XHTML Workbook

v.4.15 (XHTML 1.01)

Contents of this workbook:

1. Preparatory

HyperText Markup Language or HTML is the original language of the World Wide Web - the language web documents are written in. It is a subset of SGML (Standard Generalised Markup Language) which is used to describe the general structure of various different sorts of documents. This workbook is about XHTML (eXtensible HyperText Markup Language) which is a version of HTML which is compliant with the rules of XML (eXtensible Markup Language – sorry for all these acronyms!) which makes for better futureproofing and portability of web pages. The workbook is concerned with familiarising you with the things XHTML can do and guiding you to create your own infant web pages!

As its parenthood implies, (X)HTML was originally designed to deal with the structure of documents. It is impossible to create a document with XHTML which you can *guarantee* will appear in exactly the same format to everyone who looks at it. This is because web browsers vary and can be configured by the user to interpret XHTML in different ways on the screen. In other words, XHTML is not really a page description language (like Postscript, for example). Although you can use stylesheets (see chapter 12) to give instructions to achieve the look you want on your screen, it may appear differently (or even be non-functional) on another person's system with a different browser and system settings. When using XHTML therefore, try to think of describing parts of your document in a structured way, rather than simply trying to lay them out in a pleasing fashion.

All XHTML files are simply composed of plain text, and can be composed on any simple text editor or word processor. If you use a word processor to create XHTML files, you must be sure to save them as plain text (ASCII or similar – you may specify the character encoding you have used in the webpage, see chapter 2) or they will not be recognisable. A number of XHTML editors and converters are also available.

This workbook will provide you with all the information you need to compose simple web pages, and comes complete with a select list of tags and their attributes as an appendix. If you are thinking "Why are we using XHTML rather than HTML or XML?" good for you, but you don't have to worry. XHTML is actually 95% the same as HTML, it's simply written in slightly more strict term to comply with the rules of XML and thus improve predictability and portability – XHTML is an XML "application", i.e. a tag scheme for a specific purpose written to comply with the rules of XML.

It's a good idea to also take a look at other examples of XHTML to see how people achieve the effects you see on the screen. If you see something you like and you want to know how to do it, use the View Source option to take a look at the XHTML behind the page and see how it's done. This can sometimes be a bit confusing as nowadays many people include a lot of complex code like Javascript (q.v.) or use tools to build their sites which create messy, impenetrable code, but well-designed pages can often give you good ideas. You can also go to the W3C website to get further information on any particular tags and requirements (see resources section at end) which are not included in the appendix, which are only two dozen pages of the

most common and useful elements, when the full XHTML spec is much bigger! There are also lots of useful guides, and validation tools at the W3C site, so do visit it.

The first thing to do before starting to compose any XHTML document is to think about the context and structure in which it will be found. Few web pages exist in isolation; most are part of a number of linked pages, referred to either as a web *document* (although some use the term *document* as an alternative to *page*) or a web *presentation*. In this workbook, we will use the term *page* to indicate a single XHTML file and the term *document* to indicate a collection of related pages.

In following this workbook you will be able to create a document consisting of a number of pages. Before you start to create your first page, however, it is vitally important to think about the structure of the document it will be part of. What you do initially will probably form only a very simple document, and you may feel a detailed consideration of the structure to be unnecessary; however it is good practice of a very important discipline. Any document of more than two or three pages will benefit from this sort of analysis.

Exercise

You have already been asked to think about the document you are going to create today. Before you start to write any XHTML, draw a structure map, organisation chart or storyboard for your pages, and identify which should link to which. Think about your topic, your users and what they will expect from the document, and which basic method of organisation is best for you: Linear, Hierarchical, Hybrid or Web.

2. Fundamental elements

In XHTML **all code must be in lower case** as XHTML is case sensitive, unlike HTML where tags can be in lower or upper case. All tags in XHTML are enclosed in angle brackets, and generally come in pairs; the first of the pair identifies the position at which the particular element handling starts, and the latter showing where it finishes. The closing tag is distinguished from the opening tag by the addition of a slash, e.g.:

```
<html> ..... </html>
```

XHTML requires that all tags are closed, again unlike HTML, where some tags do not require a closing tag. This of course gives rise to the need for an alternative method of closure for certain tags which are not normally applied around blocks of text, for instance the horizontal rule tag, and in these cases the tag is "closed" by including the forward slash at the end of the tag, e.g.

```
<hr />
```

Tags like this are also referred to (somewhat confusingly) as "empty" tags.

Tags can be nested, i.e. pairs of tags can be contained within other pairs. When this happens, care must be taken as the effects can sometimes be surprising!. There are examples of this later in the workbook, so don't worry about the details just yet - but do remember that pairs of tags should **never** overlap each other, but always be nested, otherwise they may not work properly (this is also a requirement for XHTML compatibility). There are also some logical rules which constrain which tags can appear within which other tags – for instance, you can't put a paragraph tag inside a heading tag – but don't worry too much about these for now.

Each page should always contain a number of standard elements. Sometimes browsers may read pages without some of these elements quite successfully, but it is good practice to include them all, as it promotes compatibility and consistency.

Every page should contain a pair of HTML tags (not XHTML, slightly confusingly… but see more below!), which identify where the main code begins and ends. Within these tags there should be two other pairs of basic structural tags, HEAD and BODY. These delineate the administrative and additional information associated with the file (the HEAD) from the data which represents the page to be displayed on the screen (the BODY). Thus, every page you create should have a basic structure like this:

```
<html>
        <head>
        </head>

        <body>
        </body>
</html>
```

Of course, a lot more will go in the page in due course! The most common data element in the HEAD is the TITLE data. This information is used in a number of ways, most obviously as the document title display in the window title bar of any Windows browser, such as Firefox or Explorer. Note that this data doesn't appear on the page itself: the onscreen "title" must be in the BODY, and is typically designated with a Heading tag of some sort (more on headings in a moment).

When composing a title for your page, make sure you choose a meaningful and descriptive one. It should make clear what the page is independent of context, and not be too long in case the browser can't display it all. The title of the home page of your document might be the same as the "title" you put on the screen; sub-pages may well not be. The best test is to think to yourself "if I saw this title in isolation, would it give me an accurate idea of what this page was about and where it came from?". Never use titles such as "part 2" or "example" as they are totally dependent on context.

So, your document should now look something like this:

```
<html>
        <head>
        <title>Andy Dawson's test document for a Web Course</title>
        </head>

        <body>
        Hello! Here's a little text to try out your first web page...
        </body>
</html>
```

There are some other tags and attributes which can be used in the head section of a web page: please see the appendix for further details of these. But most importantly, we need something in addition to tell the system that our document is XHTML, not plain HTML, and we do this by providing a DOCTYPE DECLARATION, which needs to come *before* the HTML tag. So the first line in your code should be a <!DOCTYPE> tag along the lines of

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
```

(or similar – q.v. W3C, but for now just copy this one) which will tell modern browsers what version of XHTML you have written the page in. Don't worry about what the details mean for now, just make sure you include it exactly as above!

You should also include a <meta> tag in the head of the document which gives information about the character encoding that is used for the page, which should look something like

```
<meta http-equiv="Content-Type" content="text/html; charset=iso-8859-1" />
```

(again, copy this one unless you have a specific reason to do otherwise). This goes somewhere in the HEAD of the document, ideally right at the top of that section.

Lastly, although these are not required, it's best to also include an XML declaration (which needs to come immediately before the Doctype declaration – think about why!) and we should also include namespace information as an attribute of the opening HTML tag (we will talk more about attributes later).

The XML declaration looks like this:

```
<?xml version="1.0" encoding="iso-8859-1"?>
```

and the revised version of the opening HTML tag with a namespace attribute like this:

```
<html xmlns="http://www.w3.org/1999/xhtml">
```

So to sum up: putting all the above together, every page we create can be based on a standard bare template, which should look like this:

```
<?xml version="1.0" encoding="iso-8859-1"?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">

<html xmlns="http://www.w3.org/1999/xhtml">

<head>
        <title>A basic XHTML 1.0 Transitional template</title>
        <meta http-equiv="Content-Type" content="text/html; charset=iso-8859-1" />
</head>

<body>
   <p>… Your HTML content goes here …</p>
</body>

</html>
```

It's not absolutely vital to include all these elements but it's potentially very useful, especially if you later want to validate your pages with tools like those at W3C which require this information before they can process your page automatically.

Exercise

Create an initial web page along the above lines by typing the data in by hand (you can use Notepad or any text editor). Save the file as filename.htm (do not use filenames with spaces, and stick to lowercase), and look at it in your browser by using the File, Open option. Then try adding some more text in the body. Also try including tabs, blank lines, new lines and extra spaces. What happens?

3. Headings and basic layout

As you will have seen from the preceding exercise, text in XHTML isn't handled in quite the same way as you might expect! All ASCII characters which affect the physical layout of the data, such as tabs, carriage returns and spaces, are all treated in the same way by XHTML. Any number of such characters in any order or combination is treated as a *single space character* when displayed by an XHTML browser. If you want text to break or be otherwise manipulated, you must use specific XHTML tags to achieve the effect you want.

This is not to say that you shouldn't use tabs, new lines and spaces in your XHTML documents! Indeed, the use of "normal" page layout, especially indentation and white space, makes an enormous difference to the readability of XHTML source code, and is strongly recommended. The point is that you should use such "normal" layout for exactly that purpose, *to enhance source readability*, but you must use XHTML tags to create those effects on screen.

The two most basic tags for text separation are the Paragraph and Break tags. Paragraphs must be closed (with a at the end of the paragraph in the normal fashion), but break tags are an example of an "empty" tag and therefore occur singly, with a slash at the end. The insertion of a Break tag

causes a carriage return and newline action, and a Paragraph tag

 ….

causes the same action but also adds an additional blank line (and also applies any other defined formatting to the paragraph – see stylesheets chapter 12). These two tags should be used for the simplest organisation of your body text. Until the browser encounters one of these tags it will treat all text as a single, contiguous block.

The other basic tags for the discrimination of text are the Header tags. There are six of these tags, from <h1> to <h6>. Each is applied in pairs with a closing tag </h1> to </h6>, and each gives a specific emphasis and size to the text it is applied to. Heading tags also add a blank line and carriage return by default, to make them stand on their own.

IMPORTANT! Do not use the Header tags simply as an easy method of giving visual emphasis to a particular piece of text in a block. Remember what we said about XHTML being a structural language, and about what you see being dependent on the configuration of your browser? If you do use the headers in this fashion, it is possible some viewers of your page may see something rather different from what you intended. Lynx, for instance, is a perfect example of this, being a non-graphical browser. It interprets different headings not by using large type etc., but by changing the starting position of the text on the page, indenting or centering it. This would still make sense if you were using the tags in a structured way, but not if you're just using

them for emphasis. If you want to adapt the look of a piece of text which is not actually a heading, there are different ways to achieve this (see later).

A good tip is to lay out the heading structure of your page as if you were using an outliner, in a hierarchical fashion, e.g.

```
<h1>Building a house</h1>
        <h2>Clear the site</h2>
        <h2>Lay the foundations</h2>
                <h3>Mark out</h3>
                <h3>Dig the trenches</h3>
                        <h4>Check for services</h4>
                        <h4>Cut trench</h4>
                <h3>Pour the concrete</h3>
                        <h4>Place formwork</h4>
                        <h4>Mix concrete</h4>
                        <h4>Pour</h4>
                        <h4>Agitate</h4>
                        <h4>Cover and cure</h4>
        <h2>Build the walls</h2>

        (etc.....)
```

Then fill in your body text under the headings. Of course, simply laying out your headings in this indented way in your source won't directly affect the appearance on the screen, as we've said, but it will help you to think about the correct structure for the page and the correct use of headings to describe it.

Exercise

Try creating a structured document using Headings, Paragraphs and Breaks. Start with a template document like the one described in the previous chapter. Enter some headings and some body text, marking it up appropriately. It is probably easiest to do a dummy document based on the material you hope to use in your real web pages, but don't force it if it doesn't suit. Feel free to use the example for structure if you're stuck, and add some body text. See how the different Headings appear on your browser, and the effects of the Paragraph and Break tags. What happens if you nest tags within tags?

Do also please use plenty of white space inside your source code - as you have seen, it doesn't affect the end product, and it makes the source much easier to read and edit.

4. Stylistic and other basic tags

So far so good, but although we've learnt to do quite a lot in terms of structuring our document and making headings stand out, we haven't done anything about manipulating parts of the body text. We've been emphasising taking a *structural* view, and you shouldn't compromise that approach. Nevertheless, there are times when physical format or layout is important, or when we need to stress parts of the text. The best way to achieve this is by using stylesheets (q.v.), but there are also a few style tags and one or two other useful elements we should mention here.

There are two basic kinds of style tag: physical style tags and logical style tags. A full list of these is given in the pages at the back of the workbook. The most commonly used of these tags are :

Physical:

.....	Boldface
.....	Italic

Logical:

.....	The Emphasis tag
.....	The Strong emphasis tag

These tags can be used anywhere within the body text of a page. Most character tags may be nested, so for instance

```
<b><i>Really stressed text!</i></b>
```

would appear in bold italic.

Purists would suggest that given the structural nature of XHTML, you should use and rather than and , but in almost every case in practice, they are interpreted in exactly the same way, i.e. is implemented as bold type and as italic, so feel free to use and as they are more intuitive.

None of this helps us out with the physical layout on the page however, which as we said could not be readily addressed by XHTML as it was not a page layout language but a structural one. We can center text with the tag: however, and there are times when physical layout is critical, for instance in aligning tables or columns of figures. We can use tables (see chapter 8) to deal with this, and this is usually the best way to do it, or we can try to do some things with stylesheets (chapter 12). It is also possible in a limited way to control character positioning through the use of the Preformatted (<pre>) tag, at the expense of having to use a monospaced font for display. This is very widely supported, but "deprecated" in favour of other methods (i.e. it's preferred that they are not used), so we will not discuss them further here.

Another tag which is frequently used is the Address tag. It is good practice to always put an address block at the bottom of each of your pages. You can use the Address tag to identify that element, e.g.

```
<address>
Andy Dawson<br />
UCL SLAIS, Gower Street<br />
London WC1E 6BT
</address>
```

It is also usual to include phone/fax numbers, and email addresses, in such a block, and you can modify how it looks with stylesheets (q.v.).

A further device you can use to "break up" your pages into logical elements is the Horizontal rule tag, <hr/>. As you would expect, this tag draws a horizontal line across the screen, and is very useful for visually emphasising breaks between sections etc. The Horizontal rule tag does not need a closing partner tag, simply insert the single tag where you want the rule to be inserted. Many people put a Horizontal rule above their Address to help make it stand out.

One other note here: you can also place *comments* in your source code which will only appear there, and not be printed on the screen. This can be useful if, for instance, you want to put information in the code about the page structure, or reminders about developments you want to make, etc. Comment tags take the following form:

```
<!-- Here is a comment -->
```

i.e. the comment tag is a single tag, *inside which* you place the comment you want to make. NB the tag boundaries include the exclamation mark at the beginning and the dashes, you must include them in every comment tag you make.

There are also further tags and attributes which can control the size and type of font in use (e.g. attributes of the BODY tag, and the FONT tag), but again these are deprecated and the best way to achieve such effects is to use *stylesheets* which are addressed in chapter 12.

Exercise

Expand the body text in your last page, add an address and experiment with some of the above features. Try the Preformatted tag: how does such text look? What happens when you include other tags inside the preformatted block?

By now there should be quite a bit of data on your page. Take the opportunity to tidy it up as source code by adding white space and using comment lines.

5. Lists

Often, data in a web page comes as, or is best displayed as, some form of list. Menus, directories, subdivisions, categories, hierarchies, all have inherent relationships which can be brought out by formatting, and XHTML provides us with several kinds of list tags to help us make the best presentation of such data.

The most commonly used list types are Ordered lists, Unordered lists, and Definition lists. You may also hear them referred to as numbered, unnumbered and glossary lists respectively.

Lists begin and end with a pair of opening and closing tags which define the type of list enclosed within. These tags are:

 ..... Ordered list
 ..... Unordered list
<dl> ..... <dl> Definition list

Within the boundaries of these tags, individual list elements are delineated by secondary, unpaired tags:

 ..… For elements in an Ordered or Unordered list

<dt> ….. </dt> For the terms in a definition list
<dd> ….. </dd> For the definitions of the terms in a definition list

Let's see how these different lists actually work. For an Ordered list, the XHTML might look something like this:

```
<ol>
    <li> The first list element </li>
    <li> The second list element </li>
    <li> The third list element </li>
</ol>
```

An Unordered list would look the same, except that the outside tags would be <ul> instead of <ol>. The effect of this code in most browsers is twofold: firstly, the output list elements are numbered (in an ordered list) in order of appearance, and secondly, they are indented. Each new <li> tag indicates the next line, so
 or similar tags are unnecessary at the end of each piece of data. Each <li> tag also causes the next number to be applied. You do not have to specify the numbering explicitly; the browser adds the numbers starting at one by default. If you subsequently add extra items within the list, it is renumbered automatically.

With an Unordered list, the effects are the same except that instead of sequentially numbering each element in the list, they are bullet-pointed instead.

The actions within a Definition list are slightly different. As its name suggest, this was intended for two-part items such as list entries where there is a term and an associated definition for that term. They can be used wherever a list with associated elements is needed. The format of a Definition list might look like this:

```
<dl>

    <dt>Telnet </dt>
        <dd>The Internet's remote login protocol. </dd>

    <dt>Gopher </dt>
        <dd>A menu-based Internet browsing system </dd>

    <dt>World Wide Web </dt>
        <dd>A wonderful thing which uses XHTML!!! </dd>

</dl>
```

Each <dt> has a <dd> associated with it, and all are enclosed within the <dl> tags.

Lists may be listed within other lists, of the same or different types. Different browsers interpret nested lists differently, sometimes e.g. using different bulleting symbols or "numbering" sections with letters (a, b, c, ...) by default. It is possible to identify preferences for such things by using styles or attributes in later versions of XHTML,.

If you place <li> tags outside pairs of list tags you will still typically see an indenting effect. However, this should not be done for the same reasons of interpretation and structure vs layout that we have mentioned before – it is not only unreliable, but creates technically invalid XHTML.

There are some further, advanced attributes you can use with lists: see the appendix for further details.

Exercise

Create a new page which incorporates at least one list. Ideally, this should be a page which fits in with your general document plan, but if that isn't convenient, just make one up. Ideas you could use would be a contents page, a glossary, a parts list, or a set of instructions for doing something. Don't forget to include the standard elements we've already discussed!

Make a second level within one part of your list (a list within a list) if you haven't already done so. Experiment with the way the browser handles different combinations. If it doesn't do what you want/expect, can you arrange things differently to get the structure you want? What attributes allow you further control (check the appendix)?

Try adding other tags within lists and see what happens.

6. Links

We've now learnt how to do most types of manipulation of text on a web page. The next step is to learn how to do what web pages are all about - making links to other web pages!

Links to other resources in XHTML are accomplished by the use of yet another tag, the slightly confusingly-named Anchor tag. Anchor tags are a bit more complicated than the ones we've been using so far, and they make constant use of something we haven't much mentioned yet, tag *attributes*. Fundamentally though, anchor tags act like any other tags, and come in pairs:

<a> .....

Anchor tags are basically used for doing two things: making links to other places, and identifying places to which links can be made! As we said, Anchor tags come with a number of attributes, parameters if you like, which define things like where a link points to, and whether an Anchor is a "go to" link or a "come here" point.

The most basic form of anchor is a simple link to another web page, and might look like this:

 Andy's home page

Look closely at this anchor. It might not look like a normal pair of tags, but it is. It's easy to see the closing tag , but the opening tag is very different from what we've seen so far. You can see the first angle bracket and the A, but then there's this funny HREF bit. Look down the line, and you will see the end of the opening tag after the closing quote. The opening tag is not just the angle brackets and the A, it also includes the HREF. HREF (Hyperlink REFerence) is an attribute of the Anchor tag. It allows us to state where the link points to by giving the URL (Uniform Resource Locator - like an extended sort of pathname) or location of another resource.

So, *where* you want to go is included as the HREF attribute inside the opening tag. What happens to the text *between* the tags? That text becomes the link on the page. If an HREF attribute is given, the text (here, "Andy's home page") will be highlighted within your page, and made clickable or selectable. Activating it will cause the browser to follow the link to the HREF given.

There are quite a few attributes which can be used with Anchors (and several other tags also have attributes - see the list at the end of the workbook for more details) but we will limit ourselves to just the basic ones for now. Another attribute for an Anchor is the NAME attribute. The most basic form of link takes us to another web page, but we can be more specific than that. If we have a page with several elements in it, we might want to be able to jump directly to a part of it, like going to a chapter in a book. If we want to do this, we have to put a marker in the text to show where those "reference points" might be. We do this with the NAME attribute, like this:

The middle of the text

This text will not be clickable, as there is no HREF attribute in the tag. However, imagine it was in the text of my home page, half way down; I could then have a link somewhere else to jump, not to the top of my home page, but to that particular point on it, like this:

The middle!
^^^^^^^

This Anchor has the same HREF as the previous example, except that it has been extended with a # and the name of the anchor we made above. This is what is known as an internal link – one to a point *inside* a given page.

A common use of internal links is within single pages, where you might have a contents list at the top with links to points further down the page, like this:

```
<h2>Contents of my home page:</h2>

  <ul>
    <li>Stuff about <a href="#chocolate">chocolate</a> </li>
    <li>Stuff about <a href="#work">work</a> </li>
    <li>Stuff about <a href="#web">the World Wide Web</a> </li>
  </ul>
```

and then, in appropriate places further down the page:

```
<a name="chocolate">Chocolate</a> is my favourite pastime...

<a name="work">I work as</a> a lecturer at UCL.....

Here are some sources for <a name="web">good XHTML editors</a>...
```

Note that, within the same page, the HREF attributes simply need the # and the name of the Anchor; which brings us on to another important point, that of *relative* and *absolute* references.

An absolute reference is one like our first example, where the location of the resource is spelt out in full using its URL, giving the type of resource (the "http" bit), the system it's located on ("www.ucl.ac.uk"), where in that system (~uczcw11) and the name of the file (home.htm). As we said, this is rather like an extended version of a pathname, as you would find in DOS or UNIX.

Those of you familiar with DOS or UNIX however, will know that you can use a *relative* pathname as a shorthand way of identifying the location of something. The concept works like this in XHTML: imagine you retrieve my home page. The system knows the path it had to follow to find my file (home.htm), and remembers it. If I want it to get something from the same place, I don't have to repeat the full pathname; it's already pointing to the right place. If I just give it a filename, the browser will assume it comes from the same place as the last item and know where to look.

Likewise, if something isn't exactly where the last thing was but is nearby, you can just specify that part of the path which is different, extending or backtracking from where you last were. For example, imagine you retrieved the file

http://www.ucl.ac.uk/~uczcw11/slais/slais.htm

and that there was a directory below the *slais* directory called *projects*, in which there was a file called *myproj.htm*. You could have a link in the file slais.htm to retrieve myproj.htm which looked like this:

My project

Or you could link to the *home.htm* file we mentioned before, in the parent directory *~uczcw11,* by using a backward relative path:

My home page from a different way!

The "../" means "one directory level higher. "../../" would represent two levels, etc.

You should always use relative links between pages in your own web document. The big plus of using this approach is that if you want to move all the pages in a document from one place to another, if you have used relative links, the internal references will still all work with no modification. If instead you used absolute links, you would have to go into every page and edit every link to the new location! Only use absolute links when you need to load a document from a different location from your own material (i.e. generally, if you want a link to someone else's website)

One last important tip about names and paths: Although a Microsoft environment (i.e. Windows) allows spaces in filenames and is case insensitive, many server environments using Unix **don't**. Therefore it is good practice to keep all filenames (and subdirectory names, if you use them) in lower case only, and not to use spaces in them (use an underscore instead if you feel the need).

Exercise

Now it's time to put the hyper in your text! You should have at least two pages completed by now. Make a link from one to the other and back again. These should be relative links as you're working on a local disk here, not through a server (so don't include a pathname, just the filename). Create a third page and incorporate links to that too. Browse around the net and put in a couple of links to external sources also - these will have to be absolute links. Don't forget to test them and see if they work! Also, try creating a NAMEd Anchor inside one or more of your pages and linking to it from somewhere else within the page. See what happens when you activate the link.

Think about the structure of your document, with particular regard to "return" and "escape" routes. Can you find a standardised way to do this which might help users navigate around your pages? Look at some other peoples' pages to get some ideas, and try to implement them. Hint: you'll almost always want a link back to your home page...

7. Images

OK, so we've got the text side pretty much wrapped up. What about images? Including images is fairly simple, but there are some key points to remember. Firstly, try to use GIF or JPG format images wherever possible - they are the most commonly readable and should be fine in any browser. Secondly, remember that pictures take up a lot of transmission time - don't use them unnecessarily (but you can allow yourself free rein today - it's practice!). Thirdly, remember that the browser of the user who views your pages may not have the same capability as yours to display colours or resolution - so keep your images simple and with a minimum number of colours (this will also help to keep the data size of the images down, making them download faster).

To incorporate an image on your page, what is called an *inline* image, you simply use a single tag with some attributes, the <img> tag. The key attribute is the SRC attribute, which defines where the image is. Imagine you have a picture called *rabbit. jpg* in the same directory as your page; you would display it with the tag

which is a relative reference, just as we discussed before with links. You can point to a picture elsewhere in exactly the same way using an absolute reference instead.

Simply getting the image to appear is easy - but getting it in the right place is more difficult. An image is treated just like text, in that it needs
 or <p> tags (or Headers, etc.) to separate text from it (assuming that is what you want to do). Another attribute which affects the screen display of images is the ALIGN attribute, which has five possible values, TOP, MIDDLE, BOTTOM, LEFT and RIGHT. The first three dictate whether the image is aligned to any adjoining text by its top or bottom edge, or middle; the latter two place the picture in the left or right margin (respectively) and run the following text down the side of the image.

Apart from just decorating the page, images can be used as links themselves; i.e. if you click on the picture, it activates a link. The format for such an "active image" is a combination of an ordinary Anchor tag with the Image tag: for example

The Image tag takes the place of what would normally be the highlighted text, inside the Anchor tags. The above is actually a bad example: it is always a good idea to include a text line adjacent to the linking image, and also another attribute ALT which contains text which is displayed instead of the image to users of non-graphic browsers or others who have turned images off. A better version would thus be something like:

My home page

You can make use of images as icons for choices, or coloured lines as alternatives to horizontal rules, or even as navigation icons.

Another common use of images is to have small versions ("thumbnails") of pictures on your page which, when clicked on, load a full-sized version of the image. This is an excellent way of providing access to graphic resources, as it gives a good visual guideline to what's available without having to transfer enormous volumes of graphic data which might be unwanted.

If you look up the IMG tag in the appendix, you will see that amongst other attributes which allow you to control borders and positioning of images, you can resize images dynamically using the HEIGHT and WIDTH attributes. NB you should **never** do this to reduce a large image to a small size! If you do, you are simply downloading data which you are then throwing away, which is extremely inefficient and slows everything down. If you have a big picture and you need a small version (as for a thumbnail, above), use image processing software to create a shrunken version of the original and save (and access) it separately.

Exercise

Try adding some graphics to your existing pages, both as inline images and as links. There may already be some images available locally which you can use, but you can always download images from elsewhere to use too. However - remember copyright issues! Never use a downloaded image from the net without first asking permission (unless it comes from a copyright-cleared or copyright-free source - there are quite a lot of image libraries containing material like this which can be used freely).

Experiment with the effects of the ALIGN attribute, and putting text either side of an image. How do
, <p> and Header tags affect the display of images?

Forget the normal rules of keeping graphic content down for a while. Why not create a picture page? Play around and enjoy!

8. Image maps

Image maps (another popular aspect of web pages) were traditionally made available by server-side processing, but with V4 of HTML tags were introduced to allow client-side processing of image maps, which can be prepared quite simply. It works by in effect “overlaying” an ordinary image with a transparent “map”, based on pixel co-ordinates, which make parts of the image become clickable. In order to create an image map, therefore, you will need a pre-existing image (any format will do), and you will need to know the co-ordinates of the areas which you wish to make clickable on the picture. One way of working out the co-ordinates of parts of your picture would be to load it into an image processing program, which always has the ability to display the co-ordinates of the cursor position. Alternatively, if you have a simple image and you know its overall size, you may be able to calculate the co-ordinates for simple shapes in your head.

Where you want the image to appear, include a normal image tag like the example below:

```
<img src="image.gif" usemap="#mapname" width="120" height="300" />
```

The SRC attribute, as normal, identifies the image you wish to use. The WIDTH and HEIGHT attributes are optional, but can be useful to remind you where the limits of the picture are - when you set your clickable areas (see below), if you set one outside the visible area of the picture, it will be unusable! The key attribute for the imagemap is the USEMAP attribute, which points to a named block of code (a MAP block, called “mapname” in this example) elsewhere in the page which defines the clickable areas.

So in this example, somewhere else in the file, there would be a MAP block something like this:

```
<map name="mapname">
  <area shape="rect" coords="10,10, 100,100" href="file1.htm" alt="rectangle" />
  <area shape="poly" coords="10,110, 20,110, 20,200, 100,200, 20,200, 30,150,
20,120"
    href="file2.htm" alt="Rather unusual shape" />
  <area shape="circle" coords="50,250,25" href="file3.htm" alt="circle, radius 25" />
</map>
```

Everything between the MAP tags are part of the definition, and the name of the definition (referred to in the IMG tag USEMAP attribute) is specified in the MAP tag by the NAME attribute.

Within the block you can include a number of AREA tags which define the actual clickable areas. As can be seen in the above example, there are three different "shapes" which can be defined with the SHAPE attribute: RECT (for a rectangle), POLY (for a polygon), and CIRCLE (for a circle). Each is then followed by a set of values in a COORDS attribute. The values vary according to the shape: for a rectangle, two pairs of x-y co-ordinates are used, specifying the top left and bottom right of the rectangle; for a polygon, any number of pairs of x-y co-ordinates can be specified to define the shape, which results in a line being drawn from point to point in the order listed, with the final point being linked back to the first one; for a circle, three values are required, first a pair of x-y co-ordinates which represent the location of the centre point of the circle, and then a single value representing the radius of the circle.

Each AREA also includes an HREF attribute which, of course, contains the location of the file to be loaded if the user clicks within the specified area.

A MAP block can contain any number of AREAs, in any combination of shapes. However, when constructing image maps, be careful to ensure three things: firstly, that all clickable areas can actually be shown on the image (i.e. don't plot an area which falls outside the boundary of the image); secondly, that the areas do not overlap (otherwise which one will work?); and thirdly, make sure you provide elsewhere in your page an alternative route to the locations in your map which are text-based, for anyone who cannot see your image.

Exercise

Make a simple client-side imagemap to link to other pages in your site (or elsewhere). Remember to cater for people using older or non-graphic browsers...

9. Tables

Table tags are relatively simple in their basic form, however, even if somewhat longwinded to write. They take the form of an overall <table> tag , followed by <tr> (Table Row) tags to create the vertical divisions, and then <td> (Table Data) tags to delineate the individual cells on the row, into which you can enter the data you want to show. Unfortunately, this means that you need to create a pair of <td> tags for every cell in your table! That could be a lot of typing... (but there's a hint below!)

A simple 2x2 table is then prepared thus:

```
<table>
        <tr>
                <td>Data 1,1</td>
                <td> Data 1,2</td>
        </tr>
        <tr>
                <td> Data 2,1</td>
                <td> Data 2,2</td>
        </tr>
</table>
```

The use of the BORDER attribute in the TABLE tag draws a border around the cells (e.g. <table border>), and you can specify a value (e.g. border="5") in pixels if you want to adjust the width. You can also incorporate heading cells with the <th> tag and adjust data alignment etc. with attributes: see the appendix for the full range of options.

NB you may want to have cells which are simply blank, with no data in: if so, you must include a pair of <td> </td> tags with nothing in between. You mustn't leave them out or you will make your table lopsided! You can otherwise format the data in table cells just as you would any other text in an XHTML page, including making it a link.

One BIG practical hint when making tables in native XHTML: draw up your basic table structure empty, making copious use of copy and paste, then go back to fill in the data afterwards! This not only speeds up the process of table construction but helps to prevent errors of table structure. In anything but a small table, it's also a good idea to put comment tags in to identify each row (or an ID attribute in the TR tags) and make it easier to see where you are in a mass of table structure.

Quite a lot of control can be had over table appearance and layout: a more complex table example is given below, showing the use of various attributes, multiple column and row headings etc. (this html example is taken from Laura Lemay's book, fig. 11.26 on p340, and viewable online at www.ucl.ac.uk/~uczcw11/bscb001/lltablexample.htm)

```
<table border="5" bgcolor="#FFFFCC" bordercolor="#CC9900" cellspacing="0"
cellpadding="5">
  <caption align=="left">Drive Belt Deflection</caption>
  <tr align=="center">
    <th rowspan="2" colspan="2"> </th>
    <th colspan="2">Used Belt Deflection</th>
    <th rowspan="2">Set<br>
    deflection<br>
    of new belt</th>
  </tr>
  <tr align==“center”>
    <th>Limit</th>
    <th>Adjust<br>
    Deflection</th>
  </tr>
  <tr align==“center”>
    <th rowspan="2" align==“left”>Alternator</th>
    <td align==“left”>Models without AC</td>
    <td>10mm</td>
    <td>5-7mm</td>
    <td rowspan="2">5-7mm</td>
  </tr>
  <tr align==“center”>
    <td align==“left”>Models with AC</td>
    <td>12mm</td>
    <td>6-8mm</td>
  </tr>
  <tr align==“center”>
    <th colspan="2" align==“left”>Power Steering Oil Pump</th>
    <td>12.5mm</td>
    <td>7.9mm</td>
    <td>6-8mm</td>
  </tr>
</table>
```

Exercise

Create a simple 3x3 table using the tags described above. What happens if you miss out a pair of <td> tags?

Create a table appropriate to your website. Experiment with the effects of various attributes. Review the tags and attributes in the appendix, and add some headers to your columns. Try creating some “nested tables” (tables within tables).

10. Forms

A further element in XHTML is the use of Forms. By using the Form tags and their many attributes, you can produce pages which contain fill-in, drop-down or all sorts of other fancy entry methods for users to enter data, including things like "radio buttons". You can incorporate form layouts in your pages today, but you will be unable to *process* the data which is entered, which is usually the point: forms are normally used e.g. to get search queries, collect key data etc. which is then passed to a computer for something to be done with it, and usually, for some other information (e.g. the results of that search) to be passed back to the originator. See the final chapter for more information on how this is normally achieved.

However, there is one additional element which can be used with forms to collect data in simple form without the use of complex processing, and which can also be used independently as a communication device. This is MAILTO, used in Anchors to call up a simple electronic mailer, or in forms to have entered data emailed somewhere. By using MAILTO, you can have links which allow people to immediately send email to you (or whoever you specify). An example of its freestanding use would be:

```
<a href="mailto:andrew.dawson@ucl..ac.uk">Mail me here!</a>
```

In a form, you first have to define a form with a pair of FORM tags which will include attributes which tell the system what to do with the data. To mail the responses to yourself with MAILTO, you will need to set up your basic form using

```
<form method=post action="mailto:andrew.dawson@ucl.ac.uk">
.....(form elements here).....
</form>
```

There are many other things you can do with form data, but these are beyond the scope of a simple introduction to deal with. You can get some idea of the possibilities by looking at the FORM tag attributes listed in the appendix.

As for collecting information in a form, there are a number of ways you can do it. Text you want to appear as prompts on your form is mostly entered normally: input data can be gathered in several different ways, but normally by the INPUT tag.

In its basic form, INPUT takes two attributes, NAME and TYPE. NAME is the identifier for the piece of data when it is passed on: it is like a variable name in a programming language, a holder for the data. TYPE tells the system what kind of data element this is. There are several possibilities, the most common being "text" for text-entry fields, "RADIO for 'radio' buttons, and "CHECK" for check boxes." Thus a simple text-entry box can be generated using this tag (within the overall FORM tags):

```
Enter your name: <input type="text" name="namefield" />
```

Normally, you may want also to restrict the size of the text entry box (usually referred to as a "widget") and the amount of data which can be input. This can be done with SIZE (to set the size of the box) and MAXLENGTH (to set the maximum length of input string) attributes. You can also draw a text box of multiple lines rather than have a one-line box using the <textarea> tag (see appendix for more details).

Radio buttons (type="radio") normally used to allow a single choice from a list of items. To do this, mutually exclusive groups of buttons are indicated by using the same NAME but different VALUEs; e.g.

```
<ol>
<li><input type="radio" name="Material" value="Animal" />Animal</li>
<li><input type="radio" name="Material" value="Vegetable"/>Vegetable</li>
<li><input type="radio" name="Material" value="Mineral" />Mineral</li>
</ol>
```

would produce a list of three buttons labelled animal, vegetable and mineral, only one of which could be selected. The data in the VALUE field is sent attributed to the NAME given if the button is checked: because all buttons have the same NAME only one can be selected. By default all buttons are off to start with. The text outside following the tag is simply displayed next to the button like a label.

Check boxes are similar to radio buttons, but are typically not mutually exclusive (so more than one can be checked). To recreate the list above as checkboxes:

```
<ol>
<li><input type="checkbox" name="Animal"/>Animal</li>
<li><input type="checkbox" name="Vegetable"/>Vegetable</li>
<li><input type="checkbox" name="Mineral"/>Mineral</li>
</ol>
```

NB that now we have three different NAMES and no VALUES. Giving different NAMEs allows us to check more than one box simultaneously, and we don't have to supply a separate VALUE if the NAME is unique – if the box is checked that item is there.

There is actually nothing to stop us using radio buttons for non-exclusive choices and checkboxes for mutually exclusive ones – it's all down to how we assign the NAMEs and VALUEs to the tags – but conventionally they are used in the illustrated ways.

An alternative (and perhaps neater) way of dealing with selecting one item from a list is to have a drop-down menu: this is handled by using the <select> and <option> tags. E.g. to have a dropdown version of the above:

```
<select name="Material">
<option>Animal</option>
<option>Mineral</option>
<option>Vegetable</option>
</select>
```

The different OPTION entries are treated as the VALUEs in the previous examples. Notice also the similarity between the use of <select> and <option> and list tags n terms of structure. .

Finally, you've got your data, how do you send it off? The simplest choice is to include a "submit" button, which is done with another INPUT statement using TYPE submit, like this:

```
<input type="submit" value="Send it off!" />
```

The value here is the text which will appear on the button. Normally, all the data collected on a form is sent to a program or script (see chapter 11) which then processes it in some way, but unfortunately this kind of processing is beyond our abilities to provide during the practical session.

Here is a complete form using the above elements (based on Lemay's "Surrealist Census", viewable at www.ucl.ac.uk/~uczcw11/bscb001/llformexample.htm)

```
<form method="post" action="mailto:name@address.co.uk">

  <p><strong>Name: </strong><input type="text" name="theName"></p>

  <p><strong>Sex: </strong>
   <input type="radio" name="theSex" value="male" />Male
   <input type="radio" name="theSex" value="female" />Female
   <input type="radio" name="theSex" value="null" />Null
  </p>

  <p><strong>Contains (Select all that Apply): </strong><br />
   <input type="checkbox" name="humor" />Vitreous Humor<br />
   <input type="checkbox" name="fish" />Fish<br />
   <input type="checkbox" name="glycol" />Propylene Glycol<br />
   <input type="checkbox" name="svga" />SVGA Support<br />
   <input type="checkbox" name="angst" />Angst<br />
   <input type="checkbox" name="catcon" />Catalytic Converter<br />
   <input type="checkbox" name="vitamin" />Ten Essential Vitamins and
Nutrients<br />
  </p>



  <p>Select a hair colour:
```

```
  <select name="hcolor">
    <option>Black</option>
    <option>Blonde</option>
    <option>Brown</option>
    <option>Red</option>
    <option>Blue</option>
  </select>
 </p>

 <p><input type=“submit” value="Submit Your Votes" />
  <input type=“reset” value="Clear Form" />
 </p>

</form>
```

Exercise

Design a simple form appropriate to your website, set up with a “Mailto:” action (N.B. this may not work today as it requires the browser which uses the form to be configured with valid user information and email parameters – which will normally be the case for real users!). Note the differences in effect of the different input types. Which might be suitable for any data capture you might want to achieve? Check the appendix for some of the attributes which can be applied to certain elements to control e.g. size of text input boxes.

11. Frames

Frames are less popular on web pages today than they once were, and their use is deprecated. They can be problematic unless used with care, but are still sometimes useful. They are a way of breaking up your page into multiple mini-pages, rather like having a series of concurrently open windows in a graphical operating system environment, so that you can have different things going on in each one at the same time - or more precisely like having several versions of your browser on the go at once!

One big problem with frames is that they can be impossible for users to bookmark effectively. The frames are also (obviously) smaller than a normal full page and therefore you can't see so much in them.

The author's personal viewpoint on the use of frames is - avoid using them in most circumstances! They seldom add much to a page that can't be achieved by other means (like tables), and unless very carefully designed, can lead to all sorts of trouble for users. However, to allow you to experiment with them, a full listing of the relevant tags and attributes is given in the appendix.

In essence, a framed page resides within a <frameset> definition, which can be broken into individual <frame>s or broken into columns and rows (and otherwise controlled) by attributes. You will also need to NAME your frames, and in all pages that are intended to be used within them, add TARGET attributes to all the <a>anchor tags within them to control where the data actually appears when clicked. If you are really keen to work with frames, the best thing is to consult the relevant chapters in a good XHTML guide.

Here is an example simple frame definition:

```
<frameset cols="150,*">
        <frame src="file1.htm" name="framename1" />
        <frame src="file2.htm" name="framename2" />
</frameset>
<noframes>
        XHTML for frame-free version goes here - don't forget it!
</noframes>
```

This defines a vertically split screen, with a fixed, 150 pixel column on the left, and a variable-width column occupying all the remaining available space on the right. The SRC attributes specify which XHTML files will be initially loaded into each frame: the NAME attributes specify a name for the frame which can be referred to by TARGET attributes in links within the XHTML files. NB do make sure you create the files you specify to go in the windows or nothing will happen!

You should also always enter data between the NOFRAMES tags. This should be some standard XHTML (NB the page as a whole is a frameset, it doesn't go within the normal XHTML set of tags, but replaces them - you should put a BODY inside the NOFRAMES element) which is for any user whose browser can't use frames.

To mix rows (ROWS) and columns (COLS) in a frameset definition, define a second frameset within a frame (i.e. replace one of the FRAME tags with a complete, new, nested FRAMESET, e.g.

```
<frameset cols="150,*">
        <frame src="file1.htm" name="framename1" />
        <frameset rows="100,*">
                        <frame src="file2.htm" name="framename2" />
                        <frame src="file2.htm" name="framename3" />
        </frameset>
</frameset>
```

Definitions for the space allocated to rows or columns can be specified by absolute width in pixels (number), % of space (%) or available space (*). If multiple asterisks are entered as values, available space will be distributed equally between all elements so identified.

It is crucial to note that, if you click on a link inside a given frame, the new file will load into the same frame by default (it's just like a normal browser window). To load a file from a link in one window into *another* window you must specify a TARGET attribute (target="framename", where framename is the name of the frame you want the data to appear in) in the link.

Exercise

Create a frameset that divides the browser window into three parts as follows:

- Left section - a column the height of the browser window and one third of the width of the browser window, named "contents"
- Right section(s) - divided into two rows, each half the height of the browser window, named "top" and "bottom" respectively

Now create a page for use as a table of contents in the left frame: create two links on the page, one to load a page in the top window, one to load a page in the bottom window.

Experiment with other frame layouts and pages suitable to your topic.

12. Stylesheets

Stylesheets are the best way of providing a broader and more flexible way of controlling the "look and feel" of the contents of (X)HTML pages. They are actually written in a language all of their own (indeed there are several varieties of stylesheeting language).

Stylesheets are a complex subject, almost worthy of a course on their own! A cribsheet is provided separately which covers some common parameters of CSS1 (a common stylesheet language which we will be using), but for a full specification of both CSS1 and stylesheets you will need to visit W3C's website - do NOT look for CSS or stylesheet tags in the appendix as they're not there! There are very many things you can do with advanced stylesheeting, but we are going to limit ourselves to the basics of modifying how individual tags and blocks of text look.

Before we proceed, we need to mention that colour is conventionally expressed in CSS as three bytes holding red, green and blue values, expressed in hexadecimal preceded by a hash (e.g. #ff0000 – red) and using the American spelling "color". Links to tools which give the hex values for colours are given in the resource list on the last page of this workbook.,

Here is an example of a very simple style sheet:

```
<style type="text/css">
<!--
body {  background-color: #ccffff; font-family: Arial, Helvetica, sans-serif; color: #330066}
  a:link {  color: #cc9900}
  a:visited {  color: #660000}
  a:hover {  color: #ffcc00}
  a:active {  color: #ff0000}
-->
</style>
```

This stylesheet simply sets up some standard settings for body text in an XHTML file: it is written using the CSS1 specification (Cascading Style Sheets 1) for stylesheet language, hence the TYPE attribute in the STYLE tag saying "text/css". You will see that, within the STYLE tags, the whole of the actual style definition is enclosed in an XHTML comment tag: this is because, as we said, the actual style definition is not written in XHTML but in CSS1, and we don't want it accidentally displaying on the page if the browser doesn't understand styles. You will see a close relationship between the attributes used, although the names are not exactly the same.

If the above block were included in the head of a file of XHTML, the body within the file (and all the A tags) would be treated as shown above. However the real value of stylesheets are that they can be used *independently* of individual pages. A style definition can be saved separately from any XHTML files as a freestanding definition,

and then referred to by other pages by including a special kind of link in the head of a document. For instance, if we had saved the above example (Just the CSS bits, minus the actual XHTML tags) as a separate file called "mystyle.css", and we had an XHTML document we wished to have use that style, we would add a line

<link rel="stylesheet" href="mystyle.css" />

which would go in the HEAD section of the document. This would then enable the document to use the style specified in mystyle.css, just as if it had been written into the document itself. NB this example is using relative addressing, i.e. it assumes the "mystyle.css" file is in the same directory as the XHTML file, hence it is identified by name only. You may also use an absolute address here if you need to (a full URL).

This is obviously a hugely useful facility, as it enables you to provide consistent "look and feel" across a range of documents, and to make any style changes to all of them at once (in effect) by simply changing the separately-held style definition.

There are a range of other facilities relating to stylesheets which can also be useful, for instance the ability to define different *classes* of a tag, so that different special effects can be created: e.g. if you had defined a standard way of presenting an H1 tag, perhaps in red, you could create some variant classes H1.green and H1.blue so that (if you wanted to) you could have them display in other colours in specific places by adding an attribute class="green" (or blue) to the tag in question. You can also apply style attributes directly to tagged data within a document, know as *inline* style, to apply a particular effect at a particular point as a one-off. A STYLE attribute can be applied to almost any tag, in the following form:

Red and bold!

This example is applied to a P tag, but the same principle applies to all other tags. The basic format of the style definition is the same as that applied in the separate style block at the beginning of this chapter: i.e. each style parameter is expressed (in CSS1) in the form of

parameter: value; parameter: value (,value, value)

so, each parameter is followed by a colon and then its value, and if multiple values are given (a preference list) each subsequent value is separated from the last by a comma. Each parameter and its associated value(s) are separated from the next by a semicolon.

But what if you want to apply some style to text which is not identified by tags, e.g. a single word in a paragraph, or to a block of materials containing other tags? Two additional tags exist for this purpose: and . If you wish to modify text which does not contain other tags (e.g. again the words within the paragraph) use with a style attribute. If the block contains other tags (e.g. a heading plus some paragraphs) then use similarly. Please note that can also be used for more advanced purposes not covered here, such as creating floating menus – see the CSS specification for further information.

The nature of Cascading Style Sheets (CSS) means that, within a document, inline styles take precedence over an internal style definition, which in turn takes precedence over an external style definition. Thus you can set a general scheme in an external style definition for a document which can be overridden, in whole or in part, either at the page level (by an internal style definition) or at the element level (by an inline style definition).

Full and up-to-date definitions of CSS1, CSS2 and other stylesheeting languages can be obtained from the W3C online (see the resources section).

Exercise

Create or load some simple web pages and experiment with style rules. Create an external style sheet and apply it to a series of pages. Also try an embedded (internal) style sheet and some inline style definitions. Use an online color palette tool to identify the colour codes you need (see the resources section at the end of this workbook for where to find these).

13. Other elements and considerations

There are quite a few other tags, attributes and options available within and without XHTML which go beyond the scope of this workbook and today's opportunities, which we should make some mention of. Do look through the list of tags at the end of this workbook, and (if you're feeling keen) take a look at the full specifications for HTML 4.01 and XHTML, which you can find at the W3C website (www.w3c.org).

The mechanism which performs server-side processing (for instance, enabling us to run a search on a database from a webpage) is normally something called a Common Gateway Interface (CGI) Script. or an Applications Programming Interface (API). It is like a computer program which has been written to expect data from a certain form, process it, and to create an (X)HTML-based reply page which incorporates the output data. Writing CGI or API scripts is not a simple matter for the uninitiated, and more importantly, requires appropriate software to run under, which is not available here today. Unless you have some background or experience in programming, and you are interested in this aspect, it is definitely recommended that you get some specialist help if you want to proceed with it.

Similarly, some effects (rollovers for instance) on webpages are achieved with client-side processing, by the inclusion of Javascript applets (mini-applications) embedded in web pages. Again, these are actual programs written in Javascript language, which also is well beyond the scope of this course element. If you are interested in learning how to use Javascript, you will really need to undertake a separate course on it.

Do remember that you can view other people's code by selecting the View Source option in your browser - this can help you to understand how certain effects are achieved.

Exercise

If you have not already been working on parts of your planned web pages, now is the time to start putting together a real first attempt at your intended product. Look through the lists of tags at the end of this workbook, and try incorporating any of the other tags and attributes you think might be appropriate.

Don't forget to look at other peoples' ideas and code, and to draw on and learn from them.

At the end of the day, make sure you copy your files to a floppy disk and take them with you, so that you can continue to work on them and use them later elsewhere!

And lastly, if there are any things which you want some assistance with - **please ask!**

Resource list

Online resources:

W3 consortium: http://www.w3.org
HTML 4.01 spec: http://www.w3.org/TR/HTML4/
XHTML 1.0 spec: http://www.w3.org/TR/xhtml1/
CSS1 spec: http://www.w3.org/TR/REC-CSS1
The W3C site has endless additional resource material, from style guides to links to editor and browser downloads, validators, and absolutely everything you will ever need! It's absolutely THE source for web materials and information.

Colour mapping information and tools:
http://www.klixxx.com/tools/html_colors.shtml or http://websitetips.com/color/tools/

Special character codes

Any special character can be specified using the form &#?; where ? is the coded value of the character. Many common special characters have "named" alternatives also. Some of the latter are given in the list below.

*NB all character code values ultimately depend on the **character set encoding used**, where it is specified in the html page: this may result in differences in rendering of special characters in display!*

" " double quotation mark
& & ampersand
° ° degree sign
< < less than
> > greater than
¬ ¬ not sign
± ± plus-or-minus sign
¹ ¹ superscript one (etc)
  nonbreaking space
 - soft hyphen
´ ´ acute accent
¢ ¢ cent sign
£ £ pound sign

¸ ¸ cedilla
¨ ¨ umlaut/dieresis
¦ ¦ broken (vertical) bar
¶ ¶ pilcrow (paragraph sign)
§ § section sign
¯ ¯ macron
» » angle quote mark, right
« « angle quote mark, left
½ ½ fraction (etc)
© © copyright sign
® ® registered sign
× × multiply sign
÷ ÷ division sign

Á: A vowel with an acute accent, &[vowel]acute;, e.g. Á
â: A vowel with a circumflex accent, &[vowel]circ;, e.g. â
è: A vowel with a grave accent, &[vowel]grave;, e.g. è
ÿ: A vowel with a dieresis or umlaut mark over it, &[vowel]uml;, e.g. ÿ
å: "a" with a ring, å
ç: "c" with cedilla, ç
ø: "o" with a slash through it, ø
ñ: A letter with a tilde over it, &[a or n or o]tilde;, e.g. ñ
æ: ae diphthong (ligature), æ