

Module II

HTML Tables and Forms & Advanced CSS

2.1 Introduction to 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.
- A table is a matrix of cells. The cells in the top row often contain column labels, those in the leftmost column often contain row labels, and most of the rest of the cells contain the data of the table.
- The content of a cell can be almost any document element, including text, a heading, a horizontal rule, an image, and a nested table.

Basic Table Tags

A table is specified as the content of the block tag `<table>`.

There are two kinds of lines in tables:

- the line around the whole table is called the border;
 - the lines that separate the cells from each other are called rules.
-
- A table that does not include the border attribute will be a matrix of cells with neither a border nor rules.
 - The browser has default widths for table borders and rules, which are used if the border attribute is assigned the value “border.” Otherwise, a number can be given as border’s value, which specifies the border width in pixels. For example, border = “3” specifies a border 3 pixels wide.
 - A border value of “0” specifies no border and no rules. The rules are set at 1 pixel when any nonzero border value is specified. The border attribute is the most common attribute for the `<table>` tag.
 - In most cases, a displayed table is preceded by a title, given as the content of a `<caption>` tag, which can immediately follow the opening `<table>` tag.
 - The cells of a table are specified one row at a time. Each row of a table is specified with a row tag, `<tr>`.
 - Within each row, the row label is specified by the table heading tag, `<th>`. Each data cell of a row is specified with a table data tag, `<td>`

The first row of a table usually has the table’s column labels. For example, if a table has three data columns and their column labels are, Apple, Orange, and Screwdriver respectively, the first row can be specified by the following:

```
<tr>
<th> Apple </th>
<th> Orange </th>
<th> Screwdriver </th>
</tr>
```

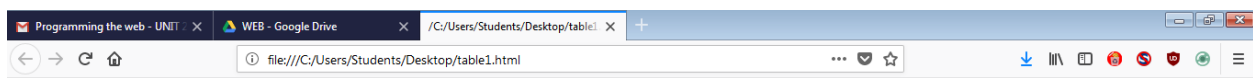
Each data row of a table is specified with a heading tag and one data tag for each data column. For example, the first data row for our work-in-progress table might be as follows:

```
<tr>
<th> Breakfast </th>
<td> 0 </td>
<td> 1 </td>
<td> 0 </td>
</tr>
```

The following document describes the whole table:

Eg:

```
<html>
<body>
<br />
<table border="1">
<caption> Diet </caption>
<tr>
    <th>Breakfast</th>
    <th>Lunch</th>
    <th>Dinner</th>
</tr>
<tr>
    <td>Apple</td>
    <td>Rice</td>
    <td>Cucumber</td>
</tr>
<tr>
    <td>Watermelon</td>
    <td>Rice</td>
    <td>Papaya</td>
</tr>
</table>
</body>
</html>
```

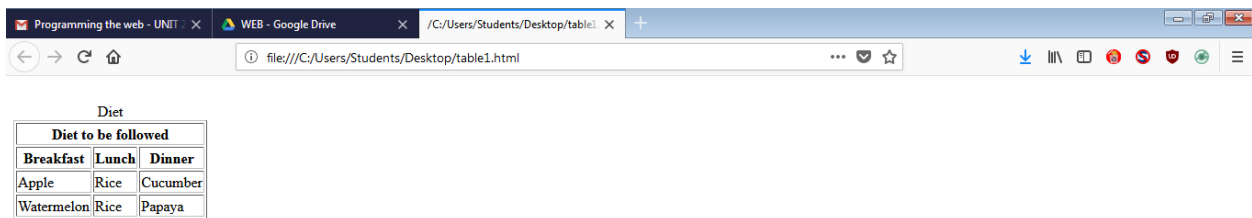


Breakfast	Lunch	Dinner
Apple	Rice	Cucumber
Watermelon	Rice	Papaya

The rowspan and colspan Attributes

In many cases, tables have multiple levels of row or column labels in which one label covers two or more secondary labels. For example, consider the display of a partial table shown in Figure below. In this table, the upper-level label ‘Diet to be followed’ spans the three lower-level label cells. Multiple-level labels can be specified with the rowspan and colspan attributes.

```
<tr>
<th colspan="3">Diet to be followed</th>
</tr>
```



Diet to be followed		
Breakfast	Lunch	Dinner
Apple	Rice	Cucumber
Watermelon	Rice	Papaya

The colspan attribute specification in a table header or table data tag tells the browser to make the cell as wide as the specified number of rows.

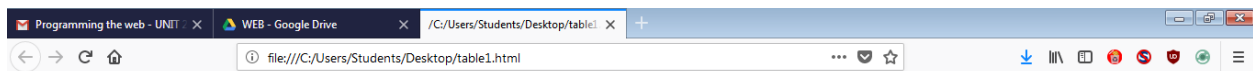
Eg:

```
<html>
<body>
  <table border="1">
    <caption> Diet </caption>
    <tr>
      <th rowspan="2"></th>
      <th colspan="3">Fruit Juice Drinks</th>
    </tr>
    <tr>
      <th>Apple</th>
      <th>Orange</th>
      <th>Strawberry</th>
    </tr>
    <tr>
      <th>Breakfast</th>
      <td>0</td>
      <td>1</td>
      <td>0</td>
    </tr>
    <tr>
      <th> Lunch</th>
      <td>1</td>
      <td>1</td>
```

```

        <td>1</td>
    </tr>
    <tr>
        <th>Dinner</th>
        <td>0</td>
        <td>1</td>
        <td>0</td>
    </tr>
</table>
</body>
</html>

```



Diet			
	Fruit Juice Drinks		
	Apple	Orange	Strawberry
Breakfast	0	1	0
Lunch	1	1	1
Dinner	0	1	0

Additional Table Elements

- The <thead>, <tfoot>, and <tbody> elements can also be used in table.
- The headings of the table are put in <thead> element, the content of rows in <tbody> element and if any summaries in <tfoot> element.
- These elements divide the table into different section. CSS can be applied on these sections separately.
- The <col> and <colgroup> elements are also mainly used to aid in the styling of the table. Number of columns can be grouped and similar style is applied to the whole group. The possible properties that can be set are borders, backgrounds, width, and visibility.
- HTML tables were frequently used to create page layouts, which divide the window into many frames. Images, link tag <a> and text can be put in different cells of the table. Some of the problems occurred due to this approach are –
 - Tend to dramatically increase the size of the HTML document
 - Large number of extra tags are required for <table> elements
 - These files take longer to download and are difficult to maintain because of the extra markup.
 - It is not semantic, as tables are meant to indicate tabular data

```

<table>
<tr>
    <td>
        
    </td>
    <td>
        <h2>Castle</h2>
        <p>Lewes, UK</p>
        <p>Photo by: Michele Brooks</p>
    </td>
</tr>
</table>

```

<p>Built in 1069, the castle has a tremendous view of the town of Lewes and the surrounding countryside.

</p>

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

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

The screenshot shows a web browser window titled 'Chapter 4' with the address bar displaying 'figure04-07.html'. The page content is a table with the title '19th Century French Paintings'. The table has three columns: 'Title', 'Artist', and 'Year'. The rows are as follows:

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

2.2 Styling Tables

All the CSS properties can be applied on tables, the other styling properties for only table are-

- Table Borders
- Boxes and Zebras

Table Borders

Table borders can be assigned to both the <table> and the <td> element (or <th>). Borders cannot be assigned to the <tr>, <thead>, <tfoot>, and <tbody> elements.

The border-collapse property selects the table's border model. By default, each cell has its own unique borders. The space between the adjacent borders can be changed by using the border-spacing property.

Boxes and Zebras

By using the CSS style, it is possible to change the background colors and borders, change the appearance of a row when mouse moves over it and also change the format of nth child



The screenshot shows a web browser window titled 'Chapter 4' with the address bar displaying 'figure04-09.html'. The page content is a table with the title '19TH CENTURY FRENCH PAINTINGS'. The table has three columns: 'Title', 'Artist', and 'Year'. The rows are as follows:

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;
}
```

19TH CENTURY FRENCH PAINTINGS		
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. Another way is by using hyperlinks.
- Using a form, the user can enter text, choose items from lists, and click buttons. Programs running on the server will take the input from HTML forms and processes it, or save it.
- 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.

Sending of data to the server (query string):

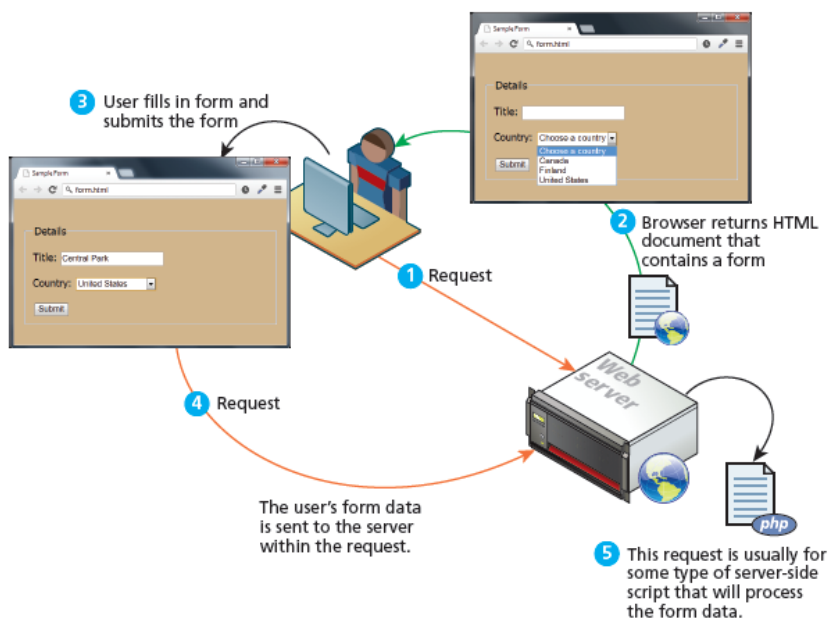
- The browser packages the user's data input into something called a query string.
- A **query string** is a series of name=value pairs separated by ampersands (the **&** character). The names in the query string are defined in the HTML form; each form element contains a name attribute, which is used to define the name for the form data in the query string.
- The values in the query string are the data entered by the user.
- 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.

```
<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'. A 'Submit' button is located below the 'Country' field. A blue arrow points from the 'title' attribute in the code above to the 'Title' input field.

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

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

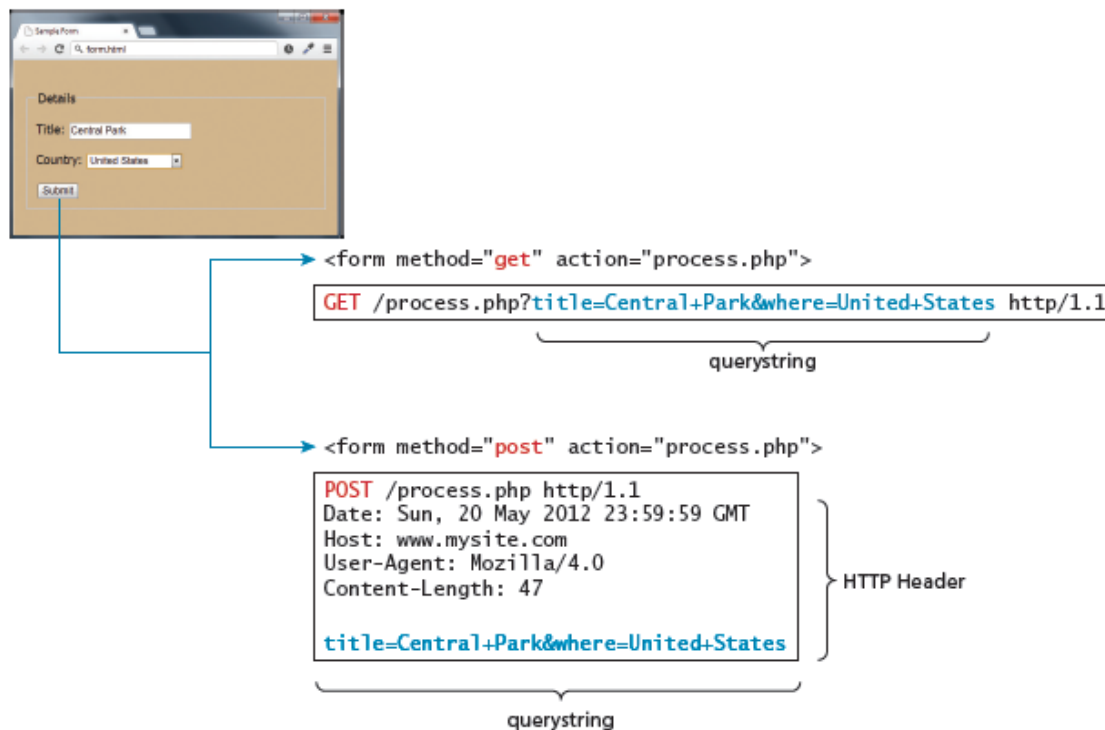


Example: Suppose the user a request for an HTML page from the server (1) that contains some type of form on it. This could be something as complex as a user registration form or as simple as a search box. After the user fills out the form, there needs to be some mechanism for submitting the form data back to the server. This is typically achieved via a **submit** button. As 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.

The <form> Element

- The HTML form contains two important attributes that are essential features of any form, namely the **action** and the **method** attributes.
- 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.

The use of GET or POST method decides where the browser locates the user input in the HTTP request. Using **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.



	GET Method	POST Method
1	Clearly seen in the address bar	Data is hidden from the user
2	Usually used during development, for testing	Used once the product is ready
3	Data remains in browser history and cache.	Submitted data is not stored in browser history or cache.
4	Limits on the number of characters in the form.	There is no limit on number of characters entered in form.

Generally, form data is sent using the POST method. However, the GET method is useful when you are testing or developing a system, since you can examine the query string directly in the browser's address bar.

2.4 Form Control Elements

Some of the form related HTML elements are –

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

Text Input Controls

Text Input Controls – used to get text information from the user. It is used in search box to enter search element, in a login form to enter the name, in user registration form to enter name, parents name, caste etc. Different text input controls are –

Type	Description
text	Creates a single-line text entry box. <code><input type="text" name="title" /></code>
textarea	Creates a multiline text entry box. <code><textarea rows="3" cols="50" /></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>
email	Creates a single-line text entry box suitable for entering an email address. This is an HTML5 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. <input type="tel" ... />
url	Creates a single-line text entry box suitable for entering a URL. This is an HTML5 element. Browsers perform validation on submission. <input type="url" ... />


Choice Controls


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

Select Lists

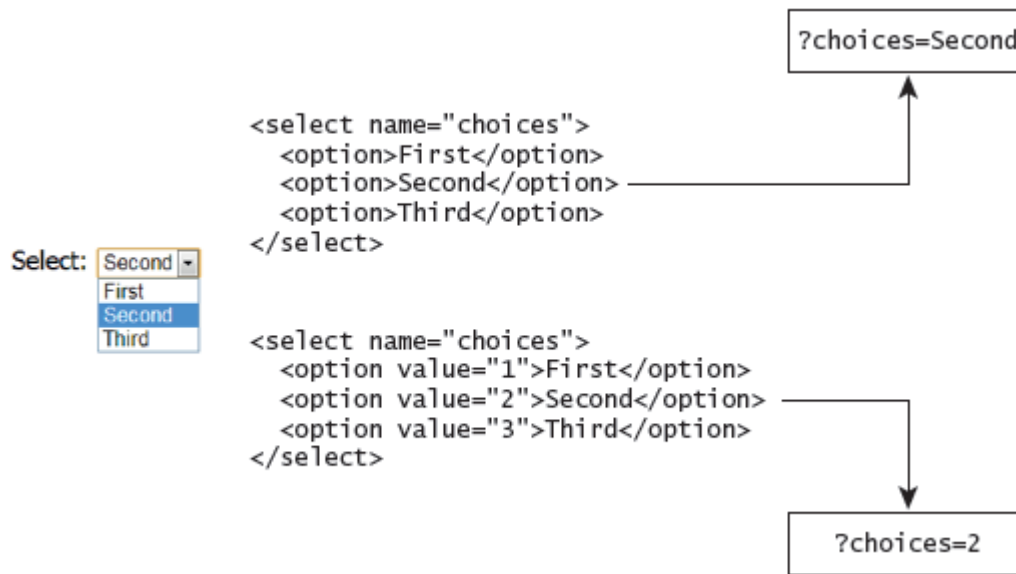
- The <select> element is used to create a multiline box for selecting one or more items.
- The options (defined using the <option> element) can be hidden in a dropdown list or multiple rows of the list can be visible.
- Option items can be grouped together via the <optgroup> element. The selected attribute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

The new <datalist> element is a new addition to HTML5. This element allows to define a list of values that can appear in a drop-down autocomplete style list for a text element. This can be helpful for situations in which the user must have the ability to enter anything, but are often entering one of a handful of common elements.

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

 `<select size="3" ... >`

Cities:  `<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>`



```
<input type="text" name="city" list="cities" />
<datalist id="cities">
  <option>Calcutta</option>
  <option>Calgary</option>
  <option>London</option>
  <option>Los Angeles</option>
  <option>Paris</option>
  <option>Prague</option>
</datalist>
```

Radio Buttons

Radio buttons are used when the user has to select a single item from a small list of choices and all the choices have to be visible. They are added by using 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, while the value entered in value attribute is sent to the server when submit button is clicked.

Eg:

```
<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
```

Continent:

- ☐ North America
- ☒ South America
- ☐ Asia

Checkboxes

Checkboxes are used for getting yes/no or on/off responses from the user. Checkboxes are added by using the `<input type="checkbox">` element. You can also group checkboxes together by sharing the same name attribute. Each checked checkbox will have its value sent to the server.

I accept the software license ☒ `<input type="checkbox" name="accept" >`

Where would you like to visit?
☒ Canada
☐ France
☒ Germany

`<label>Where would you like to visit? </label>

<input type="checkbox" name="visit" value="canada">Canada

<input type="checkbox" name="visit" value="france">France

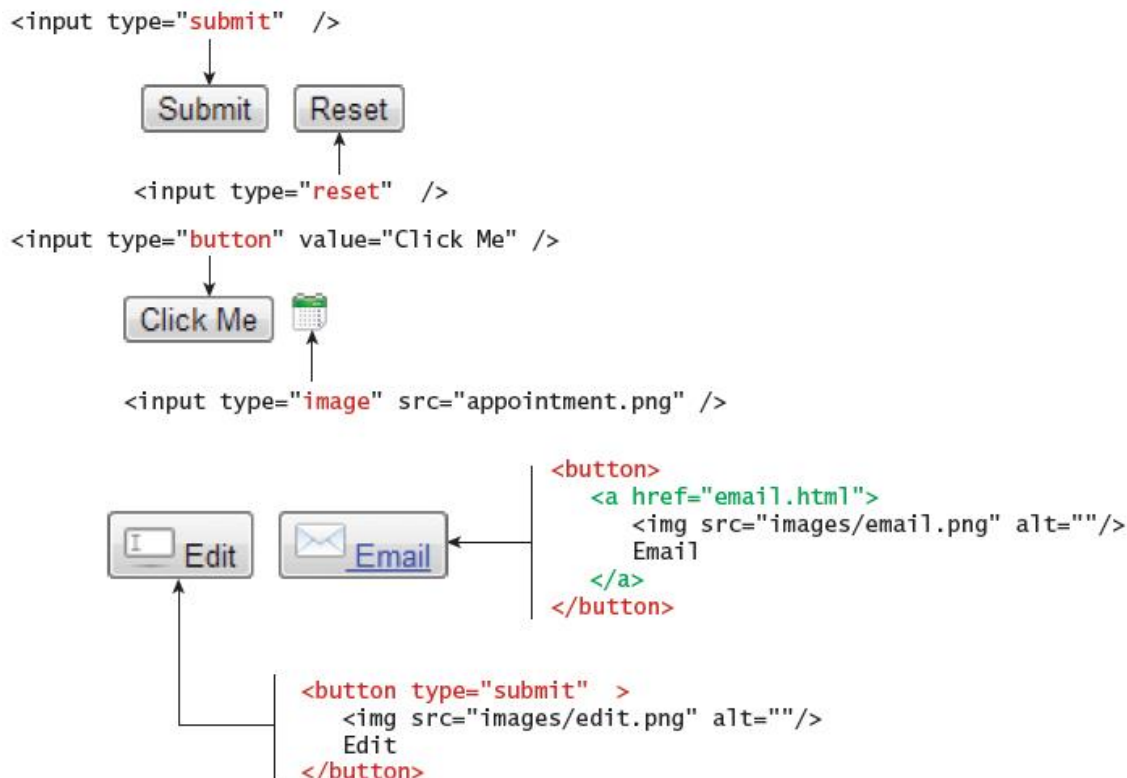
<input type="checkbox" name="visit" value="germany">Germany`

`?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 the user's already entered form data.
<code><input type="button"></code>	Creates a custom button. This button may requires a script 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. It can be used to include both images and text.



Date and Time Controls

The user entering of date or time may need to be validated, as it may cause error when manually entered. The new HTML date and time controls make it easier for users to input these tricky date and time values. The format output of all date & time controls vary depending on the browser.

Type	Description
date	Creates a general date input control. The format for the date is “mm-dd-yyyy”.
time	Creates a time input control. The format for the time is “HH:MM AM/PM,” for hours:minutes
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 “mm- yyyy.”
week	Creates a control in which the user can specify a week in a year. The format is “W##- yyyy.”

Date:

```
<label>Date: <br/>
<input type="date" ... />
```

Time:

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

DateTime:

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

DateTime Local:

```
<input type="datetime-local" ... />
```

Month:

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

Week:

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

2.5 Table and Form Accessibility

Users with sight disabilities, for instance, experience the web using voice reading software. Color blind users might have trouble differentiating certain colors in proximity; users with muscle control problems may have difficulty using a mouse, while older users may have trouble with small text and image sizes.

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.

In order to improve the accessibility of websites, the W3C created the **Web Accessibility Initiative (WAI)** in 1997. The WAI produces guidelines and recommendations, as well as organizing different working groups on different accessibility issues.

Perhaps the most important guidelines for web accessibility 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.*
- *Provide ways to help users navigate, find content, and determine where they are.*

The guidelines provide detailed recommendations on how to achieve this advice.

Accessible Tables

HTML tables can be quite frustrating, for people with visual disability. One important way to improve the accessibility is to only use tables for tabular data, not for layout. Using the following accessibility features for tables in HTML can improve the using of tables for those users:

1. Describe the table's content using the <caption> element. 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.
2. Connect the cells with a textual description in the header. It is quite revealing to listen to reader software recite the contents of a table that has not made these connections. It sounds like this: "row 3, cell 4: 45.56; row 3, cell 5: Canada; row 3, cell 6: 25,000; etc." However, if these connections have been made, it sounds instead like this: "row 3, Average: 45.56; row 3, Country: Canada; row 3, City Count: 25,000; etc.,".

Accessible Forms

HTML forms are also potentially problematic with respect to accessibility. The use of the <fieldset>, <legend>, and <label> elements, provide a connection between the input elements in the form.

The main purpose of <fieldset> and <legend> is to logically group related form input elements together with the <legend> providing a type of caption for those elements.

Each <label> element should be associated with a single input element, by using the 'for' attribute. So that if the user clicks on or taps the <label> text, that control will receive the form's focus (i.e., it becomes the current input element and any keyboard input will affect that control).

Associating label and the input tag -

```
<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

The web has millions of pages and there are many similar information from site to site. Most sites have Contact Us page, in which addresses and other information are displayed; calendar of upcoming events or information about products or news. These types of common information can be tagged in a similar way, and automated tools can be used to gather and transform the information.

A **microformat** is a small pattern of HTML markup and 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.

Module II-Chapter -2

Advanced CSS

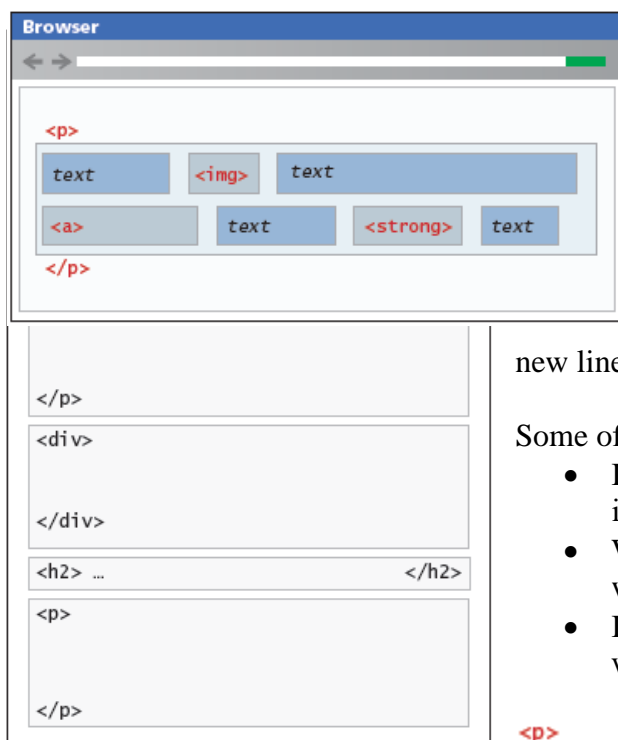
2.7 Normal Flow

The **normal flow** in html refers 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 elements that are contained on their own line, because block-level elements begin with a line break (new line). Two block-level elements can't exist on the same line, without styling.

Some of the properties of block-level elements -

- Each block exists on its own line.
- It is displayed in normal flow from the browser window's top to its bottom.
- By default each block level element fills up the entire width of its parent (browser window).
- CSS box model properties can be used to customize, for instance, the width of the box and the margin space between other block level elements



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

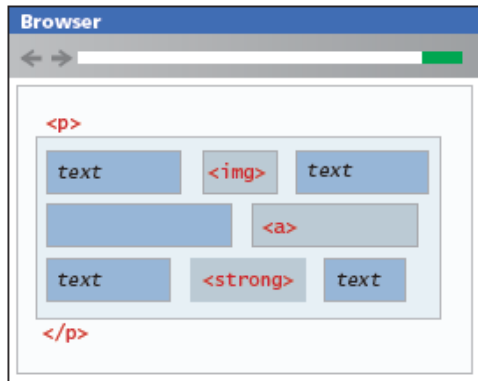
new line.

Some of the properties of inline elements are –

- Inline element is displayed in normal flow from its container's left to right.
- When a line is filled with content, the next line will receive the remaining content, and so on.
- If the browser window resizes, then inline content will be “re-flowed” based on the new width.

```
<p>
This photo  of Conservatory Pond in
<a href="http://www.centralpark.com/">Central Park</a> New York City
was taken on October 22, 2015 with a <strong>Canon EOS 30D</strong>
camera.
</p>
```

If the `<p>` tag contains many tags as shown, the inline tags are placed within the `<p>` tag as shown. If the window is re-sized the elements are re-flowed and now occupies three rows.



There are two types of inline elements: replaced and nonreplaced. **Replaced inline elements** are elements whose content and 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. Eg: `<a>`, ``, `<i>`, ``.

Replaced inline elements have a width and height that are defined by the external resource, eg. the size of image is defined externally.

In a document with normal flow, block-level elements and inline elements are placed together. Block-level elements will flow from top to bottom, and inline elements flow from left to right within a block. A block element can contain another block.

It is possible to change whether an element is block-level or inline using the CSS 'display' property. Consider the following two CSS rules:

```
span { display: block; }
```

```
li { display: inline; }
```

These two rules will make all `` elements behave like block-level elements and all `` elements like inline (that is, each list item will be displayed on the same line).

2.8 Positioning Elements

It is possible to

- 1) move an item from its regular position in the normal flow
- 2) move an item outside of the browser viewport so that it is not visible
- 3) move to position so that 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, and the possible values are

Type	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.

The left, right, top, and bottom properties are used to indicate the distance the element will move.

Relative Positioning

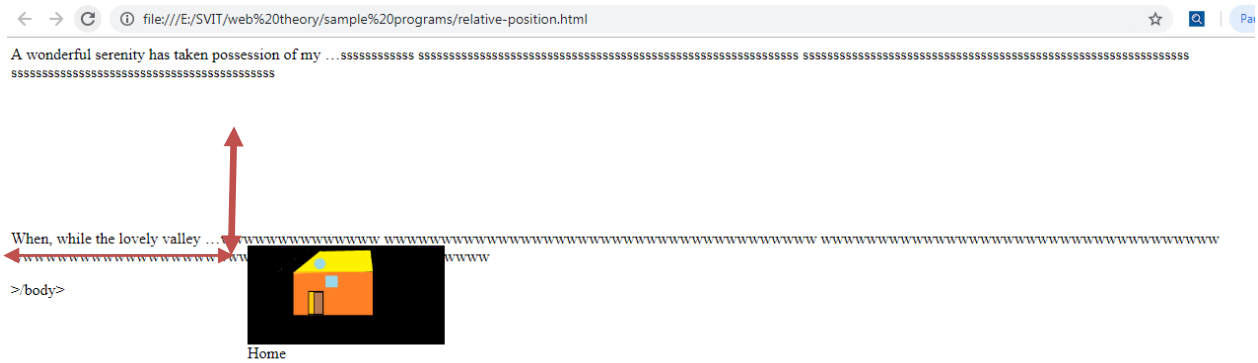
In **relative positioning** an element is displaced out of its normal flow position and moved relative to where it would have been placed normally. The other contents around the relatively positioned element remain in its old position in the flow; thus the space the element would have occupied is preserved as shown in the example below.

Eg:

```
<html>
<head>
  <style>
    figure {
      position: relative;
      top: 150px;
      left: 200px;
    }
  </style>
</head>
<body>
<p>A wonderful serenity has taken possession of my ...ssssssssss
ssssssssssssssssssssssssssssssssssssssssssssssssssssssssss
ssssssssssssssssssssssssssssssssssssssssssssssssssssssssss
ssssssssssssssssssssssssssssssssssssssssssssssssssssssssss</p>
<figure>

<figcaption>Home</figcaption>
</figure>
<p>When, while the lovely valley ...wwwwwwwwwwwwwww
wwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwww
wwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwww
wwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwww</p>

</body>
</html>
```



The repositioned element overlaps other content: that is, the `<p>` element following the `<figure>` element does not change to accommodate the moved `<figure>`.

Absolute Positioning

When an element is positioned absolutely, it is removed completely from normal flow. Here, 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. In the below example, `<figure>` block's container is body block.

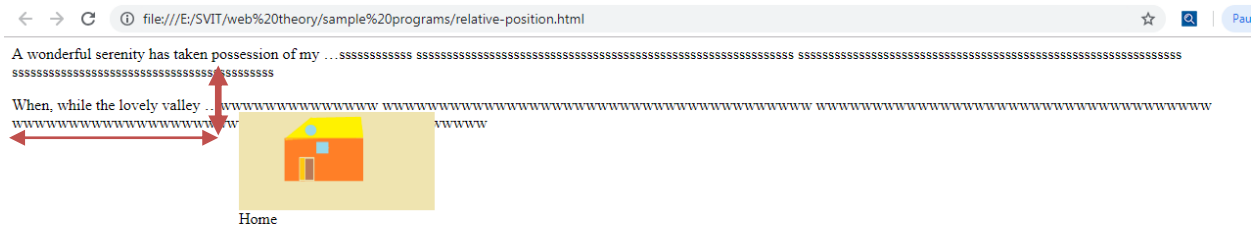
The moved block can overlap the content in the underlying normal flow.

Eg:

```
<html>
<head>
  <style>
    figure {
      position: absolute;
      top: 60px;
      left: 200px;
    }
  </style>
</head>
<body>
  <p>A wonderful serenity has taken possession of my ...ssssssssss
  sssssssssssssssssssssssssssssssssssssssssssssssssssssssssssss
  sssssssssssssssssssssssssssssssssssssssssssssssssssssssssssss
  sssssssssssssssssssssssssssssssssssssssssssssssssssssssssssss</p>
  <figure>
    
    <figcaption>Home</figcaption>
  </figure>
  <p>When, while the lovely valley ...wwwwwwwwwwwwwwww
  wwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwww
  wwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwww
  wwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwww</p>

</body>
```

</html>



Z-Index

Each positioned element has a stacking order defined by the z-index property (named for the z-axis). Items closest to the viewer (and thus on the top) have a larger **z-index** value, as shown in the example below.

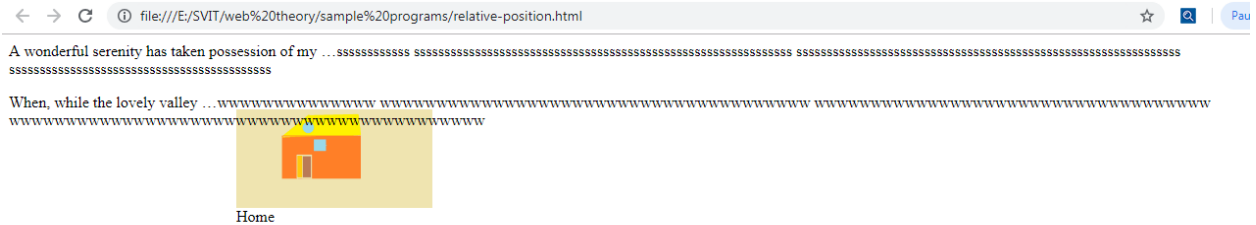
Eg:

```
<html>
<head>
  <style>
    figure {
      position: absolute;
      top: 60px;
      left: 200px;
      z-index:-1;
    }

    body{
      z-index:1:
    }
  </style>
</head>
<body>
<p>A wonderful serenity has taken possession of my ...ssssssssss
ssssssssssssssssssssssssssssssssssssssssssssssssssssssssssss
ssssssssssssssssssssssssssssssssssssssssssssssssssssssssss
ssssssssssssssssssssssssssssssssssssssssssssssssssssssssss</p>
<figure>

<figcaption>Home</figcaption>
</figure>
<p>When, while the lovely valley ...wwwwwwwwwwwwwwww
wwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwww
wwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwww
wwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwww</p>

</body>
</html>
```



Observe the `<p>` tag content has appeared over the image.

Fixed Position

The fixed position value is used relatively infrequently. The element is positioned 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.

The fixed position is used to ensure that navigation elements or **advertisements are always visible**.

`<head>`

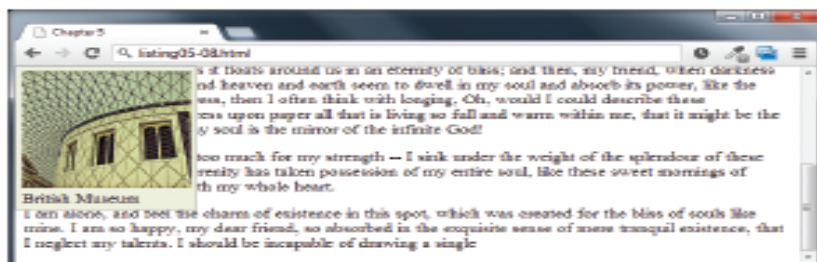
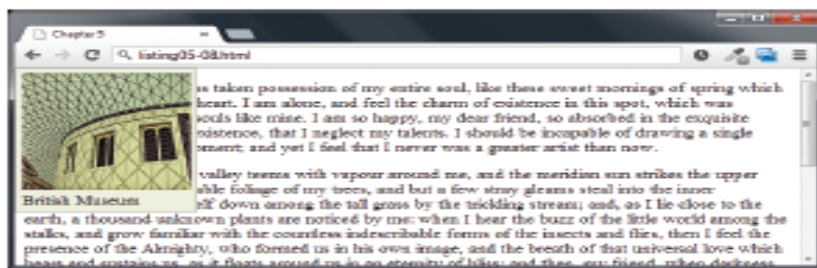
`<style>`

```
figure {
  position: fixed;
  top: 60px;
  left: 200px;
}
```

`</style>`

`</head>`

```
figure {
  ...
  position: fixed;
  top: 0;
  left: 0;
}
```



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.

Notice that a floated block-level element must have a width specified; otherwise, the width will be set to auto, which will mean it implicitly fills the entire width of the containing block, and there will be no room available to flow content around the floated item.

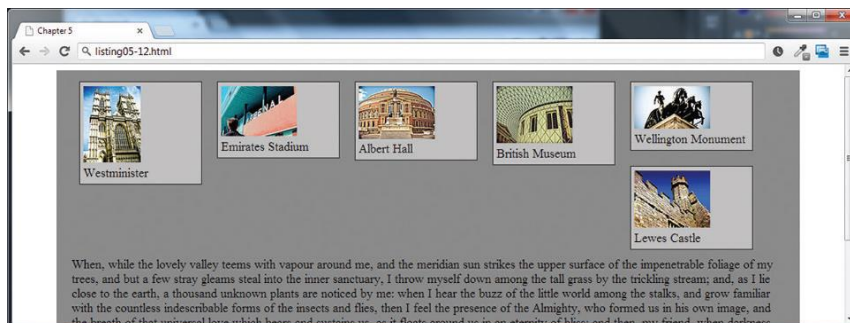
Floating within a Container

It should be reiterated that a floated item moves to the left or right of its container.

The floated figure contained within an <article> element that is indented from the browser’s edge. The relevant margins and padding areas are color coded to help make it clearer how the float interacts with its container.

Floating Multiple Items Side by Side

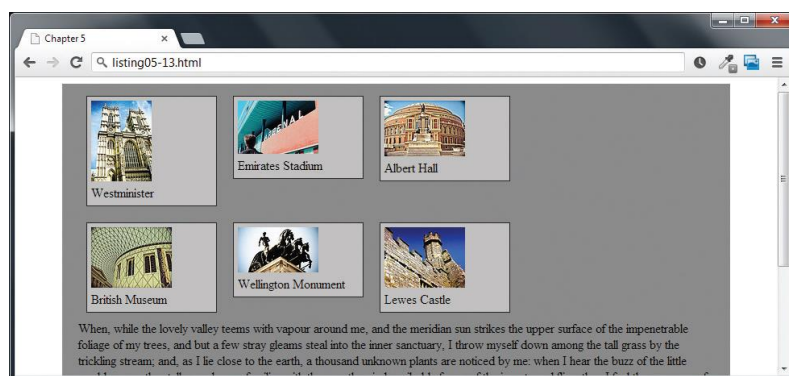
A common use of float property is to place multiple items side by side on the same line. When multiple items are floated, each element will be nestled up beside the previously floated item. All other content in the containing block will flow around all the floated elements.



```
figure {  
...  
width: 150px;  
float: left;  
}
```

This arrangement of images floated changes as the browser window size changes. If suppose any element has to be stopped from flowing around a floated element, it can be done by using the **clear CSS property**.

By setting the clear property of third image to left, it means that there should be no elements to its left. { clear : left; }



The other values for clear property are described 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	the left-hand and right-hand edges of the element cannot be adjacent
none	The element can be adjacent to other elements.

Overlaying and Hiding Elements

One of the more common design tasks with CSS is to place two elements on top of each other, or to selectively hide and display elements. Positioning is important to both of these tasks. Positioning is often used for smaller design changes, such as moving items relative to other elements within a container.

An image that is the same size as the underlying one is placed on top of the other image using absolute positioning.

There are in fact two different ways to hide elements in CSS: using the display property and using the visibility property. The display property takes an item **out of the flow**: it is as if the element no longer exists. The visibility property just **hides the** element, but the space for that element remains.



```
figure {
  ...
  display: auto;
}
```



```
figure {
  ...
  display: none;
}
```



```
figure {
  ...
  visibility: hidden;
}
```

2.10 Constructing Multicolumn Layouts

The previous sections showed two different ways to move items out of the normal top-down flow, by using positioning (relative, absolute or fixed) and by using floats. They are the techniques that can be used to create more complex layouts. The below topics are about the creation of layout using float and positioning property of CSS.

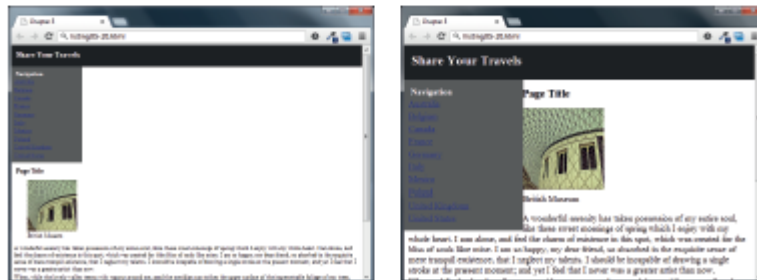
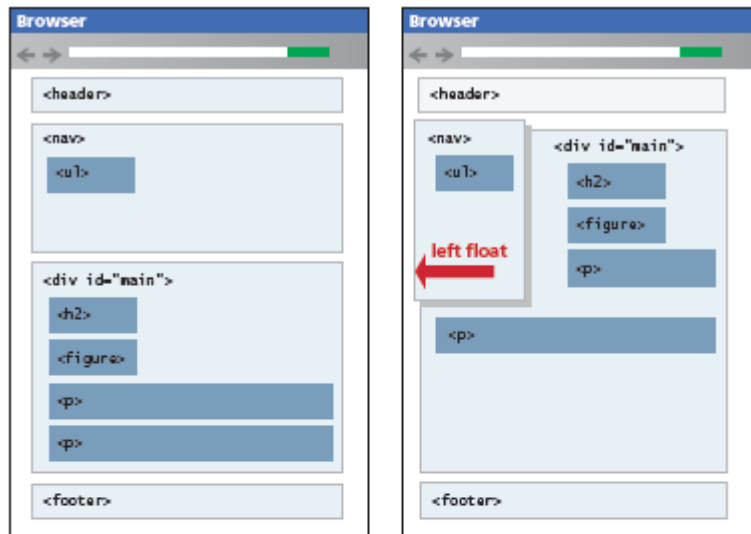
Using Floats to Create Columns

Using floats is the most common way to create columns of content. The steps for this approach are as follows –

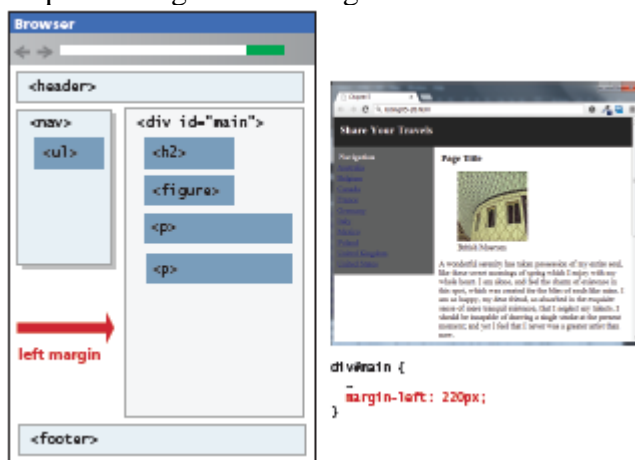
1. float the content container that will be on the left-hand side. (the floated container needs to have a width specified).
2. The other content will flow around the floated element.
3. Set the left – hand side margin for the non-floated element.

The layout without float
Property

The layout after using the float property for
left side element



Step 3: setting the left margin -



```
<!DOCTYPE html>
<html>
  <head>
    <title> Simple Form </title>
    <style type = "text/css">
      nav{
```

```

        width:200px;
        float:left;
        background-color:red;
    }
    div{
        margin-left: 220px;
        margin-right: 100px;
        background-color:yellow;
    }
</style>
</head>
<body>
    <header style="background-color:pink;align:center;">
        <h1 style="text-align:center;">Sai Vidya Institute of Technology</h1>
    </header>
    <aside style="background-color:black;color:white;float:right;">
        <p>extra content</p>
    </aside>
    <nav>
        <p>hi</p>
    </nav>
    <div>
        <p> main content </p>
        <p> main content </p>
        <p> main content </p>
        <p> main content </p>
        <p> main content </p>
    </div>

    <footer style="background-color:blue;color:white;">
        <p style="text-align:center;">&copy svit</p>
    </footer>
</body>
</html>

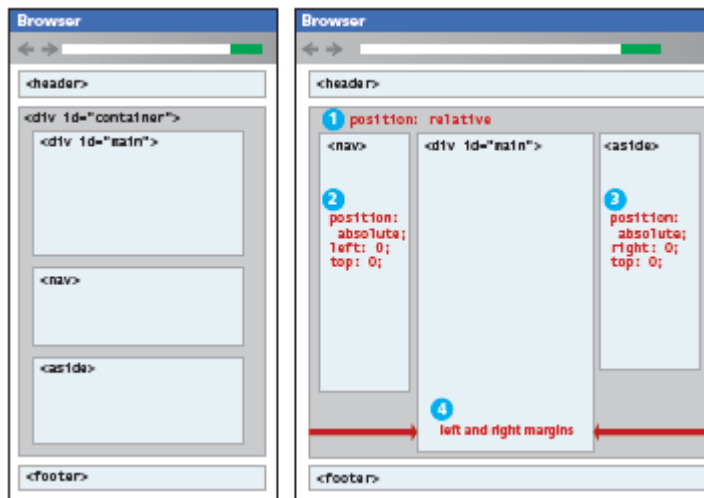
```



Using Positioning to Create Columns

Positioning can also be used to create a multicolumn layout. Typically, the approach is to absolute position the elements.

This approach uses some type of container, in which the elements are positioned.



The following steps are followed –

1. Position the container element (into which all other elements are positioned) with respect to the browser window.
2. Position the other elements with respect to the container element, created in step 1.

```
<html>
<head>
<style>
```

```
#container{
    position:relative;           //main container
    top:0px;
    left:0px;
}
nav
{
    position:absolute;           // positioned with respect to main container
    top: 0px; left:0px;
    width:150px;
    height:300px;
    background-color: #cc0055
}

#side{
    margin-left: 150px;           // column in the middle
    margin-right:150px;
    height: 300px;
    background-color:#CCCCCC}
h1{ background-color:red; }
```

```
</style>
</head>
<body>

<h1 align="center" > HHHHH</h1>

<div id="container">

<nav>
<ul>
<li>abc</li>
<li>def</li>
<li>abc</li>
<li>def</li>
</ul>
</nav>

<article id="side">
<figure>

<figcaption>Home</figcaption>
</figure>

<p>A wonderful serenity has taken possession of my ...ssssssssss
ssssssssssssssssssssssssssssssssssssssssssssssssssssssssss
ssssssssssssssssssssssssssssssssssssssssssssssssssssssss
ssssssssssssssssssssssssssssssssssssssssssssss</p>

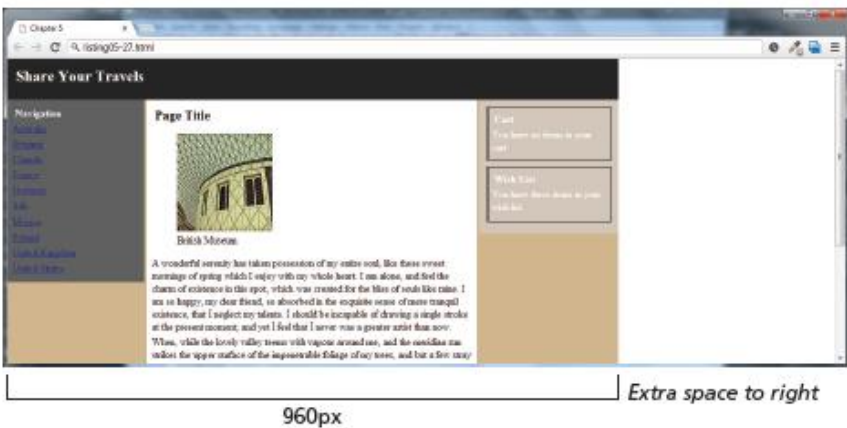
</article>

<p style = "clear:left" >When, while the lovely valley ...wwwwwwwwwwwwwwww
wwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwww
wwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwww
wwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwww</p>
</div>
</body>
</html>
```

One of the main problems faced by web designers is that the size of the screen used to view the page can vary. Some users will visit a site on a 21-inch wide screen monitor that can display 1920×1080 pixels (px); others will visit it on an older iPhone with a 3.5 screen and a resolution of 320×480 px. Users with the large monitor might expect a site to take advantage of the extra size; users with the small monitor will expect the site to scale to the smaller size and still be usable.

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. A common width used is something in the 960 to 1000 pixel range, which fits nicely in the common desktop monitor resolution (1024 × 768). This content may be positioned on the left or the center of the monitor.



Fixed layouts are created using pixel units, typically with the entire content within a <div> container whose width property has been set to some width.

```
<style>
div#wrapper {
width: 960px;
background_color: tan;
}
</style>
```

```
<body>
<div id="wrapper">
.....
</div>
```

```
.....
</body>
```

The advantage of a fixed layout –

- easy to produce
- predictable visual result
- optimized for typical desktop monitors

The disadvantage of a fixed layout –

- For larger screens, there may be an excessive amount of blank space to the left and/or right of the content.
- When the browser window is less than the fixed width; the user will have to horizontally scroll to see all the content.
- If smaller mobile devices are used, more horizontal scrolling has to be done

Liquid Layout

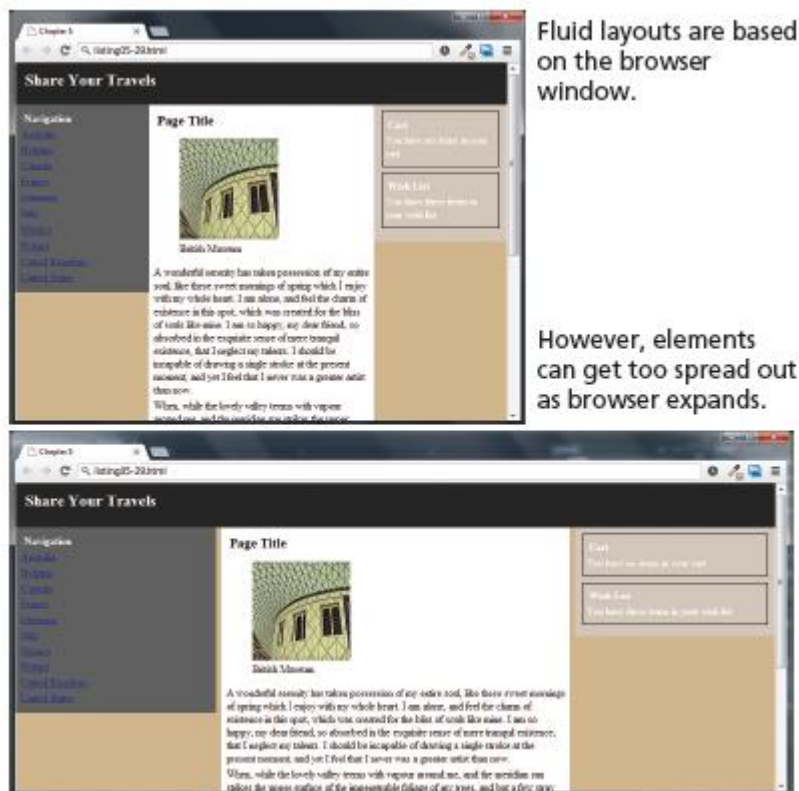
In Liquid Layout, the widths are not specified using pixels, but percentage values. Percentage values in CSS are a percentage of the current browser width, so a layout in which all widths expressed as percentages should adapt to any browser size.

The advantage of a liquid layout –

- Adapts to different browser sizes, so there is neither wasted white space nor any need for horizontal scrolling

The disadvantage of a liquid layout –

- more difficult to create because some elements, such as images, have fixed pixel sizes.
- The screen may grow or shrink dramatically.



Other Layout Approaches

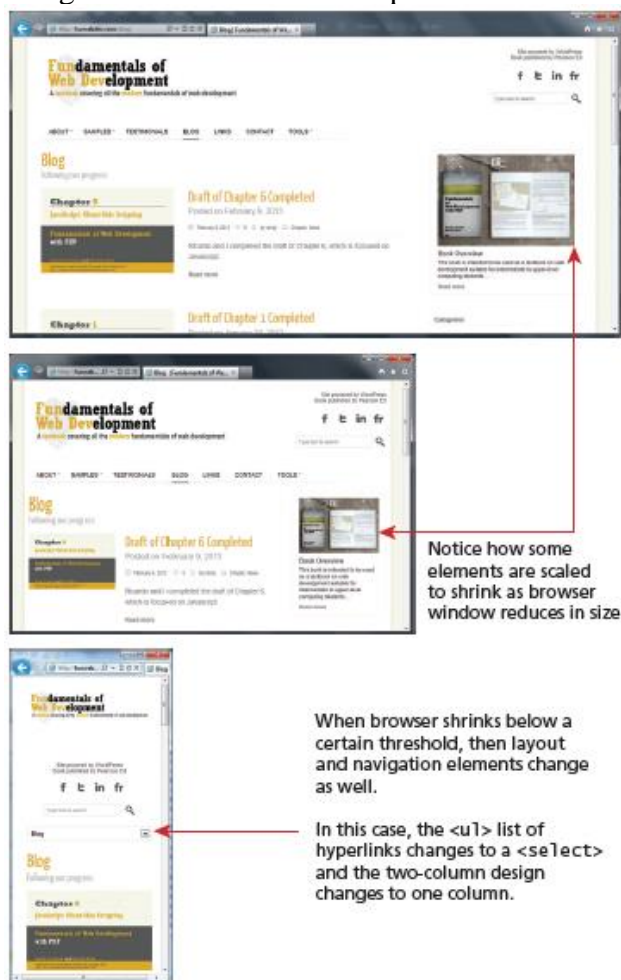
While the fixed and liquid layouts are the two basic paradigms for page layout, there are some other approaches that combine the two layout styles. Most of the other approaches are **hybrid layout**, where there is a fixed layout and liquid layout.

Fixed layout is commonly used for a sidebar column containing graphic advertising images that must always be displayed and which always are the same width. But liquid layout are used for the main content or navigation areas, with perhaps min and max size limits in pixels set for the navigation areas.

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, as shown in the figure below.



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

Responsive designs begin with a liquid layout, in which most elements have their widths specified as percentages. Making images scale in size is done as follows:

```
img {  
max-width: 100%;  
}
```

But this does not change the downloaded size of the image; it only shrinks or expands its visual display to fit the size of the browser window, never expanding beyond its actual dimensions.

Setting Viewports

A key technique in creating responsive layouts is the ability of current mobile browsers to shrink or grow the web page to fit the width of the screen. 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. The mobile Safari browser introduced the viewport `<meta>` tag as a way for developers to control the size of that initial viewport.

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

By setting the viewport as above, the page is telling the browser that no scaling is needed, and to make the viewport as many pixels wide as the device screen width. This means that if the device has a screen that is 320 px wide, the viewport width will be 320 px; if the screen is 480 px, then the viewport width will be 480 px.

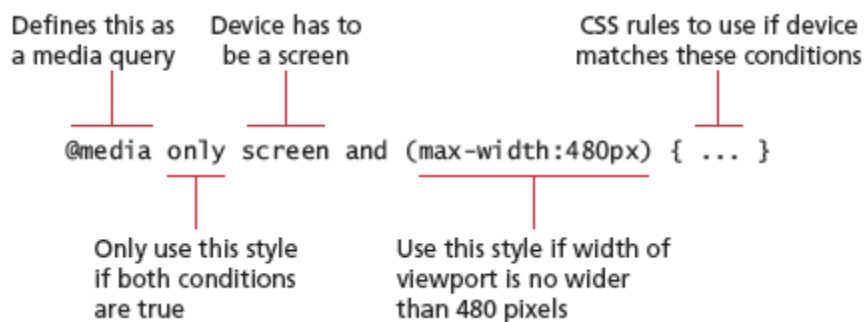


Media Queries

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. Use these queries to look at the capabilities of the device, and then define CSS rules to target that device.

Example of media query

@media only screen and (max-width: 480px) {.....}



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.

Few elements of the browser features that can be examined with media queries are –

Feature	Description
width	Width of the viewport
height	Height of the viewport
device-width	Width of the device
device-height	Height of the device
orientation	Whether the device is portrait or landscape
color	The number of bits per color

Contemporary responsive sites will typically provide CSS rules for phone displays first, then tablets, then desktop monitors, an approach called **progressive enhancement**. The media queries can be within your CSS file or within the <link> element; the later requires more HTTP requests but results in more manageable CSS files.

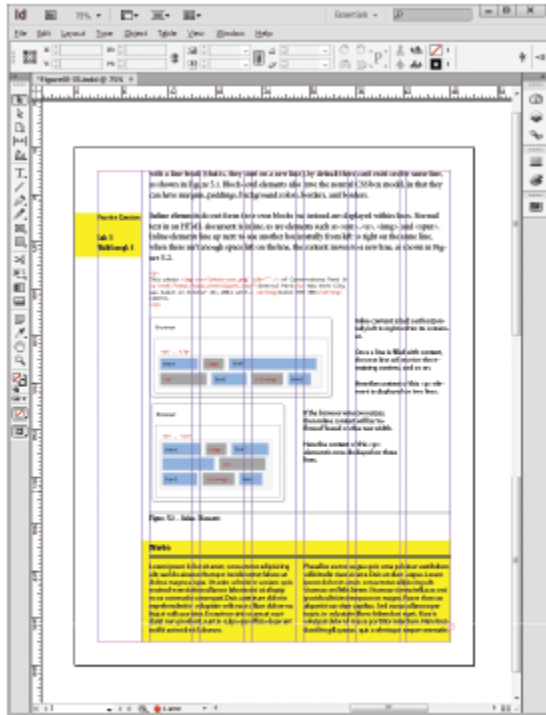
2.13 CSS Frameworks

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.

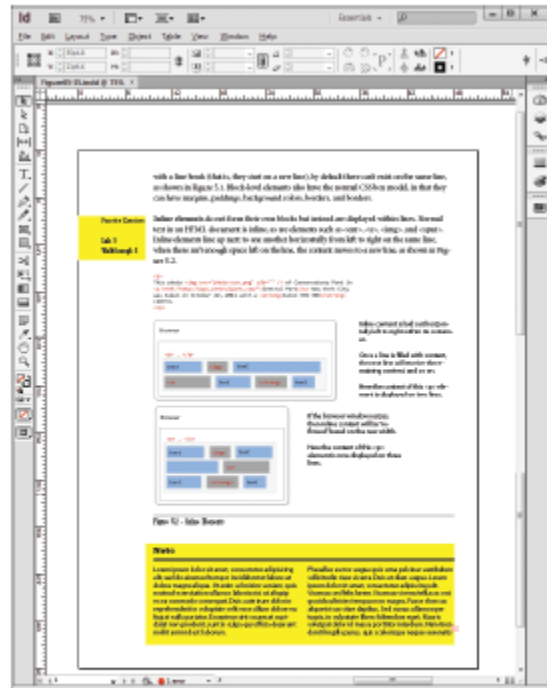
Grid Systems

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).

The most important of these capabilities is a grid system. Print designers typically use grids as a way to achieve visual uniformity in a design. In print design, the very first thing a designer may do is to construct, for instance, a 5- or 7- or 12-column grid in a page layout program. The rest of the document, whether it be text or graphics, will be aligned and sized according to the grid, as shown below.



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.

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 the site are then placed within these <div> elements.

Eg –

```
<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>
```

The above code creates a three column layout similar as shown in the above figure. The frameworks allow columns to be nested, making it quite easy to construct the most complex of layouts.

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. One of the best ways to see the power of a CSS preprocessor is with colors. Most sites make use of some type of color scheme, perhaps four or five colors. Many items will have the same color.

As shown in the below figure, the background color of the .box class, the text color in the <footer> element, the border color of the <fieldset>, and the text color for placeholder text within the <textarea> element, might all be set to #796d6d. The trouble with regular CSS is that when a change needs to be made, then some type of copy and replace is necessary, which always leaves the possibility that a change might be made to the wrong elements. Similarly, it is common for different site elements to have similar CSS formatting, for instance, different boxes to have the same padding.

In a programming language, a developer can use variables, nesting, functions, or inheritance to handle duplication and avoid copy-and-pasting and search-and-replacing. CSS preprocessors such as LESS, SASS, and Stylus provide this type of functionality.

```
$colorSchemeA: #796d6d;  
$colorSchemeB: #9c9c9c;  
$paddingCommon: 0.25em;
```

```
footer {  
  background-color: $colorSchemeA;  
  padding: $paddingCommon * 2;  
}
```

```
@mixin rectangle($colorBack, $colorBorder) {  
  border: solid 1pt $colorBorder;  
  margin: 3px;  
  background-color: $colorBack;  
}
```

```
fieldset {  
  @include rectangle($colorSchemeB, $colorSchemeA);  
}
```

```
.box {  
  @include rectangle($colorSchemeA, $colorSchemeB);  
  padding: $paddingCommon;  
}
```

SASS source file, e.g. source.scss

This example uses SASS (Syntactically Awesome Stylesheets). Here three variables are defined.

You can reference variables elsewhere. SASS also supports math operators on its variables.

A mixin is like a function and can take parameters. You can use mixins to encapsulate common styling.

A mixin can be referenced/called and passed parameters.

SASS Processor

The processor is some type of tool that the developer would run.

```
footer {  
  padding: 0.50em;  
  background-color: #796d6d;  
}
```

```
fieldset {  
  border: solid 1pt #796d6d;  
  margin: 3px;  
  background-color: #9c9c9c;  
}
```

```
.box {  
  border: solid 1pt #9c9c9c;  
  margin: 3px;  
  background-color: #796d6d;  
  padding: 0.25em;  
}
```

Generated CSS file, e.g., styles.css

The output from the processor is a normal CSS file that would then be referenced in the HTML source file.