# Chapter 2, Structuring Web Pages with HTML5

John M. Morrison

August 10, 2017

## Contents

# 0 Introduction

When you view web pages on the Internet, you use a piece of software called a *browser*. Your browser is actually a computer in software, i. e. a *virtual computer*, which understands three languages. These are as follows.

- **HTML** The acronym stands for *Hypertext Markup Language*. This gives web pages their structure. It describes such things as document sections, paragraphs, lists, tables, the placement of illustrations, and hypertext links. All of these exists inside portions of a document called *elements*. HTML is a markup language with a fairly simple grammar that understood by the browser. The browser uses HTML to format and display a document. We will be using HTML5, which is the current standard.
- **CSS** This acronym stands for *Cascading Style Sheets*. Style sheets determine the appearance of web pages. They control such things as page layout, colors, and fonts; to wit, they govern the *appearance* of a web page. CSS works with HTML and controls the style of various elements of your document. One style sheet can control the appearance of a whole group of documents, giving them a consistent appearance.
- **JavaScript** This is a full-featured computer language, unlike its friends HTML and CSS. It gives web pages their *behavior*. JavaScript can cause web pages to change their appearance or structure in response to user input. This chapter will focus on the first two languages. After this chapter will will learn the basic constructs of programming in JavaScript, and use it to create dynamic web pages.

Throughout we will use the Firefox or Chrome browsers; it is best to install both on your machine. You should install these browsers and you should also install the Web Developer and Firebug addons; these will help you to explore debug your HTML and CSS code and to ensure that it is grammatically correct. You will also be able to view the CSS from other people's web sites, and check the validity of any CSS files you create. Chrome also has a very nice array of web development tools. These are quite nice as well and will do many of the things that Firefox's tools do. It is a good idea to have both installed and to experiment with both, since they are both very widely used.

- Download Firefox here: `http://www.mozilla.com`
- Get the addons here `http://addons.mozilla.com`; you will need to use the search window to find them.
- You can download Chrome here: `https://www.google.com/intl/en-US/chrome/browser/`

# 1 The User Experience, Explained

To view sites on the Internet, you must have some kind of Internet connection. At your home or school, you will need to use an ISP (internet service provider) to gain access to the Internet. Your browser works via this connection.

The Internet is organized in a manner entirely analogous to that of a computer's file system. It is a file system that is spread across millions of computers that are connected to each other worldwide. In a computer file system, you specify the location of a file using a path. The internet works similarly; its paths are called URLs, or *uniform resource locators.* All resources on the Internet (web pages, media, etc) have a unique URL that gives their location. Computers on the internet that offer items for you to browse are called *servers.* Your computer, when you are viewing websites, is called a *client* on the internet.

Just as with file systems, URLs can be absolute or relative. Absolute URLS begin with a prefix of the form `foo://`. The most common prefixes are `http://`, `https://` and `ftp://`. Relative URLs often occur on web pages; these are relative to the page you are viewing. They will have no prefix. The prefix `http://` means *Hypertext Transfer Protocol.* The text in an HTML document is called hypertext; this name owes to the presence of links to other documents in HTML; the additional `s` in `https://` means "secure;" you will often see this on sites that require a password and login. The prefix `ftp://` demarcates *file transfer protocol*; this is used download files from servers in any format.

## 1.1 What's Behind the User Experience?

Here is a rough description of what happens when you go to a website. Suppose you start Firefox and type the URL

```
http://www.ncssm.edu/~morrison
```

into the URL window at the top of your browser. You then hit the ENTER key to get things started and.....

1. When you surf the web, your computer, the client, is the seeker of data on the web. Via your Internet connection, your computer contacts the *host* or *server* `www.ncssm.edu` and makes an *HTTP request* for the page there belonging to user `morrison`.
2. The host `www.ncssm.edu` sends back the contents of the *index page* for the site. The most common name for an index page is `index.html` or `index.php`. The response from the server is handled by a program called a *web server*; this software is aware of the name of the index page. The Apache Server is the most common web server used today; information about it can be found at [1] The browser will also download any CSS or

JavaScript present on the page. If media are to be displayed, these are downloaded, too.

3. Your browser interprets the HTML and CSS and formats the page. Any JavaScript present is loaded into the browser process's memory. The page is displayed in the content pane of your browser.
4. If you click on interactive elements on the page, the JavaScript on them runs in your browser on your machine. If a modification occurs on the page, the page itself is not changed, rather the change occurs on the copy of the page your browser has downloaded; the server's page is not changed by JavaScript. You will often, for that reason, hear JavaScript described as a *client-side language*. If you click on a link to another page, the target page loads and the process begins anew.

## 2 Setting up a Website

You will need access to a Linux server capable of hosting a website to have your materials display on the web. However, you can do this offline and see one on your local machine. We will go through the process for both.

Begin by making a directory to hold your site. If you are using a server, the most common name for this directory is `public_html`. You can check with the server's administrator; Apache allows the administrator of your server to choose any name, but almost all sites use `public_html`. If you are using your own machine, create a directory with this name to hold your web page. This directory is the root of your *public subtree*. You can create directories inside of `public_html`; in fact this is a good way to organize a site containing several parts. Each of these directories should have an index file as well, which displays by default when the directory is visited. It should cause the contents of the directory to be linked, so they can be viewed by visitors to your site.

We will use the name `public_html` throughout to designate the root of your public subtree. This directory must go right inside of your home directory.

If you are using a server, make sure the permission for your home directory is 711, which allows it to be seen through, and that the permission of your `public_html` is 755. You want the world to be able to read your web page, of course. The following commands will do this, and leave you back in `public_html`.

```
$ cd
$ chmod 711 .
$ chmod 755 public_html
$ cd public_html
```

Now you will need an index page. Again, the name of this depends on how

Apache is configured, but it is most commonly called `index.html`. Place the text shown below in the file; it is a minimal HTML5 document. Also, make a copy of this file and call it `shell.html`. You can copy this shell as you make other pages. Doing so will save you some a lot of typing.

```
<!DOCTYPE html>
<html>
<head>
<title>
My first Page
</title>
<meta charset="utf-8"/>
</head>
<body>
<p>Hello</p>
</body>
</html>
```

Make sure that the all HTML pages have permission level 644 so they can be read by the world. This can be done quickly using

```
$ chmod 644 *.html
```

This command will make all files in your current working directory with extension `.html` visible to the world.

Finally we will see how to view the website. If you are just seeing it on your local machine, use the File menu in the browser and select Open File. Then browse to the index file to see view it. If you are on a server, your URL will look like this

```
http://www.ncssm.edu/~morrison
```

Use your server's name instead of `www.ncssm.edu` and your user name instead of `morrison`. Do not omit the tilde ˜. This gives you the *base URL* for your page. You will see a white page with the word "Hello" on it. In the top of the tab or the title bar of the browser you will see the title "My First Page."

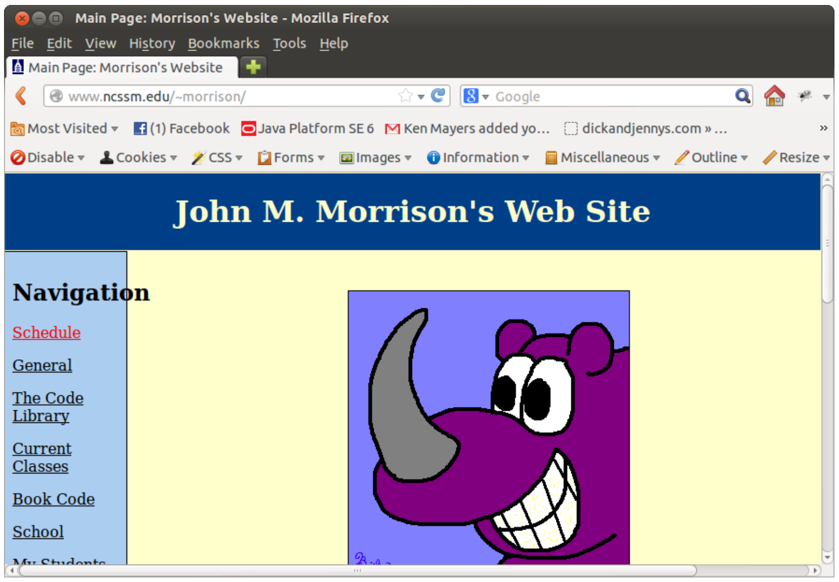
You can place an index page in any subdirectory of `public_html`. The URL of this page can be found by appending the relative path from `public_html` to your directory to the base URL of your page. This index page should link the contents of the directory so they can be viewed.

Where should you be now? You should be able to see your very unfancy web page. At this point, you are not expected to know anything about HTML or its grammar. That will come soon. But first, we will get your browser ready do develop your website, then we will begin in earnest.

## 2.1 Preparing your Browser

If you have not already done so, install the Firefox browser which is available at `http://www.mozilla.org`. Open Firefox and go to the Mozilla Addons Site `http://addons.mozilla.org`. Obtain the addons WebDeveloper and Firebug and install them. Type these names in the search bar there and click through the instructions to install them. Then restart Firefox.

Your window will look something like this


Main Page: Morrison's Website - Mozilla Firefox
File Edit View History Bookmarks Tools Help
Main Page: Morrison's Website
www.ncssm.edu/~morrison/
Google
Most Visited (1) Facebook Java Platform SE 6 Ken Mayers added yo... dickandjennys.com » ...
Disable Cookies CSS Forms Images Information Miscellaneous Outline Resize
John M. Morrison's Web Site
Navigation
Schedule
General
The Code Library
Current Classes
Book Code
School

You will see a new toolbar just below the URL window and Google search window contains a variety of menus. These include the following items.

- CSS
- Outline
- Resize
- Tools
- View Source
- Options

In the upper-right hand corner of your browser window next to the little house you will see a beetle icon. That is Firebug. If you click on the bug, it opens a console at the bottom that allows you to explore HTML in a document.

To get to Chrome's tools, open Chrome. In the upper right hand corner you will see an icon that looks like this: ≡. Click on this and it opens a menu.

Select the sub-menu marked “More Tools”, then select “Developer tools.” The developer console will open in your browser window.

Now we are ready to study HTML and CSS.

## 3 What’s in the index file?

Let us begin by looking at the contents of the index file and learning about what they do. The header

```
<!DOCTYPE html>
```

specifies the document’s type. This document is an HTML5 document. It will enable the W3C validator, which you shall meet soon, to check your HTML for correctness. The line

```
<html>
```

begins your document. Now let’s look at the rest of it.

```
<head>                      This begins the document’s head.
<title>                     This begins the document’s title.
My first Page               The title is "My first page".
</title>                    This ends the title.
<meta charset="utf-8"/>     This says we are using ASCII
                            characters.
</head>                     This ends the head.
<body>                      This begins the body.
<p>Hello</p>                Here is all the document text in a
                            paragraph.
</body>                     This ends the body.
</html>                     This ends the document.
```

## 4 The Parts of Speech in HTML

Now we will explain what we are seeing. HTML is, is what is called a *markup language*; it specifies the structure of a document. It is a very simple language and we will soon learn its grammar and its parts of speech. We can use it to “talk” to the browser and get it to display text, images, and other items in its content pane.

We shall now look at the parts of speech in HTML and how they relate to each other. The most basic part of speech in HTML is the *tag*.

Text forms the nouns of HTML. It is flowed onto a web page. All text in HTML is bounded by tags.

Tags are tokens that have the form `<foo>`, where `foo` is is a group of alphanumeric characters. Tags come in three types: opening tags, closing tags and self-closing tags. Here is a simple field guide. In each case here `tagname` is called the *type* of the tag.

1. Opening tags look like this: `<tagname>`. You can see that `<head>` is an opening tag.
2. Closing tags look like this: `</tagname>`. The string inside of the closing tag always begins with a slash. You can see that `</html>` is a closing tag.
3. Self-closing tags look like this: `<tagname/>`. Notice that they always end with a slash. An example from our little document is

   ```
   \texttt{<meta charset="utf-8"/>}
   ```

   The type of this tag is `meta`; the `charset="utf-8"` is called an `attribute`; in this case, the attribute is saying, “Use the ASCII character set.” Notice that, in an attribute, we do not put spaces around the `=` sign. This is a nearly universally-observed style convention.

Grammatically, tags are verbs in imperative form. Open tags say “Begin ...!” Closing tags say, “Stop ...!”, and self-closing tags say “Do ... right now!”

An opening tag and a closing tag *match* if they have the same type. Material between matching tags is called the *element* bounded by the tags. Self-closing tags bound *empty elements*. The purpose of tags is to *delimit* elements; i.e., they tell where elements begin and end.

Text by itself is an element. In the example we saw the line

```
<p>Hello</p>
```

The text `Hello` is an element all by itself. It is an odd exception because there is no tag specifying the end of the text element. Text elements can be thought of as self-beginning and self-ending. When the text runs out, that is it. The beginning or end of text is always attended by some other tag. In this example a paragraph tag does that job. Text elements must always occur inside of some other element. They should never be placed directly into the body of a document.

Every HTML tag has a set of default properties. For instance, the `body` tag by default makes all text black and left-justified, and the background white. You can use *attributes* to modify the behaviour of elements bounded by tags. Attributes attached to a tag are in force inside of that tag’s element. You can have several attributes as shown here. Note that the *value* attached to each attribute is always inside of quote marks.

```
<tagname attribute1="blah" attribute2="ugh"...>stuff</tag>
```

Attributes behave like adverbs that modify the action indicated by the imperative conveyed by the tag. Notice that the value attached to each attribute must be enclosed in quotes. You may use single or double quotes, but you must use the same type of quote on both ends of the attribute value. Closing tags must not contain any attributes; only opening or self-closing tags may have attributes.

## 4.1 Some Examples of Tags

Now let us look at our very simple HTML document. Line numbers have been added here for convenience.

```
1  <!DOCTYPE html>
2  <html>
3  <head>
4  <title>
5  My first Page
6  </title>
7  <meta charset="utf-8"/>
8  </head>
9  <body>
10 <p>Hello</p>
11 </body>
12 </html>
```

Line 2 contains the first actual tag. This is the tag `<html>` which demarcates the beginning of the document. Its matching tag, `</html>` occurs on line 12. The `html` element contains the entire document, save for the `DOCTYPE` declaration. Anything outside of the `html` element will not be seen by the browser.

Line 3 begins the head of the document and line 8 ends it. This document contains one self-closing, the `meta` tag.

Try inserting this after line 10 and before lines 11 and 12.

```
<p><img src="http://www2.ncssm.edu/~morrison/rhino.jpg"
alt="rhino picture"/></p>
```

What we have here is image tag inside of a paragraph tag; the `img` tag places an image on the page. If the image is not available the `alt` text is displayed instead. The `alt` text is also used by screen readers to describe images to blind computer users. Using this attribute makes your page accessible, and it is a standard on the Web. It also avoids the ugly little icon with the red X in it you often see on the web. Images are like text, they cannot be naked inside of the body; you should place them inside another element, such as a paragraph element.

If you are working locally, you can navigate to the page with the rhino image, download it, save it as `rhino.jpg`, and place it in the same directory as your HTML file. Display it using the following code.

```
<p><img src="rhino.jpg" alt="rhino picture"/></p>
```

Notice the slash at the end of a self-closing tag.

**Programming Exercises**

1. Place an image tag inside of this paragraph element.

   `<p style="text-align:center"></p> and see it get centered on your page.`

   How do you think you can left-justify or right-justify the image?
2. Add this attribute to the image tag, `width = "50%`. What happens?
3. Now replace the width's value with `200px` then `400px`. What happens?

## 5 Proper Nesting

Observing these rules will make our pages render correctly. They determine the hierarchy among your page's elements.

Every opening tag must be matched with a closing tag of the same type. This way, the opening and closing tags bound the element corresponding to that tag. Otherwise, the element "leaks" out of the tag and the browser has to figure out what you are doing. This, at the least, interferes with the browser's efficiency, and at worst, produces unanticipated errors in the rendering of your document.

**Note.** The browser is happy to make choices for you. Sadly, these will often be crappy choices. Using proper HTML tells the browser *exactly* what do do. Do not allow the browser to make choices, or you will only live a short time to regret it.

Self-closing tags close themselves, so you do not have to worry about matching for them. Since they bound empty elements, you do not have to worry about bounding the element inside of a self-closing tag.

The *proper nesting rule* says that you must close tags in the reverse order that they were opened. This ensures that your document is *well–formed*, and helps the user's browser to format your page efficiently to the screen.

Another way to say this is: Two elements are disjoint, i.e., they do not intersect at all, or one element is completely inside of the other. Elements cannot overlap any other way.

Tags should consist solely of lower-case letters or numbers. You will see upper-case tags on some older pages. Do not do this. Following these simple rules will make your page load faster and work better.

## 5.1 Exploring HTML files

We begin by using Firebug to explore your document.

With your index page open, click on the Firebug icon in the lower right-hand corner of your browser window. Now click on the HTML tab in Firebug. This allows you to explore the structure of your document. You will see elements of your document listed in the window. Some will be preceded by a little box with a + or - in it. Clicking on + opens that element for inspection; clicking on - closes it. Close the `html` element and you will only see it remaining. Now click on its + box to expand it. On the next line, you see the `head` element. Open it. Inside you can see the title element displayed with its contents, `"My First Page"`.

Now let us introduce a grammatical error and see how Firebug handles it. Switch the closing tags for `title` and `head` so they close in the wrong order. Reload the page. It still looks the same. Now open the `head` element. Look inside the `title` element. You will see that the browser has turned your illegal closing `head` tag gray; this signifies that the browser has chosen to ignore that tag. It then corrected your error and you see a closing `head` tag in the correct place. Despite our incorrect HTML, the browser renders the document correctly and interprets it correctly. Now remove the error, save your index page, and reload the document. You will see that peace is restored to the kingdom.

Next, let us explore the body. Open it; you will see that its contents consist of a single paragraph with the word “Hello” in it. The firebug indentation makes it clear that the `p`(paragraph) element lives inside the `body` element, which lives inside the `html` element. The `html` element is the it root element, since it contains all other elements on the page.

As you build your web pages, use Firebug to explore the hierarchy of your documents. It can help you to avert errors. You should add some more elements to your page, save it, and explore them with Firebug.

## 5.2 Validating Pages

You will also want to use the validator to check the grammar on your page. To do this, go under the Tools menu in the WebDeveloper toolbar. If you have installed WebDeveloper and cannot see this toolbar, click on the View menu, select Toolbars, and check the WebDeveloper toolbar in the sub-menu.

If you are using Chrome, click on the green checkmark to the right of the URL window to launch the validator.

The Tools menu is now in the WebDeveloper toolbar is now visible. Under it, select Validate HTML. A new tab will open. A green banner shows if your page validates. Otherwise, a red banner shows. If this happens, scroll down and look at the error messages.

Go to the first error message. It will indicate that the fault lies on a particular line. Use your text editor to go to that line in your HTML file.

You should look on and around that line. Fix the first error message's problem. If the next couple of errors are easily fixable, fix them too. If not, it's time to revalidate. On revalidating, repeat the process until the document validates.

You should take a document you know validates, place errors in it and look at the error messages. This way, you will become familiar with how the messaging works and you will more easily debug your document. This ability to read and debug with error messages is a very valuable one to a programmer; it pays to gain a lot of skill at it.

There is a web-based validator at [5] that allows you to validate by address, entering HTML directly, or by file upload. You should familiarize yourself with it.

## 6 Block Level and Inline Elements

HTML has two types of elements, *block-level* and *inline* elements. Block-level elements may go directly into the body of a document. The paragraph element bounded by the `<p>` tag, is a block-level element. It determines the vertical structure of your page. Inline elements must occur inside of block-level elements. An example of this is the `img` tag. Text elements are inline elements. Inline elements describe texticographical portions of your documents.

Tags that appear in the `<head>` element do not admit to these classifications; they are metadata (data about) the document and they specify document properties, including such things as the document's title, any style sheets linked to the document, and character set being used.

As we introduce new tags, we will specify if they are top-level (block) tags or inline tags.

## 7 Expanding Your HTML Vocabulary

Now put this code into the body of your document.

```
<h1>Hello</h1>
<h2 style="text-align:right">Hello</h2>
<h3 style="text-align:center">Hello</h3>
<h4 style="text-align:left">Hello</h4>
<h5>Hello</h4>
<h6>Hello</h6>
```

The tags `h1`–`h6` produce *headline text*; by default, this text is bold and left-justified. The larger the number, the smaller the text. The headline text elements are all block-level. The text `Hello` appear in each headline element above is an inline element.

The `style` attribute indicates that you are using a *local style sheet* this affects the display of the text. Also notice that when a tag closes, the attributes given it are forgotten. This occurs because the attributes of a tag are only in force inside of that tag's element.

If you omit a closing tag, you will get a "leak:" the effect of the tag will leak beyond the point where you intended it to stop. Remember the *Robert Fulghum Rule*: if you open a tag make sure you close it when you are done! Remember, self-closing tags close themselves so you don't have to worry about closing them. You can validate your page as you create it. This makes it easier to extirpate errors and keep your page valid. The validator will tell you where to look if your HTML is not valid.

Let's add a paragraph of text and make some of the text bold. Bold text is produced by the `<strong>` tag in its default guise. We will also add italics with the `<em>` tag. Observe how the closing tag tells the browser where to start and stop bold-facing and italicizing text. Append this code to the headline text you placed in the body of your page.

```
<p>Here is some text that says something <strong>very important
</strong> <em>Never</em> overdo changes in font.</p>

<p>Notice how paragraphs are begun and ended by using open and
close paragraph tags.  <strong>Always</strong> close your
paragraphs.  This way, your intent is clear and your paragraphs
will render cleanly.  Failing to close paragraphs will result
in nastygrams from the validator.</p>
```

Observe that the `<strong>` and `<em>` tags shown here are inline elements.

Now let's make our first link to another page. To make a link, you use the inline tag `<a>`; the `a` stands for "anchor." The anchor tag has an attribute called href, which stands for "hypertext reference." To make a link to Google, enter this text.

```
<p>Here is a link to <a href =
"http://www.google.com">Google</a>.</p>
```

The contents of the `<a>` element form the visible link text. We put this example in a paragraph by itself. You can put links anywhere in a text element. They can reside alone in a paragraph or they can be embedded in text. Since anchor tags are in inline element, they may not appear directly in the body of a document.

By default the browser displays the link text underlined in the familiar blue. If you fail to close the anchor tag, the link text will leak onto the rest of your document; `vi`'s syntax coloring will make leaks clear to you. To see this add a couple of sentences to the text and remove the closing tag.

The quote-enclosed value assigned to `href` is a URL. This URL can also be a file name for a file located in your `public_html` directory, or any of its subdirectories.

You can link to any file the `public_html` subtree of your file system by using an absolute path or a path relative to the location of the page you are linking from. If your site gets fairly large, you should organize it into several directories, each of which an contain further directories and other files. Each directory you create in your `public_html` subtree should have an `index.html` file.

You now have a small HTML vocabulary, which you should seek to expand. We will return to look at tables and lists, after we learn about style sheets.

1. Mozilla Tag Reference, [3] When using this list, avoid using non-standard, deprecated, and unimplemented tags. Click on the link for each tag and it will give you details on proper usage. Make sure you note if the tag is self-closing and if it is, use the `/>` to end it. Some attributes alter appearance; avoid these. Later we will learn how to use CSS to achieve any effects you want for formatting and color.
2. W3Schools, [6] has lots of useful tutorials. The Try It Editor provides a nice sandbox for experimentation.
3. Be a top dog with this site [2]. It seems to be an up-and-coming place for standards-compliant HTML. There is a very nice book that accompanies the site.

## 8 Sniffing Around for More, and How to Avoid the Bad Kids on the Block

If you are looking at a website and see something interesting, use the View Source feature to see its HTML. You can attempt to reverse-engineer it and replicate it for yourself.

Remember, however, it's a jungle out there and there are evil influences that will violate standards. We list some here in a rogue's gallery and tell you how to achieve the desired effect correctly. Many of these tags attempt to style a document in HTML. Don't do this. Remember: HTML is about structure, CSS is about appearance. Keep them separate.

- The `<u>` tags has been used for underlining text. This just confuses that text with a link. Do not do this. The effect can be achieved with CSS, but it clearly should be avoided.
- The `<big>` makes the text inside bigger. Clearly this is presentational. Use the `font-size` property in CSS instead. The same goes for `<small>`.
- The `<blink>` and `<marquee>` tags are just too vile to countenance. Just don't.
- The `<font>` tag is presentational. Use CSS.
- The `<center>` tag is presentational. Use `text-align:center` instead in CSS.

### 8.1 I'm Innocent, Don't Frame Me!

You will see pages out there that use frames. These make a hash of things. They confuse search engines (You want your page seen, don't you?), prevent people from bookmarking stuff on your site the find useful, and stymie screen readers used by people with limited vision. That makes your page less accessible. You can tell by the tenor of this paragraph that frames are a bad idea. They are obviated by CSS.

### 8.2 Son, you have some undesirable attributes

Another rogue's gallery is presented here; instead of tags, these are attributes you should not use. Most of this stuff can be done properly by using CSS. Again, we are trying to maintain the separation of style and structure.

- The `name` attribute is supplanted by `id`. It only now appears in form elements.
- The `text` attribute is supplanted the style command `color:someColor;`.

- The `bgcolor` attribute in the `body` tag is supplanted the style command `background-color:someColor;`.
- The `background` attribute in the `body` tag is supplanted the style command `background-image:url(someFileName);`.
- The `align` attribute is clearly presentational. Use CSS.
- The `target` opens a new window on a user's browser; this is a breach of netiquette. It renders the back button inaccessible. Users can place your link in a new window themselves by right-clicking. You should leave this choice to the user.
- The `body` attributes `alink`, `vlink`, and `link` are all designed to control the colors of link text. This is clearly styling. In CSS, use `a:visited`, `a:active`, and `a:link` instead.

The page [7] maintains a list of deprecated elements, and tell you the preferred way of accomplishing what those deprecated elements did. You should have a look at it.

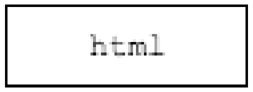
## 9 The Document Tree

Key to understanding how the style sheets of the next section work is an understanding of the document tree. Well-formed HTML documents have a tree structure that the browser uses to format them. This is a rooted tree, and it has an appearance entirely similar to that of your file system.

Let us begin with a super simple HTML file.

```
<html>
</html>
```

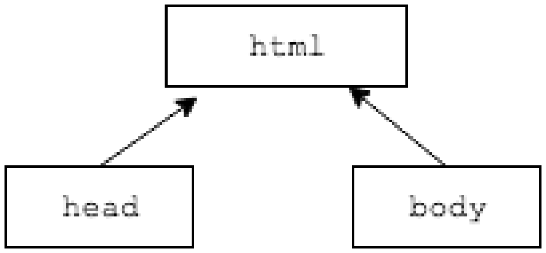
The tree just has a lonely root.



Now let is install a head and a body.

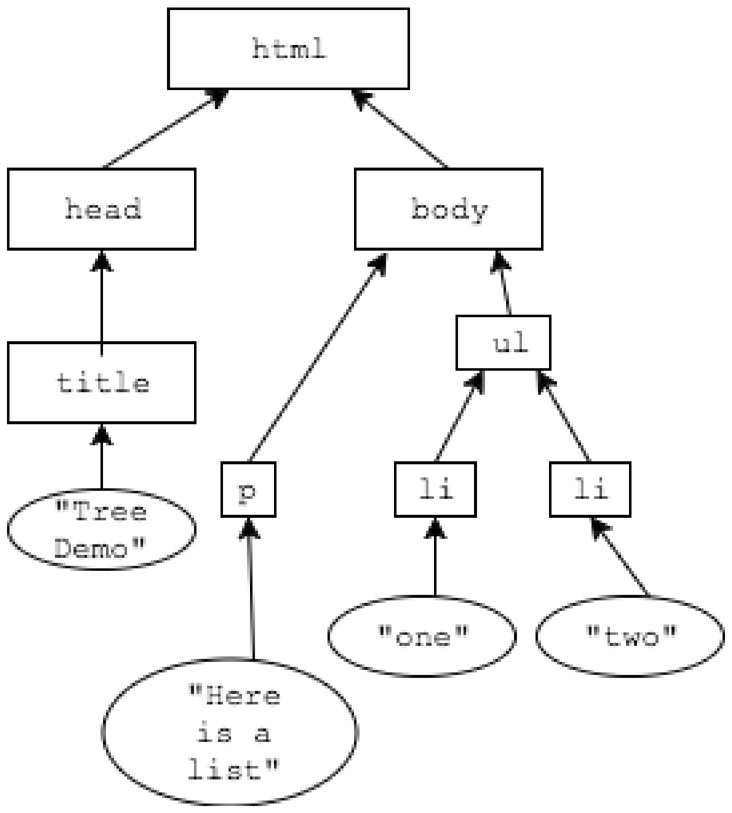
```
<html>
<head>
</head>
<body>
</body>
</html>
```

Here is what happens to the tree; the root node now has two (sibling) children.



Now we will put in a little content.

```
<html>
<head>
<title>Tree Demo</title>
</head>
<body>
<p>Here is a list</p>
<ul>
    <li>one</li>
    <li>two</li>
</ul>
</body>
</html>
```



Here is what we have. The `html` element is the root containing the entire documeent. Its children are the `head` and `body` elements. Inside of the `title` element is a *text element*, shown in an oval with its text content. You can see a similar thing in the body elelement.

Now you ask, "Why is this tree structure important?" One reason is that it completely reveals the structure of a document. It shows which elements are inside of other elements. All of the tag nodes (inside of rectangles) actually store any attributes for that element.

When we develop CSS and JavaScript, we will locate elements in a document using this tree and we will change their appearance using CSS or change the tree itself using JavaScript. It is absolutely critical to understand how the document tree works, for all that follows depends mightily upon it.

**Exercise**

1. Create an HTML file and draw its document tree.
2. Draw a document tree and create the HTML file from it. In the diagram we just studied, here is what to do. Start in the `html` block. This is the

<html>. Find the left child (head) and go there. This is <head>. Now enter the title element. This is <title> Inside of the title element is a text element. Put that text ine file and return immediately to title; note that text elements are self-terminating.
Now exit the title element; this yields </title>. By now you should have the idea. Use the tree to generate the entire document. Make your own file and do this yourself.

**References** The Net Ninja videos, [4], present the basics of HTML in a clear, succinct way. Videos 1-13 cover the material of this chapter and add a few new things in for you.

## References

[1] Apache Foundation. Apache website. http://www.apache.org.

[2] Patrick Griffiths. Html dog. https://htmldog.com.

[3] Mozilla. Mozilla tag reference. https://developer.mozilla.org/en/HTML/Element.

[4] Net Ninja. The net ninja html. https://www.youtube.com/playlist?list=PL4cUxeGkcC9ibZ2TSBaGGNrgh4ZgYE6Cc.

[5] Sister Mary Rapaknuckle. Html5 validator. https://html5.validator.nu.

[6] Inc. Refsnes Data. Firefox web browser. http://www.w3schools.com.

[7] w3schools.com. Html element reference. https://www.w3schools.com/TAGs/default.asp.