Module-1:Traditional HTML and XHTML:

First Look at HTML and XHTML, Hello HTML and XHTML World, HTML and XHTML: Version History, HTML and XHTML DTDs: The Specifications Up Close, (X)HTML Document Structure, Browsers and (X)HTML, The Rules of (X)HTML, Major Themes of (X)HTML, The Future of Markup—Two Paths?

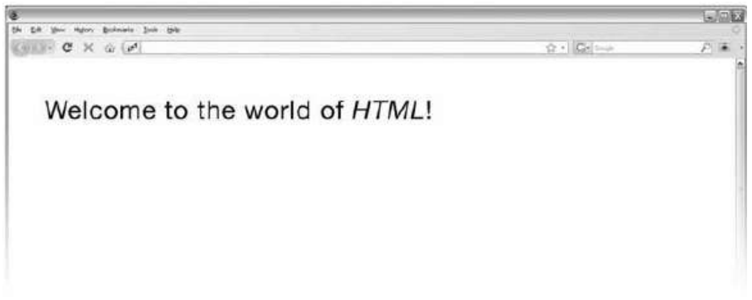
Markup languages are ubiquitous in everyday computing. Although you may not realize it, word processing documents are filled with markup directives indicating the structure and often presentation of the document. In the case of traditional word processing documents, these structural and presentational markup codes are more often than not behind the scenes. However, in the case of Web documents, markup in the form of traditional Hypertext Markup Language (HTML) and its Extensible Markup Language (XML)-focused variant, XHTML, is a little more obvious. These not-so-behind-the-scenes markup languages are used to inform Web browsers about page structure and, some might argue, presentation as well.

First Look at HTML and XHTML

In the case of HTML, markup instructions found within a Web page relay the structure of the document to the browser software. For example, if you want to emphasize a portion of text, you enclose it within the tags **<em>** and **</em>**, as shown here:
<em>This is important text!**</em>**

When a Web browser reads a document that has HTML markup in it, it determines how to render it onscreen by considering the HTML elements embedded within the document:

```
Welcome to the world of <em>HTML</em>!
```

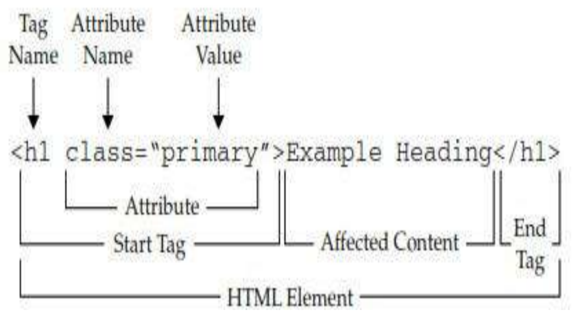

Welcome to the world of *HTML*!

an HTML document is simply a text file that contains the information you want to publish and the appropriate markup instructions indicating how the browser should structure or present the document. Markup *elements* are made up of a start tag, such as **<strong>**, and typically, though not always, an end tag, which is indicated by a slash within the tag, such as **</strong>**. The tag pair should fully enclose any content to be affected by the element, including text and other HTML markup.
Under traditional HTML (not XHTML), the close tag for some elements is optional because their closure can be inferred. For example, a **<p>** tag cannot enclose another **<p>** tag, and thus the closing **</p>** tag can be inferred when markup like this is encountered:
<p>This is a paragraph.
<p>This is also a paragraph.
Such shortened notations that depend on inference may be technically correct under the specification, but stylistically they are not encouraged. It is always preferable to be precise, so use markup like this instead:

<p>This is a paragraph.**</p>**
<p>This is also a paragraph.**</p>**

There are markup elements, called *empty elements*, which do not enclose any content, thus need no close tags at all, or in the case of XHTML use a self-close identification scheme. For example, to insert a line break, use a single **<br>** tag, which represents the empty **br** element, because it doesn't enclose any content and thus has no corresponding close tag:
<br>
However, in XML markup variants, particularly XHTML, an unclosed tag is not allowed, so you need to close the tag
<br></br>
or, more commonly, use a self-identification of closure like so:
<br />
The start tag of an element might contain *attributes* that modify the meaning of the tag. For example, in HTML, the simple inclusion of the **noshade** attribute in an **<hr>** tag, as shown here:
<hr noshade>
indicates that there should be no shading applied to this horizontal rule. Under XHTML, such style attributes are not allowed, because all attributes must have a value, so instead you have to use syntax like this:
<hr noshade="noshade" />
As the preceding example shows, attributes may require values, which are specified with an equal sign; these values should be enclosed within double or single quotes. For example, using standard HTML syntax,
<img src="dog.gif" alt="Angus-Black Scottish Terrier" height="100" width="100">
specifies four attributes for this **<img>** tag that are used to provide more information about the use of the included image. Under traditional HTML, in the case of simple alphanumeric attribute values, the use of quotes is optional, as shown here:
<p class=fancy>
Regardless of the flexibility provided under standard HTML, you should always aim to use quotes on all attributes. You will find that doing so makes markup more consistent, makes upgrading to stricter markup versions far easier, and tends to help reduce errors caused by inconsistency.

A graphical overview of the HTML markup syntax shown so far is presented here:



Hello HTML and XHTML World
Our first complete example written in strict HTML 4 is shown here:

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN"
"http://www.w3.org/TR/html4/strict.dtd">
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=utf-8">
<title>Hello HTML 4 World</title>
<!-- Simple hello world in HTML 4.01 strict example -->
</head>
<body>
<h1>Welcome to the World of HTML</h1>
<hr>
<p>HTML <em>really</em> isn't so hard!</p>
<p>Soon you will &hearts; using HTML.</p>
<p>You can put lots of text here if you want.
We could go on and on with fake text for you
to read, but let's get back to the book.</p>
</body>
</html>
```

in the case of XHTML, which is a form of HTML that is based upon the syntax rules of XML, we really don't see many major changes yet in our example:

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
<meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
<title>Hello XHTML World</title>
<!-- Simple hello world in XHTML 1.0 strict example -->
</head>
<body>
<h1>Welcome to the World of XHTML</h1>
<hr />
<p>XHTML <em>really</em> isn't so hard either!</p>
<p>Soon you will &hearts; using XHTML too.</p>
<p>There are some differences between XHTML
and HTML but with some precise markup you'll
see such differences are easily addressed.</p>
</body>
</html>
```

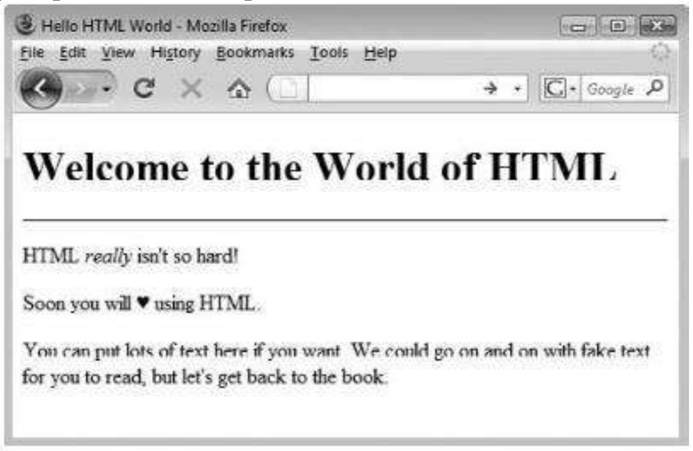
The preceding examples use some of the most common elements used in (X)HTML documents, including:

- The **<!DOCTYPE>** statement, which indicates the particular version of HTML or XHTML being used in the document. The first example uses the strict 4.01 specification, the second uses a reduced form for HTML5 the meaning of which will be explained a bit later on, and the final example uses the XHTML 1.0 strict specification.
- The **<html>**, **<head>**, and **<body>** tag pairs are used to specify the general structure of the document. The required inclusion of the **xmlns** attribute in the **<html>** tag is a small difference required by XHTML.
- The **<meta>** tag used in the examples indicates the MIME type of the document and the character set in use. Notice that in the XHTML example, the element has a trailing slash to indicate that it is an empty element.

- The **<title>** and **</title>** tag pair specifies the title of the document, which generally appears in the title bar of the Web browser.
- A comment is specified by <!-- -->, allowing page authors to provide notes for future reference.
- The **<h1>** and **</h1>** header tag pair indicates a headline specifying some important information.
- The **<hr>** tag, which has a self-identifying end tag (<**hr** />) under XHTML, inserts a horizontal rule, or bar, across the screen.
- The <**p**> and </**p**> paragraph tag pair indicates a paragraph of text.
- A special character is inserted using a named entity (**&hearts;**), which in this case inserts a heart dingbat character into the text.
- The <**em**> and </**em**> tag pair surrounds a small piece of text to emphasize which a browser typically renders in italics.

Viewing Markup Locally

Using a simple text editor, type in either one of the previous examples and save it with a filename such as helloworld.html or helloworld.htm; you can choose which file extension to use, .htm or .html, but whichever you pick for development


Hello HTML World - Mozilla Firefox
File Edit View History Bookmarks Tools Help
Google

Welcome to the World of HTML

HTML *really* isn't so hard!

Soon you will ♥ using HTML.

You can put lots of text here if you want. We could go on and on with fake text for you to read, but let's get back to the book.

If for some reason you didn't save your file with the appropriate extension, the browser shouldn't attempt to interpret the HTML markup. For example, notice here what happens when you try to open the content with a .txt extension:
To make your files available via the server, you might use a process of uploading a file from your system to a remote server via an FTP (File Transfer Protocol) program, as shown here:

If you want to make a change to the document, you could update the markup, save the file, go back to the browser, and click the Reload or Refresh button. Sometimes the browser will still reload the page from its cache; if a page does not update correctly on reload, hold down the SHIFT key while clicking the Reload button, and the browser should refresh the page.

Viewing Markup with a Web Server

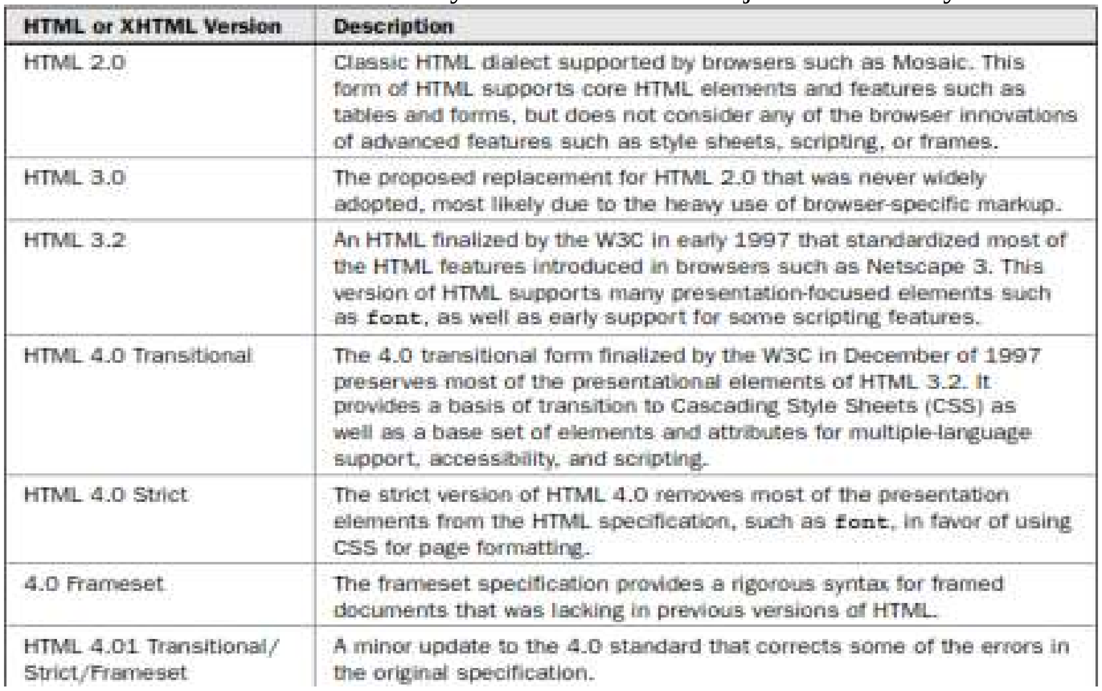
There are many options for employing a Web server. You may decide to run your own local development Web server on your desktop system or use some hosted server instead. In either case, you need to get the files somewhere under the Web server's document root folder so that they can be served out. Very often this directory has a common name like inetpub, htdocs, site, or www, but it really could be just about anything, so make sure you check the server you end up using.

Many Web editors also allow you to synchronize files between a local directory and your remote server.

On the Web server, you most likely will use the .html or .htm file extension for your files. When HTML files are placed in the appropriate directory, the user would issue a URLin their browser like
http://yoursitename/sitepath/helloworld.html
and that will then return the file in question. However, note that when a marked-up document is delivered over the network, it is not the file extension that indicates to the browser that the content is HTML, but rather the Content-Type: header found in the network stream:
The browser then takes the header and maps it to the action of parsing the document as HTML.

HTML and XHTML: Version History

Since its initial introduction in late 1991, HTML (and later its XML-based cousin, XHTML) has undergone many changes. Interestingly, the first versions of HTML used to build the earliest Web pages lacked a rigorous definition. Fortunately, by 1993 the Internet Engineering Task Force (IETF) began to standardize the language and later, in 1995, released the first real HTML standard in the form of HTML 2.0. You will likely encounter more than just the latest style of markup



HTML or XHTML Version	Description
HTML 2.0	Classic HTML dialect supported by browsers such as Mosaic. This form of HTML supports core HTML elements and features such as tables and forms, but does not consider any of the browser innovations of advanced features such as style sheets, scripting, or frames.
HTML 3.0	The proposed replacement for HTML 2.0 that was never widely adopted, most likely due to the heavy use of browser-specific markup.
HTML 3.2	An HTML finalized by the W3C in early 1997 that standardized most of the HTML features introduced in browsers such as Netscape 3. This version of HTML supports many presentation-focused elements such as `font`, as well as early support for some scripting features.
HTML 4.0 Transitional	The 4.0 transitional form finalized by the W3C in December of 1997 preserves most of the presentational elements of HTML 3.2. It provides a basis of transition to Cascading Style Sheets (CSS) as well as a base set of elements and attributes for multiple-language support, accessibility, and scripting.
HTML 4.0 Strict	The strict version of HTML 4.0 removes most of the presentation elements from the HTML specification, such as `font`, in favor of using CSS for page formatting.
4.0 Frameset	The frameset specification provides a rigorous syntax for framed documents that was lacking in previous versions of HTML.
HTML 4.01 Transitional/ Strict/Frameset	A minor update to the 4.0 standard that corrects some of the errors in the original specification.

HTML5	Addressing the lack of acceptance of the XML reformulation of HTML by the mass of Web page authors, the emerging HTML5 standard originally started by the WHATWG² group and later rolled into a W3C effort aimed to rekindle the acceptance of traditional HTML and extend it to address Web application development, multimedia, and the ambiguities found in browser parsers. Since 2005, features now part of this HTML specification have begun to appear in Web browsers, muddying the future of XHTML in Web browsers.
XHTML 1.0 Transitional	A reformulation of HTML as an XML application. The transitional form preserves many of the basic presentation features of HTML 4.0 transitional but applies the strict syntax rules of XML to HTML.
XHTML 1.0 Strict	A reformulation of HTML 4.0 Strict using XML. This language is rule enforcing and leaves all presentation duties to technologies like CSS.
XHTML 1.1	A restructuring of XHTML 1.0 that modularizes the language for easy extension and reduction. It is not commonly used at the time of this writing and offers minor gains over strict XHTML 1.0.

HTML and XHTML DTDs: The Specifications Up Close

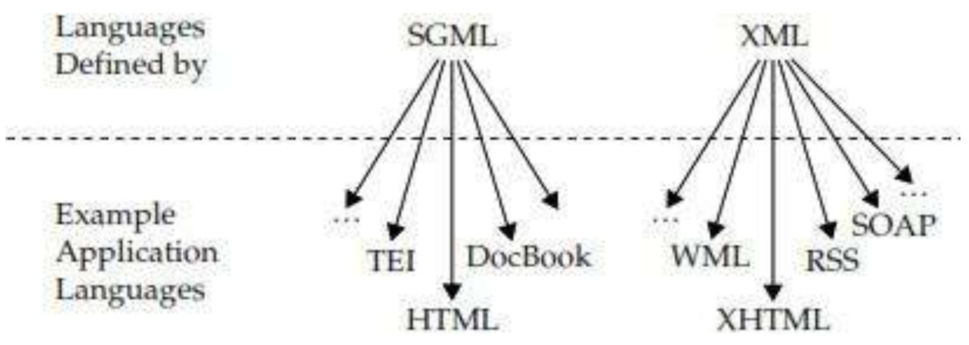
Traditionally, the W3C defined HTML as an pplication of the *Standard Generalized Markup Language* (SGML). SGML is a technology used to define markup languages by specifying the allowed document structure in the form of a *document type definition* (DTD). A DTD indicates the syntax that can be used for the various elements of a language such as HTML.

```
<!--=================== Paragraphs ========================================--
>
<!ELEMENT P - O (%inline;)*          -- paragraph -->
<!ATTLIST P
  %attrs;                                        -- %coreattrs,  %i18n,          %events --
 >
```

The first line is a comment indicating what is below it. The second line defines the **P** element, indicating that it has a start tag (**<P>**), as shown by the dash, and an optional close tag (**</P>**), as indicated by the O.

The final line defines the attributes for a **<P>** tag as indicated by the entity %attrs which then expands to a number of entities like %core, %i18n, and %coreevents which finally expand into a variety of attributes like **id**, **class**, **style**, **title**, **lang**, **dir**, **onclick**, **ondblclick**, and many more.

As mentioned in the previous section on the history of HTML, in 1999 the W3C rewrote HTML as an application of XML and called it XHTML. XML, in this situation, serves the same purpose as SGML: a language in which to write the rules of a language. In fact, XML is in some sense just a limited form of SGML. XML and SGML can be used to write arbitrary markup languages, not just HTML and XHTML. These would be called *applications* or, maybe more appropriately, *application languages*.



The DTD defined in XML for the XHTML language is actually quite similar to the DTD for traditional HTML

```
<!--=================== Paragraphs ===================================-->
<!ELEMENT p %Inline;>
<!ATTLIST p
  %attrs;
  >
```

Document Type Statements and Language Versions

(X)HTML documents should begin with a **<!DOCTYPE>** declaration. This statement identifies the type of markup that is supposedly used in a document. For example,

<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">

indicates that we are using the transitional variation of HTML 4.01 that starts with a root element **html**. In other words, an **<html>** tag will serve as the ultimate parent of all the content and elements within this document.

A **<!DOCTYPE>** declaration might get a bit more specific and specify the URI (Uniform Resource Identifier) of the DTD being used as shown here:

<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN" "http://www.w3.org/TR/html4/loose.dtd">

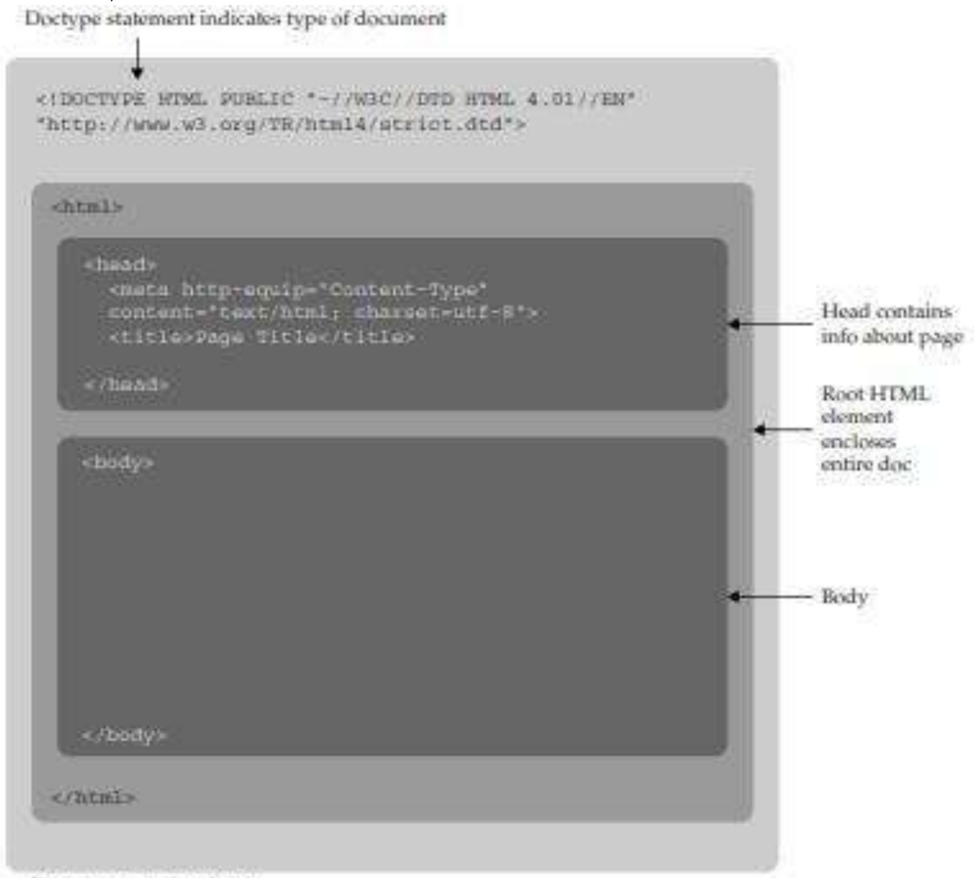
In the case of an XHTML document, the situation really isn't much different:

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

However, do note that the root **html** element here is lowercase, which hints at the case sensitivity found in XHTML.

(X)HTML Document Structure

The DTDs define the allowed syntax for documents written in that version of (X)HTML. The core structure of these documents is fairly similar. Given the HTML 4.01 DTD, a basic document template can be derived from the specification, as shown here:

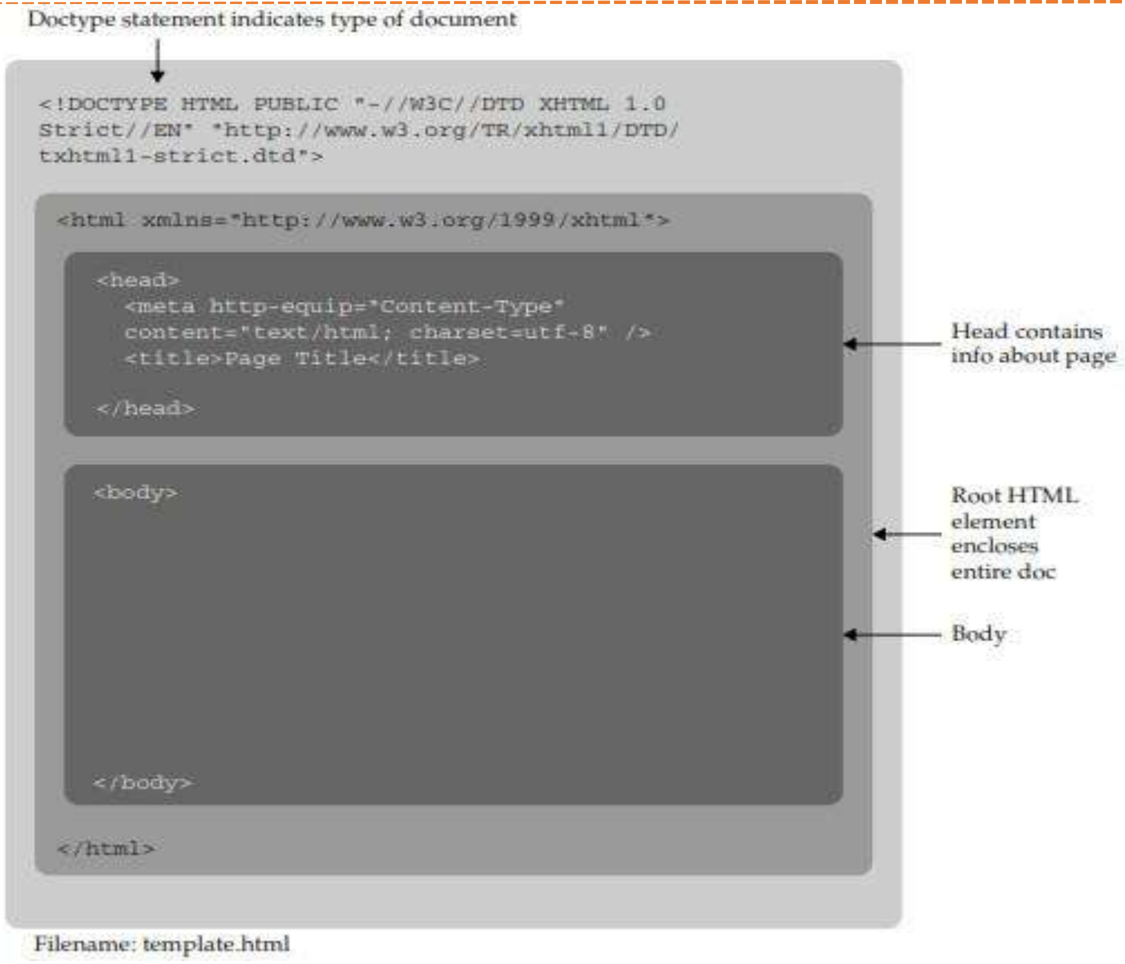

Doctype statement indicates type of document
Head contains info about page
Root HTML element encloses entire doc
Body

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN"
"http://www.w3.org/TR/html4/strict.dtd">

<html>
  <head>
    <meta http-equiv="Content-Type"
    content="text/html; charset=utf-8">
    <title>Page Title</title>

  </head>

  <body>

  </body>

</html>
```

Filename: template.html

In this graphical representation, the **<!DOCTYPE>** indicator, which, as previously mentioned, shows the particular version of HTML being used, in this case 4.01 Transitional. Within a root **html** element, the basic structure of a document reveals two elements: the **head** and the **body**. The **head** element contains information and tags describing the document, such as its **title**, while the **body** element houses the document itself, with associated markup required to specify its structure.

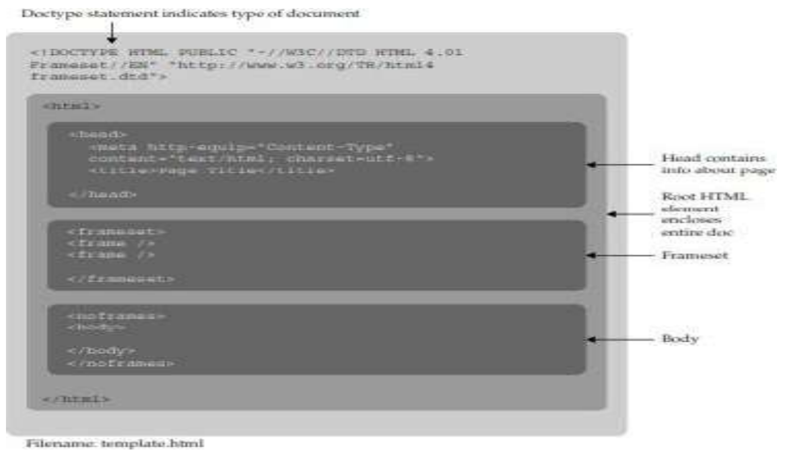
The structure of an XHTML document is pretty much the same with the exception of a different **<!DOCTYPE>** indicator and an **xmlns** (XML name space) attribute added to the **html** tag so that it is possible to intermix XML more easily into the XHTML document:


Doctype statement indicates type of document
Head contains info about page
Root HTML element encloses entire doc
Body
Filename: template.html

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD XHTML 1.0
Strict//EN" "http://www.w3.org/TR/xhtml1/DTD/
txhtml1-strict.dtd">

<html xmlns="http://www.w3.org/1999/xhtml">

  <head>
    <meta http-equip="Content-Type"
    content="text/html; charset=utf-8" />
    <title>Page Title</title>

  </head>

  <body>

  </body>

</html>
```

Alternatively, in either HTML or XHTML (but not in HTML5), we can replace the **<body>** tag with a **<frameset>** tag, which encloses potentially numerous **<frame>** tags corresponding to individual portions of the browser window, termed *frames*. Each frame in turn would reference another HTML/XHTML document containing either a standard document, complete with **<html>**, **<head>**, and **<body>** tags, or perhaps yet another framed document. The **<frameset>** tag also should include a **noframes** element that provides a version of the page for browsers that do not support frames. Within this element,

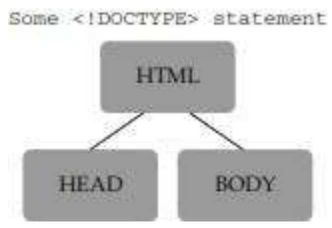
a **<body>** tag should be found for browsers that do not support frames. A visual representation of this idea is shown here:


Doctype statement indicates type of document
Head contains info about page
Root HTML element encloses entire doc
Frameset
Body
Filename: template.html

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01
Frameset//EN" "http://www.w3.org/TR/html4
frameset.dtd">

<html>

  <head>
    <meta http-equip="Content-Type"
    content="text/html; charset=utf-8">
    <title>Page Title</title>

  </head>

  <frameset>
  <frame />
  <frame />

  </frameset>

  <noframes>
  <body>

  </body>
  </noframes>

</html>
```

HTML5 does not support standard frames, though it does preserve inline frames.

Roughly speaking, the structure of a non-framed (X)HTML document breaks out like so:



The Document Head
The information in the **head** element of an (X)HTML document is very important because it is used to describe or augment the content of the document. . In many cases, the information contained within the **head** element is information about the page that is useful for visual styling, defining interactivity, setting the page title, and providing other useful information that describes or controls the document.

The title Element
A single **title** element is required in the **head** element and is used to set the text that most browsers display in their title bar.

<**title**>Simple HTML Title Example</**title**>

Only one **title** element should appear in every document, and most user agents will ignore subsequent tag instances.

However, character entities such as **&copy;** (or, alternatively, **&#169;**), which specifies a copyright symbol, are allowed in a title:

<**title**>Simple HTML Title Example, **&copy;** 2010 WebMonopoly, Inc.</**title**>

To set the appropriate character set, you should include a <**meta**> tag before the page title even though traditionally **title** is considered the first element.

<meta>: Specifying Content Type, Character Set, and More
A <**meta**> tag has a number of uses. For example, it can be used to specify values that are equivalent to HTTP response headers. For example, if you want to make sure that your MIME type and character set for an English-based HTML document is set, you could use

<meta http-equiv="Content-Type" content="text/html; charset=ISO-8859-1">

In summary at the point of writing this edition, it is recommend specifying a `Content-Type` of `text/html` and the UTF-8 character **set**, and doing so as your first element within the head, like so:

```
<head>
<meta http-equiv="Content-Type" content="text/html; charset=utf-8" >
<title>Page title here</title>
</head>
```

Other Elements in the head

The elements allowed within the **head** element include **base**, **link**, **object**, **script**, and **style**. Comments are also allowed

<base> A **<base>** tag specifies an absolute URL address that is used to provide server and directory information for partially specified URL addresses, called *relative links*, used within the document:

<base href="http://htmlref.com/basexeample" >

link> A **<link>** tag specifies a special relationship between the current document and another document. Most commonly, it is used to specify a style sheet used by the document

<link rel="stylesheet" media="screen" href="global.css" type="text/css" >

<object> An **<object>** tag allows programs and other binary objects to be directly embedded in a Web page. Here, for example, a nonvisible Flash object is being referenced for some use:

<script> A **<script>** tag allows scripting language code to be either directly embedded within,

```
<script type="text/javascript">
 alert("Hi from JavaScript!");
 /* more code below */
</script>
```

or, more appropriately, linked to from a Web page:

<script type="text/javascript" href="ajaxtcr.js"></script>

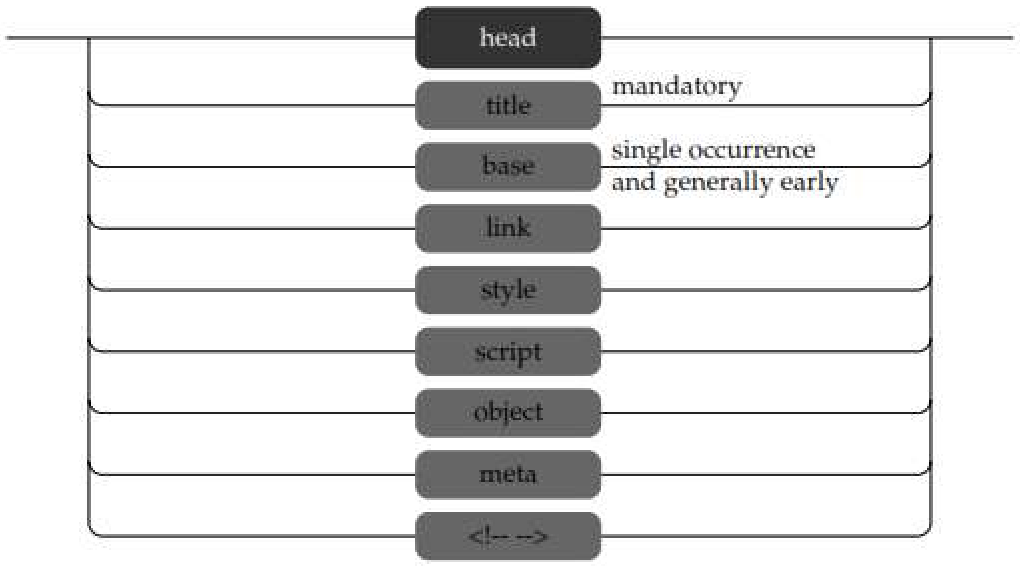
<style> A **<style>** tag is used to enclose document-wide style specifications, typically in Cascading Style Sheet (CSS) format, relating to fonts, colors, positioning, and other aspects of content presentation:

```
<style type="text/css" media="screen">
 h1 {font-size: xx-large; color: red; font-style: italic;}
 /* all h1 elements render as big, red and italic */
</style>
```

Comments Finally, comments are often found in the head of a document. Following SGML syntax, a comment starts with <!-- and ends with --> and may encompass many lines:

```
<!-- Hi I am a comment -->
<!-- Author: Thomas A. Powell

    Book: HTML: The Complete Reference
    Edition: 5
-->
```

The complete syntax of the markup allowed in the head element under strict (X)HTML is shown here:



<p>Some body content here.**</p>**
</body>
</html>

The Document Body

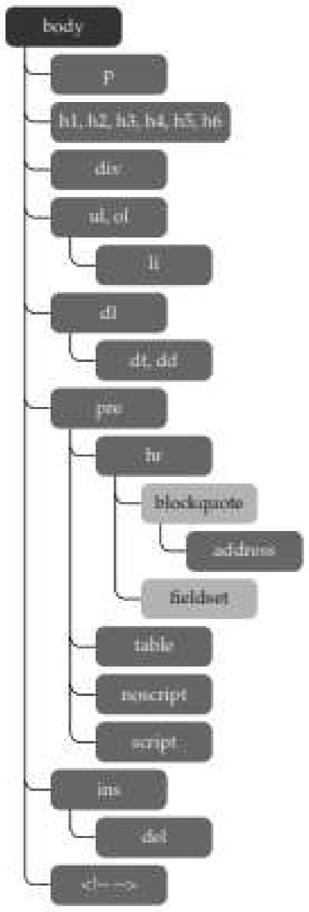
After the head section, the body of a document is delimited by **<body>** and **</body>**. Under the HTML 4.01 specification and many browsers, the **body** element is optional, but you should always include it, particularly because it is required in stricter markup variants. Only one **body** element can appear per document.

Within the body of a Web document is a variety of types of elements. For example, *blocklevel Elements* define structural content blocks such as paragraphs (**p)** or headings (**h1-h6**). Block-level elements generally introduce line breaks visually. Special forms of blocks, such as unordered lists (**ul**), can be used to create lists of information.

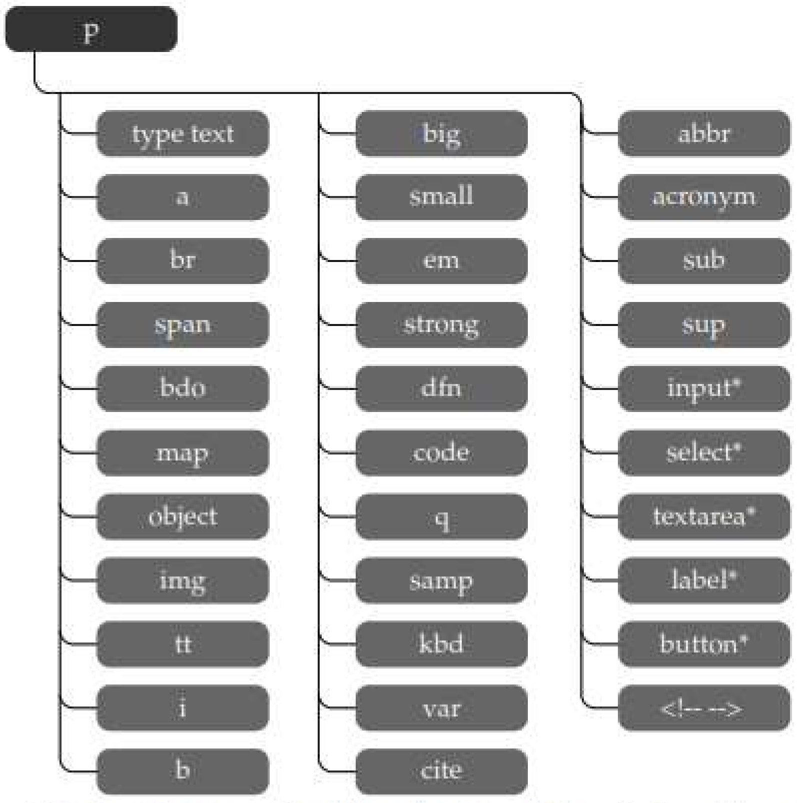
Within nonempty blocks, *inline elements* are found. There are numerous inline elements, such as bold (**b**), italic (**i**), strong (**strong**), emphasis (**em**), and numerous others. These types of elements do not introduce any returns.

Other miscellaneous types of elements, including those that reference other objects such as images (**img**) or interactive elements (**object**), are also generally found within blocks, though in some versions of HTML they can stand on their own. Within block and inline elements, you will find textual content, unless the element is empty. Typed text may include special characters that are difficult to insert from the keyboard or require special encoding. To use such characters in an HTML document, they must be "escaped" by using a special code. All character codes take the form **&*code***;, where ***code*** is a word or numeric code indicating the actual character that you want to put onscreen. For example, when adding a less-than symbol (<) you could use **&lt;** or **&#060;**. Character entities also are discussed in depth in Appendix A.

The full syntax of the elements allowed in the **body** element is a bit more involved than the full syntax of the **head**. This diagram shows what is directly included in the body:



going deeper into the full syntax in a single diagram is unreasonable to present.

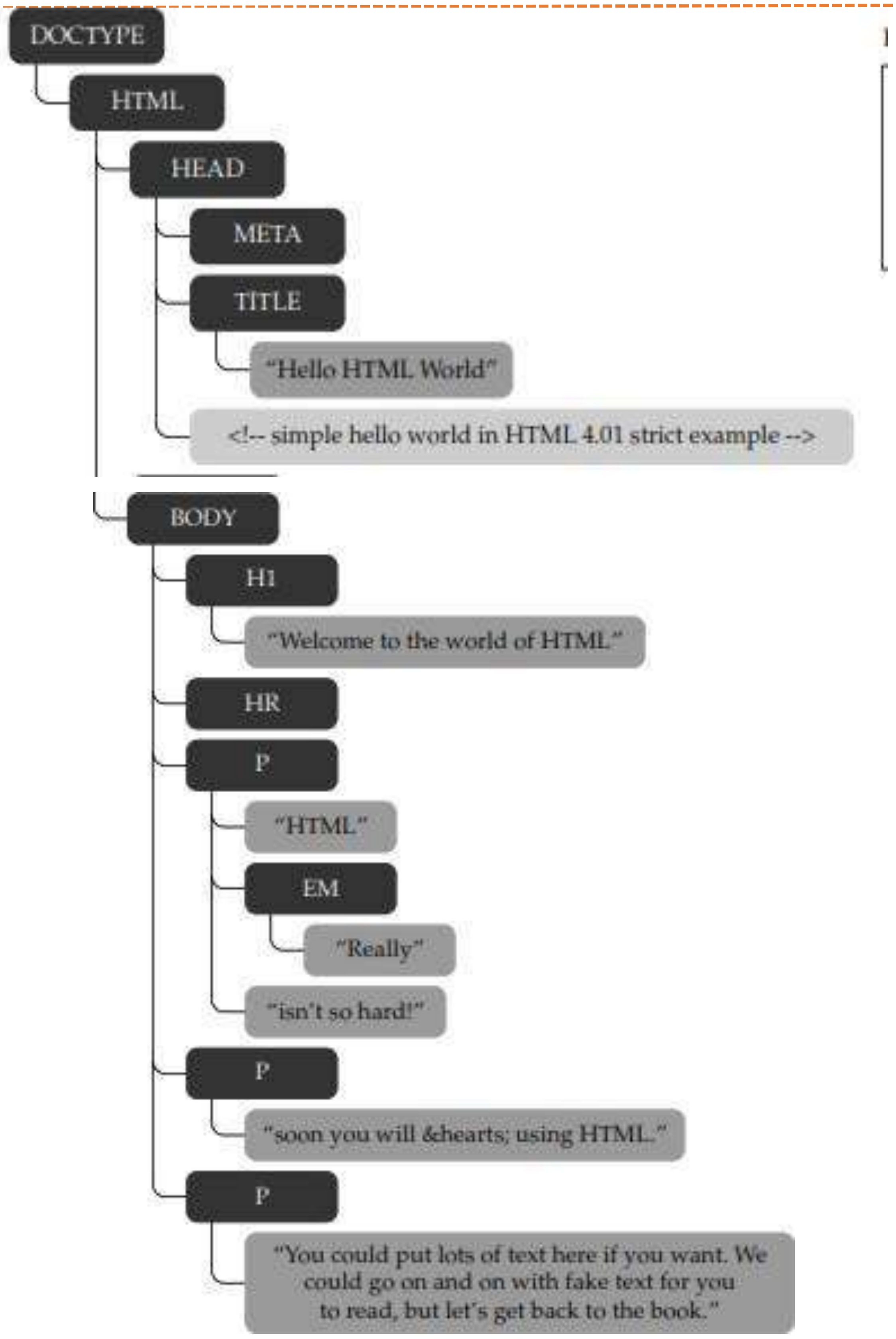


(*) when the element is ultimately a descendent of a form element

Browsers and (X)HTML

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN"
"http://www.w3.org/TR/html4/strict.dtd">
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=utf-8">
<title>Hello HTML World</title>
<!-- Simple hello world in HTML 4.01 strict example -->
</head>
<body>
<h1>Welcome to the World of HTML</h1>

<hr>
<p>HTML <em>really</em> isn't so hard!</p>
<p>Soon you will &hearts; using HTML.</p>
<p>You can put lots of text here if you want.
We could go on and on with fake text for you
to read, but let's get back to the book.</p>
</body>
</html>
```

Example for XHTML type of document


DOCTYPE
HTML
HEAD
META
TITLE
"Hello HTML World"
<!-- simple hello world in HTML 4.01 strict example -->
BODY
H1
"Welcome to the world of HTML"
HR
P
"HTML"
EM
"Really"
"isn't so hard!"
P
"soon you will &hearts; using HTML."
P
"You could put lots of text here if you want. We could go on and on with fake text for you to read, but let's get back to the book."

These parse trees, often called DOM (Document Object Model) trees, are the browsers' interpretation of the markup provided and are integral to determining how to render the page visually using both default (X)HTML style and any CSS attached.

The Rules of (X)HTML

HTML Is Not Case Sensitive, XHTML Is

These markup examples are all equivalent under traditional HTML:

<**B**>Go boldly</**B**>
<**B**>Go boldly</**b**>
<**b**>Go boldly</**B**>
<**b**>Go boldly</**b**>

Attribute Values May Be Case Sensitive

Consider **<img SRC="test.gif">** and **<IMG src="test.gif">**. Under traditional HTML, these are equivalent because the **<img>** tag and the **src** attribute are not case sensitive. However, given XHTML, they should always be lowercase.

(X)HTML Is Sensitive to a Single Whitespace Character

Any white space between characters displays as a single space. This includes all tabs, line breaks, and carriage returns. Consider this markup:

```
<strong>T e s t o f s p a c e s</strong><br>
<strong>T   e   s   t   o f   s p a c e s </strong><br>
<strong>T
e s
t o f s p          a c e s</strong><br>
```

As shown here, all the spaces, tabs, and returns are collapsed to a single element:

T e s t o f s p a c e s
T e s t o f s p a c e s
T e s t o f s p a c e s

However, it is possible to force the whitespace issue. If more spaces are required, it is possible to use the nonbreaking space entity, or **&nbsp;**

Further note that in some situations, (X)HTML does treat whitespace characters differently. In the case of the **pre** element, which defines a preformatted block of text, white space is preserved rather than ignored because the content is considered preformatted. It is also possible to use the CSS property white-space to change default whitespace handling

(X)HTML Follows a Content Model

All forms of markup support a content model that specifies that certain elements are supposed to occur only within other elements. For example, markup like this

```
<ul>
  <p>What a simple way to break the content model!</p>
        </ul>
```

Elements Should Have Close Tags Unless Empty

Under traditional HTML, some elements have optional close tags. For example, both of the paragraphs here are allowed, although the second one is better:

<**p**>This isn't closed

<p>This is**</p>**

A few elements, like the horizontal rule (**hr**) and line break (**br**), do not have close tags because they do not enclose any content. These are considered empty elements and can be used as is in traditional HTML. However, under XHTML you must always close tags, so you would have to write **<br></br>** or, more commonly, use a self-closing tag format with a final "/" character, like so: **<br />**.

Unused Elements May Minimize

Sometimes tags may not appear to have any effect in a document. Consider, for example, the **<p>** tag, which specifies a paragraph. As a block tag, it induces a return by default, but when used repeatedly, like so,

<p></p><p></p><p></p>

Elements Should Nest

A simple rule states that tags should nest, not cross; thus

<b><i>is in error as tags cross**</b></i>**

whereas

<b><i>is not since tags nest**</i></b>**

and thus is syntactically correct.

Entities Should Be Used for Special Characters

Markup parsers are sensitive to special characters used for the markup itself, like < and >. Instead of writing these potentially parse-dangerous characters in the document, they should be escaped out using a character entity. For example, instead of <, use **&lt;** or the numeric equivalent **&#60;**. Instead of >, use **&gt;** or **&#62;**. Given that the ampersand character has special meaning in an entity, it would need to be escaped as well using **&amp;** or **&#38;**.

Browsers Ignore Unknown Attributes and Elements

For better or worse, keep in mind that browsers will ignore unknown elements and attributes; so,

<bogus>this text will display on screen**</bogus>**

Major Themes of (X)HTML

Logical and Physical Markup

Physical markup refers to using a markup language such as (X)HTML to make pages look a particular way; *logical markup* refers to using (X)HTML to specify the structure or meaning of content while using another technology, such as CSS, to designate the look of the page.

Physical markup is obvious; if you want to highlight something that is important to the reader, you might embolden it by enclosing it within a **<b>** tag:

<b>This is important!**</b>**

Logical markup is a little less obvious; to indicate the importance of the phrase, it should be enclosed in the logical **strong** element:

<strong>This is important.**</strong>**

Remember, the **<strong>** tag is used to say that something is important content, not to indicate how it looks. If a CSS rule were defined to say that important items should be big, red, and italic

```
<style="text/css">
  strong {font-size: xx-large; color: red; font-style: italic;}
```

</style>
confusion would not necessarily ensue, because we shouldn't have a predisposed view of what **strong** means visually.

HTML unfortunately mixes logical and physical markup thinking. Even worse, common renderings are so familiar to developers that tags that are logical are assumed physical.
The benefits of logical elements might not be obvious to those comfortable with physical markup. To understand the benefits, it's important to realize that on the Web, many browsers render things differently.
Whether you subscribe to the physical (specific) or logical (general) viewpoint, traditional HTML is neither purely physical *nor* purely logical.

The Future of Markup—Two Paths?

strict markup will likely be a bit slower than people think and probably not ideal. The sloppy syntax from the late 1990s is still with us and is likely to be so for some time. The desire to change this is strong, but so far the battle for strict markup is far from won.

XHTML: Web Page Markup XML Style

A new version of HTML called XHTML became a W3C recommendation in January 2000. you can feed a browser just about anything and it will render. XHTML would aim to end that. Now if you make a mistake, it should matter

Theoretically, a strictly XHTML-conforming browser shouldn't render a page at all if it doesn't conform to the standard, though this is highly unlikely to happen because browsers resort to a backward-compatibility quirks mode to display such documents. The question is, could you enforce the strict sense of XML using XHTML? The short answer is, maybe not ideally.

Interestingly, most browsers, save Internet Explorer, will not have a problem with this. Internet Explorer will treat the apparent XML acting as HTML as normal HTML markup, but if we force the issue, it will parse it as XML and then render an XML tree rather than a default rendering: In summary, if a browser really believes it is getting XML, it will enforce parsing rules and force well-formedness. Regardless of whether rules are enforced or not, without Internet Explorer rendering markup visually, it would appear that we have to deliver XHTML as standard HTML, as mentioned earlier in the chapter, which pretty much makes the move to an XML world pointless.

QUESTIONS
1. with example explain HTML syntax.
2. Discuss the structure of (X)HTML documents
3. Explain any six HTML elements
4. Briefly explain the history of markup languages.
5. Wirte a note on XHTML and HTML.
6. What is HTML? Explain the structures of HTML documents.
7. Briefly explain the First look at XHTML and XHTML.
8. Mention the rules to be followed of XHTML and briefly explain them.
9. What do you mean by Future of Markup-Two paths, explain briefly.
10. Mention the difference between HTML and XHTML.
11. Explain following tags with example i)<p> ii) <em> iii) <strong> iv)<br>
12. Explain following terms i)DTD ii)SGML iii) XML iv)W3C
13. whic are the tags can be enclosed inside head tag? Explain
14. Explain the two complementary paths for the future of markup