HTML & CSS

This 2 day course provides an introduction to creating web pages using **HTML & CSS**.

Introduction

HyperText Markup Language (HTML) was created in 1991, by CERN physicist **Tim Berners-Lee**, and has been used over the last two decades, to create billions of pages on the web.

These notes concentrate on **HTML 4.0.1** and the associated stylesheet language. **CSS 2.1**.

Markup

HTML uses markup to add **structure** and **meaning** to the content of a web page.

Web pages are created using plain text **.HTML** documents.

HTML **elements** are added to the text, to define structure and meaning.

Elements usually consist of a pair of **opening and closing tags**.

This example wraps a heading and paragraph in opening/closing tags.

```
<h4>The history of HTML</h4>
<p>An Internet-based hypertext system</p>
```

HTML is not a **programming language**. It is not possible to define conditional logic (if .. else) using HTML.

However, the DOM can be controlled by the **Javascript** programming language, to create dynamic, interactive web pages.

Semantic content

By marking up a page,the page becomes more **semantic** or meaningful.

A marked up page

- is more searchable by engines like **Google**.
- can be read aloud by **screen readers** for blind users.
- can be **styled** using the **CSS** stylesheet language.
- can be **scripted** using the **Javascript** programming language.
- provides a **common vocabulary** for web developers to work with.

A separation of concerns

This phrase describes an approach to web page development, where **structural/semantic** decisions about content are handled in the **HTML**. **Styling/presentation** are handled in the stylesheet language **CSS**.

HTML is used to **define structure and meaning**. The developer is making journalistic decisions about content : *which heading is more important? Should this phrase be emphasized.*

Content to be **reused** in flexible ways. The same HTML content can be displayed differently, using **responsive web** design techniques, such as **media queries**, which serve up different sets of CSS rules, based on screen width.

The separation creates **less code**. The page downloads faster. The page is not bloated with table tags. The same CSS file can be reused across multiple HTML pages.

In the early days of web development, HTML tables were used for layout. This approach created bloated code, which made page maintenance difficult. *Tables still have a role in marking up tabular data*.

The separation allows developers and designers to work in **teams**, making separate structural and presentational decisions.

Hypertext

HTML is a **hypertext** markup language. It includes the ability to markup up words or phrases as **links**.

When the user clicks on a link, the browser will react in one of several ways.

- The browser will navigate to another page on the same server.
- The browser will navigate to another web page, on a remote web server.
- The browser will scroll up/down to another part of the same web page.
- the browser will open a window in the users default email client

```
<!-- Email link -->
<li><a href="mailto:info@biff.com">Email</a></li>

<!-- Anchor link within a page -->
<li><a href="#copy">Copyright</a></li>

<!-- External link -->
<li><a href="www.biff.com/history">History</a></li>

<!-- Local link -->
<li><a href="history.html">History</a></li>
```

HTML abstracts away the details of DNS servers, IP addresses, physical servers, and network connection protocols.

This code is sufficient to create a hyperlink, which navigates to the Adobe web server.

```
<!-- Create a hyperlink to a remote web server -->
<a href="http://www.adobe.com">Click here</a>
```

Writing HTML

HTML can be created with simple, free tools : a **text editor** and a **web browser**.

The basic structure of a page.

An HTML page consists of two major parts: **head** and **body** .

The body contains content visible to the end user. The head is used to define metadata, the page title, and to load stylesheets and scripts.

```
<!DOCTYPE html>
<html>
    <head>
        <title>HTML</title>
    </head>
    <body>
        <p>Content</p>
    </body>
</html>
```

This fragment uses the simplified HTML5 **Doctype** .

Headings

HTML4 provides **6 levels of heading**, that can be defined within a single page. The heading levels should reflect the relative importance of each heading.

The exact size of each heading can be set by your CSS.

```
    <h1>Books</h1>
    <h2>Fiction</h2>
    <h3>Crime</h3>
    <h4>Ian Rankin</h4>
    <h5>Inspector Rebus</h5>
    <h6>Middle-aged detective based in Edinburgh</h6>
```

Older **screen readers** may search for the first *h1* element on a page, to indicate where to start reading content from.

HTML5 introduces the ability to define multiple heading hierarchies, (from H1 to H6) within a single page.

DIV elements

In HTML4, the **div** element is used, with **class and id attributes**, to define separate parts of the page.

```
<div id="header"></div>
```

```
<div id="content"></div>
<div id="sider"></div>
<div id="footer"></div>
```

Nested divs

Divs can be **nested** to reflect the logical structure of the page.

```
<div class="outer">
    <div class="inner">
    </div>
</div>
```

Wrapper divs

A **wrapper div** is a common pattern, where the entire page is contained within a div, which is styled to centre the page.

Divs and HTML5

There is no **common vocabulary** for naming divs. Different people will use different ids and classes to describe the same type of div.

```
<div class="header">
<div class="heading">
<div class="hdr">
<div class="head">
```

HTML5 attempts to address this, and introduces a number of new **semantic tags** which more clearly describe the meaning of the content they contain.

```
section, article, header, footer, nav, aside
```

Block and inline elements

HTML provides two broad types of elements for structuring a page, block and inline elements.

Block elements

```
paragraphs, headings, lists, tables, divs
```

By default, without any further styling, browsers will display each of these elements on a **new line**, and each will occupy the **full width** of the browser window.

This means that if you construct a page, without any CSS styling, it will display as a **stacked vertical column of block level elements**.

Additional CSS will be required to change this to a multiple column layout.

The browser will ignore any additional **blank lines** that you add to the HTML source code.

The browser will ignore any additional **space characters** that you add to the HTML source code.

Inline elements

```
span, em, strong, b, i
```

Inline elements are typically nested within block elements. They allow you to structure a document at a more detailed level.

For example, **span** tags can be wrapped around a word or phrase, to give it different meaning, without a sentence or paragraph.

```
<p>Phillip moved to <span>Madrid</span> in 1472</p>
```

An individual span can be styled or scripted with CSS and Javascript.

em, strong, b, i

strong is used to mark up content with strong importance. Visually, this commonly appears **bold**, but em does not mean bold. *It is a semantic tag, not a presentational one. When read out by a screen reader, this tag may require a change of tone/voice.*

em means an emphasized piece of content, and is commonly visually

displayed in **italics**.

Older versions of HTML used the bold and italic **b** and **i** tags.

Attributes

HTML elements can be further qualified using **attributes**. Attributes are typically name/value pairs, which provide extra information about an element.

In a link, the **href** defines the URL for the link

```
<a href="www.bbc.co.uk/news">BBC news</a>
```

Some elements have attributes, but no content, such as images

```
<img src="media/flower.jpg" />
```

The **class** and **id** attributes are referred to by CSS style rules, and by Javascript program commands.

```
<div class="special">Hello</div>
<div id="special">Hello</div>
```

Occasionally, attributes do not have any value, e.g. the HTML5 video controls:

```
<video controls>
```

HTML5 introduces the ability to create custom **data attributes**. These are prefixed with "data-", and allow you to create custom markup.

```
data-full="Rembrandt Harmenszoon van Rijn"
data-born="15 July 1606"
data-file="details/rembrandt.txt"
data-wiki="http://en.wikipedia.org/wiki/Rembrandt"
```

Images

HTML pages can contain **static** images in a number of formats.

```
jpeg, png, gif, bmp
```

HTML5 also introduces the ability to generate images **dynamically** on the page, at run time, using a combination of the new **canvas** tag and related **Javascript**.

IMG element

The image tag typically does not contain any content, just attributes that point to an external image file, and other related attributes.

```
<img href="flower.jpg" />
```

It is good practice to use **relative pathnames** for image files. This gives you the flexibility to relocate files/folders to a different location on your web server.

This pathname includes a folder relative to the location of the HTML file that contains this image tag.

```
<img href="media/special/flower.jpg" />
```

Images can be styled further with CSS.

Modern browsers support **transparency**. *IE6 does not support transparent PNGs.*

By adding an **alt attribute** providing a description of the image, the image can be described using a **screen reader**.

Lists

HTML lists are useful to semantically group together **related sets** of items.

This **unordered list (ul)** groups together related fruit list **items (li)**.

```
<ul>
    <li>Apples</li>
    <li>Pears</li>
</ul>
```

Without any further styling, this list will display as a **vertical bulleted** list.

Navigational lists

A common HTML pattern uses an unordered list, to group **navigational links** on a web page (Products, Support, About Us, Contact Us.)

This pattern typically uses **CSS floats** to display the list **horizontally on a single row**.

Here, each list item *li* contains a nested link *a*

```
    <ul>
        <li><a href="">Cheese</a></li>
        <li><a href="">Beer</a></li>
        <li><a href="">Cakes</a></li>
        <li><a href="">Pudding</a></li>
        <li><a href="">Pastrami</a></li>
    </ul>
```

Form lists

Unordered lists provide a convenient way to group a set of related **form controls** as block level elements, which appear vertically aligned, and separated onto new lines.

Tables

HTML tables can be used to display **tabular data**.

Each **row** is defined with nested table cells.

```
<tr><td>Easy Readers</td><td>Jason Grigsby</td></tr>
```

Multiple rows can be grouped with a table element.

```
    <table>
        <tr>
            <td>Aaron Gustafson</td>
            <td>Adaptive Web Design</td>
```

```
        </tr>
        <tr>
            <td>Mobile First</td>
            <td>&pound;24.50</td>
        </tr>
    </table>
```

Note **& pound;** is a character entity reference for the pound symbol. Use of this entity ensures this symbol displays correctly on different platforms.

Forms

An HTML **form** groups together a related set of **controls**. Forms can be used to gather content from the user, *without the need to necessarily code any Javascript*.

Controls can include text fields, radio buttons, combo drop-down lists, and submit buttons.

```
    <form action="server.com/script.php" method="post">

        <input type="text" name="first" >

        <select name="cake">
            <option value="sponge">Sponge</option>
            <option value="carrot">Carrot</option>
            <option value="almond">Almond</option>
            <option value="muffin">Muffin</option>
        </select>

        <input type="submit" value="Send" />
    </form>
```

The user fills in the form, and then clicks **submit** to send information to a server.

Data is sent to the server as **name/value pairs**.

If the user enters *Fred Bloggs* and selects *Muffin* in the form, this object is

sent:

```
{ name : "Fred Bloggs" , cake : "Muffin" }
```

CSS

The **cascading stylesheet language CSS** allows HTML content to be styled, using a ruled-based language.

Loading stylesheets

Stylesheets are defined in separate **.CSS** text files, and loaded, using **link** elements in the head of the .HTML file.

```
    <link rel="stylesheet" href="reset.css">
    <link rel="stylesheet" href="tower.css">
```

The **order** that these stylesheets are loaded, matters. Later stylesheets may override rules defined in earlier stylesheets.

Reset stylesheets

Browser added their own **default styling** to web pages. And this default styling may vary between browsers.

To start a project from a level playing field, it is good practice to remove these default styles with a **reset stylesheet**.

The example code above uses **reset.css** to do this.

A rule based language

CSS is a **rule-based** language.

CSS rules have this general form:

```
selector {
    property : value ;
```

```
    property : value ;
}
```

A simple rule to italicise and colour paragraphs looks like this:

```
p {
    color : red;
    font-style: italic;
}
```

Selectors and the DOM

The **Document Object Model DOM** is a runtime hierarchical representation of the structure of the web page, constructed by the browser when it reads your .HTML file.

The DOM may **differ** from the original .HTML file, because the DOM can be modified at runtime, with the addition/deletion of HTML nodes, using Javascript.

CSS uses a **selector syntax** to choose elements in the DOM based on

- type of **element**, e.g. paragraph, heading
- **class** attributes
- **id** attributes
- **relative** position of elements in the DOM hierachy
- the contents of **attributes**
- pseudo properties, such as rollover/hover states.

Element selectors

Rules can use types of HTML element as their selector.

```
h4 {
    color : #00FF00;
    font-weight: bold;
}
```

Multiple types of element can be selected by using a comma-delimited list.

```
body, h1, div, p {
    color : #00FF00;
    font-weight: bold;
}
```

Class selectors

Class selectors are useful in cases, where the selection may be a a mix of different elements.

The **class attribute** is added to any block or inline element

```
<h1 class="special">The history of steam</h1>
<p>He walked into <span class="special">town</span></p>
```

The rule then styles all instances that have this class attribute.

```
.special {
    border: 1px solid blue;
    background-color: yellow;
}
```

ID selectors

An **ID attribute** applies to 1 **unique element** on each page.

```
<h4 id="info">Special information</h4>

#info {
    border: 1px solid blue;
    background-color: yellow;
}
```

Contextual selectors

More specific selections within the hierachical DOM can be made using **contextual selectors**, which use the **relative position** of 1 element to another to make a selection.

This rule only selects paragraphs **nested** within any element with a class of "panel"

```
.panel p {
    line-height: 1.4em;
}
```

This rule will work, even in cases where other elements are nested between the "panel" element and the paragraph:

```
<div class="panel">
    <div>
        <p>Info</p>
    </div>
</div>
```

We can use a **descender selector** to limit this rule, only to immediate descendants:

```
.panel > p {
    line-height: 1.4em;
}
```

This rule will work with this HTML fragment:

```
<div class="panel">
    <p>Info</p>
</div>
```

Pseudo selectors

Pseudo-selectors allow you to define rules based on the state of interactive link elements

```
a:link
/* Style links that have NOT been visited */

a:visited
/* have been clicked on */
```

```
a:hover
/* when roll over with mouse */

a:active
/* when link is being clicked */

a:focus
/* style the current in-focus element */
```

Inheritance and the cascade

Stylesheets are constructed with inheritance in mind.

Rules at the start of a stylesheet may define font and colour decisions, which are applied to the body element, and are inherited by all elements, nested within it.

```
body {
    font-family : Georgia, serif;
    background-color : #DD0044;
}
```

This approach allows stylesheets to be **concise** and avoid **repitition**.

Later rules can choose to **override** specific inherited properties.

Some rules, such as fonts and colour, are inherited. Others, such as margin and border, are not.

Specificity

In some cases, multiple styling rules may target the same element. CSS applies certain principles to decide which rule applies.

A rule with a **more specific DOM-selector** will override a rule with a more general selector.

```
ID selectors override
> Class selectors
> contextual selectors
```

```
> element selectors.
```

In the case, where two rules have equal specificity, the rule that is written latest in the stylesheet, wins.

The box model

The box model is a **conceptual model**, which describes a set of nested styles, that wrap around each HTML element.

Each element contains **content** wrapped in **padding**, surrounded by a **border**, with a **margin** outside, between it and the next element.

```
CONTENT : PADDING : BORDER : MARGIN :
```

Web developers need an awareness of this model, in order to create precise CSS based layouts.

A common pattern for **reset stylesheets** is to remove all margin and padding from all elements:

```
* {
    margin : 0;
    padding : 0;
}
```

This example constructs a debugging rule using margin, padding, border and a background colour, so that all elements with a class of "debug" are clearly visible, within their box models.

```
.debug {
    margin: 4px;
    padding: 4px;
    border: 1px solid black;
    background-color : #DDDDDD;
}
```

The box model and box-sizing

The **default behaviour** for the box model, is that the addition of margins and padding, will increase the overall width/height of an element.

This is significant, because changes in margins/padding, may cause a grid based layout to break, and cause elements to wrap onto the next line.

This behaviour can be overridden, so that content shrinks, when margins/padding are increased, so that overall width/height does not change, by applying this CSS rule.

```
box-sizing: border-box;
```

Using fonts in CSS

CSS provides a number of tools for defining typography.

Font-family

The font-family rule allows you to define a list of fonts, in order of preference. If the users computer does not have a font installed, the next font is the list is attempted, until a generic font-face is used as the final fallback.

```
body {
    font-family: Georgia, serif;
    line-height: 1.4em;
}
```

This pattern relies on **fonts that are installed on the users computer**.

Web fonts

A more flexible approach is to load in external font data from a file. This allows a web page to use a font **that may not be installed on the users machine**.

Google web fonts

Google has a web font server **fonts.googleapis.com** which you can access in the head of your HTML :

```
<link href='fonts.googleapis.com/css?family=Roboto' >
```

The font can then be referenced in your CSS:

```
font-family: 'Roboto Condensed', sans-serif;
```

This approach is simple, but does introduce a **dependency** : the Google web font server needs to be up.

Web fonts on your server

Alternatively, you can carry web fonts on your server, and use a **font-face** declaration to load this font.

```
@font-face {
    font-family: 'Biff';
    src: url('font/Biff.eot');
    src: url('font/Biff.eot?#iefix') format('opentype'),
         url('font/Biff.woff') format('woff'),
         url('font/Biff.ttf') format('truetype'),
         url('font/Biff.svg') format('svg');
    font-weight: normal;
    font-style: normal;
}
```

Layout

If you construct an HTML page, with no CSS, you will create a **single vertical column of block-level elements**. The elements will expand to occupy the full width of the browser window.

To construct a grid based layout, with content organised into columns, will typically require the use of **floats**.

FLOATs

Float takes elements out of the **normal flow** and places them as far left/right as possible within their containing element.

The **float** property can be used in conjunction with other properties, such as **width** to define column/grid layouts.

```
.column {
    float: left;
    width: 24%;
}
```

The floating behaviour will continue until a **clear** command is issued.

A lot of work has gone into finding a robust flexible rule that will clear floats in a reliable way. Some of these rules can look obscure :

```
.clearfix:after {
     visibility: hidden;
     display: block;
     font-size: 0;
     content: " ";
     clear: both;
     height: 0;
}
```

Floated navigation

A common pattern is to combine an unordered list **ul** with floats, to create a horizontal navigation bar.

```
    <ul>
        <li>Products</li>
        <li>Services</li>
        <li>Contact Us</li>
        <li>About Us</li>
    </ul>

    li {
        float : left;
        padding : 10px;
    }
```

Future grid developments in HTML5

HTML5 is introducing alternative grid layout systems such as **FlexBox**, which allow column layouts, without the need for floats. Browser support for these alternatives is still limited to modern browsers (IE10+)

Collapsing parent float children issue

If a div contains floated child elements, that container div may end up **collapsing**, i.e. CSS will treat it as having an height of zero. The solution is to apply a clearfix rule to the container.

Tools

Editors

HTML and CSS can be **hand-coded** with a good text editor. This approach allows you to learn and understand the code you write, rather than use tools which generate code.

HTML files are written as **Unicode UTF8** plain text. Do not use a word processor like Ms Word, which will add binary formatting characters to your file.

- Sublime Text (Mac,PC)
- TextWrangler (Mac,free)

Books

- HTML&CSS, **John Duckett** htmlandcssbook.com
- Learning Web Design, 4th Edition, Oreilly, **Jennifer Niederst Robbins**.

Notes by John Coumbe. Version 1.00