HTML

HTML is now very well standardized, but sites are still not getting it right
**HTML**

*HTML = HyperText Markup Language* is used for the structural part of web pages.

Our strategy for backward compatibility plus future proofing is to use HTML5.

For development, *either* we will use XHTML5 and send pages with content type `application/xhtml+xml` to get instant feedback and occasionally validate, *or* we will set up a validation scheme.

We may support 🔄 [old browsers](#).
To support old browsers, if you are using XHTML5, you need to do two things:

One is to use the Polyglot HTML5 standard to write pages which work both as HTML5 and XHTML5.

The other is to do content negotiation, i.e. check if the request Accepts header contains `application/xhtml+xml` or not and deliver pages with a response Content-Type header which is either `application/xhtml+xml` or `text/html` accordingly.
Some general sources of information are:

- HTML5 standard, Polyglot HTML5
- Mozilla Developer Network
- Web Platform
- Web fundamentals, Can I use, w3schools

Random tutorial or Q&A sites generally contain obsolete, incorrect and incomplete info

This chapter shows some available tags, leaving style, scripts, images, media etc. until later
The Template

To start a new web page, copy and paste this:

```html
<!DOCTYPE html>
<html xmlns="http://www.w3.org/1999/xhtml"
     lang="en-GB" xml:lang="en-GB">
<head>
    <meta charset="UTF-8"/>
    <title>Put Title Here</title>
</head>
<body>
    <h1>Put Heading Here</h1>
</body>
</html>
```

This *must* be exact, so never type it:

- `doctype`
- `lang`
- `charset`
- `title`
- `bom`
- `HTML4`
The Doctype tag

The purpose of the DOCTYPE tag is to trigger standards mode in all browsers (not "quirks" or "limited quirks" or "almost standards" mode) and these are wrong:

```xml
<?xml version="1.0" ?>
<!DOCTYPE html>
<!-- comment -->
<!DOCTYPE html>
<!doctype html>
```

Getting this wrong is a disaster, because then even validators may fail to spot errors.
The purpose of the `lang` attributes is to tell spell checkers, translation tools, and screen-readers for the blind, what human language the page is written in, so they are required.

The language code (en-GB for UK English) is repeated twice, so both HTML and XHTML browsers understand...
A common mistake is to think that `<meta charset="UTF-8"/>` tells browsers that the page uses the UTF-8 charset. Browsers ignore it and use the response header, so the server has to be set up properly to deliver the right response header.

The meta tag is for humans and offline tools, especially validators, where there is no response header.
The title doesn't appear in the page, it appears in the browser's title bar, or tab name, the user's history, and in bookmarks.

A title should make reasonable sense in these contexts, though it is often OK to repeat the main heading, which does appear on the page.

The standard insists that a title element should be present, and that its content should not just be white space.
The Byte Order Mark is an initial character used to distinguish the order of bytes in UTF-16

While allowed for historical reasons in UTF-8 (even in Polyglot HTML5), it is usually invisible, and it often causes trouble

I strongly recommend making sure your editor doesn't use it
The HTML4 Template

The equivalent for HTML4/XHTML1.0 is

```html
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
 "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<html xmlns="http://www.w3.org/1999/xhtml"
 lang="en-GB" xml:lang="en-GB">
<head>
<title>Put Title Here</title>
</head>
<body>
<h1>Put Heading Here</h1>
</body>
</html>
```

But don't use it in this course, unless you have permission for a *very* exceptional reason
Many pages use *fixed layout* (usually fixed width) because the author wants total control. With *fluid layout*, you say *what* you want displayed, and the browser and visitor choose *how*. With *responsive layout* you use a framework so that the layout of the page adapts itself to different browsers, including mobiles.

Using a fixed layout is (usually) *wrong*; it violates the principle of assuming nothing about visitors

Fluid layout is what you get if you keep everything really simple and don't try to do anything clever
Visitor assumptions

Web authors need to avoid being arrogant; a web page is a cooperation between an author and a visitor, and a visitor's preferences matter.

You shouldn't assume anything about desktop versus tablet versus mobile, or screen size, or resolution, or aspect ratio, or preferences.

For example, even if you assume a visitor is using a modern big-screen desktop, they may use half-width browser windows because they have many windows on screen at once (especially if they are computer scientists).
Block and inline tags

Tags are divided roughly into *block* tags (vertical) and *inline* tags (horizontal)

The HTML5 document calls them (roughly) flow and phrasing tags, but the words block and inline are used in stylesheets and scripts

Some tags are now allowed in either context, and have 'transparent' content, meaning what's allowed inside is the same as what's allowed outside
Some block tags are:

- `<h1>`  `<h2>`  ...  `<h6>` headings
- `<p>` paragraphs
- `<br>` line breaks (use sparingly)
- `<ul>`  `<ol>`  `<li>`  `<dl>`  `<dt>`  `<dd>` lists
- `<fieldset>` box round form fields
- `<pre>` fixed width & format text
- `<hr>` horizontal rule (line)
The div (division) element is a block-level wrapper which has no effect, except that it provides a place to add style or scripting.

For example, a group of paragraphs can be given a different background.

We will see that there are new HTML5 tags which say why you are adding a wrapper, so you will hardly ever need div, but knowing about it helps in understanding tutorial sites.
Some inline tags are:

<em>  <strong>  <dfn>  <code>  <kbd>  <samp>
<var>  <cite>  <q>  <sub>  <sup>

Some that you should use a lot less are

<b>  <i>  <u>  <ins>  <del>  <big>  <small>  <tt>

The trouble is that they don't say what you mean, so it is generally better to use more meaningful tags, then use a stylesheet to say how you want them to look
The **span** tag

The **span** element is an inline-level wrapper which has no effect, *except* that it provides a place to add style or scripting.

It is the inline version of the **div** element.

For example, a sentence can be given a different background.

Again, there are more meaningful new tags in HTML5 which reduce the need for **span** tags.
In HTML5, there are new (div-like) sectioning tags, e.g.:

```
<header> <footer> <nav> <main>
<section> <article> <aside> <figure>
```

There are new (span-like) phrasing tags, e.g.:

```
<mark> <time> <command>
```

There are new media tags, e.g.:

```
<video> <audio> <canvas> <svg> <math>
```

See the [HTML5 differences from HTML4](#) document
Void elements

For compatibility, the polyglot standard insists that the self-closing tag syntax must be used for void (always empty) elements:

area base br img input keygen link meta param source track wbr

Explicit open and close tags must be used for all other empty elements:

```html
<img src="pic.png" />
<script src="do.js"></script>
```
One consequence of fluid design is that in your source text, white space makes no difference:

```html
<p>This is a paragraph of text written on two lines in the source</p>
```

The source text above might end up in the browser looking like the text on the right or the text below:

This is a paragraph of text written on two lines in the source
In XHTML, there should be no newline at the start of `pre` or `textarea` and for tags with `inline` text content, you should not put a space after the start tag or before the end tag, because in some browsers it can look wrong or spoil copy-and-paste:

```html
<pre>no initial newline...
<p>No space at either end</p>
<td>Same with a table cell</td>
<a href="...">Same with a link</a>
Same with <i>italics</i>
```
Tables

Simple tables look like this (see `tbody` note):

```html
<table><tbody>
  <tr>
    <td>top left</td>
    <td>top right</td>
  </tr>
  <tr>
    <td>bottom left</td>
    <td>bottom right</td>
  </tr>
</tbody></table>
```

Tables used to be used a lot for magazine-style columns, pinboards etc., but it is usually better to use stylesheets or other techniques now.
Although we can't make forms do anything yet, the tags which are relevant to forms are: `<form>  <input>  <textarea>  <button>  <select>  <option>  <fieldset>` and a simple form is:

```html
<form action="params.html" method="get">
  <input type="text" name="pet" value="cat" />
  <input type="text" name="car" value="bmw" />
  <input type="submit" value="Go" />
</form>
```

The action is a URL saying what page to visit, the method is "post" ("get" for debugging), the form content is any HTML, plus inputs of various types
For maximum clarity, indent links:

```html
<p class="indent"><a href="url">Description</a></p>
```

The url should be

- page-local (#explain) in the same page
- folder-local (x.html or dir/x.html) if close by
- site-local (/dir/x.html) if same site
- protocol-local (//www...) for http/https
- full (http://www...) otherwise

Replace & by &amp; (e.g. page?x=1&amp;y=2)
Ids and classes

Any element can be given a unique id, so it can be picked up by code in a script (or targeted by a stylesheet)

```html
<button id="save">...</button>
```

Any element can be given a class, or a space-separated list of classes, so it can be targeted by a stylesheet (or picked up by a script)

```html
<code class="java">...</code>
<code class="c">...</code>
```
Character Entities

It is OK to use UTF-8 characters directly (if you set the server up suitably) or to use numeric entities (e.g. &#960; or &amp;#x3C0;)

But in XHTML, the only accepted named entities are &lt; &gt; &amp; &apos; and &quot;

This sometimes causes problems if you use libraries which insist on using entities like &nbsp; or &copy;

There is a way of defining extra entities
Extra entities

Here's how you can define extra entities (works with XHTML delivery only)

```html
<!DOCTYPE html [
  <!ENTITY nbsp "&#160;" >
  <!ENTITY copy "&#169;" >
]>html
```

This can be used with frameworks which insert entities for you and don't let you customize
Adaptation

When you have a valid page that works in your development browser, it *should* instantly work on other new browsers.

The next step is to make it work on old browsers.

It is no longer regarded as sensible to spend huge amounts of time and effort on this.

Instead, it is better to look for polyfills and frameworks that do the work for you.
Tags as hooks

HTML tags describe the structure of a page; they are not supposed to control how a page looks or behaves.

In a way, they are all just hooks on which to hang style and scripting.

Therefore almost all the work to support old browsers is done in stylesheets and scripts, and not in the HTML.
Example: a simple polyfill

```javascript
document.createElement('section');
```

```css
section { display: block; }
```

The new `section` element works exactly the same as the old `div` element; it is just a block-level wrapper.

One line of JavaScript stops old browsers from ignoring it, and one line of CSS (or JavaScript) makes it look right.
Your pages **must** be valid, otherwise they won't look the same on all browsers, but you have three choices of standard to follow:

- HTML5
- XHTML5
- Polyglot HTML5 = HTML5 + XHTML5
For this, take out the `xmlns` attribute from the `<html>` tag, and configure the server to deliver all pages as `text/html` instead of using content negotiation.

**Advantages:** better for some frameworks, a bit less fussy

**Disadvantages:** you don't get any feedback from the browser for invalid pages, so you **must** use `vnu.jar` for validation (and you need to understand the fussy 'unfussy' rules)
The XHTML5 Choice

For this, include the `xmlns` attribute in the `<html>` tag, configure the server to deliver pages as `application/xhtml+xml`, and follow the tighter rules (e.g. no missing close tags)

**Advantages:** you get instant feedback from the browser for pages which are not well formed

**Disadvantages:** you cannot support old browsers, and your framework may not support it
For this, include the `xmlns` attribute in `<html>`, configure the server to use content negotiation to send pages as `text/html` or `application/xhtml+xml` according to the browser, and follow even tighter rules (e.g. `<img.../>` and `<script..>`) 

**Advantages:** you get instant feedback from the browser for pages which are not well formed, and you support old browsers

**Disadvantages:** the rules are a bit fussy, you may need `vnu.jar` occasionally, and your framework may not support it