

HTML TABLES AND FORMS

2.1 INTRODUCING TABLES

A table in HTML is created using the `<table>` element and can be used to represent information that exists in a two-dimensional grid. Tables can be used to display calendars, financial data, pricing tables, and many other types of data. Just like a real-world table, an HTML table can contain any type of data: not just numbers, but text, images, forms, even other tables.

Basic Table Structure

An HTML `<table>` contains any number of rows (`<tr>`); each row contains any number of table data cells (`<td>`). Some browsers do not by default display borders for the table; however, we can do so via CSS.

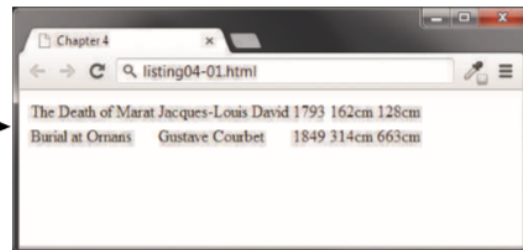
`<table>`

<code><tr></code>	The Death of Marat <code><td></code>	Jacques-Louis David <code><td></code>	1793 <code><td></code>	162cm <code><td></code>	128cm <code><td></code>
<code><tr></code>	Burial at Ornans <code><td></code>	Gustave Courbet <code><td></code>	1849 <code><td></code>	314cm <code><td></code>	663cm <code><td></code>

```

<table>
  <tr>
    <td>The Death of Marat</td>
    <td>Jacques-Louis David</td>
    <td>1793</td>
    <td>162cm</td>
    <td>128cm</td>
  </tr>
  <tr>
    <td>Burial at Ornans</td>
    <td>Gustave Courbet</td>
    <td>1849</td>
    <td>314cm</td>
    <td>663cm</td>
  </tr>
</table>

```



Many tables will contain some type of headings in the first row. In HTML, you indicate header data by using the `<th>` instead of the `<td>` element. Browsers tend to make the content within a `<th>` element bold, but you could style it anyway you would like via CSS. The main reason you should use the `<th>` element is not, however, due to presentation reasons. Rather, you should also use the `<th>` element for accessibility reasons and for search engine optimization reasons.

Spanning Rows and Columns

If you want a given cell to cover several columns or rows, then you can do so by using the colspan or rowspan attributes.

Title	Artist	Year	Size (width x height)	
The Death of Marat	Jacques-Louis David	1793	162cm	128cm
Burial at Ornans	Gustave Courbet	1849	314cm	663cm

```

<table>
<tr>
<th>Title</th>
<th>Artist</th>
<th>Year</th>
<th colspan="2">Size (width x height)</th>
</tr>
<tr>
<td>The Death of Marat</td>
<td>Jacques-Louis David</td>
<td>1793</td>
<td>162cm</td>
<td>128cm</td>
</tr>
<tr>
<td>Burial at Ornans</td>
<td>Gustave Courbet</td>
<td>1849</td>
<td>314cm</td>
<td>663cm</td>
</tr>
...
</table>

```

Notice that this row now only has four cell elements.

Fig: Spanning Column

Artist	Title	Year
Jacques-Louis David	The Death of Marat	1793
	The Intervention of the Sabine Women	1799
	Napoleon Crossing the Alps	1800

```

<table>
<tr>
<th>Artist</th>
<th>Title</th>
<th>Year</th>
</tr>
<tr>
<td rowspan="3">Jacques-Louis David</td>
<td>The Death of Marat</td>
<td>1793</td>
</tr>
<tr>
<td>The Intervention of the Sabine Women</td>
<td>1799</td>
</tr>
<tr>
<td>Napoleon Crossing the Alps</td>
<td>1800</td>
</tr>
...
</table>

```

Notice that these two rows now only have two cell elements.

Fig: Spanning Row

Additional Table Elements

The `<caption>` element is used to provide a brief title or description of the table, which improves the accessibility of the table, and is strongly recommended.

The `<thead>`, `<tfoot>`, and `<tbody>` elements tend in practice to be used quite infrequently. However, they do make some sense for tables with a large number of rows.

The `<col>` and `<colgroup>` elements are also mainly used to aid in the eventual styling of the table. Rather than styling each column, you can style all columns within a `<colgroup>` with just a single style.

A title for the table is good for accessibility.

These describe our columns, and can be used to aid in styling.

Table header could potentially also include other `<tr>` elements.

Yes, the table footer comes *before* the body.

Potentially, with styling the browser can scroll this information, while keeping the header and footer fixed in place.

```

<table>
  <caption>19th Century French Paintings</caption>
  <col class="artistName" />
  <colgroup id="paintingColumns">
    <col />
    <col />
  </colgroup>

  <thead>
    <tr>
      <th>Title</th>
      <th>Artist</th>
      <th>Year</th>
    </tr>
  </thead>

  <tfoot>
    <tr>
      <td colspan="2">Total Number of Paintings</td>
      <td>2</td>
    </tr>
  </tfoot>

  <tbody>
    <tr>
      <td>The Death of Marat</td>
      <td>Jacques-Louis David</td>
      <td>1793</td>
    </tr>
    <tr>
      <td>Burial at Ornans</td>
      <td>Gustave Courbet</td>
      <td>1849</td>
    </tr>
  </tbody>
</table>

```

Title	Artist	Year
The Death of Marat	Jacques-Louis David	1793
Burial at Ornans	Gustave Courbet	1849
Total Number of Paintings		2

Using Tables For Layout

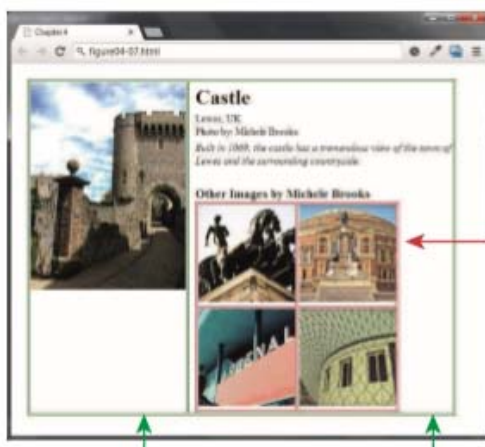
Prior to the broad support for CSS in browsers, HTML tables were frequently used to create page layouts. Since HTML block-level elements exist on their own line, tables were embraced by developers in the 1990s as a way to get block-level HTML elements to sit side by side on the same line.

The disadvantages of using tables for layout are as given below

- Unfortunately, this practice of using tables for layout had some problems. The first of these problems is that this approach tended to dramatically increase the size of the HTML document. These larger files take longer to download, but more importantly, were often more difficult to maintain because of the extra markup.
- A second problem with using tables for markup is that the resulting markup is not semantic.
- The other key problem is that using tables for layout results in a page that is not accessible, meaning that for users who rely on software to voice the content, table-based layouts can be extremely uncomfortable and confusing to understand.



```
<table>
<tr>
  <td>
    
  </td>
  <td>
    <h2>Castle</h2>
    <p>Lewes, UK</p>
    <p>Photo by: Michele Brooks</p>
    <p>Built in 1069, the castle has a tremendous
      view of the town of Lewes and the
      surrounding countryside.
    </p>
  </td>
</tr>
</table>
```



```
<h3>Other Images by Michele Brooks</h3>

<table>
<tr>
  <td></td>
  <td></td>
</tr>
<tr>
  <td></td>
  <td></td>
</tr>
</table>
</td>
</tr>
</table>
```

2.2 STYLING TABLES

There is certainly no limit to the way one can style a table. While most of the styling that one can do within a table is simply a matter of using the CSS properties, there are a few properties unique to styling tables that needs to be discussed.

Table Borders

Borders can be assigned to both the `<table>` and the `<td>` element (they can also be assigned to the `<th>` element as well). Interestingly, borders cannot be assigned to the `<tr>`, `<thead>`, `<tfoot>`, and `<tbody>` elements.



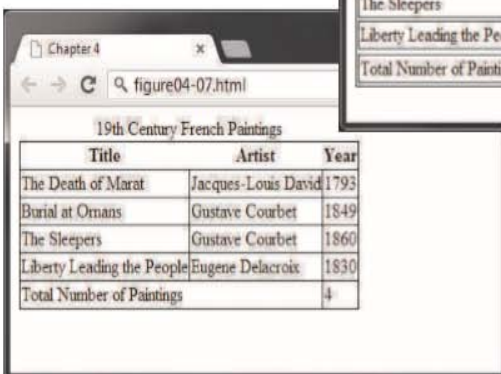
Title	Artist	Year
The Death of Marat	Jacques-Louis David	1793
Burial at Ornans	Gustave Courbet	1849
The Sleepers	Gustave Courbet	1860
Liberty Leading the People	Eugene Delacroix	1830
Total Number of Paintings	4	

```
table {
  border: solid 1pt black;
}
```



Title	Artist	Year
The Death of Marat	Jacques-Louis David	1793
Burial at Ornans	Gustave Courbet	1849
The Sleepers	Gustave Courbet	1860
Liberty Leading the People	Eugene Delacroix	1830
Total Number of Paintings	4	

```
table {
  border: solid 1pt black;
}
td {
  border: solid 1pt black;
}
```



Title	Artist	Year
The Death of Marat	Jacques-Louis David	1793
Burial at Ornans	Gustave Courbet	1849
The Sleepers	Gustave Courbet	1860
Liberty Leading the People	Eugene Delacroix	1830
Total Number of Paintings	4	

```
table {
  border: solid 1pt black;
  border-collapse: collapse;
}
td {
  border: solid 1pt black;
}
```

Notice as well the border-collapse property. This property selects the table's border model. The default, shown in the second screen, is the separated model or value. In this approach, each cell has its own unique borders. You can adjust the space between these adjacent borders via the border-spacing property, as shown in the final screen capture. In the third screen capture, the collapsed border model is being used; in this model adjacent cells share a single border

Title	Artist	Year
The Death of Marat	Jacques-Louis David	1793
Burial at Ornans	Gustave Courbet	1849
The Sleepers	Gustave Courbet	1860
Liberty Leading the People	Eugene Delacroix	1830
Total Number of Paintings	4	

```
table {
  border: solid 1pt black;
  border-collapse: collapse;
}
td {
  border: solid 1pt black;
  padding: 10pt;
}
```

Title	Artist	Year
The Death of Marat	Jacques-Louis David	1793
Burial at Ornans	Gustave Courbet	1849
The Sleepers	Gustave Courbet	1860
Liberty Leading the People	Eugene Delacroix	1830
Total Number of Paintings	4	

```
table {
  border: solid 1pt black;
  border-spacing: 10pt;
}
td {
  border: solid 1pt black;
}
```

Boxes and Zebras

While there is almost no end to the different ways one can style a table, there are a number of pretty common approaches. We will look at two of them here. The first of these is a box format, in which we simply apply background colors and borders in various ways. We can then add special styling to the :hover pseudo-class of the <tr> element, to highlight a row when the mouse cursor hovers over a cell. That figure also illustrates how the pseudo-element nth-child can be used to alternate the format of every second row.

Title	Artist	Year
The Death of Marat	Jacques-Louis David	1793
Burial at Ornans	Gustave Courbet	1849
The Sleepers	Gustave Courbet	1860
Liberty Leading the People	Eugene Delacroix	1830
Mademoiselle Caroline Riviere	Jean-Auguste-Dominique Ingres	1806

```
tbody tr:hover {
  background-color: #9e9e9e;
  color: black;
}
```

Title	Artist	Year
The Death of Marat	Jacques-Louis David	1793
Burial at Ornans	Gustave Courbet	1849
The Sleepers	Gustave Courbet	1860
Liberty Leading the People	Eugene Delacroix	1830
Mademoiselle Caroline Riviere	Jean-Auguste-Dominique Ingres	1806

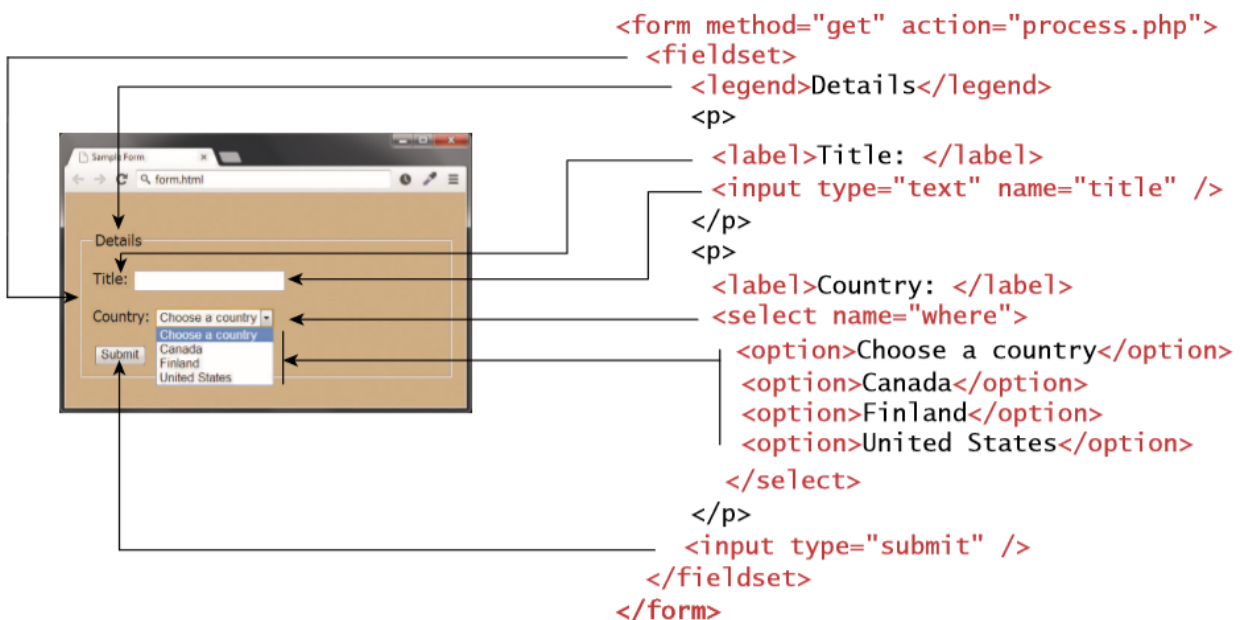
```
tbody tr:nth-child(odd) {
  background-color: white;
}
```


2.3 INTRODUCING FORMS

Forms provide the user with an alternative way to interact with a web server. Typically, programs running on the server will take the input from HTML forms and do something with it, such as save it in a database, interact with an external web service, or customize subsequent HTML based on that input.

Form Structure

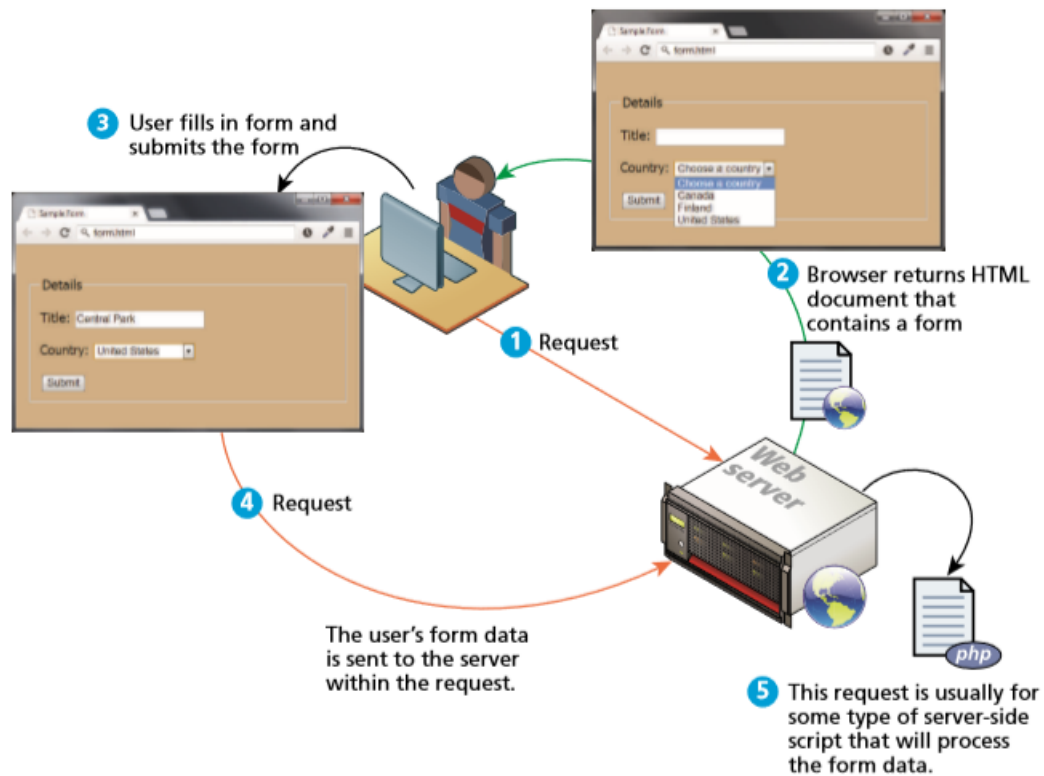
A form is defined by a `<form>` element, which is a container for other elements that represent the various input elements within the form as well as plain text and almost any other HTML element.



How Forms Work

While forms are constructed with HTML elements, a form also requires some type of server-side resource that processes the user's form input.

The process begins with a request for an HTML page that contains some type of form on it. This could be something as complex as a user registration form or as simple as a search box. After the user fills out the form, there needs to be some mechanism for submitting the form data back to the server. This is typically achieved via a submit button, but through JavaScript, it is possible to submit form data using some other type of mechanism. Because interaction between the browser and the web server is governed by the HTTP protocol, the form data must be sent to the server via a standard HTTP request. This request is typically some type of server-side program that will process the form data in some way; this could include checking it for validity, storing it in a database, or sending it in an email.



Query Strings

The browser packages the user's data input into something called a query string. A query string is a series of name=value pairs separated by ampersands (the & character).

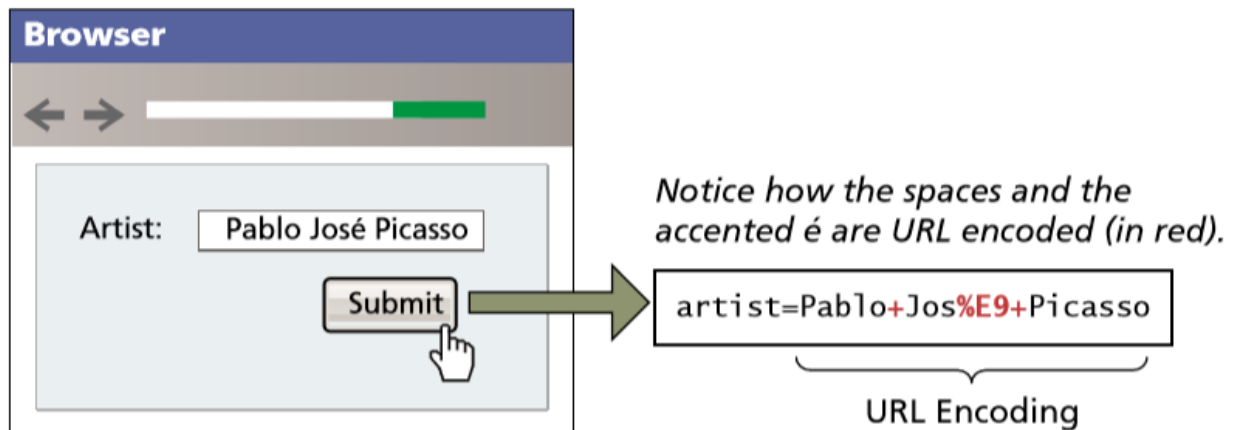
```
<input type="text" name="title" />
```

The screenshot shows a web browser window with a form titled "Sample Form". The form has two input fields: "Title" with the value "Central Park" and "Country" with the value "United States". There is a "Submit" button at the bottom.

```
title=Central+Park&where=United+States
```

```
<select name="where">
```

Query strings have certain rules defined by the HTTP protocol. Certain characters such as spaces, punctuation symbols, and foreign characters cannot be part of a query string. Instead, such special symbols must be URL encoded (also called percent encoded).



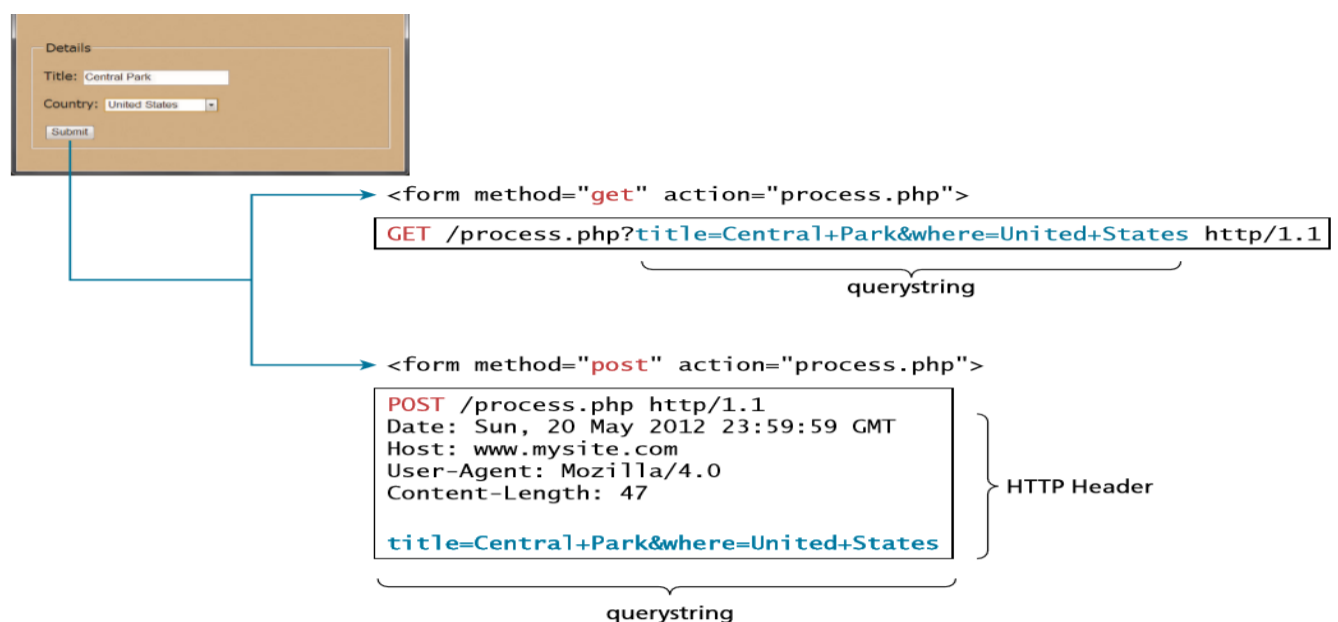
The Form <form> Element

There are two important attributes that are essential features of any form, namely the action and the method attributes.

The action attribute specifies the URL of the server-side resource that will process the form data. This could be a resource on the same server as the form or a completely different server.

The method attribute specifies how the query string data will be transmitted from the browser to the server. There are two possibilities: GET and POST.

What is the difference between GET and POST? The difference resides in where the browser locates the user's form input in the subsequent HTTP request. With GET, the browser locates the data in the URL of the request; with POST, the form data is located in the HTTP header after the HTTP variables.



Type	Advantages and Disadvantages
GET	<p>Data can be clearly seen in the address bar. This may be an advantage during development but a disadvantage in production.</p> <p>Data remains in browser history and cache. Again this may be beneficial to some users, but a security risk on public computers.</p> <p>Data can be bookmarked (also an advantage and a disadvantage).</p> <p>Limit on the number of characters in the form data returned.</p>
POST	<p>Data can contain binary data.</p> <p>Data is hidden from user.</p> <p>Submitted data is not stored in cache, history, or bookmarks.</p>

2.4 FORM CONTROL ELEMENTS

Type	Description
<button>	Defines a clickable button.
<datalist>	An HTML5 element that defines lists of pre-defined values to use with input fields.
<fieldset>	Groups related elements in a form together.
<form>	Defines the form container.
<input>	Defines an input field. HTML5 defines over 20 different types of input.
<label>	Defines a label for a form input element.
<legend>	Defines the label for a fieldset group.
<option>	Defines an option in a multi-item list.
<optgroup>	Defines a group of related options in a multi-item list.
<select>	Defines a multi-item list.
<textarea>	Defines a multiline text entry box.

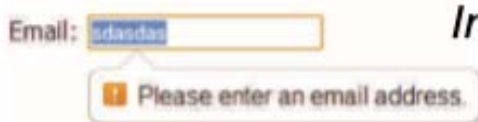
Text Control Elements

Most forms need to gather text information from the user. Whether it is a search box, or a login form, or a user registration form, some type of text input is usually necessary. The following table lists the different text input controls.

```
<input type="email" ... />
```



In Opera



In Chrome

Type	Description
text	Creates a single-line text entry box. <code><input type="text" name="title" /></code>
textarea	Creates a multiline text entry box. You can add content text or if using an HTML5 browser, placeholder text (hint text that disappears once user begins typing into the field). <code><textarea rows="3" ... /></code>
password	Creates a single-line text entry box for a password (which masks the user entry as bullets or some other character) <code><input type="password" ... /></code>
search	Creates a single-line text entry box suitable for a search string. This is an HTML5 element. Some browsers on some platforms will style search elements differently or will provide a clear field icon within the text box. <code><input type="search" ... /></code>
email	Creates a single-line text entry box suitable for entering an email address. This is an HTML5 element. Some devices (such as the iPhone) will provide a specialized keyboard for this element. Some browsers will perform validation when form is submitted. <code><input type="email" ... /></code>
tel	Creates a single-line text entry box suitable for entering a telephone. This is an HTML5 element. Since telephone numbers have different formats in different parts of the world, current browsers do not perform any special formatting or validation. Some devices may, however, provide a specialized keyboard for this element. <code><input type="tel" ... /></code>
url	Creates a single-line text entry box suitable for entering a URL. This is an HTML5 element. Some devices may provide a specialized keyboard for this element. Some browsers also perform validation on submission. <code><input type="url" ... /></code>

```
<input type="text" ... placeholder="L#L #L#" pattern="[a-z][0-9][a-z] [0-9][a-z][0-9]" />
```

Postal:

Postal:

! Please match the requested format.

```
<input type="text" ... />
```

Text:

```
<textarea>
  enter some text
</textarea>
<textarea placeholder="enter some text">
</textarea>
```

TextArea:

TextArea:

```
<input type="password" ... />
```

Password:


Password:


Choice Controls


Forms often need the user to select an option from a group of choices. HTML provides several ways to do this.


- Select List

The `<select>` element is used to create a multiline box for selecting one or more items. The options (defined using the `<option>` element) can be hidden in a dropdown list or multiple rows of the list can be visible. Option items can be grouped together via the `<optgroup>` element. The `selected` attribute in the `<option>` makes it a default value.









```

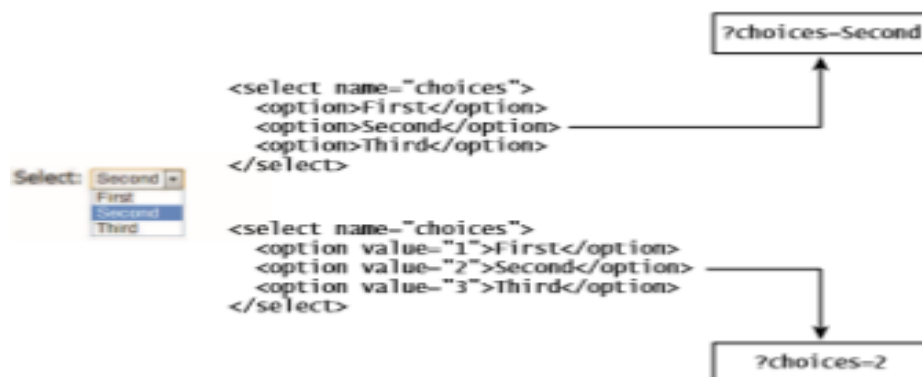
<select name="choices">
  <option>First</option>
  <option selected>Second</option>
  <option>Third</option>
</select>

<select size="3" ... >

<select ... >
  <optgroup label="North America">
    <option>Calgary</option>
    <option>Los Angeles</option>
  </optgroup>
  <optgroup label="Europe">
    <option>London</option>
    <option>Paris</option>
    <option>Prague</option>
  </optgroup>
</select>

```

The `value` attribute of the `<option>` element is used to specify what value will be sent back to the server in the query string when that option is selected. The `value` attribute is optional; if it is not specified, then the text within the container is sent instead



- Radio Button

Radio buttons are useful when you want the user to select a single item from a small list of choices and you want all the choices to be visible. Radio buttons are added via the `<input type="radio">` element. The buttons are made mutually exclusive (i.e., only one can be chosen) by sharing the same name attribute. The checked attribute is used to indicate the default choice.

Continent:

☐ North America

☒ South America

☐ Asia

```
<input type="radio" name="where" value="1">North America<br/>
<input type="radio" name="where" value="2" checked="">South America<br/>
<input type="radio" name="where" value="3">Asia
```

- Checkboxes

Checkboxes are used for getting yes/no or on/off responses from the user. checkboxes are added via the `<input type="checkbox">` element. You can also group checkboxes together by having them share the same name attribute. Each checked checkbox will have its value sent to the server. Like with radio buttons, the checked attribute can be used to set the default value of a checkbox.

Where would you like to visit?

☒ Canada

☐ France

☒ Germany

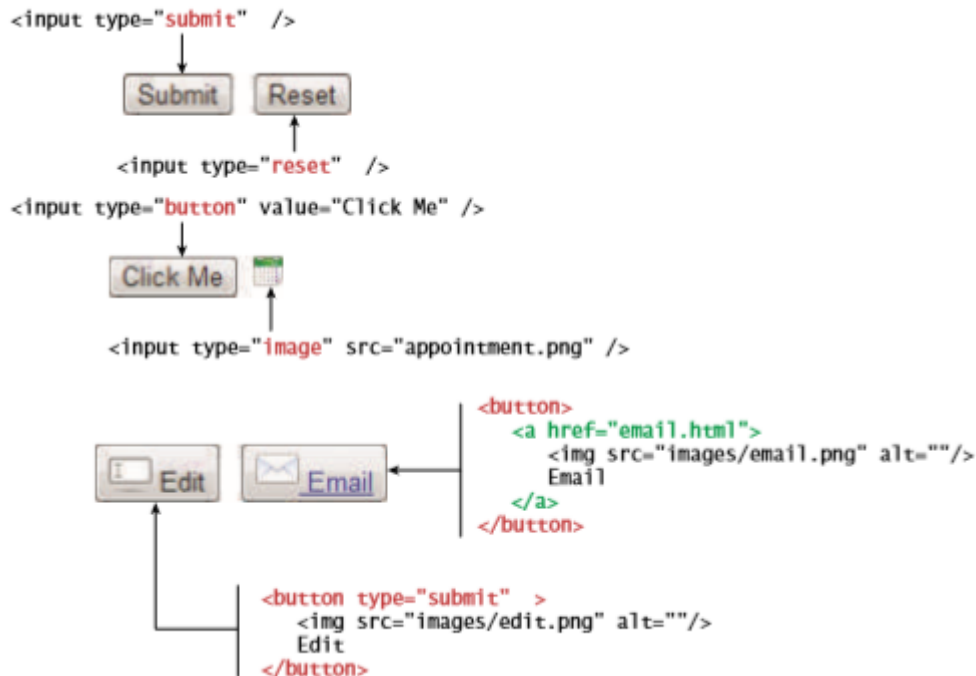
```
<label>Where would you like to visit? </label><br/>
<input type="checkbox" name="visit" value="canada">Canada<br/>
<input type="checkbox" name="visit" value="france">France<br/>
<input type="checkbox" name="visit" value="germany">Germany
```

?accept=on&visit=canada&visit=germany

Button Controls

HTML defines several different types of buttons.

Type	Description
<code><input type="submit"></code>	Creates a button that submits the form data to the server.
<code><input type="reset"></code>	Creates a button that clears any of the user's already entered form data.
<code><input type="button"></code>	Creates a custom button. This button may require JavaScript for it to actually perform any action.
<code><input type="image"></code>	Creates a custom submit button that uses an image for its display.
<code><button></code>	Creates a custom button. The <code><button></code> element differs from <code><input type="button"></code> in that you can completely customize what appears in the button; using it, you can, for instance, include both images and text, or skip server-side processing entirely by using hyperlinks. You can turn the button into a submit button by using the <code>type="submit"</code> attribute.

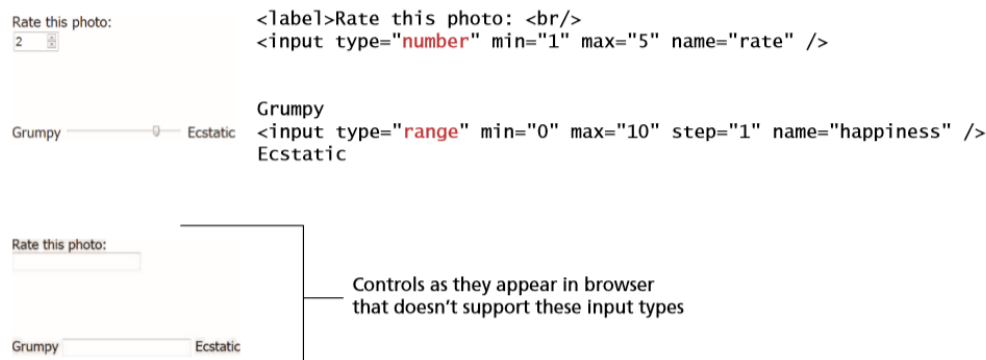


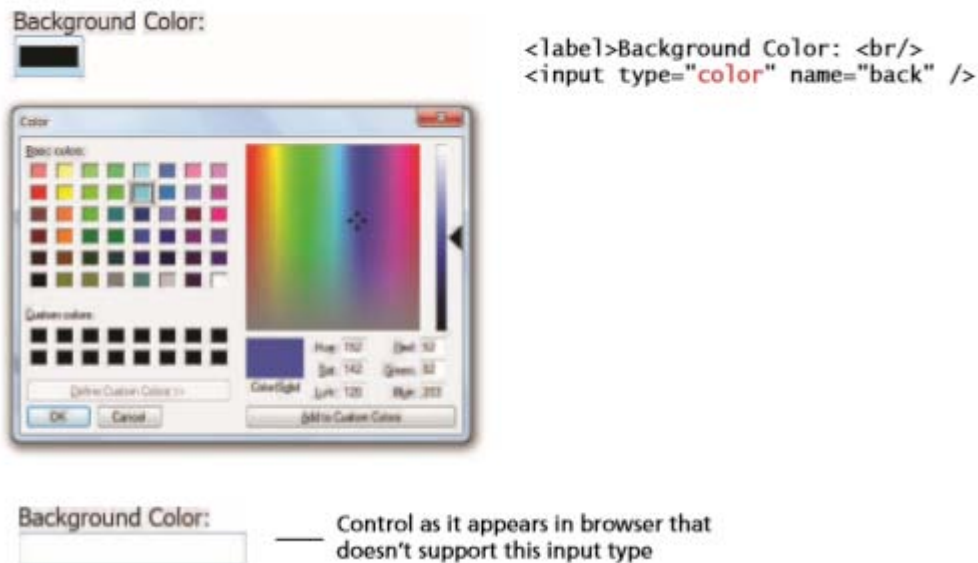
Specialized Controls

There are two important additional special-purpose form controls that are available in all browsers. The first of these is the `<input type="hidden">` element. The other specialized form control is the `<input type="file">` element, which is used to upload a file from the client to the server.

- Number and Range

HTML5 introduced two new controls for the input of numeric values. When input via a standard text control, numbers typically require validation to ensure that the user has entered an actual number and, because the range of numbers is infinite, the entered number has to be checked to ensure it is not too small or too large. The number and range controls provide a way to input numeric values that eliminate the need for client-side numeric validation (for security reasons you would still check the numbers for validity on the server).





Date and Time Controls

The new date and time controls in HTML try to make it easier for users to input these tricky date and time values.

Type	Description
date	Creates a general date input control. The format for the date is "yyyy-mm-dd."
time	Creates a time input control. The format for the time is "HH:MM:SS," for hours:minutes:seconds.
datetime	Creates a control in which the user can enter a date and time.
datetime-local	Creates a control in which the user can enter a date and time without specifying a time zone.
month	Creates a control in which the user can enter a month in a year. The format is "yyyy-mm."
week	Creates a control in which the user can specify a week in a year. The format is "yyyy-W##."

2.5 TABLE AND FORM ACCESSABILITY

The term web accessibility refers to the assistive technologies, various features of HTML that work with those technologies, and different coding and design practices that can make a site more usable for people with visual, mobility, auditory, and cognitive disabilities.

Perhaps the most important guidelines in that document are:

- Provide text alternatives for any nontext content so that it can be changed into other forms people need, such as large print, braille, speech, symbols, or simpler language.
- Create content that can be presented in different ways (for example simpler layout) without losing information or structure.
- Make all functionality available from a keyboard.

- Provide ways to help users navigate, find content, and determine where they are.

Accessible tables

Using the following accessibility features for tables in HTML can also improve the experience for those users:

1. Describe the table's content using the <caption> element (see Figure 4.6). This provides the user with the ability to discover what the table is about before having to listen to the content of each and every cell in the table. If you have an especially long description for the table, consider putting the table within a <figure> element and use the <figcaption> element to provide the description.
2. Connect the cells with a textual description in the header. While it is easy for a sighted user to quickly see what row or column a given data cell is in, for users relying on visual readers, this is not an easy task.

Accessible forms

The forms make use of the <fieldset>, <legend>, and <label> elements, which provide a connection between the input elements in the form and their actual meaning. In other words, these controls add semantic content to the form. While the browser does provide some unique formatting to the <fieldset> and <legend> elements, their main purpose is to logically group related form input elements together with the <legend> providing a type of caption for those elements. You can of course use CSS to style (or even remove the default styling) these

elements.

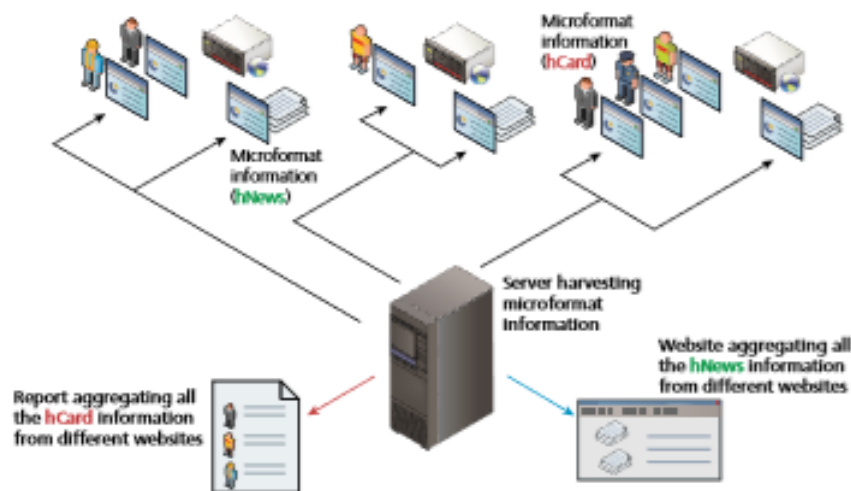
```
<label for="f-title">Title: </label>
<input type="text" name="title" id="f-title"/>

<label for="f-country">Country: </label>
<select name="where" id="f-country">
  <option>Choose a country</option>
  <option>Canada</option>
  <option>Finland</option>
  <option>United States</option>
</select>
```

2.6 MICROFORMATS

A microformat is a small pattern of HTML markup and attributes to represent common blocks of information such as people, events, and news stories so that the information in them can be extracted and indexed by software agents.

One of the most common microformats is hCard, which is used to semantically mark up contact information for a person. Google Map search results now make use of the hCard microformat so that if you used the appropriate browser extension, you could save the information to your computer or phone's contact list.



ADVANCED CSS: LAYOUT

2.7 NORMAL FLOW

Normal flow, which refers here to how the browser will normally display block-level elements and inline elements from left to right and from top to bottom.

Block-level elements such as <p>, <div>, <h2>, , and <table> are each contained on their own line. Because block-level elements begin with a line break (that is, they start on a new line), without styling, two block-level elements can't exist on the same line.

Inline elements do not form their own blocks but instead are displayed within lines. Normal text in an HTML document is inline, as are elements such as , <a>, , and . Inline elements line up next to one another horizontally from left to right on the same line; when there isn't enough space left on the line, the content moves to a new line.

There are actually two types of inline elements: replaced and nonreplaced. Replaced inline elements are elements whose content and thus appearance is defined by some external resource,

such as and the various form elements. Nonreplaced inline elements are those elements whose content is defined within the document, which includes all the other inline elements.

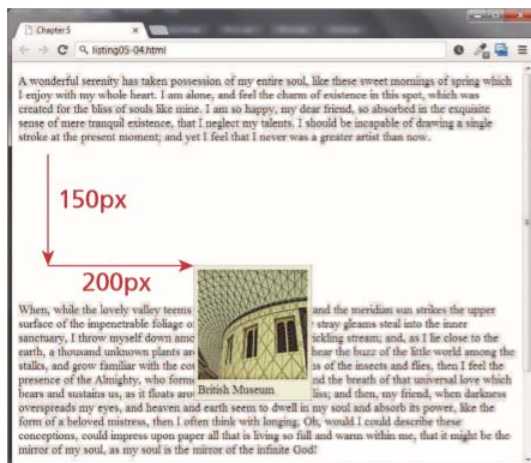
2.8 POSITIONING ELEMENTS

It is possible to move an item from its regular position in the normal flow, and even move an item outside of the browser viewport so it is not visible or to position it so it is always visible in a fixed position while the rest of the content scrolls. The position property is used to specify the type of positioning.

Value	Description
absolute	The element is removed from normal flow and positioned in relation to its nearest positioned ancestor.
fixed	The element is fixed in a specific position in the window even when the document is scrolled.
relative	The element is moved relative to where it would be in the normal flow.
static	The element is positioned according to the normal flow. This is the default.

- Relative position

In relative positioning an element is displaced out of its normal flow position and moved relative to where it would have been placed. When an element is positioned relatively, it is displaced out of its normal flow position and moved relative to where it would have been placed. The other content around the relatively positioned element “remembers” the element’s old position in the flow; thus the space the element would have occupied is preserved



```
figure {
  border: 1pt solid #A8A8A8;
  background-color: #EDEDDE;
  padding: 5px;
  width: 150px;
  position: relative;
  top: 150px;
  left: 200px;
}
```

- Absolute position

When an element is positioned absolutely, it is removed completely from normal flow. Thus, unlike with relative positioning, space is not left for the moved element, as it is no longer in the normal flow. Its position is moved in relation to its container block.



```
figure {
  margin: 0;
  border: 1pt solid #A8A8A8;
  background-color: #EDEDDE;
  padding: 5px;
  width: 150px;
  position: absolute;
  top: 150px;
  left: 200px;
}
```

- Fixed position

The fixed position value is used relatively infrequently. It is a type of absolute positioning, except that the positioning values are in relation to the viewport (i.e., to the browser window). Elements with fixed positioning do not move when the user scrolls up or down the page.

2.9 FLOATING ELEMENTS

It is possible to displace an element out of its position in the normal flow via the CSS float property. An element can be floated to the left or floated to the right. When an item is floated, it is moved all the way to the far left or far right of its containing block and the rest of the content is “re-flowed” around the floated element.

Note : (REFER TEXT FOR EXAMPLES AND DETAILS)

- Floating within a container
- Floating multiple items side by side
- Containing floats
- Overlay and hiding elements

```

<article>
  <h1>Float example</h1>
  <p>A wonderful serenity has taken possession of ... </p>

  <figure>
    
    <figcaption>British Museum</figcaption>
  </figure>

  <p>when, while the lovely valley teems with ...</p>

  <p>O my friend -- but it is too much for my ...</p>
</article>

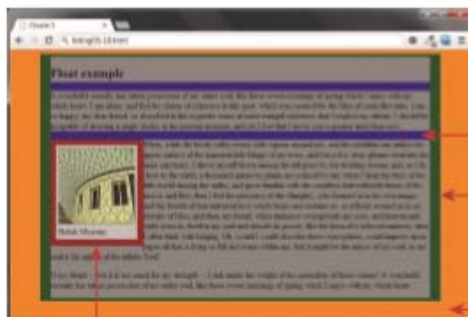
```



```

article {
  background-color: #898989;
  margin: 5px 50px;
  padding: 5px 20px;
}
p { margin: 16px 0; }
figure {
  border: 1pt solid #262626;
  background-color: #c1c1c1;
  padding: 5px;
  width: 150px;
  float: left;
  margin: 10px;
}

```



Clear property

You can stop elements from flowing around a floated element by using the clear property. The values for this property are shown below

Value	Description
left	The left-hand edge of the element cannot be adjacent to another element.
right	The right-hand edge of the element cannot be adjacent to another element.
both	Both the left-hand and right-hand edges of the element cannot be adjacent to another element.
none	The element can be adjacent to other elements.

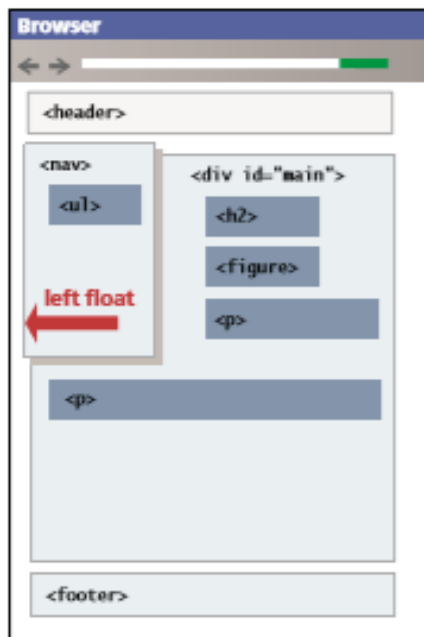
2.10 CONSTRUCTING MULTI COLUMN LAYOUT

Using floats is perhaps the most common way to create columns of content. The approach is shown in below figure. The first step is to float the content container that will be on the left-hand side. Remember that the floated container needs to have a width specified. As can be seen in the second screen capture in Figure, the other content will flow around the floated element.

1 HTML source order (normal flow)



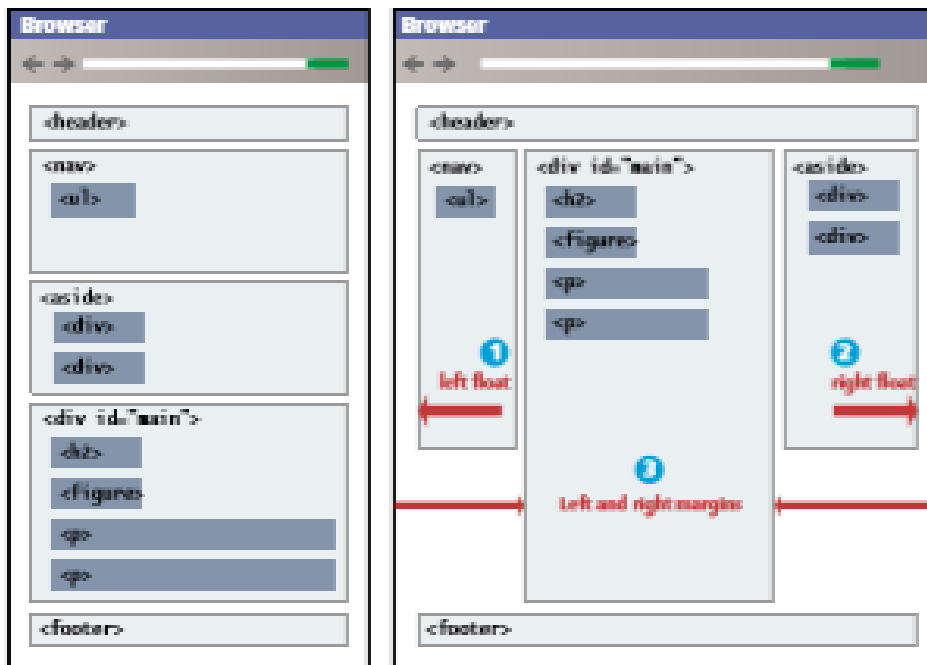
2 Two-column layout (left float)



```
nav {  
  width: 12em;  
  float: left;  
}
```

The left and right floated content must be in the source before the main nonfloated `<div>`. If the `<aside>` element had been after the main `<div>`, then it would have been floated below the main `<div>`.

A 3 column layout can be achieved as below



2.11 APPROCHES TO CSS LAYOUT

The first approach is to use a fixed layout. In a fixed layout, the basic width of the design is set by the designer, typically corresponding to an “ideal” width based on a “typical” monitor resolution. Fixed layouts are created using pixel units, typically with the entire content within a `<div>` container (often named "container", "main", or "wrapper") whose width property has been set to some width.

The second approach to dealing with the problem of multiple screen sizes is to use a liquid layout (also called a fluid layout). In this approach, widths are not specified using pixels, but percentage values. CSS are a percentage of the current browser width, so a layout in which all widths are expressed as percentages should adapt to any browser size.

While the fixed and liquid layouts are the two basic paradigms for page layout, there are some other approaches that combine the two layout styles. Most of these other approaches are a type of hybrid layout, in that they combine pixel and percentage measurements. Fixed pixel measurements might make sense for a sidebar column containing mainly graphic advertising images that must always be displayed and which always are the same width. But percentages would make more sense for the main content or navigation areas, with perhaps min and max size limits in pixels set for the navigation areas.

(Note: refer text for examples)

2.12 RESPONSIVE DESIGN

In a responsive design, the page “responds” to changes in the browser size that go beyond the width scaling of a liquid layout. One of the problems of a liquid layout is that images and horizontal navigation elements tend to take up a fixed size, and when the browser window shrinks to the size of a mobile browser, liquid layouts can become unusable. In a responsive layout, images will be scaled down and navigation elements will be replaced as the browser shrinks.

There are four key components that make responsive design work. They are:

1. Liquid layouts
2. Scaling images to the viewport size
3. Setting viewports via the <meta> tag
4. Customizing the CSS for different viewports using media queries

- Setting view port

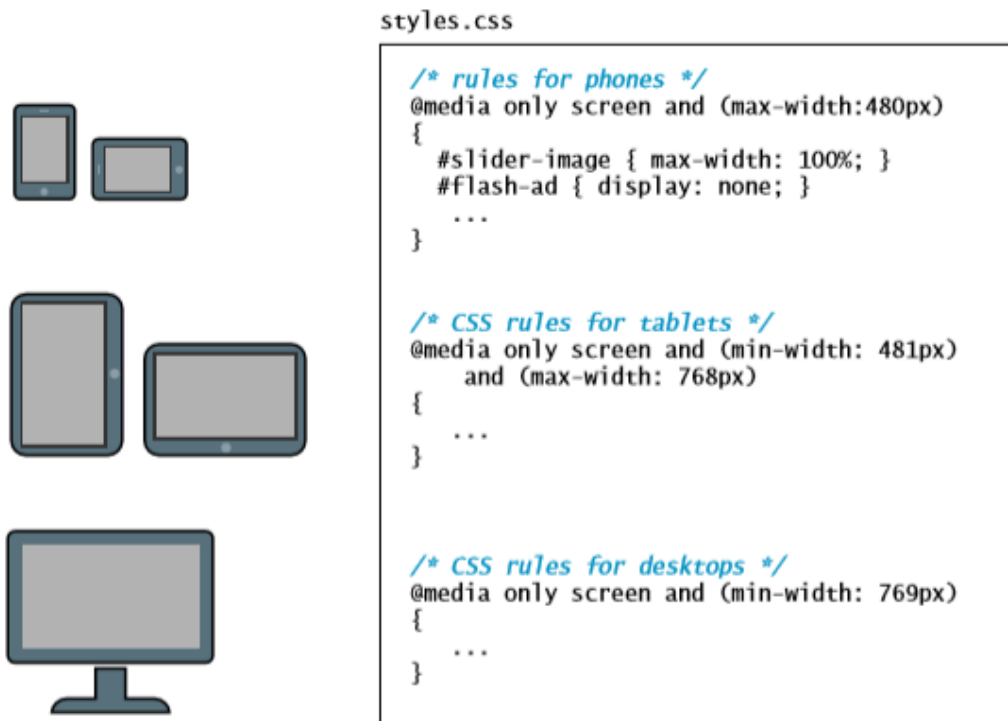
The web page was scaled to fit into the small screen of the browser. The way this works is the mobile browser renders the page on a canvas called the viewport. On iPhones, for instance, the viewport width is 980 px, and then that viewport is scaled to fit the current width of the device (which can change with orientation and with newer versions that have more physical pixels in the screen),

```
<html>
<head>
  <meta name="viewport" content="width=device-width" />
```

- Media queries

The other key component of responsive designs is CSS media queries. A media query is a way to apply style rules based on the medium that is displaying the file. You can use these queries to look at the capabilities of the device, and then define CSS rules to target that device. These queries are Boolean expressions and can be added to your CSS files or to the <link> element to conditionally use a different external CSS file based on the capabilities of the device.

Contemporary responsive sites will typically provide CSS rules for phone displays first, then tablets, then desktop monitors, an approach called progressive enhancement, in which a design is adapted to progressively more advanced devices, an approach you will also see in the JavaScript chapter.



Instead of having all the rules in a single file, we can put them in separate files and add media queries to `<link>` elements.

```

<link rel="stylesheet" href="mobile.css" media="screen and (max-width:480px)" />
<link rel="stylesheet" href="tablet.css" media="screen and (min-width:481px)
and (max-width:768px)" />
<link rel="stylesheet" href="desktop.css" media="screen and (min-width:769px)" />

<!--[if lt IE 9]>
<link rel="stylesheet" media="all" href="style-ie.css"/>
<![endif]-->

```

Handles Internet Explorer 8 and earlier using IE conditional comments.

FIGURE 5.24 Media queries in action

2.13 CSS Frameworks

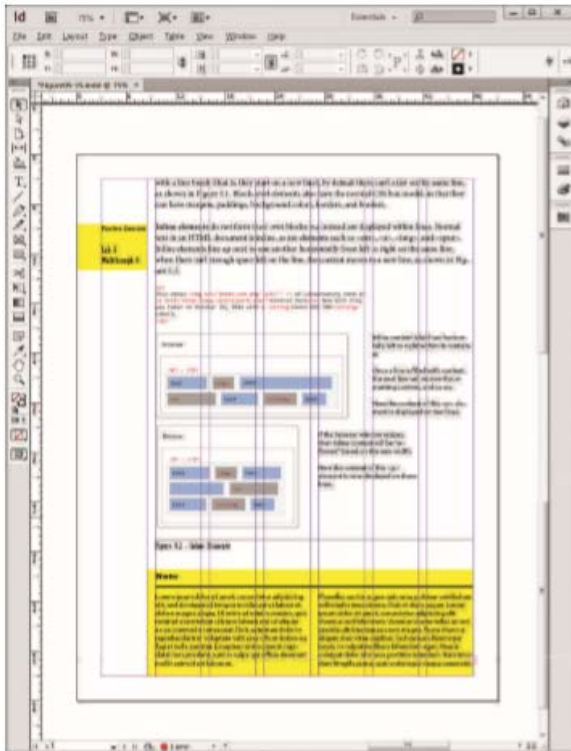
A CSS framework is a precreated set of CSS classes or other software tools that make it easier to use and work with CSS. They are two main types of CSS framework: grid systems and CSS preprocessors.

- Grid System

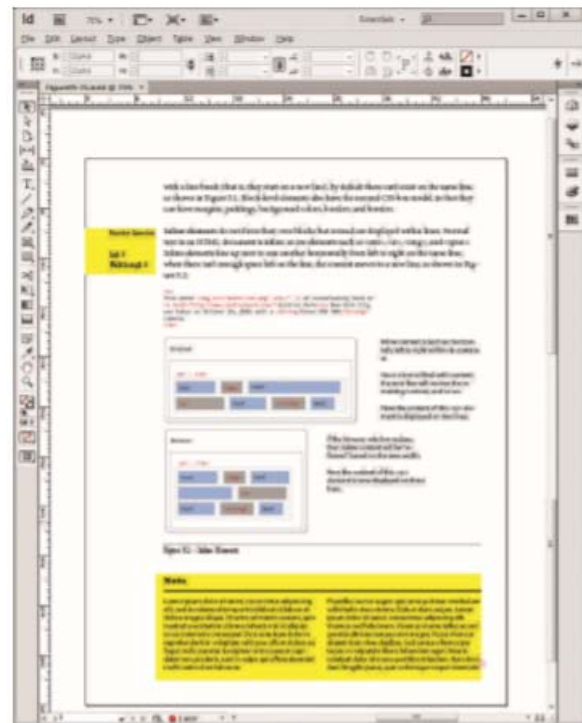
Grid systems make it easier to create multicolumn layouts. There are many CSS grid systems; some of the most popular are Bootstrap (twitter.github.com/bootstrap), Blueprint (www.blueprintcss.org), and 960 (960.gs). Most provide somewhat similar capabilities. The most important of these capabilities is a grid system. CSS frameworks provide similar grid features.

The 960 framework uses either a 12- or 16-column grid. Bootstrap uses a 12-column grid. Blueprint

uses a 24-column grid. The grid is constructed using <div> elements with classes defined by the framework. The HTML elements for the rest of your site are then placed within these <div> elements.



Most page design begins with a grid. In this case, a seven-column grid is being used to layout page elements in Adobe InDesign.



Without the gridlines visible, the elements on the page do not look random, but planned and harmonious.

```
<head>
  <link rel="stylesheet" href="reset.css" />
  <link rel="stylesheet" href="text.css" />
  <link rel="stylesheet" href="960.css" />
</head>
<body>
  <div class="container_12">
    <div class="grid_2">
      left column
    </div>
    <div class="grid_7">
      main content
    </div>
    <div class="grid_3">
      right column
    </div>
    <div class="clear"></div>
  </div>
</body>
```

- CSS preprocessors

CSS preprocessors are tools that allow the developer to write CSS that takes advantage of programming ideas such as variables, inheritance, calculations, and functions. A CSS preprocessor is a tool that takes code written in some type of preprocessed language and then converts that code into normal CSS. The advantage of a CSS preprocessor is that it can provide additional functionalities that are not available in CSS.

