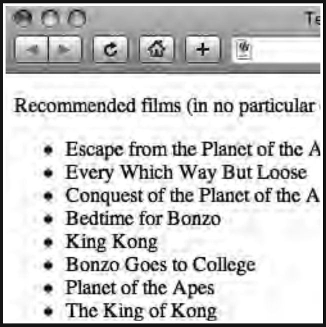
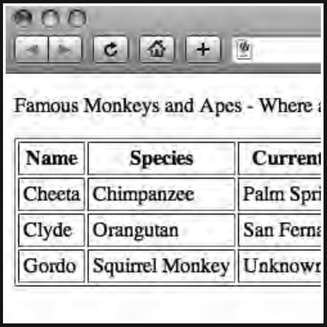
CHAPTER 4

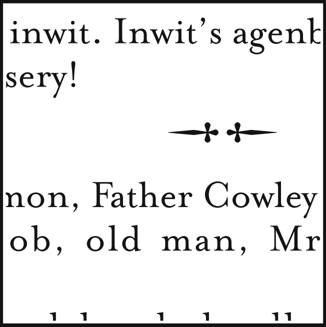
MARKUP THAT ADDS MEANING


Recommended films (in no particular

- Escape from the Planet of the A
- Every Which Way But Loose
- Conquest of the Planet of the A
- Bedtime for Bonzo
- King Kong
- Bonzo Goes to College
- Planet of the Apes
- The King of Kong


Famous Monkeys and Apes - Where

Name	Species	Current
Cheeta	Chimpanzee	Palm Spri
Clyde	Orangutan	San Ferna
Gordo	Squirrel Monkey	Unknowr


inwit. Inwit's agenb
sery!

non, Father Cowley
ob, old man, Mr

In this chapter we look at the topic of adding additional meaning to your markup through the use of a number of new tags we introduce. Our primary focus is to cover a number of methods of organizing and grouping information. In particular, we explore the importance of using lists—unordered lists, ordered lists, and definition lists (don't worry, we'll introduce each of these fully in due course)—to help group together related information. We also introduce tables, often mistakenly maligned, but useful nonetheless for giving form to tabular data.

By the end of this chapter, you'll be ready to build basic lists that will form the backbone of your web site's navigation after we've introduced creating links in Chapter 6. At this point, like your other markup, your lists will remain unstyled, but rest assured you'll style them in good time. You'll also have an understanding of how to use tables to organize tabular data, enabling you to apply structure and meaning to calendars, charts, schedules, and timetables, to give but a few examples.

Finally, and for good measure, we introduce a number of additional tags—bonus tags for every occasion. In a veritable XHTML feast, we supply you with a sizable number of tags that no Web Standardista should be without: tags for quotations—`<blockquote>`, `<q>`, and `<cite>`; tags we're rescuing—`<hr />`; tags for nerds—`<code>` and `<pre>`; tags for writers—`<del>` and `<ins>`; and finally, `<sup>` and `<sub>`, useful, in particular, for our scientist friends.

Lists: First-level organizers

Lists are everywhere: shopping lists, to-do lists, top ten lists, lists of links, lists of links for navigation, and so on. Lists are semantic: they suggest structure or indicate related groups of information, which is why we're using them.

Look at any well-designed web page, and it's likely that the navigation at the top or the side of the page is a list of links that's been styled with CSS. Lists provide a means of grouping information together and making the separate elements easier to grasp. Lists also give us meaningful tags that we can target with CSS, which we'll cover when we get to Chapter 12, where we show you how to style a variety of lists.

Why use lists?

Look at the following two examples—a list of primate-related films. In the first, we structure the list as an inline paragraph; in the second we structure the list using bullet points. Two different approaches that, as you'll see, have an important impact on how they render in a browser and can have an impact on how they're absorbed as groups of information.

Looking at the two examples, it's clear that using a list to both group and structure the information provides an additional layer of meaning for the information supplied.

Recommended films, version 1 (in no particular order):

Escape from the Planet of the Apes, Every Which Way But Loose, Conquest of the Planet of the Apes, Bedtime for Bonzo, King Kong, Bonzo Goes to College, Planet of the Apes, The King of Kong, Beneath the Planet of the Apes, 2001: A Space Odyssey.

Recommended films, version 2 (in no particular order):

- Escape from the Planet of the Apes
- Every Which Way But Loose
- Conquest of the Planet of the Apes
- Bedtime for Bonzo
- King Kong
- Bonzo Goes to College
- Planet of the Apes
- The King of Kong
- Beneath the Planet of the Apes
- 2001: A Space Odyssey

Looking at the preceding examples, we would argue that certain information lends itself to display in list format. Rendered in the browser as a series of bullet points, the list of films is easier to read than the paragraph with the list of films rendered inline. With each film beginning on a new line and with bullet points clearly indicating each new film in the list, we would argue that the list is easier to read when presented this way. As with the previous chapters, we're using the right tag for the job to indicate the structure and grouping of information to the reader, adding design through the careful use of structured markup.

While it's worth noting that we could use CSS to switch off bullet points and render our list inline as a paragraph (something we'll cover in Chapter 12), the point we'd like to make here is that through the careful selection of appropriate tags, we can amplify the meaning of our raw, unstyled information—no bad thing.

In the next section, we show you how to use the basic components of a list in XHTML; along the way we introduce you to three types of list: **unordered** lists, **ordered** lists, and **definition** lists. We'll focus primarily on the first two types of list, as these will provide you with useful methods of marking up groups of information that you can style later using CSS.

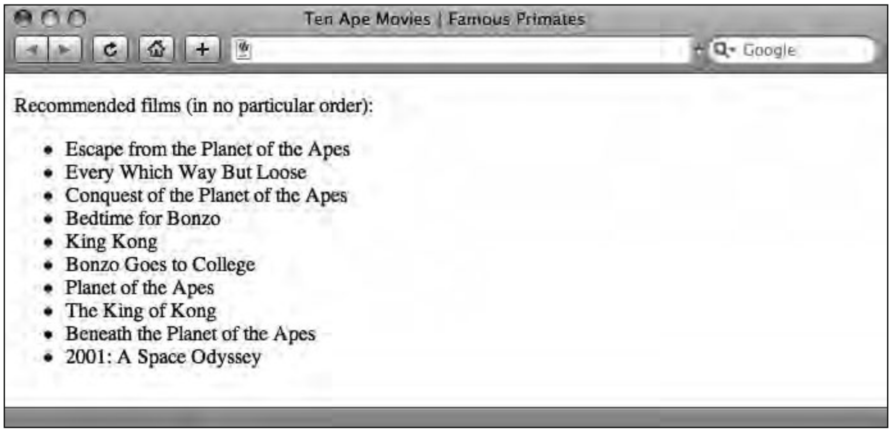
Unordered and ordered lists

We structure unordered lists using two elements: `ul` and `li`. The `ul` element indicates that we are grouping our items in an unordered list, that is, each item in the list is of equal value and the list suggests no inherent order. The `li` element is used for each item—or **list item**—in the list.

Recall the version 2 example of the recommended films (in no particular order) mentioned previously. Marked up, the list looks like this:

```
<ul>
  <li>Escape from the Planet of the Apes</li>
  <li>Every Which Way But Loose</li>
  <li>Conquest of the Planet of the Apes</li>
  <li>Bedtime for Bonzo</li>
  <li>King Kong</li>
  <li>Bonzo Goes to College</li>
  <li>Planet of the Apes</li>
  <li>The King of Kong</li>
  <li>Beneath the Planet of the Apes</li>
  <li>2001: A Space Odyssey</li>
</ul>
```

The preceding markup, when rendered in a browser, looks like what you see in Figure 4-1.



Recommended films (in no particular order):

- Escape from the Planet of the Apes
- Every Which Way But Loose
- Conquest of the Planet of the Apes
- Bedtime for Bonzo
- King Kong
- Bonzo Goes to College
- Planet of the Apes
- The King of Kong
- Beneath the Planet of the Apes
- 2001: A Space Odyssey

Figure 4-1. Our unordered list as it displays in a browser

We open and close our list with `<ul>` and `</ul>` tags, respectively, indicating that what follows is a group of related information. We then list each item in the list within `<li>` tags.

But what if we want to give our list some order? After all, a top ten list isn't much use if we haven't ordered it.

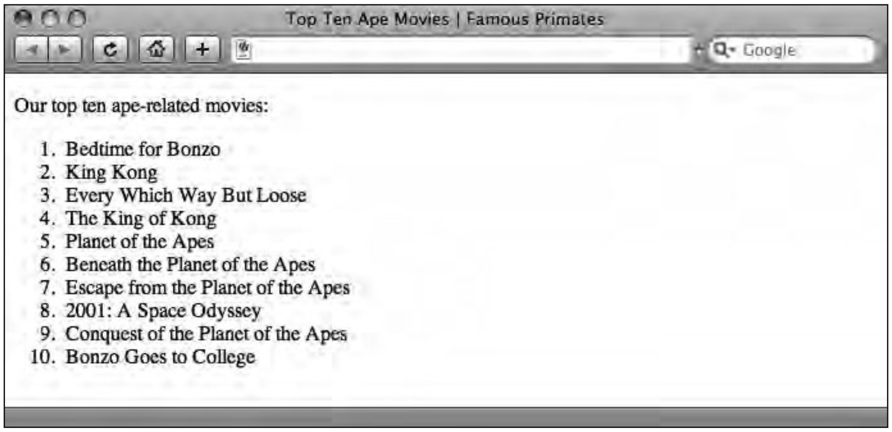
Enter the ordered list

Perhaps you need to get your lists in order, and an unordered list doesn't meet your requirements. Have no fear, HTML provides a means of doing this. Meet the `ol`, or ordered list.

In the previous example, we introduced you to a great list of films, but the browser's default bullet points of a `ul` didn't give us much of a sense of order; in fact, we prefaced the list with the words *in no particular order*. What if you wanted to make a top ten list? Good news, we have an alternative to the `ul` at our disposal. Enter the ordered list, or `ol`. Change the `<ul>` tags in the preceding example to `<ol>` tags, and we now have a top ten list that looks like the following example (you'll notice our list is now ordered differently—in the authors' order of preference):

```
<ol>
  <li>Bedtime for Bonzo</li>
  <li>King Kong</li>
  <li>Every Which Way But Loose</li>
  <li>The King of Kong</li>
  <li>Planet of the Apes</li>
  <li>Beneath the Planet of the Apes</li>
  <li>Escape from the Planet of the Apes</li>
  <li>2001: A Space Odyssey</li>
  <li>Conquest of the Planet of the Apes</li>
  <li>Bonzo Goes to College</li>
</ol>
```

The preceding markup, when rendered in a browser, looks like the list in Figure 4-2.



Our top ten ape-related movies:

1. Bedtime for Bonzo
2. King Kong
3. Every Which Way But Loose
4. The King of Kong
5. Planet of the Apes
6. Beneath the Planet of the Apes
7. Escape from the Planet of the Apes
8. 2001: A Space Odyssey
9. Conquest of the Planet of the Apes
10. Bonzo Goes to College

Figure 4-2. Our ordered list as it displays in a browser

The only thing we had to do, apart from putting the films in the order we felt appropriate, was to change the `ul` to an `ol`. With just that simple change, the browser takes care of the numbering for us. So, if in the event of a recount we wanted to put *The King of Kong* in at number one, we wouldn't have to renumber the whole list; we could simply move `<li>The King of Kong</li>` up three lines in our HTML document. Simple.

Another benefit of the `ol` is that we're not just limited to numerals. As you'll see when you get to Chapter 12, which covers styling lists with CSS, we can exchange our default **1, 2, 3, 4 . . .** (`decimal`) for

- **A, B, C, D . . .** (upper-alpha)
- **a, b, c, d . . .** (lower-alpha)
- **I, II, III, IV . . .** (upper-roman)
- **i, ii, iii, iv . . .** (lower-roman)

We can even use none, which switches numbering off completely, although why we would want an unordered ordered list is another question altogether.

Nesting lists

Before we add a further layer of complexity to our lists, it's worth noting that both the ul and ol elements are block level and can only contain li elements. No text or other elements can appear in a ul or an ol element unless they are contained within <li> tags.

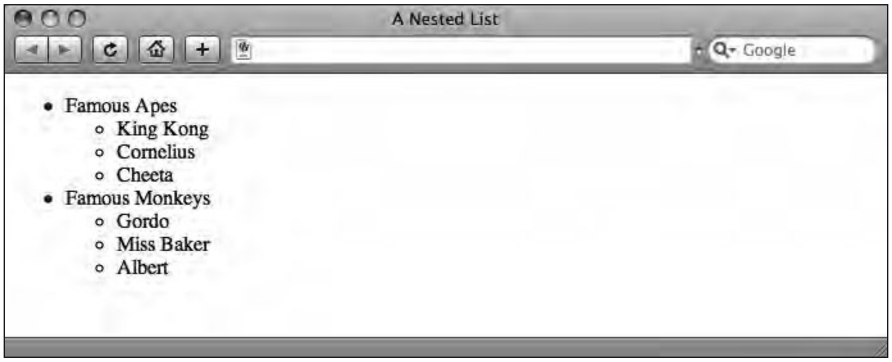
It's possible, however, to create more complex lists through nesting other elements or even other lists within li elements. This is best demonstrated with an example. Let's take a look at how a nested list is constructed:

```
<ul>
  <li>Famous Apes
    <ul>
      <li>King Kong</li>
      <li>Cornelius</li>
      <li>Cheeta</li>
    </ul>
  </li>
  <li>Famous Monkeys
    <ul>
      <li>Gordo</li>
      <li>Miss Baker</li>
      <li>Albert</li>
    </ul>
  </li>
</ul>
```

As lists become more complex, particularly as lists are nested within lists, it's easy to make mistakes, resulting in pages that fail to validate. The preceding list is perfectly valid; however, the apparent lack of a closing </li> tag on the *Famous Apes* and *Famous Monkeys* list items can be confusing for the beginner. At first glance, these list items appear to open, but not to close; however, they are in fact closed, *after* the ul they contain is closed, a number of lines below.

When nesting a list, the containing list item is not closed until the nested list is closed. The easiest way to get this right is to use indentation or white space to clearly indicate the list's structure within your markup.

The preceding markup, when rendered in a browser, looks like what you see in Figure 4-3.



- Famous Apes
  - King Kong
  - Cornelius
  - Cheeta
- Famous Monkeys
  - Gordo
  - Miss Baker
  - Albert

Figure 4-3. The nested list as it displays in a browser

As you can see in the preceding example, as we nest lists within lists, the browser alters the default bullet point to differentiate between the different levels of the nested list. The bullet points your browser uses will depend upon its default style sheet. We'll cover how to control this (and indeed what a default style sheet is) when we get to Chapter 12.

Definition lists

Meet our final list type, the dl, or **definition list**. Definition lists are perfect for lists of definitions, for example, for use in a glossary of terms as might be found at the back of a technical reference. Definition lists consist of three elements: a container—a dl; a definition term—a dt; and a definition description—a dd.

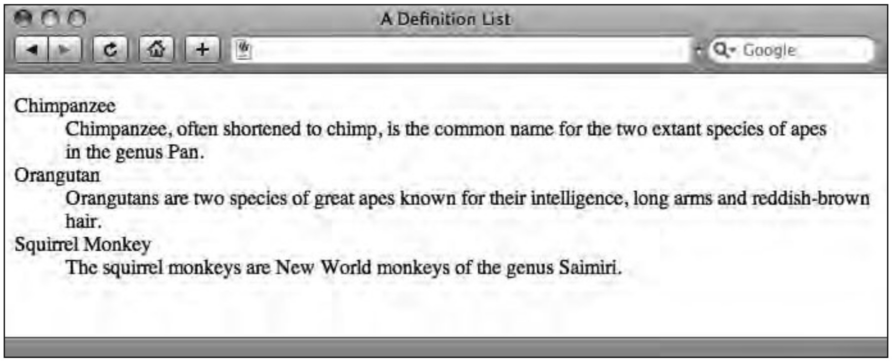
We can demonstrate definition lists best by showing one in action:

```
<dl>
  <dt>Chimpanzee</dt>
    <dd>Chimpanzee, often shortened to chimp, is the common name for
    the two extant species of apes in the genus Pan.</dd>

  <dt>Orangutan</dt>
    <dd>Orangutans are two species of great apes known for their
    intelligence, long arms and reddish-brown hair.</dd>

  <dt>Squirrel Monkey</dt>
    <dd>The squirrel monkeys are New World monkeys of the genus
    Saimiri.</dd>
</dl>
```

The preceding markup, when rendered in a browser, looks like the list in Figure 4-4.



Chimpanzee
    Chimpanzee, often shortened to chimp, is the common name for the two extant species of apes in the genus Pan.
Orangutan
    Orangutans are two species of great apes known for their intelligence, long arms and reddish-brown hair.
Squirrel Monkey
    The squirrel monkeys are New World monkeys of the genus Saimiri.

Figure 4-4. The definition list as it displays in a browser

In this example, our dl gathers related terms and definitions together; the dt elements are the **definition terms** (i.e., the terms we're defining); and the dd elements are the **definition descriptions** (i.e., the descriptions or definitions of the terms).

It's possible to use multiple dt or dd elements within a definition list. The following examples show first one definition term with two different possible definitions, and second, two different definition terms with one definition:

```
<dl>
  <dt>monkey</dt>
    <dd>a primate</dd>
    <dd>a mischievous person, esp. a child</dd>
</dl>
```

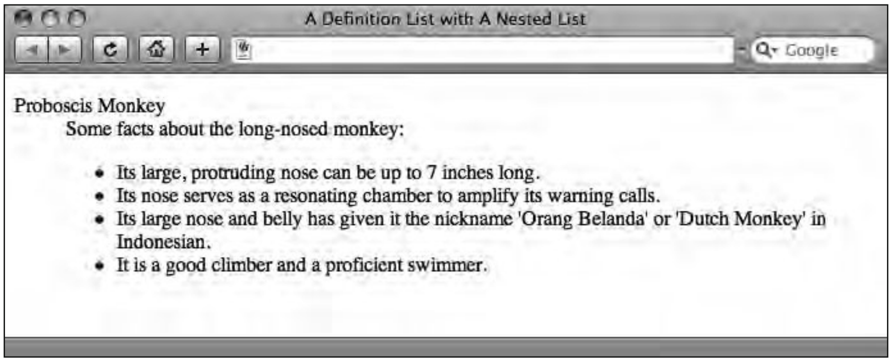
Or

```
<dl>
  <dt>monkey</dt>
  <dt>ape</dt>
    <dd>a member of the primate family</dd>
</dl>
```

We can also nest block level elements within a definition description (dd), as in the following example. Note, however, that block-level elements *cannot* be nested within the dt element:

```
<dl>
  <dt>Proboscis Monkey</dt>
  <dd>Some facts about the long-nosed monkey:
    <ul>
      <li>Its large, protruding nose can be up to 7 inches long.</li>
      <li>Its nose serves as a resonating chamber to amplify its
      warning calls.</li>
      <li>Its large nose and belly has given it the nickname 'Orang
      Belanda' or 'Dutch Monkey' in Indonesian.</li>
      <li>It is a good climber and a proficient swimmer.</li>
      </ul>
    </dd>
  </dl>
```

Figure 4-5 shows how this example looks in a browser.



Proboscis Monkey
Some facts about the long-nosed monkey:

- Its large, protruding nose can be up to 7 inches long.
- Its nose serves as a resonating chamber to amplify its warning calls.
- Its large nose and belly has given it the nickname 'Orang Belanda' or 'Dutch Monkey' in Indonesian.
- It is a good climber and a proficient swimmer.

Figure 4-5. A definition term with a number of possible definition descriptions nested as a list

In short, definition lists offer a way to tie together *terms* and *definitions* and, as with all our examples so far, provide an additional layer of meaning or structure to information.

Tables: The good, the bad, and the alternatives

In the "Wild West" days of the interwebs, as HTML evolved, web designers started to push the tags at their disposal far beyond what they were originally intended for, in an effort to make the Web a more beautiful place.

Although many of these designers were incredibly inventive and managed to create some spectacular layouts, their beauty was only skin deep. (The web sites' beauty that is, not the designers', who, as we know, are beautiful people.)

Looking behind the scenes at the code producing these layouts, however, revealed a complex mass of nested table tags with countless rows and columns creating a precarious scaffolding holding the different pages together.

This was never the purpose of the `table` tag in HTML, which was always intended for gathering together tabular data and giving it structure. Nonetheless, this practice became widespread in an effort to control the look and feel of a rapidly growing Web.

Fortunately for us, browsers evolved, soon developing enough support for CSS to make CSS-based layouts a viable alternative to table-based layouts when designing for the Web. Gone was the need to resort to tables to achieve something they were never intended for.

So, are tables evil, as many early Standardistas believed?

When tables are used for layout purposes, each page includes all the presentational information as well as the content, rather than separating the presentational information into a separate style sheet. As a consequence, table-based layouts result in extra markup that is harder to maintain, and the resulting pages are less accessible to screen readers, mobile devices, and search engines. In short, it's safe to say—when used for layout purposes—tables *are* evil, and many early Standardistas avoided them at all costs.

Used in the right way, however, tables aren't evil and are in fact very useful. Used correctly, tables still have a place in the Web Standardista's arsenal. We know that using tables for layout is strictly off limits, so what do we use tables for? The answer is simple: we use tables for **tabular data**. But what is tabular data?

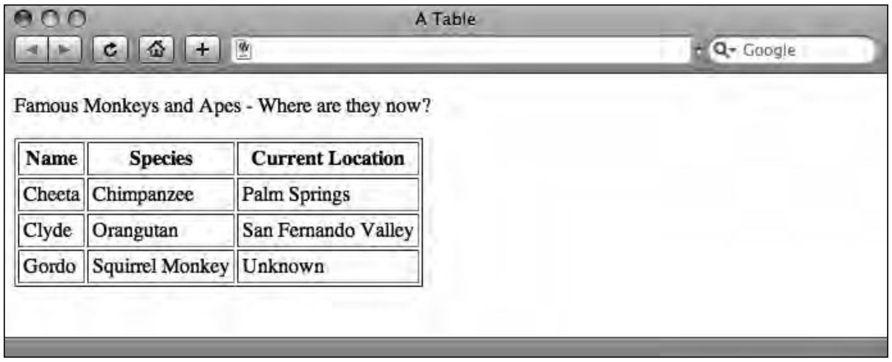
What is tabular data?

Before we look at an actual `table` in action, let's rewind just a little and look at some different types of data. First, let's look at a simple list:

```
<ul>
  <li>Cheeta</li>
  <li>Clyde</li>
  <li>Gordo</li>
</ul>
```

This list can be read from the top down, vertically: Cheeta, Clyde, Gordo. In other words, this data is **one dimensional**. But what if our information is a little more complex? Now we're going to step things up a little and add another dimension to the equation.

Consider the table shown in Figure 4-6.



Famous Monkeys and Apes - Where are they now?

Name	Species	Current Location
Cheeta	Chimpanzee	Palm Springs
Clyde	Orangutan	San Fernando Valley
Gordo	Squirrel Monkey	Unknown

Figure 4-6. A simple three-row, three-column table with table headers. (A table border has been added to reveal the table's underlying structure.)

As you can see, this data can be read in two dimensions: horizontally we can determine that Cheeta is a chimpanzee who lives in Palm Springs. Vertically we can get a list of names

(Cheeta, Clyde, and Gordo), species (chimpanzee, orangutan, and squirrel monkey), and locations (well . . . you get the picture).

Each piece of data can be related horizontally *and* vertically: Clyde is an orangutan who lives in San Fernando Valley. Clyde is also part of a list of names including Cheeta and Gordo. Side to side, up and down. In other words, this data is **two dimensional**. Now we know our two-dimensional tabular data from our one-dimensional list data.

The easiest way to determine whether a table is the right choice for marking up a section of content is to think about whether it could fit into a spreadsheet, like Excel for example. If the grid-like structure of a spreadsheet fits your content like a glove, it's time to roll out the tables.

<table>, <tr>, and <td>

As you've figured out by now, a table consists of rows and columns. We can create a simple table using just three sets of tags, resulting in something like this:

```
<table>
  <tr>
    <td>Cheeta</td>
  </tr>
</table>
```

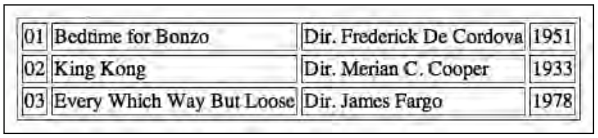
In this example, we start the table with an opening `<table>` tag. Next, the `<tr>` tag starts our **table row**. Between the opening `<tr>` and the closing `</tr>`, we have a single table cell, denoted by the opening `<td>` and the closing `</td>`. `<td>` means **table data**; all content in a table is contained inside `<td>` tags. When all the columns and rows are finished, we have our closing `</table>` tag, wrapping everything up.

A one-row, one-column table is of course of rather limited use, so let's get a bit more adventurous by revisiting our top ten movies list. If we were to add a little more detail, for example, the film's position in our top ten list, its title, its director, and the year it was made, we'd be heading into table territory. Let's see this in action. (Note: We've added a 1px border—`border="1"`—to the opening tag to reveal the table's underlying structure in the illustrations. This, however, is presentational and would normally be handled using CSS.)

```
<table border="1">
  <tr>
    <td>01</td> <td>Bedtime for Bonzo</td>
    <td>Dir. Frederick De Cordova</td> <td>1951</td>
  </tr>
  <tr>
    <td>02</td> <td>King Kong</td>
    <td>Dir. Merian C. Cooper</td> <td>1933</td>
  </tr>
  <tr>
    <td>03</td> <td>Every Which Way But Loose</td>
```

```
      <td>Dir. James Fargo</td> <td>1978</td>
    </tr>
  </table>
```

The preceding example renders in a browser as shown in Figure 4-7.



01	Bedtime for Bonzo	Dir. Frederick De Cordova	1951
02	King Kong	Dir. Merian C. Cooper	1933
03	Every Which Way But Loose	Dir. James Fargo	1978

Figure 4-7. A simple three-row, four-column table

Although this table has three rows and four columns, it follows exactly the same structure as the simple example that we introduced earlier. First, a `<table>` tag instructs the browser we're dealing with a table. Each table row starts and ends with a `<tr>` and `</tr>` tag, respectively. The cells containing the table data are nested within `<td>` and `</td>` tags. Finally, a closing `</table>` tag ends our table.

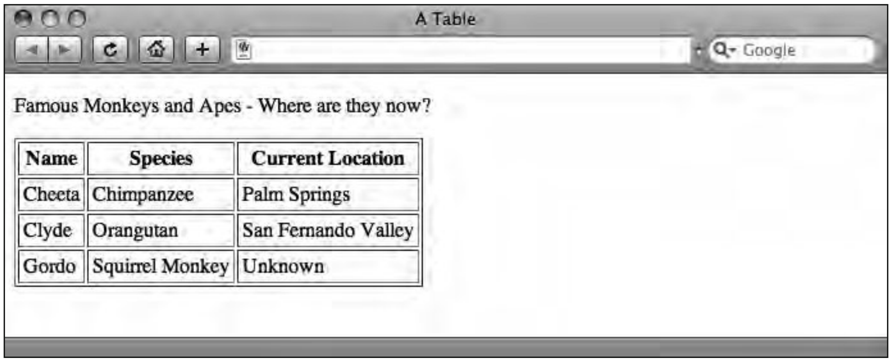
It's worth noting that the number of columns must remain the same in each row. That said, there are a number of ways of merging table cells—a topic we've decided to steer clear of in this book to avoid confusing beginners. If you're feeling adventurous, Paul Haine—our technical reviewer—covers advanced tables in his book *HTML Mastery: Semantics, Standards, and Styling* (friends of ED, 2006).

Improving table accessibility

There are several ways to describe the contents of a table, and, in addition to building your table as outlined previously, it's good practice to describe its contents in summary form. This is particularly useful when accommodating nonvisual browsers for the visually impaired, for example.

We have three tools at our disposal to improve the accessibility of tables: the `th` or **table header**; the `caption`, which provides an indication of the table's content; and the `summary` attribute, a means of describing the content of the table in greater depth. Of these, the first two are aimed at *both* visual browsers and screen readers, while the latter is only aimed at screen readers.

You might argue that we could use `<td><strong>Species</strong></td>` at the top of our Species column in Figure 4-8 (our original table example, repeated here to save you from having to flip back a few pages to find it) to help to visually differentiate the column's header from its contents. However, using `<th>Species</th>` is better. The `th`—a table header—achieves the same *visually* but adds a layer of meaning. `th` is both semantic and, as an added benefit, will be repeated by screen readers as each row of the table's data is read, helping the visually impaired to understand how the table's information is interrelated.



Famous Monkeys and Apes - Where are they now?

Name	Species	Current Location
Cheeta	Chimpanzee	Palm Springs
Clyde	Orangutan	San Fernando Valley
Gordo	Squirrel Monkey	Unknown

Figure 4-8. Our original table example

A screen reader would read the contents of row three of the table as follows:

Name: Gordo; Species: Squirrel Monkey; Location: Unknown

Now that we've got our table headers sorted, we can give our table a `caption`. The purpose of the `caption` is to give the table a title, which displays, by default, above the table. In our original example, the caption reads Famous Monkeys and Apes—Where are they now? This adds a further, useful layer of meaning to our table, summarizing at a glance what the table is about.

Finally, we can include a `summary` attribute, which we'll look at more closely in a moment. For a simple table, table headers and a caption might be sufficient to describe the data they contain. As tables become more complex, however, a well-written `summary` attribute can prove invaluable to the visually impaired user, browsing the table with a screen reader.

Adding a descriptive summary to a table

Tables are great for condensing information. As you've seen in the preceding examples, they're perfect for drawing together connected information in an easy-to-digest manner. Take train timetables, for example. If you wanted to find out when the next train to Paris leaves, a table is the place to look. The same information displayed in paragraphs or even lists would take up far more space, be harder to cross-reference, and, as a consequence, be harder to digest (remember, it would be *one dimensional*).

But what if you didn't want to take the next train to Paris, but wanted to take the next train to Bordeaux instead? Easy, just glance at the Paris table, determine that this information isn't for you, skip right over it, and find the Bordeaux table. Simple, right?

It's only simple if you're using a visual browser. What if you were blind and couldn't use a visual browser? What if you were using a screen reader? A quick glance over the table's content wouldn't work for you because you were visually impaired.

For anyone who uses a screen reader, listening through each row of a table, column after column, to painstakingly find out whether the table contains the information they need could be a somewhat torturous affair. This is where the summary attribute comes to the rescue.

A table's summary is not rendered in visual browsers, but is especially useful for more complex tables where headers or a caption would not be enough to explain the contents of the table; it's perfect for screen readers and as a consequence should be *high on the accessibility agenda*. A well-written summary should give enough information about the contents and structure of a table to give the users of screen readers an idea of a particular table's usefulness. It should clearly suggest whether it's worth sitting through a table's information or whether it's best to just skip it.

So, let's put our summary attribute into action. Wrapping up, our perfectly formed table—now more accessible than our first version—is listed here:

```
<table summary="The most recent established location of monkeys and
apes made famous by Hollywood and/or NASA.">
  <caption>
  Famous Monkeys and Apes - Where are they now?
  </caption>
  <tr>
    <th>Name</th> <th>Species</th> <th>Current Location</th>
  </tr>
  <tr>
    <td>Cheeta</td> <td>Chimpanzee</td> <td>Palm Springs</td>
  </tr>
  <tr>
    <td>Clyde</td> <td>Orangutan</td> <td>San Fernando Valley</td>
  </tr>
  <tr>
    <td>Gordo</td> <td>Squirrel Monkey</td> <td>Unknown</td>
  </tr>
</table>
```

Although we've only covered the essentials of well-formed and accessibly marked-up tables, we've hopefully given you the basics to create well-formatted tables that not only look good in visual browsers, but also function well for users of screen readers.

Formatting tables using CSS can be complicated for the beginner; however, building on a solid foundation of well-formed XHTML is half the battle. We recommend the chapter "Tables are Evil?" in Dan Cederholm's *Web Standards Solutions: The Markup and Style Handbook* (friends of ED, 2004) for anyone wishing to further their knowledge of using CSS to style well-formed CSS tables.

Quoting text

In this section, we introduce two methods of marking up quotes: the first, using the <blockquote> tag, is block-level and is generally used for substantial quotations; the second, using the <q> tag, is inline-level and is generally used for shorter quotations that are better handled inline. Along the way, we encounter both the cite element and the cite attribute, perfect for citing the source of our quotations.

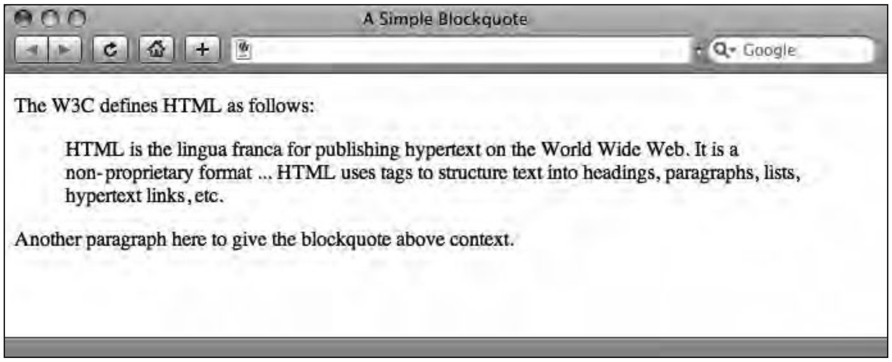
If you're writing an essay, read this!

What's a <blockquote>?

A <blockquote> is a quote block consisting of one or more paragraphs of text, often accompanied by a cite attribute indicating the source, in the form of a URL, from which the block quote was referenced, as in the following example:

```
<p>The W3C defines HTML as follows:</p>
<blockquote cite="http://www.w3.org/MarkUp/">
  <p>HTML is the lingua franca for publishing hypertext on the
  World Wide Web. It is a non-proprietary format ... HTML uses
  tags to structure text into headings, paragraphs, lists, hypertext
  links, etc.</p>
</blockquote>
<p>Another paragraph here to give the blockquote above context.</p>
```

This renders in the browser as shown in Figure 4-9.



The W3C defines HTML as follows:

HTML is the lingua franca for publishing hypertext on the World Wide Web. It is a non-proprietary format ... HTML uses tags to structure text into headings, paragraphs, lists, hypertext links, etc.

Another paragraph here to give the blockquote above context.

Figure 4-9. A blockquote in action

A blockquote element indents the quote using the browser's default settings (although this can be altered using CSS as you'll see in Chapter 10). The blockquote element adds structure to your document, clearly identifying quotes within your marked-up content.

It's worth noting that the blockquote needs to contain a block-level element, usually a paragraph, to remain valid. This is valid markup:

```
<blockquote>
  <p>So what are you, Mr. Driscoll, a lion, or a chimpanzee?</p>
</blockquote>
```

Whereas this markup would not validate:

```
<blockquote>
  So what are you, Mr. Driscoll, a lion, or a chimpanzee?
</blockquote>
```

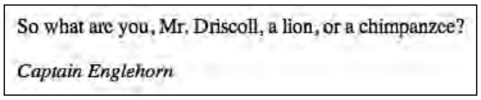
Going back to the example displayed in Figure 4-9, we used the cite attribute to reference the source of our quote. Although the cite attribute is not displayed in the browser, it's worth getting into the practice of using it, as it lets you easily track down the source of a quote you've made by looking at the markup of your document. If you want to display this information in the browser, look no further than the cite element, which comes up next.

Citations (or <cite>)

The cite *element*, not to be confused with the cite *attribute*, is used to indicate a citation or reference to another source. Let's jump straight in and look at an example:

```
<blockquote>
  <p>So what are you, Mr. Driscoll, a lion, or a chimpanzee?</p>
  <p><cite>Captain Englehorn</cite></p>
</blockquote>
```

Figure 4-10 shows this example in a browser.


So what are you, Mr. Driscoll, a lion, or a chimpanzee?

Captain Englehorn

Figure 4-10. A block quote with a cite element, by default displayed in *italics*

You can use the cite element anywhere you may need to reference a different source—for example, a book title, the name of a newspaper or magazine, or the title of a movie. As you've already seen, you can also use the cite element to denote the name of the source a quote is attributed to.

The cite element is not confined to being used within a block quote. For instance, you can use the cite element in a paragraph as in the following example:

```
<p><cite title="King Kong (1933)">King Kong</cite> is a film about a
huge gorilla who takes a shine to the blonde star Ann Darrow.</p>
```

Here we have used cite to denote the title of a movie. As you can see, we have added a title attribute within the cite element, providing additional information. If the source you were referencing was to another web page, you might also consider creating a link to that reference using the a element, something we cover in great detail in Chapter 6.

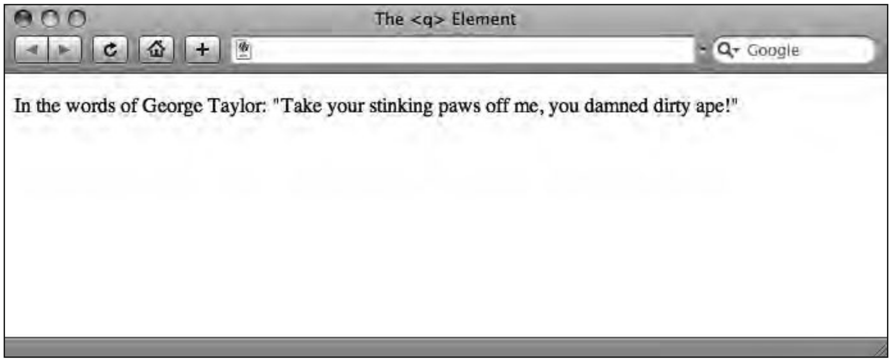
Quotations (or <q>)

A word of warning before we begin: support for <q> tags across browsers is poor at best. However, in the interests of completeness, we're covering them here. Don't say we're not comprehensive.

<q> tags are used to define short quotations that can be included inline, for example:

```
<p>In the words of George Taylor:
<q>Take your stinking paws off me, you damned dirty ape!</q>
</p>
```

This *should* render in the browser as shown in Figure 4-11.



In the words of George Taylor: "Take your stinking paws off me, you damned dirty ape!"

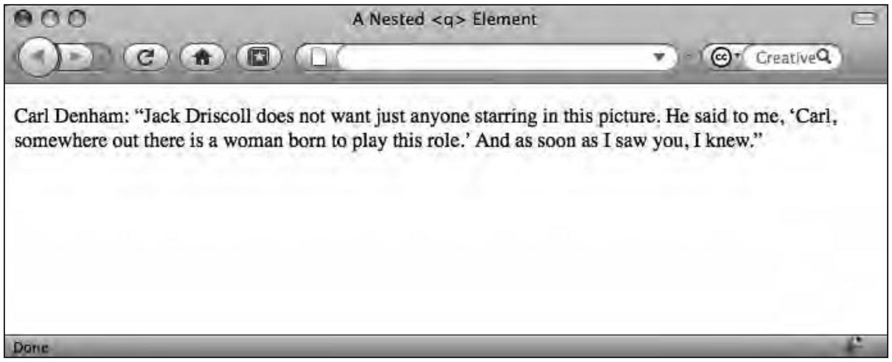
Figure 4-11. A <q> tag rendered correctly by Safari

However, support for the <q> tag is poor, particularly in Internet Explorer, which omits the opening and closing quote marks. This screenshot, taken in Safari, displays the inline <q> as it's supposed to display. (Well done, Safari!)

However, when we nest <q> tags, Safari isn't so hot. What should happen is that the browser should alternate the display of double and single quotes as quotes are nested within quotes, as in the following example, courtesy of Firefox. Safari, however, fails this task miserably, displaying only double quotes. (A little homework for you Safari!) As for Internet Explorer . . . well, let's not go there.

```
<p>Carl Denham: <q>Jack Driscoll does not want just anyone starring
in this picture. He said to me, <q>Carl, somewhere out there is a
woman born to play this role.</q> And as soon as I saw you, I
knew.</q></p>
```

This *should* render in the browser as shown in Figure 4-12.



Carl Denham: "Jack Driscoll does not want just anyone starring in this picture. He said to me, 'Carl, somewhere out there is a woman born to play this role.' And as soon as I saw you, I knew."

Figure 4-12. A nested <q> rendered correctly by Firefox

In closing, using `blockquote` and `q` elements (and including `cite` attributes) adds additional structure and meaning to your XHTML documents. These elements can be styled using CSS, as we'll cover in Chapters 8 to 13, but first and foremost they are semantic, which we by now know is a Web Standardista best practice.

Other tags in the Standardistas' toolbox

There are tags we use day in, day out—`<p>` and `<h1>`, for instance; however, other useful tags exist that, although specialized, can be useful to add specific meaning to your content. In this last section, we take a look at these, which you'll want to add to your Web Standardista's toolbox.

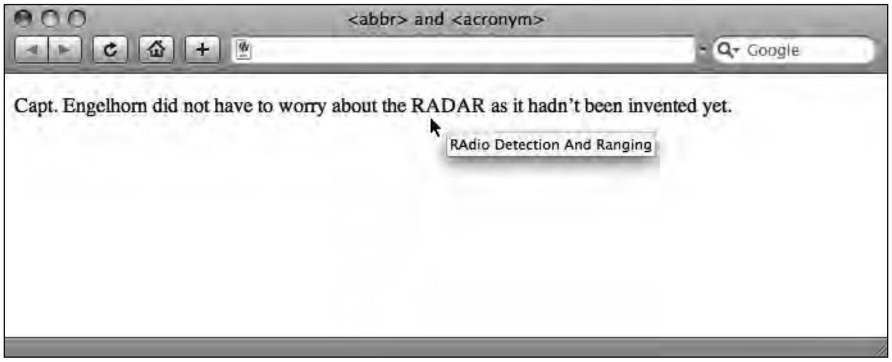
Abbreviations

There are two elements at our disposal when dealing with abbreviations, a shortened form of a word or a phrase. The first is `abbr`, which is used to identify the enclosed text as an abbreviation, for example, *Dr.,* which is short for Doctor or *abbr.*, short for abbreviation. The second is `acronym` which, no surprises here, is used to indicate an acronym. An acronym is a special kind of abbreviation formed from the initial letters of other words, for example, NATO.

Although all acronyms are abbreviations, all abbreviations aren't acronyms. You would think it would be safe to just use the `abbr` element. Unfortunately, earlier versions of Internet Explorer lacked proper support for the `abbr` element, so in past practice, `acronym` has often been used for any kind of abbreviation. Looking forward, however, using `abbr` for all kinds of abbreviations should be the way to go. Let's have a look at `abbr` and `acronym` in action:

```
<p><abbr title="Captain">Capt.</abbr> Engelhorn did not have to worry
about the <acronym title="RAdio Detection And Ranging">RADAR</acronym>
as it wasn't invented yet.</p>
```

As you can see, the `title` attribute in each element is used to provide the expanded form of the abbreviation. As shown in Figure 4-13, standards-aware browsers display the contents of the respective `title` attributes as a tooltip.



Capt. Engelhorn did not have to worry about the RADAR as it hadn't been invented yet.

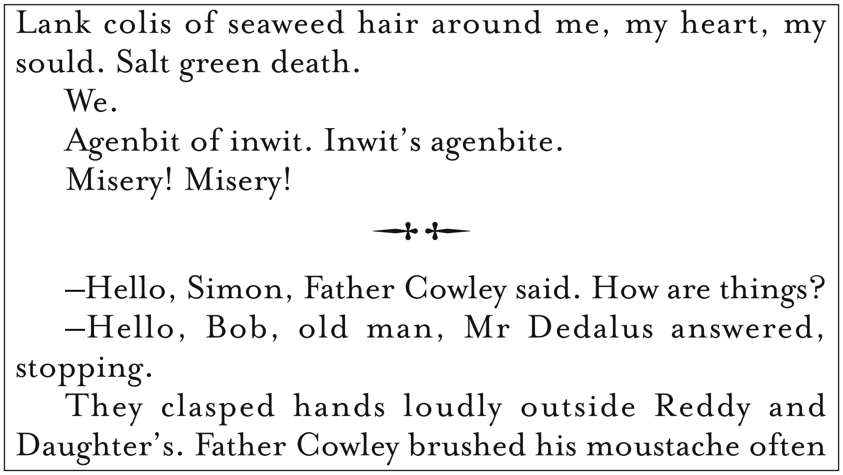
Figure 4-13. The `abbr` and `acronym` elements displayed in a browser

It's worth pointing out that the first time you introduce an abbreviation, it's helpful to include the expanded version in the text at least once. Very common abbreviations—Dr. or NASA, for instance—probably won't need this kind of formal introduction. Less common abbreviations—APE (Advocates for Primate Empowerment), for instance—would benefit from an initial explanation.

Making a case for rules: <hr />

The humble `<hr />` or **horizontal rule** gets a bit of an unfair beating at the hands of a number of noted Standardistas. Patrick Griffiths, the acclaimed writer behind the excellent web-based resource HTML Dog (`www.htmldog.com`), even goes so far as to describe it as a **bad tag**. We, however, beg to differ.

Yes, there are many ways we can create rules or borders using just CSS, but the humble `<hr />` can serve a semantic purpose: as a simple separator of content. Take a look at the example in Figure 4-14, an excerpt from James Joyce's *Ulysses*: essentially this `<hr />` is an example of a lightweight structural section separator.


Lank colis of seaweed hair around me, my heart, my sould. Salt green death.

We.

Agenbit of inwit. Inwit's agenbite.

Misery! Misery!

–Hello, Simon, Father Cowley said. How are things?

–Hello, Bob, old man, Mr Dedalus answered, stopping.

They clasped hands loudly outside Reddy and Daughter's. Father Cowley brushed his moustache often

Figure 4-14. If an `<hr />` is good enough for James Joyce, it's good enough for us.

As this example shows, the `<hr />` can take a variety of forms, in this case an elegant, decorative separator. As XHTML evolves and new tags are introduced, we hope to see a replacement for `<hr />` in the form of a `<separator />` element that clearly reflects a more flexible structural use. This would allow, for example, for its use as a vertical separator for languages written vertically, such as Japanese. For now, however, we're happy to use `<hr />` where it's appropriate as a section separator.

A note on self-closing tags

As you already know, an element consists of a start tag, some content, and an end tag. You also know that, when writing XHTML, you must close a tag whenever you open it. For example, if you open a `<p>` to mark up a paragraph, you must close it with a `</p>` at the end of your paragraph.

There are, however, a few HTML elements that can't hold any content within them and consequently never had a closing tag; some examples include `<br>` (line break), `<img>` (image), and `<hr>` (horizontal rule).

With the stricter rules of XHTML, in particular the insistence that all tags must be closed, the elements that didn't have closing tags in HTML are now treated as self-closing in XHTML. We can make a tag self-closing by adding a space and a forward slash (/) to the end of the tag as in the following example:

```
<p>This is a paragraph. It opens and closes like most elements.</p>
<hr />
<p>The tag above is self-closing.</p>
```

Although the space before the forward slash is not required, some older browsers will get confused without it; in addition, it makes the markup a little easier to read, and therefore we recommend including it.

<code> and <pre>

There are a number of phrase elements useful for describing code samples—of particular use when you're a full-fledged Web Standardista and writing your own web pages with examples of coding best practice. In this section, we look at two: code and pre.

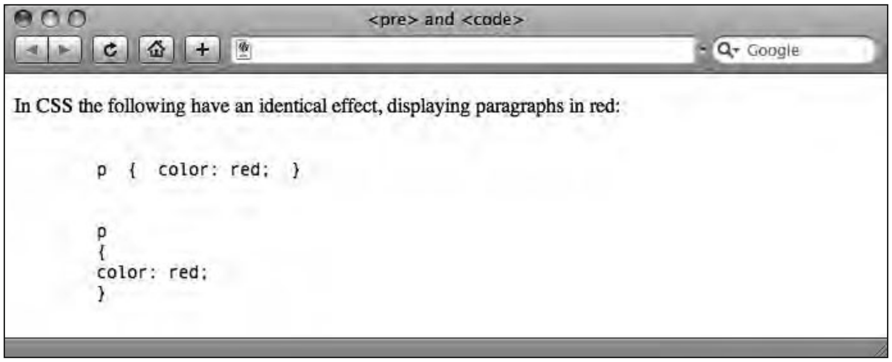
In Chapter 2 we briefly mentioned retaining white-space formatting when displaying poetry within a browser; we're revisiting that concept here in the context of writing code. Cue: the <pre> tag.

If we wanted to include a sample of code or markup in a web page, a very simple CSS declaration for instance, we first need to wrap the markup in some <code> tags that tell the browser that the enclosed contents are code. By default, the browser will render anything nested in <code> tags in the browser's default monospace font, helping to identify that text as an example of code. If we also want to retain the formatting—our white space, tabs, and indentations—we then need to wrap the code element in <pre> tags.

This is demonstrated in the following example:

```
<p>In CSS the following have an identical effect, displaying
paragraphs in red:</p>
<pre>
  <code>
    p  {  color: red;  }
    p
    {
    color: red;
    }
  </code>
</pre>
```

This example renders in a browser as shown in Figure 4-15.



In CSS the following have an identical effect, displaying paragraphs in red:

```
        p  {  color: red;  }


        p
        {
        color: red;
        }
```

Figure 4-15. An example of code and pre used to display a code example in a browser

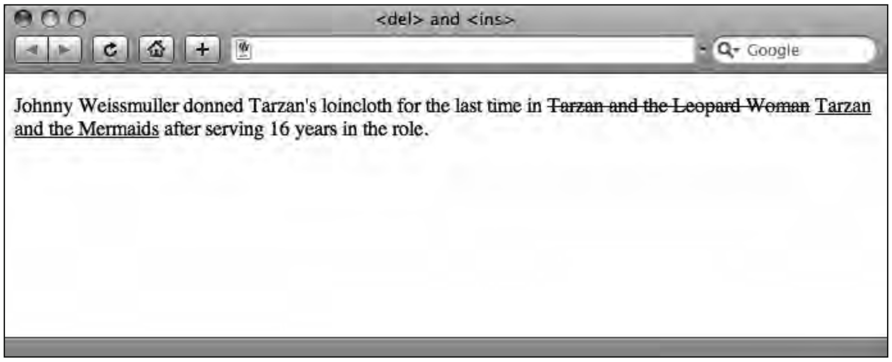
Marking up changes with <del> and <ins>

During the process of writing this book, we've made a number of changes as is only natural. One or the other of us writes some text, and then the other finds a better way of phrasing the same thing (or occasionally spots an error). We can use <del> (delete) and <ins> (insert) tags to indicate these changes clearly.

Take a look at the following example, which shows <del> and <ins> in action:

```
<p>Johnny Weissmuller donned Tarzan's loincloth for the last time
in <del>Tarzan and the Leopard Woman</del> <ins>Tarzan and the
Mermaids</ins> after serving 16 years in the role.</p>
```

Figure 4-16 shows how this example displays.



Johnny Weissmuller donned Tarzan's loincloth for the last time in ~~Tarzan and the Leopard Woman~~ Tarzan and the Mermaids after serving 16 years in the role.

Figure 4-16. Using <del> and <ins> to track changes to a document's content

By default, browsers visually render anything enclosed in <del> tags with a strikethrough or a line through it, while underlining anything enclosed in <ins> tags.

The default underline style that browsers give the ins element is a little bit unfortunate. Most people see underlined text as links and might become frustrated when clicking your perfectly valid <ins> text. A good alternative to underlining is to use CSS to give your ins elements a background color (something that is easily achievable when you have a firm grasp of CSS).

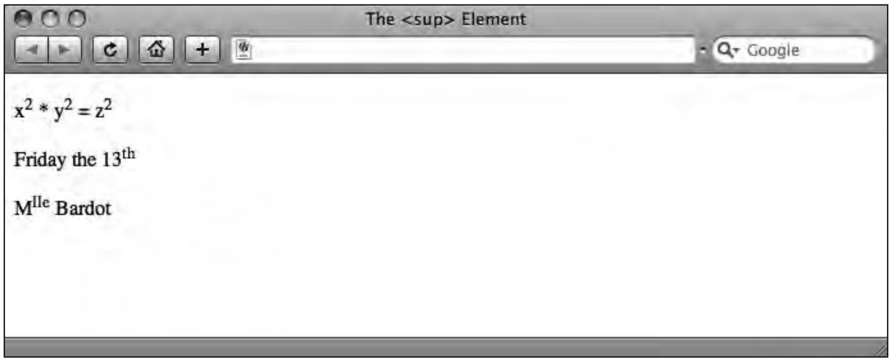
<sup> and <sub>

The <sup> (superscript) and <sub> (subscript) tags have come under fire as being largely presentational; however, there *are* instances where you might wish to use them to add meaning to your markup. In this section, we take a look at some examples that use <sup> and <sub> to convey meaning. If you're a scientist, take note, this section's for you.

First, the superscript. Imagine the Pythagorean theorem without <sup> to supply those all-important squares. Or the classic slasher *Friday the 13th* (not to mention its countless sequels). Or what if you're French and you needed to refer to the abbreviated form of *Mademoiselle*? In all of these cases, <sup> is for you.

```
<p>x<sup>2</sup> * y<sup>2</sup> = z<sup>2</sup></p>
<p>Friday the 13<sup>th</sup></p>
<p>M<sup>lle</sup> Bardot</p>
```

Figure 4-17 shows these examples in action.



$x^2 * y^2 = z^2$

Friday the 13th

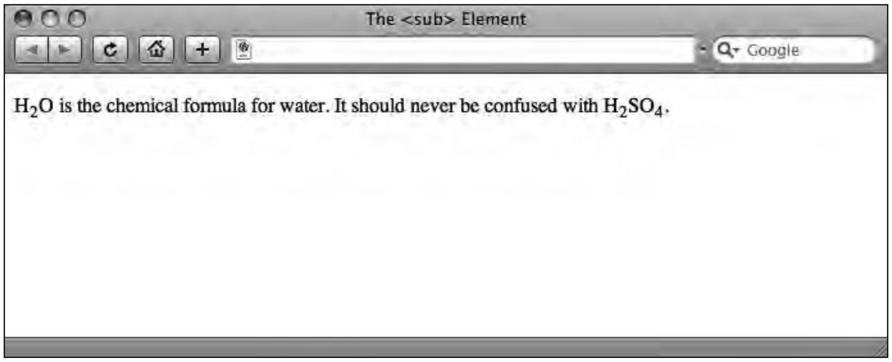
M^{lle} Bardot

Figure 4-17. Some examples of <sup> in action

Now the subscript. Imagine you're a noted chemist and you're ordering a glass of water using only the language you know. You're really looking for a glass of H_2O, not H2O. Without a <sub>, you're not getting the water.

```
<p>H<sub>2</sub>O is the chemical formula for water. It should
never be confused with H<sub>2</sub>SO<sub>4</sub>.</p>
```

Figure 4-18 shows how this example appears in a browser.



H_2O is the chemical formula for water. It should never be confused with H_2SO_4.

Figure 4-18. Example of <sub> in action

In all of the preceding examples, although the use of <sup> and <sub> is presentational, they convey information more clearly than adding a layer of style using CSS. Equally importantly, using <sup> and <sub> ensures our equations and other examples render as we intend them to, even with styling removed.

Summary

So what have we covered? In this chapter, we've explored a variety of methods of adding additional meaning to your markup. We've introduced two important methods of organizing and grouping information: using lists (unordered, ordered, and definition lists) and tables. The former will prove instrumental when we start to add navigation lists to our web pages and link them together.

Throughout the chapter, we've again stressed the importance of using meaningful markup. Finally, we introduced a variety of additional tags that no aspiring Web Standardista should be without.

We're now at a point where we can build complex web pages that are well formed and marked up semantically. Great news, but everything so far has been text based. By now, you're doubtlessly hungry for some imagery.

In the next chapter that's just what we'll cover. Onward.

Homework: Gordo's Adventure

Last chapter's web page for Miss Baker introduced a little more complexity than the humble web page you built for Albert I in Chapter 2. We've introduced a lot in this chapter and, although we're not demanding you use every single tag we've covered, we'd like you to include some of the important tags in another web page you'll create for noted space pioneer Gordo.

Once again, we'll provide you with all the information you need on Gordo. Your job will be to use the appropriate markup introduced in the chapter as and where you see fit.

In addition to the variety of tags you've added to your two web pages so far, this chapter's homework will include the following: both unordered and ordered lists, covering `<ul>`, `<ol>`, and `<li>`; and adding a quote using the `<blockquote>` and `<cite>` tags.

Once again, we encourage you to validate your web page when you've completed the homework using the W3C Markup Validation Service. Yes, you've guessed it, this is something you should be getting into the habit of doing.

1. Explore the content

As with the previous chapter's homework, we encourage you to read over the content first and get a feel for it before diving into the markup. You'll find this chapter's text file with facts on Gordo here:

```
www.webstandardistas.com/04/gordo.txt
```

Again, as you read the text, focus on where it might be appropriate to amplify the text's meaning through the inclusion of unordered lists and ordered lists, and where you'll be including the `<blockquote>` and `<cite>` tags.

Of course, you'll be adding headings and paragraphs, but by now we expect you to do that as a matter of course.

2. Adding unordered lists

We've added a number of facts about Gordo's flight, detailing his reentry speed, his flight's launch time, how long he was weightless for, and his total journey time—a historic 15 minutes. These short lists of facts are the perfect place to introduce unordered lists.

Take a look at the content and, using `<ul>` and `<li>` tags as appropriate, mark up the unordered lists on the Gordo web page.

Once again, we've created a file for you to refer to. Using your browser's View Source menu command, you can look at how we've structured our matching web page for King Kong here:

```
www.webstandardistas.com/04/king_kong.html
```

3. Structuring the references

Our list of references at the bottom of Gordo's page lends itself to being marked up as an ordered list of references using `<ol>` and `<li>` tags. Again, you can refer to our King Kong page for guidance.

It's worth noting that at this point we're simply referencing our sources, as all good students should; we're not including links. We'll add links to these references in Chapter 6 when we introduce links properly.

4. Marking up a block quote

Your Gordo page has a quotation by Donald "Deke" Slayton, one of the original Mercury Seven NASA astronauts. Using the `<blockquote>` and `<cite>` tags, mark this quotation up, citing Deke Slayton as the source.

5. Check for errors

As outlined in the previous chapter's homework, this is a good to time to avail yourself of the W3C Markup Validation Service to check you've got everything right. If as a result of the added complexity of the page you're seeing the dreaded red banner, don't—repeat don't—put it to the back of your mind. Fix the problems and revalidate.

If we're mentioning this again, it's simply to emphasize that using the validator will save you problems later and in the process will add to your understanding.

When you're error free, feel free to put the kettle on and enjoy a cup of *Russian Caravan* as you prepare yourself for the next chapter!