Exercise

- Write a program that conducts a quiz.

- Program structure:
  - Ask questions
  - Let quiz-taker know if he's right or wrong
  - Print the results of the quiz

Based on an example in McFarland, D.S, JavaScript: the missing manual, O'Reilly, 2009

2.4 Exercise

- Start with the data structures
- Keep a running score

```
var score = 0;
```

- The questions

```
var questions = [
   ['How many moons does Earth have?', 1],
   ['How many moons does Saturn have?', 61],
   ['How many moons does Venus have?', 0]
];
```

Based on an example in McFarland, D.S, JavaScript: the missing manual, O'Reilly, 2009

2.4 Exercise

- Ask questions

```
for (var i=0; i<questions.length; i++) {
  score += askQuestion(questions[i]);
}
```

Based on an example in McFarland, D.S, JavaScript: the missing manual, O'Reilly, 2009

2.4 Exercise

- Ask a question

```
function askQuestion(question) {
  var answer = prompt(question[0], '');
  if (answer == question[1]) {
    alert('Correct');
    return 1;
  }
  else {
    alert('Sorry, the correct answer is ' + question[1]);
    return 0;
  }
}
```

Based on an example in McFarland, D.S, JavaScript: the missing manual, O'Reilly, 2009

2.4 Exercise

- Print results

```
var message = 'You got ' + score;
message += ' out of ' + questions.length;
message += ' questions correct. ';
document.write('<p>' + message + '</p>');
```

Based on an example in McFarland, D.S, JavaScript: the missing manual, O'Reilly, 2009

3. Objects in Java

- **Java** is an object-oriented language
- Uses classes and objects
- To refer to methods of the object, use objectName.methodName
- For *public* properties of the object, use objectName.propertyName
- For *private* properties of the object,
  - Define an getter/setter method
  - use objectName.methodName

```
Meal curry = new Meal();          // curry is an instance of the class Meal
curry.setPrice(3.99);             // setPrice is a method of Meal
int cost = curry.getPrice();      // Get the price of the Meal curry and
                                  // store it in the variable subtotal


Person p = new Person();          // p is an instance of the class Person
p.setAge(18);                     // setAge is a method of Person
int age = p.getAge();             // Get the age of the Person p and store it
                                  // in the variable age
```


Private Properties

```
function example(param) {
  this.a = param;
  var b = true;

  this.getB = function() {
    return b;
  }
  this.setB = function(x) {
    b = x;
  }

}
```

Access b using example.getB().

4. Objects in JavaScript

- JavaScript is an 'object oriented programming language'
  - does not use classes, but does have objects
- dot notation is used to refer to methods *and* properties
  - object.property and object.method()
  - can also use object['property'] and object['method']()

```
                                    // Note: length is a property, toUpperCase is a method
var str="Hello World!";
document.write(str.length);                    // 12
document.write(str.toUpperCase());             // HELLO WORLD!
```


4.1 Creating objects

```
function increaseAge() {
  return this.age;
}

var personObj = new Object();
personObj.firstname = "John";          // add a property
personObj.lastname = "Smith";
personObj.age = 42;
personObj.increaseAge = increaseAge; // add a method

document.write(personObj);
document.write("<br \/>");
document.write(personObj.age);
document.write("<br \/>");
personObj.increaseAge();
document.write(personObj.age);
```

Objects can be created and methods and properties can be added

Note there are no () for increaseAge

A method is a function attached to an object

An object literal

```
personObj = {firstname:"John",lastname:"Smith",age:
42,increaseAge:increaseAge};
```


4.1 Creating objects

```
function increaseAge() {
  this.age += 1;
}

function Person(firstname, lastname, age) {
  this.firstname = firstname;
  this.lastname = lastname;
  this.age = age;
  this.increaseAge = increaseAge;
}

aPerson = new Person("John","Smith",42);

document.write(aPerson);
document.write("<br \/>");
document.write(aPerson.age);
aPerson.increaseAge();
document.write("<br \/>");
document.write(aPerson.age);
```

An object constructor

4.1 Creating objects

```
function computearea() {
  var area = this.width*this.height;
  return area;
}

function Rectangle(w, h) {
  this.width = w;
  this.height = h;
  this.area = computearea;
}

var rect1 = new Rectangle(2,3);
document.write("<p>rect1 area = " + rect1.area() + "<\/p>");
```

4.1 Creating objects

```
function computearea() {
  var area = this.width*this.height;
  return area;
}

function Rectangle(w, h) {
  this.width = w;
  this.height = h;
  this.area = computearea;
}

var rect1 = new Rectangle(2,3);
var rect2 = new Rectangle(2,4);
document.write("<p>rect1 area = " + rect1.area() + "</p>");
document.write("<p>rect2 area = " + rect2.area() + "</p>");
```

4.1 Creating objects

```
function computearea() {
  var area = this.width*this.height;  return area;
}

function perimeter() { return (this.width+this.height)*2; }

function Rectangle(w, h) {
  this.width = w;  this.height = h;  this.area = computearea;
}

var rect1 = new Rectangle(2,3);
var rect2 = new Rectangle(2,4);
document.write("<p>rect1 area = " + rect1.area() + "</p>");
document.write("<p>rect2 area = " + rect2.area() + "</p>");

rect1.perimeter = perimeter;

document.write("<p>rect1 perimeter = " + rect1.perimeter() + "</p>");
document.write("<p>rect2 perimeter = " + rect2.perimeter() + "</p>");
```

No output for last line rect2.perimeter does not exist

4.2 Creating objects – prototype

```
function computearea() {
  var area = this.width*this.height;  return area;
}

function Rectangle(w, h) {
  this.width = w;  this.height = h;  this.area = computearea;
}

function perimeter() { return (this.width+this.height)*2; }

var rect1 = new Rectangle(2,3);
var rect2 = new Rectangle(2,4);
document.write("<p>rect1 area = " + rect1.area() + "<\/p>");
document.write("<p>rect2 area = " + rect2.area() + "<\/p>");

Rectangle.prototype.perimeter = perimeter;

document.write("<p>rect1 perimeter = " + rect1.perimeter() + "<\/p>");
document.write("<p>rect2 perimeter = " + rect2.perimeter() + "<\/p>");
```

rect2.perimeter now exists because we have modified the prototype

4.3 Built-in objects in JavaScript

- Array, Boolean, Date, Error, Function, Math, Number, Object, RegExp, String
- The Date object methods
  - getDate, getDay, getHours, ..., getTime, ..., setDate, setHours, ...
- String
  - charAt, substring, toLowerCase, ...
- Array
  - join, sort, ...
- There are also objects that relate to the DOM hierarchy and to the browser – see later

4.4 Functions are objects

- Functions may be stored in variables, passed to other functions or stored in objects, where they become methods.

```
function average(a,b) {
  return (a+b)/2;
}

var f = average;
f(124,68);    // answer is 96
```

4.4 Functions are objects

- Functions may be passed to other functions
- The sort() method (of the Array object) will sort the elements alphabetically by default.

```
function cmp(x,y) {
  return x < y ? -1 : x == y ? 0 : 1;
}

function fillArray(data) {
  for (var i=0; i<10; i++) {
    number = Math.floor(Math.random()*10);
    data.push(number)
  }
}

function displayArray(data) {
  document.write(data);
  document.write("<br \/>");
}

var data = [];
fillArray(data);
document.write("unsorted: ");
displayArray(data);
data.sort(cmp);
document.write("sorted: ");
displayArray(data);
```

5. Forms

- Forms make it possible for Web sites to collect information from their visitors
  - e.g. order form on Amazon
- Pre HTML5:
  - Can use JavaScript to check and process the data entered – client side
  - Can validate data on the Web server – server side
- HTML5:
  - Validation inbuilt for common data types

Chapman and Chapman, 06

5.1 <form>...</form>

- Element: <form>...</form>

```
<form name="myName" action="howToHandleTheData"
      method="howToSendTheData">
  ...form contents...
</form>
```

- The action attribute is the URL of the method or program which will process the data
- The method attribute determines how data is sent to the server:
  - get – use when processing of data has no side-effects
  - post – used if there are side-effects, e.g. update a database

5.2 Example 1


```
<form name="myform"
      action="http://www.dcs.shef.ac.uk/cgi-bin/FormMail.pl"
      method="get">

<input type="hidden" name="recipient"
value="s.maddock@dcs.shef.ac.uk" />

<p>
<label for="from">Your name:</label>
<input type="text" name="from" id="from" value="type your
name..." maxlength="40" size="20" onclick="this.value=''" />
</p>

<p>
<input type="submit" name="submit" id="submit"
value="Submit" />
<input type="reset" />
</p>

</form>
```

Aside: Type 'hidden'

- Input elements with the type set to hidden are useful in two ways:
- Information not explicitly entered by the user can be included in the data sent to the server
  - Could be used to identify the page the user is on or the product described on that page
- Can be used by the server to save data in a page that it sends to a user, so that it can subsequently be sent back when a form is submitted
  - Keeps track of a user during a session

5.2 Example 1

- Each form input should have a label
- The for attribute of the <label> tag should be equal to the id attribute of the related element to bind them together

```
<form name="myform"
      action="http://www.dcs.shef.ac.uk/cgi-bin/FormMail.pl"
      method="
<input type="h
value="s.maddo
<p>
<label for="from">Your name:</label>
<input type="text" name="from" id="from" value="type your
name..." maxlength="40" size="20" onclick="this.value='"" />
</p>
<p>
<input type="submit" name="submit" id="submit"
value="Submit" />
<input type="reset" />
</p>
</form>
```

5.2 Example 1

- There are a number of different input elements
- Both ‘name’ and ‘id’ attributes should be supplied and set to the same value
  - name is used in submission of form
  - id can be used to find the specific filed in the form (e.g. with a #)

```
<p>
<label for="from">Your name:</label>
<input type="text" name="from" id="from" value="type your
name..." maxlength="40" size="20" onclick="this.value=''" />
</p>
```

- The ‘value’ attribute defines the default value for the input field
- There are also attributes that respond to user interaction, i.e. event handlers, e.g. onclick – we’ll revisit this later

5.2 Example 1

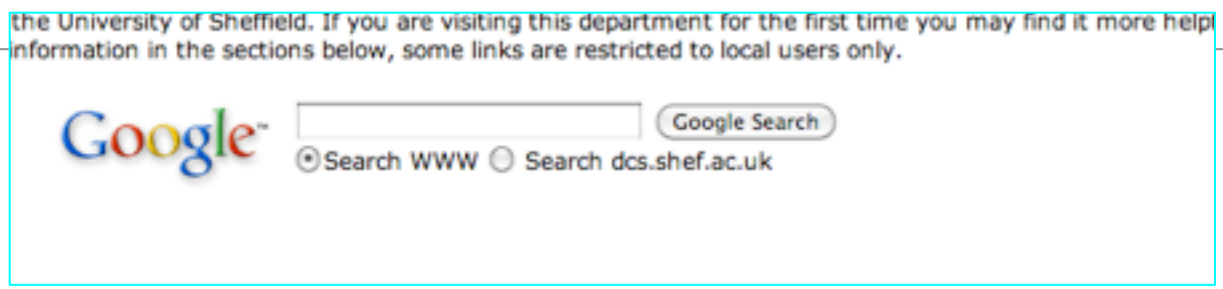

```
<form name="myform"
      action="http://www.dcs.shef.ac.uk/cgi-bin/FormMail.pl"
      method="get">

<input type="hidden" name="recipient"
value="s.maddock@dcs.shef.ac.uk" />

<p>
<label for="fi
<input type="t
name..." maxle
</p>

<p>
<input type="submit" name="submit" id="submit"
value="Submit" />
<input type="reset" />
</p>

</form>
```

- Three kinds of button can be created
  - submit button – submits the form
  - reset button – resets all controls to their initial values
  - push button – attach a client-side script to these

5.3 Example 2: A search form

- A radio button set share the same name so that only one of them can be ‘on’
- The initial one that is ‘on’ is indicated by the attribute ‘checked’
  - `checked="checked"` or just `checked`

```
<form action="http://www.google.com/search" method="get">
  <input type="text" name="q" size="31" maxlength="255"
value="" />
  <input type="submit" value="Google Search" /><br />
  <input type="radio" name="sitesearch" value="" />The Web
  <input type="radio" name="sitesearch"
         value="http://staffwww.dcs.shef.ac.uk/people/
S.Maddock"
         checked />Local search<br />
</form>
```



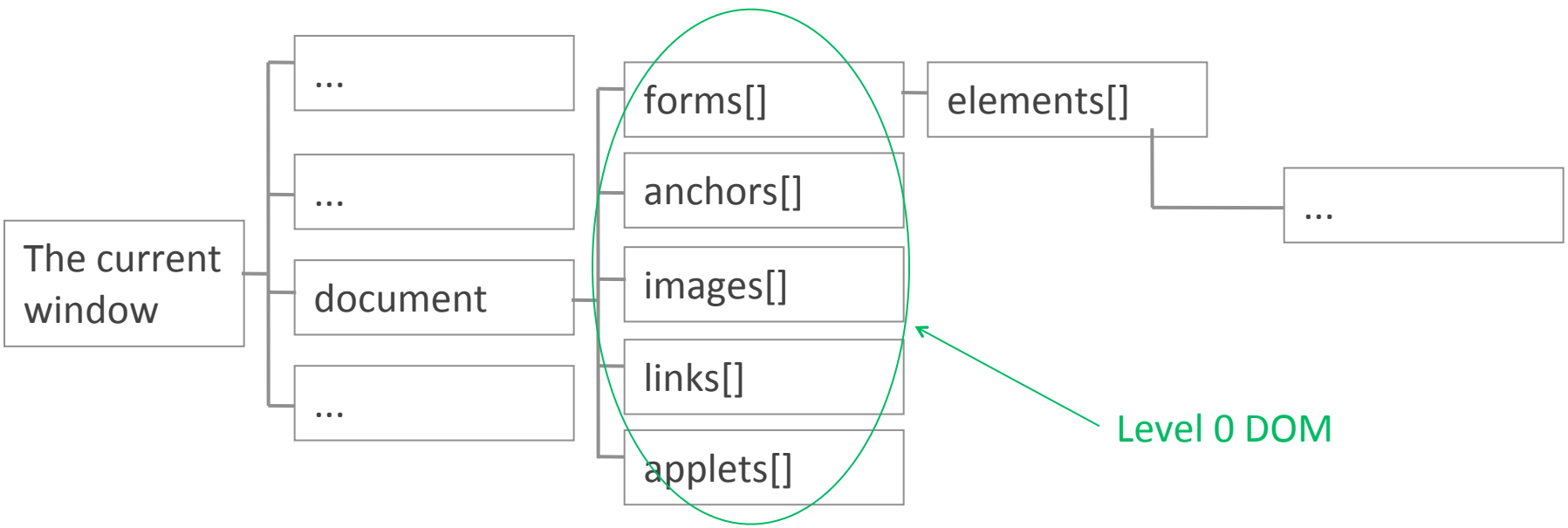
6.1 The Window object

```
var answer = 42;
window.answer = 42;
```

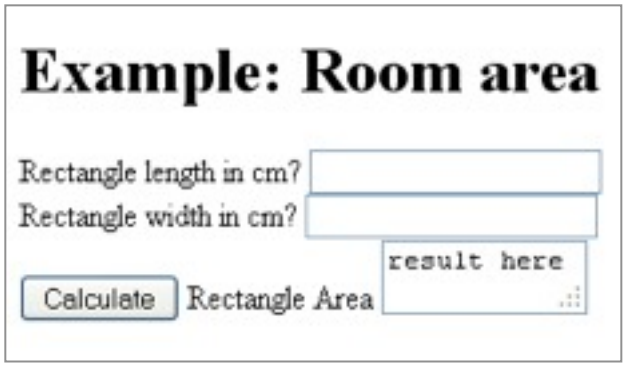
- The following are equivalent:
- As another example, the first element in the first form on a web page can be accessed using:

```
window.document.forms[0].elements[0]
```

- Can also use name attribute to access a form element – see later

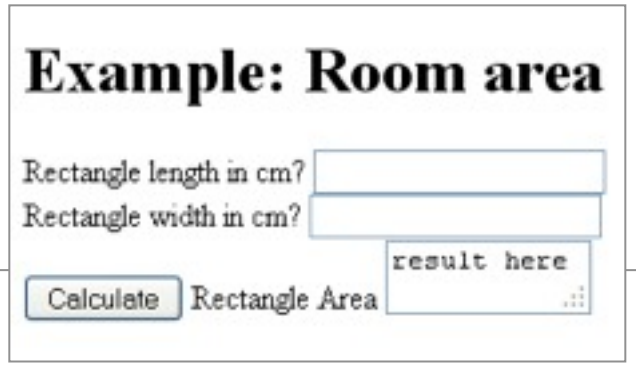


6.2 Example




```
<form id="calcarea" name="calcarea" onsubmit="area(); return false;">

<label for="mylength">Rectangle length in cm?</label>
<input type="text" id="mylength" name="mylength"
required="required"  />
<br />

<label for="mywidth">Rectangle width in cm?</label>
<input type="text" id="mywidth" name="mywidth" required="required" />
<br />

<input type="submit" value="Calculate" />

<label for="output">Rectangle Area</label>
<textarea rows="1" cols="10" id="output" name="output" >result here</
textarea>

</form>
```

6.2 Example




```
<head>...
<script>
  function area() {
    var length
      = document.forms["calcarea"].elements["mylength"].value;

    var width = document.calcarea.mywidth.value;

    document.forms["calcarea"]["output"].value=length*width;
  }
</script>
</head>
```

7. Summary

- Functions turn useful code into reusable commands
- JavaScript is an 'object oriented programming language'
  - does not use classes, but does have objects
- Forms make it possible for Web sites to collect information from their visitors
- The Web browser provides a programming environment with the following features:
  - A global Window object
  - A client-side object hierarchy – the DOM
  - An event-driven programming model
- Next lecture: Events, the DOM

Event Handling

2. Event handling

- Dealing with user interaction on a web page
- Some example events:
  - Loading a document
  - Clicking a button or other form control
  - Browser screen changing size
- A function/script is assigned to an event handler (element property)
  - E.g. *element*.onclick = **clickHandler**;
  - clickHandler can be some JavaScript code, e.g. a function call
- There are lots of event handlers for different HTML elements
  - All these properties begin with '**on**'

2. Event handling

- A function/script is assigned to an event handler, which is an element property
- The system automatically calls the relevant handler when the user interacts with the element

Event Name	Meaning	Restrictions
load	All the content of a page has been loaded.	body element only
unload	The document has been removed from the window	
click	The mouse was clicked with the cursor over the element.	
dblclick	The mouse was double-clicked with the cursor over the element.	Not a DOM event
mousedown	The mouse button was pressed with the cursor over the element.	
mouseup	The mouse button was released with the cursor over the element.	
mouseover	The cursor was moved onto the element.	
mousemove	The cursor was moved while it was over the element.	
mouseout	The cursor was moved away from the element.	
focus	The element has received the focus, i.e. it will accept input.	a, area and form control elements
blur	The element has lost the focus.	
keypress	A key was pressed and released while the cursor was over the element.	Not a DOM event
keydown	A key was pressed while the cursor was over the element.	
keyup	A key was released while the cursor was over the element.	
submit	The form was submitted.	form elements only
reset	The form was reset.	
select	The user selected some text in a text field.	input and textarea elements only
change	The element has lost the focus and its value has been modified since it received the focus.	input, select and textarea elements only

Chapman and Chapman, 06

To create an event Handler add “on” in Front of the event name click -> onclick

2.1 Example 1

```
<head>...
<script type="text/javascript">
  function printDateInfo() {
    var now = new Date();
    alert(now);
  }
</script>
</head>

<body>

<form name="myform">
<p><input type="button"
   name="dateinfo" id="dateinfo"
   value="Date info"
   onclick="printDateInfo();" /></p>
</form>

</body>
</html>
```

Note: this form has no method or action because it has no submit button. The action is calling the JS method whenpressing the button

- When button is clicked the event handler is called

2.1 Example 1


```
<head>...
<script>
  function printDateInfo() {
    var now = new Date();
    alert(now);
  }

  function init() {
    document.myform.dateinfo.onclick=printDateInfo;
  }

</script>
</head>

<body onload="init();">

<form name="myform">
   <p><input type="button" name="dateinfo" id="dateinfo"
             value="Date info" /></p>
</form>

</body>
</html>
```

- Always try to remove JavaScript clutter from button definition
- Use **onload** event to call some JavaScript that will add the event handler to the button

2.1 Example 1


```
<head>...
<script>
  function printDateInfo() {
    var now = new Date();
    alert(now);
  }
  function init() {
    document.getElementById('dateinfo').onclick=printDateInfo;
  }
  if (document.getElementById) window.onload=init;
</script>
</head>

<body>
<form name="myform">
<p><input type="button" name="dateinfo" id="dateinfo"
          value="Date info" /></p>
</form>
</body>
</html>
```

- Even better: Use onload event for window to call some JavaScript that will add the event handler to the button
- Unobtrusive JavaScript

- The if test checks that a DOM has been built for the document

3. The Document Object Model

- "An API that defines how to access the objects that compose a document"

David Flanagan, "JavaScript: The Definitive Guide", 5th edition, O'Reilly, 2006

- We've briefly looked at the "Level 0" DOM (the legacy DOM)
  - Provides a document object
  - Also includes legacy properties,
    - e.g. document.write(document.lastModified);
- Now: The W3C DOM standard
  - Defines a Document API (for any XML document)
  - Defines a specialized HTMLDocument API (includes most of the features of the Level 0 DOM)
  - Rather than use document.write(), use W3C DOM to insert content into any part of the document, even after it has been parsed

3.1 A hierarchy

- The structure of a document's element hierarchy is modelled as a tree
  - Indentation shows the levels→
- This contains **element nodes**
  - h1, p, ...
  - Shown in **blue**→
- And text nodes
  - 'A list', 'one', etc
  - Shown in orange→

```
html
  head
    sub-tree for head contents
  body
    text[\n]
    h1
      text[A list]
    text[\n]
    ul
      text[\n]
      li
        text[one]
      text[\n]
      li
        text[two]
      text[\n]
    text[\n]
```

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="utf-8" />
  <title>test</title>
</head>
<body>
<h1>A list</h1>
<ul>
<li>one</li>
<li>two</li>
</ul>
</body>
</html>
```

A list

- one
- two

3.2 The document object

- The *document object* represents the entire document. Its methods include:
- createElement
  - make new element objects
- createTextNode
  - make new text objects
- getElementById
  - access individual element objects
- getElementsByTagName
  - access all elements of a specific type.

Object	Property or Method	Meaning
document	createElement(n)	Make a new Element object of type n.
	createTextNode(t)	Make a new text node with the text t.
	getElementsByTagName(n)	Find all the Element objects of type n in the document and return them in a NodeList.
Element	getAttribute(n)	Return the string value of this element's attribute with name n.
	setAttribute(n,v)	Set the value of this element's attribute with name n to v.
	getElementsByTagName(n)	Find all the Element objects of type n in this element's children and return them in a NodeList.
	appendChild(e)	Add the Element e to this element's children after all its existing children.
	insertBefore(e,c)	Add the Element e to this element's children, before the child c.
	removeChild(c)	Remove the Element c from among this element's children.
	replaceChild(c, d)	Replace the Element's child d with c.
	firstChild	This element's first child Element.
	lastChild	This element's last child Element.
	previousSibling	The Element to the left of this element in the tree.
	nextSibling	The Element to the right of this element in the tree.
	parentNode	The Element above this element in the tree.
	childNodes	A NodeList containing all this element's children.
	nodeType	Always equal to 1.
NodeList	item(i)	Get the object at position i in the list.
	length	The number of items in the list.
Text	data	The string of text.
	length	The number of characters in the text.
	nodeType	Always equal to 3.

Principal DOM objects, from Chapman and Chapman, 06

3.3 Element objects

- *Element objects* correspond to the element nodes.
- Methods include:
  - getAttribute
  - setAttribute
  - appendChild
- Properties include:
  - childNodes, which holds a NodeList object;
  - firstChild
  - lastChild

Object	Property or Method	Meaning
Element	getAttribute(n)	Return the string value of this element's attribute with name n.
	setAttribute(n,v)	Set the value of this element's attribute with name n to v.
	getElementsByTagName(n)	Find all the Element objects of type n in this element's children and return them in a NodeList.
	appendChild(e)	Add the Element e to this element's children after all its existing children.
	insertBefore(e,c)	Add the Element e to this element's children, before the child c.
	removeChild(c)	Remove the Element c from among this element's children.
	replaceChild(c, d)	Replace the Element's child d with c.
	firstChild	This element's first child Element.
	lastChild	This element's last child Element.
	previousSibling	The Element to the left of this element in the tree.
	nextSibling	The Element to the right of this element in the tree.
	parentNode	The Element above this element in the tree.
	childNodes	A NodeList containing all this element's children.
	nodeType	Always equal to 1.

Principal DOM objects, from Chapman and Chapman, 06

3.4 Functions to reach an element in the page

Let's look at the following two methods for the document object

- getElementById("elementID")
  - returns the element with the id elementID as an object
- getElementsByTagName("tag")
  - returns all elements with the name tag as an array
- Example:

```
<body>
<ul id="mylist">
<li>one</li>
<li>two</li>
<li>three</li>
</ul>
</body>
```

3.4.1 getElementById('elementID')

- returns the element with the id elementID as an object
- Elements labelled with an id are easy to access and are very useful in conjunction with JavaScript

```
<body>
<ul id="mylist">
<li>one</li>
<li>two</li>
<li>three</li>
</ul>

<script>
  var mylist = document.getElementById("mylist");
  document.write(mylist);
</script>
</body>
```

- one
- two
- three

[object HTMLUListElement]

3.4.2 getElementsByTagName('tag')

```
<body>
<ul id="mylist">
<li>one</li>
<li>two</li>
<li>three</li>
</ul>

<script>
  var listItems = document.getElementsByTagName('li');
  document.write(listItems);
  document.write('<br \/>');
  for (var i=0; i<listItems.length; i++) {
    document.write(listItems[i]);
  }
</script>
</body>
```

- getElementsByTagName('tag')
  - returns **all elements** with the name *tag* as an array of elements
- Here, all the li tags are returned.

- one
- two
- three

[object HTMLCollection]
[object HTMLLIElement][object HTMLLIElement][object HTMLLIElement]

3.4.3 More examples

- document.getElementById('navigation').getElementsByTagName('a')**[3]**;
- // returns the fourth link inside the element with the ID 'navigation'

- var e = document.getElementById('navigation');
- var fourthLink = e.getElementsByTagName('a')[3];
- // returns the fourth link inside the element with the ID 'navigation'

- document.getElementsByTagName('div')[2].getElementsByTagName('p')[0];
- // returns the first paragraph inside the third div in the document.

http://www.onlinetools.org/articles/unobtrusivejavascript/

3.5 an earlier example…


```
<head>...
<script>
  function printDateInfo() {
    var now = new Date();
    alert(now);
  }
  function init() {
    document.getElementById('dateinfo').onclick=printDateInfo;
  }
  if (document.getElementById) window.onload=init;
</script>
</head>

<body>

<form name="myform">
<p><input type="button" name="dateinfo" id="dateinfo"
          value="Date info" /></p>
</form>

</body>
</html>
```

- Even better: Use onload event for window to call some JavaScript that will add the event handler to the button

- The if test checks that a DOM has been built for the document

3.6 Tools to navigate from a certain element

- childNodes
  - returns an array of all the nodes inside the current one.
  - There is also firstChild and lastChild, which are shorter versions for childNodes[0] and childNodes[childNodes.length-1].
- parentNode
  - The element containing this one
- nextSibling
  - the next element on the same level in the document tree
- previousSibling
  - the previous element on the same level in the document tree

3.6.1 childNodes

- returns an array of all the nodes inside the current one
- Use array notation or method 'item' to access each child node

```
<body>
<div id="mylist">
<ul>
<li>one</li>
<li>two</li>
<li>three</li>
</ul>
</div>
<script>
  var mylist = document.getElementById("mylist");
  document.write(mylist);                  // [object HTMLDivElement]
  document.write("<br \/>");
  document.write(mylist.firstChild);       // [object Text]     \n
  document.write(mylist.childNodes[0]);    // [object Text]     \n
  var nodes = mylist.childNodes;
  document.write(nodes.item(0));           // [object Text]     \n
  document.write(nodes.item(1));           // [object HTMLUListElement]
ul
  document.write(nodes[2]);                // [object Text]     \n
</script>
</body>
```

- one
- two
- three

[object HTMLDivElement]
[object Text][object Text][object Text][object HTMLUListElement][object Text]

3.6.1 childNodes

- one
- two
- three

2
[object HTMLDivElement] 3
[object Text][object Text][object Text][object HTMLUListElement][object Text]
4 5 6 7 8

```
<body>
<div id="mylist">
<ul>
<li>one</li>
<li>two</li>
<li>three</li>
</ul>
</div>
<script>
  var mylist
    = document.getElementById("mylist");
1 document.write(mylist);
2 document.write("<br \/>");
3 document.write(mylist.firstChild);
4 document.write(mylist.childNodes[0]);
5 var nodes = mylist.childNodes;
  document.write(nodes.item(0));
6 document.write(nodes.item(1));
7 document.write(nodes[2]);
8 </script>
</body>
```

```
html
  head
    sub-tree for head contents
  body
    text[\n]
    div  (id=mylist)
1     text[\n]
4,5,6 ul
7       text[\n]
        li
          text[one]
        text[\n]
        li
          text[two]
        text[\n]
        li
          text[three]
        text[\n]
      text[\n]
8 ...
```

3.6.2 More examples

- `var other = document.getElementById('nav').childNodes[3].firstChild;`
- // returns the 4th element's first sub element inside the element with the ID nav

- `var prevlink = o.parentNode.previousSibling.firstChild.childnodes[2];`
- // returns the third node inside the previous element
- // that is on the same level as the parent element of o

http://www.onlinetools.org/articles/unobtrusivejavascript/

3.7 Attributes and functions for elements

- In the HTML DOM, every **Element** object has properties corresponding to the element's **attributes**

attributes	returns an array of all the **attributes** of this element
data	returns or sets the textual data of the node
nodeName	returns the name of the node (the HTML element name)
nodeType	returns the type of the node — 1 is an element node, 2 attribute and 3 text
nodeValue	returns or sets the value of the node. This value is the text when the node is a textnode, the attribute if it is an attribute or null if it is an element
getAttribute(attribute)	returns the value of the attribute

http://www.onlinetools.org/articles/unobtrusivejavascript/

3.7.1 Example

```
<div id="mylist">
<ul>
<li id="firstLink">one</li>
<li>two</li>
<li>three</li>
</ul>
</div>
<script>
  var listItems = document.getElementsByTagName("li");
  document.write(listItems);
  document.write("<br \/>");
  for (var i=0; i<listItems.length; i++) {
    var e = listItems[i];
    document.write(e.nodeType+" "+e.nodeName);
    document.write(" " + e.getAttribute("id"));
    document.write("<br \/>");
  }
  document.close();
</script>
```

- one
- two
- three

[object HTMLCollection]
1 LI firstLink
1 LI null
1 LI null

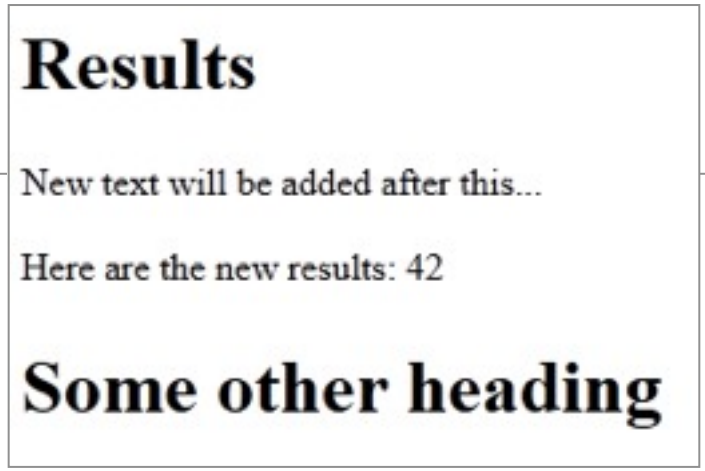
3.8 Creating new content

- createElement(element) – creates a new element
- createTextNode(string) – creates a new text node with the value string

```
<head>...
  <script>
    ...
  </script>
</head>

<body>
<h1>Results</h1>
<div id="results">
<p>New text will be added after this...</p>
</div>
<h1>Some other heading</h1>
</body>
```

Insert new text here using JavaScript

3.8 Creating new content




```
<head>...
  <script>

    var  results = null;

    function updateResults() {
      results = document.getElementById("results");
      var element = document.createElement("p");
      var str = "Here are the new results: 42";
      var textNode = document.createTextNode(str);
      element.appendChild(textNode);
      results.appendChild(element);
    }

    if (document.getElementById) window.onload=updateResults;

  </script>

</head>
```

Note! No brackets in “<p>”

3.9 Altering the existing content

setAttribute(attribute,value)	Adds a new attribute with the value to the
appendChild(child)	Adds child as a childNode to the object. child needs to be an object, you cannot use a string.
cloneNode()	Copies the whole node with all childNodes.
hasChildNodes()	Checks if an object has childNodes, and returns true if that is the case.
insertBefore(newchild,oldchild)	Adds newchild before oldchild to the document tree.
removeChild(oldchild)	Removes the childnode oldchild.
replaceChild(newchild,oldchild)	Replaces oldchild with newchild.
removeAttribute(attribute)	Removes the attribute from the object.

http://www.onlinetools.org/articles/unobtrusivejavascript/

3.10 innerHTML

- Not an official part of the DOM, but available in all modern browsers
- For an HTML element it returns or sets a string of HTML text that represents the children of the element

```
var table = document.createElement("table");
var tablecontents = "<tr><th>N</th><th>Stars</th></tr>";

for (var j=0; j<data.length; j++) {
  var n = Math.floor(data[j]);
  tablecontents += ("<tr><td>"+n+"</td><td>");
  for (var i=0; i<n; i++) {
    tablecontents += "*";
  }
  tablecontents += "</td></tr>";
}
table.innerHTML = tablecontents;
document.body.appendChild(table);
```

4. Unobtrusive JavaScript

- No JavaScript visible in the HTML file
  - Separation of form and behaviour

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="utf-8" />
  <title>JavaScript examples</title>
  <script src="nodes3b.js"></script>
</head>
<body>
  <h1>Results</h1>
  <div id="results">
  <p>New text will be added after this...</p>
  </div>
  <h1>Some other heading</h1>
</body>
</html>
```

4. Unobtrusive JavaScript

- No JavaScript visible in the HTML file

Content of the file **nodes3b.js**

```
var  results = null;

function updateResults() {
  results = document.getElementById("results");
  var element = document.createElement("p");
  var str = "Here are the new results: 42";
  var textNode = document.createTextNode(str);
  element.appendChild(textNode);
  results.appendChild(element);
}

if (document.getElementById)
window.onload=updateResults;
```

Note: the file is pure JS; it does not contain HTML tags (e.g. “<script>”)

4. Unobtrusive JavaScript

- If multiple JavaScript files included, which one gets to run after the window.onload event?
  - Perhaps have an init method that calls all the relevant initialisation code for each JavaScript include file

```
...<head>...
  <script src="script1.js"></script>
  <script src="script2.js"></script>

  <script>
     function init() { ...}
     if (document.getElementById) window.onload=init;
  </script>

</head>
<body>...
```

- This script could also be in a separate file

4. Unobtrusive JavaScript

http://www.onlinetools.org/articles/unobtrusivejavascript/chapter1.html

1. Never, under any circumstances, add JavaScript directly to the document (always use a separate file)
   - `<script src="scripts.js"></script>`
2. JavaScript is an enhancement, not a secure functionality
3. Check the availability of an object before accessing it
4. Create JavaScript, not browser specific dialects
5. Don't hijack other script's variables
6. Keep effects mouse independent

5. Summary

- The Web browser provides a programming environment with the following features:
  - A global window object
  - A client-side object hierarchy
  - An event-driven programming model
- The DOM contains many methods to manipulate the HTML document
- We are aiming for unobtrusive JavaScript
- Further reading: https://developer.mozilla.org/en/Gecko_DOM_Reference
- *Next lecture:* Graphics on the Web