Week 2 – HTML Refresher

Introduction

HTML is a mark up language. It uses tags to *describe* or mark up elements on the page. Nearly all HTML tags have an opening and a closing tag:

`<body>` - opening tag
`</body>` - closing tag indicated with a `/`

There are a few 'self-closing' tags such as:

RIGHT	WRONG
`<img>`	`<img> </img>`
`<link>`	`<link> </link>`

EXAMPLES:

`<body> </body>` describes the body of a page. It is where all visible content is placed. All HTML needs to go inside of the body section. You should only have one set of `<body> </body>` tags per page.

`<header> </header>` describes an area of the page that contains the heading (e.g. a logo, text inside `<h1>` to `<h6>` tags, a table of contents).

```
<header>
  <h1>The most important heading on this page</h1>
  <p>With some supplementary information</p>
</header>
```

N.B. The <header> should not be confused with the <head> </head> section.

`<p> </p>` describes a paragraph.

`<a>` describes an anchor. When the `href` attribute is used an anchor creates a hyperlink to another web page or a specific place on the same page or another page.

Nesting

HTML tags can be nested inside one another.

EXAMPLES:

```
<header>
  <h1>The most important heading on this page</h1>
  <p>With some supplementary information</p>
</header>
```

Here, we have an `<h1>` and a `<p>` nested inside of a `<header>`.

Sometimes it is unnecessary to nest tags. For example:

```
<h1><p><strong>A Heading</strong></p></h1>
```

The same effect can be achieved just by using the `<h1>` tag. The `<strong>` tag is unnecessary because the default font-weight of an `<h1>` is bold. Similarly, the `<p>` is unnecessary because the `<h1>` is already a block level element.

Block vs. Inline

Speaking broadly, we can divide tags into two camps:

block level elements	inline level elements

Block level elements create a *block* on the page: They create an invisible rectangle around the content that is as wide as its parent element allows. If we add a background colour, this becomes a bit clearer:

```
<p>Hello I am a paragraph</p>
```

Hello I am a paragraph

As you can see, the <p> is as wide as it possibly can be. This means that any content coded after it gets pushed down into the next line:

```
<p>Hello I am a paragraph</p>
<ul>
  <li>List item 1</li>
  <li>List item 2</li>
</ul>
```

Hello I am a paragraph

- List item 1
- List item 2

Other examples of block level elements include:

- <h1> to <h6>
- <header>
- <section>
- <footer>
-
- <p>
-
-
- <article>
- <aside>

Inline elements by contrast sit "in line" with the text. They do not create the same block effect:

```
<p>Hello I <strong>am</strong> a paragraph</p>
```

Hello I **am** a paragraph

You can see that the tag does not affect the flow of the text. Examples of inline elements include:

- `<strong>`
- `<em>`
- `<a>`
- `<span>`

Attributes

Attributes provide additional information about HTML tags. Attributes can be added to any HTML tag but browsers only recognise certain uses of them. Attributes are *always* placed in the opening HTML tag.

EXAMPLES:

```
<a href="http://www.somewebpage.com">Click me</a>
```

`href` is an attribute. It is most commonly found in the `<a>` tag.

```
<img src="images/cat.jpg" alt="image of a cat">
```

`src` is also an attribute and most commonly found in the `<img>` tag.

`alt` is an attribute that must be included in the `<img>` tag. `alt` tag specifies alternative text if the image cannot be displayed for some reason.

Attributes we'll be using throughout this unit include:

- `class`
- `id`
- `href`
- `src`
- `alt`
- `data`
- `rel`

Week 2/3 – CSS Refresher

If HTML is how we structure and describe content, then CSS is how we present it. CSS tells the browser what HTML elements look like and there are a few ways to achieve this.

If you code the following into an HTML document you'll notice the browser demonstrates that various elements have their own distinct presentational features:

```
<h1>A Heading</h1>
<p>Lorem ipsum dolor sit amet, consectetur adipiscing elit. Ut mattis
mi a nulla eleifend vitae <a href="another-page.html">sagittis
libero</a> imperdiet. Vivamus <strong>consequat risus</strong> sit
amet <em>tortor laoreet</em> vitae mattis enim ornare.</p>
<ul>
  <li>List item 1</li>
  <li>List item 2</li>
</ul>
```

<h1>	Big and bold text. Has some margin around it to give it some space from preceding and following elements.
<p>	Font-weight is normal. Text is around 15px in height. Some spacing between lines and a margin to create some distance between preceding and following elements.
 	Indents the text. Adds a bullet point. Each is on its own line.
	Makes the text **bold**
	Makes the text *italicised*
<a>	Makes the text blue, underlined and clickable. Once the link has been visited, the text will appear purple and underlined.

Many HTML tags have default CSS properties built into them. We can override these by redefining the values of the properties in a CSS stylesheet.

CSS Rules – HTML tags

There are many CSS properties available to every HTML element, some are set by default and some are not. Using a stylesheet we can redefine the appearance of HTML tags.

For example, let's take the `<em>` tag. By default it *puts text in italics*, but using CSS we could redefine its properties to make it look completely different:

```
em {
  font-style: normal;
  color: #ff0000;
  font-weight: bold;
  border: 1px solid #000000;
}
```

This CSS rule would redefine the `<em>` tag so if we were then to code the following into an HTML page linked to the stylesheet:

```
<em>Here is some italicised text</em>
```

Instead of *Here is some italicised text* we would get **Here is some italicised text**.

Using CSS we can completely change the appearance of any HTML tag.

Another example:

```
section {
  background-color: #cccccc;
  border: 8px solid #000000;
  width: 800px;
  height: 500px;
  margin: auto;
  padding: 10px;
}

<section>
<p>Lorem ipsum dolor sit amet, consectetur adipiscing elit. Ut mattis
mi a nulla eleifend vitae sagittis liberoimperdiet. Vivamus consequat
risus</strong> sit amet <em>tortor laoreet</em> vitae mattis enim
ornare.</p>
</section>
```

This redefinition of the section tag gives us:

> Lorem ipsum dolor sit amet, consectetur adipiscing elit. Ut mattis mi a nulla eleifend vitae sagittis liberoimperdiet. Vivamus consequat risus sit amet **tortor laoreet** vitae mattis enim ornare.

Looks pretty ugly hey? But it should demonstrate the point.

Anatomy of CSS Rules

Before we move onto classes and IDs it's worth taking a moment to discuss the anatomy of a CSS rule because the terminology will be useful for when we later get onto jQuery.

A CSS rule is made up of three components:

1. Selector
2. Property
3. Value

Selector

In CSS the selector represents a pattern of elements you want to style. In the previous example `section` was the selector. Thus, all the section tags on the page would share the same CSS styling.

Property and Value

Once you've established the selector you want to style, property and value pairs allow you to set the various presentation features such as:

- width
- height
- border
- background-color
- font-size
- float
- margin
- padding
- color (N.B This controls the font colour)

Overview

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

You can, as we've seen, have a list of properties that you want to set. Some important things to remember with CSS:

- Always separate property from value with a colon
- Always follow a value with a semi-colon
- Remember to use US English spelling: colour becomes color

- Properties with more than one word use a hyphen as a separator, e.g. font-size, background-color

CSS Rules – Classes

A class is another type of selector, but it doesn't refer to anything that already exists in the HTML vocabulary. You create the name of the class yourself and then apply it to an HTML tag using the `class` attribute. Classes *always* have a full stop before their name in the stylesheet. For example:

```
.odds {
  background-color: #222222;
  color: #ffffff;
}
.evens {
  background-color: #ffffff;
  color: #222222;
}
```

Here I've created two classes. The .odds class creates white text on a dark grey background whilst .evens creates dark grey text on a white background. If we were to apply these classes to alternating paragraphs we'd end up with this:

```
<p class="odds">Lorem ipsum dolor sit amet, consectetur adipiscing
elit. Ut mattis mi a nulla eleifend vitae sagittis libero
imperdiet.</p>
<p class="evens">Vivamus consequat risus sit amet tortor laoreet
vitae mattis enim ornare. Suspendisse rhoncus sagittis urna quis
sodales.</p>
<p class="odds">Quisque pharetra arcu eu ipsum iaculis ac auctor ante
posuere. Integer urna dui, tristique eu consequat id, fermentum ac
arcu.</p>
<p class="evens">Integer cursus metus sit amet erat tempus eu
pulvinar justo congue. Sed vitae mi id purus auctor pulvinar.</p>
```

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Ut mattis mi a nulla eleifend vitae sagittis libero imperdiet.

Vivamus consequat risus sit amet tortor laoreet vitae mattis enim ornare. Suspendisse rhoncus sagittis urna quis sodales.

Quisque pharetra arcu eu ipsum iaculis ac auctor ante posuere. Integer urna dui, tristique eu consequat id, fermentum ac arcu.

Integer cursus metus sit amet erat tempus eu pulvinar justo congue. Sed vitae mi id purus auctor pulvinar.

So you can see from this that classes are declared in the same way as when redefining HTML tags, only you have to come up with a name for the selector and then apply it into the HTML tag using the `class` attribute.

Once you've created a class you can call it into *any* HTML tag you want. Classes are designed to be used multiple times and you can even use more than class on an HTML tag by separating them with a space:

```
<p class="odds anotherclass">some text</p>
```

CSS Rules – IDs

The third and final type of selector is the ID. It is very similar to a class. They are constructed in exactly the same way with one significant difference: An ID is prefixed with a hash rather than a full stop. For example:

```
#first {
   color: #ff33aa;
   font-size: 18px;
}
```

IDs are called into HTML tags in a manner similar to classes:

```
<section id="first"> … </section>
```

So what's the difference between ID and class?

Classes are used on multiple tags whereas an ID can only be used once on a page. It's a way to uniquely identify a particular element. This comes in handy when scripting using jQuery but also in CSS. For example, imagine a navigation bar with all the links to your web pages across the top. You could create an ID to indicate to the user's current page in the navigation bar. Certainly, you could do this with a class (and there's nothing wrong with doing so) but indication of the current page should be a unique identifier (because you can only view one page at a time) and therefore you should use an ID.

ID tricks

IDs can also offer a trick that classes can't. IDs have special browser functionality. If you have a URL like http://www.mysite.com/about.html#comments then the browser will go to about.html and also look for an element with the ID of comments and then scroll to that position on the page. We'll be exploiting this later in the unit along with some jQuery. This is an important reason for why IDs need to be unique.

An HTML element can have both an ID and a class:

```
<section id="first" class="comment"> … </section>
```

This is a useful way of applying CSS styles to multiple elements (using a class) whilst being able to uniquely identify each individual instance (using IDs).

When it comes to styling, there's nothing you can do with an ID that you can't do with a class, but IDs are useful because a) sometimes it's just the correct approach to take and b) being able to identify individual items is sometimes necessary with jQuery.

Nesting CSS

We've seen previously how we can nest HTML tags, but we can also nest CSS styles. This is really useful to specifically target selectors. Take this as an example:

```
<section id="first">
  <ul>
    <li>List item 1</li>
    <li>List item 2</li>
  </ul>
</section>
<section id="second">
  <ul>
    <li>List item 1</li>
    <li>List item 2</li>
  </ul>
</section>
```

Let's say we want to style the first `<ul>` but not the second. If we simply redefined the `<ul>` tag it would affect every unordered list on the page. We can affect only the first unordered list by being very specific with our selector:

```
section#first ul {
  …
}
```

This selector refers to an unordered list nested inside a section tag with the ID first. You can see how the CSS selector reflects the nesting of the HTML.

Nesting CSS is especially useful for restyling hyperlinks as you probably want the links in a navigation bar to appear different to those in the text of the page.

It's also good for ensuring that your design works whilst your HTML remains semantically correct. For example, you might have a heading in your header as well as headings in sections on your page. Reflecting the HTML nesting in your CSS selectors makes this easy. For example:

```
header h1 {
  …
}
section h1 {
  …
}
```

The h1 indicates a heading, but might serve different presentational purposes depending on its structural (semantic) placement.

Week 3 – New Positions

In MMCC2140 we showed you how to position elements using a combination of floats, margins and padding. Whilst this remains the correct way to position elements much of the time, there will be times where these techniques are unable to achieve what you want.

We're going to introduce you to a new CSS property and its values.

position

The position property offers four values:

static	This is the default setting for HTML elements. A static positioned element is always positioned according to the normal flow of the page.
fixed*	An element with a fixed position is positioned relative to the browser window. It will not move even if the page is scrolled. Fixed position elements are removed from the normal flow, their presence has no effect on other elements.
relative*	A relative position element is positioned relative to its normal position. For example, you can provide an offset to the left (this can be positive or negative).
absolute*	An absolute position element is removed from the normal flow. Its position is relative to its first parent element that has a position other than static. If no such parent element is found then the containing element is the body. Absolute position elements can be stacked on top of each other using the `z-index` property. The lower the number the further towards the back.

* These properties enable four other properties, but two are mutually exclusive:

- top and bottom (values set in pixels)
- left and right (values set in pixels)

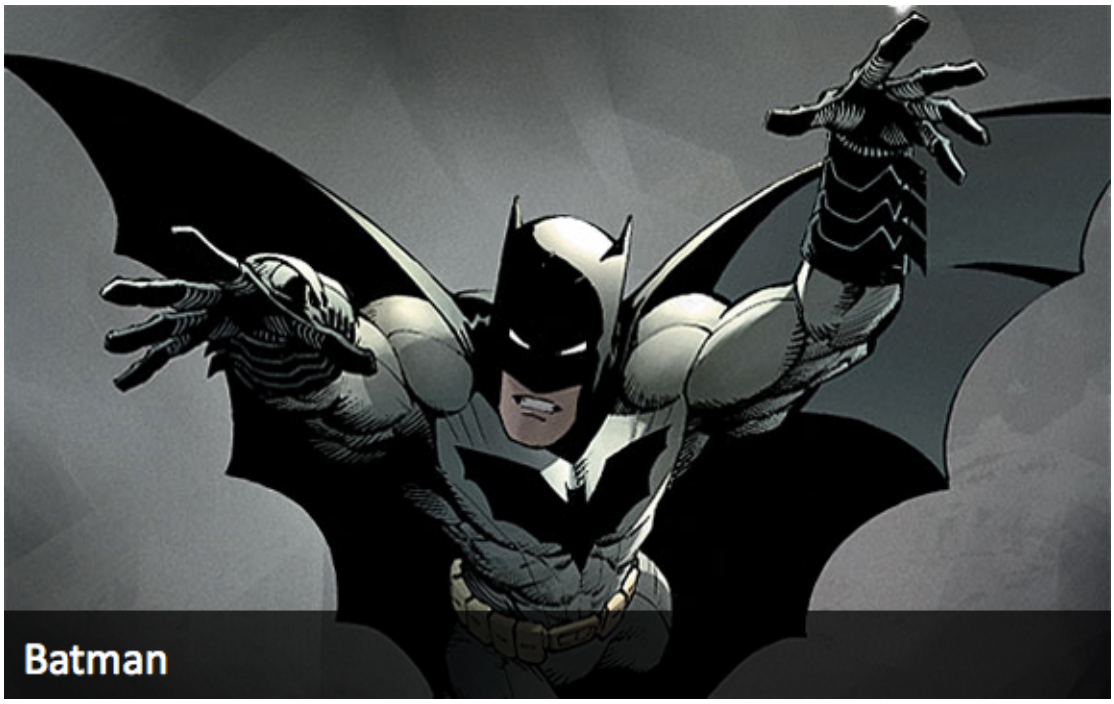
So you can't use top and bottom at the same time *or* left and right at the same time.

During the workshop we're going to show you a really useful trick using absolute positioned elements inside of a relatively positioned element.

Absolute inside Relative

Using absolute position elements nested inside relative position elements open the doors to some neat tricks.

Take this captioned image of Batman as an example:



What we have here is a `<figure>` with a class that sets the `height` and `width` to match the image (in this case 576px x 365px) and the `position` property set to `relative`.

Nested inside the `<figure>` is the `<img>` and a `<figcaption>` (figure caption) with the text "Batman" and a class that sets the appearance of the caption. Both `<figcaption>` and the `<img>` use the CSS property `position` but set its value to `absolute`.

When an absolute position element is nested inside a relative position element, the top-left corner (in CSS, top: 0px; left: 0px) become relative to the element in which it is nested.

So, in this instance, the `<img>` is set to `position: absolute;` and `top: 0px; left: 0px;` which ensures its top-left corner is snug with the top-left corner of the relative position `<figure>` in which the `<img>` is nested.

Similarly, the `<figcaption>` is set to `position: absolute;` but instead of using `top` and `left` properties it uses `bottom` and `left` (both set to 0px).

Aside from `top/bottom`, `left/right` properties, absolute position elements also unlock the `z-index` property. Absolute position elements can be positioned in front or behind one another, and the `z-index` property defines the order: The

higher the number the further toward the front it is placed. The lower the number, the further toward the back.

In this instance the `<img>` has a `z-index` of 1 and the `<figcaption>` has a `z-index` of 2 ensuring that it sits in front of the image.

You'll also note that the caption is semi-transparent. There are two ways to achieve this effect but one is superior to the other.

Colours and Codes

The easiest way to achieve the semi-transparent caption is to set the `opacity` property to the class applied to the `<figcaption>`. If we go down this route though, the opacity applies not only to the `<figcaption>` but everything nested instead it, including the white text. A better method is to make only the background colour to `<figcaption>` semi-transparent, but to achieve this you need to know a bit about how colours are generated on computers.

In MMCC2140 we defined colours using a six-digit HEX code. For example `#ff0000` is the colour code for red.

There are several colour methods available. The three most common ones are Greyscale (black and white), CMYK (cyan, magenta, yellow and black) and RGB (red, green and blue). These modes refer to the base group of colours that are mixed to create all other colours. Computers tend to use RGB. In the six-digit colour code, the first two values refer to the amount of red, the second to the amount of green and the final two to the amount of blue that are mixed to create all other colours. As the values are in hexadecimal (0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F) you can see that in the case of `#ff0000` the maximum amount of red is mixed with zero amounts of green and blue to produce red.

The HEX-code, however, is not the only method for declaring colours in CSS. There is also the RGB(r,g,b) which uses values between 0 and 255. For example: `background-color: #ff0000;` and `background-color: rgb(255,0,0);` will both set the background colour to red.

Personally, I don't use the RGB(r,g,b) method very often, but there is an extension of it – RGBA(r,g,b,a) that adds an alpha (opacity) value that ranges from 0 to 1 (so you need to use fractional values). For example:

```
background-color: rgba(255,0,0,0.6);
```

This will set a semi-transparent red background, but will not affect the opacity of anything nested inside the element.