

An Introduction to HTML

More Tags, CSS, Frames, and Forms

by

Elizabeth T. Hobbs

<i>More Tags</i>	3
<i>Cascading Style Sheets</i>	5
Document-Level Style Sheets	5
External Style Sheets.....	9
Cascading Styles	9
A Few Final Points about CSS	11
<i>Frames</i>	12
Frame Document.....	12
Contents of the Frames	14
No Frames	16
<i>Forms</i>	18
Form Tag	18
Input	18
Cause Something to Happen	21
Form Example	21
<i>Resources</i>	23
Books for learning HTML.....	23
HTML Reference Book.....	23
CSS Reference Book	23
Web Sites.....	23

More Tags

There are several tags you can use to do some formatting. Let's say you want to put a list of steps in your page, with each step numbered. Use the tag, which stands for Ordered List. Tag each item in your list with , for List Item.

```
<ol>  
    <li>Do this.  
    <li>Now do this.  
    <li>Do this next.  
</ol>
```

An Ordered List

And here is what the above list looks like in the browser:

1. Do This
2. Now Do this
3. Do This Next

The List Rendered in the Browser

One good thing about this arrangement is that you can add items in the middle and the browser will renumber them automatically.

If you just want bulleted items, use the (Unordered List) instead. Note that you can specify your bullet type in the tag. The default is round, but if you like square instead, use

```
<LI type=square>
```

Another useful tag is the <BLOCKQUOTE> tag. If you have a section of text you want to indent, just surround it in the <BLOCKQUOTE> </BLOCKQUOTE> tags.

```
<p>Here is some text that is normally indented and goes across the entire  
width of the browser window.  
</p>  
<blockquote>  
    But this text is indented on both sides to make it set off from the rest of the  
text.  
</blockquote>
```

The Blockquote Tag

And, this is how the above looks in a very narrow browser window:

Here is some text that is normally
indented and goes across the entire
width of the browser window.

But this text is indented
on both sides to make it
set off from the rest of the
text.

Using the Blockquote Tag

Yet another useful tag set is the three used to create definition lists, `<DL>`, `<DT>`, and `<DD>`. These three work together to produce a nicely formatted list, with the definition indented from the term being defined. For example,

```
<DL>
<DT>Amethyst
<DD>A purple-colored quartz
<DT>Obsidian
<DD>Black, glassy igneous rock
</DL>
```

Definition List Tags

The term being defined goes after `<DT>` and the definition after `<DD>`. The result looks like this:

```
Amethyst
  A purple-colored quartz
Obsidian
  Black, glassy igneous rock
```

Example of a Definition List

Cascading Style Sheets

Once you have familiarized yourself (and had fun) with tags like to change text styles, colors, and sizes, you should start using Cascading Style Sheets (CSS) to manage the look of your HTML pages. CSS provides a way to define a style and then apply that style where you want. It also makes managing a collection of web pages easier. You define the style once in one place and you only need to update in one place should you decide to make modifications.

You can define styles in three ways. The first is inline, where you define the style inside the tag. The style is active for that tag only. This use is limited and rather defeats the purpose of defining a style in the first place – namely, using the style in many places. The second way is to define the style at the top of the web page. This style definition is only available for that page. The third method is to define the style in an external file and include that file in each web page where you want the style to be used.

We'll start by defining some styles inside a single page and then describe how to put them in an external file and use them in multiple pages.

Document-Level Style Sheets

You define a style by putting it inside the <STYLE> and </STYLE> tags. You can also put comments around your style definitions so that older browsers will ignore the style. The style section **MUST** go in the HEAD of the document.

Styles have a specific format.

```
tag-selector {property: value }
```

There are a number of rules that describe the tag-selector. We'll stick to the simplest ones for this introduction. At its simplest, the tag-selector is the name of an HTML tag that you want to apply the style to.

There are 53 properties that let you control how the browser presents your web page. The properties fall into six groups – font, color and background, text, boxes and layout, lists, and tag classification. Not all properties can be used to define the look of all HTML tags. And, for this discussion, we'll stick to the most common properties.

Here are some common properties: color, font-family, font-weight, font-style, font-size, text-indent, and text-decoration.

The value is what you want to make the property be. If the property is color, a legal value is red.

```
H1 {color: red}
```

Now, every H1 tag in the web page will be in red text.

You can set multiple properties at a time, just separate them with a semi-colon.

```
H1 {color: red; font-family: arial}
```

Now, all the H1 tags will be in red, Arial type. You can put the information on separate lines for increased readability.

```
H1 {  
    color: red;  
    font-family: arial  
}
```

Multi-line Style Definition

The last value doesn't need a semi-colon after it.

You can combine multiple definitions in one <STYLE> section.

```
<STYLE>  
<!--  
    H1 {color: red; font-family: arial}  
    P {font-family: verdana, arial, sans-serif}  
-->  
</STYLE>
```

Style Section in an HTML Document

In this example, all H1 tags will be in red, Arial text, but all text in P tags will be the default color (black), and either Verdana, Arial, or Sans-Serif (depending upon which font the browser can use).

Some properties can accept multiple values, as in the font-family. In this situation, the browser will choose the first value that it has. For font-family you could use an obscure font as the first value, and include a common font for those browsers that don't have the obscure one.

```
P {font-family: obscure-font, arial, sans-serif}
```

Note that with the P tag, you'll need to use the closing </p> tag, too. This example will cause the P style to be applied to the text:

`<p>Here is some text.</p>`

But this example won't.

`<p>Here is some text. <p>Here is more text.`

Here is the way to turn off underlining of all A tags in a web page:

`A {text-decoration: none}`

For more information about the properties that are available and their legal values, see the resource list at the end of this document.

In addition to using the HTML tags to name a style, you can create classes of styles. For example, let's say that you want some of your `<A>` links to be green. You can create a class of `<A>` tags, by using a dot then the name of your style class. (You pick the name.)

`A.green {color: green}`

`A.green` is a class definition. Now you can use the class in an `<A>` tag to make the text green, by saying `class=green`.

`<A class=green href="/">My green link</a>`

Of course, you could have done the same thing without styles.

`<a href="/"><font color=green>My green link</font></a>`

But what if you have 15 green links on your page and you decide you want to make them bold, or change the shade of green. If you use a `<FONT>` tag for each one, you'll have to make 15 changes. If you use the style definition, you'll only have to make 1 change.

`A.green {color: #99FF99}`

Note that this `<A>` tag will not cause the link to be green because the green class is not used.

`<A href="/">Not green</a>`

Here is another example using classes, this time for P tags.

```

<HTML><HEAD>
  <TITLE>Example</TITLE>
  <STYLE>
    <!--
      P {font-family: verdana, arial, sans-serif;
        color: blue}
      P.special {font-weight: bold; color: red; font-size: 80%; text-
indent: 3em}
    -->
  </STYLE>
</HEAD>
<BODY>
  <p>This text will be in either Verdana, Arial, or sans-serif. It
will be the color blue.</p>
  <p class=special>This text will be in the same font, but will be
bolded and in red. It will also be 80% -- and indented.</p>
  <p>More blue text will show up here.</p>
</BODY>
</HTML>

```

Example HTML Document with Paragraph Styles

In this example, we see a use of font-size, here with a value of 80%. This tells the browser to make the text 80% the size of its normal text. Besides percentages, you can use point sizes, like 12pt, words like “small”, and relative terms like “larger”. Be warned, however. IE and NN will often display these sizes differently, so you’ll have to test on both browsers to get the look you want.

The property text-indent will indent the paragraph by the value you specify. This example uses “3em”, which will be a variable sized-indent based upon the font size. Other types of lengths are “0.5cm” and “0.5in”.

And now to see what this looks like.

This text will be in either Verdana,
Arial, or sans-serif. It will be the
color blue.

**This text will be in the same font,
but will be bolded and in red. It will also
be 80% -- and indented.**

More blue text will show up here.

Paragraph Styles Rendered in a Browser

The term “cascading” in CSS means that style definitions build on each other, sometimes overriding each other. The overriding part generally comes into play when external and document-level style definitions are used, so we’ll look at external styles next.

External Style Sheets

External styles are almost identical to internal ones. Edit a file with a .css extension. Put in your style definitions just like you would in a web page, but **without** the <STYLE> and </STYLE> tags around them.

Here is an example external style sheet, called stylesheetfile.css.

```
A {text-decoration: none}
P {font-family: verdana, arial, sans-serif; font-style: italic}
P.indent {margin-left: 3em}
```

External Style Sheet

One way to use an external style sheet is with the LINK tag. To load an external style sheet called stylesheetfile.css in the path subdirectory use the LINK tag in the **head** of your web page. For example,

```
<HTML>
  <HEAD>
    <TITLE>My document</TITLE>
    <LINK rel=stylesheet type="text/css"
      href="http://www.yourwebsite.com/path/stylesheetfile.css"
      title="My Stylesheet">
```

Including an External Style Sheet

Use the REL and TYPE attributes just as you see in the above example. For the HREF attribute, put the URL to your style sheet. You can also title your style sheet.

Cascading Styles

What happens if you define styles in an external style sheet and in a document-level style sheet? What happens if you define a style for A and for A.green?

The rules of CSS say that a document-level style overrides an external one (and an inline style overrides document-level and external styles). Let's say the external style sheet defines this rule:

```
A.green {color: green}
```

And the document-level style is this:

```
<STYLE>
  A.green {color: #99FF99}
</STYLE>
```

The use of the class green in an <A> tag will use the color #99FF99 because that style is defined in an document-level style which overrides the external style.

But what if you have two A style definitions, where one defines a style for the <A> tag and the other defines a style for A.green? For example,

```
A {text-decoration: none}
A.green {color: green}
```

Then any <A> tag that uses the green class will also have no "text decoration" (no underlining). The plain A style is considered the "parent". Any of its properties that are not overridden by a more specific A style, like A.green, will stay in effect.

Here is a little HTML snippet that creates two links, one of which is of class green.

```
<p>
Here is a <a href="/">link</a>.<br>
And here is another <a class=green href="/sub/index.html">link</a>.
</p>
```

Using Two Link Styles

And here is what they look like in the browser. Note that neither link has any underlining. The first link is the default unvisited link color; the other is green.

Here is a [link](#).
And here is another [link](/sub/index.html).

Rendered Links

A Few Final Points about CSS

You can use style definitions for many things, like positioning DIV's on a page, controlling whether something is visible or hidden, and much more. Some of these uses particularly come into play when you start writing DHTML (dynamic HTML).

IE and Netscape Navigator have some proprietary style components and values, of course. If you want your web page to look the same on both browsers, obviously you'll have to avoid the proprietary ones.

Watch the format. If you make a mistake in the format or legal property name or value, the browser may ignore your style definition completely (especially Netscape).

You can do a lot more with CSS, more than can be covered in this introduction. See the resources listed at the end of this document.

Frames

Frames allow you to break up a single page into multiple sections, each with its own HTML document. These frames can also be scrolled individually.

I'm not a huge fan of frames and don't generally use them. Also, the kind of search engines that build their directories by crawling web pages do not handle frames, so documents that appear inside of the frame will not be indexed and available for search.

However, sometimes a frame can be useful, especially for presenting a navigation selection down one side that you can scroll separately from the information. For an example, visit <http://www.art-of-adornment.com>.

Frame Document

One HTML document, the frame document, is used to control the frames. It specifies how many frames, their arrangement, and their size. This document also identifies the source from which the frames will be filled. We'll look at a simple example that has two frames.

The frames are grouped in a <FRAME> tag. It says how many rows and columns there are, plus the size of the rows and columns. Our example will just have one row with two columns; that is, the two frames will be side by side.

```
<FRAMESET cols="50%,50%">
```

Here we have two columns of equal size, each taking 50% of the available width. These columns will get smaller or larger as the window is resized.

```
<FRAMESET cols="25%,75%">
```

Now we have one narrow column and one wide one.

You can also specify specific widths.

```
<FRAMESET cols="150,500">
```

This example creates one column 150 pixels wide and another column 500 pixels wide.

You can use percentages or pixel lengths when defining rows, too.

Here part of a frame document.

```
<HTML>
<HEAD>
<TITLE>Frame example</TITLE>
</HEAD>
<FRAMESET cols="25%,75%">
</FRAMESET>
</HTML>
```

The Frameset Tag

IMPORTANT: In a document with a FRAMESET tag you do **not** use a BODY tag.

The example above sets up the layout, but we need to add the actual frames. Frames are defined using the <FRAME> tag. The most important attribute of the FRAME tag is the SRC attribute, which says where to get the HTML or image from that goes in that frame. You can also name your frame and turn off scrolling if you wish.

```
<FRAME src="/docs/mynav.html">
<FRAME src="/docs/mystuff.html" scrolling="no" name="display">
```

The Frame Tag

By default, if the browser needs to add a scroll bar to access the content of the frame, it will add one. If you don't want scroll bars, use scrolling="no".

The second frame is named "display". You can use names to specify where to load information. For example, in the navigation frame, if a person clicks on a link, the new data should show up in the other frame. By using the name "display" in the <A> tag, we can control where the data is displayed.

Finally, the order the <FRAME> tags appear in determines which frame they apply to. Start counting in the upper left-hand corner and go across row-wise. Because we only have two frames, it is easy to tell which <FRAME> tag applies to which one. The first frame tag goes with the left-hand column and the second with the right-hand column.

Here is our example, in a file called myframe.html.

```
<HTML>
<HEAD>
<TITLE>Frame example</TITLE>
</HEAD>
<FRAMESET cols="25%,75%">
    <FRAME src="/docs/mynav.html">
    <FRAME src="/docs/mystuff.html" scrolling="no" name="display">
</FRAMESET>
</HTML>
```

The Entire Frame Document

Contents of the Frames

To bring up your frame document, load the html file that contains the FRAMESET definition. In this example, the file that is loaded is myframe.html. What you will see displayed depends upon what is in the contents of each frame's HTML document.

Each frame's contents a separate HTML document and can have its own background, styles, etc. In the example, the left-hand frame will get filled with whatever is in mynav.html and the right-hand frame will get the stuff in mystuff.html.

Here are the contents of mynav.html.

```
<html>
<head>
<title>Nav</title>
</head>
<body>
<a target="display" href="doc1.html">First document</a><br>
<a target="display" href="doc2.html">Second document</a><br>
<a target="display" href="doc3.html">Third document</a><br>
</body>
</html>
```

Example Frame Contents – Navigation Frame

It is a series of links to other documents, like a table of contents. Note the use of the TARGET attribute. When the link is clicked, its document will get loaded in the frame named "display". In the example, this is the right-hand frame.

Now for the contents of mystuff.html. This gets loaded into the right-hand column when the frameset page, myframe.html, is brought up.

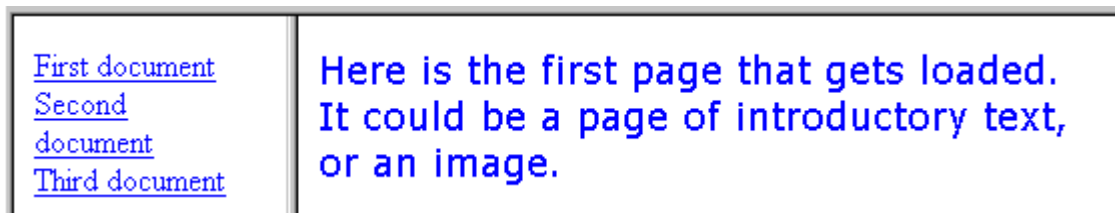
```

<html>
<head>
<title>Stuff</title>
<style>
<!--
      P {color: blue; font-family: Verdana; font-size: 120%}
-->
</style>
</head>
<body>
<p>Here is the first page that gets loaded. It could be a page of
introductory text, or an image.
</p>
</body>
</html>

```

Example Frame Contents – Display Frame

If you load up myframe.html, this is what it looks like initially. Note that text in the right-hand column uses the style defined in the <STYLE> section.



Initial Loading of myframe.html

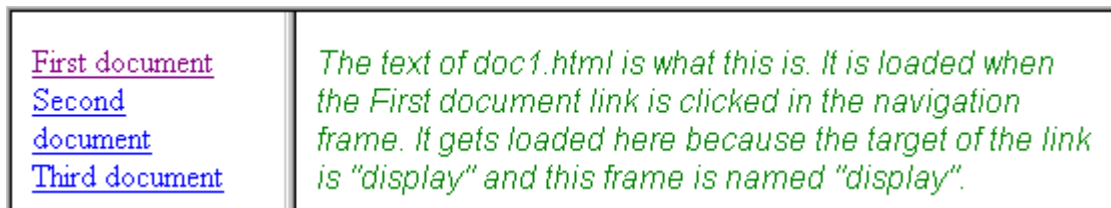
If you click on the First document link in the navigation frame, the contents of doc1.html will be loaded in the right-hand column, replacing what is already there. Remember, the reason the link will replace the contents of the right-hand column is because we named that frame “display” and we used target=”display” in the URLs in mynav.html.

Here is what is contained in doc1.html.

```
<html>
<head>
<title>Stuff</title>
<style>
<!--
      P {color: green; font-family: Arial; font-style: italic}
-->
</style>
</head>
<body>
<p>The text of doc1.html is what this is. It is loaded when the First document link
is clicked in the navigation frame. It gets loaded here because the target of the link
is "display" and this frame is named "display".
</p>
</body>
</html>
```

Contents of doc1.html

And, tah-dah:



Loading the Contents of doc1.html in the Display Frame

Note: at no time do you ever see any of the text in myframe.html. myframe.html is just the layout for the document; the contents come from the SRC attributes in the FRAME tags.

No Frames

Finally, to handle those browsers that can't process frames, you can add the <NOFRAMES> </NOFRAMES> tag pair to your FRAME document (here myframe.html). Inside of these tags, you can put whatever you want a no-frames browser to see, like "This web page requires a frames-capable browser."

Here is myframe.html with the <NOFRAMES> section added (inside of the <FRAMESET>).

```
<HTML>
<HEAD>
  <TITLE>Frame example</TITLE>
</HEAD>
<FRAMESET cols="25%,75%">
  <FRAME src="/docs/mynav.html">
  <FRAME src="/docs/mystuff.html" scrolling="no" name="display">
<NOFRAMES>
  This web page requires a frames-capable browser.
</NOFRAMES>
</FRAMESET>
</HTML>
```

Frame Document with a NOFRAMES Section

Forms

Forms are used to send information from the browser to the web site. Forms generally require a CGI program on the web site that will process the information in the form. You don't need to be a programmer to use forms, however. Many times you can install a CGI script written by someone else on your web host and write a form to send the script information.

Alternatively, you can mail a form. This method does not always work correctly, unfortunately. It depends upon how email is set up on the user's computer. Please refer to one of the HTML books listed at the end for more information on using mailto:

We'll set up a practice form without worrying about sending the information anywhere.

Forms consist of several parts – the <FORM> tag that identifies the form, tags that control the input, and buttons that cause something to happen.

Form Tag

In the FORM tag you will commonly use the METHOD and ACTION attributes. Use method=post for your forms and put the URL to the script that will handle the form. (Or use mailto:name@address.)

```
<FORM method=POST action="/cgi-bin/input.cgi">
```

End your form with the closing </FORM> tag.

Input

There are three main ways of creating items for input. One is the <INPUT> tag, the second is the <TEXTAREA> tag, and the last uses the <SELECT> and <OPTION> tags.

The <INPUT> tag has many types. For the moment we'll discuss radio buttons, checkboxes, and text. Radio buttons allow you to select one option out of several. Checkboxes allow you to select several items from a set. Text input allows you to enter a short piece of textual data. You use the TYPE attribute to identify what kind of INPUT you want.

All <INPUT> tags should be named. That is how the script will identify the input data. For inputs that go together in groups, like radio buttons and checkboxes, give all the members of the group the same name. In the case of the radio buttons, only one of the members of that group can be selected.

Most <INPUT> tags also need a VALUE. This is the value that will be sent to the CGI script if the input item is selected.

Finally, you can enter text that will be displayed to the user after the <INPUT> tag.

Now for some examples. Let's create 3 radio buttons that go together.

```
<form>
  <input type=radio name=stoplight value=red>Red<br>
  <input type=radio name=stoplight value=yellow>Yellow<br>
  <input type=radio name=stoplight value=green>Green<br>
</form>
```

Creating Radio Buttons

Note that the text contains HTML formatting (here
). The form does not do formatting. Often elements of a form are placed in a table to get precise layout.

Note also that all three of these radio buttons have the name stoplight. That identifies them as belonging to the same group. Because they are radio buttons only one of them will be selected at a time.

Checkboxes also appear in groups, but you can select more than one.

```
<form>
  <input type=checkbox name=fruit value=apple>Apple<br>
  <input type=checkbox name=fruit value=banana>Banana<br>
  <input type=checkbox name=fruit value=grape>Grape<br>
  <input type=checkbox name=fruit value=lemon>Lemon<br>
</form>
```

Creating Checkboxes

If you want to make one of the radio or checkboxes selected by default, just add the attribute CHECKED in the tag like so

```
<input type=radio checked name=stoplight value=yellow>Yellow<br>
```

If you want the user to enter text, then you use a TEXT type of <INPUT> tag. Text inputs can be given a maximum length – which is always a good idea, to prevent someone entering a ridiculously long value – as well as a display size. You can also give a default value.

This example will create a text field that is approximately 10 characters wide with a maximum length of 50 characters. It has no default value.

```
<INPUT type=text name=choice size=10 maxlength=50>
```

This example sets a default value.

```
<INPUT type=text name=choice size=10 maxlength=50 value="paper">
```

The other type of input is the `<TEXTAREA>` tag. This tag allows a person to enter as much text as they want. You specify the name of the text area (which is how it will be identified to the script) and the number of columns and rows the text area will take up on the screen.

```
<TEXTAREA name=story cols=50 rows=10></TEXTAREA>
```

Don't forget the closing `</TEXTAREA>` tag.

The last input allows you to create a selection list. You use the `<SELECT>` tag to group the items in the selection list and to control the name and size of the list. The `<OPTION>` tag identifies each option in the list.

The `<SELECT>` tag has a few attributes of particular importance. The name identifies the name of the list to the script. The size tells the browser how many elements to display in the selection window (the browser will add scroll bars as needed). If you use the attribute `multiple`, you can select multiple items in the list.

The `<OPTION>` tag will allow you to specify a value. If you don't, it will use the value you type after the option tag. You can also tell the browser if an item is pre-selected by using the `SELECTED` attribute.

This example will let you select multiple desserts. The selection window will be long enough to show 2 items. The browser will automatically add scroll bars.

```
<SELECT name=dessert size=2 multiple>
  <OPTION value=chcake>Chocolate cake
  <OPTION value=apie>Apple pie
  <OPTION value=cpie>Cherry pie
  <OPTION value=sccake>Strawberry Cheesecake
  <OPTION value=truff>Truffle
</SELECT>
```

Creating a Selection List

Note that you don't do any formatting except size here. The browser will format this into a selection list. Also, the entire `<SELECT>` goes inside the form.

Cause Something to Happen

The last thing you need is a button for the user to click on to send the input data to the script. This is typically done with another `<INPUT>` tag, this time a SUBMIT one.

```
<INPUT type=submit>
```

You can also name submit buttons and/or give them a value. Sometimes the CGI script requires a name and value on its submit buttons.

```
<INPUT type=submit value=doit name=inputbutton>
```

If you use the VALUE attribute, that is the name the button will have on the screen. Otherwise the browser will use a default value.

Form Example

Now we'll pull all this together in a silly example. In the example, we'll use a table to give the form a simple layout. This example is just designed to show you that you can use table elements in the form to control where things are placed.

Stoplight ☐ Red ☐ Yellow ☐ Green	Fruit ☐ Apple ☐ Banana ☐ Grape ☐ Lemon
Story 	Dessert Chocolate cake Apple pie
GO	Do it

Example Form Elements in a Table

The html file follows.

```

<html><head>
<title>Form Example</title>
</head>
<body>
<form method=post action="/cgi-bin/input.cgi">
<table cellpadding=4>
  <tr>
    <th>Stoplight</th>
    <th>Fruit</th>
  </tr>
  <tr>
    <td>
      <input type=radio name=stoplight value=red>Red<br>
      <input type=radio name=stoplight value=yellow>Yellow<br>
      <input type=radio name=stoplight value=green>Green<br>
    </td>
    <td>
      <input type=checkbox name=fruit value=apple>Apple<br>
      <input type=checkbox name=fruit value=banana>Banana<br>
      <input type=checkbox name=fruit value=grape>Grape<br>
      <input type=checkbox name=fruit value=lemon>Lemon<br>
    </td>
  </tr>
  <tr>
    <th>Story</th>
    <th>Dessert</th>
  </tr>
  <tr>
    <td><TEXTAREA name=story cols=50 rows=10></TEXTAREA></td>
    <td>
      <SELECT name=dessert size=2 multiple>
        <OPTION value=chcake>Chocolate cake
        <OPTION value=apie>Apple pie
        <OPTION value=cpie>Cherry pie
        <OPTION value=sccake>Strawberry Cheesecake
        <OPTION value=truff>Truffle
      </SELECT>
    </td>
  </tr>
  <tr>
    <th>GO</th>
    <td><input type=submit value="Do it"></td>
  </tr>
</table>
</body>
</html>

```

Resources

If you want to pursue HTML and CSS further, here are some books and web sites you can look into.

Books for learning HTML

Designed for the absolute beginner and for getting you going with HTML quickly:
HTML for the World Wide Web Visual Quickstart Guide –
<http://www.amazon.com/exec/obidos/ASIN/0201354934/crazyforlife>

A very comprehensive book that teaches HTML:
HTML Black Book –
<http://www.amazon.com/exec/obidos/ASIN/1576106179/crazyforlife>

HTML Reference Book

HTML: The Definitive Guide –
<http://www.amazon.com/exec/obidos/ASIN/1565924924/crazyforlife>

CSS Reference Book

Cascading Style Sheets: The Definitive Guide –
<http://www.amazon.com/exec/obidos/ASIN/1565926226/crazyforlife>

Web Sites

Web Reference HTML Tutorials – <http://webreference.com/html/tutorials/>

Web Monkey HTML Cheatsheet –
http://hotwired.lycos.com/webmonkey/reference/html_cheatsheet/

Web Monkey Stylesheets Guide --
http://hotwired.lycos.com/webmonkey/reference/stylessheet_guide/