

Semantic HTML

© Ben Hunt, Scratchmedia, 2008

Contents

- Introduction to semantic HTML
- Why semantically correct HTML is better (ease of use, accessibility, SEO and repurposing)
- Comprehensive list of HTML tags, which to avoid and which ones to use, each with their semantically appropriate uses
- Tips for writing better semantic HTML, including a neat DHTML trick for automatically wrapping HTML tags in new tags
- Worked examples

What is Semantic HTML?

Semantics is the study of meaning: how meaning is created and applied to signs. “Why does X mean X?” is a question of semantics.

HTML is the markup language that we use to write web pages. It's understood by standard web browsers, as well as dozens of other types of “user agents”, including mobile phones, search engine spiders, aural browsers etc.)

HTML consists of two types of things:

- Tags
- Text content

A few tags can be content of their own (like images, Flash movies, or metadata), but most HTML tags are used to apply structure to content.

Semantic HTML, or “semantically-correct HTML”, is HTML where the tags used to structure content are selected and applied appropriately to the meaning of the content.

So, if you're wanting your HTML to be semantically-correct...

A `<p></p>` paragraph tag pair should **only** be used to indicate a paragraph (which is a structural concept). It should never be used to apply space to a web page. Never, **ever**, use a series of `<p>` tags to create space!

The HTML tags `` (for bold), and `<i></i>` (for italic) should never be used, because they're to do with formatting, not with the meaning or structure of the content. Instead, use the replacements `` and `` (meaning emphasis), which by default will turn text bold and italic (but don't have to do so in all browsers), while adding meaning to the structure of the content.

Always separate style from content

HTML tags should never be used to apply presentation – that's the job of CSS (Cascading Style Sheets). See <http://webdesignfromscratch.com/how-html-css-js-work-together.cfm> to learn more about how HTML, CSS and JavaScript fit together in web pages. (Note, perfect production practice also removes all JavaScript functions and event handlers from the markup as well!)

Why semantically correct HTML is better

Writing semantic HTML brings a wide range of benefits:

- Ease of use
- Accessibility
- Search Engine Optimisation
- Repurposing

Ease of use

First of all, semantic HTML is clean HTML. It's much easier to read and edit markup that's not littered with extra tags and inline styling. Clean markup also saves time and money when other people have to interact with it – say, a web developer who has to implement your page template in a content management system or any other web application.

A corollary benefit is that your HTML files are also smaller, so they load quicker.

Accessibility

Unless you've had to interact with HTML markup through media other than your web browser, it doesn't seem obvious to imagine that your web pages have a life outside the browser window – but they very often do. Web pages can be consumed by humans and machines in lots of different ways!

When you separate visual aspects (i.e. style) from the actual meaning of a document, you end up with a document that always means the same thing. The way it's presented or consumed can vary. One common technique web designers use is to apply different style sheets for different media. For example, you can apply a certain stylesheet only when a document is printed to paper, another one when it's viewed on screen, and yet another when it's accessed by a text-to-speech aural browser.

A text-to-speech reader also understands the tags `` or `` but it treats text output with those tags very differently to the way a visual browser responds. The TTS reader adjusts vocal tone or volume, rather than contrast or text style, which conveys the same meaning but through a different medium.

Search Engine Optimisation

Search engine spiders and crawlers, like Googlebot, represent another genus of user agents. They also consume web page content, in an attempt to discern the meaning within.

When a crawler finds a web page, it stores its assessment of what the page is *about* on an indexed database to use when matching people's search queries. The big question is – how do search engines match search terms to known pages to create a prioritised list?

Of course, they all do it a bit differently, but one of the keys to Search Engine Optimisation is to use plain old common sense. If you were a search engine, how would you do it? If you work through the problems a search engine faces, a few things soon become clear, often easily expressed prefixed with “all other things being equal...”.

Let's say you have two web pages, each with exactly the same text content (10 kilobytes).

One of the pages has an additional 5KB of HTML markup, neatly annotating the semantic meaning in the content.

The second page has 30KB of additional markup, with inline styles, lots of nested `<div>` tags, and decorative imagery.

Now, the more graphically intense page might look better to human visitors (might!), but if each page contains the search term “bluebottle” 5 times, which would you (pretending to be a search engine) judge was most relevant to someone searching for “bluebottle”?

Clearly, it's the first, more lightweight page, for a few possible reasons:

1. The **keyword density** of the lightweight page is greater. It features the search term five times in 15KB of markup, whereas the second page features it five times in 40KB of markup. Whatever the additional markup is for (the search engine might not be able to tell), it doesn't seem to be about “bluebottle”.
2. Each occurrence of the search term is likely to be **higher up** towards the start of the document in the lightweight page than it is in the 40KB page. All other things being equal, the earlier you find a search term within a document, it's more likely that the document is *about* that term, or the term is more prominent in the document's content.
3. Assuming that the first document is neatly marked up with semantically correct HTML, it's more likely that the search term will be placed inside a **higher-value tag** (such as a heading, or link) than in a more graphical page (which might use an image as a link, perhaps without a proper `alt` attribute).

Repurposing

When your markup (content, with meaning) is separated from your styles (style sheets for different media), obviously the content can be understood more easily by **all** user agents. That means not only user agents you already know about, but ones you don't yet know about (like automated crawlers that create custom RSS news feeds on a certain topic, or image- or video-specific search engines), as well as others that have not yet been invented!

The last couple of years have seen mixing and mashing content emerge as a major feature of new web sites and applications. This can happen without the knowledge of the original site owner, but in most cases this freedom of content to move around the web, adapting to various media, is beneficial to the original creator.

Often in these situations, the content taken from a web page is formatted differently on the new remixed page, which makes it all the more important to remove any style content from the markup itself. (Note that inline styles, applied directly within HTML tags, override any other styles implemented through separate stylesheets, and so they would have to be stripped off programatically.)

Clearly, it's easier to grab and re-use content from any source, and apply it to any medium, when it does not contain any hard-coded style information, and also when it does contain semantic markup that can help a computer program understand the meaning and structure of the content.

Comprehensive list of HTML tags with their semantically appropriate uses

I haven't listed absolutely every HTML tag ever, as some of them are too obscure to be worth mentioning.

Tag	What it is	When to use it
<code><a></code>	Anchor (most commonly a link)	Vital. Use to create links in content. Use the title attribute whenever the contents of the <code><a>...</code> pair do not accurately describe what you'll get from selecting the link. Title attribute often displays as a tooltip in visual browsers, which may be a helpful usability aid.
<code><abbr></code>	Defines an abbreviation	Works in a similar way to <code><dfn></code> and <code><acronym></code> , using a title attribute (displays a tooltip in standard visual browsers). e.g. <code><abbr title="Hypertext markup language">HTML</abbr></code>
<code><acronym></code>	Defines an acronym	Works in a similar way to <code><abbr></code> and <code><dfn></code> , using a title attribute (displays a tooltip in standard visual browsers).
<code><address></code>	Used for marking up a physical (e.g. mailing) address	Not commonly used. Recommend looking into microformats, which allow for more detail and interoperability.
<code><applet></code>	Inserts a Java applet	The old way to insert a Java app. Use <code><object></code> instead today.
<code><area></code>	Hotspot in image map	Avoid image maps where possible. Occasionally necessary.
<code><base></code>	Specifies the base location of the document.	Use only when necessary. Adjusts any relative links and paths within the document.
<code><basefont></code>	Sets default font size	Display info – never use it
<code><big></code>	Larger text	Display info – never use it
<code><blink></code>	Makes text blink	You go to hell if you use this
<code><blockquote></code>	Large quoted block of text	Use for any quoted text that constitutes one or more paragraphs (note: should contain <code><p></code> tags as well). Use <code><q></code> for quotations within a paragraph. Often used in conjunction with <code><cite></code> to cite the quotation's source.
<code><body></code>	Document body	Essential (unless you're using frames)
<code>
</code>	Line break	This is arguably display information. Still in common use, but use with restraint.
<code></code>	Bold text	Display info – never use it
<code><button></code>	Used for a standard clickable button within a form	Often better than <code><input type="button" /></code> or <code><input type="submit" /></code> , as it allows you to assign different styles based on the HTML element alone, whereas differentiating style based on the type of input is less well supported.
<code><caption></code>	Caption for a table: describes the table's contents	The correct way to assign a title to a table

<code><center></code>	Centred block	Display info – never use it. Use <code><div></code> or some other block-level tag with the style <code>text-align:center</code> instead
<code><cite></code>	Defines a citation	Defines the source of a quotation (in conjunction with content in <code><q></code> or <code><blockquote></code> pairs).
<code><code></code>	Defines an extract of code	Not commonly used. Similar to <code><pre></code> tag, but collapses consecutive white spaces and line breaks in the source.
<code><col></code>	Identifies a particular column in a table	Can be very useful. e.g. <code><col class="namecol"></code> can be applied to each first column in a series of tables, then the width of each column may be set to be equal in the stylesheet, overriding the table's natural tendency to adjust its own column widths to fit its contents.
<code><dfn></code>	Definition of a term	Works in a similar way to <code><abbr></code> and <code><acronym></code> , using a <code>title</code> attribute (displays a tooltip in standard visual browsers).
<code><dir></code>	Directory list	Now deprecated. Use a standard <code></code> or other list instead.
<code><div></code>	Division	Specifies a logical division within a document. Use it to separate or identify chunks of content that are not otherwise distinguished naturally using other tags. One of the most common HTML tags.
<code><dl></code>	Definition list	Contains one or more definition-term / definition-description pairs.
<code><dt></code>	Definition term	Used as part of a <code><dt></dt><dd></dd></code> pair within a definition list (<code><dl></dl></code>)
<code><dd></code>	Definition description	
<code></code>	Emphasis	Commonly used in place of the old <code><i></code> (italics) tag to indicate emphasis (but less than <code></code>)
<code></code>	Font settings	Display info – never use it
<code><form></code>	Input form	Essential for data input
<code><h1></code>	Level 1 header	Aim to have one H1 on each page, containing a description of what the page is about.
<code><h2></code>	Level 2 header	Defines a section of the page
<code><h3></code>	Level 3 header	Defines a sub-section of the page (should always follow an H2 in the logical hierarchy)
<code><h4></code>	Level 4 header	Etc. Less commonly used
<code><h5></code>	Level 5 header	Less commonly used. Only complex academic documents will break down to this level of detail.
<code><h6></code>	Level 6 header	Less commonly used
<code><head></code>	Document head	Essential. Contains information about a page that does not constitute content to be communicated as part of the page.
<code><hr></code>	Horizontal rule	Display info with no semantic value – never use it. "Horizontal", by definition, is a visual attribute.
<code><html></code>		Core element of every web page.
<code></code>	Show an image	Vital. Always use the <code>alt</code> or <code>longdesc</code> attributes when the image has content value

<code><input></code>	Input fields within forms	Vital. (I prefer to use <code><button></code> for buttons and submit buttons though)
<code><isindex></code>	Old type of search input	Not really used any more. Use <code><form></code> instead.
<code><i></code>	Italicised text	Display info – never use it
<code><kbd></code>	Keyboard input	Display info – never use it
<code><link></code>	Defines a relationship to another document	Commonly used to reference external stylesheets, but has other minor uses
<code></code>	List item	Specifies an item in an unordered or ordered list (<code></code> or <code></code>)
<code><map></code>	Client-side imagemap	May have occasional value, but only use when absolutely necessary
<code><marquee></code>	Makes text scroll across the screen	See <code><blink></code>
<code><menu></code>	Menu item list	Deprecated. Do not use. Use other standard list types instead.
<code><meta></code>	Meta-information	Useful way to insert relevant information into the <code><head></code> section of the page that does not need to be displayed.
<code></code>	Ordered list	Type of list where the order of elements has some meaning. Generally rendered with item numbers (best managed with CSS).
<code><option></code>	Selection list option	Vital for options within a drop-down control.
<code><param></code>	Parameter for Java applet	Used in conjunction with an <code><object></code> or <code><applet></code> tag to pass additional setting information at runtime.
<code><pre></code>	Preformatted text	Renders text in a pre-formatted style, preserving line breaks and all spaces present in the source. May be useful. <i>(This one's a paradox, as it is strictly display info that applies only to visual browsing, but it's still so commonly used and useful that I'm hesitant to advise against using it.)</i>
<code><p></code>	Paragraph	Only use to denote a paragraph of text. Never use for spacing alone.
<code><q></code>	Short quotation	Use for inline quotations (whereas <code><blockquote></code> should be used for quotations of a paragraph or more). Often used in conjunction with <code><cite></code> to cite the quotation's source.
<code><samp></code>	Denotes sample output text	Similar to the <code><code></code> tag. Rarely used. Avoid.
<code><script></code>	Inline script (e.g. JavaScript)	It's better to have all scripts as separate files than to write inline or in the <code><head></code> section, however still has its uses.
<code><select></code>	Selection list	A drop-down selector for a form.
<code><small></code>	Smaller text	Display info – never use it
<code></code>	An inline span within text	Use to apply meaning (and style) to a span of text that goes with the flow of content (whereas a <code><div></code> tag is block-level and breaks the flow)
<code><strikeout></code>		Display info – never use it
<code></code>	Strong emphasis	Use this instead of the old <code></code> tag.

<code><style></code>	CSS style settings	Normally used in <code><head></code> section of a page. Try to use external stylesheets, to enable you to apply different styles for different output media.
<code><sub></code>	Subscript text	Arguably display info – recommend using alternative tags (e.g. <code><cite></code>). May be required in some academic uses, e.g. Chemical formulas.
<code><sup></code>	Superscript text	
<code><table></code>	Table	Use for repeated data that has a naturally tabular form. Never use for layout purposes.
<code><td></code>	Table data cell	A cell containing actual data. If a cell actually contains a descriptor or identifier for a row or column, use a <code><th></code> (table header) tag, not a <code><td></code> . This usually applies to column headers (within a <code><thead></code>), column footers (within a <code><tfoot></code>), as well as row headers (usually the first cell in a row in the <code><tbody></code>).
<code><textarea></code>	Multi-line text input area in a form	Essential
<code><th></code>	Table column or row header cell	May appear in a <code><thead></code> (to denote a column header cell), <code><tbody></code> (to denote a row header), and in <code><tfoot></code> (to denote a column foot cell, e.g. a total)
<code><tbody></code>	Indicates the main body of a data table	It is always worth using this tag, as well as using <code><thead></code> and <code><tfoot></code> where appropriate. Note that it is permissible to have more than one <code><tbody></code> , <code><thead></code> , and <code><tfoot></code> in the same table.
<code><thead></code>	The head section of a table	The place to put column header cells (<code><th></code>)
<code><tfoot></code>	The foot section of a table	Good place to put e.g. summary data, such as totals. Note that it goes before the <code><tbody></code> tag!
<code><title></code>	Document title	Essential
<code><tr></code>	Table row	Essential with tables
<code><tt></code>	“Teletype” - simulates typewriter output	Similar to <code><pre></code> , except that it collapses white space like normal HTML (whereas <code><pre></code> leaves all consecutive white space intact). Avoid if possible
<code></code>	Unordered list	Essential. Use for lists where the order or items has no particular importance.
<code><u></code>	Underline text	Display info – never use it
<code><var></code>	Variable in computer code	Obscure tag, may only be useful in academic documents. Avoid.

How to write better semantic HTML

Here are my tips to producing better HTML markup.

- HTML first, then CSS!
- Perfect semantic correctness isn't necessarily best!
- Consider other applications of your document (other media, plus DOM manipulation).
- Semantics applies to IDs and Classnames, as well as tags!
- When to use an ID, when to use a Class.
- Consider using DHTML to insert tags for complex design.

HTML first, then CSS!

The single most useful suggestion I have for writing better HTML is to do it first, before you start applying any styles, or even thinking about the styles.

This is quite a challenge, I know, but this method really makes you think about what you write in your HTML tags! Basically, you're forcing yourself to compose HTML based on the content alone, separately from the problems of CSS production ("How do I achieve that effect?"), and only when the HTML is written do you start to address the page styling.

You can't always implement every design with minimal, basic HTML, but it's a really worthwhile goal. (There are other techniques I'll demonstrate that can help you implement complex styles that would normally require additional nested HTML elements.)

One good way to know when your HTML markup is right is to show it to another designer or developer, and ask them to read it out loud, explaining what each piece of content *means*.

Perfect semantic correctness isn't necessarily best!

This may seem to go against the point of the book, but it's worth saying. The fact is that, as with anything in web page production, there's never just one best way to achieve anything. And web pages are complex creations, with more considerations and dependencies than the stuff that's markup-related.

The semantically perfect HTML page would have the absolute minimum number of tags, with the minimum of description (by way of IDs and Classnames) required to communicate the meaning. But the absolute minimum may not always also be useful, so some pragmatism is also required.

The fact is that sometimes you do need to put in one or two extra tags that may not be required to assign meaning, but simply make your life as a CSS producer so much easier, so it's worth the trade-off.

You may also need to insert IDs or classes to facilitate DHTML coding, or for the benefit of a middleware developer or 3rd-party system.

At the end of the day, there's no actual agreed standard for semantic markup.

It's your page, so do what you feel is best.

Consider other applications of your document (other media, plus DOM manipulation)

Always bear in mind that your web page won't necessarily only ever be a web page. When you publish content online today, it becomes part of a general mass of information, which may be consumed using all kinds of browsers and other devices, by humans as well as programs.

Another way to interact with an HTML document is by querying and manipulating the DOM (Document Object Model) using DHTML or other methods. We'll see an example of DHTML manipulation later. Pure, semantic HTML makes all programmatic interaction and manipulation much easier.

Semantics applies to IDs and Classnames as well as tags!

Semantic HTML doesn't stop with tag selection.

Structural meaning is also contained within tag parameters, including `alt` parameters, IDs and Classnames. These should follow some basic common-sense rules.

In the same way that you should only use a table for structuring tabular data, and not for layout purposes, every HTML element should only have classnames that describe accurately what the element does or contains.

For example, you have a side column on the left of your main content, which contains links to selected sections of your site, plus advertisements. First instinct may be to call the column `id="leftCol"`, but is that correct?

The key question to ask is:

What is the property of this element that differentiates it from other content?

Then, try to move up the hierarchy of meaning, striving for a simpler and more generic descriptor, until you can get no more generic and simple without losing the specificity of the descriptor, and you'll have your answer.

Taking our side left column as an example:

- Obviously we need to give the left column a useful classname or ID parameter that we can use in CSS to shift it alongside the main content.
- "Left" is not an appropriate descriptor, as with CSS you (or someone else) might choose to switch the content over to the right hand side. *Left-ness* is not a core property of the content itself – it's display property, so has no place in semantic markup.
- What about "column"? Well, taking the same strict stance, a column is actually a visual organisation of content, it's a style property, not a semantic property, so really we shouldn't use that either. (Some smart CSS layouts can switch from laying content out side-by-side in columns, where width allows, to displaying the content in the additional or minor column beneath the main content in narrower displays.)
- So, it's not *left-ness*, or *side-ness*, and it's not *column-ness*, because these are all **stylistic** attributes. What property is it, then, of our left-hand column that differentiates it **semantically** from the main right-hand content? Well, there's never just one right answer, but you might find that `<div class="minor-content">` or `class="secondaryContent"` would fit the bill. Something like this would be meaningful enough to a human reader, or indeed a computer program, and would still be

flexible enough to make sense if the content were rearranged for some medium, or even if part of it were borrowed for publication elsewhere.

When to use an ID, when to use a Class

This is another interesting area. I'm finding it appropriate to use more classnames these days, where I might have used IDs in the past.

What's the difference between IDs and classnames?

Strictly, there should only ever be at most one element with any particular ID on a page. Now, this doesn't matter to CSS. You could have several `<p id="callout">` on the same page, and the CSS would work just fine.

But IDs aren't only to do with CSS! They're actually much more important and useful in the world of Dynamic HTML (DHTML), where you can programatically manipulate the document using the DOM (Document Object Model).

In DHTML, you can grab any element using the code:

```
var someParagraph = document.getElementById("para1") ;
```

If you use any ID more than once, `getElementById` can't work.

What about classnames? There are a couple of obvious differences first-up.

- You can re-use the same classnames several times within the same document (page)
- An element can have multiple classnames (separated with spaces, e.g. `<ul class="nav special banana">`)

That's all well and good, but what other differences are there in the context of semantic HTML?

Basically, use a **Class** to describe a **property** of an element (if not already implicit in its tag type). Use an **ID** to identify the unique element **itself**. This is the element's core, unchangeable essence.

So, going back to our side column example from earlier. We settled on the "*minor content-ness*" of the element as being the most generic-yet-useful differentiating feature. Now, is this the core essence of the thing, or is it a property?

I guess that it's theoretically possible to stick in a second side column (which might turn into some other kind of formatting device under different circumstances), so the "*minor_content*" should arguably be a *Classname* of the element, not its *ID* (which is what I used in the example anyway). But it's common to find the ID parameter used for things that are actually circumstantial properties rather than core essence (most of the sites I've ever coded probably do it!), so look out for that in your own semantic HTML code.

Consider using DHTML to insert tags for complex design

Let's say your design has a number of boxes with 4 rounded corners. The boxes can be any height, depending on how much content is in them, and also any width, because the layout is zoom (i.e. the master width is defined using **em** units rather than pixels or percents, so it goes wider or narrower if the browser base font size is changed).

Normally, you'd use the CSS **background** or **background-image** property to achieve this, and you'd need to wrap your content in 4 different elements, as each element can only have one background image. You'd end up with something like this in your HTML:

```
<div class="boxout topleft">
  <div class="bottomleft">
    <div class="bottomright">
      <div class="topright">
        Content goes here...
      </div>
    </div>
  </div>
</div>
```

The problem, of course, is that you're using 4 nested HTML elements to create what is essentially one thing, because of the limitations of CSS. This is not semantically correct HTML!

The problem is that, while the page might work fine when viewed in a regular web browser, there's a lot of code noise in there that other user agents have to ignore or strip out. Plus, it's increasing file size and makes your files more clunky etc. etc. If you have 10 such boxes on a page, it's 10x the clunkiness!

So, the designer insists that 4 rounded corners are essential to the design, but that's only the case in a web browser, not in a phone, RSS, or aural user interface. How do we satisfy the requirement to implement the web design as specified, while fulfilling our urge to be a pure shining semantic production angel?

The solution I'd suggest is to use correct minimal markup to start with, and then to manipulate the document programmatically using JavaScript (DHTML), essentially wrapping the contents in 4 additional divs. This code would be kept in a separate JavaScript file, run when the page loads, and would only apply to web browsers. Other user agents could happily ignore it, and they'd just get the minimal, clean, semantic markup.

Here's the markup I'd *like* to use:

```
<div class="boxout">
  Content here...
</div>
```

Now we just need a JS function that finds any elements with the classname "boxout", and wraps its contents inside 4 extra divs (could do it with 3, but let's use 4 for simplicity).

Here's my starting HTML:

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN" "http://www.w3.org/TR/html4/strict.dtd">
<html>
  <head>
    <title>DHTML Wrapper example</title>
    <script type="text/javascript" language="javascript" src="../core.js"></script>
    <script type="text/javascript" language="javascript" src="dhtml_wrapper.js"></script>
  </head>
  <body>
    <div class="boxout">
      Content goes here...
    </div>
  </body>
</html>
```

I'm calling in 2 JavaScript files:

- **core.js** (which contains a bunch of handy DHTML/JS functions), then
- **dhtml_wrapper.js** (This one will depend on some of the functions in core.js, so core must come first. Just including this file should cause the wrapping to happen.)

Here's what I want it effectively to become (just showing the body contents; the new bits are in bold):

```
<div class="boxout">
  <div class="topleft"><div class="topright"><div class="bottomright"><div class="bottomleft">
    Content goes here...
  </div></div></div></div>
</div>
```

Here's the JavaScript code for `dhtml_wrapper.js`. I won't explain exactly what it does in this book – it'll be part of my next DHTML guide.

To use it, you just need to add a `wrapelements()` function all inside the `setup()` function.

The line below marked with the `// *****` comment indicates how to call the function. It says, "Wrap any elements of type "div" that have the class "boxout" with some more elements of type "div". Then the last parameter gives a list of new div classnames to use. I've put a comma-separated list of 4 classnames here, so it'll create 4 new divs, nested inside each other, with the target divs (i.e. the "boxout" divs) in the centre.

Note that the classnames at the beginning of the list will be the central elements, with the rest of the list wrapped around the outside.

(You can find the source files at <http://webdesignfromscratch.com/ebooks/semantic-html/dhtml-wrapper/>, if you'd like to have a play yourself.)

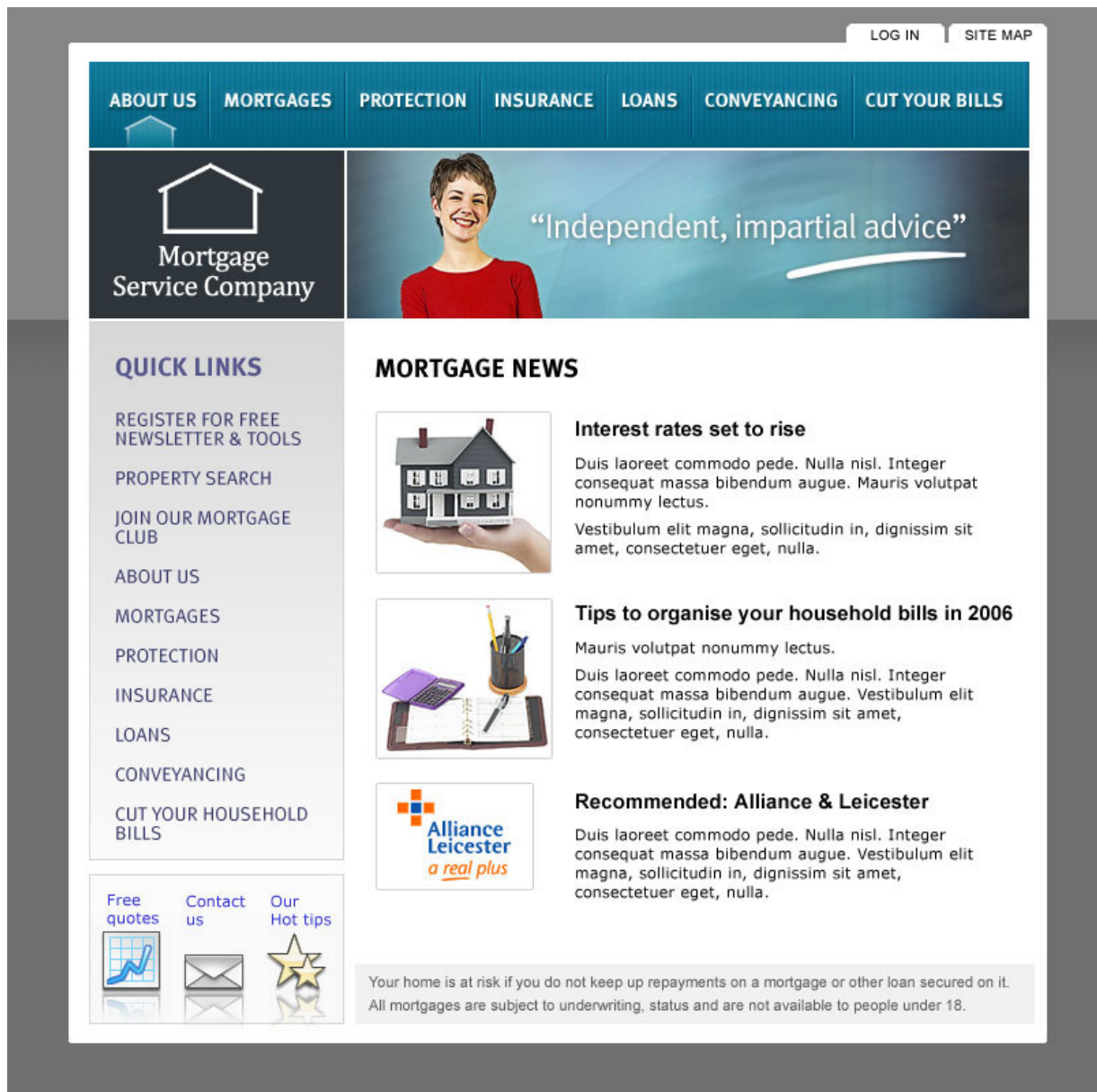
```
addEvent(window, "load", setup, false);

function setup() {
    wrapelements("div", "boxout", "div", "topleft,topright,bottomright,bottomleft") ; // *****
}

function wrapelements(targetElType, targetElClass, newElType, newElClasses) {
    var newElClassList = newElClasses.split(",") ;
    // Look for any div elements with className
    var targetElements = getElementsByClassName(document, targetElType, targetElClass) ;
    // For each element found...
    for (var i=0; i<targetElements.length; i++) {
        // Get the target element
        var elementToWrap = targetElements[i] ;
        // Get the wrappable element's parent element
        var targetElParent = elementToWrap.parentNode ;
        // Work out method to reattach later..
        var targetElPosition = 0 ;
        for (var sibCount=0; sibCount<targetElParent.childNodes.length; sibCount++) {
            if (targetElParent.childNodes[sibCount] === elementToWrap) {
                targetElPosition = sibCount ;
                break ;
            }
        }
        // For each className in the wrapping list
        for (var j=0; j<newElClassList.length; j++) {
            // Create the new element
            var newElement = document.createElement(newElType) ;
            // Add the class name
            newElement.className = newElClassList[j] ;
            // Put the wrappable element inside the new element
            newElement.appendChild(elementToWrap) ;
            // Use the new combo as the thing to wrap from now on
            elementToWrap = newElement ;
        }
        // Finally, put the new combo inside the parent element
        if (targetElPosition == 0) {
            targetElParent.appendChild(elementToWrap) ;
        }
        else {
            targetElParent.insertBefore(elementToWrap,
targetElParent.childNodes[targetElPosition]) ;
        }
    }
}
```

Worked example

Here's a sample page design. I'll work through and document my thought processes as I compose the HTML markup (prior to slicing & CSS coding).



First questions

My normal approach is to start working from the outside in, to get the major structure together. Then, I'll normally just go from the top of the HTML and work through to the end.

The first question a producer will always ask is: Is the design fixed-width, liquid (i.e. full-width), or zoom-width? Of course, that's a display question, so shouldn't affect HTML semantics!

You may find you have to go back and adjust HTML to be able to implement certain design features within an acceptable time & complexity, and within your CSS capabilities, but that's your pragmatic decision to make.

Fortunately, the design has a simple background, which tiles horizontally only, so I know that will apply easily to the `<body>` tag.

The next question I'll normally ask is: In what order shall I present the structure in the HTML document?

CSS allows producers some flexibility over the flow of the HTML (depending on the layout) to reorder page elements, for example, putting a right-hand main content column before a minor left-hand column in the markup. There are SEO benefits to doing stuff like this. You should try to get your content, with its keywords, as high up the HTML as possible, within reason.

With this design, I want the main navigation to come first, followed by the main content.

I think I'd prefer to put the "Quick links" side bar after the main content in the flow. The "Your home is at risk..." disclaimer can come last of all.

I'll put the two links "Log in" and "Site map" first of all.

It's helpful to put yourself in the position of a person with a serious visual impairment who's using an aural text-to-speech browser to read your page out loud. What will they hear first on every page? When will the experience get annoying? How quickly can you get them to the most useful content or links that will help them proceed as quickly as possible to what they really want?

Initial structure

Let's assume we're going for a zoom or fixed-width layout (the HTML is the same). For this, we'll need one container to set the master width (in ems for zoom, or in pixels for fixed). The body tag will probably be align-center, to get the master container to float in the middle of the window.

Here's the initial HTML structure. I've given my container div an ID "page" as that's what it is. Alternatively, I might consider an ID that is less visual-specific, like "structure"...

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<html>
  <head>
    <title>Mortgage Service Company, UK advice on mortgages and insurance</title>
    <link rel="stylesheet" type="text/css" href="styles.css" />
  </head>
  <body>
    <div id="page">
      I'll put content here...
    </div>
  </body>
</html>
```

The `<title>` tag describes the home page quite succinctly, containing some relevant key words.

Navigation



Let's stick in the 2 groups of top navigation: The primary nav, and what I'd call "secondary global nav" (i.e. navigation that's available on every page, but which doesn't describe the information architecture (sections) of the site. Here, "Log in" is a function, and "Site map" is just a single generic page.

```
<div id="page">
  <ul id="primaryNavigation">
    <li><a href="about.html">About</a></li>
    <li><a href="mortgages.html">Mortgages</a></li>
    <li><a href="mortgage-protection.html">Protection</a></li>
    <li><a href="insurance.html" title="Household insurance
deals">Insurance</a></li>
    <li><a href="loans.html" title="Personal loans">Loans</a></li>
    <li><a href="conveyancing.html">Conveyancing</a></li>
    <li><a href="cut-your-bills.html">Cut your bills</a></li>
  </ul>
  <ul id="secondaryNavigation">
    <li><a href="login.html">Log in</a></li>
    <li><a href="site-map.html">Site map</a></li>
  </ul>
</div>
```

I've used unordered lists (``) for both navigation sets. Unordered lists are semantically appropriate for any logical, sequential collection of simple elements where the order does not have any particular meaning. If the order of the elements is significant, an ordered list (``) is more appropriate; if the elements are complex, i.e. made up of several other elements, then a table or just a sequence of divs may be best.

I've used IDs for both lists, as the names describe what they *are*, not properties of the things. I've also put the primary nav first in the flow, as it's arguably more useful to most visitors. The positioning can be fixed using CSS easily enough.

Notice that the links contain text instead of images. This is the right thing to do, as the text is more broadly meaningful and useful than an image. In the CSS, I could use image-replacement techniques to hide the text and show background images for each link.

Site identity and banner



Next in the visual we have two images: the site ID/logo, and an ad banner. I have a choice here whether to put the images next in the flow, or to move them after the main page content. One consideration is whether

At the same time, I'd ask myself whether this would be repetitive for someone listening to this page. Now, if the banner ad is the same on every page, then perhaps hearing the same `alt` text time and again will be unhelpful.

Some designers put the *site identity* as the `<h1>` tag on every page, but this is wrong. The contents of your `<h1>` are very valuable for SEO, so each page should have key words in the `<h1>` that specifically relate to what the page is about, so the contents need to be different on each page.

In this case, I'll keep the site ID and banner inline, as they'll only have short titles that do help describe the page in a meaningful flow of information.

```
<img id="siteID" width="200" height="75" alt="Mortgage Service Company UK" />
<img id="bannerAd" width="600" height="75" alt="Independent, impartial advice on
mortgages, insurance ... " />
```

This will read as “Image, Mortgage Service Company UK, Image, Independent, impartial advice on mortgages, insurance...” (etcetera).

Next in the visual hierarchy comes the “Quick links” list in the left-hand column, followed by another list of links for “Free quotes” / “Contact us” / “Our hot tips”. Should this go before the main content, or after it? One issue is that many of the links are repeated in the main content, which I might raise with the designer. It might be better SEO to vary the link text.

Skip nav

What I'll decide at this point is to keep all the nav options together, as they are on the visual image, but to add a “skip navigation” link for the benefit of aural readers. This simply goes right at the front of the `<body>` and gives people an easy way to jump straight to the page content (which should start with a `<h1>`).

Here's the link.

```
<body>
<a class="notDisplayed" href="#content" title="Skip to content" accesskey="s">Skip to
content</a>
<div id="page">
```

The person using a text-to-speech program could hit the standard “s” hot key to jump straight to the start of the actual content. We'll use CSS to make the link non-visible (to people browsing visually).

It's quite common to link this type of link to a special anchor in the document, such as:

```
<a name="content"></a> <h1>Etc.</h1>
```

The anchor is invisible, as it has no contents. However... this is adding an unnecessary element, because you can also link to an element's ID using the `href="#id"` syntax. So, we'd achieve the same thing by putting all the main content in a `<div id="content">` and save one element.

Of course, if we don't need an additional `<div>` for the content (which I don't think we will here), it's possible to add the ID to the `<h1>` tag instead. So we'll insert the “skip” link, and assign the ID to the Heading 1.

More nav

Next item is the title for the “Quick links” list. There's no `<caption>` tag for use with lists (it only applies to tables), so I'd probably use a `<h2>` tag here. The list can then follow as a standard `` with text contents. We can either apply image-replacement, or even change the font and force upper case text, using CSS later.

Because the left column in my design only contains links, I think I can get away without using a special division to separate it from the main content (but I might choose to put one in anyway, in case the client wanted to add other elements to the side column at a later date!).

The `` has the ID “quickLinks”, as that's what it *is*. There will never be 2 such lists on a page. The rest is straightforward. I've left the hrefs and titles blank here. It might be worth using different title text in those links that also feature in the main nav.

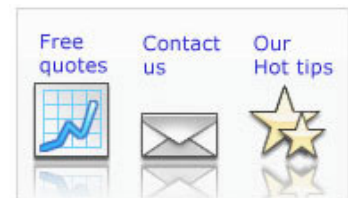
Here's the HTML:

```
<h2>Quick Links</h2>
<ul id="quickLinks">
  <li><a href="..." title="">Register for free newsletter & tools</a></li>
  <li><a href="..." title="">Property search</a></li>
  <li><a href="..." title="">Join our mortgage club</a></li>
  <li><a href="..." title="">About us</a></li>
  <li><a href="..." title="">Mortgages</a></li>
  <li><a href="..." title="">Protection</a></li>
  <li><a href="..." title="">Insurance</a></li>
  <li><a href="..." title="">Loans</a></li>
  <li><a href="..." title="">Conveyancing</a></li>
  <li><a href="..." title="">Cut your household bills</a></li>
</ul>
```

QUICK LINKS

- REGISTER FOR FREE NEWSLETTER & TOOLS
- PROPERTY SEARCH
- JOIN OUR MORTGAGE CLUB
- ABOUT US
- MORTGAGES
- PROTECTION
- INSURANCE
- LOANS
- CONVEYANCING
- CUT YOUR HOUSEHOLD BILLS

Next comes another list of links: “Free quotes, Contact us, and Our Hot tips”. These are text with images beneath them. I'd be tempted simply have these as another unordered list, but on second thoughts *semantically*, they're nothing more than additional “Quick links”, are they?




A more correct approach would be to include them in the previous ``, and include properties to let you render them differently in CSS.

```
<ul id="quickLinks">
  <li><a href="" title="">Register for free newsletter & tools</a></li>
  <li><a href="" title="">Property search</a></li>
  <li><a href="" title="">Join our mortgage club</a></li>
  <li><a href="" title="">About us</a></li>
  <li><a href="" title="">Mortgages</a></li>
  <li><a href="" title="">Protection</a></li>
  <li><a href="" title="">Insurance</a></li>
  <li><a href="" title="">Loans</a></li>
  <li><a href="" title="">Conveyancing</a></li>
  <li><a href="" title="">Cut your household bills</a></li>
  <li class="withpic" id="freeQuotesLink"><a href="" title="">Free quotes</a></li>
  <li class="withpic" id="contactUsLink"><a href="" title="">Contact us</a></li>
  <li class="withpic" id="ourHotTipsLink"><a href="" title="">Our Hot tips</a></li>
</ul>
```

The way I've approached this is to use a new class “withpic”, which relates to the layout and background properties of these 3 links. However, I've used special IDs to identify the 3 special links individually. (Some browsers like IE6 don't properly understand multiple classnames, so using a class in conjunction with an ID is easier.) Because the IDs like “freeQuotesLink” actually describe what each link *is*, this approach is fine semantically.

Main content


MORTGAGE NEWS



Interest rates set to rise

Duis laoreet commodo pede. Nulla nisl. Integer consequat massa bibendum augue. Mauris volutpat nonummy lectus.


Vestibulum elit magna, sollicitudin in, dignissim sit amet, consectetur eget, nulla.



Tips to organise your household bills in 2006

Mauris volutpat nonummy lectus.

Duis laoreet commodo pede. Nulla nisl. Integer consequat massa bibendum augue. Vestibulum elit magna, sollicitudin in, dignissim sit amet, consectetur eget, nulla.



Recommended: Alliance & Leicester

Duis laoreet commodo pede. Nulla nisl. Integer consequat massa bibendum augue. Vestibulum elit magna, sollicitudin in, dignissim sit amet, consectetur eget, nulla.

Your home is at risk if you do not keep up repayments on a mortgage or other loan secured on it. All mortgages are subject to underwriting, status and are not available to people under 18.

Next comes the heading “Mortgage News”, which should definitely be our `<h1>` tag (one per page, and describes what the page is about). We were going to add the ID “content” to the `<h1>` to link to it from the skip-nav link. In fact, I’ve changed that to `id=“startOfContent”`, which does describe an attribute of the heading (although it’s not semantically perfect, as it doesn’t describe what the heading *is*).

I’ve left the text in normal case, because the upper case can be applied in CSS (by applying the property `text-transform: uppercase;`).

```
<h1 id="startOfContent">Mortgage News</h1>
```

Next we have 3 news items, each comprising an image, a title (which should also be a link, and should certainly have a different colour to the normal text colour).

What’s the right markup structure for this series? Are they an unordered list? A table?

I don’t think an unordered (or ordered) list would be appropriate, as these are *complex* elements. If it were just a list of the 3 titles, a `` would be absolutely right. But this isn’t really a list of content elements; it’s more a series of elements that have a similar structure and format.

One option might be to use a definition list (`<dl>`). With a definition list, you have a series of definition terms (`<dt></dt>`), each followed by a definition description (`<dd></dd>`). Definition lists have been in the HTML definition since always, but have become more popular recently. They’re good when you have a series of slightly complex elements, comprising a *title* and *further description*. That does hold true in this instance: we have a title of each mortgage news item, followed by a summary or intro to the article.

To render this as a definition list, the HTML would look like

```
<dl>
  <dt><a href="">Interest rates set to rise</a></dt>
  <dd>Duis laoreet commodo pede... etc...</dd>
  <dt><a href="">Tips to organise your household bills in 2006</a></dt>
  <dd>Duis laoreet commodo pede... etc...</dd>
  <dt><a href="">Recommended: Alliance & Leicester</a></dt>
  <dd>Duis laoreet commodo pede... etc...</dd>
</dl>
```

```
</dl>
```

The wrinkle comes with the *images*. The crux question is, Are the images part of the content? i.e. Do they have any content value? Because if they're decoration only, we could perhaps use CSS to apply them as background images. If they're content, then they should be in `` tags of their own. And, if we've got a title, an image, and a couple of paragraphs, we can't do that using a definition list.

It's also worth bearing in mind that the format of these blocks need not always be set at what's shown in this initial design. If the client decided to put a totally different element in this series, it could easily break the `<dl>` format, which leads me to discount the `<dl>` as the best pragmatic solution.

What about tables? Using a table would certainly make it easy to align the contents easily.

It does seem that we have a roughly tabular data series here: a repeated pattern of title, description, image. Now, if we were looking at product search results, I can certainly imagine rendering those in a table, but in this instance, the structure isn't quite so fixed. For the same reason as mentioned above (throwing in a new format), a tabular structure wouldn't be flexible enough. Also, the content isn't visually laid out in a strict grid, which means we'd be messing around with `<td rowspan="2">` in our HTML, which is clearly too messy (messiness is always a sign of poor semantics).

That really leaves us with a series of divs! Each one would have a `<h2>` for the article title, one or more paragraphs, and an image. I would imagine that these divs should have a classname that identifies its layout type (actually, *content* type). We'll leave CSS to manage the layout of the various component elements.

For a well-formed document, headings must cascade downwards from a `<h1>` in strict tree order, never missing out a heading level, so you couldn't have a `<h3>` directly following a `<h1>` without a `<h2>` in between.

Here's the HTML I'd like to write:

```
<div class="newsItemSummary">
  <h2><a href="">Interest rates set to rise</a></h2>
  <p>Duis laoreet commodo pede etc. etc...</p>
  
</div>
```

This throws up some interesting CSS challenges, which I'm confident have a solution. (*What I'd do in my CSS is put a large 100px+ padding on the left side of the div, to make any contents move over and leave space for the image, then put the image in as position:absolute; and give it a subtle border; then finally I'd give the div a min-height to ensure that the entire image is displayed, even if there's less content.*)

So, we simply have 3 of these blocks, and below there's a disclaimer. Options for this are probably div or paragraph (`<p>`). The question is, is it a logical division within the content (i.e. a `<div>`), or is it actually a paragraph of body text? I think it's more a division than a paragraph, so I'll go with that, using the classname "disclaimer" as a handle for CSS.

Final HTML

Slightly trimmed to fit across the page, but you should find it clear and easy to read. If this markup makes good sense to you, it will make good sense to screen readers, spiders, and all other types of user agent too!

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN" "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<html>
<head>
<title>Mortgage Service Company, UK advice on mortgages and insurance</title>
<link rel="stylesheet" type="text/css" href="styles.css" />
</head>
<body>
<a class="notDisplayed" href="#content" title="Skip to content" accesskey="s">Skip to content</a>
<div id="page">
<ul id="primaryNavigation">
<li><a href="about.html">About</a></li>
<li><a href="mortgages.html">Mortgages</a></li>
<li><a href="mortgage-protection.html">Protection</a></li>
<li><a href="insurance.html" title="Household insurance, insurance deals">Insurance</a></li>
<li><a href="loans.html" title="Personal loans">Loans</a></li>
<li><a href="conveyancing.html">Conveyancing</a></li>
<li><a href="cut-your-bills.html">Cut your bills</a></li>
</ul>
<ul id="secondaryNavigation">
<li><a href="login.html">Log in</a></li>
<li><a href="site-map.html">Site map</a></li>
</ul>
<img id="siteID" width="200" height="75" alt="Mortgage Service Company UK" />
<img id="bannerAd" width="600" height="75" alt="Independent, advice on mortgages, insurance etc." />
<h2>Quick Links</h2>
<ul id="quickLinks">
<li><a href="" title="">Register for free newsletter & tools</a></li>
<li><a href="" title="">Property search</a></li>
<li><a href="" title="">Join our mortgage club</a></li>
<li><a href="" title="">About us</a></li>
<li><a href="" title="">Mortgages</a></li>
<li><a href="" title="">Protection</a></li>
<li><a href="" title="">Insurance</a></li>
<li><a href="" title="">Loans</a></li>
<li><a href="" title="">Conveyancing</a></li>
<li><a href="" title="">Cut your household bills</a></li>
<li class="withpic" id="freeQuotesLink"><a href="" title="">Free quotes</a></li>
<li class="withpic" id="contactUsLink"><a href="" title="">Contact us</a></li>
<li class="withpic" id="ourHotTipsLink"><a href="" title="">Our Hot tips</a></li>
</ul>
<h1 id="startOfContent">Mortgage News</h1>
<div class="newsItemSummary">
<h2><a href="">Interest rates set to rise</a></h2>
<p>Duis laoreet commodo pede etc. etc...</p>

</div>
<div class="newsItemSummary">
<h2><a href="">Tips to organise your household bills in 2006</a></h2>
<p>Duis laoreet commodo pede etc. etc...</p>

</div>
<div class="newsItemSummary">
<h2><a href="">Recommended: Alliance & Leicester</a></h2>
<p>Duis laoreet commodo pede etc. etc...</p>

</div>
<div class="disclaimer">Your home is at risk if you do not keep up repayments on a loan etc...</div>
</body>
</html>
```

I hope this introduction to semantic HTML has convinced you that to write your own HTML this way is both **worthwhile** and **achievable**. The discipline does present its own challenges, both on the semantics side, and testing your CSS skills, but the more you do it the easier it gets, and the better web pages you produce.

Always remember that there's never just one perfect solution. It's a problem solving game, and it should be fun!

If you have any feedback or comments on this ebook, please feel free to email me direct at ben@scratchmedia.co.uk, and I'll be happy to hear your thoughts.