Your First HTML Lab Session

***Martin Stacey***

I’ve tried to write down the things that I say to everyone in their first HTML lab sessions.

**Your Own Web Pages**

The whole point of the HTML+HCI component of this module is that you learn to produce real web pages, that are accessible to everyone all over the world, by actually doing it. We enable you to publish real web pages via a DMU web server. Files in the *public\_html* folder in your DMU network file space (your H: drive) are accessible via the web, at the URL ***https://my.tech.dmu.ac.uk/~yourPnumber/*** (Note the ~ for a user account.)

We want you to **do all your work** for the module directly **in your *public\_html* folder** in your H: drive. We recommend that you create subfolders inside it for each module for which you produce web pages. Call the one for this module *ctec1412*. Put all the files that you download or create for this module in there, unless there’s a good reason why you shouldn’t.

The *http://...* prefix for your URLs will work for computers inside the university, but to read student web pages from elsewhere, you need to use the *https://...* prefix.

**Creating Your *public\_html* Folder**

Just making a new folder in the usual way isn’t enough.

**If you don’t already have a *public\_html* folder in your H: drive**, what you should do is double-click on the icon you’ll find on your windows desktop, labelled ***‘Make my public\_html’***, and wait a couple of minutes for the script that creates a *public\_html* folder for you with the right settings to finish executing.

**Be Sensible**

We don’t want to have to police what you put on your university web pages, but remember that you can get the university into extremely serious trouble. So think twice about whether what you’re putting on your pages is appropriate and sensible, and avoid content that’s pornographic or grossly insulting or violates copyrights that people are going to care about (especially content belonging to celebrities or entertainment industry companies). It will only take one obnoxious idiot to get the vice chancellor to come and force us to have a much more rigid and restrictive policy on student websites, and put time and effort into enforcing it, or alternatively not have them at all. So don’t do anything completely stupid.

**Index files: Just don’t**

If you have a file called *index.html* in one of your folders, the web server will return the *index.html* file instead of the list of files in the folder in response to the URL of the folder. They are very useful almost all the time. But *we don’t want them* in student websites at DMU. This is because we frequently need to look at listings of files in folders to find files when fixing problems and marking assignments, and an index file may make it difficult or impossible to find your work. **Please don’t put any file called *index.html*** in any of your DMU folders.

**Writing Web Pages in XHTML Strict**

We expect you to write all your web pages in **XHTML Strict**. (XHTML is a markup language that is an implementation of HTML in XML, which is a meta-language for specifying markup languages; it requires you to do things a little differently so your files are legal XML documents.)

This means they should include the **XHTML 1.1 doctype declaration** at the beginning of the file. The (XHTML 1.0 Strict doctype declaration is obsolete, but also allowable for this course; you’ll need to include a <meta …/> tag to declare the charset.)

This is absolutely required for all work for this module. Not using XHTML 1.1 or 1.0 Strict doctype declarations for your assignment is the quick way to get a fail mark. (You may encounter XHTML 1.0 Transitional, which allows the old-fashioned HTML formatting we’re teaching you to avoid: do not use it; if you do, expect no mercy from your tutor.)

We’re insisting on this despite all the HTML5 hype because XHTML imposes much stricter standards for correct code. So learning to write correct standards-compliant XHTML Strict out of habit – to the point where you know it’s correct give or take the occasional typo – will get you to produce good disciplined maintainable code and not be sloppy. This has major advantages, not least for making sure your code works for visually handicapped users.

But we’re not discouraging you from using all the new HTML5 features, even if they’re not correct in XHTML Strict. (We know how to read validator reports: we can distinguish slips from bad code from advanced features.)

**Using the W3C Validator**

We expect you to use the W3C Validator at ***http://validator.w3.org*** to test whether your code is correct XHTML Strict. Do this early and often, until you *know* you’re writing correct code. Thinking that validating is something you do at the end is a BIG mistake. Remember that the validator is your friend: it will teach you how to write good code if you let it.

**Getting XHTML Strict Pages to Work Properly in Internet Explorer**

Unfortunately Internet Explorer doesn’t simply display all pages according to current standards. Instead it tries to figure out what version of IE you wrote your pages for, and impersonates an earlier, out-of-date version of IE, unless you use an HTML5 doctype declaration. Internet Explorer thinks you want XHTML 1.0 Strict displayed by IE7. In order to stop IE doing this, you need to include the following *meta* element in the *head* of your document, directly after the <title>…</title> element.

**<meta http-equiv="X-UA-Compatible" content="IE=edge" />**

**CSS**

You should use CSS for absolutely all formatting beyond bolds and italics. If you’ve learnt HTML this way, good. (These days students who think they know it all generally do. *Of course we use CSS for all formatting! What do you expect?*)

If you’ve learnt HTML using old-fashioned HTML formatting, you’re in trouble. You’re going to have to unlearn nasty habits, starting now. If you don’t, you’re going to fail your assignment. (Students in this category are now fairly rare, but a few years ago they were a major headache, especially when they didn’t realise they didn’t know the stuff. But bad HTML teaching persists in some schools.)

**Choosing an HTML Editor**

There are a range of HTML editors around for you to choose from, and your tutor isn’t familiar with all of them. It’s essential that you (1) understand everything you’re doing with your own web pages, and (2) take responsibility for producing well-formatted disciplined high quality code. If all you’ve got to edit a page with is Notepad, using Notepad should be a minor irritation not an actual problem. All you’re doing is editing a text file. The most important requirement for an editor is that it doesn’t get between you and your text file.

Nevertheless it’s useful to get a bit of support. Useful features to have include: **Colour coding**: the editor changes the colour of the text according to its role in the syntax; it’s very helpful to see clearly what the editor thinks is a string. **Auto-formatting**: the editor indents lines in your document for you (at least if you’ve put line breaks in the right places) into a clear structure where you can see exactly what is inside an element because it’s indented a bit further to the right. **Automatic reformatting**: the editor reformats your page into the layout it thinks you should have; this saves work and is useful for tracking down mistakes. **Templates**: the editor creates files that have the right suffix plus standard content you’ll always need, such as doctype declarations. **Auto-completion**: the editor automatically puts in closing tags to match the opening tags you put in; the editor offers you possible completions for your tag names.

**Aptana Studio 3**: This is the editor I usually use for my own HTML editing. It’s available for free download. The dark colour scheme isn’t to everyone’s taste, but you can turn that off. It does good auto-completion and formatting. The automatic reformatting works well but removes line breaks you want to keep.

**JPadPro**: For many years the DMU standard editor for web pages, and capable of supporting many other languages. The company that produced it has been out of business for some years, so it hasn’t been updated; this has the consequence that the built-in HTML previewer doesn’t recognize relatively recent HTML and CSS features. But it’s still quite a good editor, that makes using templates easy, and does auto-formatting and automatic reformatting well, and turns strings bright red; and if you’re using HTML and CSS features the previewer can’t handle, just look at your pages using an independent browser. The installation kit for JPadPro can be downloaded from Blackboard; you need to put in a licence code, but this comes with the kit. JPadPro opens with three panes; the one on the right is where you write code; you’re very unlikely to ever want the left-hand pane, so feel free to close it.

**Brackets** has its enthusiasts.

I’m not a fan of **Dreamweaver**. If you use it just as a glorified text editor, it’s a good glorified text editor. But the startup costs of learning Dreamweaver are high, and likely to get in the way of learning to write your own HTML, and we want to discourage the use of drag’n’drop WYSIWYG editors, as they will get in the way of achieving the educational objectives of your course. Dreamweaver’s advanced features for creating content by filling in templates, and managing the publication of pages, can wait until you can write your own code competently.

**Problems Accessing Your Own Web Pages**

Remember, you’re not done until you’ve tested your pages by looking them up over the web, using their full URLs. There are problems that only turn up when you do this, that don’t appear if you look up locally-stored pages by their local file addresses. We recommend looking up a page you’re developing via its URL as early as possible, so you can see what it really looks like at each stage.

**Wrong names**. Remember that web servers are very case sensitive, much more so than local file systems. So *Myfile.html* and *myfile.html* and *Myfile.htm* and *myFile.html* are different names. So if your links don’t work and your images don’t appear when you think they should, the usual reason is that the capitalization of the name of a file used by the reference in the web page doesn’t match the capitalization of the name of the file on disk. It’s a good idea to have a systematic policy for how you always capitalize names (or don’t). (If the names are the same, check that the references point to the right folder.)

**Wrong permissions**. People can only read, write or execute your files if they have permission. Normally, you don’t want other people reading your files, but other people reading them is the whole point of web pages. File systems can give ‘read’, ‘write’ and ‘execute’ permissions to ‘owner’, ‘group’ and ‘other’. For web pages, you need to have all the permissions, and give ‘read’ and ‘execute’ permissions to ‘group’ and ‘other’. Normally this is done for you by the file system.

If the web server can find the file you’re asking for, but won’t let you have it, you’ll see a ‘403 Forbidden’ error message. If you see that for your own files, you need to sort out your file permissions. At DMU, the way to do this is to **run** the ***‘Make my public\_html’*** script **again**, by double clicking on the icon on the Windows desktop.

Everything in your *public\_html* folder should always have the right permissions, so you shouldn’t ever have to change file permissions directly for web pages at DMU. However, what normally happens is that files get the permissions of the folder they are created in (that is, first saved to) and keep those permissions if they get moved. So you can run into trouble if you create files elsewhere and then move them to your web server filespace, because they are still private files; this used to be a problem at DMU when we used a Linux-based file system. Sorting out permissions in Windows can be a bit of a faff. However, if you use Linux, you can use the *chmod* command to set the file permission to 755 for a web page rather than 700 for a private file: *chmod 755 Myfile.html*

**Strings**

As you will remember from every programming course you’ve ever done, a *string* is a sequence of characters. In HTML, the values of attributes should be in strings. XHTML absolutely insists on it. Strings can be delimited with either " double " quote marks or ' single ' quote marks.

A good text editor will colour code strings so you can easily see what’s a string and what isn’t. For instance, JPadPro turns strings bright red. You can use bits of your document turning red or turning black again as a signal that you’ve written what you wanted to write, or not.

**Warning about copying strings from MS Word or PowerPoint**. Web browsers will only recognize character sequences as proper HTML strings if they begin and end with old-fashioned plain quote marks that are the same at the beginning and end, like "these". Word and PowerPoint automatically put in smart quote marks, like **“**these**”** or **‘**these**’**, that are different at the beginning and end. These are different characters, and don’t turn character sequences into strings. Replacing these with 'plain quotes' in Word or PowerPoint documents is a pain, which is why we usually don’t do it. If you copy and paste smart quotes from Word or PowerPoint into your HTML document, you keep the smart quotes, so you need to replace them with ordinary quote marks in your HTML editor.

**The W3C Validator doesn’t like smart quotes**. It refuses to validate documents that have characters that it doesn’t recognize. More often than not these turn out to be smart quotes or apostrophes pasted from Word documents that haven’t been corrected. Sometimes they are characters that are invisible because they display as spaces.

**Myfile.html or Myfile.xhtml?**

If you use the ***.html*** file extension, a browser will ignore the doctype declaration and try to display the page as HTML, mopping up after your mistakes as best it can. If you use the ***.xhtml*** file extension, a browser will treat it as an XML file and apply the doctype definition specified by the doctype declaration, and will respond to syntax mistakes by showing an error message instead of the page. If the page is correct XHTML, you shouldn’t see a difference.

There is one strong reason for preferring to use a ***.xhtml*** file extension. Instead of using proper standards all of the time, Internet Explorer tries to figure out which version of IE you wrote your page for, which is disastrous if it gets it wrong, which it will. (One way to avoid this is to use XHTML5, as HTML5 documents are shown in an up-to-date manner.)

If you use the XHTML Strict doctype declaration (which you need to for this module, unless you use XHTML5), and give the file a ***.html*** extension, the most recent versions of IE think you want them to pretend to be IE7. This introduced glitches I'd never seen before in my example web pages that run fine with other browsers, and worked with intermediate versions of IE. To get round this, you can try to tell IE to act its age by putting this meta tag directly after the <title></title> in the head of the document: <meta http-equiv="X-UA-Compatible" content="IE=edge" />

If you want to use recently features of HTML that aren’t in the XHTML specification, test to see whether the page works with either a ***.html*** or a ***.xhtml*** file extension, and ask advice.

**XHTML5**

There is another way… you can have both the advantages of HTML5 advanced features and XHTML discipline… you can use *XHTML5*.

The HTML5 standard allows people to write both old-fashioned sloppy HTML tag soup, and standards-compliant XHTML with the syntax features that are needed for XHTML documents to be correct XML. “XHTML5” isn’t an official standard, but it’s a way to write pages so that they are both correct HTML5 and legal XML documents.

What we need to do to have XHTML5 is declare the document to be both XML and HTML5, thus:

**<?xml version="1.0" encoding="UTF-8"?>**

**<!DOCTYPE html>**

In order to have ‘proper’ XHTML5 you need to serve the page as an XML document, so the browser actually uses the doctype instead of treating it like any other web page. We don’t need that for *CTEC 1412* – what we want is strict validation.

There is a validator at [***https://validator.nu***](https://validator.nu) that you can use to check whether a document is correct XHTML5 – just choose one of the presets that include XHTML5. If you use both an xml declaration and an HTML5 doctype, and use a ***.xhtml*** file extension (see below), this validator will check the document as XHTML5 automatically; if you use a ***.html*** file extension, you need to actively choose the right preset to get the validator to validate according to XHTML5 rather than HTML5. Unfortunately, validator.nu produces output that’s rather harder to read than the usual W3C validator.

This is what we’d ideally like you to do, as it gives you HTML5 plus the pressure to write disciplined correct code and not get sloppy, but it’s maybe a bit tricky right at the beginning.