understanding web browsers

A *web browser* is the most important piece of software a Web designer must work with. It's the program you use to view pages and navigate the Web. The core purpose of a web browser is to connect to web servers, request documents, and then properly format and display those documents. Web browsers can also display files on your local computer, download files not meant to be displayed, send/receive email, and, depending on the features enabled, be able to play Flash animations, open pdf files, or run Java applets.

What the browser does best, however, is deal with retrieving and displaying web documents. Different browsers might format and display the same file in different ways, depending on the capabilities of the system and how the browser is configured.

Rendering Engines

A *rendering engine,* also known as a *layout engine*, is the code that tells the browser how to display web content and available style information in the browser window. The rendering engine is responsible for the size of an unstyled <h1> heading or how a horizontal rule looks on the page. It's also the key to the correct implementation of CSS and other web standards.

The first separate and reusable rendering engine was Gecko, released by the Mozilla developers in 1998. It was notable for its small size and excellent support for standards. Today web developers pay close attention to underlying rendering engines as a key to understanding a browser's performance.

The story of the browser provides useful context for the way web sites are currently designed and developed. Here are a few of the significant events in the development of the major browsers that have led to the current web design environment:

1991 to 1993: The World Wide Web is born.

Tim Berners-Lee started his hypertext-based information management at the CERN physics research labs. Text-only pages could be viewed using a simple line-mode browser.

1993: NCSA Mosaic is released.

The Mosaic browser was created by Marc Andreessen, a student at the National Center for Supercomputing Applications (NCSA) . Although it was not the first browser to allow graphics to be placed on the page, it was certainly the most popular due to its cross-platform availability. The ability to add images to documents was one of the keys to the Web's rapid rise in popularity. Mosaic also supported sound, video, bookmarks, and forms. All web pages at this time were displayed in black text on a gray background .

1994: Netscape 0.9 is released.

Marc Andreessen formed Mosaic Communications Corp. (which later became Netscape Communications) and released the Netscape 0.9 browser. The early browsers were not free (except to students and teachers). To offer a superior experience over such freely available browsers as Mosaic and thereby attract customers, Netscape created its own HTML tags without regard for the traditional standards process. For example, Netscape 1.1 included tags for changing the background color of a web page and formatting text with tables.

1996: Microsoft Internet Explorer 3.0 is released.

Microsoft finally got into the Web game with its first competitive browser release, complete with its own set of tags and features. It was also the first browser to support style sheets, which at the time were an obscure authoring technique.

1996 to 1999: The Browser Wars begin.

For years, the web development world watched as Netscape and Microsoft battled it out for browser market dominance. The result was a collection of proprietary HTML tags and incompatible implementations of new technologies, such as JavaScript, Cascading Style Sheets, and Dynamic HTML. On the positive side, the competition between Netscape and Microsoft also led to the rapid advancement of the medium as a whole.

1998: Netscape releases its Communicator code under an open source license.

This bold move enabled the thousands of developers to participate in improving Communicator. In the end, they decided to scrap it all and start from scratch. The Mozilla group, made up in part of Netscape employees, guided the development of the open source browser and soon expanded to a complete application platform.

2000: Internet Explorer 5 for the Mac is released.

This is significant because it is the first browser to fully support the HTML 4.01 and CSS 1 Recommendations, setting the bar high for other browsers in terms of standards compliance. It is also the first browser to fully support the PNG format with alpha transparency.

2000: Netscape is sold to AOL.

This was regarded as Netscape's official loss to Microsoft in the Browser War. Entwined in the operating system of every PC running the Windows operating system, Internet Explorer was a formidable foe. Netscape lost important ground by releasing bloated all-in-one applications and taking several years off to rewrite its browser from scratch for the Netscape 6 release. As of this writing, Netscape is just a blip on the browser usage charts at a mere 1% for all combined versions, compared with approximately 90% for all combined versions of Internet Explorer.

2003: The Mozilla Foundation is formed.

Open source Mozilla code continued development under the newly formed Mozilla Foundation (funded in part by AOL).

2005: Mozilla's Firefox browser is released.

Firefox 1.0 caused much fanfare in the development community due to its strong support of web standards and its improved security over Internet Explorer. Firefox is important because it was the first browser to make a significant dent in Microsoft's share of the browser market.

web standards & the W3C

The World Wide Web Consortium (W3C) creates and oversees the development of web technologies, including HTML, CSS, and their numerous applications. They also keep their eye on higher-level issues such as making content accessible to the greatest number of devices and users, as well as laying a common foundation for future development, thus making web content "forward compatible."

The W3C is not an official standards body, but rather a joint effort by experts in web-related fields to bring order to the development of web technologies. The W3C releases its final word on how various tasks (such as HTML markup) should be handled in documents called "Recommendations." Most of their recommendations become the standards for web development.

Okay, so standards are great, but what standards are we talking about? Let's look at the current standards for the structural, presentational, and behavioral aspects of web design:

- Use *HTML* to **structure** your documents
- Use *CSS* to attach **presentational** information
- Use *JavaScript* to apply **behavioral** instructions

Keep in mind that presentation and behavior are always present, even if you don't give any specific instructions. Web browsers will apply their own default styles and event handlers, either preprogrammed or user-defined. For example, most browsers will apply margins to `<p>` elements or display the contents of `title` attributes as tool tips when a user hovers over an element.

Web design and development is commonly discussed in terms of layers, borrowing a layer model from one commonly used for describing network protocols. The marked-up document forms the *structural layer*, which is the foundation on which other layers may be applied. Next comes the *presentation layer*, specified with Cascading Style Sheets, that provides instructions on how the document should look on the screen, sound when it is read aloud, or be formatted when it is printed. On top of these layers, there may also be a *behavioral layer* , the scripting and programming that adds interactivity and dynamic effects to a site.

HTML: the structural layer

The *structural layer* is created with HTML. The *tags* (words or letters enclosed in brackets) form *elements* that describe the meaning of the content. For example, the <h1> element conveys the information, "This content is a heading." But the element does not include any information about how the content should be displayed:

```
<h1> This is a heading.</h1>
```

The original intent of a markup language is to describe the structure of the document, not to provide instructions for how it should look. The structural markup of the document forms the foundation on which the presentational and behavioral layers may be applied. These are the current standard languages for structural markup:

• *XML 1.0 (eXtensible Markup Language)*
XML is a set of rules for creating new markup languages. It allows developers to create custom tag sets for special uses.

• *XHTML 1.0 (eXtensible Hypertext Markup Language) and XHTML 1.1*
XHTML 1.0 is simply HTML 4.01 rewritten according to the stricter syntax rules of XML. XHTML 1.1 finally did away with deprecated and legacy elements and attributes and has been modularized to make future expansions easier. The last version of HTML was HTML 4.01, which is still universally supported by today's browsers, but is not forward compatible. XHTML 2.0 development was then abandoned in favor of HTML5.

• *HTML5*
This revision of the language introduces more meanful stuctural elements, multimedia handling, offline storage and geolocation.

It is very important to recognize markup for what it is: markup such as XHTML is structured content, not a visual construct such as an image with elements placed at different coordinates. When you have a proper, valid XHTML document you can then access and change it via CSS or DOM Scripting.

CSS: the presentational level

The *presentational layer*, achieved with Cascading Style Sheets (CSS), describes how the content should be displayed. You can use CSS to declare, “Paragraphs should be colored grey and use Arial or some other sans-serif font”:

```
p {
    color: grey;
    font-family: “Arial”, sans-serif;
}
```

With all presentation instructions removed from the markup, styling content is the exclusive job of Cascading Style Sheets. Style sheets standards are being developed in phases, as follows.

• *CSS Level 1*
This style sheet standard has been a Recommendation since 1996 and is now fully supported by current browser versions. Level 1 contains rules that control the display of text, margins, and borders.

• *CSS Level 2.1*
This recommendation is best known for the addition of absolute positioning of web page elements. Level 2 reached Recommendation status in 1998, and the 2.1 revision is a Candidate Recommendation as of this writing. Support for CSS 2.1 is still somewhat inconsistent in current browser versions.

• *CSS Level 3*
This recommendation contains many new modules supporting animation, 2D/3D transitions, text effects, multiple columns, and other effects.

JavaScript: the behavioral level

The *behavior layer* describes how the content should react to events. This is the domain of Document Object Model (DOM) scripting, or JavaScript. Using the DOM, you can specify, “When the user clicks on a paragraph, display an alert dialog”. The scripting and programming of the behavioral layer adds interactivity and dynamic effects to a site.

The DOM allows scripts and applications to access and update the content, structure, and style of a document by formally naming each part of the document, its attributes, and how that object may be manipulated. In the early days of the Web, each major browser had its own DOM, making it difficult to create interactive effects for all browsers.

• *DOM Level 1 (Core)*
This version covers core HTML and XML documents as well as document navigation and manipulation.

• *DOM Level 2*
Level 2 includes a style sheet object model, making it possible to manipulate style information.

Netscape introduced its web scripting language, *JavaScript*, with its Navigator 2.0 browser in 1995. It was originally called *Livescript* but was later co-branded by Sun, and “Java” was added to the moniker. Microsoft countered with its own *JScript* while supporting some level of JavaScript in its Version 3.0 browser. The need for a cross-browser standard was clear.

• *JavaScript 1.5/ECMAScript 262*
The W3C is developing a standardized version of JavaScript in coordination with the ECMA International, an international industry association dedicated to the standardization of information and communication systems. According to Mozilla, Netscape’s JavaScript is a superset of the ECMAScript standard scripting language, with only mild differences from the published standard. In general practice, most developers simply refer to “JavaScript,” and the standard implementation is implied.

Standards-compliant browsers such as Chrome, Firefox, Safari, and Opera are now in common use, so there’s no reason for designers and developers not to create standards-compliant Web content. For those coming from various other design fields, the biggest mind shift towards making standards-compliant sites is keeping presentation *separate* from structure.

creating standards-based designs

It was difficult to recognize HTML as a structural language when it was full of presentational elements and attributes (like `bgcolor`, `align`, and of course, <font> that define how elements look on the page. The W3C *deprecated* those elements in the HTML 4.01 Recommendation and removed them entirely from XHTML onward. What remains is a markup language suited for the original purpose of logically describing the meaning of content elements (*semantic markup*) and establishing the basic hierarchical outline (or structure) of the document. The way the document is visually (or aurally, in the case of speech browsers) presented should be handled entirely by style sheets.

Here are some guidelines that will get you on the right track for designing with web standards:

Don't choose an element based on how it looks in the browser.

Now that you can make any element look the way you want with a style sheet rule, there is no reason to use an <h3> because it looks less clunky than an <h1>, or a <blockquote> just because you want an indent. Take the time to consider the *meaning* or *function* of each element in your document and mark it up accurately.

Avoid deprecated elements and attributes.

There is a well-supported CSS property to replace every element and attribute that has been deprecated in the HTML 4.01 Specification. Using a style sheet will give you greater control and can potentially make future changes easier.

Avoid using tables for layouts.

Ideally, tables should be used exclusively for tabular data. It is now entirely possible to create rich page layouts using CSS alone with no tables. In addition to being semantically incorrect, nested tables result in bloated files and take browsers several passes to display. For those accustomed to thinking in terms of tables, it requires relearning page layout from the ground up, but now is the time to start the process.

Use a DOCTYPE Declaration.

Every HTML document should begin with a DOCTYPE declaration that tells the browser which language your document was written in. An example of a DOCTYPE declaration for a document written in strict HTML5 looks like this:

```
<!doctype html>
```

Not only is it the correct thing to do according to the W3C, but current browsers have the ability to switch into different rendering modes based on the DOCTYPE. Omitting the DOCTYPE may adversely affect the way your page renders in the browser in addition to prevent you from validating your code.

Validate Your Markup.

You can't play fast and loose with modern HTML the way you could with old school HTML. Code written incorrectly may render strangely or not at all. While HTML was always meant to be validated, it is now more important than ever to validate your markup before you publish your content on the Web. Some HTML editors have built-in validators. You may also use the W3C's free validation tools for HTML (validator.w3.org) and CSS (jigsaw.w3.org/css-validator). The Web Developer Toolbar plug-in for Firefox and Chrome gives you a direct link to the validator to test both local and remote Web pages.

Sometimes the error reports a validator spits out can be overwhelming. One of the problems is that errors are inherited, so if you make a mistake early on (such as forgetting to close a tag) the validator gripes about it in multiple error lines. Try fixing early mistakes and then validating again; chances are, the error list will reduce. The W3C has published a list of common error messages and how to interpret them at validator.w3.org/docs/errors.html.

building a basic web page

HTML is a *markup language*. A markup language describes only the structure of a document; it is not concerned with how it looks. When you are marking up a document for the Web, the special meaning you are adding indicates the *structure* of the document, and the markup indicates which part of the document is a heading, which parts are paragraphs, what belongs in a table, and so on. This markup in turn allows a Web browser to display your document appropriately. Markup actually describes what you will find between the tags, and the added meaning the tags give is describing the structure of the document.

```
<!doctype html>
<html>

<head>
<meta charset="utf-8">
<title>Untitled</title>
</head>

<body>

<!-- begin content -->

</body>
</html>
```

These are the minimum requirements for a valid, well-formed HTML5 document.

adding HTML comments

Comments enable you to hide content or markup for temporary purposes or backward compatibility, to identify sections within your document, and to provide directives for other who might be working on the page. The syntax for an XHTML comment looks like this: `<!-- comments -->`

What you are hiding, identifying, or providing in terms of guidance goes between the opening and closing portions of a comment.

declaring and identifying the document

The first thing you'll want to do in your page is add a bit of code that declares which type of document you're using and identifies the language version. With this declaration at the top of your document, you will be able to author the document and then run it through a validator to check for conformance. The validator uses the information in the declaration and compares your document to the DTD you've declared. A DTD is simply a long laundry list of allowed elements and attributes for that language and language version.

```
<!doctype html>
```

adding the <html> root element

After the DOCTYPE declaration, you'll want to begin building your document from its *root* element. It is the element that encloses all following elements.

```
<html>

</html>
```

Remember that an HTML document is built using elements, and elements are the building blocks that create the structures of a web page. The <html> element is considered the root element of any HTML document. Additionally, in HTML we have to add one other important piece to the opening tag, and that's the XML namespace for HTML. This is just another way of identifying the language being used within the document and it must be there to validate.

```
<meta charset="utf-8">
```

These are the very basic beginnings of a document, with the DOCTYPE declaration and the <html> root element in place. The <html> element can contain only two child elements: the <head> element and <body> element. You'll begin adding these two other important pieces of the document next.

relics: *the XHTML doctype*

As the previous standard for many years, you will still see many pages on the Web written in XHTML. An XHTML page contains a slightly more involved doctype which has the same purpose as the HTML5 doctype, but comes in two varieties. *Transitional* was meant to allow for the use of depreciated elements while *Strict* was meant to be be used for future-proof, strictly semantic pages containing no depreciated elements or attributes:

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

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

the document <head>

Often referred to as the header of the page, The <head> element contains information about the document, but no information that will be displayed on the page itself. The header provides a place to include important information about the document to users, browsers, and search engines. It can also contain scripts and embedded style sheets.

The <head> element may include any of the following elements in any order: <title>, <meta>, <style>, <link>, and <script>. The <head> element merely acts as a container of these elements and does not have any content of its own. Most of the data contained within the document header is never actually rendered as content visible to the user.

<title>

You should specify a title for every page that you write inside the <title> element. The <title> element is the only *required* element within the <head> element. This element displays any text within it in the browser title bar along with the browser's name. If users decide to add the page to their bookmarks or favorites, the <title> will be used to name the bookmark. Aside from the fact that you have to have the <title> element in place, a good title accomplishes three things:

- provides a title for the page
- offers users orientation to help them know where they are within the site
- provides additional information about the site page
- important keywords should go in a page's title

The <title> element is used by search engines to determine what your page contains, and what information about it should be displayed in the search results. You can include some of the same keywords as part of the <meta> information, but <meta> information is not as important to search engines as the actual content of the page, such as it's title.

<meta>

Although it is not required in a document, the <meta> element performs so many different functions that it's a good idea to become familiar with it. The <meta> element is an example of an *empty element* (or self-closing element) and doesn't require a closing tag.

Meta information (called *meta tags* for short) is a mechanism you can use to provide information about a page such as keywords and descriptions, character encoding, and document authorship (the term derives from the Greek word "meta", which means "behind" or "hidden"). Meta refers to the aspect of something that is not immediately visible, perhaps because it is in the background, but which is there nonetheless and has an impact. Here are some of the meta element's common attributes:

• *Character encoding*:
It is recommended (although not required) that the media type and character encoding be specified within HTML documents as a way to keep that information with the document. Character encoding means setting the character set for your page, which is particularly important when writing documents in other languages. The character encoding describes the set of actual glyphs, or character shapes, that your document uses.

UTF-8 includes the characters needed for displaying web pages in just about any language. UTF-8 can handle just about any language there is and most computers can read it, so UTF-8 is always a safe bet. For documents written in English, another common character encoding is ISO-8859-1, which consists of all the characters in Western European languages.

```
<meta charset="utf-8">
```

• *Meta names for search engines*:
The most common meta tags provide a description and keywords for telling a search engine what your web site and pages are all about. Each meta tag begins with a name attribute that indicates what the meta tag represents.

```
<meta name="description" content="This course will provide an introduction to the interactive design field as well as familiarity with both Macintosh and PC platforms. Some topics covered will be basic digital imaging and programming, including basic HTML and CSS." />

<meta name="keywords" content="Photoshop, web design, interactive media design, web programming, Flash, web design business, web animation, writing, digital photography, Dreamweaver, image processing, vector graphics" />
```

Although the word 'web' is used a great deal, it's in combination with other keywords. Most search engines will lock you out if you use multiple single keywords. This used to be a way of getting higher ranking, but no longer. Use keywords that make sense, or if you want to have multiple instances of a word, use it in a realistic combination. In the early days of the Web, intelligent use of meta information was a crucial element of Search Engine Optimization. But since anyone can add any meta keyword and description information they'd like to a site or page, the feature has been widely abused and today's search engines discount meta data in favor of other variables.

Meta names can also identify the author of page and the copyright information for a document:

```
<meta name="author" content="Joseph Student" />
<meta name="copyright" content="2015, My Media Company" />
```

• *Client Pull:*
Client-pull refers to the ability of the browser (the client) to automatically request (pull) a new document from the server. The effect for the user is that the page displays, and after a period of time, automatically refreshes with new information or is replaced by an entirely new page. This technique can be used to automatically redirect readers to a new URL.

The following example instructs the browser to reload the page after 30 seconds:

```
<meta http-equiv="refresh" content="30" />
```

To reload a different file, provide the URL for the document within the `content` attribute:

```
<meta http-equiv="refresh" content="1; url=http://index.html" />
```

<style>

One method for attaching a style sheet to an HTML document is to embed it in the head of the document with the <style> element. This is called, unsurprisingly, an *embedded* style sheet:

```
<style type="text/css">
<!--
.headings {font-family: Geneva, Arial, Helvetica, sans-serif}
-->
</style>
```

<link>

The <link> element is most commonly used to link to an *external* Cascading Style Sheet, although it can be used for purposes such as linking to an alternative page for accessibility, or to link to a favicon, those icons you see in the address bar on certain websites.The following example links an external style sheet to the document:

```
<link rel="stylesheet" href="pathname/stylesheet.css" type="text/css" />
```

<script>

This element enables you to insert scripts directly into your document or, as is the preference, link from the page to the script you'd like to use. JavaScript and VBScript code may be added to the document using this element. Externalizing your JavaScript gives you the opportunity to include the same functions or functionality on multiple pages:

```
<script src="my_script.js"></script>
```

An embedded example:

```
<script>
function MM_popupMsg(msg) { //v1.0
  alert(msg);
}
//-->
</script>
```

the document <body>

Often referred to as the *body* of the page; the body element holds the *actual content* of the page that is displayed in the browser window or read by a speech browser. It consists of the opening <body> tag, closing </body> tag, and everything in between. The `body` element contains all the elements in the normal document flow. For visual browsers, the `body` acts as a canvas where the content appears. Audio browsers may speak the content of the `body`.

block-level vs. inline-level elements

All elements can be thought of as either *block-level* or *inline-level* elements. In most cases, block-level elements will always be displayed on a new line, like a new paragraph in a book. Inline-level elements will be displayed one after the other in the same line, like the next word in a paragraph. Keep in mind that this is just the browser's default way of displaying these elements however, and you can change it all with CSS later.

block-level elements

Block-level elements are considered the biggest structural pieces of the Web page, and make up the main content structure. They can usually contain other block-level elements, inline-level elements, and text. Compared to inline-level elements, there are relatively few block-level elements.

Block-level elements appear on the screen as if they have a vertical margin before and after them. A block-level element cannot appear within an inline element. For example:

paragraphs `<p>` (paragraph elements are unique in that they can't contain other block-level elements)
headings `<h1>, <h2>, <h3>, <h4>, <h5>, <h6>`
lists `<ul>, <ol>, <li>, <dl>, <dt>, <dd>`
preformatted text `<pre>`
horizontal rule `<hr>`
line break `<br>`
quotations `<blockquote>`
tables `<table>, <tr>, <th>, <td>, <thead>, <tfoot>, <tbody>`
video `<video>`

These are all block-level elements. They all start on their own new line, and any block-level element that follows them appears on its own new line, too.

Other block-level elements are intended to establish a meaningful page structure:

`<div>, <header>, <footer>, <nav>, <article>, <section>, <aside>, <figure>, <figcaption>`

inline-level elements

Inline-level elements occur in the flow of text and do not cause line breaks by default. Inline-level elements can only contain other inline-level elements and text.

An inline-level element can appear as a child of a block-level element or another inline-level element. However, an inline-level element cannot be a *direct child* of the <body> element. Inline-level elements can appear *within* sentences and do not have to appear on a new line of their own. Most text elements are inline-level elements (spans of characters within the flow of text). Inline-level elements by default do not add line breaks or margins.

Phrase elements leave the specific rendering of the element to style sheets, either the author's or the browser's default rendering:

```
<abbr>, <acronym>, <cite>, <code>, <dfn>, <em>, <kbd>, <samp>, <strong>, <var>
```

The other inline elements HTML are mainly concerned with adding deeper meaning to inline text:

```
<b>, <i>, <u>, <sup>, <sub>, <big>, <small>, <li>, <ins>, <del>
```

Additionally, the following inline elements allow you to add functionality or :

linking pages together `<a>`
images `<img>`
audio `<audio>`
bitmap drawing `<canvas>`
inline content divisions `<span>`

about *deprecated* <body> attributes...

The HTML 3.0 Recommendation added a number of presentational attributes for the `body` element that had been introduced by browser developers and were in common use. At the time, they were the only way to do things like set the color for all the links and text in the document or add a background color or image to the page. A single `body` opening tag may contain a number of specific attributes, as shown here:

```
<body bgcolor="#CCFF00" text="#CC33CC">
```

Today, CSS is the correct way to handle matters of presentation, so all of the presentational attributes for the `body` elements are officially deprecated and are discouraged from use. The following achieves the same result as above but uses the `style` attribute:

```
<body style="background-color:#CCFF00; color:#CC33CC;">
```

The deprecated align attribute indicates whether a heading or paragraph element appears to the left, center, or right of the page (the default is the left):

```
<h1 align="center">Center Heading</h1>
<h2 align="right">Right Subheading</h2>
```

The `align` attribute has been replaced with the text-align property in CSS and the ability to float block level elements:

```
<h1 style="text-align: center;"></h1>
<h2 style="float: right;"></h2>
```

block-level elements

The distinct parts that make up a document are known as *block-level elements*, and are defined by block element tags. Block-level elements appear in the browser as if they have a margin or line break before and after them. The <p>, <h1>, <h2>, <h3>, <h4>, <h5>, <h6>, <ul>, <ol>, <dl>, <li>, <pre>, <hr />, <table>, and <blockquote> elements are all block level elements. They all start on their own new line, and any block-level item that follows them appears on its own new line, too. A block-level element cannot appear *within* an inline element.

Basic Text Formatting Elements

There are many elements that help you mark up your text, adding structural information to your documents. Keep in mind that you will not really be learning how to control the appearance (typefaces, colors, and font sizes) of text at this point. We will use style sheets for that purpose.

Remember that while one browser might display each of these elements in a certain way, another browser could display very different results. In particular, font sizes (and therefore the amount of space a section of text takes up) will change between browsers, as will the typefaces used. You don't know how wide the browser's window will be, so there is no guarantee that the lines will break the way they do on your machine.

If you want people to read what you have written, then structuring your text well is even more important on the Web than when writing for print. People have trouble reading wide, long, paragraphs of text on Web sites unless they are broken up well.

Here's a closer look at some of the elements commonly used to structure text on the Web:

<h1> <h2> <h3> <h4> <h5> <h6>

No matter what sort of document you are creating, most documents have *headings* in some form or other. Not only are page headings necessary for just about any web page, but when marked up properly, they can be powerful both to the design and accessibility of a site. Visually, a page heading is commonly treated with a larger font size and maybe a different color or typeface than the normal flow of body text.

Headings can help structure a document. If you look at the table of contents for a book, you can see how different levels of headings have been arranged to add structure to the book, with subheadings under the main headings. Even without any styling at all, a heading tag is obviously a heading: visual browsers will render headings in a larger, bold font, and an unstyled view of the page will show the document structure as it was intended, with the proper heading tag conveying *meaning* rather than just presentational instructions. Screen readers, PDAs, phones, and other visual and nonvisual browsers will also know what to do with a heading tag, handling it correctly and treating it with importance over normal text on the page.

In XHTML you have six levels of headings, which use the elements <h1>, <h2>, <h3>, <h4>, <h5>, and <h6>. While browsers can display headings differently, they tend to display the <h1> element as the largest of the six and <h6>= as the smallest, although CSS can be used to override the size and style of any of the elements.

Search engines love heading tags. A plain paragraph that is bold, on the other hand, means less to them. Properly marking up your headings with <h1> through <h6> tags takes little effort, yet can make it easier for search engines to index your pages, and ultimately for people to find them as well. Search engine robots place special importance on heading tags—a place where you're likely to have some keywords. Just as they index <title> and <meta>, they'll set their sights next on any heading tags you may have down the page. If you don't use them, those keywords contained within them won't be as valuable and could get overlooked. By using them correctly, you'll be increasing the likelihood of someone finding your site based on its *content*.

Paragraphs are the most basic elements of a text document. Paragraph elements are unique in that they may contain text and inline elements, but they may *not* contain other block elements, including other paragraphs. Each paragraph of text should go in between an opening and closing tag.

Because paragraphs are block elements, they always start a new line. Most browsers also add margins above and below block elements. Text is formatted flush-left, ragged right for left-to-right reading languages (and flush-right for right-to-left reading languages). As always, style sheets may be used to override any default browser rendering.

```
<p>Here is a paragraph of text.</p>
<p>Here is a second paragraph of text.</p>
<p style="text-align: center;">Here is a third paragraph of text, aligned center.</p>
```


Whenever you use the <br /> element, anything following it starts on the next line like a line-break. It's an example of an *empty* element, where you do not need opening and closing tags, as there is nothing to go in between them.

Try not to overuse line breaks. Remember, when you resize the browser window, your text is reformatted to fit. If you try to perfect your paragraphs with line breaks, you'll just end up with pages that look bizarre at different sizes. A good rule of thumb is to avoid line breaks in ordinary paragraphs. Instead, use them the to force breaks in addresses, outlines, poems, and other types of text whose spacing you want to tightly control. Don't use them for bulleted and numbered lists.

<pre>

Preformatted Text: Sometimes you want your text to follow the exact format of how it is written in the XHTML document. You don't want the text to wrap onto the next line to fit the browser window's width; you want consecutive spaces preserved and line breaks where you wrote them. Any text between the opening <pre> tag and the closing </pre> tag will preserve the formatting of the source document. Most browsers will display this text in a monospaced font by default, typically Courier. A common use of the <pre> element is to represent computer source code.

<blockquote>

When you want to quote a passage from another source, you should use the <blockquote>= element. Text inside a <blockquote> element is usually indented from the left and right edges of the surrounding text, and sometimes uses an italicized font. While <blockquote> is designed for use when quoting sections from another work, some designers use it for the effect on text, although it is possible to achieve the same look with CSS.

By using CSS to set up margins, padding, or positioning, it is possible to create indented text. If what you want to do is indent text, use CSS rules. Do not use a blockquote element merely to indent text. The <blockquote> element should only be used for its logical purpose, namely to quote a block of material.

Many designers have used <blockquote> in the past because it was like a paragraph that was indented. If you needed to indent a block or text, you wrapped it in <blockquote> and that was that. Unfortunately, it's a bad habit to get into, and one that's remedied by instead applying padding-left or margin-left values to the proper elements using CSS. Historically, <blockquote> has been abused in this way, being exploited more for presentational reasons, rather than for structural circumstances.

<hr />

The <hr /> element creates a horizontal rule across the page. It is an empty element, like the <br /> element. This is frequently used to separate distinct sections of a page where a new heading is not appropriate.

special characters

There are many symbols, marks, and characters needed in writing that are not among the letters and numbers on a keyboard. These are called *character entities* and *special characters*. A code number is assigned to each of these symbols or characters. Some characters have special meaning in XHTML, and for some characters there is not an equivalent on the keyboard you can enter. For example, you cannot use the less-than or greater-than brackets that start and end tags, as the browser can mistake the following letters for markup.

You can, however, use character entities to represent these special characters. All special characters can be added into a document using the numeric code for that character, and some also have alpha code, as you can see in the table below:

Sample	Character	Numeric Code	Alpha Code
–	En dash	&#8211;	&ndash;
—	Em dash	&#8212;	&mdash;
“	Left or opening double quote	&#8220;	&ldquo;
”	Right or closing double quote	&#8221;	&rdquo;
<	Less than or opening angle bracket	&#60;	&lt;
>	Greater than or opening angle bracket	&#62;	&gt;
&	Ampersand	&#38;	&amp;
¢	Cent	&#162;	&cent;
£	Pound	&#163;	&pound;
™	Trademark	&#8482;	&trade;
©	Copyright	&#169;	&copy;

Using the standard desktop publishing keyboard commands (such as Option-G for the © symbol on the Mac) within an XHTML document will not produce the desired character when the document is rendered in a browser. Other character entities are abbreviated names for characters, such as `<` for the less-than symbol, or ` ` to create a non-breaking space.

An em dash is a dash the width of the character M for the font size in use; it's usually used to set off a phrase within a sentence. There is a slightly shorter dash called an en dash. An en dash is the width of the character N for the font size in use and is used to join words within a phrase, as in “the Canada–United States border.” An en or em dash for a font like Arial might be relatively shorter than an en or em dash for a font like Verdana, because Arial is a rather narrow font while Verdana is a rather wide font.

creating lists

  <dl>

There are three kinds of lists that can be defined with XHTML. Each list type has it's own tag that indicates the beginning and end of the whole list, and each item in the list is surrounded by the *list item element* :

ordered, numbered lists
unordered, bulleted lists
definition lists for terms and definitions <dl> <dt> <dd>

Any list can be nested within another list. For example, you could add a bulleted list item under an item within a numbered list:

```
<ol>
<li>What's generally refered to as AJAX is a combination of these technologies:</li>
        <ul>
        <li>XHTML</li>
        <li>CSS</li>
        <li>The DOM: Document Object Model</li>
        <li>XML</li>
        </ul>
<li>Your site should contain three types of text files:</li>
        <ul>
        <li>XHTML</li>
        <li>CSS</li>
        <li>Javascript</li>
        </ul>
</ol>
```

The `type` attribute on the list element allows you to change the ordering of list items from the default of numbers by giving the type attribute the corresponding character. If you want to specify the number that a numbered list should start at, you can use the *start* attribute on the <ol> element. The value of this attribute should be the numeric representation of that point in the list, so a "D" in a list that is ordered with capital letters would be represented by the value "4".

```
<ol type="i" start="4">
<li>Point number one</li>
<li>Point number two</li>
<li>Point number three</li>
</ol>
```

The `type` attribute was deprecated in HTML 4.1 in favor of the CSS `list-style-type` property; the `start` attribute was deprecated in HTML 4.1 in favor of the CSS `marker-offset` property.

grouping elements

The and elements allow you to group together several elements to create sections or subsections of a page. On their own, they will not affect the appearance of a page, but they are commonly used with CSS to allow you to attach a style to a section of a page. For example, you might want to put all of the footnotes on a page within a element to indicate that all of the elements within that element relate to the footnotes. You might then attach a style to this element so that they appear using a special set of style rules.

The element is used to group *block-level elements* together:

```
<div class="footnotes">
<h2>Footnotes</h2>
<p><strong>1</strong> The World Wide Web was invented by Tim Berners-Lee.</p>
<p><strong>2</strong> The W3C is the World Wide Web Consortium who maintain many Web standards.</p>
</div>
```

The element, on the other hand, can be used to group *inline-level elements* ***only***. So, if you had a part of a sentence or paragraph you wanted to group together you could use the element:

```
<div class="footnotes">
<h2>Footnotes</h2>
<p><span class="inventor"><strong>1</strong> The World Wide Web was invented by Tim Berners Lee</span></p>
<p><strong>2</strong> The W3C is the World Wide Web Consortium who maintain many Web standards.</p>
</div>
```

On its own, this would have no effect at all on how the document looks visually, but it does add extra meaning to the markup, which now groups together the related elements. This grouping could either be used by a processing application, or could be used to attach special styles to these elements using CSS rules.

inline-level elements

Once text is in place, it's customary to add inline-level elements to format and add more meaning to the content. These are text-identification tools such as acronyms, emphasis, and citations. This is accomplished through the use of *logical elements* or *physical elements*.

Logical elements, also known as *phrase* elements, are designed to describe what their content *is*. Physical elements, also known as *presentational* elements, simply define what the content should *look like*.

Many physical elements are considered obsolete in HTML, such as <blink>, which was used to flash text on and off, while others are considered depreciated: <u> underline and <s> strikethrough have CSS equivalents using the `text-decoration` property (`text-decoration: underline;` and `text-decoration: line-through;`)

using logical inline elements

While some logical elements are visually rendered in a similar manner to physical elements, such as <b> and= both displaying bold text or <i> and= = displaying italicized text, you should preferably get into the habit of using them where appropriate for several reasons.

Applications such as screen readers (which can read pages to Web users with visual impairments) could add suitable intonation to their voice so that users with visual impairments could hear where the emphasis should be placed. A screen reader will emphasize the text surrounded by tags, but not text surrounded by the <i> tag, although both will render the same in a visual browser.

Automated programs could be written to find the words with emphasis and pull them out as keywords within a document, or specifically index those words so that a user could find important terms in a document.

The list of logical elements as outlined by the W3C includes the following:

 Used to stress emphasis
 Used to indicate strong importance
<i> text in an alternate voice
<b> used for stylistically offset text, such as keywords or dropcaps
<cite> Contains a citation or a reference to other sources
<dfn> Indicates that this is the defining instance of the enclosed term
<code> Designates a fragment of computer code
<samp> Designates sample output from programs, scripts, etc.
<kbd> Indicates text to be entered by the user
<var> Indicates an instance of a variable or program argument
<abbr> Indicates an abbreviated form (e.g., WWW, HTTP, URI, Mass., etc.)
<acronym> Indicates an acronym (e.g., WAC, radar, etc.)
<address> Used to insert a mailing address

Here's a closer look at these phrase elements and how they are used:

acronyms & abbreviations

<acronym> <abbr>

Using the <abbr> (for abbreviations) and <acronym> (for acronyms) elements can improve the accessibility of web pages by giving definitions to abbreviations and acronyms, so that all users are informed. It's considered adequate to identify the acronym with an acronym element including a title attribute the first time it's used, and the acronym can be used after that without being defined with the acronym element in the HTML.

The <acronym> element includes a title attribute that gives the actual meaning of the words that create the acronym. Some browsers may display the acronym in italics, while others may not show any visual clue that the <acronym> element is there. All browsers should display the information in the title attribute as a tool tip if the user's cursor is held over the acronym.

The W3C's specification on what the <abbr> and <acronym> elements are used for:

> *<abbr> Indicates an abbreviated form (e.g., WWW, HTTP, URI, Mass., etc.)*
> *<acronym> Indicates an acronym (e.g., WAC, radar, etc.)*

Using these elements along with a suitable title attribute will help users who would be otherwise unfamiliar with the term. For instance, if we were marking up the abbreviation "HTML", we could use the <abbr> tag like this:

```
<abbr title="HyperText Markup Language">HTML</abbr>
```

Using <abbr> is this case can provide a cue for screen readers in regards to spelling out the text (X-H-T-M-L), rather than reading it as a normal word. Conversely, the use of <acronym> provides a cue to speak the word normally, rather than spell it out. An example of the <acronym> tag could be applied to the following:

```
<acronym title="North Atlantic Treaty Organization">NATO</acronym>
```

There are also two CSS rules that could be added to an aural style sheet to further reinforce these directives:

```
abbr {speak: spell-out;}
acronym {speak: normal;}
```

Aural style sheets allow authors to construct CSS rules specifically for screen reader applications. Harnessing structural markup, changes in pitch, voice-type, inflection, etc. can be altered to present the page aurally, more in line with what it reads like, visually.

```
<abbr title="Cascading Style Sheet">CSS</abbr>
<acronym title="National Public Radio">NPR</acronym>

abbr {
border-bottom: 1px dotted #000000;
background-color: yellow;
}
```

citations & definitions

<cite>

<cite> is used to reference a citation of a source: an author or publication. The <cite> element is intended to be used for citations such as book and magazine names. Sometimes cite elements contain a reference to another source, with a `<cite>` *attribute* giving the location of the original document. Most browsers render the <cite> element in italics, but keep in mind the accessibility requirement for logical formatting. None of the other ways to create italic text for a book or magazine title carry the logical element characteristics inherent in the cite element, which is expressly intended to be used for citations.

You may suggest marking up a publication's title with instead—but when referencing a book or other publication, we're not intending to add emphasis; we're merely trying to set it apart from normal text. We're also trying to stay in line with conventional typography practices, where titles are often shown in italics (underlining is also common in the print world, but would create obvious confusion for a hyperlink).

Here enters the <cite> tag, specifically created for the job. Most browsers will even render text contained within <cite> tags in italics by default, and we can support that by adding a general CSS declaration that will do the same.

the <cite> specification

The W3C is somewhat brief regarding the <cite> tag:

> *<cite> Contains a citation or a reference to other sources.*

It's unclear exactly what types of data we can wrap <cite> around, but by "sources," we can take to mean at least people and publications. Let's take a look at <cite> in use:

```
The novel, <cite>The Scarlet Letter</cite> is set in Puritan Boston
and like this book, was written in Salem, Massachusetts.
```

From the use of <cite>, the title The Scarlet Letter will, in most browsers, be rendered in italics. To be sure, we'll add the following, utterly simplistic, CSS rule—for cases where the browser doesn't:

```
cite {font-style: italic;}
```

<dfn>

The <dfn> element allows you to specify that you are introducing a special term. Its use is similar to the words that are in italics in the midst of paragraphs in this book when new key concepts are introduced.

Typically, you would use the <dfn> element the first time you introduce a key term and only in that instance. Most recent browsers render the content of a <dfn> element in an italic font. For example, you could indicate that the term "HTML" in the following sentence is important and should be marked as such:

```
<p>This class teaches you how mark up your documents for the web using
<dfn>HTML</dfn>.</p>
```

emphasis

The content of an element is intended to be a way to *stress* ***emphasis*** in your document, and it is usually displayed in italicized text. The kind of emphasis intended is on words such as “must” in the following sentence, or something that would be read aloud differently:

```
<p>You <em>must</em> remember to close elements in HTML.</p>
```

You should use this element only when you are trying to stress emphasis, not just because you want to make the text appear italicized. If you just want italic text for stylistic reasons—without adding emphasis—you can use CSS.

The element is intended to show *strong* ***importance*** for its content. As with the element, the element should be used only when you want to add strong importance to part of a document. Rather than being rendered in an italic font, most visual browsers display the strong emphasis in a bold font.

```
<p><em>Never</em> stare at the sun directly as it <strong>can cause blindness</strong>.</p>
```

Remember that how the elements are presented (italics or bold) is largely irrelevant. You should use these elements to add emphasis to phrases, and therefore give your documents greater meaning, rather than to control how they appear visually. It’s quite simple with CSS to change the visual presentation of these elements.

programming-oriented content

Each of these text formatting elements has a purpose, although their display results are usually the same. For example, <kbd> is used for formatting keyboard instructions. If you need to display computer code, use <code>. These logical elements will display their contents, by default, in monospaced fonts.

<code>

The <code> element is designed for demonstrating code examples within HTML pages. For instance, if you’d like to share a CSS example, you could do something like this:

```
<code>#content {width: 80%; padding: 20px; background: blue;}</code>
```

Generally, visual browsers will render text held within <code> tags in a monospaced serif font, but we could, of course, style code examples any way we wish by adding a CSS rule such as this which renders all text contained in <code> with the Courier typeface in red:

```
code {
font-family: Courier, serif;
color: red;
}
```

<kbd>

The <kbd> element is used to signify text to be entered by the user. For example, if I were explaining how someone might use the accesskey we had assigned to switch focus to a search box, I might say

```
<p>To quickly change focus to the search input field, Mac users type
<kbd>Command+9</kbd>.</p>
```

Through a simple CSS rule, you can customize the style of all <kbd> elements, just as we had previously with the other phrase elements.

<samp>

The <samp> element is used to show sample output from programs and scripts. It's mainly used when documenting programming concepts. For example, the results of a Perl script may say something like:

```
<p>When the script has executed, at the command line you will see the
message <samp>script was successful!</samp>.</p>
```

This is essentially "quoting" the output of a script, and a similar CSS rule could be defined for styling program samples uniquely—just as with <code> elements.

<var>

Related to the <samp> element, the <var> element is used to designate a program parameter or variable. For instance, if I were talking about an XSLT style sheet, I could code the following:

```
<p>I'm going to pass the parameter <var>lastUpdated</var> to my main.xsl file.</p>
```

Many browsers will render text within <var> tags in italics—but feel free to write a simple rule that would override that. If we don't like italics, we could use the font-style property in CSS:

```
var {
font-style: normal;
font-family: Courier, serif;
color: purple;
}
```

simulating tracking features

For simulating the tracking features, if not the functionality, of word processors:

<del> ~~deleted text~~
<ins> inserted text

address

<address>

Many documents need to contain a mailing address, and there is a special <address> element that is used to contain addresses. The address may also contain other contact information. For example, here is the address for Bryant & Stratton College, inside an <address> element, which is itself in a <p> element:

```
<p><address>Bryant & Stratton College, 1500 Jefferson Road, Rochester, NY 14614</
address></p>
```

A browser can display the address differently than the surrounding document, such as Internet Explorer displays it in italics. Indicating who wrote a document or who is responsible for it adds credibility to a document that is otherwise anonymous. The address element is a good way to add this at the end of the document. It can also help automated applications read addresses from documents.

superscript & subscript

<sub> <sup>

In order to create street names or numbers like 1st, 2nd, and 3rd, use the sup (for superscript) element. The code would look like:

```
<p>I live at 1<sup>st</sup> and York.</p>
```

For chemical formulas such as H2O, use the sub (for subscript) element:

```
<p>H<sub>2</sub>O.</p>
```

and then there's these guys...

<b> <i>

In the old days, <b> and <i> were font style elements used to set text to bold and italic. While using them will still give this effect, it is the incorrect reason to use them in this way today (use CSS instead). In HTML5, they have been given new semanic roles which you should be aware of:

The <b> element represents a span of text to which attention is being drawn for utilitarian purposes without conveying any extra importance and with no implication of an alternate voice or mood, such as key words in a document abstract, product names in a review, actionable words in interactive text-driven software, or an article lede.

The <i> element represents a span of text in an alternate voice or mood, or otherwise offset from the normal prose in a manner indicating a different quality of text, such as a taxonomic designation, a technical term, an idiomatic phrase from another language, a thought, or a ship name in Western texts.

hyperlinks

A typical Web site is composed of sets of multiple files and folders, organized and connected with each other through elements called *links*. You write links to allow navigation among all the files within a given site, to web pages outside the site, and to resources beyond just HTML pages: e-mail links, links to sound clips, and links to PDF documents. Web pages can also contain links to specific parts of a given page.

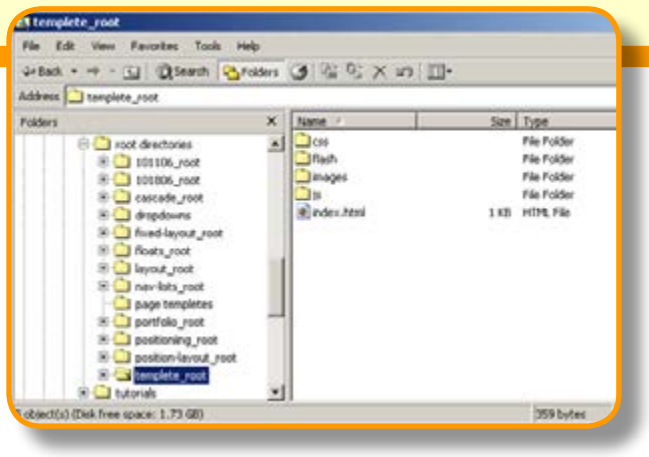
Known as *hyperlinks*, these links allow visitors to navigate by clicking on words, phrases, and images.The average Web site is a group of pages users navigate between using hypertext links. These pages often include links to other Web sites as well as to other pages in the same site. Structuring your site into folders, or *directories,* is very important at this stage so that you can use *relative URLs* to link between the pages within your site.

organizing and managing files

When beginning work on a new site, create a new folder (directory) on your computer where you will save every document that will be transferred to the server when the site is complete. This folder is known as the *root directory*. This includes all the files that will eventually be transferred to a server as part of the site. You can add as many subfolders (subdirectories) to this folder as you need, with separate folders in each section for assets such as images, sounds, scripts, and style sheets as long as all of the files are kept within the root directory.

The organization that you set up on your own computer for a website must be created in *exactly* the same way when you put your files on the server. That is because of the way links work: a link is like a path or a map to a file's location. If the file isn't in the exact location that you write in the link, then the browser will return a "File Not Found" error.

Being well organized in the beginning is much easier than going back after you've started working. Relative links you already have built into your site will not work properly if you move files or reorganize the folder structure. If you move files and folders after a site is already on a server, it will change URLs within your site.



URLs (Uniform Resource Locator)

A URL is used to locate a resource on the Internet. Every file on the Internet has a unique URL, the address that can be used to find that particular file. A URL is a global address that can be used to locate anything on the Web, including HTML pages, audio, video, and many other forms of Web content, and no two files on the Internet share the same URL. Most commonly used URLs incorporated in Web pages are `http:`, `ftp:`, and `mailto:`.

The first part of the URL tells you the *protocol* that needs to be used to retrieve the resource, such as `http://`. HTTP is also known as the *HyperText Transfer Protocol*. It's an agreed-upon method (a protocol) for transferring hypertext documents around the Web. While "hypertext documents" are usually just HTML pages, the protocol can also be used to transfer images, or any other file that a Web page might need.

HTTP is a simple request and response protocol. Each time you type an URL into your browser's address bar, the browser asks the server for the corresponding resource using the HTTP protocol. If the server finds the resource, it returns it to the browser and the browser displays it. If the resource can't be found, the servers reports a "404 Error " back to the browser.

The Web site part, such as "`www.bryantstratton.edu/`", consists of the server name and the domain name. It tells the browser which computer on the Internet to get the resource from.

the anchor tag

<a>

A link is specified using the inline <a> element. Anything between the opening <a> tag and the closing tag becomes part of the link a user can click in a browser. The opening <a> tag must carry the `href` attribute, whose value is the page you are linking to. This example contains a link to a second page called "`index.html`":

```
<p>Return to the <a href="index.html">index page</a>.</p>
```

As long as index.html is in the *same folder* as this document, when you click the words "index page," the index.html page will be loaded into the same window, replacing the current page. In the above example, the content of the <a> element forms the link.

absolute and relative URLs

Absolute URLs provide the full URL for the document, including the protocol, domain name, and pathname as necessary. If you want to link to a file outside your own site folder, known as an "external" link, you must write an absolute link, just as you type the absolute URL for that page into the address bar in your browser to visit it. An absolute URL contains everything you need to uniquely identify a particular file on the Internet.

```
<p>Find out more about Web Standards at the <a href="http://www.w3.org/">World Wide Web Consortium's</a> Web site.</p>
```

When writing an absolute link in HTML that ends with a server or folder name, include the trailing slash: `http://www.example.com/`.

Relative URLs indicate where the resource is in *relation to the current page*. Because absolute URLs can get so long, the relative URL provides convenient shorthand for URLs that point to files *within* your Web site, which is why they are known as "internal" links; they call for a page on your own server. Behind the scenes, the browser creates an absolute path out of that relative path and the path of the page that you click on so all the Web server ever sees are absolute paths. The following link points to a file called 'newstuff.html', which is located in the directory 'gallery':

```
<p>New acquisitions can be found in our <a href="updates.html">New Works Gallery </a>.</p>
```

A benefit to using relative URLs within your site is that you can change your domain name or copy a subsection of one site to a new site without having to change all of the links because each link is relative to other pages within the same site. Keep in mind that relative URLs work *only* on links within the same directory structure; you cannot use them to link to pages on other servers.

If the files aren't in the same directory, you have to give the browser directions by including the *pathname* in the URL. When linking to a file in a lower directory, the pathname must contain the names of the subdirectories you go through to get to the file:

```
<a href="products/records/singles.html">Singles & EP's</a>
```

To go in the other direction and link to upper directories, begin a pathname with a "dot-dot-slash" (`../`) When you begin a pathname with a `../` construction, it's the same as telling the browser to "back up one level" and then follow the path to the specified file. Each `../` at the beginning of the pathname tells the browser to go up one directory to look for the file, and you can use it as many times as you need to move through the directory structure:

```
<a href="../../index.html">[back to the home page]</a>
```

targets within a page

If you have a long Web page, you might want to link to a specific part of that page. You will usually want to do this when the page does not fit in the browser window and the user might otherwise have to scroll to find the relevant part of the page. This is accomplished using the `id` attribute to create a *destination anchor* as the target of the link. The destination anchor allows the page author to mark specific points in a page that a source link can jump to. Common examples of linking to a specific part of a page that you might have seen used on Web pages include:

- "Back to top" links at the bottom of long pages
- A list of contents for a page that takes the user to the relevant section

Linking to targets on a page is a two-part process. First, you need add the destination anchor. The browser moves to whatever location the target anchor specifies, so make sure that you place your anchor in the correct spot on the page:

```
<a id="summary">summary</a>
```

Next, you place a link to it somewhere else on the page:

```
<a href="#summary">Skip to the Summary…</a>
```

The value of the `href` attribute in the source anchor is the value of the `id` attribute preceded by a pound or hash sign (`#`). You can use any combination of upper and lowercase characters in your `id` attributes. Always start your `id` with a letter and follow it with any letter, digit, hyphen, or underscore. You can't use spaces, so you can't have a name like "More Info", but you can have "More-Info", "More_ Info", "MoreInfo", and so on. Just make sure you are consistent and always use the same upper and lowercase letters in your `hrefs` and destination anchor `id` (which is why it is often easier to make these names entirely lowercase every time). If you aren't consistent, your links may not work correctly on every browser.

A destination anchor should always have some content, otherwise some browsers will not find the destination. Also, if you enable users to jump down the page, you should also enable them to jump back up. Don't forget to add links at the bottom of the page to return the visitor to the top of the page.

email links

The `mailto` source attribute contains a value command that launches the user's email client when the user clicks on the email link, ready to send an email to that address. Some even contain subject lines. To create a link to an email address, use the following syntax with the <a> element:

```
<a href="mailto:email@domain.com">Email us!</a>
```

As with any link, the content of the <a> element is the visible part of the link displayed in the browser, so you can put whatever you like in there. There is one drawback to using this technique: some less scrupulous inhabitants of the Web use little programs to automatically search Web sites for e-mail addresses. After they have found e-mail addresses on Web sites they will start sending spam (junk mail) to those addresses.

There are a few alternatives to creating a link to an e-mail address:

- Use an email form instead so that visitors fill in a form on your Web site to send you an email. Once you have received the mail, you can then reply as normal because automated programs do not use contact forms to collect email addresses. Use of an email form requires either a CGI script or a server side scripting language such as ASP, JSP, or PHP.

- Write your e-mail address into the page using JavaScript. The idea behind this technique is that the programs that scour the Web for email addresses cannot read the JavaScript version of an address.

adding hyperlinks to images

When creating an <a> element, you can insert more than text into the clickable area; you can also add images. For example, instead of using the word “Next”, you might insert a navigational graphic such as an arrow. It’s also usually a good idea to include some explanatory text or add a text link under the image so that the purpose of the image will be absolutely clear. An image can be used with either relative or absolute links.

```
<a href="next.html" title="Go to the Next Page"><img src="next.gif" alt="Next Page" /></a>
```

Whether you’re using text or image links, it’s a good idea to make your links concise and to make the content of the <a> element actually describe what is at the other end of the link. Many people scan pages for links when they want to go to the next page without really reading the document. The content of the <a> element clearly sticks out more than the text around it (usually because it’s presented in a different color). Users are less likely to stay on your site and follow your links if all of them just say “click here” because the link will not show them clearly and quickly where they are going.

the title attribute

It is also good practice to use the `title` attribute on a link, as this will be displayed in a tooltip (a simple popup that appears stating the title) in most visual browsers when the user hovers over the link. This can also help the visually impaired if they use a voice browser via an auditory clue. A `title` attribute is vital for any links that are images. The *value* of the `title` attribute should be a description of what the link will take you to. For example, here is a link to the Google home page:

```
<p><a href="http://www.Google.com/" title="Search the Web with Google">Google</a> is a
very popular search engine.</p>
```

other link tips...

Keep link labels concise. Don’t make entire sentences or large pieces of text into links. In general, keep them to a few words. Provide additional information in the `title` attribute.

Keep link labels meaningful. Avoid using link labels like “click here” or “this page”. Remember that users tend to scan pages for links first, and then read pages second. So, providing meaningful links improves the usability of your page. Test your page by reading just the links on it; do they make sense? Or do you need to read the text around them?

Be careful when placing links directly next to each other; users may have trouble distinguishing between links that are placed too closely together.

images

There are only three image formats predominantly used in web pages: *GIF*, *JPEG*, and *PNG*. The JPEG is often used for photos or images with thousands of colors. The GIF is often used for images with fewer colors or when a transparent background is needed. No matter which image type you use, it's important to optimize the image to its smallest possible file size so that it downloads as quickly as possible. It's also important, in most cases, to create the image in the size you actually intend to display on the web page.

The <img> element is an *empty element*, and an empty element is one that doesn't include any text to be displayed on screen, so you don't need a closing tag to close the <img> element.

The <img> element has several must-have attributes. The first is `src`, for source. A `src` attribute is a lot like an `href` attribute because it gives the path to the location of the image. Like `href`, the path used for the `src` attribute can be relative or absolute.

```
<img src="images/creek_sm.jpg">
```

It's good practice to create a separate directory (folder) in your site for images. If you have a very large site, with separate directories for each section, you can create an image folder for each section of the site. Generally speaking, images for your site should always reside on your server. If you link to images on other sites and the owner of the other site decides to move that image your users will no longer be able to see the image on your site.

the width, height attributes

The <img> element with a src attribute will display an image in the browser, but it's not all you need. You also need the width and height of the source image. Although *width* and *height* attributes are not strictly required, it helps the browser render the page better. Specifying the size of the image can help browsers lay out pages faster because they can allocate the correct amount of space to the image and continue to render the rest of the page before the image has finished loading. With the width and height added, the <img> element looks like this:

```
<img src="images/creek_sm.jpg" width="200" height="109">
```

You will need to know the size of each graphic in pixels. If you have software such as Photoshop or Fireworks, you can open an image in that software to get the image size. If you don't have any graphics software, open the image in the Netscape, Mozilla, Firefox, or Safari browser and write down the image width and height shown the browser title bar.

If you want to scale an image, you can just provide the value for one of the attributes and the browser will maintain the correct ratio for the image, such as its width compared to the height. You can also distort images by providing a different width in relation to its height.

If you want to display the image a lot smaller than the original version rather than just specifying the smaller dimensions for the same image, you should resize the image in an image processing program to create the smaller version for use on the Web site. If you just reduce the size of the image using the height and width attributes the user will still have to download the full sized image, which will take longer than a special small version and use up more bandwidth.

the alt attribute

The final important requirement is the `alt` attribute, for alternative text. This is very important to ensure accessible information. The `alt` text describes the image. If the user is unable to see the image, the `alt` text will display, and the use and purpose of the image is explained by the `alt` text. The value of this attribute should really describe the image for users who cannot see it—either because the browser did not download the file correctly because the file cannot be found, or because the user has visual impairment that prevents him or her from seeing the image. The `alt` text should not just be the same as the filename.

Internet Explorer 6 on Windows displays the `alt` value as a tool tip when you mouse over an image. This is not what `alt` text is supposed to do, and it doesn't appear as a tool tip in other browsers. (that it what the `title` attribute is for). `Alt` text is supposed to be visible when the image is *not* visible—that is, it is an alternative to the image. Suitable `alt` text might be something like "a color photograph of tree-lined creek."

```
<img src="images/creek_sm.jpg" width="200" height="109" alt="A color photograph of a tree-lined creek">
```

If the <img> element is used as a button, the `alt` text should match the button text. In other words, if the button says "Home", the `alt` text should say "Home", too. If your image is just used for layout and is not strictly visible (for example a block of color used to help position another element rather than something the user is supposed to see), then the `alt` attribute should still be used but given no value: `alt=" "`.
--

other attributes: longdesc, title

Complex images such as charts and graphs often need a `longdesc` attribute for long descriptions. If the image contains informational content that would make the page meaningless if the image were not seen, then `longdesc` is essential. The `longdesc` attribute points to a separate HTML file that gives a detailed description of the content of the image:

```
<img src="images/chart.jpg" height="400" width="400" alt="monthly sales figures" longdesc="sales.html">
```

The `title` attribute can also be used with <img> elements. All browsers show the text of a `title` attribute visibly to all users if the element is hovered over, and most browsers show it as a tool tip. The `title` attribute can include further description of an image or a link and often gives accessibility hints such as key combinations for access keys or tab index numbers or points to the location of long description (`longdesc`) files.

tables

A *table* is usually a collection of numbers and words arranged in *rows* and *columns* of *cells*. Most cells contain the data values; others contain row and column headers that describe the data. You can put nearly anything you might have within the body of an HTML document inside a table cell, including images, forms, rules, headings, and even another table (referred to as nesting). The browser treats each cell as a window unto itself, flowing the cell's content to fill the space, but with some special formatting provisions and extensions.

Creating tables in HTML can be somewhat complex considering how many different types of tables there are: a table can be a 3-by-3 grid with labels across the top, or two side-by-side cells, or a complex Excel spreadsheet that comprises many rows and columns of various sizes. Representing tables in HTML is heavy on tags, and the tags can be hard to keep track of once you get going.

creating basic tables

The basic approach with table creation is that you represent tabular data in a linear fashion, specifying what data goes in which table cells using the appropriate tags. In HTML, tables are created from left to right and top to bottom. You start by creating the *upper-left* cell, and finish with the *bottom-right* cell. Working from left to right and top to bottom, you define, in sequence, the header and data for each column cell across and down the table. The number of columns is automatically calculated based on how many cells there are in each row.

Most tables are created from five primary elements:

```
<table border="1">
<caption>Top 25 Songs of 1969</caption>
<tr>
<th>Track</th>
<th>Song</th>
<th>Artist</th>
<th>Album</th>
</tr>
<tr>
<td>1</td>
<td>Come Together</td>
<td>The Beatles</td>
<td>Abbey Road</td>
</tr>
</table>
```

a troubled past...

In versions of HTML prior to 1996, CSS was not available. Without CSS to lay out a page, designers took what they knew about publishing in print and applied that to web page design.

When laying out a magazine or newspaper page, the designer thinks in terms of a *grid*, an arrangement of columns and rows. Designers saw the table as the equivalent layout tool for web pages, and for several years tables were the layout tool of choice. There were tables nested within tables inside of more tables. Tables, tables everywhere.

About the same time that designers grew really masterful at creating complicated arrangements of myriad numbers of nested tables, it became apparent that nested table layouts were a serious barrier to users with accessibility needs because they were often rendered as senseless gibberish when not accessed visually. More and more people were using the Internet, and more and more people were finding barriers to accessing the Internet in the process.

...tables today

Recent advances in the use of CSS for page layout have had a major impact on the way tables are used on the Web. Today the recommended use for tables is for displaying tabular information. The table element is perfect for this task, since a table is basically a matrix of rows and columns: `http://icant.co.uk/csstablegallery/`

CSS can be used to determine table size, color, position, padding, border, alignment, and many other properties that contribute to the effectiveness and attractiveness of the display, including the border-collapse and border-spacing properties. Because CSS is taking over the job of laying out a page, people are moving away from using tables for layout.

Many designers will still design pages with simple and accessible tables-based page layouts. These layouts are refered to as *hybrid* or *transitional* designs. To a great extent, the decision as to whether a completely CSS-based layout or a hybrid design of CSS with simple tables for layout is best depends on your audience and your understanding of who will be seeking your information.

<table>

<table> encapsulates a table and its elements in the document's body content. The only content allowed within the <table> is one or more <tr> tags, which define each row of table contents, along with the various table sectioning tags: <thead>, <tfoot>, <tbody>, <col>, and <colgroup>. These tags allow you to define and control whole sections of tables, including adding running headers and footers.

the border attribute

The border attribute specifies if a border should be displayed around the table cells or not. The value "1" indicates borders should be displayed, and that the table is NOT being used for layout purposes. It can only have the value of 1 or empty (i.e. border=""). If you want to change the width of the border, you should use the `border-width` CSS property.

`<table border="">` or `<table border="1">`

<caption>

The <caption> element goes inside the <table> element just before the table rows, and it contains the title of the table, telling what the table is for. The <caption> element displays the text inside the tag as the table caption, usually centered above the table.

Although you could use a regular paragraph or a heading as a caption for your table, tools that process HTML files can extract <caption> elements into a separate file, automatically number them, or treat them in special ways simply because they're captions. Captions are not a required element, but it's usually a good idea to use a caption wherever possible.

<th>

Header cells announce what type of information appears in a row or column. <th> elements are used for heading cells. Generally, browsers center the contents of a <th> cell and render any text in the cell in boldface. You can place the headings along the top by defining the <th> elements inside the first row, or put the headings along the left edge of the table instead by putting each <th> in the first cell in each row and follow it with the data that pertains to each heading.

the scope attribute

The `scope` attribute is used to associate <td> cells with the appropriate headers. The `scope` attribute tells a browser or screen reader that the contents of the data cells in each column are related to the heading at the top of the column.

If a cell is a row or column heading it should always be marked up as <th> instead of <td>. Table headings can be given a `scope` attribute value of row or col to define whether it's a row or column heading; `rowgroup` or `colgroup` can be used if they relate to more than one row or column. To tell data cells which header to connect to, you can use the `scope` attribute on the header, or an `id` on the header and a `headers` attribute on the cells (see the header attribute below).

```
<thead>
<tr>
<th id="playlistPosHead" scope="col">Playlist Position</th>
<th scope="col">Track Name</th>
<th scope="col">Artist</th>
<th scope="col">Album</th>
</tr>
</thead>
```

<tr> <td>

The actual data presented in a table is enclosed in <td> tags. <tr> defines a table row, and <td> elements are used for the data cells.

empty cells

CSS has the ability to hide table cells which contain no content. Table cells containing no content can occur due to the grid format of tables; sometimes there are more cells in a particular row than are required. Creating a style rule with the CSS property `empty-cells: hide` will cause the browser not to display the border and background of any cell that contains absolutely no content. Cells that contain any content at all, even if it's simply a   non-breaking space entity or <br /> line break, will still remain visible.

Note that empty cells that are hidden will continue to have a visual effect on the table layout due to their border-spacing values being preserved even though the cell itself is hidden.

the headers attribute

The `headers` attribute is used to associate specific data cells with specific headers. First an `id` is assigned to each <th> element. Next, the `headers` attribute is used on the <td> to associate specific data cells with specific headers.

adding accessiblity

Beyond these core elements, there are additional structural elements that allow assistive technology to decipher table information. Data tables can be frustrating and confusing for people using assisitive technologies such as screenreaders, and these elements, along with `summary` and <caption>, are intended to increase the accessibility of data tables for these devices; it's definitely good practice to use them where possible.

They also allow designers to define and control whole sections of tables, including adding running headers and footers. Each tag has one or more required and optional attributes, some of which affect not only the tag itself but also related tags.

<thead> <tbody> <tfoot>

These elements allow developers to break up tables into logical sections. For example, all of the column headings can be placed inside the <thead> element, providing a means of separately styling that particular area.

When using <thead> or <tfoot>, you must use at least one <tbody> element. Only one <thead> or <tfoot> element can be used per table, but multiple <tbody> elements can be used to break complex tables into manageable chunks. Also note that <tfoot> must always appear immediately following <thead> and before the first <tbody>. Browsers are very literal in their interpretation of <thead>, <tbody>, and <tfoot>. If you mark up the first row in this table as <tfoot>, the browser will display it at the foot of the table.

<col> <colgroup>

While the <tr> element allows developers to apply styles to whole rows, applying a style to an entire column is a bit more difficult. Although not many browsers currently support it, <colgroups> are a way of defining and grouping one or more columns.

tables & accessibility

The U.S. government added Section 508 to the Americans with Disabilities Act, which required certain accessibility features to be integrated with the website of any federal agency or federal contractor. The WAI and the Section 508 requirements cover a range of web-based technologies, including audible screen readers and Braille readers. They also address issues relating to site graphics, frames, animations, image maps, scripting languages, plug-ins, and forms.

Tables still present accessibility problems for users who try to gather the information in a table using an assistive browsing device. However, there are tools, including <caption>, <thead>, and <tfoot> that designers can use to ensure that the table is easily understood with any Internet-capable device.

Two excellent sources of accessibility information and training are WebAIM at `www.webaim.org` and Knowbility at `www.knowbility.org`.

styling tables

You can use many of the standard CSS properties to style a table and its contents. The `color` property, for example, sets a table's text color, just like anywhere else. There are few properties, however, that are particularly useful with tables, as well as a couple aimed specifically at formatting tables.

Because tables are composed of several HTML tags, it helps to know which tag to apply a particular CSS property to; applying padding to a <table> tag has no effect, for example.

padding

When it comes to tables, the borders are the edges of a cell, so padding adds space around any content you've placed inside of a table cell. It works a lot like the <table> tag's depreciated `cellpadding` attribute, with the added benefit that you can individually control space between a cell's content and each of its four edges.

You apply padding to either a <th> or a <td> element, but not to the <table> element itself. So, to add 10 pixels of space to the inside of all table cells, you'd use this style:

```
td, th { padding: 10px; }
```

If you place an image into a table cell using the <img> tag, and notice that there's unwanted space below the image, then set its `display` property to `block`.

rows & columns

Adding alternating column stripes is a common table design technique. By alternating the appearance of every other row of data, you make it easier for people to spot the data in each row. The solution is to apply a class (like `<tr class="odd">`) to every other row, and then create a style to format that row:

```
tr.odd { background-color: skyblue; }
```

You can also use background images to create more interesting looks like gradients or patterns.

vertical & horizontal text alignment

To control where content is positioned within a table cell, use the `text-align` and `vertical-align` properties. Text-align controls horizontal positioning, and can be set to `left`, `right`, `center`, and `justify`. It's an inherited property. When you want to right align the contents of all table cells, create a style like this:

```
table { text-align: right; }
```

This property is useful with <th> elements, since browsers usually center align them. The following style rule will make table headers align with table cells:

```
th { text-align: left; }
```

Table cells have a height as well. Web browsers normally align content vertically in the middle of a table cell. You can control this behavior using the `vertical-align` property. Use one of these four values: `top`, `baseline`, `middle`, or `bottom`. `top` pushes content to the top of the cell, `middle` centers content, and `bottom` pushes the bottom of the content to the bottom of the cell. `baseline` works just like `top`, except the browser aligns the baseline of the first line of text in each cell in a row. Unlike `text-align`, the `vertical-align` property isn't inherited, so you can use it only on styles that apply directly to <th> and <td> elements.

gone but not (yet) forgotten

There are several `<table>` attributes that are part of the HTML 4.01 and XHTML specification that have become obsolete in HTML5 due to the fact that there are CSS properties that can handle these things much better. But you'll still see tables accross the Web that still use these attributes due to their being in use for so long. They will still work of course, but the W3 validator will let you know that they have been depreciated if you are using an HTML5 doctype with your page.

So for the sake of completeness, here are some of the most common depreciated table attributes along with HTML5 alternatives:

the border attribute

Prior to the HTML5 spec, the CSS `border` property accepted any value from 0 upwards. Applying a border to a style that formats the <table> tag outlines just the table, not any of the individual cells. Applying borders to cells leaves you with a visual gap between cells. To control how borders appear, you would have needed to use the <table> element's `cellspacing` attribute along with the CSS `border-collapse` property.

Collapsing table borders: Even if you eliminate the cellspacing of a table, borders applied to table cells may double up where the bottom border of one cell adds to the top border of the cell below, creating a line that's twice as thick as the border setting. The best way to eliminate this (and eliminate cell-spacing at the same time) is to use the `border-collapse` property. It accepts two values: `separate` and `collapse`. The `separate` option is normally how tables are displayed, with the cell spaces and doubled borders. Collapsing a table's borders eliminates the gaps and doubled borders. The `collapse` value is applied to the <table> style: `table { border-collapse: collapse; }`

The appearance of a collapsed border is determined by comparing the visibility, width, style, and color of the borders that are to be collapsed:

Visibility Where one of the borders to be collapsed has a `border-style` value of `hidden`, that value takes precedence, so the collapsed border at that location will be hidden.	*Width* Where two visible borders with different `border-width` values are to be collapsed, the highest value takes precendence and the collapsed border will be the greater width.
Style Where two visible borders of equal width are to be collapsed their `border-style` value sets the precedence in the decending status order of: `double`, `solid`, `dashed`, `dotted`, `ridge`, `outset`, groove, `inset` – in that order. The collapsed border at that location will be in the style of the highest status.	*Color* Where two visible borders of equal width and identical style are to be collapsed the `border-color` value is determined in the decending status order of cell, row, row group, column, column group, table –so the collapsed border will be in the color of highest status.

Alternative: If you want to change the width of the border, you should use the `border-width` CSS property.

the summary attribute

Every time you create a table you must include the `summary` attribute just as you must include `alt` text for all of your images.The value of the `summary` should be a short description of the table's contents. This value isn't used by normal visual browsers; instead, it's intended for screen readers and other browsers created for users with disabilities. For example:

```
<table summary="vital statistics">
```

Alternative: You should describe the structure of the table in a `<caption>` or put the entire table in a `<figure>` and describing it in a `<figcaption>`. Alternatively, you could simplify the structure of the table so that no explanation is needed.

the cellspacing attribute

The `cellspacing` attribute controls the amount of space placed between adjacent cells in a table and along the outer edges of cells along the edges of a table. By including the `cellspacing` attribute, you can widen or reduce the interior cell borders. For instance, to make the thinnest possible interior cell borders, include the `cellspacing=0` attribute in the table's tag:

```
<table cellspacing="0">
```

Alternative: Use the CSS property `border-spacing` on the table.

the cellpadding attribute

The `cellspacing` attribute controls the amount of space placed inside of each cell between the cell contents and the border of the cell. By including the `cellspacing` attribute, you can increase the size of the table cells and create more space around each cell's contents:

```
<table cellspacing="0" cellpadding="8">
```

Alternative: Use the CSS `padding` property on the table's `<td>` and `<th>` elements.

the width & align attribute

The `align` attribute specifies the horizontal alignment of the content in a cell, and the `width` attribute specifies the width of a table. If the `width` attribute is not set, a table takes up the space it needs to display the table data.

```
<th align="center" width="200px">
```

Alternative: For `align` use the CSS `margin` property instead, and for `width` use the CSS `width` property.