

Programming the Internet—Lecture Notes

Week 2: More Advanced HTML5 and CSS

Dividing Up the Web Page

We saw in last week's Lecture Notes that HTML5 has declared frames obsolete. Frames were very useful in dividing a Web page into different sections. This particular function of frames can be carried out by the `<div>` tag, which can be used to specify different divisions of the page. For practical purposes they can be regarded as rectangular boxes, which can be given borders and different coloured backgrounds if required. They can also be manipulated using the CSS Box Model, described in these Lecture Notes. CSS alternatives to the use of frames are discussed at WebPelican (2012), WebAIM (2012) and 456 Berea Street (2006).

Before we can proceed, we need to be able to have more than one *div* on the same page and to apply different styles to each *div* if required. So far, we have only been able to apply the same styles to every `<p>` or `<h1>` tag that appears on a particular page. CSS provides the ability to define an *id* or a *class*. We shall look at classes soon. What we need here is the *id* attribute. The *id* attribute can be applied to *div* tags in this case, but it can be applied to other elements. Only one *div* can use each *id* on the same Web page. It is therefore a unique identifier of a particular *div* on a particular page. In our HTML we shall say something like this:

```
<div id = "header"> </div>
<div id = "main"> </div>
```

Between the `<div>` and `</div>` tags we place any code that will determine the content to be displayed in that division of the Web page.

In the CSS we can define a style for each *div* like this (note the # sign in front of the *id*):

```
#header {
width: 100%;
}

#main {
float: right;
}
```

Here is a worked example of how to divide a page using *div* tags. It is a simple example which should act as a guide.

```
<!DOCTYPE html>
<html>
  <head>
    <title> Dividing a Page Using Div Tags </title>
    <meta charset="utf-8" />
```

```

        <link href="style1.css" type="text/css"
rel="stylesheet" />
        <link href="style2.css" type="text/css"
rel="stylesheet" />
        <link href="style4.css" type="text/css"
rel="stylesheet" />
        <link href="style6.css" type="text/css"
rel="stylesheet" />
    </head>
    <body>
        <div id = "header">
            <h1> How to Divide a Page Using Div Tags </h1>
        </div>

        <div id = "menu">
            <ol>
                <li><a href =
"http://www.w3schools.com/css/css_font.asp">Serif and Sans-
Serif </a></li>
                <li><a href =
"http://www.w3schools.com/html5/default.asp">HTML5 Tutorial
</a></li>
                <li><a href =
"http://www.w3schools.com/css/default.asp">CSS Tutorial
</a></li>
            </ol>
            <p>Etc</p>
            <p>Etc</p>
            <p>Etc</p>
            <p>Etc</p>
            <p>Etc</p>
        </div>

        <div id = "main">
            <p> This is where the main body of the text will be
displayed. </p>
        </div>

        <div id = "footer">
            <p> Less important menu items can be displayed here
horizontally. </p>
        </div>
    </body>
</html>

```

Figure 1: DivExample.html

Figures 2 and 3 provide an example of what we can do with CSS. The line in Figure 2 referring to *border-radius* will provide rounded corners (but not necessarily in older browsers). A useful reference about which browsers support the latest CSS features can be found at W3 Schools (2012).

```

#header {
color: black;
width: 90%;
margin: 20px;
padding: 5px;
border-style: solid;

```

```

border-color: blue;
border-width: 2px;
border-radius: 20px;
}

#menu {
color: white;
float: left;
clear: both;
width: 25%;
background-color: black;
margin: 20px;
padding: 5px;
}

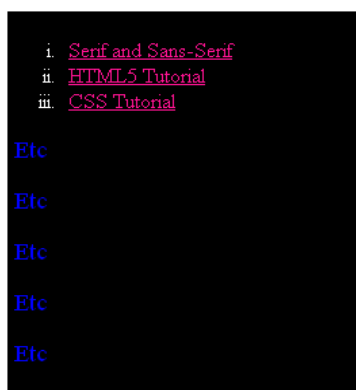
#main {
margin: 20px;
padding: 5px;
}

#footer {
width: 90%;
clear: both;
margin: 20px;
padding: 5px;
border-style: solid;
border-color: blue;
border-width: 2px;
}

```

Figure 2: style6.css

How to Divide a Page Using Div Tags



This is where the main body of the text will be displayed.

Less important menu items can be displayed here horizontally.

Figure 3: Output from DivExample.html and style6.css

Although there can only be one *div* called 'header' on a particular page, there can be a *div* with the same name on another page. This means that our style sheet could be applied to a number of pages, each of which has a *div* called 'header'. Using this method can help make Web pages look consistent with each other throughout a Web site.

There are two key CSS properties that allow us to position different divisions on a Web page. They are *float* and *clear*.

The possible values of *float* are 'left', 'right' and 'none' (the default value). If, in our CSS, we say of a *div* element:

```
float: left
```

then the *div* will be placed (floated) as far to the left of the page as the browser is able. When an element is 'floated' left, a browser will naturally try to float the next *div* that is described in the code to the right of the first floated *div*. The way to prevent this is by using *clear*. The possible values of *clear* are 'right', 'left', 'both' and 'none' (the default value). If we say in our CSS for the second *div*:

```
clear: left
```

this means that a previously floated *div* (it was floated earlier in the document) cannot appear to the left of the next *div* which has *clear: left* defined. It effectively forces the second *div* to be displayed below the first because the instruction means 'keep the area to the left of this *div* clear of any previously defined *divs*'. Setting the width of a *div* to 100% will also stop other *divs* from being displayed alongside because 100% means the full width of the page.

HTML5 introduces new tags which can be used to refer to different parts of the page. These include `<header>`, `<footer>`, `<nav>` and `<section>`. However, browser support for such tags is still not comprehensive, and hence at this stage we have restricted ourselves to describing features that will work in a wide range of browsers. They are usually referred to as 'semantic elements' because by reading the code we can get an indication of the kind of content that is contained in different sections of the page. For instance, `<nav>` (navigation) would be used for a division of the Web page containing a number of links.

Thus, you could rewrite DivExample.html as follows:

```
<!DOCTYPE html>
<html>
  <head>
    <title> Dividing a Page Using Div Tags </title>
    <meta charset="utf-8" />
    <link href="style1.css" type="text/css" rel="stylesheet" />
    <link href="style2.css" type="text/css" rel="stylesheet" />
    <link href="style4.css" type="text/css" rel="stylesheet" />
```

```

    <link href="style6a.css" type="text/css" rel="stylesheet" />
</head>
<body>
    <header>
    <h1> How to Divide a Page Using Div Tags </h1>
    </header>

    <nav>
    <ol>
    <li><a href = "http://www.w3schools.com/css/css_font.asp">Serif
and Sans-Serif </a></li>
    <li><a href =
"http://www.w3schools.com/html5/default.asp">HTML5 Tutorial </a></li>
    <li><a href = "http://www.w3schools.com/css/default.asp">CSS
Tutorial </a></li>
    </ol>
    <p>Etc</p>
    <p>Etc</p>
    <p>Etc</p>
    <p>Etc</p>
    <p>Etc</p>
    </nav>

    <section>
    <p> This is where the main body of the text will be displayed.
</p>
    </section>

    <footer>
    <p> Less important menu items can be displayed here
horizontally. </p>
    </footer>
</body>
</html>

```

Figure 4: DivExample2.html

In a browser such as Google Chrome, which supports these new HTML5 tags, the output will look exactly the same as in Figure 3. However, in IE8, which does not support these tags, this is how the page will be displayed:

How to Divide a Page Using Div Tags

- i. [Serif and Sans-Serif](#)
- ii. [HTML5 Tutorial](#)
- iii. [CSS Tutorial](#)

Etc

Etc

Etc

Etc

Etc

This is where the main body of the text will be displayed.

Less important menu items can be displayed here horizontally.

Figure 5: Output from DivExample2.html and style6a.css

Remember that Windows XP cannot support any version of Internet Explorer after IE8, so workplaces that do not upgrade their systems are unlikely to get the benefit of HTML5 support in later versions of IE. Large parts of the British Civil Service have not been able to justify the expense of upgrading from IE6, so there are difficult decisions to be made in employing the latest technology that might not be usable by many of our customers.

The CSS Box Model

CSS regards every HTML element as being in a box. Every heading, paragraph, image and division is in a box. The words and images we display are the content at the centre of the diagram below. The CSS Box Model provides a way of adjusting the spacing between the different elements on our Web page (W3C, 2012a).

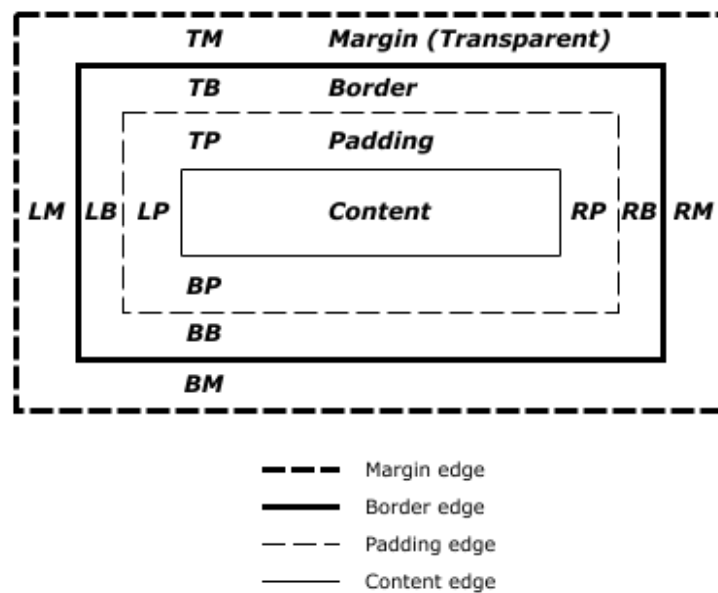


Figure 6: The CSS Box Model

A visible border can actually be drawn around many of these elements. In last week's Lecture Notes we saw an example of a border drawn around an image.

The padding is the distance between the content edge and the border. If padding = 0 (the default setting), then the content edge and the border are the same. Even in this case there will be a little space between the content (text, image, etc.) and the content edge (the box around the element). Where we say something like the following in our CSS, the same padding, expressed in pixels, will be applied to all four sides of the content.

```
h1{
padding: 10px;
}
```

We could, however, specify different padding for the four sides like this:

```
h1{
padding-top: 5px;
padding-bottom: 10px;
padding-left: 15px;
padding-right: 20px;
}
```

The margin is the distance between the current element and its neighbours. If the margin = 0 (the default setting), then the borders of neighbouring elements will touch. This will be clear if we specify a visible border or if we give elements a background colour. A background colour will be applied to the border of an element but not the margin.

Different margins can be set for the four sides in the same way as with padding.

Class Selectors in CSS

So far, we have seen how to apply styling, for example, to the content of all `<p>` tags, but what if we want to apply different styles to different paragraphs? We could use an *id* attribute as we did in the case of naming a *div* or division of the Web page. However, the *id* would be unique to one paragraph, and sometimes we might want to apply the same style to more than one paragraph (some paragraphs on a page, but not all). The answer is that we can use one or more *class* selectors.

First, in our HTML, instead of just using a plain `<p>` tag, we add a *class* attribute and a value to give a particular paragraph a unique name. For instance:

```
<p class = "first">And the paragraph's text goes here.</p>
<p class = "second">And some more text in our second paragraph</p>
```

In our CSS, we can now apply different styles to those two paragraphs:

```
p.first{
color: navy;
font-size: 14pt;
font-family: "Times New Roman", serif;
font-weight: bold;
text-align: left;
}
p.second{
color: black;
font-size: 12pt;
font-family: Arial, sans-serif;
font-style: italic;
text-align: center;
}
```

Every time we want to use the paragraph style defined in the CSS by the *class* selector “p.first,” we shall use as the opening tag in the HTML

```
<p class = "first">
```

and those CSS settings will be applied to that paragraph. By using the *class* selector in the CSS and the attribute in the HTML, the number of different styles that can be defined for one type of tag using class selectors is unlimited. A class attribute can also be applied to any number of paragraphs in the HTML. As long as different pages are pointing to the relevant external style sheet, this technique can be applied across a number of Web pages.

HTML Forms

One of the most important ways that the user can interact with a Web site is through the use of HTML forms. Here the user can make selections and enter data. In Weeks 5 and 6, we shall see how user input can result in customised content being generated by programs on the Web server. It is important for businesses to encourage potential customers to provide information about their preferences so that the Web server delivers content most suited to their needs.

So let us build up an HTML form, piece by piece, following our previous method of adding features gradually so that it will be easier to absorb what is going on.

```
<!DOCTYPE html>
<html>
  <head>
    <title> An Example of an HTML Form </title>
    <meta charset="utf-8" />
    <link href="style1.css" type="text/css" rel="stylesheet" />
  </head>
  <body>
    <h1> Tell Us What You Think </h1>
    <p> Please answer the following questions so that we can
provide a better service for you. </p>
    <form method = "post" action = "customer.php">
      <p>
        <input type = "submit" name = "submit" value =
"Submit" />
        <input type = "reset" value = "Clear" />
      </p>
    </form>
  </body>
</html>
```

Figure 7: ExampleForm.html

Tell Us What You Think

Please answer the following questions so that we can provide a better service for you.

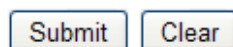
The image shows a web browser window displaying the output of the HTML form. It features a large heading "Tell Us What You Think" in a serif font. Below the heading is a paragraph of text in blue: "Please answer the following questions so that we can provide a better service for you." At the bottom of the visible area, there are two buttons: "Submit" and "Clear", both with a light blue gradient and a thin border.

Figure 8: Output from ExampleForm.html

In Figures 7 and 8, we are putting together a form that will allow the customers to provide us with information and tell us what they think. The code above is a mere skeleton of a Web page. It starts with a headline and a brief introductory paragraph. Then comes the form between the `<form>` and `</form>` tags.

The form tag has attributes called *method* and *action*, each of which can be given a value in double quotes. We shall learn more about this line of code when we look at server-side programming. It will indicate the method that the form is using to send information across the Internet to the Web server, and it will usually name in the *action* attribute a program which is to be launched on the server in order to process information sent from the form.

The two *input type* statements place two buttons on the screen, usually at the end of the form. The *submit* button will be clicked by the user to send to the Web server information and choices recorded in the form by the user. The *reset* button will clear the form so the user can start again. The *input type* names are part of HTML and cannot be changed. The *value* indicates the text that will be displayed on the button. This can be anything the programmer wants it to be.

What we now need are some HTML features to allow the customer to input data and make choices. Some features are defined by the programmer and allow users to select their choice from a defined list. Other features allow the users to create and provide information themselves. Let's start with choices that are defined by the programmer.

Radio Buttons

All of us will have used these. Now we shall learn how to create them.

```
<!DOCTYPE html>
<html>
  <head>
    <title> Radio Buttons in an HTML Form </title>
    <meta charset="utf-8" />
    <link href="style1.css" type="text/css" rel="stylesheet"
  />
</head>
<body>
  <h1> Tell Us What You Think </h1>
  <p> Please answer the following questions so that we can
provide a better service for you. </p>
  <form method = "post" action = "customer.php">
    <fieldset>
      <legend> Your company _ s Web site provides far more
information about your services than other companies do about theirs.
</legend>
      <input type = "radio" name = "SiteQuality" id = "SQ1" value =
"5" /> <label for="SQ1"> Strongly Agree </label> <br />
      <input type = "radio" name = "SiteQuality" id = "SQ2" value =
"4" /> <label for="SQ2"> Agree </label> <br />
      <input type = "radio" name = "SiteQuality" id = "SQ3" value =
"3" /> <label for="SQ3"> Neutral/Undecided </label> <br />
      <input type = "radio" name = "SiteQuality" id = "SQ4" value =
"2" /> <label for="SQ4"> Disagree </label> <br />
      <input type = "radio" name = "SiteQuality" id = "SQ5" value =
"1" /> <label for="SQ5"> Strongly Disagree </label> <br />
      <input type = "radio" name = "SiteQuality" id = "SQ6" value =
"0" checked = "checked" /> <label for="SQ6"> Question Not Answered
```

```

</label> <br />
    </fieldset>
    <p>
        <input type = "submit" name = "submit" value = "Submit" />
        <input type = "reset" value = "Clear" />
    </p>
</form>
</body>
</html>

```

Figure 9: ExampleFormRadio.html

Tell Us What You Think

Please answer the following questions so that we can provide a better service for you.

Your company's Web site provides far more information about your services than other companies do about theirs. _____

☐ Strongly Agree
☐ Agree
☐ Neutral/Undecided
☐ Disagree
☐ Strongly Disagree
☒ Question Not Answered

Figure 10: Output from ExampleFormRadio.html

This example uses radio buttons to find out what the user thinks about a statement concerning the quantity of information on the company's Web site. The respondent can choose one of five replies ranging from 'Strongly Agree' to 'Strongly Disagree'. This is known as a *Likert scale* and it is often used in questionnaires.

Each radio button has to be defined separately as being of input type "radio". The *name* attribute, which is here "SiteQuality", is the name of a set of radio buttons. It is the fact that each of these buttons has the same name that means they are regarded as a set, and that means that only one button in a set can be selected at any one time. The current selection will be deselected if we click on another button.

Although a set of radio buttons is normally only useful if all its buttons are physically adjacent to each other, as in the above example, they do not need to be. Radio buttons located in different places throughout a form will still be regarded as belonging to a set if they have the same *name*.

The *id* attribute provides us with a way of identifying a single radio button.

The *value* of the selected radio button is what will be returned to the Web server when the *submit* button is clicked. In statistical analysis of survey results, it is often useful to assign each reply a numerical value. The five main

responses have been assigned the values one to five. This will mean the researcher does not need to hand code the replies; it has already been done for him or her. The users will not see these values unless they select View → Page Source (or some similar command) in their browsers.

There is a facility to have one response selected by default (the *checked* attribute). Often, the programmer chooses the first option. The problem here is that if the user does not answer the question, the form will return the value associated with the checked option, even though this was not the user's selection. If there are many people who do not answer a question, the results will be distorted as the first option will seem to have been chosen on many occasions when this is not the case.

The solution offered here is to provide a sixth option, *Not Answered*, which is by default checked (the *checked* attribute associated with this option is given the value "checked"). Non-responses can be identified by the "0" default value returned by the form, and then the user can possibly be asked whether he or she has inadvertently overlooked this question.

Note that in Figure 9 the *input type* tag is properly terminated by a forward slash. After the `<input>` tag has been closed, we shall then find a pair of `<label>` tags at the end of each line. They enclose text such as *Strongly Agree*, and this is what is shown in the browser immediately after each radio button. It is possible to display radio buttons side by side. In this case, the `
` tags will mean each radio button will be displayed on a separate line.

The `<label>` tags use the *for* attribute to link a specific radio button with a particular label. For example, `<label for="SQ1">`, where "SQ1" is the *id* of the first radio button. Those with normal vision can usually see which label belongs to which radio button in the browser. However, the visually disabled may be relying on screen readers which turn the textual content of the Web page into speech. Where the label is explicitly associated with a particular radio button in the HTML, this increases the accessibility of our code to the visually disabled.

The `<fieldset>` tags, which enclose the set of radio buttons in Figure 9, tend to group them visually by placing a border around them. We do not have to use them, but they can improve the appearance of the page. The `<legend>` tags, which are associated with `<fieldset>`, help us to insert a heading or question which the user is being asked to answer.

Checkboxes

There are times when the programmer wants to give the user the option of making multiple choices rather than just one. This can be done by using checkboxes. In the example below, we have removed the radio buttons. The code below should be largely self-explanatory, but if it is not, do not hesitate to ask your Instructor. In this case, zero or more words are returned as values to the Web server, depending on how many checkboxes the user selects.

```

<!DOCTYPE html>
<html>
  <head>
    <title> Checkboxes in an HTML Form </title>
    <meta charset="utf-8" />
    <link href="style1.css" type="text/css" rel="stylesheet" />
  </head>
  <body>
    <h1> Tell Us What You Think </h1>
    <p> Please answer the following questions so that we can
provide a better service for you. </p>
    <form method = "post" action = "customer.php">
      <fieldset>
        <legend> Which of these do you own ? </legend>
        <input type = "checkbox" name = "ItemsOwned" id = "IO1" value =
"laptop" /> <label for="IO1"> A laptop computer </label> <br />
        <input type = "checkbox" name = "ItemsOwned" id = "IO2" value =
"blackberry" /> <label for="IO2"> A Blackberry </label> <br />
        <input type = "checkbox" name = "ItemsOwned" id = "IO3" value =
"iphone" /> <label for="IO3"> An iPhone </label> <br />
        <input type = "checkbox" name = "ItemsOwned" id = "IO4" value =
"ipad" /> <label for="IO4"> An iPad </label> <br />
        <input type = "checkbox" name = "ItemsOwned" id = "IO5" value =
"labrador" /> <label for="IO5"> A Labrador called Donald </label> <br
/>
      </fieldset>
      <p>
        <input type = "submit" name = "submit" value = "Submit" />
        <input type = "reset" value = "Clear" />
      </p>
    </form>
  </body>
</html>

```

Figure 11: ExampleFormCheckbox.html

Tell Us What You Think

Please answer the following questions so that we can provide a better service for you.

Which of these do you own ?

☐ A laptop computer
☐ A Blackberry
☐ An iPhone
☐ An iPad
☐ A Labrador called Donald

Figure 12: Output from ExampleFormCheckbox.html

Drop-down Lists/Menus

An alternative to the use of radio buttons and checkboxes for selecting from programmer-defined options is to employ drop-down lists or menus. First, an example of a menu where the user can make only one selection.

```
<!DOCTYPE html>
<html>
  <head>
    <title> Selecting One Item From a Menu in an HTML Form </title>
    <meta charset="utf-8" />
    <link href="style1.css" type="text/css" rel="stylesheet" />
  </head>
  <body>
    <h1> Tell Us What You Think </h1>
    <p> Please answer the following questions so that we can
provide a better service for you. </p>
    <form method = "post" action = "customer.php">
      <fieldset>
        <legend> Which is your favourite holiday destination? </legend>
        <select name = "country" id = "country" size = "1">
          <option value = "NotAnswered" selected = "selected"> Not
Answered </option>
          <option value = "Canada"> Canada </option>
          <option value = "Spain"> Spain </option>
          <option value = "Nigeria"> Nigeria </option>
          <option value = "Britain"> Britain </option>
          <option value = "Australia"> Australia </option>
        </select>
      </fieldset>
      <p>
        <input type = "submit" name = "submit" value = "Submit" />
        <input type = "reset" value = "Clear" />
      </p>
    </form>
  </body>
</html>
```

Figure 13: ExampleFormSelectOne.html

Tell Us What You Think

Please answer the following questions so that we can provide a better service for you.



Which is your favourite holiday destination?

Not Answered ▼

Submit Clear

Figure 14: Output from ExampleFormSelectOne.html

Our menu is enclosed in `<select>` tags. The *name* attribute specifies the name of the whole menu. The *size* attribute tells the browser how many menu options are visible to users when they first see the page. Because the *size* is 1, only one menu option is shown and a classic drop-down list may be displayed, with the other options only being revealed when the user clicks on a down arrow beside the list.

Each choice in the menu is enclosed in `<option>` tags. The use of *selected* is very similar to the use of *checked* in the radio buttons example. Here, it means that the *Not Answered* option will be highlighted, and the associated value will be returned to the Web server if the user does not make a choice from the menu.

In a second example, we shall create a menu where several choices are possible by providing a *multiple* attribute with the value "multiple". This may seem like unnecessary duplication, but HTML insists that all values are specified explicitly and not just implied. The user makes second and subsequent selections by holding down the CTRL key while clicking on the menu. This is not intuitively obvious to all users; hence, many programmers prefer to use checkboxes for multiple selections. If you use the *multiple* attribute, it may be worthwhile including instructions on how it is selected in the `<legend>` tag. Because the *size* is 6 and there are only 6 options, all the menu options will be visible from the start.

```
<!DOCTYPE html>
<html>
  <head>
    <title> Selecting Several Items From a Menu in an HTML Form
  </title>
    <meta charset="utf-8" />
    <link href="style1.css" type="text/css" rel="stylesheet" />
  </head>
  <body>
    <h1> Tell Us What You Think </h1>
    <p> Please answer the following questions so that we can
provide a better service for you. </p>
    <form method = "post" action = "customer.php">
      <fieldset>
        <legend> Which of these countries have you visited? </legend>
        <label for="country">Please select a country.</label><br />
        <select name = "country" id = "country" size = "6" multiple =
"multiple">
          <option value = "NotAnswered" selected = "selected"> Not
Answered </option>
          <option value = "Canada"> Canada </option>
          <option value = "Spain"> Spain </option>
          <option value = "Nigeria"> Nigeria </option>
          <option value = "Britain"> Britain </option>
          <option value = "Australia"> Australia </option>
        </select>
      </fieldset>
      <p>
        <input type = "submit" name = "submit" value = "Submit" />
        <input type = "reset" value = "Clear" />
      </p>
    </form>
  </body>
</html>
```



```

        </p>
    </form>
</body>
</html>

```

Figure 15: ExampleFormSelectMany.html

Tell Us What You Think

Please answer the following questions so that we can provide a better service for you.

Which of these countries have you visited?

Please select a country.

Not Answered
 Canada
 Spain
 Nigeria
 Britain
 Australia

Submit
 Clear

Figure 16: Output from ExampleFormSelectMany.html

Text Boxes

Often, we want to let our users speak for themselves rather than forcing them to choose from predefined options. Text boxes are useful for inputting small amounts of text such as names and addresses. Here is an example:

```

<!DOCTYPE html>
<html>
  <head>
    <title> Textboxes in an HTML Form </title>
    <meta charset="utf-8" />
    <link href="style1.css" type="text/css" rel="stylesheet" />
  </head>
  <body>
    <h1> Tell Us What You Think </h1>
    <p> Please answer the following questions so that we can
provide a better service for you. </p>
    <form method = "post" action = "customer.php">
      <fieldset>
        <legend> Please provide your contact details. </legend>
        <label for="surname">Type your surname.</label><br />
        <input name = "surname" id = "surname" type = "text" size =
"20" maxlength = "30" /><br />
        <label for="forenames">Type all your forenames.</label><br />
        <input name = "forenames" id = "forenames" type = "text" size =
"30" maxlength = "40" /><br />
        <label for="email">Type your email address.</label><br />
        <input name = "email" id = "email" type = "text" size = "30"

```



```

maxlength = "40" /><br />
    </fieldset>
    <p>
        <input type = "submit" name = "submit" value = "Submit" />
        <input type = "reset" value = "Clear" />
    </p>
</form>
</body>
</html>

```

Figure 17: ExampleFormTextbox.html

Tell Us What You Think

Please answer the following questions so that we can provide a better service for you.

Please provide your contact details.

Type your surname.

Type all your forenames.

Type your email address.

Submit Clear

Figure 18: Output from ExampleFormTextbox.html

The text box is identified by the word *input* (note that the element is terminated by a forward slash). This HTML element has a *name* and an *id*, which can both have the same value, and its input is defined as being 'text.' The *size* is measured in characters. This is the width of the text box visible on screen (the visible window). When the user types beyond this size limit, the cursor will scroll right and the leftmost characters will temporarily disappear to make way for the newly typed text. This will only happen if the *maxlength* of the input has not been reached. If it has, then the user will be able to type no more text. Select the *size* and *maxlength* attribute values to allow for reasonable maximum limits.

If the user is being asked to type a password and we want only asterisks to be displayed as he or she types, then change *type* = "text" to *type* = "password".

Where a larger area is required for more extensive text input, use the `<textarea>` element instead. An example would be:

```
<textarea name = "feedback" rows = "10" cols = "50">
```

The attribute *rows* represents the number of rows of text that will be allowed in the textarea (the height). The *cols* attribute is short for columns and is equivalent to the number of characters that can be accommodated on one row (the width).

Buttons

The Submit and Reset buttons are provided as standard facilities for HTML forms, but there will be times when we want to define our own buttons within a form. This is accomplished easily enough as follows:

```
<button type="button">Register Now</button>
```

The text between the `<button>` tags can be anything the programmer wants. These words will appear on the button. When we come to look at JavaScript, we shall learn to link a particular event with the clicking of a button.

Tables

A table is a very useful way to set out statistical data in rows and columns. Before Web developers made great efforts to separate content and presentation, tables were widely used to determine page layout. Now, it is typically regarded as best practice to avoid this. Use `<div>` tags plus CSS instead.

Here is an example of how a table could be implemented in HTML. It introduces a little logic that we shall often come across in learning programming. Here, *p* and *q* are both propositions, i.e. statements which are either true or false. When we join them with a logical *AND*, e.g. *The moon is made of green cheese* (*p*) *AND* *British trains always run on time* (*q*), then the combined statement is only true if both *p* and *q* are true. This can be expressed in a truth table.

```
<!DOCTYPE html>
<html>
  <head>
    <title> HTML Tables </title>
    <meta charset="utf-8" />
    <link href="style7.css" type="text/css" rel="stylesheet"
  />
  </head>
  <body>
    <table id = "logicaland" >
      <caption><strong>Table 1:The Logical
AND</strong></caption>
      <thead>
        <tr>
          <th>p</th>
          <th>q</th>
          <th>p AND q</th>
        </tr>
```

```

</thead>
<tfoot>
  <tr>
    <th>p</th>
    <th>q</th>
    <th>p AND q</th>
  </tr>
</tfoot>
<tbody>
  <tr class = "alt">
    <td>True</td>
    <td>True</td>
    <td>True</td>
  </tr>
  <tr>
    <td>True</td>
    <td>False</td>
    <td>False</td>
  </tr>
  <tr class = "alt">
    <td>False</td>
    <td>True</td>
    <td>False</td>
  </tr>
  <tr>
    <td>False</td>
    <td>False</td>
    <td>False</td>
  </tr>
</tbody>
</table>
</body>
</html>

```

Figure 19: TableExample.html

After a bit of CSS styling (which we shall look at shortly), it will be displayed as follows:

Table 1: The Logical AND

p	q	p AND q
True	True	True
True	False	False
False	True	False
False	False	False
p	q	p AND q

Figure 20: Output from TableExample.html and style7.css

Here are the main tags:

Tags `<table>` and `</table>` mark the beginning and the end of the table. The opening tag contains an attribute called an *id*, so that we can distinguish this table from any others on the Web page.

The tag `<caption>` displays text which appears just above the table. It is a useful place to put a heading.

Tags `<thead>` and `</thead>` are typically used for defining the column headings in the top row of the table. Inside the `<thead>` tags, we use `<tr>` and `</tr>` (table row) to define what data is in this row of the table. Tags `<th>` and `</th>` then describe the actual contents of the table header cells. Although the three sets of `<th>` items are set out one underneath the other, they actually describe from left to right what is in cells (columns) on the same row of the table.

Tags `<tfoot>` and `</tfoot>` define the table footer, which is the information on the bottom row of the table. In this case, it is the same as the table header data, but it could contain data such as totals and averages.

The main body of the table is contained in `<tbody>` and `</tbody>` tags. Here we describe the contents one row at a time within `<tr>` and `</tr>` tags. This time, the contents of each row are contained within `<td>` and `</td>` tags. These are table data cells, and as we read down we see the contents of the row (three columns) described from left to right.

Below are just some suggestions about what could be done with CSS to style our table. Refer to Figure 20 to see what the output looks like when `style7.css` is applied.

```
#logicaland{
font-family: Helvetica, sans-serif;
width: 50%;
border-collapse: collapse;
background-color: white;
}
```

```
#logicaland caption{
text-align: left;
font-size: 12pt;
padding-top: 25px;
}
```

```
#logicaland td, #logicaland th{
text-align: center;
font-size: 14pt;
border: 1px solid black;
padding: 2px 2px 2px 2px;
}
```

```
#logicaland tr.alt td{
```

```
background-color: aqua;
}

#logicaland thead, #logicaland tfoot{
background-color: silver;
}
```

Figure 21: style7.css

Here we are using an *id* selector to identify this particular table called *logicaland*, just as we did in naming *div* elements. Note the # sign in front of the table *id*. Remember that the difference between an *id* selector and a *class* selector is that only one element with a particular *id* can appear on a Web page. An *id* is unique but a *class* is not. We want only one table with this name to appear on this page, so *id* is a good choice. We may want several paragraphs with a particular *class* name to appear on a page (because they have similar styling), so *class* was a good choice for our `<p>` tags.

It is good practice to start at the most general (highest) level in defining a style for the whole table. The alignment of text to the left or centre within a cell of the table is governed by *text-align*. The *border-collapse* style ensures that where there is a border around each cell, the lower border of the cell above and the upper border of the cell below are not forming a double-thickness line. We are telling two lines to ‘collapse’ into a single line.

In the next section, the *padding-top* facility decides how much space there is between the top of the table (in this case the *caption*) and the HTML content that is above it.

Notice that it is possible to apply styles to more than one tag as in *#logicaland td*, *#logicaland th*. Anything in this section will apply to all table data cells and all table header cells. Placing a solid border of 1 pixel width around each cell also means there will be a border around the whole table. The padding in this section of the style sheet says that there will be 2 pixels’ space between the text in a table cell and the table cell border on all four sides.

Why Are Style Sheets Cascading?

We can use the example of setting background colours in our table to explain how the word *cascading* applies to CSS. HTML consists of a series of tags inside tags. In terms of page content, we have `<body>` tags and then tags such as `<table>` tags inside them and then `<thead>` inside the `<table>` tags and `<th>` tags inside the `<thead>` tags. We might think of this as a hierarchy, with `<body>` towards the top and the other elements below it in a kind of family tree structure.

CSS tries to define styles that will be used throughout the document near the top of the hierarchy. In our example of styling a table called *#logicaland*, we start by applying styles to the whole table. By default, these styling features are applied to all the tags within the table. These features are said to

'cascade' down to the other tags in the table. In programming, it is more usual to talk of elements 'inheriting' properties from items further up the hierarchy.

In our table example, we have defined the background colour as being 'white' at the *#logicaland* (table) level. That means that if we do not overrule this setting elsewhere, the background of every cell will be white. However, towards the end of our CSS file, we do overrule this setting. At *#logicaland thead*, *#logicaland tfoot*, we specify that the background for the table header and table footer will be 'silver.' The white background which cascades from the styles defined at the table level will be replaced by the silver background defined for these specific tags.

The other example at *#logicaland tr.alt td* is a little more complicated. What we are trying to do here is to give alternate (odd numbered) lines in our table different background colours so that it is easier to read across a line of data. This is achieved by first specifying a class attribute in the HTML on every other row of the table:

```
<tr class = "alt">
```

Then, in the CSS, *#logicaland tr.alt td* means something like the following: Apply this style to every `<td>` tag of the table *#logicaland* that is inside a `<tr>` tag that has the class *alt*. In this case, we are specifying that alternate rows have the background colour 'aqua.' Because we have not overruled the background colour of the other rows set at the table level, the rows in between the aqua rows will have a white background.

In some of the coding examples we have linked our HTML code to more than one style sheet. It has been useful for teaching purposes to divide our CSS up into smaller chunks, but developers will often use one big CSS file. In the case where we point our HTML file at two or more CSS files, and they directly contradict each other at the same level, the later files will overrule the earlier ones. For instance, if we said in the first CSS file we listed that all text within `<p>` tags was to be blue (*color: blue*) and in the last CSS file we listed, we said all text within `<p>` tags should be black (*color: black*), then the browser would obey the later instruction and show text in black.

References

456 Berea Street (2006) *CSS frames v2, full-height* [Online]. Available from http://www.456bereastreet.com/archive/200609/css_frames_v2_fullheight/ (Accessed: 2 August 2012).

W3C (2012a) *Box model* [Online]. Available from <http://www.w3.org/TR/CSS2/box.html> (Accessed: 2 August 2012)

W3 Schools (2010a) *CSS3 browser support reference* [Online]. Available from http://www.w3schools.com/cssref/css3_browsersupport.asp (Accessed: 2 August 2012).

WebAim (2012) *Alternatives to frames* [Online]. Available from <http://webaim.org/techniques/frames/#alternatives> (Accessed: 2 August 2012).

WebPelican (2012) *CSS frames example/tutorial* [Online]. Available from <http://www.webpelican.com/web-tutorials/css-frames-tutorial/> (Accessed: 2 August 2012).