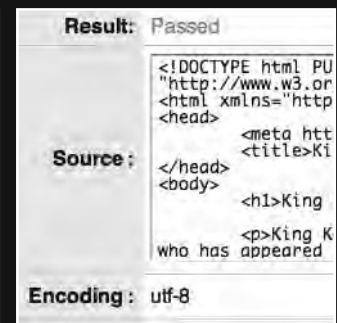
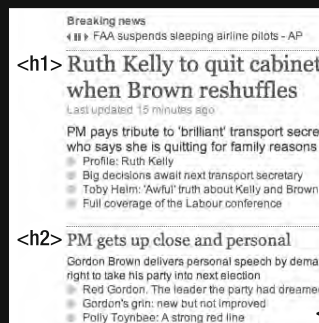
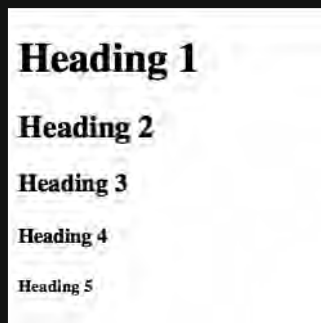


# CHAPTER 3

## STRUCTURED MARKUP



This chapter is critical to your evolution as a Web Standardista. Focused on structured markup, the concepts introduced will form a solid foundation on which you will build as we progress through the remainder of the chapters. At the heart of the chapter lies a return to fundamental principles, particularly the idea of embracing Plain Old Semantic HTML (POSH) and using it as a basis on which to build.

First, we introduce the concepts of structured markup and semantic markup and use a case study to explain the idea of using “signposts for reading” to guide the reader. Once we establish the basic principle of structured markup, we introduce you to some fundamental tags: heading tags and tags for adding emphasis. You briefly met one of the heading tags in Chapter 2; now we start to use it and its companions.

It might not seem like much, but with just the tags introduced in this chapter in your toolbox, you’ll be able to create well-structured web pages that will be the envy of your less-well-trained peers.

At this point you’re probably thinking, “Is that it? A few more tags? Where’s the design? Where’s the color? Where’s the fancy stuff?” The answer is that it’s coming, and although you might not realize it, we’re already adding design just by structuring the information on our web pages.

Good information design separates the great web sites from the ordinary ones. By the end of this chapter, you’ll be creating great web pages with well-structured markup, and that’s half the battle.

## Adding structure and meaning

**Structured markup**—sometimes referred to as **semantic markup**—is the practice of using XHTML to define the structure of a document’s content. We believe in establishing a firm foundation of structured markup *before* moving on to apply design with CSS. That’s why we’ve devoted almost half of this book to the creation of well-formed XHTML.

As we suggested previously, the process of analyzing a page’s content and applying some structure to it is *a part of the design process*. Where the beginning web designer almost always goes wrong is in hurrying too quickly into what they mistakenly believe is the “design phase,” which they equally wrongly assume is “only about CSS.”

The reality is the design phase in fact encompasses both the creation of well-structured markup in XHTML and subsequently applying style to this with CSS.

*Although sometimes used interchangeably, there are some subtle differences between structured markup and semantic markup. The term structured markup usually refers to the structure of the document: how your headings and paragraphs relate to each other, and arranging your information in a logical and meaningful order. The term semantic markup usually refers to adding meaning to your markup, using tags that say something about your content. When you add a heading or a paragraph to your page, however, you are adding both structure and meaning. In fact, structure and meaning are quite closely intertwined.*

## What is structured markup?

Structured markup lies at the heart of the Web Standardistas' approach. We believe in using the right tag for the right job. A heading should be within a heading tag; paragraphs should be within `<p>` tags; a quote should be within `<blockquote>` or `<q>` tags. It's simple: look at the content and ask yourself, "Is this a heading?" If it is, then use a heading tag; if it isn't, use a more appropriate tag.

HTML was invented to give structure and meaning to documents. The emphasis was on using tags to describe what they contained. This approach, using semantics to suggest meaning, is fundamental to the Web Standardistas' approach.

The first phase in building any web page should be a careful analysis of the information it contains and identifying its inherent structure.

Take a look at Figure 3-1 and Figure 3-2. The first has no structure at all, but the second is clearly structured.

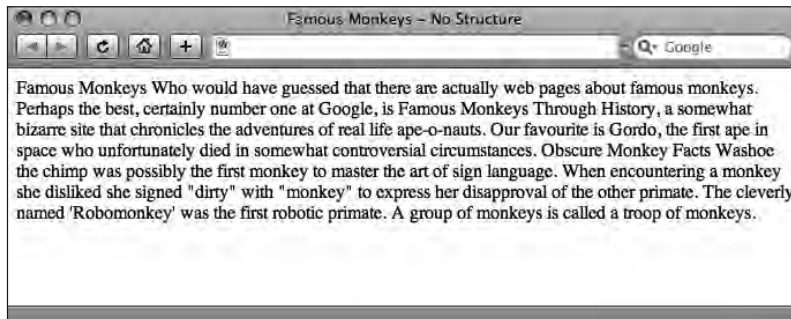


Figure 3-1. A document with no structure displayed in a browser

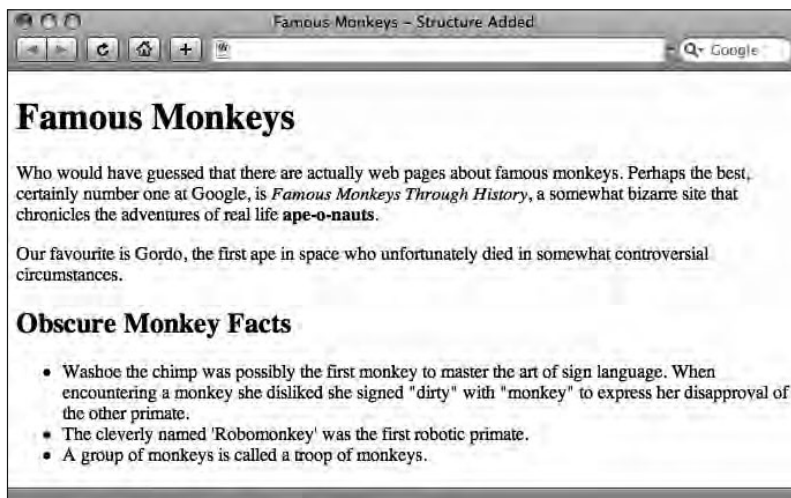


Figure 3-2. The same document with tags added—giving the page structure

Looking at these two examples, notice the second one has an implicitly defined structure and a clear information hierarchy. We would argue that *even without style added with CSS* that the document in Figure 3-2 has been “designed.” The process of adding this layer of structure or meaning is central to a structured markup approach.

## What is semantic markup?

We’ve mentioned the word *semantic* a number of times, but what exactly does the word mean? A dictionary definition gives us a start:

***semantic*** –adj. relating to meaning in language.

Concise Oxford Dictionary (Clarendon Press, 1990)

In English? The word *semantic* is derived from the Greek word for *sign*. We use semantics to give meaning to information through the creation of a logical structure. The idea of creating *signposts for reading* is one we introduce later in this chapter. Think of your markup as a series of signs that clearly inform the browser of its purpose, “I’m an `<h1>`: I’m an important heading. I’m a `<p>`: I’m a paragraph.”

Our emphasis is on semantic markup first and foremost. Is it a list? Then mark it up as a list. Is it a heading? Then mark it up as a heading. We believe code should be meaningful; it should convey some sense and some structure. We believe in using appropriate markup when it’s needed, and then styling it later.

Let’s repeat that, as it’s fundamental to the Web Standardistas’ approach: we believe in using appropriate markup when it’s needed, and then styling it later.

This is the first hurdle beginners in web design trip over. They look at how big an `<h1>` is when it renders using the browser’s default style sheet and opt instead for an `<h2>` or an `<h3>` because they’re smaller and “look nicer.” “I just couldn’t bring myself to use an `<h1>` tag—it’s just insanely big!”

This is XHTML + CSS mistake 101. Yes, minus any styling via CSS an `<h1>` tag is insanely big, but we’ll be styling it with CSS, which allows us to design it to be as large or small as we’d like. The bottom line is that, minus the CSS, the information needs to convey meaning. Switch off the users’ style sheets, and it’s clear that the `<h1>` is more important than the `<h2>` and `<p>` tags beneath it. *The tags convey meaning.* That’s the point.

## Making markup meaningful

We believe in the use of meaningful markup. Our emphasis is on semantics first and foremost. Before you even open your text editor, look at the content you’ll be working with and break it down into its component parts. Give it a hierarchy and try to tease out the meaning of specific phrases. With this knowledge, mark up the document accordingly.

This is design at its most fundamental level: looking at words and working out which tags are most appropriate. If we can convey a document's structure at this level using just XHTML, we're halfway there. The rest, handled with CSS, is just surface gloss and presentational niceness.

## POSH and proud

When we talk about being POSH and proud, we're not referring to our gentlemanly lineage, rather we're referring to POSH, which we've previously defined as an acronym for Plain Old Semantic HTML. Coined in April 2007, POSH might sound like a new invention (or an old Spice Girl), but it's not. It is, however, a great way to remember the basics of HTML and what it was first supposed to achieve.

Roger Johansson, a noted advocate of accessible web design, summarizes the appeal of the term nicely:

*POSH is short for "Plain Old Semantic HTML" and is obviously much quicker and easier to say than "valid, semantic, accessible, well-structured HTML."*

[http://www.456bereastreet.com/archive/200711/posh\\_plain\\_old\\_semantic\\_html/](http://www.456bereastreet.com/archive/200711/posh_plain_old_semantic_html/)

Rewind two decades. When Tim Berners-Lee first conceived HTML, he intended it to be a language about language, a metalanguage or a means of describing language. POSH markup returns to those original principles, using the right tag for the job, putting meaning and semantics first and adding style later. This is at the heart of the Web Standardistas' approach. Follow our guidelines, and you'll be writing POSH markup in no time; you'll also be the envy of your peers.

## Signposts for reading

Whenever we read something—a newspaper or a book, for example—our eyes are guided through the content through the use of established typographic techniques. A headline is styled differently from a paragraph, and different headlines are assigned different levels of importance through relative size and other design aspects, for example, the use of color.

By establishing a basic information hierarchy, such as using `<h1>` tags for the most important headings and using `<h6>` tags for the least important headings, we can make the reading process easier; we can also improve the accessibility of the web pages we design (making those pages accessible to, say, visually impaired users using screen readers or high contrast layouts). A welcome byproduct of this improved accessibility is that search engines find indexing your pages easier.

This section covers the relationship of tags to one another. It introduces a core concept that we've already mentioned is key to the Web Standardistas' approach: structured markup.

## Creating structure with headings and paragraphs

What's more important, an `<h1>` or an `<h2>`? Is an `<h3>` more important than an `<h5>`? How does a `<p>` relate to an `<h1>`? The answer is simple. In XHTML, there are six headings: `<h1>`, `<h2>`, `<h3>`, `<h4>`, `<h5>`, and `<h6>`. `<h1>` is the most important, and `<h6>` is the least. `<p>` tags indicate paragraphs, which generally sit under one of the heading tags, depending upon where they appear within the document's semantic structure.

In Chapter 2, you saw that you can use more than one set of `<p>` tags on a page, and heading tags are no different. We can have as many heading tags on a page as we like; the key is to define the structure and apply our markup accordingly.

The W3C states the following:

*Since some users skim through a document by navigating its headings, it is important to use them appropriately to convey document structure. Users should order heading elements properly. For example, in HTML, H2 elements should follow H1 elements, H3 elements should follow H2 elements, etc. Content developers should not 'skip' levels (e.g., H1 directly to H3). Do not use headings to create font effects; use style sheets to change font styles for example.*

<http://www.w3.org/TR/WCAG10-HTML-TECHS/#document-headers>

The following example show a series of headings marked up in XHTML:

```
<h1>Heading 1</h1>
<h2>Heading 2</h2>
<h3>Heading 3</h3>
<h4>Heading 4</h4>
<h5>Heading 5</h5>
<h6>Heading 6</h6>
```

The way the browser displays these headings by default, as shown in Figure 3-3, gives us an indication of the importance of the different headings. The `h1` is clearly more significant than the `h6`. As we mentioned earlier in this chapter, we can use CSS to adjust the style and size of these headings, so by now you know not to be tempted to pick a heading based on its default size.

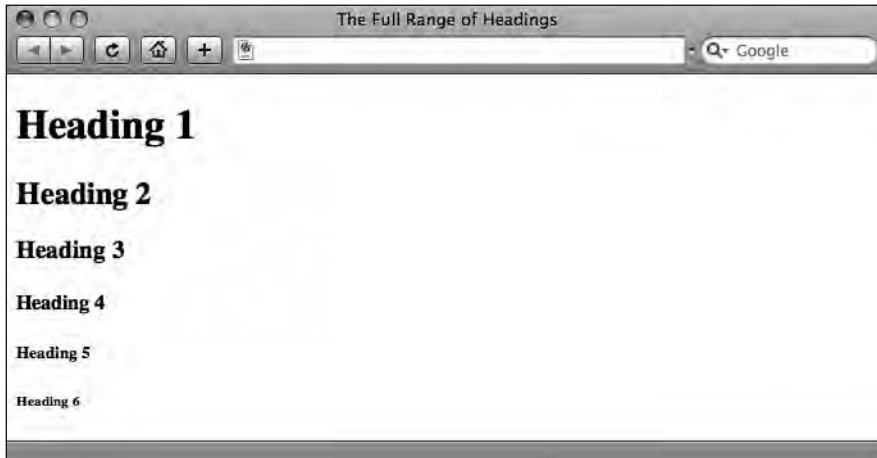


Figure 3-3. The full range of headings as they display, unstyled, within a browser

## Applying information hierarchy

When building pages using a Web Standardistas' approach, the first stage in the design process is looking at a page's content and applying some structure to it using the appropriate tags. Once this stage is completed and only then do we move on to style our well-structured markup through the use of CSS.

In Figure 3-1 and Figure 3-2, earlier in the chapter, we showed you two versions of the same page, one *with* and one *without* structure. Let's take a closer look at the text that comprises this document minus markup. By doing this, we can begin to ask some questions about the document's structure and use the answers to those questions to guide our choice of tags when marking up the document.

It might sound obvious, but the first thing we'll do is to carefully read the content. It's important to get a feel for the relationship between the different sections so that we can establish a clear information hierarchy. Our plain, unstructured content looks like this:

### Famous Monkeys

Who would have guessed that there are actually web pages about famous monkeys.

Perhaps the best, certainly number one at Google, is Famous Monkeys Through History, a somewhat bizarre site that chronicles the adventures of real life ape-o-nauts.

My favourite is Gordo, the first ape in space who unfortunately died in somewhat controversial circumstances.

### Obscure Monkey Facts

Washoe the chimp was possibly the first monkey to master the art of sign language. When encountering a monkey she disliked she signed "dirty" with "monkey" to express her disapproval of the other primate.

The cleverly named 'Robomonkey' was the first robotic primate.  
A group of monkeys is called a troop of monkeys.

After reading through the content, it is time to start marking up our headings and paragraphs. The page is all about famous monkeys, so the first line, “Famous Monkeys,” would best be marked up as an h1, the most important heading on the page. Under this heading, we have three sections that we'll mark up as paragraphs. So far, so good. Our document is slowly taking shape.

The line “Obscure Monkey Facts” is destined to become another heading. As it's a subsection and less important than the “Famous Monkeys” heading at the top of the page, we'll mark this up as an h2. Following this heading, we have three obscure monkey facts. Although the first piece of monkey trivia is a little longer than the second and third, this content is still suitable to be marked up as a list. Our final port of call is to add some additional semantic meaning to the page by pulling out some key phrases in this simple example using the <em> and <strong> tags we've already discussed.

The resulting markup looks like this:

```
<h1>Famous Monkeys</h1>
  <p>Who would have guessed that there are actually web pages about
  famous monkeys. Perhaps the best, certainly number one at Google, is
  <em>Famous Monkeys Through History</em>, a somewhat bizarre site
  that chronicles the adventures of real life
  <strong>ape-o-nauts</strong>.<p>
  <p>My favourite is Gordo, the first ape in space who unfortunately
  died in somewhat controversial circumstances.</p>
<h2>Obscure Monkey Facts</h2>
<ul>
  <li>Washoe the chimp was possibly the first monkey to master the
  art
  of sign language. When encountering a monkey she disliked she
  signed
  "dirty" with "monkey" to express her disapproval of the other
  primate.</li>
  <li>The cleverly named 'Robomonkey' was the first robotic
  primate.</li>
  <li>A group of monkeys is called a troop of monkeys.</li>
</ul>
```

The result of this process is that we've moved from an unstructured page with no information hierarchy to a well-structured XHTML page with a clear information hierarchy. You'll see as you progress through the book how you can apply style to all of the tags used here to further tease out the document's structure and meaning.

## Case study: The Guardian

Looking at an example web page and trying to identify its structure is the best way to learn what structured markup is. At this point, it's over to you. The example shown in Figure 3-4



is taken from [www.guardian.co.uk](http://www.guardian.co.uk), the web site of the UK national newspaper the *Guardian*. Looking at the example, try to work out which is the most important heading—an h1—and which are the h2 and h3 headings.



Figure 3-4. The *Guardian* home page

In our Famous Monkeys web page that we looked at in the previous section of this chapter, the most important heading, our h1, also happened to be at the very top of the page. Looking at the example in Figure 3-4, however, you can see that there's a lot of information at the top of the page that doesn't seem to fit the "most important heading" description. Sign in? Register? Search? These don't feel like headings, so we need to keep looking.

A contender for the h1 spot is the [guardian.co.uk](http://guardian.co.uk) brand, displayed in two shades of blue near the top of the page. This brand is consistently displayed throughout the site, and some might argue that this could indeed be the most important heading on the page. When you pick up a newspaper, however, you're probably more interested in the news headlines than in the paper's branding, which stays the same day in, day out, regardless of what's going on in the world.

As you can see in Figure 3-5, the first news headline on the page is in fact an h1. Following this news story are a number of other headlines, all marked up as h2s. These stories are all considered important; they're not the top story of the day, but they are *all* marked up to be the second most important pieces of information on the page. So a page may have more than one h2; ultimately this depends on the content. Further on, in the right-hand column, less urgent stories about dating and saving are marked up with h3 tags.



Figure 3-5. The Guardian home page, annotated to indicate the different levels of headings

It's worth bearing in mind when looking at this example that there may be several different ways to structure our information meaningfully. After all, what is important to you might differ from another's opinion of what should get the highest priority. However, our goal should remain the same: when writing your markup, adding your headings and paragraphs and structuring your page, strive toward adding tags that give clues as to the *meaning* of the content. Not how it should look on the page, but *what it means*.

As a general rule, there are two approaches to applying markup in this first pass: a linear approach, where an h2 is a subsection of an h1 and so on; and a less linear approach, where information is marked up in order of importance (e.g., an h2 isn't necessarily a *subsection* of the h1, but is the second most important item on the page). Which approach you take depends upon the content you're marking up, both on the page itself and throughout the site.

Bear in mind that your web pages don't exist in isolation and will have a relationship to each other. It's worth considering the information hierarchy across different pages in your site as this will also have an impact on how you mark up those pages. At the end of the day, it all depends on the content. Let the content guide your decisions over which tags to use, and you should be fine.

## An introduction to phrase elements

We briefly alluded to *elements* in Chapter 2 with a diagram (Figure 2-5) that showed an element consisting of an opening tag, some content, and a closing tag. In this section, we explain what an element is in a little more detail.

### What is an element?

You already know what a tag is, and we've briefly discussed elements already. But what exactly is an element? We could have steered clear of elements and simply referred to tags throughout; however, it's useful for Web Standardistas to have a fully functioning vocabulary (you never know which famous web standards evangelist is around the corner, and we want you to be prepared for every eventuality).

Tags, elements? It's not as complicated as it sounds. As we mentioned in Chapter 2, an element is simply a set of opening and closing tags (<p> and </p> for instance) plus the content within these tags, as indicated in Figure 3-6.

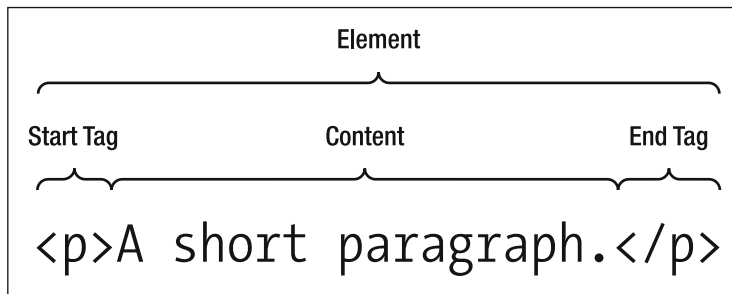


Figure 3-6. The structure of an element

A simple example of an element might be a paragraph of text, like in the following example. This would be referred to as a p element. Note the lack of the < and > brackets—we're talking about *elements* now, not *tags*. The p element in question is *everything* that follows: opening tags, closing tags, *and* content.

```
<p>And now, ladies and gentlemen, before I tell you any more,
I'm going to show you the greatest thing your eyes have ever beheld.
He was a king and a god in the world he knew, but now he comes to
civilization merely a captive - a show to gratify your curiosity.
Ladies and gentlemen, look at Kong, the Eighth Wonder of the World.
</p>.
```

What's important to remember is that elements can contain other elements. In the next example, everything contained within the opening <body> and closing </body> tags, including the tags themselves, can be referred to as the body element.

```

</head>
<body>
  <h1>King Kong - The Lost Scenes</h1>
  <p>The original version of King Kong included scenes that later
  were cut to placate the censors. One such scene was only shown
  publicly once during a preview screening in San Bernardino,
  California in 1933. This lost scene featured Kong shaking four
  sailors off a log bridge, causing them to fall into a ravine where
  they were eaten alive by giant spiders.</p>
</body>
</html>

```

## Adding meaning to fragments of text

**Phrase elements** add meaning to fragments of text. You've already encountered two phrase elements, `em` and `strong`, in our example of nesting elements, but there are many more that you can use to improve the structure and meaning of your markup.

In this section, we introduce you to a number of phrase elements to expand your XHTML vocabulary. By the end of this chapter, you'll have an extensive set of elements from which to choose.

## Adding emphasis: `<em>` and `<strong>`

Poorly trained web designers often confuse `<em>` (emphasis) with `<i>` (italics) and `<strong>` (strong emphasis) with `<b>` (bold). Given a little guidance, however, we're confident you will never make this glaring error as a Web Standardista. Why? Simply because `<i>` and `<b>` are presentational, that is, *they only affect visual display within the browser*, whereas `<em>` and `<strong>` suggest *meaning*.

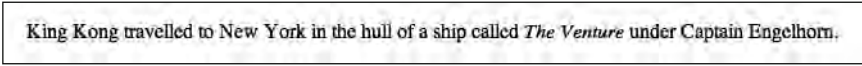
Comparing `<i>` and `<em>` should help to clarify this distinction further. `<i>` deals only with the visual display of text (i.e., it is *presentational*), whereas `<em>` conveys meaning (i.e., it is *semantic*). This meaning is interpreted by the browser to display as italic; however, it also conveys additional information to assistive devices like screen readers for the visually impaired—where display is, by definition, impossible. When using `<em>` and `<strong>`, screen readers will change volume, pitch, or rate to suggest the difference in emphasis.

There are some occasions where using italics might be appropriate, such as to indicate a ship's name, as in the following example (also shown displayed within a browser in Figure 3-7):

```

<p>King Kong travelled to New York in the hull of a ship
called <i>The Venture</i> under Captain Engelhorn.</p>

```



King Kong travelled to New York in the hull of a ship called *The Venture* under Captain Engelhorn.

**Figure 3-7.** Our italics example as displayed in a browser

Although we would like the ship's name *The Venture* to be displayed in italics in a visual browser, we don't want a screen reader to add emphasis to the ship's name when reading it aloud. This makes an `<i>` tag a more appropriate choice than an `<em>` in this case.

## Other phrase elements

3

Being restricted to only adding emphasis and strong emphasis to your content might become a little bit monotonous after a while, so it's worth getting to know a few other phrase elements. These elements can all be used to add additional meaning to your content. You might end up using some of them all the time, others might be used less frequently, but all are worth knowing about nonetheless.

- `abbr`: Used to identify the enclosed text as an abbreviation or a shortened form of a word or phrase (e.g., Dr.).
- `acronym`: Used to indicate an acronym, a word formed from the initial letters of other words (e.g., NATO).
- `cite`: Used to denote a citation, a reference to another document, especially books, magazines, and articles.
- `q`: Used to mark up short quotations. Standards-compliant browsers will add quotation marks around text marked up with `q`. Sadly, a lack of consistent browser support has made this element hard to use.
- `code`: Used to denote a sample of program code. By default, code is rendered in the browser's specified monospace font—with this type of font, every character has the same width. This makes the code easier to read and to differentiate from the rest of the page content. Other programming-related elements include `kbd` (for keyboard commands), `samp` (for code samples), and `var` (for code variables).
- `del` and `ins`: Used to indicate deleted and inserted text, useful when revising a document. Deleted text is usually displayed with a line drawn through the text, whereas inserted text usually displays with an underline.

You'll encounter these and other phrase elements again in a little more detail in the following chapter; for now, consider yourself introduced, but not yet intimate.

## Block-level and inline-level elements

There are two types of elements in XHTML: **block-level elements** and **inline-level elements**. What's the difference between them? Simply put, block-level elements generally begin rendering on a new line within the document and force a new line when they are closed. You'll have noticed by now that a browser by default inserts a line of blank space above and below an `h1` or a `p`; that is because these are block-level elements.

Inline-level elements, on the other hand, display inline. Adding `em` or `strong` elements within a paragraph does not force a line break within the paragraph because these are inline-level elements.

But surely it can’t be that easy? Wait, there’s more . . .

## Imagine a box

Every element within an XHTML document is contained within a “box” that is either block-level or inline-level (the latter is sometimes referred to as **text-level**). The easiest way to demonstrate what this means is to point out some examples in a screenshot.

In Figure 3-8, the first paragraph (contained within `<p>` tags) occupies its own block-level space. In the second paragraph, both the words *ape* (contained within `<em>` tags) and the word *monkey* (contained within `<strong>` tags) are inline-level elements and so display inline. (The paragraph that contains these inline-level elements is also block-level; however, we’ve resisted the urge to draw a box around it to keep the illustration clearer!)

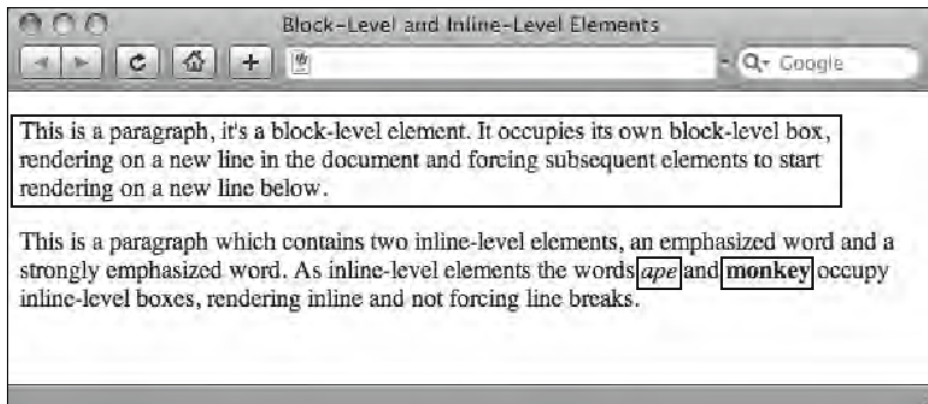


Figure 3-8. Block-level and inline-level elements

## The difference between block-level and inline-level elements

As you can see in Figure 3-8, block-level elements generate “breaks” before and after their containing box. Put simply, if we consider a paragraph, we imagine it to be a block of text with space above and below it—it is block level. A word in italics or bold, however, is contained within the paragraph—it is inline.

Some examples of block-level elements include `h1`, `h2`, and `p`.

Some examples of inline-level elements include `strong`, `em`, and `cite`.

In XHTML, block-level elements cannot be nested inside inline-level elements; for example, you cannot have a `p` element nested within `<strong></strong>` tags. In the following two examples, we show you the right way to nest elements using the First In, Last Out approach.

This example is correct:

```
<p>The strong element nested within the containing p tags in this
paragraph - <strong>me</strong> - is fine.</p>
```

The following example, however, is incorrect, as a block-level element cannot be nested within an inline element:

```
<strong>This is <p>not fine</p>.</strong>
```

It's also worth noting that inline-level elements cannot be placed directly within the body without first being enclosed within block-level elements. Failure to nest inline-level elements within block-level elements will result in pages that fail to validate.

In the next section, we will look at a way to ensure that you've nested your elements correctly and that your document is valid. We'll do so using the W3C Markup Validation Service.

## Valid code is browser-friendly markup

By now, you've been introduced to quite a bit of material. If you've been following along with the examples and exercises, and experimenting building your own pages, you should be capable of building pages with quite a bit of complexity.

As with any process, the more complexity you add, the easier it is for errors or mistakes to creep in. These errors might affect the display of your web page within different browsers across different platforms, so it's important you pick them up and correct them.

Wouldn't it be great if there were a service that offered to check all your code for you? A service that highlighted line by line where those errors lie to make your bug-hunting task just that little bit easier? Better still, a service that did all of the above *for free*. Good news, there is. Meet the W3C Markup Validation Service.

## The W3C Markup Validation Service

Why use the W3C Markup Validation Service? There are a number of reasons. First, valid pages are Google-friendly pages, and Google-friendly pages are easier to find. Second, valid pages are easier to debug.

When we build a web site, it's inevitable that things will go wrong from time to time. We get distracted, our minds wander, a mistake creeps in. The W3C Markup Validation Service highlights where the errors are, details what those errors are, and points them out line by line, ensuring that they're easy to track down and fix.

When your web page displays in an unexpected way, a brief check to see whether your code is invalid or formatted incorrectly can often highlight the problem. Using the W3C Markup Validation Service can save a lot of checking over code line by line by highlighting where any mistakes are. If you've accidentally forgotten to close a tag, for instance, using the validator will show you where you went wrong, saving you hours of looking through your code trying to find the bug yourself.

Use of the validation service is easy, and you don't even need to have your files uploaded to a server to avail yourself of the service (we'll be covering uploading your files in Chapter 7).

Let's give it a test drive. We start off with opening the following page in our browser:

```
http://validator.w3.org/
```

Clicking the `Validate by Direct Input` tab allows you to copy and paste your markup into the validator. Let's put the markup in the following example through the test:

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
  <meta http-equiv="Content-Type" content="text/html; charset=UTF-8" />
  <title>King Kong</title>
</head>
<body>
  <h1>King Kong</h1>
  <p>King Kong is the name of a fictional giant ape from the legendary
Skull Island, who has appeared in several works since 1933. Most of
these bear his name, and include the groundbreaking 1933 film, the
film remakes of 1976 and 2005, and numerous sequels.</p>
  <p>Although the 1933 movie featured crude animatronics and a giant
ape made out of a sponge, it is considered by many to be the
definitive version.</p>
</body>
</html>
```

As shown in Figure 3-9, we've pasted the preceding markup into the validator. Now it's time for the moment of truth. Click the `Check` button, and the validator will check our page.



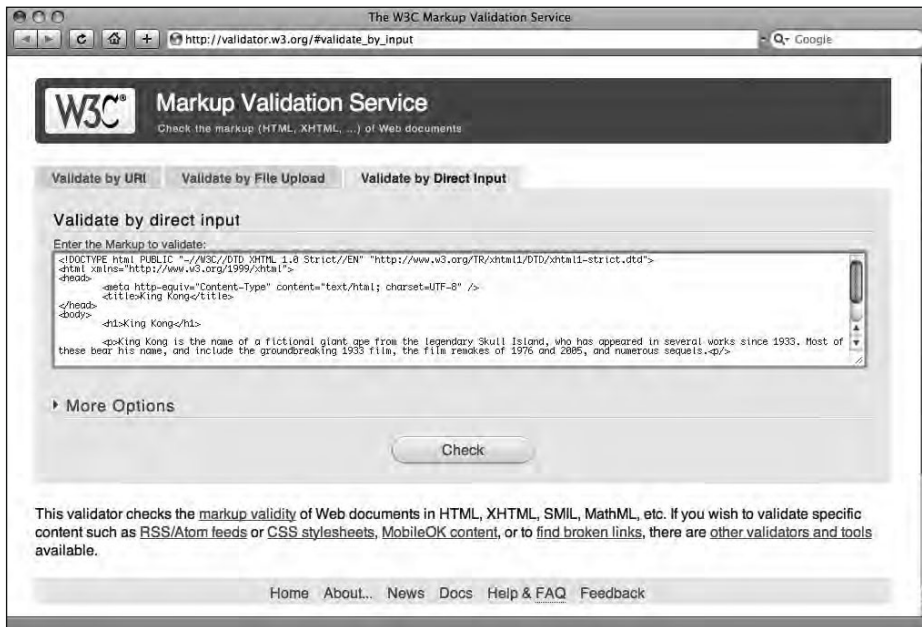


Figure 3-9. We're putting our markup through the test with the W3C Markup Validation Service.

Checking our markup reveals that there's something not quite right with our XHTML. As shown in Figure 3-10, there are a total of six errors, but how do we find out what those errors are so that we can fix them?

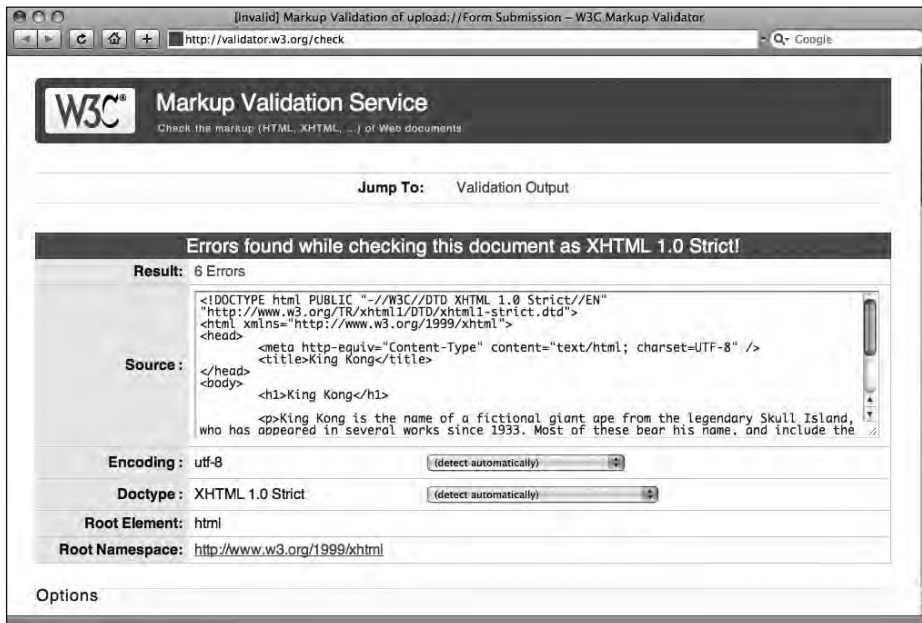


Figure 3-10. The W3C Markup Validation Service shows that our page is not valid. Let's find out why.

This is where the **validation output** comes into play. Scroll down the page, and you will see each error listed, with some helpful, if at first rather perplexing, details. Figure 3-11 shows the first of our six errors. Let's take a closer look at it and try to work out what's gone wrong.

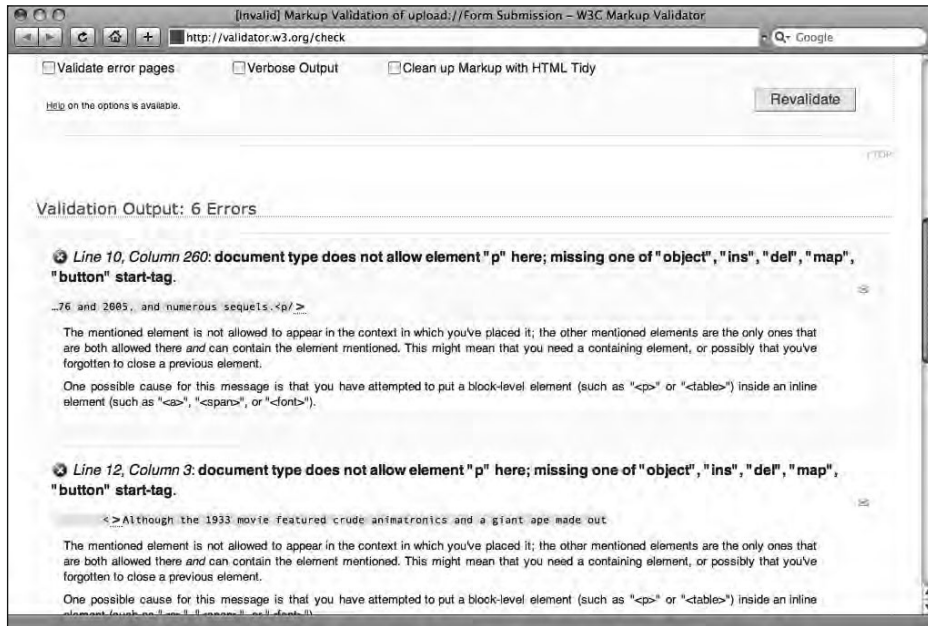


Figure 3-11. The validation output shows the details of our errors.

Let's start from the beginning. The first bit of information, displayed in *italics*, is telling us where in the document our error has occurred—in this case, on Line 10 of our XHTML document. Most plain text editors have a feature like TextWrangler's Show Line Numbers. If your text editor supports this feature, it's a good idea to turn it on, as this will allow you to quickly identify where to locate any errors, as shown in Figure 3-12.

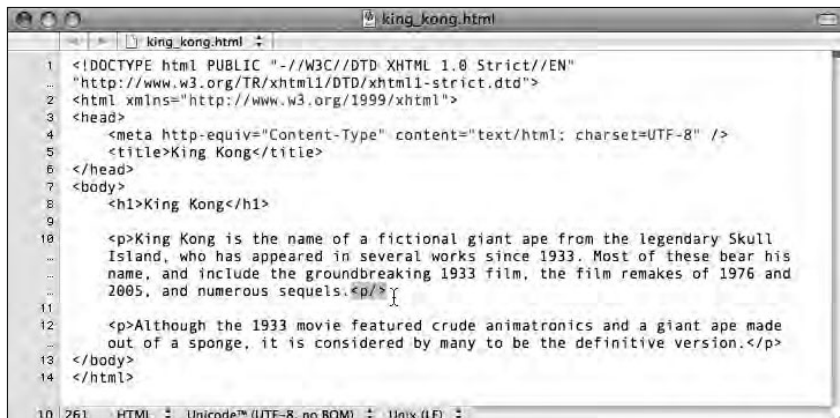


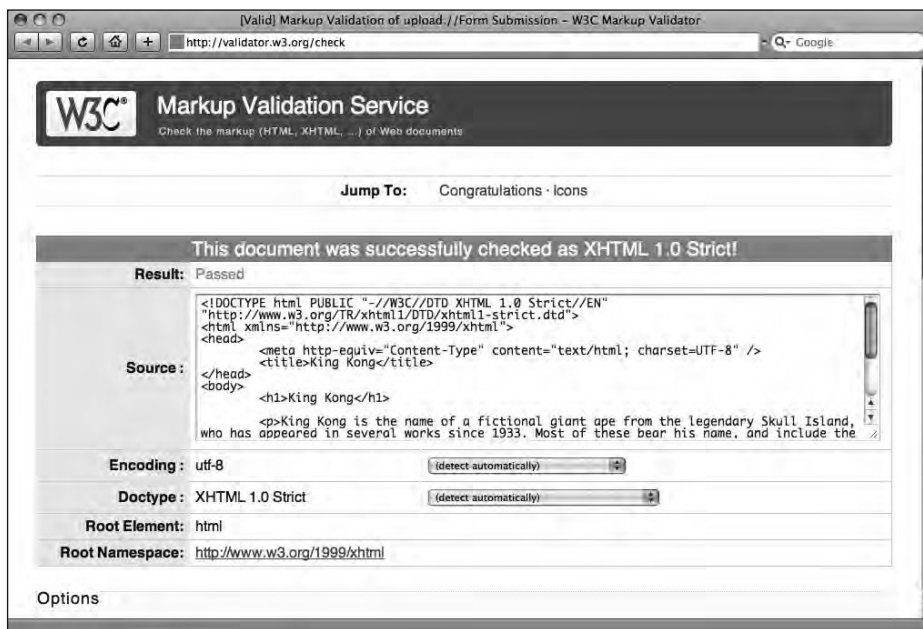
Figure 3-12. Line numbers help you find specific sections of code easily.

The second piece of information, displayed in bold, is probably a little bewildering, as it describes the probable details of your error in very dry, technical terms. Along with the more verbose paragraphs below, expanding on these details, this information is an attempt by the validator to determine the cause of the error for you. In this case, as there could be a number of possible explanations, you're left with the task of deciphering the information and working out which error applies to you.

In simple scenarios, the line of code displaying the markup where your error occurred is the best place to start. In our example in Figure 3-12 earlier, the end of Line 10 looks like this:

```
...film remakes of 1976 and 2005, and numerous sequels.<p/>
```

As you might already have spotted, the error is a simple typo. We've tried to close our paragraph tag, but accidentally written `<p/>` instead of `</p>`. Let's fix this typo and revalidate by clicking the Revalidate button. Figure 3-13 shows the result after fixing the error.



**Figure 3-13.** Fixing one error made all six disappear. Now our page validates.

Fixing just this one error clearly had a domino effect; the result is that all six errors have disappeared. It's worth revalidating your markup after fixing each error, as one simple mistake can often result in multiple errors being reported by the validator.

One of the reasons to embrace the W3C Markup Validation Service early in your career as a budding Web Standardista is that it can teach you a great deal about how to debug code and resolve web page display issues. The act of debugging a page and trying to get it to validate is extremely educational and, as a welcome byproduct, highlights the importance

of well-formed code. The W3C Markup Validation Service is a free tool; it costs nothing but time to use.

If we ensure our pages validate without errors, they should

- Render as we expect in standards-compliant browsers.
- Load faster.
- Be 100% future-proof as the Web evolves.

All good goals to be striving for; all expected of a Web Standardista.

### Valid code is not necessarily well-structured code

Passing the W3C Markup Validation Service test doesn’t automatically mean that your page is well structured. Imagine that we’ve marked up the preceding information as follows:

```
...
<p>King Kong</p>
  <h3>King Kong is the name of a fictional giant ape from the legendary
  Skull Island, who has appeared in several works since 1933. Most of
  these bear his name, and include the groundbreaking 1933 film, the
  film remakes of 1976 and 2005, and numerous sequels.</h3>
...
```

Now the most important heading on the page is marked up as a p and the subsequent paragraph has been marked up as an h3. All it takes is a quick glance over this markup to see that something isn’t quite right; it certainly hasn’t been written by an aspiring Web Standardista. The preceding is perfectly valid, however, and the validator will not find any errors in the markup. Remember, make sure that your page is well structured before you attempt to validate it, or in the words of Gary Larson: “First pants, *THEN* your shoes.”

## Getting the search mix right

Structured markup is Google-friendly markup, and Google-friendly markup increases the chances of your web site being found.

Let’s face it, with an estimated 63 billion web pages in existence (and that was just in June 2008), getting found online is a little like finding the proverbial needle in a haystack, only this is a very big haystack. Using a Web Standardistas’ approach helps your web site considerably in the eyes of Google. In this section, we explore why.

By structuring your content in a logical way, using headings, paragraphs, and phrase elements to add meaning to your content, you are not only building the foundations for a POSH site worthy of a Web Standardista, but also helping search engines, which can use your well-structured markup to help them make sense of the contents of your page.

Apart from making sure that your content contains the words or phrases that people are likely to search for, one of the simplest things you can do to guarantee your page is ranked well by Google is to cross-reference the content of your title element and your page's h1 elements and ensure that both feature an intelligent use of words relevant to your web site or web page.

Let's put ourselves in the shoes of someone desperately trying to find useful information about famous primates and take a look at a couple of examples to underline what we mean. In the following examples, we've shown two different versions of the same page. The first tells us—and Google—very little about the page:

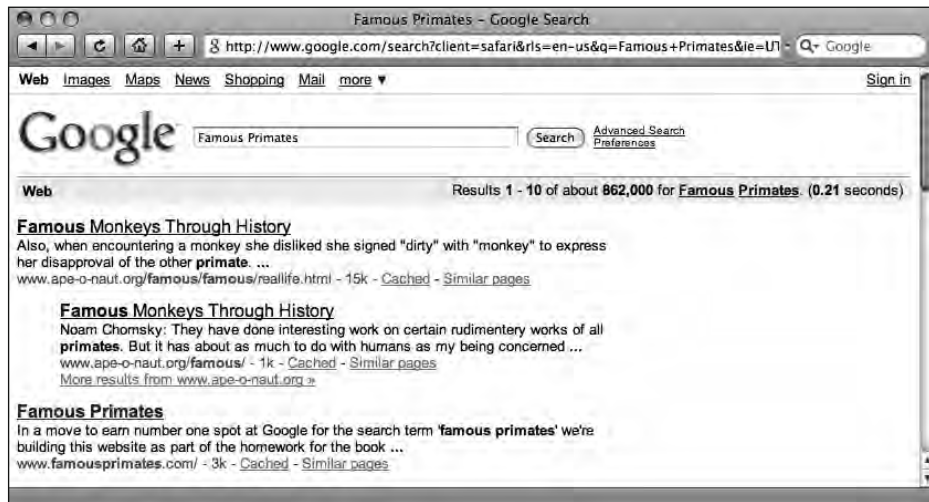
```
...
<title>Gordo and Clyde</title>
...
<h1>Furry Forest Dwellers Found Fame</h1>
...
```

Who are Gordo and Clyde? Using our human powers of deduction, we might determine that there was a chance these furry forest dwellers were indeed primates, and since they found fame they would probably also be famous, but it would be very hard for a machine to equate this information with the words *Famous Primates*.

The following example is much better. It's evident from the title element that this page is the one we've been looking for; it also provides Google with a mix of meaningful keywords that also cross-reference with the h1. Perfect.

```
...
<title>Famous Primates presents Gordo and Clyde</title>
...
<h1>Gordo and Clyde - Two Famous Primates</h1>
...
```

Although we can't completely rule out that the somewhat specialist subject matter helped a little, the homework web site we built to accompany this book is an example of how a well-structured page can make it into the top-three position in Google (as shown in Figure 3-14) within only a few weeks of its launch. 3 out of 862,000 isn't bad.



**Figure 3-14.** After only a few weeks, our page about famous primates reached the third spot in Google.

There are a number of other techniques that help with search engine optimization (SEO); however, what we want to stress here is that search engine–friendly pages are a natural byproduct of the Web Standardistas' approach.

## Summary

So what have we covered? Although you might not think we've done any "design" yet, trust us, *we have*. Even without the addition of style with CSS, we've explored the importance of establishing a strong information hierarchy. This process is central to any good piece of design, and hopefully now that you're marking your pages up using the full range of tags at your disposal, you agree.

Along the way we've looked at the W3C Markup Validation Service and considered its importance in ensuring your web pages display consistently regardless of browser or platform. Lastly, we took a look at a byproduct of the Web Standardistas' approach that ensures your web pages are search engine friendly.

In the next chapter we introduce you to lists, the building blocks of web site navigation. We also introduce you to a number of other tags, ensuring that you have a well-stocked web design toolkit moving forward.

## Homework: Introducing Miss Baker

In the last chapter's homework, you created a simple web page about Albert I, the first-ever monkey astronaut. In this chapter, we've introduced a number of tags to add additional

structure and meaning to your markup. You'll be adding these to another page you'll be creating about Miss Baker, another well-known space pioneer.

Good news: Albert I—now feeling a little lonely in the homework folder—is about to be joined by a lady friend.

As with the last chapter, we'll provide you with all the information you need on Miss Baker; your job will be to add markup to the page to give it some structure. You'll be adding the following: a range of headings from h1 to h4; some additional `<p>` tags to mark up the paragraphs; and some phrase elements, namely `em` and `strong`.

Finally, once you've marked up the Miss Baker page, you'll be validating it using the W3C Markup Validation Service to check that no errors have slipped in during the markup process.

### 1. Establish an information hierarchy

To undertake the last chapter's homework, we provided you with a very simple page about Albert I, consisting of a single h1 and three short paragraphs. In this chapter, we're increasing the level of complexity.

As before, we've supplied you with a text file that you'll be adding markup to. You can access it here:

[www.webstandardistas.com/03/miss\\_baker.txt](http://www.webstandardistas.com/03/miss_baker.txt)

At first glance, the information hierarchy of Miss Baker's page isn't quite as clear as Albert I's page. The first stage in the markup process is simple, but essential: *read everything before you mark up anything*. As you read, your goal is to try to identify an information hierarchy for the text supplied so that you can add effective and meaningful markup in the next stage.

### 2. Add `<h1>`–`<h4>` and `<p>` tags

Once you've read through the text supplied for the Miss Baker page, try to work out the meaning of the words and the page's information hierarchy. Using `<h1>`–`<h4>` and `<p>` tags, apply some structure to the page.

It's worth noting that there are potentially a number of ways to mark this page up, and the choices you make are in some respects subjective. However, to help you in the process, we have created a similarly structured page about Cornelius, famous for his role in *The Planet of the Apes*. You can refer to this using your browser's View Source command to see how we've structured the web page logically, here:

[www.webstandardistas.com/03/cornelius.html](http://www.webstandardistas.com/03/cornelius.html)

### 3. Add `<em>` and `<strong>`

The next stage in the markup process is to identify any phrase elements that might exist within the text. In this case, you're looking for any words that might benefit from the addition of emphasis. Remember, `em` and `strong` are intended for the addition of emphasis to text, not for changing its visual presentation (or look and feel).

Again, you might wish to refer to our Cornelius web page to see where we've added emphasis.

#### **4. Validate your page**

Once you've completed adding the markup to your Miss Baker web page, the final stage in the process will be to check it using the W3C Markup Validation Service. Start by opening the following page in your browser:

<http://validator.w3.org/>

Click the Validate by Direct Input tab, and copy and paste the markup for your Miss Baker page into the validator. Click Check and wait for the results. If you've been a diligent Web Standardista and written all your markup carefully, you'll be welcomed by the green banner. If you're met with the less welcoming red banner, have no fear, the process of debugging your Miss Baker page will in itself be an educational experience.

Only once you're met with the green banner are you allowed to put the kettle on and enjoy a cup of *Earl Grey* as you prepare yourself for the next chapter!