



Oldham, Lancashire  
United Kingdom

Chris Mills works for Opera Software (the Viking web browser vendor), evangelising open web standards and Opera technologies, and heading up Opera's education activities. He publishes regular design and development articles on <http://dev.opera.com>, and is the creator of the Opera Web Standards Curriculum.

Outside of work, Chris is a metal warrior, playing really fast drums in the mighty Conquest of Steel. He lives in Oldham in the Northwest of England, with Kirsty, Gabriel and Elva.

<http://www.opera.com>

<http://dev.opera.com>

<http://www.opera.com/wsc>

<http://twitter.com/chrisdavidmills>

# CHAPTER 10

# HTML Intro

by **Chris Mills**

In this chapter I will start to take you through probably the most important web technology used when building websites: Hypertext Markup Language, or HTML. HTML is the technology that you use to contain all of the data—or content—that is to appear on your website. It gives your content structure, and allows you to ascribe meaning to different parts of it. For example, is that part a paragraph, or a list, or a data table, or a form for site visitors to enter feedback into? Are those words more important than the others; are they a quote?

There are many different parts to HTML, and we will look at all the details in forthcoming chapters. Here we will look at the basics, to get you acquainted with the technology.

At the end of the chapter we will also have a brief look at the upcoming new version of HTML, HTML5, seeing how the code differs, and what the new version might mean for your work.

***In this chapter you have the opportunity to Try it yourself! with the convenient sample code downloads found on the book's companion website—<http://interactwithwebstandards.com>.***

## The history of HTML

When Tim Berners-Lee invented the World Wide Web, he created the first web server and web browser and the first version of HTML. HTML has changed considerably since its early days, but more than half of the original features of the language are still used today, and the original spirit of HTML lives on. The idea is to provide a simple technology that anyone can use to publish content that is viewable by anyone else. Web content should be accessible regardless of the device being used to browse the Web, or the user's personal circumstances (such as where they live, or if they have any disabilities).

As more people started writing web pages and more web browsers became available, more features were added to HTML. Many were adopted universally (e.g., the `<img>` element used to insert an image into a document was first implemented in the NCSA Mosaic browser, and is now supported by very nearly all browsers), whereas some were more proprietary and really only used in one or two browsers. There was a growing need for standardisation—so that authors of web browsing software had a guide that definitively described to them what HTML looked like to help them judge whether they were missing out on implementing some features. The IETF (Internet Engineering Task Force—a standards body concerned with interoperability across the Internet) published a draft proposal of HTML in 1993. This expired without becoming a standard in 1994, but prompted the IETF to create a working group to look at HTML standardisation.

In 1995, HTML 2.0 was written, taking ideas from the original HTML draft. An alternate proposal called HTML+ was written by Dave Raggett; it was used as a basis for many of the new elements implemented by browsers (such as the method for inserting images into documents, pioneered by NCSA Mosaic). A draft of HTML 3.0 followed later that year, but work on that version was discontinued because of a lack of support for the direction by browser makers. HTML 3.2 dropped many of the new features of 3.0, and instead adopted many creations of the then-popular browsers Mosaic and Netscape Navigator. In 1997, the World Wide Web Consortium (W3C) published HTML 4.0 as a recommendation that adopted more browser-specific extensions but also attempted to rationalise and clean up HTML. This was done by marking various elements as **deprecated**—this

means the elements are obsolete and, whilst they still exist in this version, they will be removed in a later revision. This was to encourage better and more meaningful use of HTML in documents (described later in “The importance of good semantics”).

HTML 4.01 was published in 1999, with some errata noted in 2001. In 2000, the W3C also published the XHTML 1.0 specification, which was HTML re-structured as valid XML.

In 2005, a splinter working group called the Web Hypertext Application Technology Working Group (WHATWG) started work on HTML5, a new version of the specification that aims to address the shortcomings of HTML 4.01. You will learn more about this version in the last section of this chapter, entitled “HTML5.”

## What HTML is

Web browsers read HTML, a type of file that usually has an extension of `.html`, just like common desktop applications such as Microsoft Word and Open Office read `.doc` files, and Photoshop and Fireworks read common image formats like `.psd`, `.gif`, `.png`, and `.bmp`. When you look at an HTML page on the Web, the `.html` file is sent to your computer and interpreted and displayed by the web browser you are using to surf the Web.

HTML consists of content—the words, references that pull in images to be displayed, and other data found on a web page—wrapped in special syntax markers that indicate what the different pieces of content are (paragraphs, lists, images, etc.) These special markers are called **elements**, which are further subdivided into opening and closing **tags**. Web browsers (sometimes known as **user agents**) take this content and the elements it is wrapped in, and display it as intended. For example, a simple paragraph and heading is marked up in HTML like so:

```
<h1>A simple HTML example</h1>
```

```
<p>This is a simple paragraph of text, marked up in  
HTML. Above it there is a heading, or title, which tells  
you instantly what this HTML page is all about.</p>
```

When interpreted in a web browser, this will appear as shown in **Figure 10.1**.



*Figure 10.1: A simple HTML example, rendered in a web browser.*

In most browsers there is a **Source** or **View Source** option, commonly under the **context menu** (the menu that comes up when you right-click on a PC or Cmd + click on a Mac while viewing the HTML page). Viewing source and looking at other people's code is how many people have learned HTML for years, and this is indeed one of the beauties of the Web; it is an open, sharing environment.



### Resource // User agent

The term **user agent** refers to more than just web browsers—it means any software that is used to access web pages on behalf of users. There is an important distinction to be made here: All types of desktop browser software (Internet Explorer, Opera, Firefox, Safari, etc.) and alternative browsers for other devices (such as the Wii Internet channel, and mobile phone browsers such as Opera Mini and WebKit on the iPhone) are web browsers. But there are other types of user agents. There are automated programs that Google and Yahoo! use to index the Web to be displayed as search engine results. These are also user agents, although no web user is controlling them directly.

One of the great things about HTML is that it is free to create. All you need to write or edit HTML is a text editor such as Notepad++ on Windows, Text Wrangler on Mac, or Gedit or KATE on Linux. All of these are free.

You could choose a more feature-rich alternative such as Dreamweaver, Coda, Quanta or Eclipse if you like, but the choice is yours. Some people

like a really basic, stripped-down text editor, and some people like a more visual approach to web design involving dragging boxes around the screen and filling in forms.

You should have chosen a text editor by this point. Just make sure it supports both UTF and ISO encodings (see Chapter 4, “Internet Fundamentals”); ideally it should also have syntax highlighting and line numbering.

When you are editing an HTML file, it is a good idea to have it open in your text editor, while at the same time having it open in a web browser. This way you can make edits to the code, save the file, and then jump over to your browser and refresh the page to check that your changes have given the desired results (some more advanced code editors have a “preview” feature, which allows you to check your web page output without leaving the editor application—look out for those).

## Try it yourself!

It’s your turn to start having a play with some HTML:

1. Look at some of your favorite web pages. Use the View Source feature to have a look at the underlying HTML code that makes up the page. Bear in mind that not all of the code you will see will necessarily constitute best practices HTML.
2. Find some simple content to mark up—it could be anything, from your favorite poem, or a recipe in a cookbook, to your life story! Research what elements are needed to mark it up.
3. In the code download for this chapter, find the file named `simple_html_example.html`, open it in your text editor, and replace the content (the stuff in between the `<body>` and `</body>` tags) with your content. Add elements as necessary to mark up your content.
4. Once you’ve finished, save your file and then load it in a web browser to see what it looks like when rendered. If it doesn’t look quite like you were hoping, feel free to make more edits.
5. Take some time to discuss the code with your teacher/tutor/other students: whether it is correct, whether it could be done better, etc.



# The structure of an HTML document

The smallest possible valid HTML document would be something like

**Figure 10.2.**



*Figure 10.2: Valid HTML document.*

Let's go through each of the features in order:

- 1 The document starts with a **document type definition** or **DOCTYPE** (described in more detail in Chapter 12, "`<head>`"). This points to a set of rules that the HTML in the document should follow to be regarded as correct.
- 2 After this, we have the `<html>` element, which is a wrapper around the entire document. The closing `</html>` tag is the last thing in any HTML document.
- 3 Inside the `<html>` element, we first find the `<head>` element. This contains information about the document (also known as metadata), which is generally not displayed in the browser (again, see Chapter 12 for more information).
- 4 Inside the `<head>` is the `<title>` element, which defines the "Example page" heading you can see in the title bar at the top of the browser window in **Figure 10.3**.
- 5 After the `<head>` element there is a `<body>` element, which contains the actual content of the page.
- 6 In this case the entire page content is a single header element—`<h1>`—that contains the text "An even simpler HTML example!!"

And that's our document in full. Rendered in a browser, it looks like **Figure 10.3**.



*Figure 10.3: An even simpler HTML example.*

As you can see, elements often contain other elements. The `<body>` of the document will invariably end up involving many elements placed inside one another (also known as **nested elements**). It might help you to think of HTML elements as boxes that can contain other smaller boxes.

## The syntax of HTML elements

As you have already seen, a basic element in HTML consists of two tags wrapped around a block of text. There are some elements that don't wrap around text, and in almost every case elements can contain sub-elements (such as `<html>` containing `<head>` and `<body>` in the previous example). Elements can also have **attributes**, which can modify the behavior of the element and introduce extra meaning. **Figure 10.4a** and **Figure 10.4b** show the anatomy of a basic HTML element.



*Figure 10.4a: The anatomy of a basic HTML element.*

- 1 Parent element
- 2 Child element



- Elements that allow us to link from our document to other people’s documents.
- Elements that allow us to specify custom semantics and divisions in content.

**Tip:** The Web is case sensitive—be very mindful of this. For example, if you are trying to display an image file and you’ve written `cats.jpg`, but the file has actually been named `Cats.JPG`, your web page won’t display it. This is often a problem for beginners working on local Windows environments, which are not case sensitive. The casing of your file names is one of the first things to check if something inexplicably fails to appear.



## Try it yourself!

Do some research! Have a look through the recommended resources available on the InterACT website (<http://interact.webstandards.org/curriculum/>) and find out some information about three to five HTML elements not already covered in this chapter. Write a blog post about what they do, with code examples.



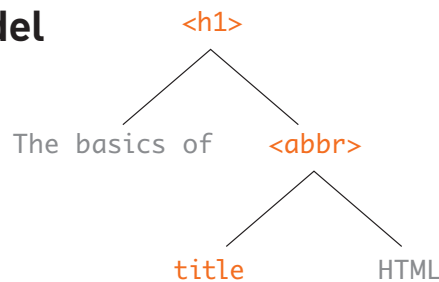
## The Document Object Model

The previous example contained a heading element with an abbreviation element inside it. When a browser renders this code to display it for you, it converts the HTML into a **Document Object Model** (DOM), which can be thought of as a tree-like structure, or hierarchy. So the following HTML:

```
<h1>The basics of
  <abbr title="Hypertext Markup Language">HTML</abbr>
</h1>
```

can be represented by the DOM seen in **Figure 10.6**.

In DOM-speak, `<h1>` is the **parent** element **node** of both the `<abbr>` element node and the text node “The basics of.” The `<abbr>` element



*Figure 10.6: A basic Document Object Model diagram.*

is referred to as a **child** element node of the `<h1>` element, and contains two children of its own: an attribute node of `title`, and another text node containing the text “HTML.”



**Tip:** Every element, attribute, and chunk of text is represented by a node, which is a generic container for a piece of content inside the DOM. When an element contains another element, the contained element can be referred to as a child, and the container can be referred to as a parent. If you have two elements inside another element, those elements are at the same level in the DOM, and are referred to as siblings. These terms are standard computer science terms used when talking about datamodels.

Now, you may be wondering what the point of this DOM representation is. Simply put, a well-formed DOM is essential to make sure that browsers interpret your web page structure correctly and consistently. When you start working with CSS and JavaScript, you’ll see that you need to travel up and down the different levels of the DOM hierarchy to select elements you want to manipulate. The only way to ensure that this will work correctly is to make sure that the DOM is well-formed—see the “HTML best practices” section later in the chapter.

## Block level and inline elements

There are two general categories of elements in HTML, which correspond to the types of content and structure those elements represent: block level elements and inline elements. **Block level** means a higher-level element, normally informing the structure of the document. It may help to think of block level elements being those that start on a new line, breaking away from what went before. Some common block level elements include paragraphs, list items, headings and tables.

**Inline elements** are those that are contained within block level structural elements and surround only small parts of the document’s content, not entire paragraphs and groupings of content. An inline element will not cause a new line to appear in the document; it is the kind of element that would appear in a paragraph of text. Some common inline elements include hypertext links, emphasized words and abbreviations, and short quotations.

# The importance of good semantics

**Semantics** means the meaning of a word, phrase, sentence, etc. In HTML terms, having good semantics means that your HTML should be self-describing—the elements wrapping the content should match the function or purpose of the content itself. Let’s look at an example. If we wanted to mark up a top-level heading followed by two lower-level headings, with content in between them, we *could* do it like this:

```
<font size="5">Information about cats</font>
<font size="2">This document contains information
about cats.</font>
<font size="4">Feeding cats</font>
<font size="2">Cats eat cat food.</font>
<font size="4">Cat games</font>
<font size="2">Cats like to play with balls of wool,
and chase mice.</font>
```

**But this is bad—`<font>` elements are bad, bad old practice, and you should never use them in your work. In the bad old days of the Web this sort of markup was common, and it can still be found on many sites today.**



Visually this results in the desired look, but in the end it’s just a bunch of text blocks that have no meaning. In addition, you should never specify how your HTML looks inside the HTML—all styling information should go in the CSS.

This is not good at all. Instead you should use the right elements to convey the right meaning:

```
<h1>Information about cats</h1>
<p>This document contains information about cats.</p>
<h2>Feeding cats</h2>
<p>Cats eat cat food.</p>
<h2>Cat games</h2>
<p>Cats like to play with balls of wool, and
chase mice.</p>
```

That’s much better. This example uses heading elements to mark up headings, and paragraph elements to mark up paragraphs. And it

doesn't rely just on how it looks for meaning, so it's unambiguous and machine-readable.

As a side note, you could also mark up the content like this:

```
<div id="top-heading">Information about cats</div>
<div class="paragraph">This document contains
information about cats.</div>
<div class="second-level-heading">Feeding cats</div>
<div class="paragraph">Cats eat cat food.</div>
<div class="second-level-heading">Cat games</div>
<div class="paragraph">Cats like to play with balls
of wool, and chase mice.</div>
```

And then use different CSS rules to style the different types of content how you want them to look. This is fine, surely? `<div>` elements with `id` and `class` attributes are perfectly allowed in modern HTML, and now we are putting all the styling information in CSS, to satisfy best practices.



**But this is still bad! `<div>` elements are generic container elements and have no intrinsic meaning, and there are much better elements available to use for marking up headings and paragraphs.**

You can style it to look like a top-level heading and two second-level headings with paragraphs in between, but the HTML elements used *do not* describe the content as being so. The content is currently described as “six pieces of text, all of the same weight and function.”

So why is a lack of semantics so bad? There are many reasons, but the two main ones are as follows:

- First of all, people with impaired vision use an assistive technology called a screen reader to read web pages out to them. These use semantics in many ways—for example, they use headings to navigate the different pieces of content, so the users can find what they want on a page. If there are no heading elements present, it is impossible for these users to effectively navigate the content.
- Second, search engines such as Google and Yahoo! use keywords they find on pages to index and rank content, and they give more weight to keywords in headings. If your content contains no headings, it will be less likely to come up in search results, so fewer users will find it.

## Generic containers: `<div>` and `<span>`

One question you might be thinking at this point is “What do I use if there are no suitable elements available to mark up my content?” Fortunately, the HTML spec contains two elements that serve as generic containers you can use to identify custom pieces of content—`<div>` and `<span>`:

- `<div>` is used to wrap block-level elements. For example, if you wanted to identify three paragraphs as the main content of the page, and a list and two paragraphs as the navigation menu of the page, you would wrap them in `<div>`s with suitable classes, such as `class="content"` or `class="menu"`.
- `<span>` is used to wrap inline elements/content. For example, if you wanted to identify a few words of text inside a paragraph as an editor’s note or warning note with a special style, you’d wrap them in `<span>`s.

You’ll see a lot of examples of these elements in action throughout this book, so all will become clear. The most common use you will see for these elements is as common page elements that you’ll see on many sites, such as headers, footers or content sections.

As you’ll see later in this chapter, HTML5 includes specific elements to mark up many of these common sections, as it was thought that they were *de facto* standards that should be officially recognised.

### Try it yourself!

Research other reasons why bad semantics in your HTML causes problems, and write a blog post about them.



## HTML best practices

There are certain best practices that you should follow when writing your HTML. Browsers are resilient with built-in error handling, so if you make mistakes (for example, a stray quote mark, or not closing an element, which leads to a poorly formed DOM), it doesn’t cause problems up front, as browsers compensate for it. They fill in the blanks and guess what you intended.

However, different browsers interpret your broken markup in different ways. This can lead to inconsistencies and weird behaviour later on

when you want to style things with CSS, add behaviour with JavaScript, etc. Therefore it's best to follow consistent proper syntax rules and best practices. It is analogous to dotting your i's and crossing your t's.

In general, wherever possible, you should try to follow these best practices when writing HTML:

Best Practice	Correct	Incorrect
Always include quotes around attribute values.	<code>title="Hypertext Markup Language"</code>	<code>title=Hypertext Markup Language</code>
Always close elements properly after they have been opened.	<code>&lt;h1&gt;This is a heading&lt;/h1&gt;</code>	<code>&lt;h1&gt;This is a heading</code>
Always nest elements properly.	<code>&lt;p&gt;&lt;em&gt;This text is emphasised&lt;br&gt;&lt;strong&gt;strongly&lt;/strong&gt; in parts&lt;br&gt;&lt;/em&gt;&lt;/p&gt;</code>	<code>&lt;p&gt;&lt;em&gt;This text is emphasised&lt;br&gt;&lt;strong&gt;strongly in parts&lt;/em&gt;&lt;/p&gt;</code>

*Table 10.1: HTML best practices.*



**Tip:** The Web is a very forgiving programming environment compared to, say, Java or C++. This is a blessing and a curse—it lowers the barrier of entry so that more people can create web pages, but it has led to some really bad code being put up there on the Web. This is what we are hoping to eventually stamp out!

The trouble in the last example is that the `<strong>` element is no longer cleanly inside the `<em>` element, and they overlap, so the logical DOM parent-child structure no longer works, and you get uncertainties: is the text “strongly in parts” made stronger by the `<strong>` element or not?

In addition, there are some best practices that are not directly related to making markup valid:

- Always separate content from presentation—you should keep all content inside the HTML, but separate all styling information into CSS. This means not using presentational elements like `<font>`.
- Make sure your text is always well-worded and easily readable. Read Chapter 5, “Writing for the Web,” for more information on this.
- Make sure your content is as usable and accessible as possible: see Chapters 22-24 for more information on accessibility.

## Character references

One last item to mention in an HTML document is how to include special characters. In HTML the characters `<`, `>`, and `&` are special. They start and end parts of the HTML syntax, rather than representing the characters less-than, greater-than, and ampersand. One of the earliest mistakes you might make when dealing with character references is including an ampersand in your HTML content and then having something unexpected appear. For example, writing “Imperial units measure weight in `stones&pounds`” could end up appearing as “...`stones&#x26`” in some browsers because they get confused and interpret special characters as markup rather than simply text. The literal string `&pound;` is a character reference in HTML.

A **character reference** is a way of including a character in a document that is difficult or impossible to enter using a keyboard, or in a particular document encoding. The ampersand (`&`) introduces the reference and the semi-colon (`;`) ends it.

References can either be numbers (numeric references) or shorthand words (entity references). An actual ampersand has to be entered into a document as `&amp;`, which is the character entity reference, or as `&#38;`, which is the numeric reference. A full chart of character references can be found on <http://evolt.org/entities>.

## HTML5

In 2005, a group of professional web developers decided to write a new version of the HTML specification—HTML5—under the mantle of the WHATWG. This was officially adopted by the W3C as a draft in 2008, and is still in progress. Why do we need a new HTML? Well, the current version works very well for the original purpose of HTML—which was to create collections of static documents connected together by links—but it doesn’t do so well for the purpose HTML has been adopted for in modern times.

These days, web developers often use HTML and associated technologies to create applications that behave more like desktop software, with more dynamic user interaction and features. So far developers have been doing

this with HTML 4.01, but it has involved a lot of hacking and complicated scripting. HTML5 removes a lot of this by providing things like built-in form validation, offline storage, video rendering and more. Different browsers already have support for some HTML5 features, although some parts of the technology are still at an early experimental stage.

So how does this affect you? Don't fret: you are not suddenly going to have to ditch the teachings in this book and learn a new version! HTML5 does not replace HTML 4. It simply adds powerful new elements and other features, while also clarifying (and in many cases simplifying) the syntax that makes a document valid.



### Resource // HTML5 tutorials

You can find a lot of good articles showing how to use HTML5 features on <http://dev.opera.com>. Another great resource for HTML5 tips, tutorials, and news is <http://html5doctor.com>. The article “Designing a blog with HTML5” is a particularly good guide to basic HTML5 syntax — see <http://html5doctor.com/designing-a-blog-with-html5/>.

HTML5 elements, although a bit different, are logical and well thought out—a sample HTML5 document looks like so:

```
<!DOCTYPE HTML>
<html>
  <head>
    <title>A sample HTML5 document</title>
  </head>
  <body>
    <header>
      <h>Sample document</h>
    </header>
    <section>
      <article>
        <h>Subheading</h>
        This is some of the content of the document.
      </article>
```

```
<article>
  <h>Subheading 2</h>
  This is some more content, in a different
  section of the document.
</article>
</section>
<footer>
  <a href="mailto:me@opera.com">Chris Mills</a>
  &copy; 2009
</footer>
</body>
</html>
```

Nothing has changed that radically—the elements are a bit different, but they reflect common semantics that you find on many websites that developers tend to create using `<div>` elements, for example `<div id="header">`, which is replaced with `<header>` in HTML5. You'll also notice that the DOCTYPE is much simpler than the ones you saw earlier.

### Try it yourself!

You can start experimenting with these new elements already, even though browsers don't yet officially support them. All you have to do is write a rule in your CSS to make them all display as block elements. Try this out!



## Summary

By this point, you have taken your first steps towards understanding how HTML works. In the following chapters we will look at HTML features in much more detail, as well as Cascading Style Sheets, the technology that styles HTML content.