

1. HTML
2. CSS
3. JAVA SCRIPT
4. J QUERY

Mr.Somasekhar Reddy  
Naresh Technologies

**SRI RAGHAVENDRA XEROX**

*Software Languages Material Available*

Beside Bangalore Ayyangar Bakery, Opp. C DAC, Ameerpet, Hyderabad.

Cell: 9951596199



Rs = 20/-

## What is HTML?

HTML stands for **Hypertext Markup Language**, and it is the most widely used language to write Web Pages. As its name suggests, HTML is a markup language.

- **Hypertext** refers to the way in which Web pages (HTML documents) are linked together. When you click a link in a Web page, you are using hypertext.
- **Markup Language** describes how HTML works. With a markup language, you simply "mark up" a text document with tags that tell a Web browser how to structure it to display.

All you need to do to use HTML is to learn what type of markup to use to get the results you want.

## Why to lean HTML?

It is possible to create webpages without knowing anything about the HTML source behind the page.

There are excellent editors on the market that will take care of the HTML parts. All you need to do is layout the page.

However, if you want to make it above average in webdesign, it is strongly recommended that you understand these tags.

The most important benefits are:

You can use tags the editor does not support.

You can read the code of other people's pages, and "borrow" the cool effects.

You can do the work yourself, when the editor simply refuses to create the effects you want.

## Words to Know

- **Tag** - tag is a command the web browser interprets. Tags look like this: <tag>
- **Element** - A complete tag, having an opening <tag> and a closing </tag>.
- **Attribute** - Used to modify the value of the HTML element. Elements will often have multiple attributes.

## HTML Elements

An HTML element is everything from the start tag to the end tag:

Start tag *	Element content	End tag *
<p>	This is a paragraph	</p>

```
<a href="default.htm" > This is a link </a>
```

```
<br />
```

\* The start tag is often called the **opening tag**. The end tag is often called the **closing tag**.

## HTML Element Syntax

- An HTML element starts with a **start tag / opening tag**
- An HTML element ends with an **end tag / closing tag**
- The **element content** is everything between the start and the end tag
- Some HTML elements have **empty content**
- Empty elements are **closed in the start tag**
- Most HTML elements can have **attributes**

## Nested HTML Elements

Most HTML elements can be nested (can contain other HTML elements).

HTML documents consist of nested HTML elements.

## Don't Forget the End Tag

Some HTML elements might display correctly even if you forget the end tag:

```
<p>This is a paragraph<br/><p>This is a paragraph
```

The example above works in most browsers, because the closing tag is considered optional.

Never rely on this. Many HTML elements will produce unexpected results and/or errors if you forget the end tag .

## Empty HTML Elements

HTML elements with no content are called empty elements.

<br> is an empty element without a closing tag (the <br> tag defines a line break).

**Tip:** In XHTML, all elements must be closed. Adding a slash inside the start tag, like <br />, is the proper way of closing empty elements in XHTML (and XML).

## Use Lowercase Tags

HTML tags are not case sensitive: <P> means the same as <p>. Many web sites use uppercase HTML tags.

World Wide Web Consortium (W3C) **recommends** lowercase in HTML 4, and **demands** lowercase tags in XHTML.

## Attributes

An attribute is used to define the characteristics of an element and is placed inside the element's opening tag. All attributes are made up of two parts: a name and a value:

The value of the attribute should be put in double quotation marks, and is separated from the name by the equals sign.

Many HTML tags have a unique set of their own attributes. Right now we want to focus on a set of generic attributes.

## Core Attributes:

The four core attributes that can be used on the majority of HTML elements (although not all) are:

- id
- title
- class
- style

### The id Attribute:

The *id* attribute can be used to uniquely identify any element within a page ( or style sheet ). There are two primary reasons that you might want to use an id attribute on an element:

- If an element carries an id attribute as a unique identifier it is possible to identify just that element and its content.
- If you have two elements of the same name within a Web page (or style sheet), you can use the id attribute to distinguish between elements that have the same name.

For now, the id attribute could be used to distinguish between two paragraph elements, like so:

```
<p id="html">This para explains what is HTML</p>
<p id="css">This para explains what is Cascading Style Sheet</p>
```

Note that there are some special rules for the value of the id attribute, it must:

- Begin with a letter (A.Z or a.z) and can then be followed by any number of letters, digits (0.9), hyphens, underscores, colons, and periods.
- Remain unique within that document; no two attributes may have the same value within that HTML document.

### The title Attribute:

The *title* attribute gives a suggested title for the element. The behavior of this attribute will depend upon the element that carries it, although it is often displayed as a tooltip or while the element is loading.

### The class Attribute:

The *class* attribute is used to associate an element with a style sheet, and specifies the class of element. You learn more about the use of the class attribute when you will learn Cascading Style Sheet (CSS). The value of the attribute may also be a space-separated list of class names. For example:

```
class="className1 className2 className3"
```

### The style Attribute:

The *style* attribute allows you to specify CSS rules within the element. For example:

```
<p style="font-family:arial; color:#FF0000;">Some text...</p>
```

### Generic Attributes:

Here's a table of some other attributes that are readily usable with many of HTML's tags.

Attribute	Options	Function
align	right, left, center	Horizontally aligns tags
valign	top, middle, bottom	Vertically aligns tags within an HTML element.
bgcolor	numeric, hexidecimal, RGB values	Places a background color behind an element
background	URL	Places a background image behind an element
Id	User Defined	Names an element for use with Cascading Style Sheets.
Class	User Defined	Classifies an element for use with Cascading Style Sheets.
width	Numeric Value	Specifies the width of tables, images, or table cells.
height	Numeric Value	Specifies the height of tables, images, or table cells.
Title	User Defined	"Pop-up" title for your elements.

We will see related examples as we will proceed to study other HTML tags.

## Use Lowercase Attributes

Attribute names and attribute values are case-insensitive.

However, the World Wide Web Consortium (W3C) recommends lowercase attributes/attribute values in their HTML 4 recommendation.

Newer versions of (X)HTML will demand lowercase attributes.

## What do I need to create HTML?

- Computer
- Text or HTML editor.
  - Adobe Dreamweaver.
  - SeaMonkey, Coffee Cup (Windows) and TextPad (Windows).
  - Notepad (for Windows), Pico (for Linux), or Simpletext/TextEdit/Text Wrangler (Mac).
- Web Browser.
  - Internet Explorer, Firefox, Chrome, opera ...etc.

## Do I need to be online?

No, you do not need to be online to create web pages. You can create web pages on your local machine. You only need to go online when you want to publish your web page to the web

## HTML Documents = Web Pages

- HTML documents describe web pages
- HTML documents contain HTML tags and plain text
- HTML documents are also called web pages

## .HTM or .HTML File Extension?

When you save an HTML file, you can use either the .htm or the .html file extension. There is no difference, it is entirely up to you.

## Basic HTML Document Structure

The basic structure for all HTML documents is simple and should include the following minimum elements or tags:

- Doctype
- <html> - The main container for HTML pages
- <head> - The container for page header information
- <title> - The title of the page
- <body> - The main body of the page

Remember that before an opening <html> tag, an XHTML document can contain the optional XML declaration, and it should always contain a DOCTYPE declaration indicating which version of XHTML it uses.

## Doctypes

A doctype declaration refers to the rules for the markup language, so that the browsers render the content correctly.

### Example

An HTML document with a doctype of HTML 4.01 Transitional:

```
1. <!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
2. "http://www.w3.org/TR/html4/loose.dtd">
3. <html>
4.   <head>
5.     <title>Title of the document</title>
6.   </head>
7.
8.   <body>
9.     The content of the document.....
10.  </body>
11.
12. </html>
```

### HTML Different Doctypes

The doctype declaration is not an HTML tag; it is an instruction to the web browser about what version of the markup language the page is written in.

The doctype declaration refers to a Document Type Definition (DTD). The DTD specifies the rules for the markup language, so that the browsers render the content correctly.

The doctype declaration should be the very first thing in an HTML document, before the `<html>` tag.

**Tip:** Always add a doctype to your pages. This helps the browsers to render the page correctly!

### HTML 4.01 Strict

This DTD contains all HTML elements and attributes, but does NOT INCLUDE presentational or deprecated elements (like font and center). Framesets are not allowed:

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN"
"http://www.w3.org/TR/html4/strict.dtd">
```

### HTML 4.01 Transitional

This DTD contains all HTML elements and attributes, INCLUDING presentational and deprecated elements (like font). Framesets are not allowed:

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"  
"http://www.w3.org/TR/html4/loose.dtd">
```

## HTML 4.01 Frameset

This DTD is equal to HTML 4.01 Transitional, but allows the use of frameset content:

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Frameset//EN"  
"http://www.w3.org/TR/html4/frameset.dtd">
```

### The <html> Element:

The <html> element is the containing element for the whole HTML document. Each HTML document should have one <html> and each document should end with a closing </html> tag.

Following two elements appear as direct children of an <html> element:

- <head>
- <body>

### The <head> Element:

The <head> element is just a container for all other header elements. It should be the first thing to appear after the opening <html> tag.

Each <head> element should contain a <title> element indicating the title of the document, although it may also contain any combination of the following elements, in any order:

- The <base> tag is used to create a "base" url for all links on the page.
- The <object> tag is designed to include images, JavaScript objects, Flash animations, MP3 files, QuickTime movies and other components of a page.
- The <link> tag is used to link to an external file, such as a style sheet or JavaScript file.
- The <style> tag is used to include CSS rules inside the document.
- The <script> tag is used to include JAVAScript or VBScript inside the document.
- The <meta> tag includes information about the document such as keywords and a description, which are particularly helpful for search applications.

### HTML Base tag.

```
1. <html>  
2. <head>  
3. <title>Practice HTML base tag</title>  
4.  
5. <base href="http://www.sekharit.com" target="_blank" />  
6.  
7. </head>  
8. <body>  
9.  
10. <p>This tag is using base path to display the image.</p>
```

```
11. 
12.
13. <p>This link will open in new window because we have used _blank in base tag.</p>
14. <p>Try to click this <a href="/php/index.htm">PHP Tutorial</a>
15.
16. </body>
17. </html>
```

## HTML Link tag

```
<link rel="stylesheet" type="text/css" href="/style.css" />
```

## HTML Style tag.

Style sheets describe how documents are presented on screens, in print, or perhaps how they are pronounced. W3C has actively promoted the use of style sheets on the Web since the Consortium was founded in 1994.

Cascading Style Sheets (CSS) is a style sheet mechanism that has been specifically developed to meet the needs of Web designers and users.

With CSS, you can specify a number of style properties for a given HTML element. Each property has a name and a value, separated by a colon (:). Each property declaration is separated by a semi-colon (;).

```
<p style="color:red;font-size:24px;">Using Style Sheet Rules</p>
```

There are three ways of using a style sheet in an HTML document:

## External Style Sheet:

If you have to give same look and feel to many pages then it is a good idea to keep all the style sheet rules in a single style sheet file and include this file in all the HTML pages. You can include a style sheet file into HTML document using <link> element. Below is an example:

```
<head>
  <link rel="stylesheet" type="text/css" href="yourstyle.css">
</head>
```

## Internal Style Sheet:

If you want to apply Style Sheet rules to a single document only then you can include those rules into that document only. Below is an example:

```
<head>
  <style type="text/css">
```

```
body{background-color: pink;}  
p{color:blue; 20px;font-size:24px;}  
</style>  
</head>
```

## Inline Style Sheet:

You can apply style sheet rules directly to any HTML element. This should be done only when you are interested to make a particular change in any HTML element only. To use inline styles you use the style attribute in the relevant tag. Below is an example:

```
<p style="color:red;font-size:24px;">Using Style Sheet Rules</p>
```

## HTML Script tag.

A *script* is a small piece of program that can add interactivity to your website. For example, a script could generate a pop-up alert box message, or provide a dropdown menu. This script could be Javascript or VBScript.

You can write your Event Handlers using any of the scripting language and then you can trigger those functions using HTML attributes.

## External Script:

If you have to use a single script functionality among many HTML pages then it is a good idea to keep that function in a single script file and then include this file in all the HTML pages. You can include a style sheet file into HTML document using <script> element. Below is an example:

```
<head>  
    <script src="yourfile.js" type="text/javascript" />  
</head>
```

## Internal Script:

You can write your script code directly into your HTML document. Usually we keep script code in header of the document using <script> tag, otherwise there is no restriction and you can put your source code anywhere in the document. You can specify whether to make a script run automatically (as soon as the page loads), or after the user has done something (like click on a link). Below is an example this would write a *Hello Javascript!* message as soon as the page loads.:

```
<head>  
    <title>Internal Script</title>  
</head>  
<body>
```

```
<script type="text/javascript">
    alert("Hello Javascript!")
</script>
</body>
```

This will produce following result:

Hello Javascript!

## Writing Event Handler:

It is very easy to write an event handler. Following example explains how to write an event handler. Let's write one simple function *myAlert* in the header of the document. We will call this function when any user will bring mouse over a paragraph written in the example.

```
<head>
    <title>Event Handler Example t</title>
    <script type="text/javascript">
        function myAlert()
        {
            alert("I am an event handler....");
            return;
        }
    </script>
</head>
<body>
    <span onmouseover="myAlert();">
        Bring your mouse here to see an alert
    </span>
</body>
```

Now this will produce following result. Bring your mouse over this line and see the result:

Bring your mouse here to see an alert

## The **<noscript>** Element:

You can also provide alternative info for users whose browsers don't support scripts and for users who have disabled scripts. You do this using the **<noscript>** tag.

```
<noscript >
    Please enable javascript
</noscript>
```

## Default Scripting Language

You can specify a default scripting language for all your *script* tags to use. This saves you from having to specify the language everytime you use a script tag within the page. Below is the example:

```
<meta http-equiv="Content-Script-Type" content="text/JavaScript" />
```

## HTML Meta tag.

HTML lets you specify metadata - information about a document rather than document content -in a variety of ways. The META element can be used to include name/value pairs describing properties of the HTML document, such as author, Expiry Date, a list of key words, author etc.

Metadata provided by using meta tag is a very important part of the web. It can assist search engines in finding the best match when a user performs a search. Search engines will often look at any metadata attached to a page - especially keywords - and rank it higher than another page with less relevant metadata, or with no metadata at all.

## Adding Meta Tags to Your Documents:

You can add metadata to your web pages by placing *<meta>* tags between the *<head>* and *</head>* tags. The can include the following attributes:

Attribute	Description
Name	Name for the property. Can be anything. Examples include, keywords, description, author, revised, generator etc.
content	Specifies the property's value.
scheme	Specifies a scheme to use to interpret the property's value (as declared in the content attribute).
http-equiv	Used for http response message headers. For example http-equiv can be used to refresh the page or to set a cookie. Values include content-type, expires, refresh and set-cookie.

**NOTE:** Core attributes for all the elements are discussed in next chapter.

## Meta Tag Examples:

Let's see few important usage of Meta Tags.

## Specifying Keywords:

We specify keywords which will be used by the search engine to search a web page. So using following tag you can specify important keywords related to your page.

```
<head>
  <meta name="keywords" content="HTML, meta tags, metadata" />
</head>
```

## Document Description:

This is again important information and many search engine use this information as well while searching a web page. So you should give an appropriate description of the page.

```
<head>
  <meta name="description" content="Learn about Meta Tags." />
</head>
```

## Document Revision date:

This information tells about last time the document was updated.

```
<head>
  <meta name="revised" content="Sekharit, 6/12/2006" />
</head>
```

## Document Refreshing:

You can specify a duration after which your web page will keep refreshing. If you want your page keep refreshing after every 10 seconds then use the following syntax.

```
<head>
  <meta http-equiv="refresh" content="10" />
</head>
```

## Page Redirection:

You can specify a page redirection using Meta Tag. Following is an example of redirecting current page to another page. You can specify a duration after which page will be redirected.

```
<head>
  <meta http-equiv="refresh" content="10; url=http://www.sekharit.com" />
</head>
```

If you don't provide a duration then page will be redirected immediately.

## Setting Cookies:

You can use Meta Tag to store cookies on client side later information can be used by then Web Server to track a site visitor.

```
<head>
  <meta http-equiv="cookie" content="userid=xyz;
    expires=Wednesday, 08-Aug-00 23:59:59 GMT; />
</head>
```

If you do not include the expiration date and time, the cookie is considered a session cookie and will be deleted when the user exits the browser.

## Setting Author Name:

You can set an author name in a web page using Meta Tag. See an example below:

```
<head>
  <meta name="author" content="Mahnaz Mohtashim" />
</head>
```

If you do not include the expiration date and time, the cookie is considered a session cookie and will be deleted when the user exits the browser.

## HTML Comment

HTML Comment lines are indicated by the special beginning tag <!-- and ending tag --> placed at the beginning and end of EVERY line to be treated as a comment.

Comments do not nest, and the double-dash sequence "--" may not appear inside a comment except as part of the closing --> tag. You must also make sure that there are no spaces in the start-of-comment string.

For example: Given line is a valid comment in HTML

```
<!-- This is commented out -->
```

But following line is not a valid comment and will be displayed by the browser. This is because there is a space between the left angle bracket and the exclamation mark.

```
< !-- This is commented out -->
```

Be careful if you use comments to "comment out" HTML that would otherwise be shown to the user, since some older browsers will still pay attention to angle brackets inside the comment and close the comment prematurely -- so that some of the text that was supposed to be inside the comment mistakenly appears as part of the document.

## Multiline Comments:

You have seen how to comment a single line in HTML. You can comment multiple lines by the special beginning tag <!-- and ending tag --> placed before the first line and end of the lastline to be treated as a comment.

For example:

```
<!--  
This is a multiline comment <br />  
and can span through as many as lines you like.  
-->
```

## Using Comment tag

There are few browsers who supports <comment> tag to comment a part of code.

```
<p>This is <comment>not</comment> Internet Explorer.</p>
```

## Paragraph Tag <p>

Publishing any kind of written works requires the use of a paragraph. The paragraph tag is very basic and a great introductory tag for beginner's because of its simplicity.

The <p> tag defines a paragraph. Using this tag places a blank line above and below the text of the paragraph. These automated blank lines are examples of how a tag "marks" a paragraph and the web browser automatically understands how to display the paragraph text because of the paragraph tag.

### HTML Code:

```
<p>Avoid losing floppy disks with important school...</p>  
<p>For instance, let's say you had a HUGE school...</p>
```

## Two HTML Paragraphs:

Avoid losing floppy disks with important school/work projects on them. Use the web to keep your content so you can access it from anywhere in the world. It's also a quick way to write reminders or notes to yourself. With simple html skills, (the ones you have gained so far) it is easy.

For instance, let's say you had a HUGE school or work project to complete. Off hand, the easiest way to transfer the documents from your house could be a 3.5" floppy disk. However, there is an alternative. Place your documents, photos, essays, or videos onto your web server and access them from anywhere in the world.

## HTML - Paragraph Justification

Paragraphs can be formatted in HTML much the same as you would expect to find in a word processing program. Here the align attribute is used to "justify" our paragraph.

### HTML Code:

```
<p align="justify">For instance, let's say you had a HUGE school or work...</p>
```

### Justified Text Alignment:

For instance, let's say you had a HUGE school or work project to complete. Off hand, the easiest way to transfer the documents from your house could be a 3.5" floppy disk. However, there is an alternative. Place your documents, photos, essays, or videos onto your web server and access them from anywhere in the world.

## HTML - Paragraph Centering

### HTML Code:

```
<p align="center">For instance, let's say you had a HUGE school or work...</p>
```

### Centered Text Alignment:

For instance, let's say you had a HUGE school or work project to complete. Off hand, the easiest way to transfer the documents from your house could be a 3.5" floppy disk. However, there is an alternative. Place your documents, photos, essays, or videos onto your web server and access them from anywhere in the world.

Each line of the paragraph has now been centered inside the display window.

## HTML - Paragraph Align Right

### HTML Code:

```
<p align="right">For instance, let's say you had a HUGE school or work...</p>
```

### Right Text Alignment:

For instance, let's say you had a HUGE school or work project to complete. Off hand, the easiest way to transfer the documents from your house could be a 3.5" floppy disk. However, there is an alternative. Place your documents, photos, essays, or videos onto your web server and access them from anywhere in the world.

Every line of the paragraph above is now aligned to the right hand side of the display box.

## Create Headings - The <hn> Elements:

Any documents starts with a heading. You use different sizes for your headings. HTML also have six levels of headings, which use the elements <h1>, <h2>, <h3>, <h4>, <h5>, and <h6>. While displaying any heading, browser adds one line before and after that heading.

Example:

```
<h1>This is heading 1</h1>
<h2>This is heading 2</h2>
<h3>This is heading 3</h3>
<h4>This is heading 4</h4>
<h5>This is heading 5</h5>
<h6>This is heading 6</h6>
```

This will display following result:

**This is heading 1**

**This is heading 2**

**This is heading 3**

**This is heading 4**

This is heading 5

*This is heading 6*

## Line Breaks

Line breaks are different than most of the tags we have seen so far. A line break ends the line you are currently on and resumes on the next line. Placing `<br />` within the code is the same as pressing the return key in a word processor. Use the `<br />` tag within the `<p>` (paragraph) tag.

### HTML Code:

```
<p>
    Will Mateson<br />
    Box 61<br />
    Cleveland, Ohio<br />
</p>
```

### Address:

Will Mateson  
Box 61  
Cleveland, Ohio

We have created a possible address for a letter head. The line break tag will also come in handy toward the end of our letter.

## HTML Code:

```
<p>Sincerely,<br />
<br />
<br />
Kevin Sanders<br />
Vice President</p>
```

## Closing Letter:

Sincerely,

Kevin Sanders

*Vice President*

## HTML Horizontal Rule

Use the `<hr />` tag to display lines across the screen. Note: the horizontal rule tag has no ending tag like the line break tag.

## HTML Code::

```
<hr />
Use
<hr /><hr />
Them
<hr />
Sparingly
<hr />
```

## Display::

---

Use

---

Them

---

Sparingly

---

Aside from our exaggerated example, the horizontal rule tag can come in handy when publishing work. A table of contents or perhaps a bibliography.

**HTML Code:**

```
<hr />
<p>1. "The Hobbit", JRR Tolkein.<br />
2. "The Fellowship of the Ring" JRR Tolkein.</p>
```

1. "The Hobbit", JRR Tolkein.
2. "The Fellowship of the Ring" JRR Tolkein.

**Lists:**

As you can see, all this tag does is draw a line across your content, and used properly, its results can be outstanding.

You can list out your items, subjects or menu in the form of a list. HTML gives you three different types of lists.

- <ul> - An unordered list. This will list items using bullets
- <ol> - A ordered list. This will use different schemes of numbers to list your items
- <dl> - A definition list. This is arrange your items in the same way as they are arranged in a dictionary.

**HTML Unordered Lists:**

An unordered list is a collection of related items that have no special order or sequence. The most common unordered list you will find on the Web is a collection of hyperlinks to other documents.

This list is created by using <ul> tag. Each item in the list is marked with a bullet. The bullet itself comes in three flavors: squares, discs, and circles. The default bullet displayed by most web browsers is the traditional full disc.

One Movie list is given below:

```
<center>
  <h2>Movie List</h2>
</center>

<ul>
  <li>Ram Teri Ganga Meli</li>
  <li>Mera Naam Jocker</li>
  <li>Titanic</li>
  <li>Ghost in the ship</li>
</ul>
```

This will produce following result:

## Movie List

- Ram Teri Ganga Meli
- Mera Naam Jocker
- Titanic
- Ghost in the ship

You can use *type* attribute to specify the type of bullet you like. By default its is a disc. Following are the possible way:

```
<ul type="square">
<ul type="disc">
<ul type="circle">
```

<code>&lt;ul type="square"&gt;</code>	<code>&lt;ul type="disc"&gt;</code>	<code>&lt;ul type="circle"&gt;</code>
<ul style="list-style-type: none"> <li>■ Hindi</li> <li>■ English</li> <li>■ Maths</li> <li>■ Physics</li> </ul>	<ul style="list-style-type: none"> <li>● Hindi</li> <li>● English</li> <li>● Maths</li> <li>● Physics</li> </ul>	<ul style="list-style-type: none"> <li>○ Hindi</li> <li>○ English</li> <li>○ Maths</li> <li>○ Physics</li> </ul>

## HTML Ordered Lists:

The typical browser formats the contents of an ordered list just like an unordered list, except that the items are numbered instead of bulleted. The numbering starts at one and is incremented by one for each successive ordered list element tagged with `<li>`

This list is created by using `<ol>` tag. Each item in the list is marked with a number.

One Movie list is given below:

```
<center>
    <h2>Movie List</h2>
</center>

<ol>
    <li>Ram Teri Ganga Meli</li>
    <li>Mera Naam Jocker</li>
    <li>Titanic</li>
    <li>Ghost in the ship</li>
</ol>
```

This will produce following result:

### Movie List

1. Ram Teri Ganga Meli
2. Mera Naam Jocker
3. Titanic
4. Ghost in the ship

You can use *type* attribute to specify the type of numbers you like. By default its is a generic numbers. Following are the other possible way:

```
<ol type="I"> - Upper-Case Numerals.  
<ol type="i"> - Lower-Case Numerals.  
<ol type="a"> - Lower-Case Letters.  
<ol type="A"> - Upper-Case Letters.
```

<b>&lt;ol type="I"&gt;</b>	<b>&lt;ol type="i"&gt;</b>	<b>&lt;ol type="a"&gt;</b>	<b>&lt;ol type="A"&gt;</b>
I. Hindi II. English III. Maths IV. Physics	i. Hindi ii. English iii. Maths iv. Physics	a. Hindi b. English c. Maths d. Physics	A. Hindi B. English C. Maths D. Physics

You can use *start* attribute to specify the beginning of any index. By default its is a first number or character. In the following example index starts from 5:

```
<center>  
    <h2>Movie List</h2>  
</center>  
<ol start="5">  
    <li>Ram Teri Ganga Meli</li>  
    <li>Mera Naam Jocker</li>  
    <li>Titanic</li>  
    <li>Ghost in the ship</li>  
</ol>
```

This will produce following result:

### Movie List

5. Ram Teri Ganga Meli
6. Mera Naam Jocker
7. Titanic
8. Ghost in the ship

### HTML Definition Lists:

HTML and XHTML also support a list style entirely different from the ordered and unordered lists we have discussed so far - definition lists . Like the entries you find in a dictionary or encyclopedia, complete with text, pictures, and other multimedia elements, the Definition List is the ideal way to present a glossary, list of terms, or other name/value list.

Definition List makes use of following three tags.

- <dl> - Defines the start of the list
- <dt> - A term
- <dd> - Term definition
- </dl> - Defines the end of the list

Example:

```
<dl>

<dt><b>HTML</b></dt>
<dd>This stands for Hyper Text Markup Language</dd>

<dt><b>HTTP</b></dt>
<dd>This stands for Hyper Text Transfer Protocol</dd>

</dl>
```

This will produce following result:

#### HTML

This stands for Hyper Text Markup Language

#### HTTP

This stands for Hyper Text Transfer Protocol

## HTML - Formatting Elements w/ Tags

As you begin to place more and more elements onto your web site, it will become necessary to make minor changes to the formatting of those elements. In our [HTML Attributes](#) lesson we discussed ways to add some flavor with attributes and align elements within other elements. Several tags exist to further amplify text elements. These formatting tags can make text bold, italic, sub/superscripted, and more.

### Bold, Italic and More

#### HTML Code:

```
<p>An example of <b>Bold Text</b></p>
<p>An example of <em>Emphasized Text</em></p>
<p>An example of <strong>Strong Text</strong></p>
<p>An example of <i>Italic Text</i></p>
<p>An example of <sup>superscripted Text</sup></p>
<p>An example of <sub>subscripted Text</sub></p>
<p>An example of <del>struckthrough Text</del></p>
<p>An example of <code>Computer Code Text</code></p>
```

#### HTML Formatting:

An example of **Bold Text**

An example of *Emphasized Text*

An example of **Strong Text**

An example of *Italic Text*

An example of superscripted Text

An example of subscripted Text

An example of struckthrough Text

An example of Computer Code Text

All of these tags add a pinch of flavor to paragraph elements. They can be used with any text type element.

### Preserve Formatting - The <pre> Element:

Sometimes you want your text to follow the exact format of how it is written in the HTML document. In those cases, you can use the preformatted tag (<pre>).

Any text between the opening <pre> tag and the closing </pre> tag will preserve the formatting of the source document.

```
<pre>
    function testFunction( strText ){
        alert (strText)
    }
</pre>
```

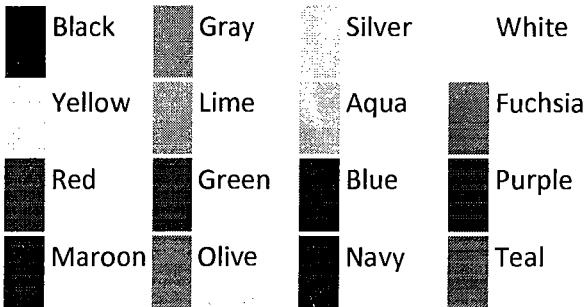
This will produce following result:

```
function testFunction( strText ) {
    alert (strText)
}
```

## HTML Color Coding System - Color Names

There are 3 different methods to set color. The simplest being the **Generic** terms of colors. Examples: black, white, red, green, and blue. Generic colors are preset HTML coded colors where the value is simply the name of each color. Here is a sample of the most widely supported colors and their respective name values.

### The 16 Basic Colors:



## HTML Coloring System - RGB Values

We do not recommend that you use RGB for safe web design because non-IE browsers do not support HTML RGB. However, if you plan on learning CSS then you should glance over this topic.

RGB stands for Red, Green, Blue. Each can have a value from 0 (none of that color) to 255 (fully that color). The format for RGB is - `rgb(RED, GREEN, BLUE)`, just like the name implies. Below is an example of RGB in use, but if you are not using a browser that supports it, do not worry, that is just one of the problems with HTML RGB.

### Red, Green, and Blue Values:

`bgcolor="rgb(255,255,255)"` White

`bgcolor="rgb(255,0,0)"` Red

`bgcolor="rgb(0,255,0)"` Green

`bgcolor="rgb(0,0,255)"` Blue

## HTML Coloring System - Hexadecimal

The hexadecimal system is complex and difficult to understand at first. Rest assured that the system becomes much, MUCH easier with practice and as a blossoming web developer, it is critical to understand hexadecimals

to be capable of using them in your own web publications. They are far more reliable and widely compatible among web browsers and are the standard for colors on the internet.

A hexadecimal is a 6 digit representation of a color. The first two digits(RR) represent a red value, the next two are a green value(GG), and the last are the blue value(BB).

Here's a hexadecimal you might see in an HTML document.

### My First Hexadecimal:

`bgcolor="#RRGGBB"`

## HTML Color Code - Breaking the Code

The following table shows how letters are incorporated into the hexadecimal essentially extending the numbers system to 16 values. Hang in there it all makes sense shortly.

### Hexadecimal Color Values:

Decimal	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Hexadecimal	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F

*So use letters as numbers?* We will answer this question as we dive into the converting hexadecimals to regular numbers. Let's have a look at real Hexadecimal.

### A Real Hexadecimal:

`bgcolor="#FFFFFF"`

The letter "F" is the maximum amount we can send each color and as you may deduce, this color (#FFFFFF) represents the color white. A formula exists to calculate the numeric equivalent of a hexadecimal.

## HTML - Font and Basefont

The `<font>` tag is used to add style, size, and color to the text on your site. Use the *size*, *color*, and *face* attributes to customize your fonts. Use a `<basefont>` tag to set all of your text to the same size, face, and color.

**The font and basefont tags are deprecated and should not be used. Instead, use css styles to manipulate your font.**

### Font Size

Set the size of your font with *size*. The range of accepted values is from 1(smallest) to 7(largest).The default size of a font is 3.

**HTML Code:**

```
<p>
<font size="5">Here is a size 5 font</font>
</p>
```

**Font Size:**

Here is a size 5 font.

**Font Color**

Set the color of your font with *color*.

**HTML Code:**

```
<font color="#990000">This text is hexcolor #990000</font>
<br />

<font color="red">This text is red</font>
```

**Font Color:**

This text is hexcolor #990000

This text is red

**Font Face**

Choose a different font face using any font you have installed. Be aware that if the user viewing the page doesn't have the font installed, they will not be able to see it. Instead they will default to Times New Roman. An option is to choose a few that are similar in appearance.

**HTML Code:**

```
<p>
<font face="Bookman Old Style, Book Antiqua, Garamond">This paragraph
has had its font...</font>
</p>
```

**Font Face:**

This paragraph has had its font formatted by the font tag!

**HTML Entities**

Some characters are reserved in HTML.

It is not possible to use the less than (<) or greater than (>) signs in your text, because the browser will mix them with tags.

To actually display reserved characters, we must use character entities in the HTML source code.

A character entity looks like this:

&entity\_name;

OR

&#entity\_number;

To display a less than sign we must write: &lt; or &#60;

**Tip:** The advantage of using an entity name, instead of a number, is that the name is easier to remember. However, the disadvantage is that browsers may not support all entity names (the support for entity numbers is very good).

## Non-breaking Space

A common character entity used in HTML is the non-breaking space (&nbsp;).

Browsers will always truncate spaces in HTML pages. If you write 10 spaces in your text, the browser will remove 9 of them, before displaying the page. To add spaces to your text, you can use the &nbsp; character entity.

## HTML Useful Character Entities

**Note:** Entity names are case sensitive!

Result Description	Entity Name	Entity Number
non-breaking space	&nbsp;	&#160;
< less than	&lt;	&#60;
> greater than	&gt;	&#62;
& ampersand	&amp;	&#38;
¢ cent	&cent;	&#162;
£ pound	&pound;	&#163;
¥ yen	&yen;	&#165;
€ euro	&euro;	&#8364;
§ section	&sect;	&#167;

© copyright &copy; ©  
® registered trademark &reg; ®  
™ trademark &trade; ™

## HTML Color - bgcolor

The bgcolor attribute is used to control the background of an HTML element, specifically page and table backgrounds. Bgcolor can be placed within several of the HTML tags. However, we suggest you only use it for your page's main background (<body>) and in tables.

### Syntax

```
<TAGNAME bgcolor="value">
```

Quick and dirty, here is how to change the background of your web page. Just use the bgcolor attribute in the <body> tag and you are golden.

### HTML Code:

```
<body bgcolor="Silver">  
<p>We set the background...</p>  
</body>
```

## HTML - Background

Images can be placed within elements of HTML. Tables, paragraphs, and bodys may all have a background image. To accomplish this, we use the background attribute as follows.

### HTML Code:

```
<table height="100" width="150"  
background="http://www.tizag.com/pics/htmlT/background.jpg" >  
<tr><td>This table has a background image</td></tr>  
</table>
```

## HTML - Background Repeat

In the first example we happen to be lucky because our image and our table had exactly the same size pixel dimensions. Everything looks great. When your HTML element is larger than the dimensions of your picture, the image simply begins to repeat itself.

### HTML Code:

```
<table height="200" width="300"  
background="http://www.tizag.com/pics/htmlT/background.jpg" >
```

```
<tr><td>This table has a background image</td></tr>
</table>
```

It is obvious this is often not the desired outcome, however, it can also be quite useful as you will see in the following example.

## HTML - Patterned Backgrounds

Repeating a generic image as a background doesn't have much practical use. We either need to find an image to fit exactly as our background or have an image editing program to adjust the dimensions of our image.

From a different angle, we can use this default attribute to our benefit say if we wanted to have some sort of pattern as our background. In an image editing program such as Adobe Photosop, or Paint Shop Pro, we could create a very small (perhaps 4X4 pixels) and create a couple of basic patterns.

### 4x4 Image:

Now here is the same image set as the background to our same table.

### HTML Code:

```
<table height="100" width="150"
      background="http://www.tizag.com/pics/htmlT/pattern.jpg" >
<tr><td>This table has a background patterned image</td></tr>
</table>
```

## HTML - Transparent Background

Another great technique, along the same lines as the patterned images, is that of transparent, colored backgrounds. Most image editors have some sort of transparency device to create images that appear see through. We're not going to cover how to do this with every single program, however, most of the time all you need to do is fill your canvas with the color you would like and then set the opacity to something below 100%. Then make sure you save your file as a gif not a jpeg, and all systems should be go.

Now that you have had the crash course on creating transparent files, you place them onto your websites the exact same way as you would a repeating background.

### HTML Code:

```
<table background="http://www.tizag.com/pics/htmlT/transparent.gif" >
<tr><td>This table has a red transparent background image</td></tr>
</table>
```

## HTML - Div Element(s)

The <div> tag is nothing more than a container for other tags. Much like the body tag, Div elements are block elements and work behind the scenes grouping other tags together. Use only the following attributes with your div element,

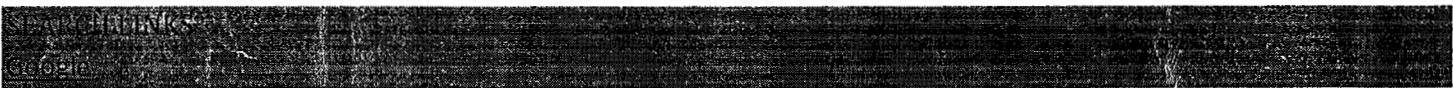
- id
- width
- height
- title
- style

For the purpose of this example, we have included the *style* attribute in order to color our div tag in order to bring a stronger visualization for our viewers.

### HTML Code:

```
<body>
<div style="background: green">
<h5>SEARCH LINKS</h5>
<a target="_blank" href="http://www.google.com">Google</a>
</div>
</body>
```

### HTML Div Element:



Above is a great visual about how a div plays the role of a container for other HTML elements, applying a background color/image is the only real way to visualize your div tags.

## HTML - Links and Anchors

The web got its spidery name from the plentiful connections between web sites. These connections are made using anchor tags to create links. Text, Images, and Forms may be used to create these links.

## HTML - Hypertext Reference (href)

The *href* attribute defines reference that the link refers to. Basically this is where the user will be taken if they wish to click this link.

Hypertext references can be Internal, Local, or Global.

- Internal - Links to anchors on the current page
- Local - Links to other pages within your domain

- Global - Links to other domains outside of your site

### HTML Code:

Internal - href="#anchorname"  
Local - href="../pics/picturefile.jpg"  
Global - href=http://www.sekharit.com/

## HTML - Text Links

Use the <a></a> tags to define the start and ending of an anchor. Decide what type of *href* attribute you need and place this attribute into the opening tag. The text you place between the opening and closing tags will be shown as the link on a page. Use the demonstration below as a reference.

### HTML Code:

```
<a href="http://www.sekharit.com/" target="_blank" >Tizag Home</a>
<a href="http://www.espn.com/" target="_blank" >ESPN Home</a>
<a href="http://www.yahoo.com/" target="_blank" >Yahoo Home</a>
```

### Global Link:

Tizag Home ESPN Home Yahoo Home

## HTML - Link Targets

The target attribute defines whether to open the page in a separate window, or to open the link in the current browser window.

### HTML Code:

target=" \_blank" Opens new page in a new browser window

\_self" Loads the new page in current window

\_parent" Loads new page into a frame that is superior to where the link lies

\_top" Loads new page into the current browser window, cancelling all frames

The example below shows how you would link to ESPN.COM, a popular sports web site. The *target* attribute is added to allow the browser to open ESPN in a new window, so that the viewer can remain at our web site. Here's the example.

### HTML Code:

```
<a href="http://www.ESPN.com" target="_blank">ESPN.COM</a>
```

**\_blank Target:**ESPN.COM**HTML - Anchors**

To link to sections of your existing page a name must be given to the anchor. In the example below, we've created a mini Table of Contents for this page. By placing blank anchors just after each heading, and naming them, we can then create reference links to those sections on this page as shown below.

First, the headings of this page contain blank, named anchors. They look like this.

**Tizag's Own Code:**

```
<h2>HTML Links and Anchors<a name="top"></a></h2>

<h2>HTML Text Links<a name="text"></a></h2>

<h2>HTML Email<a name="email"></a></h2>
```

Now create the reference links, placing the pound symbol followed by the name of the anchor in the href of the new link.

**Anchor Code:**

```
<a href="#top">Go to the Top</a>

<a href="#text">Learn about Text Links</a>

<a href="#email">Learn about Email Links</a>
```

**Local Links:**[Go to the Top](#)[Learn about Text Links](#)[Learn about Email Links](#)**HTML - Email Links**

Creating an email link is simple. If you want somebody to mail you about your site a good way to do it is place an email link with a subject already in place for them.

**HTML Code:**

```
<a href="mailto:email@sekharit.com?subject=Feedback" >Email@sekharit.com</a>
```

**Email Links:**[Email@sekharit.com](mailto:Email@sekharit.com)

In some circumstances it may be necessary to fill in the body of the Email for the user as well.

**HTML Code:**

```
<a href="mailto:email@sekharit.com?subject=Feedback&body=Sweet site!">  
Email@sekharit.com</a>
```

**Complete Email:**

Email@sekharit.com

**HTML - Download Links**

Placing files available for download is done in exactly the same fashion as placing text links. Things become complicated if we want to place image links available for download. The best solution for images is to use a thumbnail link that we discuss in the next lesson.

**HTML Code:**

```
<a href="http://www.sekharit.com/pics/htmlT/blanktext.zip">Text Document</a>
```

**Download a Text Document:**

Text Document

**HTML - Default Links; Base**

Use the `<base>` tag in the `head` element to set a default URL for all links on a page to go to. It's always a good idea to set a base tag just incase your links become bugged somewhere down the line. Usually set your base to your home page.

**HTML Code:**

```
<head>  
  
<base href="http://www.sekharit.com/">  
  
</head>
```

**HTML - Images**

Images are a staple of any web designer, so it is very important that you understand how to use them properly. Use the `<img />` tag to place an image on your web page.

**HTML Code:**

```

```

**Image:**

## HTML - Image src

Above we have defined the *src* attribute. Src stands for *source*, the source of the image or more appropriately, where the picture file is located. As with links described in a previous lesson, you may use any standard URL to properly point the src attribute to a local or external source.

There are two ways to define the source of an image. First you may use a standard URL.

(src=http://www.Tizag.com/pics/htmlT/sunset.gif) As your second choice, you may copy or upload the file onto your web server and access it locally using standard directory tree methods. (src="../sunset.gif") The location of this picture file is in relation to your location of your .html file.

### URL Types:

#### Local Src

#### Location Description

src="sunset.gif" picture file resides in same directory as .html file

src="../sunset.gif" picture file resides in previous directory as .html file

src="../pics/sunset.gif" picture file resides in the *pic* directory in a previous directory as .html file

A URL cannot contain drive letters, since a src URL is a relational source interpretation based on the location of your .html file and the location of the picture file. Therefore something like src="C:\\www\\web\\pics\\" will not work. Pictures must be uploaded along with your .html file to your web server.

Each method has its pros and cons, for instance using the URL of pictures on other sites poses a problem if the web master(s) of the other site happen to change the physical location of the picture file. Copying the file directly to your web server solves this problem, however, as you continue to upload picture files to your system, you may eventually run short on hard drive space. Use your best judgement to meet your needs.

## HTML - Alternative Attribute

The *alt* attribute specifies alternate text to be displayed if for some reason the browser cannot find the image, or if a user has image files disabled. Text only browsers also depend on the alt attribute since they cannot display pictures.

### HTML Code:

```

```

### Alternative Text:

## HTML - Image Height and Width

To define the height and width of the image, rather than letting the browser compute the size, use the *height* and *width* attributes.

### HTML Code:

```

```

### Height and Width:



Above we have defined the *src*, *height* and *width* attributes. By informing the browser of the image dimensions it knows to set aside a place for that image. Without defining an image's dimensions your site may load poorly; text and other images will be moved around when the browser finally figures out how big the picture is supposed to be and then makes room for the picture.

## Vertically and Horizontally Align Images

Use the align and valign attributes to place images within your body, tables, or sections.

1. align (Horizontal)
  - o right
  - o left
  - o center
2. valign (Vertical)
  - o top
  - o bottom
  - o center

Below is an example of how to align an image to the right of a paragraph.

### HTML Code:

```
<p>This is paragraph 1, yes it is...</p>

<p>



The image will appear along the...isn't it?

</p>

<p>This is the third paragraph that appears...</p>
```

## Image Wrap Around:

This is paragraph 1, yes it is. I think this paragraph serves as a nice example to show how this image alignment works.

The image will appear along the right hand side of the paragraph. As you can see this is very nice for adding a little eye candy that relates to the specified paragraph. If we were talking about beautiful tropical sunsets, this picture would be perfect. But we aren't talking about that, so it's rather a waste, isn't it?



This is the third paragraph that appears below the paragraph with the image!

## Images as Links

This will be a quick review of the [links - image lesson](#). Images are very useful for links and can be created with the HTML below.

### HTML Code:

```
<a href="http://www.tizag.com/">  
    
</a>
```

## Image Links:



Now your image will take you to our home page when you click it. Change it to your home page URL.

## HTML forms

HTML Forms are required when you want to collect some data from the site visitor. For example registration information: name, email address, credit card, etc.

A form will take input from the site visitor and then will post your back-end application such as CGI, ASP Script or PHP script etc. Then your back-end application will do required processing on that data in whatever way you like.

Form elements are like text fields, textarea fields, drop-down menus, radio buttons, checkboxes, etc. which are used to take information from the user.

A simple syntax of using `<form>` is as follows:

```
<form action="back-end script" method="posting method">  
  form elements like input, textarea etc.  
</form>
```

Most frequently used form attributes are:

- **name:** This is the name of the form.
- **action:** Here you will specify any script URL which will receive uploaded data.
- **method:** Here you will specify method to be used to upload data. It can take various values but most frequently used are GET and POST.
- **target:** It specifies the target page where the result of the script will be displayed. It takes values like \_blank, \_self, \_parent etc.
- **enctype:** You can use the enctype attribute to specify how the browser encodes the data before it sends it to the server. Possible values are like:
  - **application/x-www-form-urlencoded** - This is the standard method most forms use. It converts spaces to the plus sign and non-alphanumeric characters into the hexadecimal code for that character in ASCII text.
  - **multipart/form-data** - This allows the data to be sent in parts, with each consecutive part corresponding to a form control, in the order they appear in the form. Each part can have an optional content-type header of its own indicating the type of data for that form control.

There are different types of form controls that you can use to collect data from a visitor to your site.

- Text input controls
- Buttons
- Checkboxes and radio buttons
- Select boxes
- File select boxes
- Hidden controls
- Submit and reset button

## HTML Forms - Text Input Controls:

There are actually three types of text input used on forms:

- **Single-line text input controls:** Used for items that require only one line of user input, such as search boxes or names. They are created using the <input> element.
- **Password input controls:** Single-line text input that mask the characters a user enters.
- **Multi-line text input controls:** Used when the user is required to give details that may be longer than a single sentence. Multi-line input controls are created with the <textarea> element.

### Single-line text input controls:

Single-line text input controls are created using an <input> element whose type attribute has a value of text.

Here is a basic example of a single-line text input used to take first name and last name:

```
<form action="/cgi-bin/hello_get.cgi" method="get">
    First name: <input type="text" name="first_name" />
    <br>
```

```
Last name: <input type="text" name="last_name" />
<input type="submit" value="submit" />
</form>
```

This will produce following result:

First name:

Last name:

---

Following is the list of attributes for <input> tag.

- **type:** Indicates the type of input control you want to create. This element is also used to create other form controls such as radio buttons and checkboxes.
- **name:** Used to give the name part of the name/value pair that is sent to the server, representing each form control and the value the user entered.
- **value:** Provides an initial value for the text input control that the user will see when the form loads.
- **size:** Allows you to specify the width of the text-input control in terms of characters.
- **maxlength:** Allows you to specify the maximum number of characters a user can enter into the text box.

```
<html>
  <head>
    <title>Practice HTML input text tag</title>
  </head>
  <body>
    <p>Try with different input text</p>
    <form action="/cgi-bin/hello_get.cgi" method="get">
      First name: <input type="text" name="first_name" />
      <br>
      Last name: <input type="text" name="last_name" />
      <input type="submit" value="submit" />
    </form>
  </body>
</html>
```

## Password input controls::

This is also a form of single-line text input controls are created using an <input> element whose type attribute has a value of password.

Here is a basic example of a single-line password input used to take user password:

```
<form action="/cgi-bin/hello_get.cgi" method="get">
    Login : <input type="text" name="login" />
    <br>
    Password:<input type="text" name="password" />
    <input type="submit" value="submit" />
</form>
```

This will produce following result:

Login :

Password :

---

```
<html>
    <head>
        <title>Practice HTML input password tag</title>
    </head>
    <body>

        <form action="" method="get">
            Login : <input type="text" name="login" />
            <br>
            Password:<input type="text" name="password" />
            <input type="reset" value="reset" />
        </form>

    </body>
</html>
```

## Multiple-Line Text Input Controls:

If you want to allow a visitor to your site to enter more than one line of text, you should create a multiple-line text input control using the <textarea> element.

Here is a basic example of a multi-line text input used to take item description:

```
<form action="/cgi-bin/hello_get.cgi" method="get">
    Description : <br />
    <textarea rows="5" cols="50" name="description">
        Enter description here...
    </textarea>
    <input type="submit" value="submit" />
</form>
```

This will produce following result:

**Description :**

Following is the detail of above used attributes for <textarea> tag.

- **name:** The name of the control. This is used in the name/value pair that is sent to the server.
- **rows:** Indicates the number of rows of text area box.
- **cols:** Indicates the number of columns of text area box.

```
<html>
  <head>
    <title>Practice HTML input textarea tag</title>
  </head>
  <body>
    <p>Try with different cols and rows</p>
    <form action="" method="get">
      Description : <br />
      <textarea rows="5" cols="50" name="description">
        Enter description here...
      <input type="submit" value="submit" />
    </form>
  </body>
</html>
```

## HTML Forms - Creating Button:

There are various ways in HTML to create clickable buttons. You can create clickable button using <input> tag.

When you use the <input> element to create a button, the type of button you create is specified using the type attribute. The type attribute can take the following values:

- **submit:** This creates a button that automatically submits a form.
- **reset:** This creates a button that automatically resets form controls to their initial values.
- **button:** This creates a button that is used to trigger a client-side script when the user clicks that button.

Here is the example:

```
<form action="http://www.example.com/test.asp" method="get">
  <input type="submit" name="Submit" value="Submit" />
  <br /><br />
  <input type="reset" value="Reset" />
```

```
<input type="button" value="Button" />
</form>
```

This will produce following result:

You can use an image to create a button. Here is the syntax:

```
<form action="http://www.example.com/test.asp" method="get">
    <input type="image" name="imagebutton" src="URL" />
</form>
```

Here *src* attribute specifies a location of the image on your webserver.

You can use *<button>* element to create various buttons. Here is the syntax:

```
<form action="http://www.example.com/test.asp" method="get">
    <button type="submit">Submit</button>
    <br /><br />
    <button type="reset"> Reset </button>
    <button type="button"> Button </button>
</form>
```

This will produce following result:

## HTML Forms - Checkboxes Control:

Checkboxes are used when more than one option is required to be selected. They are created using *<input>* tag as shown below.

Here is example HTML code for a form with two checkboxes

```
<form action="/cgi-bin/checkbox.cgi" method="get">
    <input type="checkbox" name="maths" value="on"> Maths
    <input type="checkbox" name="physics" value="on"> Physics
    <input type="submit" value="Select Subject" />
</form>
```

The result of this code is the following form

Maths  Physics

Following is the list of important checkbox attributes:

- **type:** Indicates that you want to create a checkbox.
- **name:** Name of the control.
- **value:** The value that will be used if the checkbox is selected. More than one checkbox should share the same name only if you want to allow users to select several items from the same list.
- **checked:** Indicates that when the page loads, the checkbox should be selected.

```
<html>
  <head>
    <title>Practice HTML input checkbox tag</title>
  </head>
  <body>
    <p>Try with different checkbox and different attributes</p>
    <form action="" method="get">
      <input type="checkbox" name="maths" value="on" /> Maths
      <input type="checkbox" name="physics" value="on" /> Physics
      <input type="reset" value="reset" />
    </form>

  </body>
</html>
```

## HTML Forms - Raidobox Control:

Radio Buttons are used when only one option is required to be selected. They are created using `<input>` tag as shown below:

Here is example HTML code for a form with two radio button:

```
<form action="/cgi-bin/radiobutton.cgi" method="post">
  <input type="radio" name="subject" value="maths" /> Maths
  <input type="radio" name="subject" value="physics" /> Physics
  <input type="submit" value="Select Subject" />
</form>
```

The result of this code is the following form

Maths  Physics

Following is the list of important radiobox attributes:

- **type:** Indicates that you want to create a radiobox.
- **name:** Name of the control.
- **value:** Used to indicate the value that will be sent to the server if this option is selected.
- **checked:** Indicates that this option should be selected by default when the page loads.

```
<html>
  <head>
    <title>Practice HTML input radiobox tag</title>
  </head>
  <body>
    <p>Try with different radiobox and different attributes</p>
    <form action="" method="get">
      <input type="radio" name="subject" value="maths" /> Maths
      <input type="radio" name="subject" value="physics" /> Physics
      <input type="reset" value="reset" />
    </form>

  </body>
</html>
```

## HTML Forms - Select box Control:

Drop Down Box is used when we have many options available to be selected but only one or two will be selected..

Here is example HTML code for a form with one drop down box

```
<form action="/cgi-bin/dropdown.cgi" method="post">
  <select name="dropdown">
    <option value="Maths" selected>Maths</option>
    <option value="Physics">Physics</option>
  </select>
  <input type="submit" value="Submit" />
</form>
```

The result of this code is the following form

Following is the list of important attributes of <select>:

- **name:** This is the name for the control.
- **size:** This can be used to present a scrolling list box.
- **multiple:** If set to "multiple" then allows a user to select multiple items from the menu.

Following is the list of important attributes of <option>:

- **value:** The value that is sent to the server if this option is selected.
- **selected:** Specifies that this option should be the initially selected value when the page loads.
- **label:** An alternative way of labeling options.

## HTML Forms - File Select Boxes:

If you want to allow a user to upload a file to your web site from his computer, you will need to use a file upload box, also known as a file select box. This is also created using the <input> element.

Here is example HTML code for a form with one file select box

```
<form action="/cgi-bin/hello_get.cgi" method="post"
      name="fileupload" enctype="multipart/form-data">
    <input type="file" name="fileupload" accept="image/*" />
</form>
```

The result of this code is the following form

## HTML Forms - Hidden Controls:

If you will want to pass information between pages without the user seeing it. Hidden form controls remain part of any form, but the user cannot see them in the Web browser. They should not be used for any sensitive information you do not want the user to see because the user could see this data if she looked in the source of the page.

Following hidden form is being used to keep current page number. When a user will click next page then the value of hidden form will be sent to the back-end application and it will decide which page has be displayed next.

```
<form action="/cgi-bin/hello_get.cgi" method="get" name="pages">
  <p>This is page 10</p>
  <input type="hidden" name="pgenumber" value="10" />
  <input type="submit" value="Next Page" />
</form>
```

This will produce following result:

This is page 10

## HTML Forms - Submit and Reset Button:

These are special buttons which can be created using <input>. When submit button is clicked then Forms data is submitted to the back-end application. When reset button is clicked then all the forms control are reset to default state.

You already have seen submit button above, we will give one reset example here:

```
<form action="/cgi-bin/hello_get.cgi" method="get">
    First name: <input type="text" name="first_name" />
    <br>
    Last name: <input type="text" name="last_name" />
    <input type="submit" value="Submit" />
    <input type="reset" value="Reset" />
</form>
```

This will produce following result. Type something and click reset button.

First name:

Last name:

---

## Table

Tables are very useful to arrange in HTML and they are used very frequently by almost all web developers. Tables are just like spreadsheets and they are made up of rows and columns.

You will create a table in HTML/XHTML by using <table> tag. Inside <table> element the table is written out row by row. A row is contained inside a <tr> tag . which stands for table row. And each cell is then written inside the row element using a <td> tag . which stands for table data.

### Example:

```
<table border="1">
<tr>
<td>Row 1, Column 1</td>
<td>Row 1, Column 2</td>
</tr>
<tr>
<td>Row 2, Column 1</td>
<td>Row 2, Column 2</td>
</tr>
</table>
```

This will produce following result:

Row 1, Column 1	Row 1, Column 2
Row 2, Column 1	Row 2, Column 2

**NOTE:** In the above example *border* is an attribute of `<table>` and it will put border across all the cells. If you do not need a border then you can use `border="0"`. The border attribute and other attributes also mentioned in this session are deprecated and they have been replaced by CSS. So it is recommended to use CSS instead of using any attribute directly.

### Table Heading - The `<th>` Element:

Table heading can be defined using `<th>` element. This tag will be put to replace `<td>` tag which is used to represent actual data. Normally you will put your top row as table heading as shown below, otherwise you can use `<th>` element at any place:

```
<table border="1">
<tr>
<th>Name</th>
<th>Salary</th>
</tr>
<tr>
<td>Ramesh Raman</td>
<td>5000</td>
</tr>
<tr>
<td>Shabbir Hussein</td>
<td>7000</td>
</tr>
</table>
```

This will produce following result. You can see its making heading as a bold one:

Name	Salary
Ramesh Raman	5000
Shabbir Hussein	7000

**NOTE:** Each cell must, however, have either a `<td>` or a `<th>` element in order for the table to display correctly even if that element is empty.

## Table Cellpadding and Cellspacing:

There are two attributes called *cellpadding* and *cellspacing* which you will use to adjust the white space in your table cell. Cellspacing defines the width of the border, while cellpadding represents the distance between cell borders and the content within. Following is the example:

```
<table border="1" cellpadding="5" cellspacing="5">
<tr>
<th>Name</th>
<th>Salary</th>
</tr>
<tr>
<td>Ramesh Raman</td>
<td>5000</td>
</tr>
<tr>
<td>Shabbir Hussein</td>
<td>7000</td>
</tr>
</table>
```

This will produce following result:

Name	Salary
Ramesh Raman	5000
Shabbir Hussein	7000

## Colspan and Rowspan Attributes:

You will use *colspan* attribute if you want to merge two or more columns into a single column. Similar way you will use *rowspan* if you want to merge two or more rows. Following is the example:

```
<table border="1">
<tr>
<th>Column 1</th>
<th>Column 2</th>
<th>Column 3</th>
</tr>
<tr><td rowspan="2">Row 1 Cell 1</td>
<td>Row 1 Cell 2</td><td>Row 1 Cell 3</td></tr>
<tr><td>Row 2 Cell 2</td><td>Row 2 Cell 3</td></tr>
<tr><td colspan="3">Row 3 Cell 1</td></tr>
</table>
```

This will produce following result:

Column 1	Column 2	Column 3
Row 1 Cell 1	Row 1 Cell 2	Row 1 Cell 3
Row 3 Cell 1		

## Tables Backgrounds

You can set table background using of the following two ways:

- Using *bgcolor* attribute - You can set background color for whole table or just for one cell.
- Using *background* attribute - You can set background image for whole table or just for one cell.

**NOTE:** You can set border color also using *bordercolor* attribute.

Here is an example of using *bgcolor* attribute:

```
<table border="5" bordercolor="green" bgcolor="gray">
<tr>
<th>Column 1</th>
<th>Column 2</th>
<th>Column 3</th>
</tr>
<tr><td rowspan="2">Row 1 Cell 1</td>
<td bgcolor="red">Row 1 Cell 2</td><td>Row 1 Cell 3</td></tr>
<tr><td>Row 2 Cell 2</td><td>Row 2 Cell 3</td></tr>
<tr><td colspan="3">Row 3 Cell 1</td></tr>
</table>
```

This will produce following result:

Column 1	Column 2	Column 3
Row 1 Cell 1	Row 1 Cell 2	Row 1 Cell 3
Row 3 Cell 1		

Here is an example of using *background* attribute:

```
<table border="1" background="/images/home.gif">
<tr>
<th>Column 1</th>
<th>Column 2</th>
<th>Column 3</th>
</tr>
<tr><td rowspan="2">Row 1 Cell 1</td>
<td bgcolor="red">Row 1 Cell 2</td><td>Row 1 Cell 3</td></tr>
<tr><td>Row 2 Cell 2</td><td>Row 2 Cell 3</td></tr>
<tr><td colspan="3" background="/images/pattern1.gif">
Row 3 Cell 1
</td></tr>
</table>
```

This will produce following result:

Column 1	Column 2	Column 3
Row 1 Cell 1		Row 1 Cell 3
	Row 2 Cell 2	Row 2 Cell 3
Row 3 Cell 1		

### Table height and width:

You can set a table width and height using *width* and *height* attributes. You can specify table width or height in terms of integer value or in terms of percentage of available screen area. Following is the example:

```
<table border="1" width="400" height="150">
<tr>
<td>Row 1, Column 1</td>
<td>Row 1, Column 2</td>
</tr>
<tr>
<td>Row 2, Column 1</td>
<td>Row 2, Column 2</td>
</tr>
</table>
```

This will produce following result:

Row 1, Column 1	Row 1, Column 2
-----------------	-----------------

Row 2, Column 1	Row 2, Column 2
-----------------	-----------------

## Using Table Caption:

The *caption* tags will serve as a title or explanation and show up at the top of the table. This tag is deprecated in newer version of HTML/XHTML.

```
<table border="1">
<caption>This is the caption</caption>
<tr>
<td>row 1, column 1</td><td>row 1, column 2</td>
</tr>
</table>
```

This will produce following result:

This is the caption	
row 1, column 1	row 1, column 2

## Using a Header, Body, and Footer:

Tables can be divided into three portions: a header, a body, and a foot. The head and foot are rather similar to headers and footers in a word-processed document that remain the same for every page, while the body is the main content of the table.

The three elements for separating the head, body, and foot of a table are:

- **<thead>** - to create a separate table header.
- **<tbody>** - to indicate the main body of the table.
- **<tfoot>** - to create a separate table footer.

A table may contain several *<tbody>* elements to indicate different *pages* or groups of data. But it is notable that *<thead>* and *<tfoot>* tags should appear before *<tbody>*

```
<table border="1" width="100%">
<thead>
<tr>
<td colspan="4">This is the head of the table</td>
</tr>
</thead>
<tfoot>
<tr>
```

```
<td colspan="4">This is the foot of the table</td>
</tr>
</tfoot>
<tbody>
<tr>
<td>Cell 1</td>
<td>Cell 2</td>
<td>Cell 3</td>
<td>Cell 4</td>
</tr>
<tr>
...more rows here containing four cells...
</tr>
</tbody>
<tbody>
<tr>
<td>Cell 1</td>
<td>Cell 2</td>
<td>Cell 3</td>
<td>Cell 4</td>
</tr>
<tr>
...more rows here containing four cells...
</tr>
</tbody>
</table>
```

This will produce following result:

This is the head of the table			
This is the foot of the table			
Cell 1	Cell 2	Cell 3	Cell 4
...more rows here containing four cells...			
Cell 1	Cell 2	Cell 3	Cell 4
...more rows here containing four cells...			

## Nested Tables:

You can use one table inside another table. Not only tables you can use almost all the tags inside table data tag `<td>`.

Following is the example of using another table and other tags inside a table cell.

```

<table border="1">
<tr>
<td>
    <table border="1">
    <tr>
        <th>Name</th>
        <th>Salary</th>
    </tr>
    <tr>
        <td>Ramesh Raman</td>
        <td>5000</td>
    </tr>
    <tr>
        <td>Shabbir Hussein</td>
        <td>7000</td>
    </tr>
    </table>
</td>
<td>
    <ul>
        <li>This is another cell</li>
        <li>Using list inside this cell</li>
    </ul>
</td>
</tr>
<tr>
<td>Row 2, Column 1</td>
<td>Row 2, Column 2</td>
</tr>
</table>

```

This will produce following result:

Name	Salary	
Ramesh Raman	5000	<ul style="list-style-type: none"> <li>This is another cell</li> <li>Using list inside this cell</li> </ul>
Shabbir Hussein	7000	
Row 2, Column 1	Row 2, Column 2	

## HTML Frames

With frames, you can display more than one HTML document in the same browser window. Each HTML document is called a frame, and each frame is independent of the others.

The disadvantages of using frames are:

- Frames are not expected to be supported in future versions of HTML
- Frames are difficult to use. (Printing the entire page is difficult).

- The web developer must keep track of more HTML documents

## The HTML frameset Element

The frameset element holds one or more frame elements. Each frame element can hold a separate document.

The frameset element states HOW MANY columns or rows there will be in the frameset, and HOW MUCH percentage/pixels of space will occupy each of them.

## The HTML frame Element

The <frame> tag defines one particular window (frame) within a frameset.

In the example below we have a frameset with two columns.

The first column is set to 25% of the width of the browser window. The second column is set to 75% of the width of the browser window. The document "frame\_a.htm" is put into the first column, and the document "frame\_b.htm" is put into the second column:

```
<frameset cols="25%,75%">
  <frame src="frame_a.htm" />
  <frame src="frame_b.htm" />
</frameset>
```

**Note:** The frameset column size can also be set in pixels (cols="200,500"), and one of the columns can be set to use the remaining space, with an asterisk (cols="25%,\*").

## Basic Notes - Useful Tips

**Tip:** If a frame has visible borders, the user can resize it by dragging the border. To prevent a user from doing this, you can add noresize="noresize" to the <frame> tag.

**Note:** Add the <noframes> tag for browsers that do not support frames.

**Important:** You cannot use the <body></body> tags together with the <frameset></frameset> tags! However, if you add a <noframes> tag containing some text for browsers that do not support frames, you will have to enclose the text in <body></body> tags! See how it is done in the first example below.

**How to use the <noframes> tag (for browsers that do not support frames):**

```
<html>
```

```
<frameset cols="25%,50%,25%">
  <frame src="frame_a.htm" />
  <frame src="frame_b.htm" />
  <frame src="frame_c.htm" />
```

```
<noframes>
```

```
<body>Your browser does not handle frames!</body>
</noframes>

</frameset>

</html>
```

## **How to create a frameset with three documents, and how to mix them in rows and columns.**

```
<html>

<frameset rows="50%,50%">

<frame src="frame_a.htm" />
<frameset cols="25%,75%">
<frame src="frame_b.htm" />
<frame src="frame_c.htm" />
</frameset>

</frameset>
```

```
</html>
```

## **Frameset with noresize="noresize"**

```
<html>

<frameset rows="50%,50%">

<frame noresize="noresize" src="frame_a.htm" />
<frame noresize="noresize" src="frame_b.htm" />

</frameset>
```

```
</html>
```

## **Navigation**

```
<html>

<frameset cols="120,*">

<frame src="tryhtml_contents.htm" />
<frame src="frame_a.htm" name="showframe" />

</frameset>

</html>
```

## Website Layouts

Most websites have put their content in multiple columns (formatted like a magazine or newspaper).

Multiple columns are created by using <table> or <div> tags. Some CSS are normally also added to position elements, or to create backgrounds or colorful look for the pages.

### HTML Layouts - Using Tables

The simplest way of creating layouts is by using the HTML <table> tag.

The following example uses a table with 3 rows and 2 columns - the first and last row spans both columns using the colspan attribute:

#### Example

```
<html>
<body>

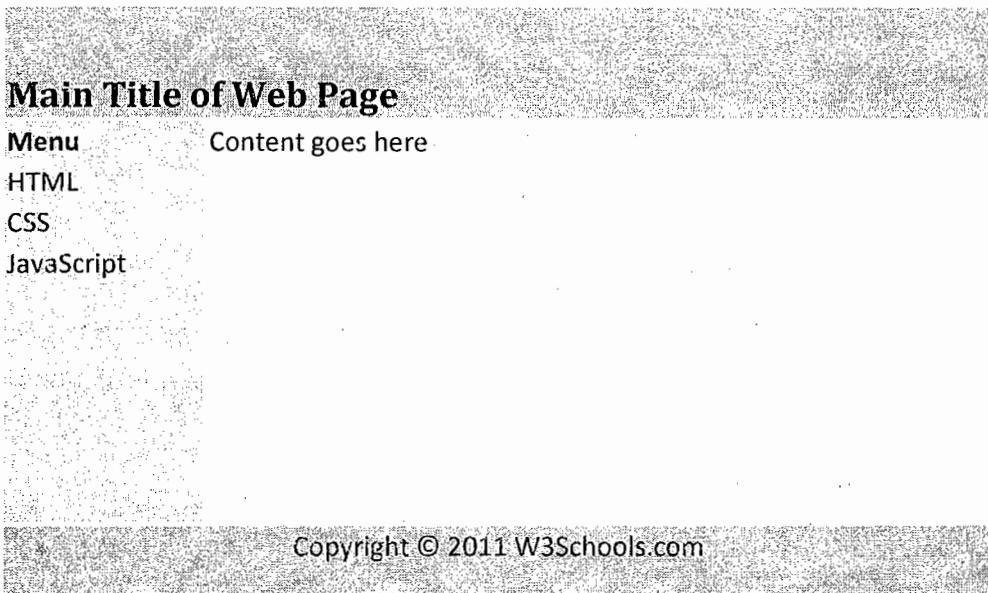
<table width="500" border="0">
<tr>
<td colspan="2" style="background-color:#FFA500;">
<h1>Main Title of Web Page</h1>
</td>
</tr>

<tr valign="top">
<td style="background-color:#FFD700;width:100px;text-align:top;">
<b>Menu</b><br />
HTML<br />
CSS<br />
JavaScript
</td>
<td style="background-color:#EEEEEE;height:200px;width:400px;text-align:top;">
Content goes here</td>
</tr>

<tr>
<td colspan="2" style="background-color:#FFA500;text-align:center;">
Copyright © 2011 W3Schools.com</td>
</tr>
</table>

</body>
</html>
```

The HTML code above will produce the following result:



**Note:** Even though it is possible to create nice layouts with HTML tables, tables were designed for presenting tabular data - NOT as a layout tool!

## HTML Layouts - Using Div Elements

The div element is a block level element used for grouping HTML elements.

The following example uses five div elements to create a multiple column layout, creating the same result as in the previous example:

### Example

```
<html>
<body>

<div id="container" style="width:500px">

<div id="header" style="background-color:#FFA500;">
<h1 style="margin-bottom:0;">Main Title of Web Page</h1></div>

<div id="menu" style="background-color:#FFD700;height:200px;width:100px;float:left;">
<b>Menu</b><br />
HTML<br />
CSS<br />
JavaScript</div>

<div id="content" style="background-color:#EEEEEE;height:200px;width:400px;float:left;">
Content goes here</div>
```

```
<div id="footer" style="background-color:#FFA500;clear:both;text-align:center;">
Copyright © 2011 W3Schools.com</div>

</div>

</body>
</html>
```

The HTML code above will produce the following result:

## Main Title of Web Page

Menu

HTML

CSS

JavaScript

Content goes here

Copyright © 2011 W3Schools.com

## HTML Layout - Useful Tips

**Tip:** The biggest advantage of using CSS is that, if you place the CSS code in an external style sheet, your site becomes MUCH EASIER to maintain. You can change the layout of all your pages by editing one file.

**Tip:** Because advanced layouts take time to create, a quicker option is to use a template. Search Google for free website templates (these are pre-built website layouts you can use and customize)

## What is CSS?

- CSS stands for Cascading Style Sheets
- Styles define **how to display** HTML elements
- Styles were added to HTML 4.0 to **solve a problem**
- External Style Sheets can save a lot of work
- External Style Sheets are stored in **CSS files**

## Styles Solved a Big Problem

HTML was never intended to contain tags for formatting a document.

HTML was intended to define the content of a document, like:

```
<h1>This is a heading</h1>
```

```
<p>This is a paragraph.</p>
```

When tags like `<font>`, and color attributes were added to the HTML 3.2 specification, it started a nightmare for web developers. Development of large web sites, where fonts and color information were added to every single page, became a long and expensive process.

To solve this problem, the World Wide Web Consortium (W3C) created CSS.

In HTML 4.0, all formatting could be removed from the HTML document, and stored in a separate CSS file.

All browsers support CSS today.

## CSS Saves a Lot of Work!

CSS defines HOW HTML elements are to be displayed.

Styles are normally saved in external .css files. External style sheets enable you to change the appearance and layout of all the pages in a Web site, just by editing one single file!

## CSS Syntax

A CSS rule has two main parts: a selector, and one or more declarations:

Selector	Declaration	Declaration	
<code>h1</code>	<code>{color:blue; font-size:12px;}</code>		
Property	Value	Property	Value

The selector is normally the HTML element you want to style.

Each declaration consists of a property and a value.

The property is the style attribute you want to change. Each property has a value.

A CSS comprises of style rules that are interpreted by the browser and then applied to the corresponding elements in your document. A style rule is made of three parts:

- **Selector:** A selector is an HTML tag at which style will be applied. This could be any tag like <h1> or <table> etc.
- **Property:** A property is a type of attribute of HTML tag. Put simply, all the HTML attributes are converted into CSS properties. They could be *color* or *border* etc.
- **Value:** Values are assigned to properties. For example *color* property can have value either *red* or *#F1F1F1* etc.

You can put CSS Style Rule Syntax as follows:

```
selector { property: value }
```

**Example:** You can define a table border as follows:

```
table{ border :1px solid #C00; }
```

Here table is a selector and border is a property and given value *1px solid #C00* is the value of that property.

You can define selectors in various simple ways based on your comfort. Let me put these selectors one by one.

### **The Type Selectors:**

This is the same selector we have seen above. Again one more example to give a color to all level 1 headings :

```
h1 {  
    color: #36CFFF;  
}
```

### **The Universal Selectors:**

Rather than selecting elements of a specific type, the universal selector quite simply matches the name of any element type :

```
* {  
    color: #000000;  
}
```

This rule renders the content of every element in our document in black.

## The Descendant Selectors:

Suppose you want to apply a style rule to a particular element only when it lies inside a particular element. As given in the following example, style rule will apply to `<em>` element only when it lies inside `<ul>` tag.

```
ul em {  
    color: #000000;  
}
```

## The Class Selectors:

You can define style rules based on the class attribute of the elements. All the elements having that class will be formatted according to the defined rule.

```
.black {  
    color: #000000;  
}
```

This rule renders the content in black for every element with class attribute set to `black` in our document. You can make it a bit more particular. For example:

```
h1.black {  
    color: #000000;  
}
```

This rule renders the content in black for only `<h1>` elements with class attribute set to `black`.

You can apply more than one class selectors to given element. Consider the following example :

```
<p class="center bold">  
This para will be styled by the classes center and bold.  
</p>
```

## The ID Selectors:

You can define style rules based on the id attribute of the elements. All the elements having that id will be formatted according to the defined rule.

```
#black {  
    color: #000000;  
}
```

This rule renders the content in black for every element with id attribute set to `black` in our document. You can make it a bit more particular. For example:

```
h1#black {  
    color: #000000;  
}
```

This rule renders the content in black for only `<h1>` elements with id attribute set to *black*.

The true power of id selectors is when they are used as the foundation for descendant selectors, For example:

```
#black h2 {  
    color: #000000;  
}
```

In this example all level 2 headings will be displayed in black color only when those headings will lie with in tags having id attribute set to *black*.

### The Child Selectors:

You have seen descendant selectors. There is one more type of selectors which is very similar to descendants but have different functionality. Consider the following example:

```
body > p {  
    color: #000000;  
}
```

This rule will render all the paragraphs in black if they are **direct child** of `<body>` element. Other paragraphs put inside other elements like `<div>` or `<td>` etc. would not have any effect of this rule.

### The Attribute Selectors:

You can also apply styles to HTML elements with particular attributes. The style rule below will match all input elements that has a type attribute with a value of *text*:

```
input[type="text"] {  
    color: #000000;  
}
```

The advantage to this method is that the `<input type="submit" />` element is unaffected, and the color applied only to the desired text fields.

There are following rules applied to attribute selector.

- **p[lang]** - Selects all paragraph elements with a *lang* attribute.
- **p[lang="fr"]** - Selects all paragraph elements whose *lang* attribute has a value of exactly "fr".
- **p[lang~="fr"]** - Selects all paragraph elements whose *lang* attribute contains the word "fr".
- **p[lang|= "en"]** - Selects all paragraph elements whose *lang* attribute contains values that are exactly "en", or begin with "en-".

## Multiple Style Rules:

You may need to define multiple style rules for a single element. You can define these rules to combine multiple properties and corresponding values into a single block as defined in the following example:

```
h1 {  
color: #36C;  
font-weight: normal;  
letter-spacing: .4em;  
margin-bottom: 1em;  
text-transform: lowercase;  
}
```

Here all the property and value pairs are separated by a **semi colon (;**). You can keep them in a single line or multiple lines. For better readability we keep them into separate lines.

## Grouping Selectors:

You can apply a style to many selectors if you like. Just separate the selectors with a comma as given in the following example:

```
h1, h2, h3 {  
color: #36C;  
font-weight: normal;  
letter-spacing: .4em;  
margin-bottom: 1em;  
text-transform: lowercase;  
}
```

This define style rule will be applicable to h1, h2 and h3 element as well. The order of the list is irrelevant. All the elements in the selector will have the corresponding declarations applied to them.

You can combine various *class* selectors together as shown below:

```
#content, #footer, #supplement {  
position: absolute;  
left: 510px;  
width: 200px;  
}
```

There are four ways to associate styles with your HTML document. Most commonly used methods are inline CSS and External CSS.

## Embedded CSS - The <style> Element:

You can put your CSS rules into an HTML document using the <style> element. This tag is placed inside <head>...</head> tags. Rules defined using this syntax will be applied to all the elements available in the document. Here is the generic syntax:

```
<head>
<style type="text/css" media="...">
Style Rules
.....
</style>
</head>
```

### Attributes:

Attributes associated with <style> elements are:

Attribute	Value	Description
type	text/css	Specifies the style sheet language as a content-type (MIME type). This is required attribute.
media	screen tty tv projection handheld print braille aural all	Specifies the device the document will be displayed on. Default value is <i>all</i> . This is optional attribute.

### Example:

Following is the example of embed CSS based on above syntax:

```
<head>
<style type="text/css" media="all">
h1{
color: #36C;
}
</style>
</head>
```

## Inline CSS - The *style* Attribute:

You can use *style* attribute of any HTML element to define style rules. These rules will be applied to that element only. Here is the generic syntax:-

```
<element style="...style rules....">
```

### Attributes:

Attribute	Value	Description
style	style rules	The value of <i>style</i> attribute is a combination of style declarations separated by semicolon (;).

### Example:

Following is the example of inline CSS based on above syntax:

```
<h1 style ="color:#36C;"> This is inline CSS </h1>
```

## External CSS - The *<link>* Element:

The *<link>* element can be used to include an external stylesheet file in your HTML document.

An external style sheet is a separate text file with **.css** extension. You define all the Style rules within this text file and then you can include this file in any HTML document using *<link>* element.

Here is the generic syntax of including external CSS file:

```
<head>
<link type="text/css" href="..." media="..." href="mystyle.css" />
</head>
```

### Attributes:

Attributes associated with *<style>* elements are:

Attribute	Value	Description
type	text/css	Specifies the style sheet language as a content-type (MIME type). This attribute is required.
href	URL	Specifies the style sheet file having Style rules. This attribute is a required.

<b>media</b> screen tty tv projection handheld print braille aural all	Specifies the device the document will be displayed on. Default value is <i>all</i> . This is optional attribute.
---	---

### Example:

Consider a simple style sheet file with a name *mystyle.css* having the following rules:

```

h1, h2, h3 {
color: #36C;
font-weight: normal;
letter-spacing: .4em;
margin-bottom: 1em;
text-transform: lowercase;
}
  
```

Now you can include this file *mystyle.css* in any HTML document as follows:

```

<head>
<link type="text/css" href="mystyle.css" rel="stylesheet" media="all" />
</head>
  
```

### Imported CSS - @import Rule:

@import is used to import an external stylesheet in a manner similar to the <link> element. Here is the generic syntax of @import rule.

```

<head>
<@import "URL";
</head>
  
```

Here URL is the URL of the style sheet file having style rules. You can use another syntax as well:

```

<head>
<@import url("URL");
</head>
  
```

**Example:**

Following is the example showing you how to import a style sheet file into HTML document:

```
<head>
@import "mystyle.css";
</head>
```

**CSS Rules Overriding:**

We have discussed four ways to include style sheet rules in a an HTML document. Here is the rule to override any Style Sheet Rule.

- Any inline style sheet takes highest priority. So it will override any rule defined in `<style>...</style>` tags or rules defined in any external style sheet file.
- Any rule defined in `<style>...</style>` tags will override rules defined in any external style sheet file.
- Any rule defined in external style sheet file takes lowest priority and rules defined in this file will be applied only when above two rules are not applicable.

**Handling old Browsers:**

There are still many old browsers who do not support CSS. So we should take care while writing our Embedded CSS in an HTML document. The following snippet shows how you can use comment tags to hide CSS from older browsers:

```
<style type="text/css">
<!--
body, td {
    color: blue;
}
-->
</style>
```

**CSS Comments:**

Many times you may need to put additional comments in your style sheet blocks. So it is very easy to comment any part in style sheet. You simple put your comments inside `/*.....this is a comment in style sheet.....*/`.

You can use `/* .... */` to comment multi-line blocks in similar way you do in C and C++ programming languages.

**Example:**

```
/* This is an external style sheet file */
h1, h2, h3 {
color: #36C;
font-weight: normal;
```

```

letter-spacing: .4em;
margin-bottom: 1em;
text-transform: lowercase;
}
/* end of style rules. */

```

## CSS Units

Before we start actual exercise, I would like to give a brief idea about the CSS Measurement Units.

CSS supports a number of measurements including absolute units such as inches, centimeters, points, and so on, as well as relative measures such as percentages and em units. You need these values while specifying various measurements in your Style rules e.g **border="1px solid red"**.

We have listed out all the CSS Measurement Units alongwith proper Examples:

Unit	Description	Example
%	Defines a measurement as a percentage relative to another value, typically an enclosing element.	p {font-size: 16pt; line-height: 125%;}
Cm	Defines a measurement in centimeters.	div {margin-bottom: 2cm;}
Em	A relative measurement for the height of a font in em spaces. Because an em unit is equivalent to the size of a given font, if you assign a font to 12pt, each "em" unit would be 12pt; thus, 2em would be 24pt.	p {letter-spacing: 7em;}
Ex	This value defines a measurement relative to a font's x-height. The x-height is determined by the height of the font's lowercase letter x.	p {font-size: 24pt; line-height: 3ex;}
In	Defines a measurement in inches.	p {word-spacing: .15in;}
mm	Defines a measurement in millimeters.	p {word-spacing: 15mm;}
pc	Defines a measurement in picas. A pica is equivalent to 12 points; thus, there are 6 picas per inch.	p {font-size: 20pc;}
pt	Defines a measurement in points. A point is defined as 1/72nd of an inch.	body {font-size: 18pt;}
px	Defines a measurement in screen pixels.	p {padding: 25px;}

## CSS - Colors

CSS uses color values to specify a color. Typically, these are used to set a color either for the foreground of an element(i.e., its text) or else for the background of the element. They can also be used to affect the color of borders and other decorative effects.

You can specify your color values in various formats. Following table tells you all possible formats:

Format	Syntax	Example
Hex Code	#RRGGBB	p{color:#FF0000;}
Short Hex Code	#RGB	p{color:#6A7;}
RGB %	rgb(rrr%,ggg%,bbb%)	p{color:rgb(50%,50%,50%);}
RGB Absolute	rgb(rrr,ggg,bbb)	p{color:rgb(0,0,255);}
keyword	aqua, black, etc.	p{color:teal;}

## Setting Backgrounds using CSS

You can set following background properties of an element:

- The **background-color** property is used to set the background color of an element.
- The **background-image** property is used to set the background image of an element.
- The **background-repeat** property is used to control the repetition of an image in the background.
- The **background-position** property is used to control the position of an image in the background.
- The **background-attachment** property is used to control the scrolling of an image in the background.
- The **background** property is used as shorthand to specify a number of other background properties.

### Set the background color:

Following is the example which demonstrates how to set the background color for an element.

```
<p style="background-color:yellow;">
    This text has a yellow background color.
</p>
```

### Set the background image:

Following is the example which demonstrates how to set the background image for an element.

```
<table style="background-image:url(/images/pattern1.gif);">
<tr>
    <td>
        This table has background image set.
    </td>
</tr>
</table>
```

## Repeat the background image:

Following is the example which demonstrates how to repeat the background image if image is small. You can use *no-repeat* value for *background-repeat* property if you don't want to repeat an image, in this case image will display only once.

By default *background-repeat* property will have *repeat* value.

```
<table style="background-image:url(/images/pattern1.gif);  
background-repeat: repeat;">  
<tr>  
  <td>  
    This table has background image which repeats multiple times.  
  </td>  
</tr>  
</table>
```

Following is the example which demonstrates how to repeat the background image vertically.

```
<table style="background-image:url(/images/pattern1.gif);  
background-repeat: repeat-y;">  
<tr>  
  <td>  
    This table has background image set which will repeat vertically.  
  </td>  
</tr>  
</table>
```

Following is the example which demonstrates how to repeat the background image horizontally.

```
<table style="background-image:url(/images/pattern1.gif);  
background-repeat: repeat-x;">  
<tr><td>  
This table has background image set which will repeat horizontally.  
</td></tr>  
</table>
```

Following is the example which demonstrates how to **not repeat** the background image.

```
<table style="background-image:url(/images/pattern1.gif); background-repeat: no-repeat;">  
<tr>  
  <td>  
    This table has background image set which will repeat horizontally.  
  </td>  
</tr>  
</table>
```

## Set the background image position:

Following is the example which demonstrates how to set the background image position 100 pixels away from the left side.

```
<table style="background-image:url(/images/pattern1.gif);  
background-position:100px;">  
<tr>  
  <td>  
    Background image positioned 100 pixels away from the left.  
  </td>  
</tr>  
</table>
```

Following is the example which demonstrates how to set the background image position 100 pixels away from the left side and 200 pixels down from the top.

```
<table style="background-image:url(/images/pattern1.gif);  
background-position:100px 200px;">  
<tr>  
  <td>  
    This table has background image positioned 100  
    pixels away from the left and 200 pixels from the top.  
  </td>  
</tr>  
</table>
```

## Set the background attachment:

Background attachment determines whether a background image is fixed or scrolls with the rest of the page.

Following is the example which demonstrates how to set the fixed background image.

```
<p style="background-image:url(/images/pattern1.gif);  
background-attachment:fixed;">  
  This paragraph has fixed background image.  
</p>
```

Following is the example which demonstrates how to set the scrolling background image.

```
<p style="background-image:url(/images/pattern1.gif);  
background-attachment:scroll;">  
  This paragraph has scrolling background image.  
</p>
```

## Background - Shorthand property

As you can see from the examples above, there are many properties to consider when dealing with backgrounds.

To shorten the code, it is also possible to specify all the properties in one single property. This is called a shorthand property.

The shorthand property for background is simply "background":

```
background: color position size repeat origin clip attachment image;
```

### Example

```
body {  
    background:#ffffff url('img_tree.png') no-repeat right top;  
}
```

When using the shorthand property the order of the property values are:

- background-color
- background-image
- background-repeat
- background-attachment
- background-position

```
<html>  
<head>  
<style type="text/css">  
body  
{  
margin-left:200px;  
background:#5d9ab2 url('images/tree.png') no-repeat top left;  
}  
  
.container  
{  
text-align:center;  
}  
  
.center_div  
{  
border:1px solid gray;  
margin-left:auto;  
margin-right:auto;  
width:90%;  
background-color:#d0f0f6;  
text-align:left;  
padding:8px;  
}  
</style>  
</head>
```

```

<body>
<div class="container">
<div class="center_div">
<h1>Hello World!</h1>
<p>This example contains some advanced CSS methods you may
not have learned yet. But, we will explain these methods in a
later chapter in the tutorial.</p>
</div>
</div>
</body>
</html>

```

It does not matter if one of the property values is missing, as long as the ones that are present are in this order.

## Setting Fonts using CSS

You can set following font properties of an element:

- The **font-family** property is used to change the face of a font.
- The **font-style** property is used to make a font italic or oblique.
- The **font-variant** property is used to create a small-caps effect.
- The **font-weight** property is used to increase or decrease how bold or light a font appears.
- The **font-size** property is used to increase or decrease the size of a font.
- The **font** property is used as shorthand to specify a number of other font properties.

## CSS Font Families

Generic family	Font family	Description
Serif	Times New Roman Georgia	Serif fonts have small lines at the ends on some characters
Sans-serif	Arial Verdana	"Sans" means without - these fonts do not have the lines at the ends of characters
Monospace	Courier New	All monospace characters have the same width

## Lucida Console

### Difference Between Serif and Sans-serif Fonts



On computer screens, sans-serif fonts are considered easier to read than serif fonts.

### Font Family

The font family of a text is set with the **font-family** property.

The **font-family** property should hold several font names as a "fallback" system. If the browser does not support the first font, it tries the next font.

Start with the font you want, and end with a generic family, to let the browser pick a similar font in the generic family, if no other fonts are available.

**Note:** If the name of a font family is more than one word, it must be in quotation marks, like **font-family: "Times New Roman"**.

More than one font family is specified in a comma-separated list:

### Example

```
pt{  
    font-family:"Times New Roman", Times, serif;  
}
```

Following is the example which demonstrates how to set the font family of an element. Possible value could be any font family name.

```
<p style="font-family:georgia,garamond,serif;">  
    This text is rendered in either georgia, garamond, or the default  
    serif font depending on which font you have at your system.  
</p>
```

## Set the font style:

Following is the example which demonstrates how to set the font style of an element. Possible values are *normal, italic and oblique*.

```
<p style="font-style:italic;">  
    This text will be rendered in italic style  
</p>
```

## Set the font variant:

Following is the example which demonstrates how to set the font variant of an element. Possible values are *normal and small-caps*.

```
<html>  
<head>  
<style type="text/css">  
    p.normal {font-variant:normal;}  
    p.small {font-variant:small-caps;}  
</style>  
</head>  
  
<body>  
    <p class="normal">My name is Hege Refsnes.</p>  
    <p class="small">My name is Hege Refsnes.</p>  
</body>  
  
</html>
```

## Set the font weight:

Following is the example which demonstrates how to set the font weight of an element. The font-weight property provides the functionality to specify how bold a font is. Possible values could be *normal, bold, bolder, lighter, 100, 200, 300, 400, 500, 600, 700, 800, 900*.

```
<p style="font-weight:bold;">  
    This font is bold.  
</p>  
<p style="font-weight:bolder;">  
    This font is bolder.  
</p>  
<p style="font-weight:900;">  
    This font is 900 weight.  
</p>
```

## Set the font size:

Following is the example which demonstrates how to set the font size of an element. The font-size property is used to control the size of fonts. Possible values could be *xx-small, x-small, small, medium, large, x-large, xx-large, smaller, larger, size in pixels or in %*

```
<p style="font-size:20px;">  
    This font size is 20 pixels  
</p>  
<p style="font-size:small;">  
    This font size is small  
</p>  
<p style="font-size:large;">  
    This font size is large  
</p>
```

## Shorthand property :

You can use the *font* property to set all the font properties at once. For example:

```
<p style="font:italic small-caps bold 15px georgia;">  
    Applying all the properties on the text at once.  
</p>
```

## Manipulating Text using CSS

You can set following text properties of an element:

- The **color** property is used to set the color of a text.
- The **direction** property is used to set the text direction.
- The **letter-spacing** property is used to add or subtract space between the letters that make up a word.
- The **word-spacing** property is used to add or subtract space between the words of a sentence.
- The **text-indent** property is used to indent the text of a paragraph.
- The **text-align** property is used to align the text of a document.
- The **text-decoration** property is used to underline, overline, and strikethrough text.
- The **text-transform** property is used to capitalize text or convert text to uppercase or lowercase letters.
- The **white-space** property is used to control the flow and formatting of text.
- The **text-shadow** property is used to set the text shadow around a text.

## Set the text color:

Following is the example which demonstrates how to set the text color. Possible value could be any color name in any valid format.

```
<p style="color:red;">  
    This text will be written in red.  
</p>
```

## Set the text direction :

Following is the example which demonstrates how to set the direction of a text. Possible values are *ltr or rtl*.

```
<p style="direction:rtl;">  
    This text will be renedered from right to left  
</p>
```

## Set the space between characters:

Following is the example which demonstrates how to set the space between characters. Possible values are *normal or a number specifying space..*

```
<p style="letter-spacing:5px;">  
    This text is having space between letters.  
</p>
```

## Set the space between words:

Following is the example which demonstrates how to set the space between words. Possible values are *normal or a number specifying space..*

```
<p style="word-spacing:5px;">  
    This text is having space between words.  
</p>
```

## Set the text indent:

Following is the example which demonstrates how to indent the first line of a paragraph. Possible values are *% or a number specifying indent space..*

```
<p style="text-indent:1cm;">  
    This text will have first line indented by 1cm  
    and this line will remain at its actual position  
    this is done by CSS text-indent property.  
</p>
```

## Set the text alignment:

Following is the example which demonstrates how to align a text. Possible values are *left, right, center, justify..*

```
<p style="text-align:right;">  
    This will be right aligned.  
</p>  
<p style="text-align:center;">  
    This will be center aligned.  
</p>  
<p style="text-align:left;">  
    This will be left aligned.  
</p>
```

## Decorating the text:

Following is the example which demonstrates how to decorate a text. Possible values are *none, underline, overline, line-through, blink..*

```
<p style="text-decoration:underline;">  
    This will be underlined  
</p>  
<p style="text-decoration:line-through;">  
    This will be striked through.  
</p>  
<p style="text-decoration:overline;">  
    This will have a over line.  
</p>  
<p style="text-decoration:blink;">  
    This text will have blinking effect  
</p>
```

## Set the text cases:

Following is the example which demonstrates how to set the cases for a text. Possible values are *none, capitalize, uppercase, lowercase..*

```
<p style="text-transform:capitalize;">  
    This will be capitalized  
</p>  
<p style="text-transform:uppercase;">  
    This will be in uppercase  
</p>  
<p style="text-transform:lowercase;">  
    This will be in lowercase  
</p>
```

## Set the white space between text:

Following is the example which demonstrates how white space inside an element is handled. Possible values are *normal*, *pre*, *nowrap*.

```
<p style="white-space:pre;">  
    This text has a line break and the white-space pre setting tells the browser to  
    honor it just like the HTML pre tag.  
</p>
```

## Set the text shadow:

Following is the example which demonstrates how to set the shadow around a text. This may not be supported by all the browsers.

```
<p style="text-shadow: 4px 4px 8px blue;font-size:50px;">  
    If your browser supports the CSS text-shadow property,  
    this text will have a blue shadow.  
</p>
```

# Styling Links

Links can be styled with any CSS property (e.g. color, font-family, background, etc.).

Special for links are that they can be styled differently depending on what state they are in.

The four links states are:

- *a:link* - a normal, unvisited link
- *a:visited* - a link the user has visited
- *a:hover* - a link when the user mouses over it
- *a:active* - a link the moment it is clicked

### Example

```
a:link {color:#FF0000;} /* unvisited link */  
a:visited {color:#00FF00;} /* visited link */  
a:hover {color:#FF00FF;} /* mouse over link */  
a:active {color:#0000FF;} /* selected link */
```

### Example

```
a:link {text-decoration:none;}  
a:visited {text-decoration:none;}  
a:hover {text-decoration:underline;}  
a:active {text-decoration:underline;}
```

### Example

```
a:link {background-color:#B2FF99;}  
a:visited {background-color:#FFFF85;}  
a:hover {background-color:#FF704D;}  
a:active {background-color:#FF704D;}
```

When setting the style for several link states, there are some order rules:

- a:hover MUST come after a:link and a:visited
- a:active MUST come after a:hover

### CSS - Lists

Lists are very helpful in conveying a set of either numbered or bulleted points. This teaches you how to control list type, position, style etc. using CSS

There are following five CSS properties which can be used to control lists:

- ↳ The **list-style-type** Allows you to control the shape or appearance of the marker.
- ↳ The **list-style-position** Specifies whether a long point that wraps to a second line should align with the first line or start underneath the start of the marker.
- ↳ The **list-style-image** Specifies an image for the marker rather than a bullet point or number.
- ↳ The **list-style** Serves as shorthand for the preceding properties.
- ↳ The **marker-offset** Specifies the distance between a marker and the text in the list.

Now we will see how to use these properties with examples.

### The **list-style-type** Property:

The *list-style-type* property allows you to control the shape or style of bullet point (also known as a marker) in the case of unordered lists, and the style of numbering characters in ordered lists.

Here are the values which can be used for an unordered list:

Value	Description
None	NA
disc (default)	A filled-in circle
Circle	An empty circle

Square	A filled-in square
--------	--------------------

Here are the values which can be used for an ordered list:

Value	Description	Example
Decimal	Number	1,2,3,4,5
decimal-leading-zero	0 before the number	01, 02, 03, 04, 05
lower-alpha	Lowercase alphanumeric characters	a, b, c, d, e
upper-alpha	Uppercase alphanumeric characters	A, B, C, D, E
lower-roman	Lowercase Roman numerals	i, ii, iii, iv, v
upper-roman	Uppercase Roman numerals	I, II, III, IV, V
lower-greek	The marker is lower-greek	alpha, beta, gamma
lower-latin	The marker is lower-latin	a, b, c, d, e
upper-latin	The marker is upper-latin	A, B, C, D, E
Hiragana	The marker is hiragana	a, i, u, e, o, ka, ki
Katakana	The marker is katakana	A, I, U, E, O, KA, KI
hiragana-iroha	The marker is hiragana-iroha	i, ro, ha, ni, ho, he, to
katakana-iroha	The marker is katakana-iroha	I, RO, HA, NI, HO, HE, TO

Here is the example:

```
<ul style="list-style-type:circle;">
  <li>Maths</li>
```

```
<li>Social Science</li>
<li>Physics</li>
</ul>

<ul style="list-style-type:square;">
<li>Maths</li>
<li>Social Science</li>
<li>Physics</li>
</ul>

<ol style="list-style-type:decimal;">
<li>Maths</li>
<li>Social Science</li>
<li>Physics</li>
</ol>

<ol style="list-style-type:lower-alpha;">
<li>Maths</li>
<li>Social Science</li>
<li>Physics</li>
</ol>

<ol style="list-style-type:lower-roman;">
<li>Maths</li>
<li>Social Science</li>
<li>Physics</li>
</ol>
```

## The **list-style-position** Property:

The *list-style-position* property indicates whether the marker should appear inside or outside of the box containing the bullet points. It can have one of the two values:

Value	Description
None	NA
Inside	If the text goes onto a second line, the text will wrap underneath the marker. It will also appear indented to where the text would have started if the list had a value of outside.
outside	If the text goes onto a second line, the text will be aligned with the start of the first line (to the right of the bullet).

Here is the example:

```
<ul style="list-style-type:circle; list-style-position:outside;">
```

```
<li>Maths</li>
<li> If the text goes onto a second line, the text will be aligned with the start of
    the first line (to the right of the bullet </li>
<li>Physics</li>
</ul>

<ul style="list-style-type:square;list-style-position:inside;">
<li>Maths</li>
<li> If the text goes onto a second line, the text will be aligned with the start of
    the first line (to the right of the bullet </li>
<li>Physics</li>
</ul>

<ol style="list-style-type:decimal;list-style-position:outside;">
<li>Maths</li>
<li>Social Science</li>
<li>Physics</li>
</ol>

<ol style="list-style-type:lower-alpha;list-style-position:inside;">
<li>Maths</li>
<li>Social Science</li>
<li>Physics</li>
</ol>
```

## The list-style-image Property:

The *list-style-image* allows you to specify an image so that you can use your own bullet style. The syntax is as follows, similar to the background-image property with the letters url starting the value of the property followed by the URL in brackets. If it does not find given image then default bullets are used.

Here is the example:

```
<ul style="list-style-image: url(images/mobile.gif); ">
<li >Maths</li>
<li>Social Science</li>
<li>Physics</li>
</ul>

<ol style="list-style-image: url(images/mobile.gif); ">
<li style="list-style-image: url(images/rose.gif); ">Maths</li>
<li>Social Science</li>
<li>Physics</li>
</ol>
```

## The list-style Property:

The *list-style* allows you to specify all the list properties into a single expression. These properties can appear in any order.

Here is the example:

```
<ul style="list-style: inside square;">
<li>Maths</li>
<li>Social Science</li>
<li>Physics</li>
</ul>

<ol style="list-style: outside upper-alpha;">
<li>Maths</li>
<li>Social Science</li>
<li>Physics</li>
</ol>
```

## The marker-offset Property:

The *marker-offset* property allows you to specify the distance between the marker and the text relating to that marker. Its value should be a length as shown in the following example:

Unfortunately, however, this property is not supported in IE 6 or Netscape 7.

Here is the example:

```
<ul style="list-style: inside square; marker-offset:2em;">
<li>Maths</li>
<li>Social Science</li>
<li>Physics</li>
</ul>

<ol style="list-style: outside upper-alpha; marker-offset:2cm;">
<li>Maths</li>
<li>Social Science</li>
<li>Physics</li>
</ol>
```

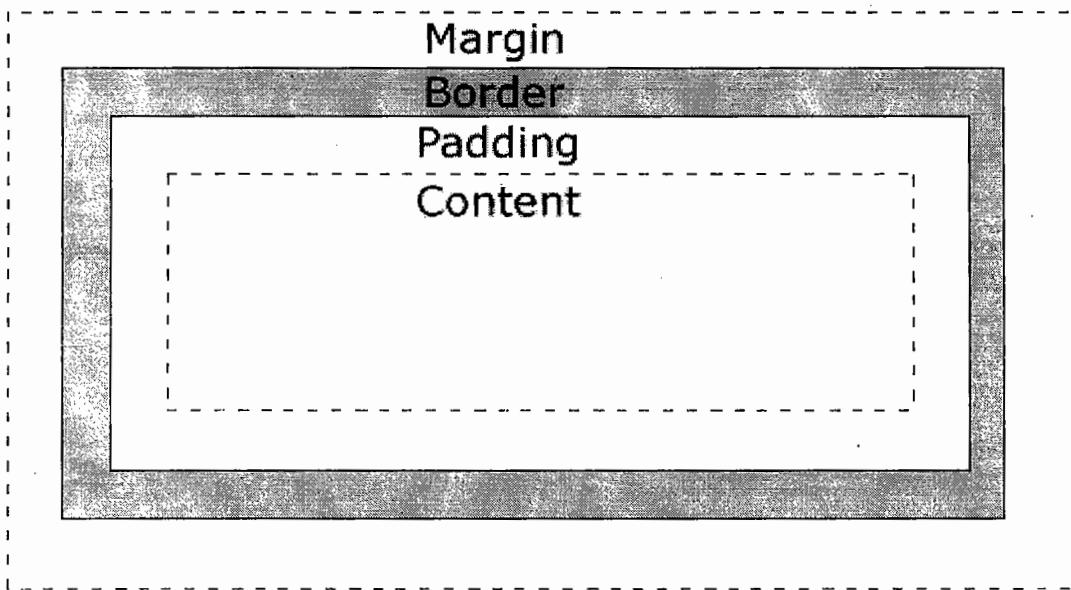
## The CSS Box Model

All HTML elements can be considered as boxes. In CSS, the term "box model" is used when talking about design and layout.

The CSS box model is essentially a box that wraps around HTML elements, and it consists of: margins, borders, padding, and the actual content.

The box model allows us to place a border around elements and space elements in relation to other elements.

The image below illustrates the box model:



Explanation of the different parts:

- **Margin** - Clears an area around the border. The margin does not have a background color, it is completely transparent
- **Border** - A border that goes around the padding and content. The border is affected by the background color of the box
- **Padding** - Clears an area around the content. The padding is affected by the background color of the box
- **Content** - The content of the box, where text or images appear

In order to set the width and height of an element correctly in all browsers, you need to know how the box model works.

## Width and Height of an Element

**Important:** When you set the width and height properties of an element with CSS, you just set the width and height of the **content area**. To calculate the full size of an element, you must also add the padding, borders and margins.

The total width of the element in the example below is 300px:

```
width:250px;  
padding:10px;  
border:5px solid gray;  
margin:10px;
```

Let's do the math:

250px (width)

+ 20px (left and right padding)

+ 10px (left and right border)

+ 20px (left and right margin)

300px

Assume that you had only 250px of space. Let's make an element with a total width of 250px:

### Example

```
width:220px;  
padding:10px;  
border:5px solid gray;  
margin:0px;
```

The total width of an element should be calculated like this:

Total element width = width + left padding + right padding + left border + right border + left margin + right margin

The total height of an element should be calculated like this:

Total element height = height + top padding + bottom padding + top border + bottom border + top margin + bottom margin

## Browsers Compatibility Issue

The example above does not display properly in IE8 and earlier versions.

IE8 and earlier versions includes padding and border in the width, if a **DOCTYPE** is NOT declared.

To fix this problem, just add a DOCTYPE to the HTML page:

### Example

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"  
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">  
<html>  
<head>  
<style type="text/css">  
  div.ex  
  {  
    width:220px;  
    padding:10px;  
    border:5px solid gray;  
    margin:0px;  
  }  
</style>  
</head>
```

## CSS - Borders

The *border* properties allow you to specify how the border of the box representing an element should look. There are three properties of a border you can change

- The **border-color** Specifies the color of a border.
- The **border-style** Specifies whether a border should be solid, dashed line, double line, or one of the other possible values.
- The **border-width** Specifies the width of a border.

Now we will see how to use these properties with examples.

## The border-color Property:

The border-color property allows you to change the color of the border surrounding an element. You can individually change the color of the bottom, left, top and right sides of an element's border using the properties:

- **border-bottom-color** changes the color of bottom border.
- **border-top-color** changes the color of top border.
- **border-left-color** changes the color of left border.
- **border-right-color** changes the color of right border.

Here is the example which shows effect of all these properties:

```
<style type="text/css">
p.example1{
    border:1px solid;
    border-bottom-color:#009900; /* Green */
    border-top-color:#FF0000;     /* Red */
    border-left-color:#330000;    /* Black */
    border-right-color:#0000CC;   /* Blue */
}
p.example2{
    border:1px solid;
    border-color:#009900;        /* Green */
}
</style>
<p class="example1">
This example is showing all borders in different colors.
</p>
<p class="example2">
This example is showing all borders in green color only.
</p>
```

## The border-style Property:

The border-style property allows you to select one of the following styles of border:

- **none**: No border. (Equivalent of border-width:0;)
- **solid**: Border is a single solid line.
- **dotted**: Border is a series of dots.
- **dashed**: Border is a series of short lines.
- **double**: Border is two solid lines.
- **groove**: Border looks as though it is carved into the page.
- **ridge**: Border looks the opposite of groove.
- **inset**: Border makes the box look like it is embedded in the page.
- **outset**: Border makes the box look like it is coming out of the canvas.
- **hidden**: Same as none, except in terms of border-conflict resolution for table elements.

You can individually change the style of the bottom, left, top, and right borders of an element using following properties:

- **border-bottom-style** changes the style of bottom border.
- **border-top-style** changes the style of top border.
- **border-left-style** changes the style of left border.
- **border-right-style** changes the style of right border.

Following is the example to show all these border styles:

```
<p style="border-width:4px; border-style:none;">  
This is a border with none width.  
</p>  
<p style="border-width:4px; border-style:solid;">  
This is a solid border.  
</p>  
<p style="border-width:4px; border-style:dashed;">  
This is a dashed border.  
</p>  
<p style="border-width:4px; border-style:double;">  
This is a double border.  
</p>  
<p style="border-width:4px; border-style:groove;">  
This is a groove border.  
</p>  
<p style="border-width:4px; border-style:ridge">  
This is a ridge border.  
</p>  
<p style="border-width:4px; border-style:inset;">  
This is a inset border.  
</p>  
<p style="border-width:4px; border-style:outset;">  
This is a outset border.  
</p>  
<p style="border-width:4px; border-style:hidden;">  
This is a hidden border.  
</p>  
<p style="border-width:4px;  
          border-top-style:solid;  
          border-bottom-style:dashed;  
          border-left-style:groove;  
          border-right-style:double;">  
This is a border with four different styles.  
</p>
```

## The border-width Property:

The border-width property allows you to set the width of an element borders. The value of this property could be either a length in px, pt or cm or it should be set to *thin, medium or thick*.

You can individually change the width of the bottom, top, left, and right borders of an element using the following properties:

- **border-bottom-width** changes the width of bottom border.
- **border-top-width** changes the width of top border.
- **border-left-width** changes the width of left border.
- **border-right-width** changes the width of right border.

Following is the example to show all these border width:

```
<p style="border-width:4px; border-style:solid;">  
This is a solid border whose width is 4px.  
</p>  
<p style="border-width:4pt; border-style:solid;">  
This is a solid border whose width is 4pt.  
</p>  
<p style="border-width:thin; border-style:solid;">  
This is a solid border whose width is thin.  
</p>  
<p style="border-width:medium; border-style:solid;">  
This is a solid border whose width is medium.  
</p>  
<p style="border-width:thick; border-style:solid;">  
This is a solid border whose width is thick.  
</p>  
<p style="border-bottom-width:4px;  
          border-top-width:10px;  
          border-left-width: 2px;  
          border-right-width:15px;  
          border-style:solid;">  
This is a border with four different width.  
</p>
```

## Border - Shorthand property

As you can see from the examples above, there are many properties to consider when dealing with borders.

To shorten the code, it is also possible to specify all the border properties in one property. This is called a shorthand property.

The shorthand property for the border properties is "border":

### Example

```
border:5px solid red;
```

When using the border property, the order of the values are:

- border-width
- border-style
- border-color

It does not matter if one of the values above are missing (although, border-style is required), as long as the rest are in the specified order.

## CSS - Margin

The margin property defines the space around an HTML element. It is possible to use negative values to overlap content.

The values of the margin property are not inherited by child elements. Remember that the adjacent vertical margins (top and bottom margins) will collapse into each other so that the distance between the blocks is not the sum of the margins, but only the greater of the two margins or the same size as one margin if both are equal.

There are following four properties to set an element margin.

- The margin A shorthand property for setting the margin properties in one declaration.
- The margin-bottom Specifies the bottom margin of an element.
- The margin-top Specifies the top margin of an element.
- The margin-left Specifies the left margin of an element.
- The margin-right Specifies the right margin of an element.

Now we will see how to use these properties with examples.

## The margin Property:

The margin property allows you set all of the properties for the four margins in one declaration. Here is the syntax to set margin around a paragraph:

```
<style type="text/css">
p {margin: 15px}
all four margins will be 15px

p {margin: 10px 2%}
top and bottom margin will be 10px, left and right margin will be 2% of the total width of the document.

p {margin: 10px 2% -10px}
top margin will be 10px, left and right margin will be 2% of the total width of the document, bottom margin will be -10px

p {margin: 10px 2% -10px auto}
top margin will be 10px, right margin will be 2% of the total width of the document, bottom margin will be -10px, left margin
will be set by the browser

</style>
```

Here is the example:

```
<p style="margin: 15px; border:1px solid black;">
all four margins will be 15px
</p>

<p style="margin:10px 2%; border:1px solid black;">
top and bottom margin will be 10px, left and right margin will be 2% of the total width of the document.
</p>

<p style="margin: 10px 2% -10px; border:1px solid black;"> top margin will be 10px, left and right margin will be 2% of
the total width of the document, bottom margin will be -10px </p>

<p style="margin: 10px 2% -10px auto; border:1px solid black;"> top margin will be 10px, right margin will be 2% of the
total width of the document, bottom margin will be -10px, left margin will be set by the browser
</p>
```

## The margin-bottom Property:

The margin-bottom property allows you set bottom margin of an element. It can have a value in length, % or auto.

Here is the example:

```
<p style="margin-bottom: 15px; border:1px solid black;">  
This is a paragraph with a specified bottom margin  
</p>  
  
<p style="margin-bottom: 5%; border:1px solid black;">  
This is another paragraph with a specified bottom margin in percent  
</p>
```

## The margin-top Property:

The margin-top property allows you set top margin of an element. It can have a value in length, % or auto.

Here is the example:

```
<p style="margin-top: 15px; border:1px solid black;">  
This is a paragraph with a specified top margin  
</p>  
  
<p style="margin-top: 5%; border:1px solid black;">  
This is another paragraph with a specified top margin in percent  
</p>
```

## The margin-left Property:

The margin-left property allows you set left margin of an element. It can have a value in length, % or auto.

Here is the example:

```
<p style="margin-left: 15px; border:1px solid black;">  
This is a paragraph with a specified left margin  
</p>  
  
<p style="margin-left: 5%; border:1px solid black;">  
This is another paragraph with a specified top margin in percent  
</p>
```

## The margin-right Property:

The margin-right property allows you set right margin of an element. It can have a value in length, % or auto.

Here is the example:

```
<p style="margin-right: 15px; border:1px solid black;">  
This is a paragraph with a specified right margin  
</p>  
  
<p style="margin-right: 5%; border:1px solid black;">  
This is another paragraph with a specified right margin in percent  
</p>
```

## Margin - Shorthand property

To shorten the code, it is possible to specify all the margin properties in one property. This is called a shorthand property.

The shorthand property for all the margin properties is "margin":

### Example

```
margin:100px 50px;
```

The margin property can have from one to four values.

- **margin:25px 50px 75px 100px;**
  - top margin is 25px
  - right margin is 50px
  - bottom margin is 75px
  - left margin is 100px
- **margin:25px 50px 75px;**
  - top margin is 25px
  - right and left margins are 50px
  - bottom margin is 75px
- **margin:25px 50px;**
  - top and bottom margins are 25px
  - right and left margins are 50px
- **margin:25px;**
  - all four margins are 25px

## CSS - Paddings

The *padding* property allows you to specify how much space should appear between the content of an element and its border:

There are following five CSS properties which can be used to control lists:

The value of this attribute should be either a length, a percentage, or the word *inherit*. If the value is *inherit* it will have the same padding as its parent element. If a percentage is used, the percentage is of the containing box.

You can also set different values for the padding on each side of the box using the following properties:

- The **padding-bottom** Specifies the bottom padding of an element.
- The **padding-top** Specifies the top padding of an element.

- The **padding-left** Specifies the left padding of an element.
- The **padding-right** Specifies the right padding of an element.
- The **padding** Serves as shorthand for the preceding properties.

Now we will see how to use these properties with examples.

### The padding-bottom Property:

The padding-bottom property sets the bottom padding (space) of an element. This can take a value in terms of length or %.

Here is the example:

```
<p style="padding-bottom: 15px; border:1px solid black;">  
This is a paragraph with a specified bottom padding  
</p>  
  
<p style="padding-bottom: 5%; border:1px solid black;">  
This is another paragraph with a specified bottom padding in percent  
</p>
```

### The padding-top Property:

The *padding-top* property sets the top padding (space) of an element. This can take a value in terms of length or %.

Here is the example:

```
<p style="padding-top: 15px; border:1px solid black;">  
This is a paragraph with a specified top padding  
</p>  
  
<p style="padding-top: 5%; border:1px solid black;">  
This is another paragraph with a specified top padding in percent  
</p>
```

### The padding-left Property:

The *padding-left* property sets the left padding (space) of an element. This can take a value in terms of length or %.

Here is the example:

```
<p style="padding-left: 15px; border:1px solid black;">  
This is a paragraph with a specified left padding  
</p>
```

```
<p style="padding-left: 15%; border:1px solid black;">  
This is another paragraph with a specified left padding in percent  
</p>
```

## The padding-right Property:

The *padding-right* property sets the right padding (space) of an element. This can take a value in terms of length of %.

Here is the example:

```
<p style="padding-right: 15px; border:1px solid black;">  
This is a paragraph with a specified right padding  
</p>  
  
<p style="padding-right: 5%; border:1px solid black;">  
This is another paragraph with a specified right padding in percent  
</p>
```

## The padding Property:

The *padding* property sets the left, right, top and bottom padding (space).of an element. This can take a value in terms of length of %.

Here is the example:

```
<p style="padding: 15px; border:1px solid black;">  
all four padding will be 15px  
</p>  
  
<p style="padding:10px 2%; border:1px solid black;">  
top and bottom padding will be 10px, left and right padding will be 2% of the total width of the document.  
</p>  
  
<p style="padding: 10px 2% 10px; border:1px solid black;"> top padding will be 10px, left and right padding will be 2% of the total width of the document, bottom padding will be 10px </p>  
  
<p style="padding: 10px 2% 10px 10px; border:1px solid black;"> top padding will be 10px, right padding will be 2% of the total width of the document, bottom padding and top padding will be 10px  
</p>
```

# CSS Tables

## Table Borders

To specify table borders in CSS, use the border property.

The example below specifies a black border for table, th, and td elements:

### Example

```
table, th, td
{
    border: 1px solid black;
```

Notice that the table in the example above has double borders. This is because both the table and the th/td elements have separate borders.

To display a single border for the table, use the border-collapse property:

## Collapse Borders

The border-collapse property sets whether the table borders are collapsed into a single border or separated:

### Example

```
table
{
    border-collapse: collapse;
}
table, th, td
{
    border: 1px solid black;
```

## Table Width and Height

Width and height of a table is defined by the width and height properties.

The example below sets the width of the table to 100%, and the height of the th elements to 50px:

### Example

```
table
{
    width: 100%;
}
th
{
    height: 50px;
```

## Table Text Alignment

The text in a table is aligned with the text-align and vertical-align properties.

The text-align property sets the horizontal alignment, like left, right, or center:

### Example

```
td
{
```

```
    text-align:right;  
}
```

The vertical-align property sets the vertical alignment, like top, bottom, or middle:

### Example

```
td  
{  
height:50px;  
vertical-align:bottom;  
}
```

## Table Padding

To control the space between the border and content in a table, use the padding property on td and th elements:

### Example

```
td  
{  
padding:15px;  
}
```

## Table Color

The example below specifies the color of the borders, and the text and background color of th elements:

### Example

```
table, td, th  
{  
border:1px solid green;  
}  
th  
{  
background-color:green;  
color:white;  
}
```

### Example

```
<style type="text/css">  
  
tr:nth-child(odd)  {  
  
background-color:red;  
  
}  
  
tr:nth-child(even)  {  
  
background-color:green;  
  
}  
  
table,tr,td,th{
```

```
        border:1px solid black;  
        border-collapse:collapse;  
    }  
  
</style>
```

### Example

```
<html>  
    <head>  
        <style type="text/css">  
            #customers  
            {  
                font-family:"Trebuchet MS", Arial, Helvetica, sans-serif;  
                width:100%;  
                border-collapse:collapse;  
            }  
            #customers td, #customers th  
            {  
                font-size:1em;  
                border:1px solid #98bf21;  
                padding:3px 7px 2px 7px;  
            }  
            #customers th  
            {  
                font-size:1.1em;  
                text-align:left;  
                padding-top:5px;  
                padding-bottom:4px;  
                background-color:#A7C942;  
                color:#ffffff;  
            }  
            #customers tr.alt td  
            {  
                color:#000000;  
                background-color:#EAF2D3;  
            }  
        </style>  
    </head>  
    <body>  
        <table id="customers">  
            <tr>  
                <th>Company</th>  
                <th>Contact</th>  
                <th>Country</th>  
            </tr>  
            <tr>
```

```

        <td>Alfreds Futterkiste</td>
        <td>Maria Anders</td>
        <td>Germany</td>
    </tr>
    <tr class="alt">
        <td>Berglunds snabbköp</td>
        <td>Christina Berglund</td>
        <td>Sweden</td>
    </tr>
    <tr>
        <td>Centro comercial Moctezuma</td>
        <td>Francisco Chang</td>
        <td>Mexico</td>
    </tr>
    <tr class="alt">
        <td>Ernst Handel</td>
        <td>Roland Mendel</td>
        <td>Austria</td>
    </tr>
    <tr>
        <td>Island Trading</td>
        <td>Helen Bennett</td>
        <td>UK</td>
    </tr>
    <tr class="alt">
        <td>Königlich Essen</td>
        <td>Philip Cramer</td>
        <td>Germany</td>
    </tr>
</table>
</body>
</html>
```

## CSS - Cursors

The *cursor* property of CSS allows you to specify the type of cursor that should be displayed to the user.

One good usage of this property is in using images for submit buttons on forms. By default, when a cursor hovers over a link, the cursor changed from a pointer to a hand. For a submit button on a form this does not happen. Therefore, using the cursor property to change the cursor to a hand whenever someone hovers over an image that is a submit button. This provides a visual clue that they can click it.

The table that follows shows possible values for the cursor property:

Value	Description
auto	The cursor is set to the browser's default value.
crosshair	The cursor is set to a crosshair shape.

Naresh i Technologies, Opp. Satyam Theatre, Ameerpet, Hyderabad, Ph: 040-23746666, 23734842  
An ISO 9001 : 2000 Certified Company

Auto	Shape of the cursor depends on the context area it is over. For example an I over text, a hand over a link, and so on...
Crosshair	A crosshair or plus sign
Default	An arrow
Pointer	A pointing hand (in IE 4 this value is hand)
Move	The I bar
e-resize	The cursor indicates that an edge of a box is to be moved right (east)
ne-resize	The cursor indicates that an edge of a box is to be moved up and right (north/east)
nw-resize	The cursor indicates that an edge of a box is to be moved up and left (north/west)
n-resize	The cursor indicates that an edge of a box is to be moved up (north)
se-resize	The cursor indicates that an edge of a box is to be moved down and right (south/east)
sw-resize	The cursor indicates that an edge of a box is to be moved down and left (south/west)
s-resize	The cursor indicates that an edge of a box is to be moved down (south)
w-resize	The cursor indicates that an edge of a box is to be moved left (west)
Text	The I bar
Wait	An hour glass
Help	A question mark or balloon, ideal for use over help buttons
<url>	The source of a cursor image file

**NOTE:** You should try to use only these values to add helpful information for users, and in places they would expect to see that cursor. For example, using the crosshair when someone hovers over a link can confuse visitors.

Here is the example:

```

<p>Move the mouse over the words to see the cursor change:</p>
<div style="cursor:auto">Auto</div>
<div style="cursor:crosshair">Crosshair</div>
<div style="cursor:default">Default</div>
<div style="cursor:pointer">Pointer</div>
<div style="cursor:move">Move</div>
<div style="cursor:e-resize">e-resize</div>
<div style="cursor:ne-resize">ne-resize</div>
<div style="cursor:nw-resize">nw-resize</div>
<div style="cursor:n-resize">n-resize</div>
<div style="cursor:se-resize">se-resize</div>
<div style="cursor:sw-resize">sw-resize</div>
<div style="cursor:s-resize">s-resize</div>
<div style="cursor:w-resize">w-resize</div>
<div style="cursor:text">text</div>
<div style="cursor:wait">wait</div>
<div style="cursor:help">help</div>

```

## CSS - Scrollbars

There may be a case when an element's content might be larger than the amount of space allocated to it. For example given width and height properties did not allow enough room to accommodate the content of the element.

CSS provides a property called *overflow* which tells the browser what to do if the box's contents is larger than the box itself. This property can take one of the following values:

Value	Description
visible	Allows the content to overflow the borders of its containing element.
hidden	The content of the nested element is simply cut off at the border of the containing element and no scrollbars is visible.
scroll	The size of the containing element does not change, but the scrollbars are added to allow the user to scroll to see the content.
auto	The purpose is the same as scroll, but the scrollbar will be shown only if the content does overflow.

Here is the example:

```
<style type="text/css">
.scroll{
    display:block;
    border: 1px solid red;
    padding:5px;
    margin-top:5px;
    width:300px;
    height:50px;
    overflow:scroll;
}
.auto{
    display:block;
    border: 1px solid red;
    padding:5px;
    margin-top:5px;
    width:300px;
    height:50px;
    overflow:auto; /* to get horizontal scroll in mozilla : overflow:-moz-scrollbars-horizontal */
}
</style>
<p>Example of scroll value:</p>
<div class="scroll">
    I am going to keep lot of content here just to show
    you how scrollbars works if there is an overflow in
    an element box. This provides your horizontal as well
    as vertical scrollbars.
</div>
<br />

<p>Example of auto value:</p>
<div class="auto">
    I am going to keep lot of content here just to show
    you how scrollbars works if there is an overflow in
    an element box. This provides your horizontal as well

```

```
as vertical scrollbars.  
</div>
```

## CSS - Images

Images are very important part of any Web Page. Though it is not recommended to include lot of images but it is still important to use good images wherever it is required.

CSS plays a good role to control image display. You can set following image properties using CSS.

- The **border** property is used to set the width of an image border.
- The **height** property is used to set the height of an image.
- The **width** property is used to set the width of an image.
- The **-moz-opacity** property is used to set the opacity of an image.

### The image border Property:

The *border* property of an image is used to set the width of an image border. This property can have a value in length or in %.

A width of zero pixels means no border.

Here is the example:

```
  
<br />  

```

### The image height Property:

The *height* property of an image is used to set the height of an image. This property can have a value in length or in %. While giving value in %, it applies it in respect of the box in which an image is available.

Here is the example:

```
  
<br />  

```

## The image width Property:

The *width* property of an image is used to set the width of an image. This property can have a value in length or in %. While giving value in %, it applies it in respect of the box in which an image is available.

Here is the example:

```
  
<br />  

```

## The -moz-opacity Property:

The *-moz-opacity* property of an image is used to set the opacity of an image. This property is used to create a transparent image in Mozilla. IE uses **filter:alpha(opacity=x)** to create transparent images.

In Mozilla (-moz-opacity:x) x can be a value from 0.0 - 1.0. A lower value makes the element more transparent (The same things goes for the CSS3-valid syntax opacity:x).

In IE (filter:alpha(opacity=x)) x can be a value from 0 - 100. A lower value makes the element more transparent.

Here is the example:

```

```

## Example 1 - Creating a Transparent Image

The CSS3 property for transparency is **opacity**.

First we will show you how to create a transparent image with CSS.

```
img  
{  
  opacity:0.4;  
  filter:alpha(opacity=40); /* For IE8 and earlier */  
}
```

IE9, Firefox, Chrome, Opera, and Safari use the property **opacity** for transparency. The opacity property can take a value from 0.0 - 1.0. A lower value makes the element more transparent.

IE8 and earlier use **filter:alpha(opacity=x)**. The x can take a value from 0 - 100. A lower value makes the element more transparent.

### Example 2 - Image Transparency - Hover Effect

The CSS looks like this:

```
img
{
    opacity:0.4;
    filter:alpha(opacity=40); /* For IE8 and earlier */
}
img:hover
{
    opacity:1.0;
    filter:alpha(opacity=100); /* For IE8 and earlier */
}
```

The first CSS block is similar to the code in Example 1. In addition, we have added what should happen when a user hover over one of the images. In this case we want the image to NOT be transparent when the user hover over it.

The CSS for this is: **opacity=1**.

IE8 and earlier: **filter:alpha(opacity=100)**.

When the mouse pointer moves away from the image, the image will be transparent again.

### Example 3 - Text in Transparent Box

```
<html>
<head>
<style type="text/css">
    div.background
    {
        width:500px;
        height:250px;
        background:url(klematis.jpg) repeat;
        border:2px solid black;
    }
    div.transbox
    {
        width:400px;
        height:180px;
        margin:30px 50px;
        background-color:#ffffff;
        border:1px solid black;
        opacity:0.6;
        filter:alpha(opacity=60); /* For IE8 and earlier */
    }
</style>
</head>
<body>
    <div class="background"></div>
    <div class="transbox">
        <p>This is a transparent box!</p>
    </div>
</body>
</html>
```

```
}

div.transbox p
{
    margin:30px 40px;
    font-weight:bold;
    color:#000000;
}

</style>
</head>

<body>

<div class="background">
    <div class="transbox">
        <p>This is some text that is placed in the transparent box.  

        This is some text that is placed in the transparent box.  

        This is some text that is placed in the transparent box.  

        This is some text that is placed in the transparent box.  

        This is some text that is placed in the transparent box.  

        </p>
    </div>
</div>

</body>
</html>
```

First, we create a div element (class="background") with a fixed height and width, a background image, and a border. Then we create a smaller div (class="transbox") inside the first div element. The "transbox" div have a fixed width, a background color, and a border - and it is transparent. Inside the transparent div, we add some text inside a p element.

## CSS Display and Visibility

The display property specifies if/how an element is displayed, and the visibility property specifies if an element should be visible or hidden.

### Hiding an Element - display:none or visibility:hidden

Hiding an element can be done by setting the **display** property to "**none**" or the **visibility** property to "**hidden**". However, notice that these two methods produce different results:

**visibility:hidden** hides an element, but it will still take up the same space as before. The element will be hidden, but still affect the layout.

### Example

```
h1.hidden {visibility:hidden;}
```

```
<html>
```

```
<head>
<style type="text/css">
    h1.hidden {visibility:hidden;}
</style>
</head>

<body>
    <h1>This is a visible heading</h1>
    <h1 class="hidden">This is a hidden heading</h1>
    <p>Notice that the hidden heading still takes up space.</p>
</body>
</html>
```

display:none hides an element, and it will not take up any space. The element will be hidden, and the page will be displayed as the element is not there:

### Example

```
h1.hidden {display:none;}
```

```
<html>
<head>
<style type="text/css">
    h1.hidden {display:none;}
</style>
</head>

<body>
    <h1>This is a visible heading</h1>
    <h1 class="hidden">This is a hidden heading</h1>
    <p>Notice that the hidden heading does not take up space.</p>
</body>

</html>
```

## CSS Display - Block and Inline Elements

A block element is an element that takes up the full width available, and has a line break before and after it.

Examples of block elements:

- <h1>
- <p>
- <div>

An inline element only takes up as much width as necessary, and does not force line breaks.

Examples of inline elements:

- <span>
- <a>

## Changing How an Element is Displayed

Changing an inline element to a block element, or vice versa, can be useful for making the page look a specific way, and still follow web standards.

The following example displays list items as inline elements:

### Example

li {display:inline;}

```
<html>
<head>
<style type="text/css">
    li{display:inline;}
</style>
</head>
<body>

    <p>Display this link list as a horizontal menu:</p>

    <ul>
        <li><a href="/html/default.asp" target="_blank">HTML</a></li>
        <li><a href="/css/default.asp" target="_blank">CSS</a></li>
        <li><a href="/js/default.asp" target="_blank">JavaScript</a></li>
        <li><a href="/xml/default.asp" target="_blank">XML</a></li>
    </ul>

</body>
</html>
```

The following example displays span elements as block elements:

### Example

span {display:block;}

```
<html>
<head>
<style type="text/css">
    span
    {
        display:block;
    }
</style>
</head>
<body>
```

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

    <h2>Nirvana</h2>
        <span>Record: MTV Unplugged in New
York</span>
        <span>Year: 1993</span>

    <h2>Radiohead</h2>
        <span>Record: OK Computer</span>
        <span>Year: 1997</span>

</body>
</html>
```

**Note:** Changing the display type of an element changes only how the element is displayed, NOT what kind of element it is. For example: An inline element set to display:block is not allowed to have a block element nested inside of it.

## CSS Positioning

### Positioning

The CSS positioning properties allow you to position an element. It can also place an element behind another, and specify what should happen when an element's content is too big.

Elements can be positioned using the **top**, **bottom**, **left**, and **right** properties. However, these properties will not work unless the **position** property is set first. They also work differently depending on the positioning method.

There are four different positioning methods.

### Static Positioning

HTML elements are positioned static by default. A static positioned element is always positioned according to the normal flow of the page.

Static positioned elements are not affected by the top, bottom, left, and right properties.

### Fixed Positioning

An element with fixed position is positioned relative to the browser window.

It will not move even if the window is scrolled:

### Example

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html>
<head>
<style type="text/css">
    img.pos_fixed
    {
        position:fixed;
        top:30px;
        right:5px;
    }
</style>
</head>
<body>



<p><b>Note:</b> IE7 and IE8 supports the fixed value only if a
!DOCTYPE is specified.</p>

<p>Some text</p><p>Some text</p><p>Some text</p><p>Some text</p><p>Some
text</p><p>Some text</p><p>Some text</p><p>Some text</p><p>Some text</p><p>Some
text</p><p>Some text</p><p>Some text</p><p>Some text</p><p>Some text</p><p>Some
text</p><p>Some text</p>
<p>Some text</p><p>Some text</p><p>Some text</p><p>Some text</p><p>Some
text</p><p>Some text</p><p>Some text</p><p>Some text</p><p>Some text</p><p>Some
text</p><p>Some text</p><p>Some text</p><p>Some text</p><p>Some text</p><p>Some
text</p><p>Some text</p>
</body>
</html>
```

**Note:** IE7 and IE8 support the fixed value only if a !DOCTYPE is specified.

Fixed-positioned elements are removed from the normal flow. The document and other elements behave like the fixed positioned element does not exist.

Fixed positioned elements can overlap other elements.

## Relative Positioning

A relative positioned element is positioned relative to its normal position.

### Example

```
<html>
<head>
<style type="text/css">
    h2.pos_left
    {
```

```
position:relative;
left:-20px;
}

h2.pos_right
{
    position:relative;
    left:20px;
}
</style>
</head>

<body>
<h2>This is a heading with no position</h2>
<h2 class="pos_left">This heading is moved left according to its normal position</h2>
<h2 class="pos_right">This heading is moved right according to its normal position</h2>
<p>Relative positioning moves an element RELATIVE to its original position.</p>
<p>The style "left:-20px" subtracts 20 pixels from the element's original left position.</p>
<p>The style "left:20px" adds 20 pixels to the element's original left position.</p>
</body>

</html>
```

The content of relatively positioned elements can be moved and overlap other elements, but the reserved space for the element is still preserved in the normal flow.

## Example

```
<html>
<head>
<style type="text/css">
    h2.pos_top
    {
        position:relative;
        top:-50px;
    }
</style>
</head>

<body>
<h2>This is a heading with no position</h2>
<h2 class="pos_top">This heading is moved upwards according to its normal position</h2>
<p><b>Note:</b> Even if the content of the relatively positioned element is moved, the reserved space for the element is still preserved in the normal flow.</p>
</body>

</html>
```

Relatively positioned elements are often used as container blocks for absolutely positioned elements.

## Absolute Positioning

An absolute position element is positioned relative to the first parent element that has a position other than static. If no such element is found, the containing block is <html>:

### Example

```
<html>
<head>
<style type="text/css">
    h2
    {
        position:absolute;
        left:100px;
        top:150px;
    }
</style>
</head>

<body>
<h2>This is a heading with an absolute position</h2>
<p>With absolute positioning, an element can be placed anywhere on a page. The heading below is placed
100px from the left of the page and 150px from the top of the page.</p>
</body>

</html>
```

Absolutely positioned elements are removed from the normal flow. The document and other elements behave like the absolutely positioned element does not exist.

Absolutely positioned elements can overlap other elements.

## CSS - Layers

CSS gives you opportunity to create layers of various divisions. The CSS layers refer to applying the *z-index* property to elements that overlap with each other.

The *z-index* property is used alongwith *position* property to create an effect of layers. You can specify which element should come on top and which element should come at bottom.

A *z-index* property can help you to create more complex webpage layouts. Following is the example which shows how to create layers in CSS.

```
<div style="background-color:red;
width:300px;
```

```
height:100px;
position:relative;
top:10px;
left:80px;
z-index:2">
</div>
<div style="background-color:yellow;
width:300px;
height:100px;
position:relative;
top:-60px;
left:35px;
z-index:1;">
</div>
<div style="background-color:green;
width:300px;
height:100px;
position:relative;
top:-220px;
left:120px;
z-index:3;">
</div>
```

## CSS Float

### What is CSS Float?

With CSS float, an element can be pushed to the left or right, allowing other elements to wrap around it.

Float is very often used for images, but it is also useful when working with layouts.

### How Elements Float

Elements are floated horizontally, this means that an element can only be floated left or right, not up or down.

A floated element will move as far to the left or right as it can. Usually this means all the way to the left or right of the containing element.

The elements after the floating element will flow around it.

The elements before the floating element will not be affected.

If an image is floated to the right, a following text flows around it, to the left:

### Example

```
<html>
<head>
<style type="text/css">
img
{
```

```
float:right;
}
</style>
</head>

<body>
<p>In the paragraph below, we have added an image with style <b>float:right</b>. The result is that the image will float to the right in the paragraph.</p>
<p>

This is some text. This is some text. This is some text.
This is some text. This is some text. This is some text.
This is some text. This is some text. This is some text.
This is some text. This is some text. This is some text.
This is some text. This is some text. This is some text.
This is some text. This is some text. This is some text.
This is some text. This is some text. This is some text.
This is some text. This is some text. This is some text.
This is some text. This is some text. This is some text.
</p>
</body>

</html>
```

## Floating Elements Next to Each Other

If you place several floating elements after each other, they will float next to each other if there is room.

Here we have made an image gallery using the float property:

### Example

```
<html>
<head>
<style type="text/css">
    .thumbnail
    {
        float:left;
        width:110px;
        height:90px;
        margin:5px;
    }
</style>
</head>

<body>
<h3>Image Gallery</h3>
<p>Try resizing the window to see what happens when the images does not have enough room.</p>
```

```









</body>
</html>

```

### Turning off Float - Using Clear

Elements after the floating element will flow around it. To avoid this, use the **clear** property.

The **clear** property specifies which sides of an element other floating elements are not allowed.

Add a text line into the image gallery, using the clear property:

### Example

```

<html>
<head>
<style type="text/css">
    .thumbnail
    {
        float:left;
        width:110px;
        height:90px;
        margin:5px;
    }
    .text_line
    {
        clear:both;
        margin-bottom:2px;
    }
</style>
</head>

<body>
<h3>Image Gallery</h3>
<p>Try resizing the window to see what happens when the images does not have enough room.</p>
    
    
    
    
    <h3 class="text_line">Second row</h3>
    
    

```

```


</body>
</html>
```

## CSS Pseudo-classes

CSS pseudo-classes are used to add special effects to some selectors.

### Syntax

The syntax of pseudo-classes:

```
selector:pseudo-class {property:value;}
```

CSS classes can also be used with pseudo-classes:

```
selector.class:pseudo-class {property:value;}
```

### Anchor Pseudo-classes

Links can be displayed in different ways in a CSS-supporting browser:

#### Example

```
a:link {color:#FF0000;} /* unvisited link */
a:visited {color:#00FF00;} /* visited link */
a:hover {color:#FF00FF;} /* mouse over link */
a:active {color:#0000FF;} /* selected link */
```

**Note:** a:hover MUST come after a:link and a:visited in the CSS definition in order to be effective!!

**Note:** a:active MUST come after a:hover in the CSS definition in order to be effective!!

**Note:** Pseudo-class names are not case-sensitive.

### Pseudo-classes and CSS Classes

Pseudo-classes can be combined with CSS classes:

```
a.red:visited {color:#FF0000;}
```

```
<a class="red" href="css_syntax.asp">CSS Syntax</a>
```

If the link in the example above has been visited, it will be displayed in red.

## CSS - The :first-child Pseudo-class

The **:first-child** pseudo-class matches a specified element that is the first child of another element.

**Note:** For :first-child to work in IE8 and earlier, a <!DOCTYPE> must be declared.

### Match the first <p> element

In the following example, the selector matches any <p> element that is the first child of any element:

#### Example

```
<html>
<head>
<style type="text/css">
    p:first-child
    {
        color:blue;
    }
</style>
</head>

<body>
    <p>I am a strong man.</p>
    <p>I am a strong man.</p>
</body>
</html>
```

### Match the first <i> element in all <p> elements

In the following example, the selector matches the first <i> element in all <p> elements:

#### Example

```
<html>
<head>
<style type="text/css">
    p > i:first-child
    {
        color:red;
    }
</style>
</head>

<body>
    <p>I am a <i>strong</i> man. I am a <i>strong</i>
man.</p>
    <p>I am a <i>strong</i> man. I am a <i>strong</i>
man.</p>
</body>
```

```
</html>
```

## Match all <i> elements in all first child <p> elements

In the following example, the selector matches all <i> elements in <p> elements that are the first child of another element:

### Example

```
<html>
<head>
<style type="text/css">
    p:first-child i
    {
        color:red;
    }
</style>
</head>

<body>
    <p>I am a <i>strong</i> man. I am a
<i>strong</i> man.</p>
    <p>I am a <i>strong</i> man. I am a
<i>strong</i> man.</p>
</body>
</html>
```

## CSS Pseudo-elements

CSS pseudo-elements are used to add special effects to some selectors.

### Syntax

The syntax of pseudo-elements:

```
selector:pseudo-element {property:value;}
```

CSS classes can also be used with pseudo-elements:

```
selector.class:pseudo-element {property:value;}
```

### The :first-line Pseudo-element

The "first-line" pseudo-element is used to add a special style to the first line of a text.

In the following example the browser formats the first line of text in a p element according to the style in the "first-line" pseudo-element (where the browser breaks the line, depends on the size of the browser window):

### Example

```
p:first-line  
{  
    color:#ff0000;  
}
```

**Note:** The "first-line" pseudo-element can only be used with block-level elements.

**Note:** The following properties apply to the "first-line" pseudo-element:

- font properties
- color properties
- background properties
- word-spacing
- letter-spacing
- text-decoration
- vertical-align
- text-transform
- line-height
- clear

### The :first-letter Pseudo-element

The "first-letter" pseudo-element is used to add a special style to the first letter of a text:

### Example

```
p:first-letter  
{  
    color:#ff0000;  
    font-size:xx-large;  
}
```

**Note:** The "first-letter" pseudo-element can only be used with block-level elements.

**Note:** The following properties apply to the "first-letter" pseudo- element:

- font properties
- color properties
- background properties
- margin properties
- padding properties
- border properties
- text-decoration
- vertical-align (only if "float" is "none")

- text-transform
- line-height
- float
- clear

## Pseudo-elements and CSS Classes

Pseudo-elements can be combined with CSS classes:

```
p.article:first-letter {color:#ff0000;}
```

```
<p class="article">A paragraph in an article</p>
```

The example above will display the first letter of all paragraphs with class="article", in red.

## Multiple Pseudo-elements

Several pseudo-elements can also be combined.

In the following example, the first letter of a paragraph will be red, in an xx-large font size. The rest of the first line will be blue, and in small-caps. The rest of the paragraph will be the default font size and color:

### Example

```
p:first-letter
{
    color:#ff0000;
    font-size:xx-large;
}
p:first-line
{
    color:#0000ff;
    font-variant:small-caps;
}
```

## CSS - The :before Pseudo-element

The ":before" pseudo-element can be used to insert some content before the content of an element.

The following example inserts an image before each <h1> element:

### Example

```
<html>
<head>
<style type="text/css">
    h1:before
    {
        content:url(images/home.jpg);
```

```

        }
</style>
</head>
<body>
    <h1>welcome to my home</h1>
</body>
</html>

```

## CSS - The :after Pseudo-element

The ":after" pseudo-element can be used to insert some content after the content of an element.

The following example inserts an image after each <h1> element:

### Example

```

h1:after
{
content:url(images/home.jpg);
}

```

## All CSS Pseudo Classes/Elements

Selector	Example	Example description
:link	a:link	Selects all unvisited links
:visited	a:visited	Selects all visited links
:active	a:active	Selects the active link
:hover	a:hover	Selects links on mouse over
:focus	input:focus	Selects the input element which has focus
:first-letter	p:first-letter	Selects the first letter of every <p> element
:first-line	p:first-line	Selects the first line of every <p> element
:first-child	p:first-child	Selects every <p> elements that is the first child of its parent
:before	p:before	Insert content before every <p> element
:after	p:after	Insert content after every <p> element
:lang( <i>language</i> )	p:lang(it)	Selects every <p> element with a lang attribute value starting with "it"

## CSS Attribute Selectors

### Style HTML Elements With Specific Attributes

It is possible to style HTML elements that have specific attributes, not just class and id.

**Note:** IE7 and IE8 support attribute selectors only if a !DOCTYPE is specified. Attribute selection is **NOT** supported in IE6 and lower.

#### Attribute Selector

The example below styles all elements with a title attribute:

#### Example

```
[title]
{
    color:blue;
}
```

#### Attribute and Value Selector

The example below styles all elements with title="WebApplication":

#### Example

```
[title= WebApplication]
{
    border:5px solid green;
}
```

#### Attribute and Value Selector - Multiple Values

The example below styles all elements with a title attribute that contains a specified value. This works even if the attribute has space separated values:

#### Example

```
[title~=hello]
{
    color:blue;
}
```

The example below styles all elements with a lang attribute that contains a specified value. This works even if the attribute has hyphen ( - ) separated values:

**Example**

```
[lang|=en]
{
    color:blue;
}
```

**Styling Forms**

The attribute selectors are particularly useful for styling forms without class or ID:

**Example**

```
input[type="text"]
{
    width:150px;
    display:block;
    margin-bottom:10px;
    background-color:yellow;
}
input[type="button"]
{
    width:120px;
    margin-left:35px;
    display:block;
}
```

## 4 Introduction

- ⇒ JavaScript was invented by Brendan Eich at Netscape (with Navigator 2.0), and has appeared in all browsers since 1996.
- ⇒ Javascript is from Netscape
- ⇒ JavaScript initially it was called as LiveScript
- ⇒ JavaScript is a lightweight, interpreted programming language that allows you to build interactivity into otherwise static HTML pages.
- ⇒ JavaScript is used in billions of Web pages to add functionality, validate forms, communicate with the server, and much more.
- ⇒ JavaScript is the most popular scripting language on the internet, and works in all major browsers, such as Internet Explorer, Firefox, Chrome, Opera, and Safari.

## What You Should Already Know

Before you continue you should have a basic understanding of the following:

- HTML and CSS

## What is JavaScript?

- JavaScript was designed to add interactivity to HTML pages
- JavaScript is a scripting language
- A scripting language is a lightweight programming language
- JavaScript is usually embedded directly into HTML pages
- JavaScript is an interpreted language (means that scripts execute without preliminary compilation)
- Everyone can use JavaScript without purchasing a license

## Are Java and JavaScript the same?

NO!

Java and JavaScript are two completely different languages in both concept and design!

## What Can JavaScript do?

- **JavaScript gives HTML designers a programming tool** - HTML authors are normally not programmers, but JavaScript is a scripting language with a very simple syntax! Almost anyone can put small "snippets" of code into their HTML pages
- **JavaScript can react to events** - A JavaScript can be set to execute when something happens, like when a page has finished loading or when a user clicks on an HTML element
- **JavaScript can read and write HTML elements** - A JavaScript can read and change the content of an HTML element
- **JavaScript can be used to validate data** - A JavaScript can be used to validate form data before it is submitted to a server. This saves the server from extra processing

- **JavaScript can be used to detect the visitor's browser** - A JavaScript can be used to detect the visitor's browser, and - depending on the browser - load another page specifically designed for that browser
- **JavaScript can be used to create cookies** - A JavaScript can be used to store and retrieve information on the visitor's computer

## Limitations with JavaScript:

We can not treat JavaScript as a full fledged programming language. It lacks the following important features:

- Client-side JavaScript does not allow the reading or writing of files. This has been kept for security reason.
- JavaScript can not be used for Networking applications because there is no such support available.
- JavaScript doesn't have any multithreading or multiprocess capabilities.

## JavaScript Development Tools:

One of JavaScript's strengths is that expensive development tools are not usually required. You can start with a simple text editor such as Notepad.

Since it is an interpreted language inside the context of a web browser, you don't even need to buy a compiler.

To make our life simpler, various vendors have come up with very nice JavaScript editing tools. Few of them are listed here:

- **Microsoft FrontPage:** Microsoft has developed a popular HTML editor called FrontPage. FrontPage also provides web developers with a number of JavaScript tools to assist in the creation of an interactive web site.
- **Macromedia Dreamweaver MX:** Macromedia Dreamweaver MX is a very popular HTML and JavaScript editor in the professional web development crowd. It provides several handy prebuilt JavaScript components, integrates well with databases, and conforms to new standards such as XHTML and XML.
- **Macromedia HomeSite 5:** This provided a well-liked HTML and JavaScript editor, which will manage their personal web site just fine.

## Where JavaScript is Today ?

The ECMAScript is the standard for scripting languages

The JavaScript 2.0 adheres to ECMAScript Edition 4.

Today, Netscape's JavaScript and Microsoft's JScript conform to the ECMAScript standard

## JavaScript Syntax

A JavaScript consists of JavaScript statements that are placed within the `<script>... </script>` HTML tags in a web page.

You can place the `<script>` tag containing your JavaScript anywhere within you web page but it is preferred way to keep it within the `<head>` tags.

The `<script>` tag alert the browser program to begin interpreting all the text between these tags as a script. So simple syntax of your JavaScript will be as follows

```
<script ...>  
    JavaScript code  
</script>
```

The script tag takes two important attributes:

- **language:** This attribute specifies what scripting language you are using. Typically, its value will be `javascript`. Although recent versions of HTML (and XHTML, its successor) have phased out the use of this attribute.
- **type:** This attribute is what is now recommended to indicate the scripting language in use and its value should be set to "`text/javascript`".

So your JavaScript segment will look like:

```
<script language="javascript" type="text/javascript">  
    JavaScript code  
</script>
```

## Your First JavaScript Script:

Let us write our class example to print out "Hello World".

```
<html>  
<body>  
<script language="javascript" type="text/javascript">  
    document.write("Hello World!")  
</script>  
</body>  
</html>
```

Next, we call a function `document.write` which writes a string into our HTML document. This function can be used to write text, HTML, or both. So above code will display following result:

```
Hello World!
```

## Whitespace and Line Breaks:

JavaScript ignores spaces, tabs, and newlines that appear in JavaScript programs.

Because you can use spaces, tabs, and newlines freely in your program so you are free to format and indent your programs in a neat and consistent way that makes the code easy to read and understand.

## Semicolons are Optional:

Simple statements in JavaScript are generally followed by a semicolon character, just as they are in C, C++, and Java. JavaScript, however, allows you to omit this semicolon if your statements are each placed on a separate line. For example, the following code could be written without semicolons

```
<script language="javascript" type="text/javascript">
<!--
var1 = 10
var2 = 20
//-->
</script>
```

But when formatted in a single line as follows, the semicolons are required:

```
<script language="javascript" type="text/javascript">
<!--
var1 = 10; var2 = 20;
//-->
</script>
```

**Note:** It is a good programming practice to use semicolons.

## Case Sensitivity:

JavaScript is a case-sensitive language. This means that language keywords, variables, function names, and any other identifiers must always be typed with a consistent capitalization of letters.

So identifiers *Time*, *TIme* and *TIME* will have different meanings in JavaScript.

**NOTE:** Care should be taken while writing your variable and function names in JavaScript.

## Some Browsers do Not Support JavaScript

Browsers that do not support JavaScript, will display JavaScript as page content.

To prevent them from doing this, and as a part of the JavaScript standard, the HTML comment tag should be used to "hide" the JavaScript.

Just add an HTML comment tag <!-- before the first JavaScript statement, and a --> (end of comment) after the last JavaScript statement, like this:

```
<html>
<body>
<script type="text/javascript">
<!--
document.getElementById("demo").innerHTML=Date();
//-->
</script>
```

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

The two forward slashes at the end of comment line (//) is the JavaScript comment symbol. This prevents JavaScript from executing the --> tag.

## Comments in JavaScript:

JavaScript supports both C-style and C++-style comments, Thus:

- Any text between a // and the end of a line is treated as a comment and is ignored by JavaScript.
- Any text between the characters /\* and \*/ is treated as a comment. This may span multiple lines.
- JavaScript also recognizes the HTML comment opening sequence <!--. JavaScript treats this as a single-line comment, just as it does the // comment.
- The HTML comment closing sequence --> is not recognized by JavaScript so it should be written as //-->.

### Example:

```
<script language="javascript" type="text/javascript">
<!--

// This is a comment. It is similar to comments in C++

/*
 * This is a multiline comment in JavaScript
 * It is very similar to comments in C Programming
 */
//-->
</script>
```

## Enabling JavaScript in Browsers

All the modern browsers come with built-in support for JavaScript. Many times you may need to enable or disable this support manually.

### JavaScript in Internet Explorer:

Here are simple steps to turn on or turn off JavaScript in your Internet Explorer:

1. Follow **Tools-> Internet Options** from the menu
2. Select **Security** tab from the dialog box
3. Click the **Custom Level** button
4. Scroll down till you find **Scripting option**
5. Select **Enable** radio button under **Active scripting**
6. Finally click OK and come out

To disable JavaScript support in your Internet Explorer, you need to select **Disable** radio button under **Active scripting**.

### JavaScript in Firefox:

Here are simple steps to turn on or turn off JavaScript in your Firefox:

1. Follow **Tools-> Options**

from the menu

2. Select **Content** option from the dialog box
3. Select *Enable JavaScript* checkbox
4. Finally click OK and come out

To disable JavaScript support in your Firefox, you should not select *Enable JavaScript* checkbox.

## JavaScript in Opera:

Here are simple steps to turn on or turn off JavaScript in your Opera:

1. Follow **Tools-> Preferences**

from the menu

2. Select **Advanced** option from the dialog box
3. Select **Content** from the listed items
4. Select *Enable JavaScript* checkbox
5. Finally click OK and come out

To disable JavaScript support in your Opera, you should not select *Enable JavaScript* checkbox.

## Warning for Non-JavaScript Browsers:

If you have to do something important using JavaScript then you can display a warning message to the user using <noscript> tags.

You can add a *noscript* block immediately after the script block as follows:

```
<html>
<body>

<script language="javascript" type="text/javascript">
<!--
    document.write("Hello World!")
//-->
</script>

<noscript>
    Sorry...JavaScript is needed to go ahead.
</noscript>
</body>
</html>
```

Now, if user's browser does not support JavaScript or JavaScript is not enabled then message from </noscript> will be displayed on the screen.

## JavaScript Placement in HTML File

There is a flexibility given to include JavaScript code anywhere in an HTML document. But there are following most preferred ways to include JavaScript in your HTML file.

- Script in `<head>...</head>` section.
- Script in `<body>...</body>` section.
- Script in `<body>...</body>` and `<head>...</head>` sections.
- Script in an external file and then include in `<head>...</head>` section.

In the following section we will see how we can put JavaScript in different ways:

### JavaScript in `<head>...</head>` section:

If you want to have a script run on some event, such as when a user clicks somewhere, then you will place that script in the head as follows:

```
<html>
<head>
<script type="text/javascript">
<!--
function sayHello() {
    alert("Hello World")
}
//-->
</script>
</head>
<body>
<input type="button" onclick="sayHello()" value="Say Hello" />
</body>
</html>
```

### JavaScript in `<body>...</body>` section:

If you need a script to run as the page loads so that the script generates content in the page, the script goes in the `<body>` portion of the document. In this case you would not have any function defined using JavaScript:

```
<html>
<head>
</head>
<body>
<script type="text/javascript">
<!--
document.write("Hello World")
//-->
</script>
<p>This is web page body </p>
</body>
</html>
```

### JavaScript in `<body>` and `<head>` sections:

You can put your JavaScript code in `<head>` and `<body>` section altogether as follows:

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

```
<script type="text/javascript">
<!--
function sayHello() {
    alert("Hello World")
}
//-->
</script>
</head>
<body>
<script type="text/javascript">
<!--
document.write("Hello World")
//-->
</script>
<input type="button" onclick="sayHello()" value="Say Hello" />
</body>
</html>
```

## JavaScript in External File :

As you begin to work more extensively with JavaScript, you will likely find that there are cases where you are reusing identical JavaScript code on multiple pages of a site.

You are not restricted to be maintaining identical code in multiple HTML files. The *script* tag provides a mechanism to allow you to store JavaScript in an external file and then include it into your HTML files.

Here is an example to show how you can include an external JavaScript file in your HTML code using *script* tag and its *src* attribute:

```
<html>
<head>
<script type="text/javascript" src="filename.js" ></script>
</head>
<body>
.....
</body>
</html>
```

To use JavaScript from an external file source, you need to write your all JavaScript source code in a simple text file with extension ".js" and then include that file as shown above.

For example, you can keep following content in filename.js file and then you can use *sayHello* function in your HTML file after including filename.js file:

```
function sayHello() {
    alert("Hello World")
}
```

## JavaScript Variables and DataTypes

### JavaScript DataTypes:

One of the most fundamental characteristics of a programming language is the set of data types it supports. These are the type of values that can be represented and manipulated in a programming language.

JavaScript allows you to work with three primitive data types:

- Numbers eg. 123, 120.50 etc.
- Strings of text e.g. "This text string" etc.
- Boolean e.g. true or false.

JavaScript also defines two trivial data types, *null* and *undefined*, each of which defines only a single value.

In addition to these primitive data types, JavaScript supports a composite data type known as *object*. We will see an object detail in a separate chapter.

**Note:** Java does not make a distinction between integer values and floating-point values. All numbers in JavaScript are represented as floating-point values. JavaScript represents numbers using the 64-bit floating-point format defined by the IEEE 754 standard.

## JavaScript Variables:

Like many other programming languages, JavaScript has variables. Variables can be thought of as named containers. You can place data into these containers and then refer to the data simply by naming the container.

Before you use a variable in a JavaScript program, you must declare it. Variables are declared with the **var** keyword as follows:

```
<script type="text/javascript">
<!--
var money;
var name;
//-->
</script>
```

You can also declare multiple variables with the same **var** keyword as follows:

```
<script type="text/javascript">
<!--
var money, name;
//-->
</script>
```

Storing a value in a variable is called variable initialization. You can do variable initialization at the time of variable creation or later point in time when you need that variable as follows:

For instance, you might create a variable named *money* and assign the value 2000.50 to it later. For another variable you can assign a value the time of initialization as follows:

```
<script type="text/javascript">
<!--
var name = "Ali";
var money;
money = 2000.50;
//-->
</script>
```

**Note:** Use the **var** keyword only for declaration or initialization once for the life of any variable name in a document. You should not re-declare same variable twice.

JavaScript is *untyped* language. This means that a JavaScript variable can hold a value of any data type. Unlike many other languages, you don't have to tell JavaScript during variable declaration what type of value the variable will hold. The value type of a variable can change during the execution of a program and JavaScript takes care of it automatically.

## JavaScript Variable Scope:

The scope of a variable is the region of your program in which it is defined. JavaScript variable will have only two scopes.

- **Global Variables:** A global variable has global scope which means it is defined everywhere in your JavaScript code.
- **Local Variables:** A local variable will be visible only within a function where it is defined. Function parameters are always local to that function.

Within the body of a function, a local variable takes precedence over a global variable with the same name. If you declare a local variable or function parameter with the same name as a global variable, you effectively hide the global variable. Following example explains it:

```
<script type="text/javascript">
<!--
var myVar = "global"; // Declare a global variable
function checkscope() {
    var myVar = "local"; // Declare a local variable
    document.write(myVar);
}
//-->
</script>
```

## JavaScript Variable Names:

While naming your variables in JavaScript keep following rules in mind.

- You should not use any of the JavaScript reserved keyword as variable name. These keywords are mentioned in the next section. For example, *break* or *boolean* variable names are not valid.
- JavaScript variable names should not start with a numeral (0-9). They must begin with a letter or the underscore character. For example, *123test* is an invalid variable name but *\_123test* is a valid one.
- JavaScript variable names are case sensitive. For example, *Name* and *name* are two different variables.

## JavaScript Reserved Words:

The following are reserved words in JavaScript. They cannot be used as JavaScript variables, functions, methods, loop labels, or any object names.

abstract boolean break byte case catch char class const continue debugger default delete do double	else enum export extends false final finally float for function goto if implements import in	instanceof int interface long native new null package private protected public return short static super	switch synchronized this throw throws transient true try typeof var void volatile while with
--	--	--	---

## JavaScript Operators

### What is an operator?

Simple answer can be given using expression  $4 + 5$  is equal to 9. Here 4 and 5 are called operands and + is called operator. JavaScript language supports following type of operators.

- Arithmetic Operators
- Comparision Operators
- Logical (or Relational) Operators
- Assignment Operators
- Conditional (or ternary) Operators

Lets have a look on all operators one by one.

### The Arithmetic Operators:

There are following arithmetic operators supported by JavaScript language:

Assume variable A holds 10 and variable B holds 20 then:

Operator	Description	Example
+	Adds two operands	A + B will give 30
-	Subtracts second operand from the first	A - B will give -10
*	Multiply both operands	A * B will give 200
/	Divide numerator by denominator	B / A will give 2
%	Modulus Operator and remainder of after an integer division	B % A will give 0
++	Increment operator, increases integer value by one	A++ will give 11
--	Decrement operator, decreases integer value by one	A-- will give 9

**Note:** Addition operator (+) works for Numeric as well as Strings. e.g. "a" + 10 will give "a10".

### The Comparison Operators:

There are following comparison operators supported by JavaScript language

Assume variable A holds 10 and variable B holds 20 then:

Operator	Description	Example
==	Checks if the value of two operands are equal or not, if yes then condition becomes true.	(A == B) is not true.
!=	Checks if the value of two operands are equal or not, if values are not equal then condition becomes true.	(A != B) is true.
>	Checks if the value of left operand is greater than the value of right operand, if yes then condition becomes true.	(A > B) is not true.
<	Checks if the value of left operand is less than the value of right operand, if yes then condition becomes true.	(A < B) is true.

<code>&gt;=</code>	Checks if the value of left operand is greater than or equal to the value of right operand, if yes then condition becomes true.	(A $\geq$ B) is not true.
<code>&lt;=</code>	Checks if the value of left operand is less than or equal to the value of right operand, if yes then condition becomes true.	(A $\leq$ B) is true.

## The Logical Operators:

There are following logical operators supported by JavaScript language

Assume variable A holds 10 and variable B holds 20 then:

Operator	Description	Example
<code>&amp;&amp;</code>	Called Logical AND operator. If both the operands are non zero then then condition becomes true.	(A $\&\&$ B) is true.
<code>  </code>	Called Logical OR Operator. If any of the two operands are non zero then then condition becomes true.	(A $\ $ B) is true.
<code>!</code>	Called Logical NOT Operator. Use to reverses the logical state of its operand. If a condition is true then Logical NOT operator will make false.	!(A $\&\&$ B) is false.

## The Bitwise Operators:

There are following bitwise operators supported by JavaScript language

Assume variable A holds 2 and variable B holds 3 then:

Operator	Description	Example
<code>&amp;</code>	Called Bitwise AND operator. It performs a Boolean AND operation on each bit of its integer arguments.	(A & B) is 2 .
<code> </code>	Called Bitwise OR Operator. It performs a Boolean OR operation on each bit of its integer arguments.	(A   B) is 3.
<code>^</code>	Called Bitwise XOR Operator. It performs a Boolean exclusive OR operation on each bit of its integer arguments. Exclusive OR means that either operand one is true or operand two is true, but not both.	(A ^ B) is 1.
<code>~</code>	Called Bitwise NOT Operator. It is a is a unary operator and operates by reversing all bits in the operand.	(~B) is -4 .
<code>&lt;&lt;</code>	Called Bitwise Shift Left Operator. It moves all bits in its first operand to the left by the number of places specified in the second operand. New bits are filled with zeros. Shifting a value left by one position is equivalent to multiplying by 2, shifting two positions is equivalent to multiplying by 4, etc.	(A $\ll$ 1) is 4.
<code>&gt;&gt;</code>	Called Bitwise Shift Right with Sign Operator. It moves all bits in its first operand to the right by the number of places specified in the second operand. The bits filled in on the left depend on the sign bit of the original operand, in order to preserve the sign of the result. If the first operand is positive, the result has zeros placed in the high bits; if the first operand is negative, the result has ones placed in the high bits. Shifting a value right one place is equivalent to dividing by 2 (discarding the remainder), shifting right two places is equivalent to integer division by 4, and so on.	(A $\gg$ 1) is 1.

>>>	Called Bitwise Shift Right with Zero Operator. This operator is just like the >> operator, except that the bits shifted in on the left are always zero,	(A >>> 1) is 1.
-----	---	-----------------

## The Assignment Operators:

There are following assignment operators supported by JavaScript language:

Operator	Description	Example
=	Simple assignment operator, Assigns values from right side operands to left side operand	C = A + B will assigne value of A + B into C
+=	Add AND assignment operator, It adds right operand to the left operand and assign the result to left operand	C += A is equivalent to C = C + A
-=	Subtract AND assignment operator, It subtracts right operand from the left operand and assign the result to left operand	C -= A is equivalent to C = C - A
*=	Multiply AND assignment operator, It multiplies right operand with the left operand and assign the result to left operand	C *= A is equivalent to C = C * A
/=	Divide AND assignment operator, It divides left operand with the right operand and assign the result to left operand	C /= A is equivalent to C = C / A
%=	Modulus AND assignment operator, It takes modulus using two operands and assign the result to left operand	C %= A is equivalent to C = C % A

**Note:** Same logic applies to Bitwise operators so they will become like <<=, >>=, >>=, &=, |= and ^=.

## Miscellaneous Operator

### The Conditional Operator (? :)

There is an oprator called conditional operator. This first evaluates an expression for a true or false value and then execute one of the two given statements depending upon the result of the evaluation. The conditioanl operator has this syntax:

Operator	Description	Example
? :	Conditional Expression	If Condition is true ? Then value X : Otherwise value Y

### The *typeof* Operator

The *typeof* is a unary operator that is placed before its single operand, which can be of any type. Its value is a string indicating the data type of the operand.

The *typeof* operator evaluates to "number", "string", or "boolean" if its operand is a number, string, or boolean value and returns true or false based on the evaluation.

Here is the list of return values for the *typeof* Operator :

Type	String Returned by <i>typeof</i>
Number	"number"
String	"string"
Boolean	"boolean"

Object	"object"
Function	"function"
Undefined	"undefined"
Null	"object"

## JavaScript if...else Statements

While writing a program, there may be a situation when you need to adopt one path out of the given two paths. So you need to make use of conditional statements that allow your program to make correct decisions and perform right actions.

JavaScript supports conditional statements which are used to perform different actions based on different conditions. Here we will explain **if..else** statement.

JavaScript supports following forms of **if..else** statement:

- if statement
- if...else statement
- if...else if... statement.

### if statement:

The **if** statement is the fundamental control statement that allows JavaScript to make decisions and execute statements conditionally.

### Syntax:

```
if (expression) {
    Statement(s) to be executed if expression is true
}
```

Here JavaScript *expression* is evaluated. If the resulting value is *true*, given *statement(s)* are executed. If *expression* is *false* then no statement would be not executed. Most of the times you will use comparison operators while making decisions.

### Example:

```
<script type="text/javascript">
<!--
var age = 20;
if( age > 18 ){
    document.write("<b>Qualifies for driving</b>");
}
//-->
</script>
```

This will produce following result:

```
Qualifies for driving
```

## if...else statement:

The **if...else** statement is the next form of control statement that allows JavaScript to execute statements in more controlled way.

### Syntax:

```
if (expression){  
    Statement(s) to be executed if expression is true  
}else{  
    Statement(s) to be executed if expression is false  
}
```

Here JavaScript *expression* is evaluated. If the resulting value is *true*, given *statement(s)* in the *if* block, are executed. If *expression* is *false* then given *statement(s)* in the *else* block, are executed.

### Example:

```
<script type="text/javascript">  
<!--  
var age = 15;  
if( age > 18 ){  
    document.write("<b>Qualifies for driving</b>");  
}else{  
    document.write("<b>Does not qualify for driving</b>");  
}  
//-->  
</script>
```

This will produce following result:

```
Does not qualify for driving
```

## if...else if... statement:

The **if...else if...** statement is the one level advance form of control statement that allows JavaScript to make correct decision out of several conditions.

### Syntax:

```
if (expression 1){  
    Statement(s) to be executed if expression 1 is true  
}else if (expression 2){  
    Statement(s) to be executed if expression 2 is true  
}else if (expression 3){  
    Statement(s) to be executed if expression 3 is true  
}else{  
    Statement(s) to be executed if no expression is true  
}
```

There is nothing special about this code. It is just a series of *if* statements, where each *if* is part of the *else* clause of the previous statement. Statement(s) are executed based on the true condition, if none of the condition is true then *else* block is executed.

### Example:

Naresh i Technologies, Opp. Satyam Theatre, Ameerpet, Hyderabad, Ph: 040-23746666, 23734842  
An ISO 9001 : 2000 Certified Company

```
<script type="text/javascript">
<!--
var book = "maths";
if( book == "history" ){
    document.write("<b>History Book</b>");
}else if( book == "maths" ){
    document.write("<b>Maths Book</b>");
}else if( book == "economics" ){
    document.write("<b>Economics Book</b>");
}else{
    document.write("<b>Unknown Book</b>");
}
//-->
</script>
```

This will produce following result:

**Maths Book**

## JavaScript Switch Case

You can use multiple *if...else if* statements, as in the previous chapter, to perform a multiway branch. However, this is not always the best solution, especially when all of the branches depend on the value of a single variable.

Starting with JavaScript 1.2, you can use a **switch** statement which handles exactly this situation, and it does so more efficiently than repeated *if...else if* statements.

### Syntax:

The basic syntax of the **switch** statement is to give an expression to evaluate and several different statements to execute based on the value of the expression. The interpreter checks each **case** against the value of the expression until a match is found. If nothing matches, a **default** condition will be used.

```
switch (expression)
{
    case condition 1: statement(s)
                      break;
    case condition 2: statement(s)
                      break;
    ...
    case condition n: statement(s)
                      break;
    default: statement(s)
}
```

The **break** statements indicate to the interpreter the end of that particular case. If they were omitted, the interpreter would continue executing each statement in each of the following cases.

We will explain **break** statement in *Loop Control* chapter.

### Example:

Following example illustrates a basic while loop:

```
<script type="text/javascript">
<!--
var grade='A';
document.write("Entering switch block<br />");
switch (grade)
{
    case 'A': document.write("Good job<br />");
                break;
    case 'B': document.write("Pretty good<br />");
                break;
    case 'C': document.write("Passed<br />");
                break;
    case 'D': document.write("Not so good<br />");
                break;
    case 'F': document.write("Failed<br />");
                break;
    default:  document.write("Unknown grade<br />")
}
document.write("Exiting switch block");
//-->
</script>
```

## Example:

Consider a case if you do not use **break** statement:

```
<script type="text/javascript">
<!--
var grade='A';
document.write("Entering switch block<br />");
switch (grade)
{
    case 'A': document.write("Good job<br />");
    case 'B': document.write("Pretty good<br />");
    case 'C': document.write("Passed<br />");
    case 'D': document.write("Not so good<br />");
    case 'F': document.write("Failed<br />");
    default:  document.write("Unknown grade<br />")
}
document.write("Exiting switch block");
//-->
</script>
```

## JavaScript while Loops

While writing a program, there may be a situation when you need to perform some action over and over again. In such situation you would need to write loop statements to reduce the number of lines.

JavaScript supports all the necessary loops to help you on all steps of programming.

### The **while** Loop

The most basic loop in JavaScript is the **while** loop which would be discussed in this tutorial.

### Syntax:

```
while (expression){  
    Statement(s) to be executed if expression is true  
}
```

The purpose of a **while** loop is to execute a statement or code block repeatedly as long as *expression* is true. Once expression becomes *false*, the loop will be exited.

### Example:

Following example illustrates a basic while loop:

```
<script type="text/javascript">  
<!--  
var count = 0;  
document.write("Starting Loop" + "<br />");  
while (count < 10){  
    document.write("Current Count : " + count + "<br />");  
    count++;  
}  
document.write("Loop stopped!");  
//-->  
</script>
```

### The **do...while** Loop:

The **do...while** loop is similar to the **while** loop except that the condition check happens at the end of the loop. This means that the loop will always be executed at least once, even if the condition is *false*.

### Syntax:

```
do{  
    Statement(s) to be executed;  
} while (expression);
```

Note the semicolon used at the end of the **do...while** loop.

### Example:

Let us write above example in terms of **do...while** loop.

```
<script type="text/javascript">  
<!--  
var count = 0;  
document.write("Starting Loop" + "<br />");  
do{  
    document.write("Current Count : " + count + "<br />");  
    count++;  
}while (count < 0);  
document.write("Loop stopped!");  
//-->  
</script>
```

## JavaScript for Loops

We have seen different variants of **while** loop. This chapter will explain another popular loop called **for** loop.

### The **for** Loop

The **for** loop is the most compact form of looping and includes the following three important parts:

- The loop initialization where we initialize our counter to a starting value. The initialization statement is executed before the loop begins.
- The test statement which will test if the given condition is true or not. If condition is true then code given inside the loop will be executed otherwise loop will come out.
- The iteration statement where you can increase or decrease your counter.

You can put all the three parts in a single line separated by a semicolon.

#### Syntax:

```
for (initialization; test condition; iteration statement){  
    Statement(s) to be executed if test condition is true  
}
```

#### Example:

Following example illustrates a basic for loop:

```
<script type="text/javascript">  
<!--  
var count;  
document.write("Starting Loop" + "<br />");  
for(count = 0; count < 10; count++){  
    document.write("Current Count : " + count );  
    document.write("<br />");  
}  
document.write("Loop stopped!");  
//-->  
</script>
```

## JavaScript **for...in** loop

There is one more loop supported by JavaScript. It is called **for...in** loop. This loop is used to loop through an object's properties.

Because we have not discussed Objects yet, so you may not feel comfortable with this loop. But once you will have understanding on JavaScript objects then you will find this loop very useful.

#### Syntax:

```
for (variablename in object){  
    statement or block to execute  
}
```

In each iteration one property from *object* is assigned to *variablename* and this loop continues till all the properties of the object are exhausted.

### Example:

Here is the following example that prints out the properties of a Web browser's **Navigator** object:

```
<script type="text/javascript">
<!--
var aProperty;
document.write("Navigator Object Properties<br /> ");
for (aProperty in navigator)
{
  document.write(aProperty);
  document.write("<br />");
}
document.write("Exiting from the loop!");
//-->
</script>
```

## JavaScript Loop Control

JavaScript provides you full control to handle your loops and switch statement. There may be a situation when you need to come out of a loop without reaching at its bottom. There may also be a situation when you want to skip a part of your code block and want to start next iteration of the look.

To handle all such situations, JavaScript provides **break** and **continue** statements. These statements are used to immediately come out of any loop or to start the next iteration of any loop respectively.

### The **break** Statement:

The **break** statement, which was briefly introduced with the **switch** statement, is used to exit a loop early, breaking out of the enclosing curly braces.

### Example:

This example illustrates the use of a **break** statement with a while loop. Notice how the loop breaks out early once *x* reaches 5 and reaches to *document.write(..)* statement just below to closing curly brace:

```
<script type="text/javascript">
<!--
var x = 1;
document.write("Entering the loop<br /> ");
while (x < 20)
{
  if (x == 5){
    break; // breaks out of loop completely
  }
  x = x + 1;
  document.write( x + "<br /> ");
}
```

```
document.write("Exiting the loop!<br /> ");
//-->
</script>
```

## The **continue** Statement:

The **continue** statement tells the interpreter to immediately start the next iteration of the loop and skip remaining code block.

When a **continue** statement is encountered, program flow will move to the loop check expression immediately and if condition remain true then it start next iteration otherwise control comes out of the loop.

### Example:

This example illustrates the use of a **continue** statement with a while loop. Notice how the **continue** statement is used to skip printing when the index held in variable x reaches 5:

```
<script type="text/javascript">
<!--
var x = 1;
document.write("Entering the loop<br /> ");
while (x < 10)
{
    x = x + 1;
    if (x == 5){
        continue; // skip rest of the loop body
    }
    document.write( x + "<br />");
}
document.write("Exiting the loop!<br /> ");
//-->
</script>
```

## Using Labels to Control the Flow:

Starting from JavaScript 1.2, a label can be used with **break** and **continue** to control the flow more precisely.

A **label** is simply an identifier followed by a colon that is applied to a statement or block of code. We will see two different examples to understand label with break and continue.

**Note:** Line breaks are not allowed between the *continue* or **break** statement and its label name. Also, there should not be any other statement in between a label name and associated loop.

### Example 1:

```
<script type="text/javascript">
<!--
document.write("Entering the loop!<br /> ");
outerloop: // This is the label name
for (var i = 0; i < 5; i++)
{
```

```

document.write("Outerloop: " + i + "<br />");

innerloop:
for (var j = 0; j < 5; j++)
{
    if (j > 3) break;           // Quit the innermost loop
    if (i == 2) break innerloop; // Do the same thing
    if (i == 4) break outerloop; // Quit the outer loop
    document.write("Innerloop: " + j + " <br />");
}
document.write("Exiting the loop!<br /> ");
//-->
</script>

```

**Example 2:**

```

<script type="text/javascript">
<!--
document.write("Entering the loop!<br /> ");
outerloop: // This is the label name
for (var i = 0; i < 3; i++)
{
    document.write("Outerloop: " + i + "<br /> ");
    for (var j = 0; j < 5; j++)
    {
        if (j == 3){
            continue outerloop;
        }
        document.write("Innerloop: " + j + "<br /> ");
    }
}
document.write("Exiting the loop!<br /> ");
//-->
</script>

```

**JavaScript - Dialog Boxes**

JavaScript has three kind of popup boxes: Alert box, Confirm box, and Prompt box.

**Alert Box**

An alert box is often used if you want to make sure information comes through to the user.

When an alert box pops up, the user will have to click "OK" to proceed.

**Syntax**

```
alert("sometext");
```

**Example**

```

<html>
<head>
<script type="text/javascript">
function show_alert()

```

```
{  
alert("I am an alert box!");  
}  
</script>  
</head>  
<body>  
  
<input type="button" onclick="show_alert()" value="Show  
alert box" />  
  
</body>  
</html>
```

## Confirm Box

A confirm box is often used if you want the user to verify or accept something.

When a confirm box pops up, the user will have to click either "OK" or "Cancel" to proceed.

If the user clicks "OK", the box returns true. If the user clicks "Cancel", the box returns false.

### Syntax

```
confirm("sometext");
```

### Example

```
<html>  
<head>  
<script type="text/javascript">  
function show_confirm()  
{  
var r=confirm("Press a button");  
if (r==true)  
{  
    alert("You pressed OK!");  
}  
else  
{  
    alert("You pressed Cancel!");  
}  
}  
</script>  
</head>  
<body>  
  
<input type="button" onclick="show_confirm()" value="Show  
confirm box" />
```

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

## Prompt Box

A prompt box is often used if you want the user to input a value before entering a page.

When a prompt box pops up, the user will have to click either "OK" or "Cancel" to proceed after entering an input value.

If the user clicks "OK" the box returns the input value. If the user clicks "Cancel" the box returns null.

### Syntax

```
prompt("sometext","defaultvalue");
```

## Example

```
<html>
<head>
<script type="text/javascript">
function show_prompt()
{
var name=prompt("Please enter your name","Harry
Potter");
if (name!=null && name!="")
{
document.write("Hello " + name + "! How are you
today?");
}
</script>
</head>
<body>

<input type="button" onclick="show_prompt()"
value="Show prompt box" />

</body>
</html>
```

## JavaScript Functions

A function is a group of reusable code which can be called anywhere in your programme. This eliminates the need of writing same code again and again. This will help programmers to write modular code. You can divide your big programme in a number of small and manageable functions.

Like any other advance programming language, JavaScript also supports all the features necessary to write modular code using functions.

You must have seen functions like `alert()` and `write()` in previous chapters. We are using these function again and again but they have been written in core JavaScript only once.

JavaScript allows us to write our own functions as well. This section will explain you how to write your own functions in JavaScript.

### Function Definition:

Before we use a function we need to define that function. The most common way to define a function in JavaScript is by using the function keyword, followed by a unique function name, a list of parameters (that might be empty), and a statement block surrounded by curly braces. The basic syntax is shown here:

```
<script type="text/javascript">
<!--
function functionname(parameter-list)
{
    statements
}
//-->
</script>
```

### Example:

A simple function that takes no parameters called `sayHello` is defined here:

```
<script type="text/javascript">
<!--
function sayHello()
{
    alert("Hello there");
}
//-->
</script>
```

### Calling a Function:

To invoke a function somewhere later in the script, you would simple need to write the name of that function as follows:

```
<script type="text/javascript">
<!--
sayHello();
//-->
</script>
```

## Function Parameters:

Till now we have seen function without a parameters. But there is a facility to pass different parameters while calling a function. These passed parameters can be captured inside the function and any manipulation can be done over those parameters.

A function can take multiple parameters separated by comma.

### Example:

Let us do a bit modification in our *sayHello* function. This time it will take two parameters:

```
<script type="text/javascript">
<!--
function sayHello(name, age)
{
    alert( name + " is " + age + " years old.");
}
//-->
</script>
```

**Note:** We are using + operator to concatenate string and number all together. JavaScript does not mind in adding numbers into strings.

Now we can call this function as follows:

```
<script type="text/javascript">
<!--
sayHello('Zara', 7 );
//-->
</script>
```

## The *return* Statement:

A JavaScript function can have an optional *return* statement. This is required if you want to return a value from a function. This statement should be the last statement in a function.

For example you can pass two numbers in a function and then you can expect from the function to return their multiplication in your calling program.

### Example:

This function takes two parameters and concatenates them and return resultant in the calling program:

```
<script type="text/javascript">
<!--
function concatenate(first, last)
{
    var full;
```

```
full = first + last;
return full;
}
//-->
</script>
```

Now we can call this function as follows:

```
<script type="text/javascript">
<!--
var result;
result = concatenate('Zara', 'Ali');
alert(result );
//-->
</script>
```

## JavaScript Special Characters

In JavaScript you can add special characters to a text string by using the backslash sign.

### Insert Special Characters

The backslash (\) is used to insert apostrophes, new lines, quotes, and other special characters into a text string.

Look at the following JavaScript code:

```
var txt="We are the so-called "Vikings" from the north.";
document.write(txt);
```

In JavaScript, a string is started and stopped with either single or double quotes. This means that the string above will be chopped to: We are the so-called

To solve this problem, you must place a backslash (\) before each double quote in "Viking". This turns each double quote into a string literal:

```
var txt="We are the so-called \"Vikings\" from the north.";
document.write(txt);
```

JavaScript will now output the proper text string: We are the so-called "Vikings" from the north.

The table below lists other special characters that can be added to a text string with the backslash sign:

Code	Outputs
'	single quote

\"	double quote
\\\	Backslash
\n	new line
\r	carriage return
\t	Tab
\b	Backspace
\f	form feed

## JavaScript Events

### What is an Event ?

JavaScript's interaction with HTML is handled through events that occur when the user or browser manipulates a page.

When the page loads, that is an event. When the user clicks a button, that click, too, is an event. Another example of events are like pressing any key, closing window, resizing window etc.

Developers can use these events to execute JavaScript coded responses, which cause buttons to close windows, messages to be displayed to users, data to be validated, and virtually any other type of response imaginable to occur.

Events are a part of the Document Object Model (DOM) Level 3 and every HTML element have a certain set of events which can trigger JavaScript Code.

### onclick Event Type:

This is the most frequently used event type which occurs when a user clicks mouse left button. You can put your validation, warning etc against this event type.

#### Example:

```
<html>
<head>
<script type="text/javascript">
<!--
function sayHello() {
    alert("Hello World")
}
//-->
</script>
</head>
<body>
<input type="button" onclick="sayHello()" value="Say Hello" />
```

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

### **onsubmit event type:**

Another most important event type is onsubmit. This event occurs when you try to submit a form. So you can put your form validation against this event type.

Here is simple example showing its usage. Here we are calling a validate() function before submitting a form data to the webserver. If validate() function returns true the form will be submitted otherwise it will not submit the data.

#### **Example:**

```
<html>
<head>
<script type="text/javascript">
<!--
function validation() {
    all validation goes here
    .....
    return either true or false
}
//-->
</script>
</head>
<body>
<form method="POST" action="t.cgi" onsubmit="return validate()">
    .....
<input type="submit" value="Submit" />
</form>
</body>
</html>
```

### **onmouseover and onmouseout:**

These two event types will help you to create nice effects with images or even with text as well. The onmouseover event occurs when you bring your mouse over any element and the onmouseout occurs when you take your mouse out from that element.

#### **Example:**

Following example shows how a division reacts when we bring our mouse in that division:

```
<html>
<head>
<script type="text/javascript">
<!--
function over() {
    alert("Mouse Over");
}
function out() {
    alert("Mouse Out");
}
```

```
//-->
</script>
</head>
<body>
<div onmouseover="over()" onmouseout="out()">
<h2> This is inside the division </h2>
</div>
</body>
</html>
```

You can change different images using these two event types or you can create help balloon to help your users.

## HTML 4 Standard Events

The standard HTML 4 events are listed here for your reference. Here script indicates a Javascript function to be executed agains that event.

Event	Value	Description
onchange	script	Script runs when the element changes
onsubmit	script	Script runs when the form is submitted
onreset	script	Script runs when the form is reset
onselect	script	Script runs when the element is selected
onblur	script	Script runs when the element loses focus
onfocus	script	Script runs when the element gets focus
onkeydown	script	Script runs when key is pressed
onkeypress	script	Script runs when key is pressed and released
onkeyup	script	Script runs when key is released
onclick	script	Script runs when a mouse click
ondblclick	script	Script runs when a mouse double-click
onmousedown	script	Script runs when mouse button is pressed
onmousemove	script	Script runs when mouse pointer moves

onmouseout	script	Script runs when mouse pointer moves out of an element
onmouseover	script	Script runs when mouse pointer moves over an element
onmouseup	script	Script runs when mouse button is released

## onblur Event

```
<html>
<head>
<script type="text/javascript">
function upperCase()
{
var x=document.getElementById("fname").value
document.getElementById("fname").value=x.toUpperCase()
}
</script>
</head>
<body>
```

Enter your name: <input type="text" id="fname" onblur="upperCase()">  
</body>  
</html>

## onchange Event

```
<html>
<head>
<script type="text/javascript">
function upperCase(x)
{
var y=document.getElementById(x).value
document.getElementById(x).value=y.toUpperCase()
}
</script>
</head>
<body>
```

Enter your name:  
<input type="text" id="fname"  
onchange="upperCase(this.id)">  
</body>  
</html>

## onclick Event

```
<html>
<body>
```

```
Field1: <input type="text" id="field1" value="Hello World!">
<br />
Field2: <input type="text" id="field2">
<br /><br />
Click the button below to copy the content of Field1 to Field2.
<br />
<button onclick="document.getElementById('field2').value=
document.getElementById('field1').value">Copy Text</button>

</body>
</html>
```

## ondblclick Event

```
<html>
<body>

Field1: <input type="text" id="field1" value="Hello World!">
<br />
Field2: <input type="text" id="field2">
<br /><br />
Click the button below to copy the content of Field1 to Field2.
<br />
<button ondblclick="document.getElementById('field2').value=
document.getElementById('field1').value">Copy Text</button>

</body>
</html>
```

## onerror Event

```

```

## onfocus Event

```
<html>
<head>
<script type="text/javascript">
function setStyle(x)
{
document.getElementById(x).style.background="yellow"
}
</script>
</head>
<body>
```

```
First name: <input type="text"
onfocus="setStyle(this.id)" id="fname">
<br />
Last name: <input type="text"
onfocus="setStyle(this.id)" id="lname">

</body>
</html>
```

## onkeydown Event

```
<html>
<body>
<script type="text/javascript">
function noNumbers(e)
{
var keynum
var keychar
var numcheck

if(window.event) // IE
{
keynum = e.keyCode
}
else if(e.which) // Netscape/Firefox/Opera
{
keynum = e.which
}
keychar = String.fromCharCode(keynum)
numcheck = /\d/
return !numcheck.test(keychar)
}
</script>

<form>
<input type="text" onkeydown="return noNumbers(event)" />
</form>

</body>
</html>
```

## onkeypress Event

```
<html>
<body>
<script type="text/javascript">
function noNumbers(e)
{
var keynum
var keychar
var numcheck

if(window.event) // IE
{
keynum = e.keyCode
}
else if(e.which) // Netscape/Firefox/Opera
{
keynum = e.which
}
keychar = String.fromCharCode(keynum)
numcheck = /\d/
return !numcheck.test(keychar)
}
</script>

<form>
```

```
<input type="text" onkeypress="return noNumbers(event)" />
</form>
</body>
</html>
```

## onKeyUp Event

```
<html>
<head>
<script type="text/javascript">
function upperCase(x)
{
var y=document.getElementById(x).value
document.getElementById(x).value=y.toUpperCase()
}
</script>
</head>
<body>
```

```
Enter your name: <input type="text"
id="fname" onkeyup="upperCase(this.id)">
</body> </html>
```

## onload Event

```
<html>
<head>
<script type="text/javascript">
function load()
{
alert("Page is loaded");
}
</script>
</head>

<body onload="load()">
<h1>Hello World!</h1>
</body>
</html>
```

## onmousedown Event

```
<html>
<head>
<script type="text/javascript">
function whichElement(e)
{
var targ
if (!e) var e = window.event
if (e.target) targ = e.target
else if (e.srcElement) targ = e.srcElement
if (targ.nodeType == 3) // defeat Safari bug
targ = targ.parentNode
```

```

var tname
tname=targ.tagName
alert("You clicked on a " + tname + " element.")
}
</script>
</head>

<body onmousedown="whichElement(event)">

<h2>This is a header</h2>
<p>This is a paragraph</p>


</body>
</html>

```

## onmousemove Event

```



```

## onmouseout Event

```

<html>
<head>
<script type="text/javascript">
function mouseOver()
{
document.getElementById("b1").src="b_blue.gif"
}
function mouseOut()
{
document.getElementById("b1").src="b_pink.gif"
}
</script>
</head>

<body>
<a href="http://www.w3schools.com" target="_blank" onmouseover="mouseOver()"
onmouseout="mouseOut()">
</a>
</body>
</html>

```

## onmouseover Event

```

<html>
<head>
<script type="text/javascript">
function mouseOver()
{
document.getElementById("b1").src ="b_blue.gif"
}
function mouseOut()
{
document.getElementById("b1").src ="b_pink.gif"

```

```

}
</script>
</head>

<body>
<a href="http://www.w3schools.com" target="_blank" onmouseover="mouseOver()"
onmouseout="mouseOut()">
</a>
</body>
</html>

```

## onmouseup Event

```

<html>
<head>
<script type="text/javascript">
function whichElement(e)
{
var targ
if (!e) var e = window.event
if (e.target) targ = e.target
else if (e.srcElement) targ = e.srcElement
if (targ.nodeType == 3) // defeat Safari bug
targ = targ.parentNode
var tname
tname=targ.tagName
alert("You clicked on a " + tname + " element.")
}
</script>
</head>

<body onmouseup="whichElement(event)">
<h2>This is a header</h2>
<p>This is a paragraph</p>

</body>
</html>

```

## onresize Event

```

<body onresize="alert('You have changed the size of the window')">
</body>

```

## onselect Event

```

<form>
Select text: <input type="text" value="Hello world!">
onselect="alert('You have selected some of the text.')">
</form>

```

## onunload Event

```

<body onunload="alert('The onunload event was triggered')">
</body>

```

## Javascript - Page Printing

Many times you would like to give a button at your webpage to print out the content of that web page via an actual printer.

JavaScript helps you to implement this functionality using **print** function of *window* object.

The JavaScript print function **window.print()** will print the current web page when executed. You can call this function directly using *onclick* event as follows:

```
<head>
<script type="text/javascript">
<!--
//-->
</script>
</head>
<body>
<form>
<input type="button" value="Print" onclick="window.print()" />
</form>
</body>
```

## How to print a page:

If someone is providing none of the above facilities then you can use browser's standard toolbar to get web pages printed out. Follow the link as follows:

File --> Print --> Click OK button.

## JavaScript - Page Redirection

When you click a URL to reach to a page X but internally you are directed to another page Y that simply happens because of page re-direction.

```
<head>
<script type="text/javascript">
    function fun1(){
        window.location="http://www.newlocation.com";
    }
</script>
</head>
```

## JavaScript and Cookies

### What are Cookies ?

Web Browser and Server use HTTP protocol to communicate and HTTP is a stateless protocol. But for a commercial website it is required to maintain session information among different pages. For example one user registration ends after completing many pages. But how to maintain user's session information across all the web pages.

In many situations, using cookies is the most efficient method of remembering and tracking preferences, purchases, commissions, and other information required for better visitor experience or site statistics.

### How It Works ?

Your server sends some data to the visitor's browser in the form of a cookie. The browser may accept the cookie. If it does, it is stored as a plain text record on the visitor's hard drive. Now, when the visitor arrives at another page on your site, the browser sends the same cookie to the server for retrieval. Once retrieved, your server knows/remembers what was stored earlier.

Cookies are a plain text data record of 5 variable-length fields:

- **Expires :** The date the cookie will expire. If this is blank, the cookie will expire when the visitor quits the browser.
- **Domain :** The domain name of your site.
- **Path :** The path to the directory or web page that set the cookie. This may be blank if you want to retrieve the cookie from any directory or page.
- **Secure :** If this field contains the word "secure" then the cookie may only be retrieved with a secure server. If this field is blank, no such restriction exists.
- **Name=Value :** Cookies are set and retrieved in the form of key and value pairs.

Cookies were originally designed for CGI programming and cookies' data is automatically transmitted between the web browser and web server, so CGI scripts on the server can read and write cookie values that are stored on the client.

JavaScript can also manipulate cookies using the *cookie* property of the *Document* object. JavaScript can read, create, modify, and delete the cookie or cookies that apply to the current web page.

### Storing Cookies:

The simplest way to create a cookie is to assign a string value to the *document.cookie* object, which looks like this:

#### Syntax:

```
document.cookie = "key1=value1;key2=value2;expires=date";
```

Here *expires* attribute is option. If you provide this attribute with a valid date or time then cookie will expire at the given date or time and after that cookies' value will not be accessible.

**Note:** Cookie values may not include semicolons, commas, or whitespace. For this reason, you may want to use the JavaScript *escape()* function to encode the *value* before storing it in the cookie. If you do this, you will also have to use the corresponding *unescape()* function when you read the cookie value.

**Example:**

Following is the example to set a customer name in *input* cookie.

```
<html>
<head>
<script type="text/javascript">
<!--
function WriteCookie()
{
    if( document.myform.customer.value == "" ){
        alert("Enter some value!");
        return;
    }

    cookievalue= escape(document.myform.customer.value) + ";";
    document.cookie="name=" + cookievalue;
    alert("Setting Cookies : " + "name=" + cookievalue );
}
//-->
</script>
</head>
<body>
<form name="myform" action="">
Enter name: <input type="text" name="customer"/>
<input type="button" value="Set Cookie" onclick="WriteCookie();"/>
</form>
</body>
</html>
```

Now your machine has a cookie called *name*. You can set multiple cookies using multiple *key=value* pairs separated by comma.

You will learn how to read this cookie in next section.

**Reading Cookies:**

Reading a cookie is just as simple as writing one, because the value of the *document.cookie* object is the cookie. So you can use this string whenever you want to access the cookie.

The *document.cookie* string will keep a list of *name=value* pairs separated by semicolons, where *name* is the *name* of a cookie and value is its string value.

You can use strings' *split()* function to break the string into key and values as follows:

**Example:**

Following is the example to get the cookies set in previous section.

```
<html>
<head>
<script type="text/javascript">
<!--
```

```

function ReadCookie()
{
    var allcookies = document.cookie;
    alert("All Cookies : " + allcookies );

    // Get all the cookies pairs in an array
    cookiearray = allcookies.split(';');

    // Now take key value pair out of this array
    for(var i=0; i<cookiearray.length; i++){
        name = cookiearray[i].split('=')[0];
        value = cookiearray[i].split('=')[1];
        alert("Key is : " + name + " and Value is : " + value);
    }
}
//-->
</script>
</head>
<body>
<form name="myform" action="">
<input type="button" value="Get Cookie" onclick="ReadCookie()"/>
</form>
</body>
</html>

```

**Note:** Here *length* is a method of *Array* class which returns the length of an array.

**Note:** There may be some other cookies already set on your machine. So above code will show you all the cookies set at your machine.

## Setting the Cookies Expiration Date:

You can extend the life of a cookie beyond the current browser session by setting an expiration date and saving the expiration date within the cookie. This can be done by setting the *expires* attribute to a date and time.

### Example:

The following example illustrates how to set cookie expiration date after 1 Month :

```

<html>
<head>
<script type="text/javascript">
<!--
function WriteCookie()
{
    var now = new Date();
    now.setMonth( now.getMonth() + 1 );
    cookievalue = escape(document.myform.customer.value) + ";";
    document.cookie="name=" + cookievalue;
    document.cookie = "expires=" + now.getGMTString() + ";"
    alert("Setting Cookies : " + "name=" + cookievalue );
}
//-->
</script>
</head>

```

```
<body>
<form name="formname" action=""
Enter name: <input type="text" name="customer"/>
<input type="button" value="Set Cookie" onclick="WriteCookie()"/>
</form>
</body>
</html>
```

## Deleting a Cookie:

Sometimes you will want to delete a cookie so that subsequent attempts to read the cookie return nothing. To do this, you just need to set the expiration date to a time in the past.

### Example:

The following example illustrates how to delete cookie by setting expiration date one Month in past :

```
<html>
<head>
<script type="text/javascript">
<!--
function WriteCookie()
{
    var now = new Date();
    now.setMonth( now.getMonth() - 1 );
    cookievalue = escape(document.myform.customer.value) + ";"
    document.cookie="name=" + cookievalue;
    document.cookie = "expires=" + now.getGMTString() + ";"
    alert("Setting Cookies : " + "name=" + cookievalue );
}
//-->
</script>
</head>
<body>
<form name="formname" action=""
Enter name: <input type="text" name="customer"/>
<input type="button" value="Set Cookie" onclick="WriteCookie()"/>
</form>
</body>
</html>
```

**Note:** Instead of setting date, you can see new time using *setTime()* function.

## Javascript Objects Overview

JavaScript is an Object Oriented Programming (OOP) language. A programming language can be called object-oriented if it provides four basic capabilities to developers:

- **Encapsulation** . the capability to store related information, whether data or methods, together in an object

- **Aggregation** . the capability to store one object inside of another object
- **Inheritance** . the capability of a class to rely upon another class (or number of classes) for some of its properties and methods
- **Polymorphism** . the capability to write one function or method that works in a variety of different ways

Objects are composed of attributes. If an attribute contains a function, it is considered to be a method of the object otherwise, the attribute is considered a property.

## Object Properties:

Object properties can be any of the three primitive data types, or any of the abstract data types, such as another object. Object properties are usually variables that are used internally in the object's methods, but can also be globally visible variables that are used throughout the page.

The syntax for adding a property to an object is:

```
objectName.objectProperty = propertyName;
```

### Example:

Following is a simple example to show how to get a document title using "title" property of document object:

```
var str = document.title;
```

## Object Methods:

The methods are functions that let the object do something or let something be done to it. There is little difference between a function and a method, except that a function is a standalone unit of statements and a method is attached to an object and can be referenced by the **this** keyword.

Methods are useful for everything from displaying the contents of the object to the screen to performing complex mathematical operations on a group of local properties and parameters.

### Example:

Following is a simple example to show how to use **write()** method of document object to write any content on the document:

```
document.write("This is test");
```

## User-Defined Objects:

All user-defined objects and built-in objects are descendants of an object called Object.

## The *new* Operator:

The *new* operator is used to create an instance of an object. To create an object, the *new* operator is followed by the constructor method.

In the following example, the constructor methods are Object(), Array(), and Date(). These constructors are built-in JavaScript functions.

```
var employee = new Object();
var books = new Array("C++", "Perl", "Java");
var day = new Date("August 15, 1947");
```

## The *Object()* Constructor:

A constructor is a function that creates and initializes an object. JavaScript provides a special constructor function called *Object()* to build the object. The return value of the Object() constructor is assigned to a variable.

The variable contains a reference to the new object. The properties assigned to the object are not variables and are not defined with the var keyword.

### Example 1:

This example demonstrates how to create an object:

```
<html>
<head>
<title>User-defined objects</title>
<script type="text/javascript">
var book = new Object(); // Create the object
    book.subject = "Perl"; // Assign properties to the object
    book.author  = "Mohtashim";
</script>
</head>
<body>
<script type="text/javascript">
    document.write("Book name is : " + book.subject + "<br>");
    document.write("Book author is : " + book.author + "<br>");
</script>
</body>
</html>
```

**Example 2:**

This example demonstrates how to create an object with a User-Defined Function. Here *this* keyword is used to refer to the object that has been passed to a function:

```
<html>
<head>
<title>User-defined objects</title>
<script type="text/javascript">
function book(title, author){
    this.title = title;
    this.author = author;
}
</script>
</head>
<body>
<script type="text/javascript">
    var myBook = new book("Perl", "Mohtashim");
    document.write("Book title is : " + myBook.title + "<br>");
    document.write("Book author is : " + myBook.author + "<br>");
</script>
</body>
</html>
```

**Defining Methods for an Object:**

The previous examples demonstrate how the constructor creates the object and assigns properties. But we need to complete the definition of an object by assigning methods to it.

**Example:**

Here is a simple example to show how to add a function along with an object:

```
<html>
<head>
<title>User-defined objects</title>
<script type="text/javascript">

// Define a function which will work as a method
function addPrice(amount){
    this.price = amount;
}

function book(title, author){
    this.title = title;
    this.author = author;
    this.addPrice = addPrice; // Assign that method as property.
}

</script>
</head>
```

```
<body>
<script type="text/javascript">
    var myBook = new book("Perl", "Mohtashim");
    myBook.addPrice(100);
    document.write("Book title is : " + myBook.title + "<br>");
    document.write("Book author is : " + myBook.author + "<br>");
    document.write("Book price is : " + myBook.price + "<br>");
</script>
</body>
</html>
```

## The **with** Keyword:

The **with** keyword is used as a kind of shorthand for referencing an object's properties or methods.

The object specified as an argument to **with** becomes the default object for the duration of the block that follows. The properties and methods for the object can be used without naming the object.

### Syntax:

```
with (object){
    properties used without the object name and dot
}
```

### Example:

```
<html>
<head>
<title>User-defined objects</title>
<script type="text/javascript">

// Define a function which will work as a method
function addPrice(amount) {
    with(this) {
        price = amount;
    }
}
function book(title, author) {
    this.title = title;
    this.author = author;
    this.price = 0;
    this.addPrice = addPrice; // Assign that method as property.
}
</script>
</head>
<body>
<script type="text/javascript">
    var myBook = new book("Perl", "Mohtashim");
    myBook.addPrice(100);
    document.write("Book title is : " + myBook.title + "<br>");
    document.write("Book author is : " + myBook.author + "<br>");
    document.write("Book price is : " + myBook.price + "<br>");
</script>

```

```
</script>
</body>
</html>
```

## JavaScript Native Objects:

JavaScript has several built-in or native objects. These objects are accessible anywhere in your program and will work the same way in any browser running in any operating system.

Here is the list of all important JavaScript Native Objects:

- JavaScript Number Object
- JavaScript Boolean Object
- JavaScript String Object
- JavaScript Array Object
- JavaScript Date Object
- JavaScript Math Object
- JavaScript RegExp Object

## Javascript - The String Object

The **String** object let's you work with a series of characters and wraps Javascript's string primitive data type with a number of helper methods.

Because Javascript automatically converts between string primitives and String objects, you can call any of the helper methods of the String object on a string primitive.

### Syntax:

Creating a **String** object:

```
var val = new String(string);
```

The *string* parameter is series of characters that has been properly encoded.

### String Properties:

Here is a list of each property and their description.

Property	Description
----------	-------------

<u>constructor</u>	Returns a reference to the String function that created the object.
<u>length</u>	Returns the length of the string.
<u>prototype</u>	The prototype property allows you to add properties and methods to an object.

## String Methods

Here is a list of each method and its description.

Method	Description
<u>charAt()</u>	Returns the character at the specified index.
<u>charCodeAt()</u>	Returns a number indicating the Unicode value of the character at the given index.
<u>concat()</u>	Combines the text of two strings and returns a new string.
<u>indexOf()</u>	Returns the index within the calling String object of the first occurrence of the specified value, or -1 if not found.
<u>lastIndexOf()</u>	Returns the index within the calling String object of the last occurrence of the specified value, or -1 if not found.
<u>localeCompare()</u>	Returns a number indicating whether a reference string comes before or after or is the same as the given string in sort order.
<u>match()</u>	Used to match a regular expression against a string.
<u>replace()</u>	Used to find a match between a regular expression and a string, and to replace the matched substring with a new substring.
<u>search()</u>	Executes the search for a match between a regular expression and a specified string.
<u>slice()</u>	Extracts a section of a string and returns a new string.
<u>split()</u>	Splits a String object into an array of strings by separating the string into substrings.

<u>substr()</u>	Returns the characters in a string beginning at the specified location through the specified number of characters.
<u>substring()</u>	Returns the characters in a string between two indexes into the string.
<u>toLocaleLowerCase()</u>	The characters within a string are converted to lower case while respecting the current locale.
<u>toLocaleUpperCase()</u>	The characters within a string are converted to upper case while respecting the current locale.
<u>toLowerCase()</u>	Returns the calling string value converted to lower case.
<u>toString()</u>	Returns a string representing the specified object.
<u>toUpperCase()</u>	Returns the calling string value converted to uppercase.
<u>valueOf()</u>	Returns the primitive value of the specified object.

## String HTML wrappers

Here is a list of each method which returns a copy of the string wrapped inside the appropriate HTML tag.

Method	Description
<u>anchor()</u>	Creates an HTML anchor that is used as a hypertext target.
<u>big()</u>	Creates a string to be displayed in a big font as if it were in a <big> tag.
<u>blink()</u>	Creates a string to blink as if it were in a <blink> tag.
<u>bold()</u>	Creates a string to be displayed as bold as if it were in a <b> tag.
<u>fixed()</u>	Causes a string to be displayed in fixed-pitch font as if it were in a <tt> tag
<u>fontcolor()</u>	Causes a string to be displayed in the specified color as if it were in a <font color="color"> tag.
<u>fontsize()</u>	Causes a string to be displayed in the specified font size as if it were in a <font size="size"> tag.

<u>italics()</u>	Causes a string to be italic, as if it were in an <i> tag.
<u>link()</u>	Creates an HTML hypertext link that requests another URL.
<u>small()</u>	Causes a string to be displayed in a small font, as if it were in a <small> tag.
<u>strike()</u>	Causes a string to be displayed as struck-out text, as if it were in a <strike> tag.
<u>sub()</u>	Causes a string to be displayed as a subscript, as if it were in a <sub> tag
<u>sup()</u>	Causes a string to be displayed as a superscript, as if it were in a <sup> tag

## Javascript - The Arrays Object

The **Array** object let's you store multiple values in a single variable.

### Syntax:

Creating a **Array** object:

```
var fruits = new Array( "apple", "orange", "mango" );
```

The *Array* parameter is a list of strings or integers. When you specify a single numeric parameter with the *Array* constructor, you specify the initial length of the array. The maximum length allowed for an array is 4,294,967,295.

You can create array by simply assigning values as follows:

```
var fruits = [ "apple", "orange", "mango" ];
```

You will use ordinal numbers to access and to set values inside an array as follows:

- `fruits[0]` is the first element
- `fruits[1]` is the second element
- `fruits[2]` is the third element

## Array Properties:

Here is a list of each property and their description.

Property	Description
<u>constructor</u>	Returns a reference to the array function that created the object.
<u>index</u>	The property represents the zero-based index of the match in the string
<u>input</u>	This property is only present in arrays created by regular expression matches.
<u>length</u>	Reflects the number of elements in an array.
<u>prototype</u>	The prototype property allows you to add properties and methods to an object.

## Array Methods

Here is a list of each method and its description.

Method	Description
<u>concat()</u>	Returns a new array comprised of this array joined with other array(s) and/or value(s).
<u>every()</u>	Returns true if every element in this array satisfies the provided testing function.
<u>filter()</u>	Creates a new array with all of the elements of this array for which the provided filtering function returns true.
<u>forEach()</u>	Calls a function for each element in the array.
<u>indexOf()</u>	Returns the first (least) index of an element within the array equal to the specified value, or -1 if none is found.
<u>join()</u>	Joins all elements of an array into a string.
<u>lastIndexOf()</u>	Returns the last (greatest) index of an element within the array equal to the specified value, or -1 if none is found.

<u>map()</u>	Creates a new array with the results of calling a provided function on every element in this array.
<u>pop()</u>	Removes the last element from an array and returns that element.
<u>push()</u>	Adds one or more elements to the end of an array and returns the new length of the array.
<u>reduce()</u>	Apply a function simultaneously against two values of the array (from left-to-right) as to reduce it to a single value.
<u>reduceRight()</u>	Apply a function simultaneously against two values of the array (from right-to-left) as to reduce it to a single value.
<u>reverse()</u>	Reverses the order of the elements of an array -- the first becomes the last, and the last becomes the first.
<u>shift()</u>	Removes the first element from an array and returns that element.
<u>slice()</u>	Extracts a section of an array and returns a new array.
<u>some()</u>	Returns true if at least one element in this array satisfies the provided testing function.
<u>toSource()</u>	Represents the source code of an object
<u>sort()</u>	Sorts the elements of an array.
<u>splice()</u>	Adds and/or removes elements from an array.
<u>toString()</u>	Returns a string representing the array and its elements.
<u>unshift()</u>	Adds one or more elements to the front of an array and returns the new length of the array.

## JavaScript - The Date Object

The Date object is a datatype built into the JavaScript language. Date objects are created with the **new Date()** as shown below.

Once a Date object is created, a number of methods allow you to operate on it. Most methods simply allow you to get and set the year, month, day, hour, minute, second, and millisecond fields of the object, using either local time or UTC (universal, or GMT) time.

The ECMAScript standard requires the Date object to be able to represent any date and time, to millisecond precision, within 100 million days before or after 1/1/1970. This is a range of plus or minus 273,785 years, so the JavaScript is able to represent date and time till year 275755.

## Syntax:

Here are different variant of Date() constructor:

```
new Date( )
new Date(milliseconds)
new Date(datestring)
new Date(year,month,date[,hour,minute,second,millisecond ] )
```

**Note:** Parameters in the brackets are always optional

Here is the description of the parameters:

- **No Argument:** With no arguments, the Date( ) constructor creates a Date object set to the current date and time.
- **milliseconds:** When one numeric argument is passed, it is taken as the internal numeric representation of the date in milliseconds, as returned by the getTime( ) method. For example, passing the argument 5000 creates a date that represents five seconds past midnight on 1/1/70.
- **datestring:** When one string argument is passed, it is a string representation of a date, in the format accepted by the Date.parse( ) method.
- **7 arguments:** To use the last form of constructor given above, Here is the description of each argument:
  1. **year:** Integer value representing the year. For compatibility (in order to avoid the Y2K problem), you should always specify the year in full; use 1998, rather than 98.
  2. **month:** Integer value representing the month, beginning with 0 for January to 11 for December.
  3. **date:** Integer value representing the day of the month.
  4. **hour:** Integer value representing the hour of the day (24-hour scale).
  5. **minute:** Integer value representing the minute segment of a time reading.
  6. **second:** Integer value representing the second segment of a time reading.
  7. **millisecond:** Integer value representing the millisecond segment of a time reading.

## Date Properties:

Here is a list of each property and their description.

Property	Description

<u>constructor</u>	Specifies the function that creates an object's prototype.
<u>prototype</u>	The prototype property allows you to add properties and methods to an object.

## Date Methods:

Here is a list of each method and its description.

Method	Description
<u>Date()</u>	Returns today's date and time
<u>getDate()</u>	Returns the day of the month for the specified date according to local time.
<u>getDay()</u>	Returns the day of the week for the specified date according to local time.
<u>getFullYear()</u>	Returns the year of the specified date according to local time.
<u>getHours()</u>	Returns the hour in the specified date according to local time.
<u>getMilliseconds()</u>	Returns the milliseconds in the specified date according to local time.
<u>getMinutes()</u>	Returns the minutes in the specified date according to local time.
<u>getMonth()</u>	Returns the month in the specified date according to local time.
<u>getSeconds()</u>	Returns the seconds in the specified date according to local time.
<u>getTime()</u>	Returns the numeric value of the specified date as the number of milliseconds since January 1, 1970, 00:00:00 UTC.
<u>getTimezoneOffset()</u>	Returns the time-zone offset in minutes for the current locale.
<u>getUTCDate()</u>	Returns the day (date) of the month in the specified date according to universal time.
<u>getUTCDay()</u>	Returns the day of the week in the specified date according to universal time.

<u>getUTCFullYear()</u>	Returns the year in the specified date according to universal time.
<u>getUTCHours()</u>	Returns the hours in the specified date according to universal time.
<u>getUTCMilliseconds()</u>	Returns the milliseconds in the specified date according to universal time.
<u>getUTCMinutes()</u>	Returns the minutes in the specified date according to universal time.
<u>getUTCMonth()</u>	Returns the month in the specified date according to universal time.
<u>getUTCSeconds()</u>	Returns the seconds in the specified date according to universal time.
<u>getYear()</u>	<b>Deprecated</b> - Returns the year in the specified date according to local time. Use <code>getFullYear</code> instead.
<u> setDate()</u>	Sets the day of the month for a specified date according to local time.
<u> setFullYear()</u>	Sets the full year for a specified date according to local time.
<u> setHours()</u>	Sets the hours for a specified date according to local time.
<u> setMilliseconds()</u>	Sets the milliseconds for a specified date according to local time.
<u> setMinutes()</u>	Sets the minutes for a specified date according to local time.
<u> setMonth()</u>	Sets the month for a specified date according to local time.
<u> setSeconds()</u>	Sets the seconds for a specified date according to local time.
<u> setTime()</u>	Sets the Date object to the time represented by a number of milliseconds since January 1, 1970, 00:00:00 UTC.
<u> setUTCDate()</u>	Sets the day of the month for a specified date according to universal time.
<u> setUTCFullYear()</u>	Sets the full year for a specified date according to universal time.
<u> setUTCHours()</u>	Sets the hour for a specified date according to universal time.

<u>setUTCMilliseconds()</u>	Sets the milliseconds for a specified date according to universal time.
<u>setUTCMilliseconds()</u>	Sets the minutes for a specified date according to universal time.
<u>setUTCMonth()</u>	Sets the month for a specified date according to universal time.
<u>setUTCSeconds()</u>	Sets the seconds for a specified date according to universal time.
<u>setYear()</u>	<b>Deprecated</b> - Sets the year for a specified date according to local time. Use setFullYear instead.
<u>toDateString()</u>	Returns the "date" portion of the Date as a human-readable string.
<u>toGMTString()</u>	<b>Deprecated</b> - Converts a date to a string, using the Internet GMT conventions. Use toUTCString instead.
<u>toLocaleDateString()</u>	Returns the "date" portion of the Date as a string, using the current locale's conventions.
<u>toLocaleFormat()</u>	Converts a date to a string, using a format string.
<u>toLocaleString()</u>	Converts a date to a string, using the current locale's conventions.
<u>toLocaleTimeString()</u>	Returns the "time" portion of the Date as a string, using the current locale's conventions.
<u>toSource()</u>	Returns a string representing the source for an equivalent Date object; you can use this value to create a new object.
<u>toString()</u>	Returns a string representing the specified Date object.
<u>toTimeString()</u>	Returns the "time" portion of the Date as a human-readable string.
<u>toUTCString()</u>	Converts a date to a string, using the universal time convention.
<u>valueOf()</u>	Returns the primitive value of a Date object.

## Date Static Methods:

In addition to the many instance methods listed previously, the Date object also defines two static methods. These methods are invoked through the Date( ) constructor itself:

Method	Description
<u>Date.parse()</u>	Parses a string representation of a date and time and returns the internal millisecond representation of that date.
<u>Date.UTC()</u>	Returns the millisecond representation of the specified UTC date and time.

## Javascript - The Math Object

The **math** object provides you properties and methods for mathematical constants and functions.

Unlike the other global objects, *Math* is not a constructor. All properties and methods of Math are static and can be called by using *Math* as an object without creating it.

Thus, you refer to the constant pi as **Math.PI** and you call the *sine* function as **Math.sin(x)**, where x is the method's argument.

### Syntax:

Here is the simple syntax to call properties and methods of Math.

```
var pi_val = Math.PI;
var sine_val = Math.sin(30);
```

## Math Properties:

Here is a list of each property and their description.

Property	Description
<u>E</u>	Euler's constant and the base of natural logarithms, approximately 2.718.
<u>LN2</u>	Natural logarithm of 2, approximately 0.693.

<u>LN10</u>	Natural logarithm of 10, approximately 2.302.
<u>LOG2E</u>	Base 2 logarithm of E, approximately 1.442.
<u>LOG10E</u>	Base 10 logarithm of E, approximately 0.434.
<u>PI</u>	Ratio of the circumference of a circle to its diameter, approximately 3.14159.
<u>SQRT1_2</u>	Square root of 1/2; equivalently, 1 over the square root of 2, approximately 0.707.
<u>SQRT2</u>	Square root of 2, approximately 1.414.

## Math Methods

Here is a list of each method and its description.

Method	Description
<u>abs()</u>	Returns the absolute value of a number.
<u>acos()</u>	Returns the arccosine (in radians) of a number.
<u>asin()</u>	Returns the arcsine (in radians) of a number.
<u>atan()</u>	Returns the arctangent (in radians) of a number.
<u>atan2()</u>	Returns the arctangent of the quotient of its arguments.
<u>ceil()</u>	Returns the smallest integer greater than or equal to a number.
<u>cos()</u>	Returns the cosine of a number.
<u>exp()</u>	Returns $E^N$ , where N is the argument, and E is Euler's constant, the base of the natural logarithm.
<u>floor()</u>	Returns the largest integer less than or equal to a number.

<u>log()</u>	Returns the natural logarithm (base E) of a number.
<u>max()</u>	Returns the largest of zero or more numbers.
<u>min()</u>	Returns the smallest of zero or more numbers.
<u>pow()</u>	Returns base to the exponent power, that is, base exponent.
<u>random()</u>	Returns a pseudo-random number between 0 and 1.
<u>round()</u>	Returns the value of a number rounded to the nearest integer.
<u>sin()</u>	Returns the sine of a number.
<u>sqrt()</u>	Returns the square root of a number.
<u>tan()</u>	Returns the tangent of a number.
<u>toSource()</u>	Returns the string "Math".

## JavaScript - Document Object Model or DOM

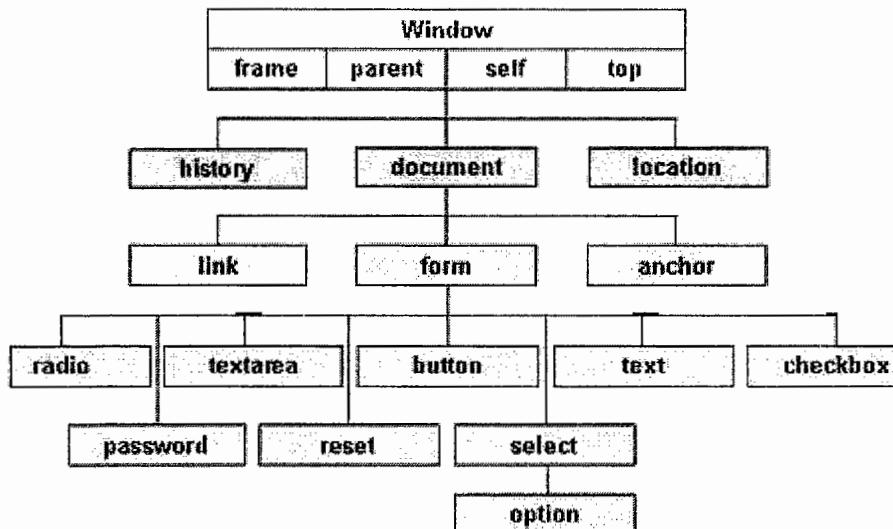
Every web page resides inside a browser window which can be considered as an object.

A Document object represents the HTML document that is displayed in that window. The Document object has various properties that refer to other objects which allow access to and modification of document content.

The way that document content is accessed and modified is called the **Document Object Model, or DOM**. The Objects are organized in a hierarchy. This hierarchical structure applies to the organization of objects in a Web document.

- **Window object:** Top of the hierarchy. It is the outmost element of the object hierarchy.
- **Document object:** Each HTML document that gets loaded into a window becomes a document object. The document contains the content of the page.
- **Form object:** Everything enclosed in the <form>...</form> tags sets the form object.
- **Form control elements:** The form object contains all the elements defined for that object such as text fields, buttons, radio buttons, and checkboxes.

Here is a simple hierarchy of few important objects:



## JavaScript Browser Detection

The Navigator object contains information about the visitor's browser.

### Browser Detection

Almost everything in this tutorial works on all JavaScript-enabled browsers. However, there are some things that just don't work on certain browsers - especially on older browsers.

Sometimes it can be useful to detect the visitor's browser, and then serve the appropriate information.

The Navigator object contains information about the visitor's browser name, version, and more.

**Note:** There is no public standard that applies to the navigator object, but all major browsers support it.

### The Navigator Object

The Navigator object contains all information about the visitor's browser:

#### Example

```

<html>
<body>
<div id="example"></div>

<script type="text/javascript">

txt = "<p>Browser CodeName: " + navigator.appCodeName + "</p>";
txt+= "<p>Browser Name: " + navigator.appName + "</p>";
txt+= "<p>Browser Version: " + navigator.appVersion + "</p>";
txt+= "<p>Cookies Enabled: " + navigator.cookieEnabled + "</p>";
txt+= "<p>Platform: " + navigator.platform + "</p>";
  
```

```
txt+= "<p>User-agent header: " + navigator.userAgent + "</p>";

document.getElementById("example").innerHTML=txt;

</script>

</body>
</html>
```

```
<html>
<head>
<title>Browser Detection Example</title>
</head>
<body>
<script type="text/javascript">
<!--
var userAgent = navigator.userAgent;
var opera = (userAgent.indexOf('Opera') != -1);
var ie = (userAgent.indexOf('MSIE') != -1);
var gecko = (userAgent.indexOf('Gecko') != -1);
var netscape = (userAgent.indexOf('Mozilla') != -1);
var version = navigator.appVersion;

if (opera){
  document.write("Opera based browser");
  // Keep your opera specific URL here.
}else if (gecko){
  document.write("Mozilla based browser");
  // Keep your gecko specific URL here.
}else if (ie){
  document.write("IE based browser");
  // Keep your IE specific URL here.
}else if (netscape){
  document.write("Netscape based browser");
  // Keep your Netscape specific URL here.
}else{
  document.write("Unknown browser");
}
// You can include version to along with any above condition.
document.write("<br /> Browser version info : " + version );
//-->
</script>
</body>
</html>
```

## JavaScript - Errors & Exceptions Handling

There are three types of errors in programming: (a) Syntax Errors and (b) Runtime Errors (c) Logical Errors:

## Syntax errors:

Syntax errors, also called parsing errors, occur at compile time for traditional programming languages and at interpret time for JavaScript.

For example, the following line causes a syntax error because it is missing a closing parenthesis:

```
<script type="text/javascript">
<!--
window.print();
//-->
</script>
```

When a syntax error occurs in JavaScript, only the code contained within the same thread as the syntax error is affected and code in other threads gets executed assuming nothing in them depends on the code containing the error.

## Runtime errors:

Runtime errors, also called exceptions, occur during execution (after compilation/interpretation).

For example, the following line causes a run time error because here syntax is correct but at run time it is trying to call a non existed method:

```
<script type="text/javascript">
<!--
windcw.printme();
//-->
</script>
```

Exceptions also affect the thread in which they occur, allowing other JavaScript threads to continue normal execution.

## Logical errors:

Logic errors can be the most difficult type of errors to track down. These errors are not the result of a syntax or runtime error. Instead, they occur when you make a mistake in the logic that drives your script and you do not get the result you expected.

You can not catch those errors, because it depends on your business requirement what type of logic you want to put in your program.

## The **try...catch...finally** Statement:

The latest versions of JavaScript added exception handling capabilities. JavaScript implements the **try...catch...finally** construct as well as the **throw** operator to handle exceptions.

You can *catch* programmer-generated and *runtime* exceptions, but you cannot *catch* JavaScript syntax errors.

Here is the **try...catch...finally** block syntax:

```
<script type="text/javascript">
<!--
try {
    // Code to run
    [break;]
} catch ( e ) {
    // Code to run if an exception occurs
    [break;]
}[ finally {
    // Code that is always executed regardless of
    // an exception occurring
}]
//-->
</script>
```

The **try** block must be followed by either exactly one **catch** block or one **finally** block (or one of both). When an exception occurs in the **try** block, the exception is placed in **e** and the **catch** block is executed. The optional **finally** block executes unconditionally after try/catch.

### Examples:

Here is one example where we are trying to call a non existing function this is causing an exception raise. Let us see how it behaves without with **try...catch**:

```
<html>
<head>
<script type="text/javascript">
<!--
function myFunc()
{
    var a = 100;

    alert("Value of variable a is : " + a );

}
//-->
</script>
</head>
<body>
<p>Click the following to see the result:</p>
<form>
<input type="button" value="Click Me" onclick="myFunc(); " />

```

```
</form>
</body>
</html>
```

Now let us try to catch this exception using **try...catch** and display a user friendly message. You can also suppress this message, if you want to hide this error from a user.

```
<html>
<head>
<script type="text/javascript">
<!--
function myFunc()
{
    var a = 100;

    try {
        alert("Value of variable a is : " + a );
    } catch ( e ) {
        alert("Error: " + e.description );
    }
}
//-->
</script>
</head>
<body>
<p>Click the following to see the result:</p>
<form>
<input type="button" value="Click Me" onclick="myFunc(); " />
</form>
</body>
</html>
```

You can use **finally** block which will always execute unconditionally after try/catch. Here is an example:

```
<html>
<head>
<script type="text/javascript">
<!--
function myFunc()
{
    var a = 100;

    try {
        alert("Value of variable a is : " + a );
    }catch ( e ) {
        alert("Error: " + e.description );
    }finally {
        alert("Finally block will always execute! " );
    }
}
//-->
</script>
</head>
```

```
<body>
<p>Click the following to see the result:</p>
<form>
<input type="button" value="Click Me" onclick="myFunc(); " />
</form>
</body>
</html>
```

## The **throw** Statement:

You can use **throw** statement to raise your built-in exceptions or your customized exceptions. Later these exceptions can be captured and you can take an appropriate action.

Following is the example showing usage of **throw** statement.

```
<html>
<head>
<script type="text/javascript">
<!--
function myFunc()
{
    var a = 100;
    var b = 0;

    try{
        if ( b == 0 ){
            throw( "Divide by zero error." );
        }else{
            var c = a / b;
        }
    }catch ( e ) {
        alert("Error: " + e );
    }
}
//-->
</script>
</head>
<body>
<p>Click the following to see the result:</p>
<form>
<input type="button" value="Click Me" onclick="myFunc(); " />
</form>
</body>
</html>
```

You can raise an exception in one function using a string, integer, Boolean or an object and then you can capture that exception either in the same function as we did above, or in other function using **try...catch** block.

## The **onerror()** Method

The **onerror** event handler was the first feature to facilitate error handling for JavaScript. The **error** event is fired on the window object whenever an exception occurs on the page. Example:

```
<html>
<head>
<script type="text/javascript">
<!--
window.onerror = function () {
    alert("An error occurred.");
}
//-->
</script>
</head>
<body>
<p>Click the following to see the result:</p>
<form>
<input type="button" value="Click Me" onclick="myFunc(); " />
</form>
</body>
</html>
```

The **onerror** event handler provides three pieces of information to identify the exact nature of the error:

- **Error message** . The same message that the browser would display for the given error
- **URL** . The file in which the error occurred
- **Line number** . The line number in the given URL that caused the error

Here is the example to show how to extract this information

```
<html>
<head>
<script type="text/javascript">
<!--
window.onerror = function (msg, url, line) {
    alert("Message : " + msg );
    alert("url : " + url );
    alert("Line number : " + line );
}
//-->
</script>
</head>
<body>
<p>Click the following to see the result:</p>
<form>
<input type="button" value="Click Me" onclick="myFunc(); " />
</form>
</body>
</html>
```

You can display extracted information in whatever way you think it is better.

You can use **onerror** method to show an error message in case there is any problem in loading an image as follows:

```

```

You can use **onerror** with many HTML tags to display appropriate messages in case of errors.

## JavaScript - Form Validation

Form validation used to occur at the server, after the client had entered all necessary data and then pressed the Submit button. If some of the data that had been entered by the client had been in the wrong form or was simply missing, the server would have to send all the data back to the client and request that the form be resubmitted with correct information. This was really a lengthy process and over burdening server.

JavaScript, provides a way to validate form's data on the client's computer before sending it to the web server. Form validation generally performs two functions.

- **Basic Validation** - First of all, the form must be checked to make sure data was entered into each form field that required it. This would need just loop through each field in the form and check for data.
- **Data Format Validation** - Secondly, the data that is entered must be checked for correct form and value. This would need to put more logic to test correctness of data.

We will take an example to understand the process of validation. Here is the simple form to proceed :

```
<html>
<head>
<title>Form Validation</title>
<script type="text/javascript">
<!--
// Form validation code will come here.
//-->
</script>
</head>
<body>
<form action="/cgi-bin/test.cgi" name="myForm"
      onsubmit="return(validate());">
<table cellspacing="2" cellpadding="2" border="1">
<tr>
  <td align="right">Name</td>
  <td><input type="text" name="Name" /></td>
</tr>
<tr>
  <td align="right">EMail</td>
  <td><input type="text" name="EMail" /></td>
</tr>
```

```
<tr>
  <td align="right">Zip Code</td>
  <td><input type="text" name="Zip" /></td>
</tr>
<tr>
  <td align="right">Country</td>
  <td>
    <select name="Country">
      <option value="-1" selected>[choose yours]</option>
      <option value="1">USA</option>
      <option value="2">UK</option>
      <option value="3">INDIA</option>
    </select>
  </td>
</tr>
<tr>
  <td align="right"></td>
  <td><input type="submit" value="Submit" /></td>
</tr>
</table>
</form>
</body>
</html>
```

## Basic Form Validation:

First we will show how to do a basic form validation. In the above form we are calling **validate()** function to validate data when **onsubmit** event is occurring. Following is the implementation of this validate() function:

```
<script type="text/javascript">
<!--
// Form validation code will come here.
function validate()
{
  if( document.myForm.Name.value == "" )
  {
    alert( "Please provide your name!" );
    document.myForm.Name.focus() ;
    return false;
  }
  if( document.myForm.EMail.value == "" )
  {
    alert( "Please provide your Email!" );
    document.myForm.EMail.focus() ;
    return false;
  }
  if( document.myForm.Zip.value == "" || 
      isNaN( document.myForm.Zip.value ) ||
      document.myForm.Zip.value.length != 5 )
  {
    alert( "Please provide a zip in the format #####." );
    document.myForm.Zip.focus() ;
  }
}</script>
```

```
<html>
<head>
<script type="text/javascript" src="jquery.js"></script>
<script type="text/javascript">
$(document).ready(function(){
    $("button").click(function(){
        $("#test").hide();
    });
});
</script>
</head>

<body>
<h2>This is a heading</h2>
<p>This is a paragraph.</p>
<p id="test">This is another paragraph.</p>
<button>Click me</button>
</body>

</html>
```

**Example : \$("p ").hide()**

```
<html>
<head>
<script type="text/javascript" src="jquery.js"></script>
<script type="text/javascript">
$(document).ready(function(){
    $("button").click(function(){
        $("p").hide();
    });
});
</script>
</head>

<body>
<h2>This is a heading</h2>
<p>This is a paragraph.</p>
<p>This is another paragraph.</p>
<button>Click me</button>
</body>
</html>
```

**Example : \$(".test").hide()**

```
        alert("Hello world!");
    });
});

```

Now we can include **custom.js** file in our HTML file as follows:

```
<html>
<head>
<title>The jQuery Example</title>
<script type="text/javascript"
src="jquery.js"></script>
<script type="text/javascript"
src="/jquery/custom.js"></script>
</head>
<body>
<div id="newdiv">
Click on this to see a dialogue box.
</div>
</body>
</html>
```

## Using Multiple Libraries:

You can use multiple libraries all together without conflicting each others. For example you can use **jQuery** and **MooTool** javascript libraries together.

## jQuery - noConflict() Method

Many JavaScript libraries use \$ as a function or variable name, just as jQuery does. In jQuery's case, \$ is just an alias for jQuery.

Run **\$ .noConflict()** method to give control of the \$ variable back to whichever library first implemented it. This helps to make sure that jQuery doesn't conflict with the \$ object of other libraries.

### Definition and Usage

The **noConflict()** method releases jQuery's control of the \$ variable.

This method can also be used to specify a new custom name for the jQuery variable.

**Tip:** This method is useful when other JavaScript libraries use the \$ for their functions.

### Syntax

`$ .noConflict(removeAll)`

```
<html>
<head>
<script type="text/javascript" src="jquery.js"></script>
<script type="text/javascript">
$(document).ready(function(){
    $("button").click(function(){
        $(".test").hide();
    });
});
</script>
</head>
<body>

<h2 class="test">This is a heading</h2>
<p class="test">This is a paragraph.</p>
<p>This is another paragraph.</p>
<button>Click me</button>
</body>
</html>
```

**NOTE:** jQuery uses a combination of XPath and CSS selector syntax

## The Document Ready Function

You might have noticed that all jQuery methods, in our examples, are inside a document.ready() function:

```
$(document).ready(function(){
    //jQuery functions go here...
});
```

This is to prevent any jQuery code from running before the document is finished loading (is ready).

Here are some examples of actions that can fail if functions are run before the document is fully loaded:

- Trying to hide an element that doesn't exist
- Trying to get the size of an image that is not loaded

## How to use Custom Scripts?

It is better to write our custom code in the custom JavaScript file : **custom.js**, as follows:

```
/* Filename: custom.js */
$(document).ready(function() {
    $("div").click(function() {
```

Parameter	Description
removeAll	Optional. A Boolean value that specifies whether or not to release jQuery's control of ALL jQuery variables (including "jQuery")

```

<html>
<head>
<script type="text/javascript" src="jquery.js"></script>
<script type="text/javascript">
var jq=$.noConflict();
jq(document).ready(function(){
  jq("button").click(function(){
    jq("p").hide();
  });
});
</script>
</head>

<body>
<h2>This is a heading</h2>
<p>This is a paragraph.</p>
<p>This is another paragraph.</p>
<button>Click me</button>
</body>
</html>

```

Here is simple way of avoiding any conflict:

```

// Import other library
// Import jQuery
$.noConflict();
// Code that uses other library's $ can follow here.

```

This technique is especially effective in conjunction with the .ready() method's ability to alias the jQuery object, as within the .ready() we can use \$ if we wish without fear of conflicts later:

```

// Import other library
// Import jQuery
$.noConflict();
jQuery(document).ready(function($) {
// Code that uses jQuery's $ can follow here.
});
// Code that uses other library's $ can follow here.
DOM Element

```

## jQuery - Basics

**jQuery** is a framework built using JavaScript capabilities. So you can use all the functions and other capabilities available in JavaScript.

This chapter would explain most basic concepts but frequently used in jQuery.

### String:

A string in JavaScript is an immutable object that contains none, one or many characters.

Following are the valid examples of a JavaScript String:

```
"This is JavaScript String"  
'This is JavaScript String'  
'This is "really" a JavaScript String'  
"This is 'really' a JavaScript String"
```

### Numbers:

Numbers in JavaScript are double-precision 64-bit format IEEE 754 values. They are immutable, just as strings.

Following are the valid examples of a JavaScript Numbers:

```
5350  
120.27  
0.26
```

### Boolean:

A boolean in JavaScript can be either **true** or **false**. If a number is zero, it defaults to false. If an empty string defaults to false:

Following are the valid examples of a JavaScript Boolean:

```
true      // true  
false     // false  
0         // false  
1         // true  
""        // false  
"hello"   // true
```

## Objects:

JavaScript supports Object concept very well. You can create an object using the object literal as follows:

```
var emp = {  
    name: "Zara",  
    age: 10  
};
```

You can write and read properties of an object using the dot notation as follows:

```
// Getting object properties  
emp.name // ==> Zara  
emp.age // ==> 10  
  
// Setting object properties  
emp.name = "Daisy" // <== Daisy  
emp.age = 20 // <== 20
```

## Arrays:

You can define arrays using the array literal as follows:

```
var x = [];  
var y = [1, 2, 3, 4, 5];
```

An array has a **length** property that is useful for iteration:

```
var x = [1, 2, 3, 4, 5];  
for (var i = 0; i < x.length; i++) {  
    // Do something with x[i]  
}
```

## Functions:

A function in JavaScript can be either named or anonymous. A named function can be defined using *function* keyword as follows:

```
function named(){  
    // do some stuff here  
}
```

An anonymous function can be defined in similar way as a normal function but it would not have any name.

A anonymous function can be assigned to a variable or passed to a method as shown below.

```
var handler = function () {
    // do some stuff here
}
```

JQuery makes a use of anonymous functions very frequently as follows:

```
$(document).ready(function(){
    // do some stuff here
});
```

## Arguments:

JavaScript variable *arguments* is a kind of array which has *length* property. Following example explains it very well:

```
function func(x) {
    console.log(typeof x, arguments.length);
}
func();           //=> "undefined", 0
func(1);          //=> "number", 1
func("1", "2", "3"); //=> "string", 3
```

The arguments object also has a *callee* property, which refers to the function you're inside of. For example:

```
function func() {
    return arguments.callee;
}
func();           // ==> func
```

## Scope:

The scope of a variable is the region of your program in which it is defined. JavaScript variable will have only two scopes.

- **Global Variables:** A global variable has global scope which means it is defined everywhere in your JavaScript code.
- **Local Variables:** A local variable will be visible only within a function where it is defined. Function parameters are always local to that function.

Within the body of a function, a local variable takes precedence over a global variable with the same name:

```

var myVar = "global";      // ==> Declare a global variable

function () {
    var myVar = "local";  // ==> Declare a local variable
    document.write(myVar); // ==> local
}

```

## Built-in Functions:

JavaScript comes along with a useful set of built-in functions. These methods can be used to manipulate Strings, Numbers and Dates.

Following are important JavaScript functions:

Method	Description
charAt()	Returns the character at the specified index.
concat()	Combines the text of two strings and returns a new string.
forEach()	Calls a function for each element in the array.
indexOf()	Returns the index within the calling String object of the first occurrence of the specified value, or -1 if not found.
length()	Returns the length of the string.
pop()	Removes the last element from an array and returns that element.
push()	Adds one or more elements to the end of an array and returns the new length of the array.
reverse()	Reverses the order of the elements of an array -- the first becomes the last, and the last becomes the first.
sort()	Sorts the elements of an array.
substr()	Returns the characters in a string beginning at the specified location through the specified number of characters.

toLowerCase()	Returns the calling string value converted to lower case.
toString()	Returns the string representation of the number's value.
toUpperCase()	Returns the calling string value converted to uppercase.

## jQuery - Selectors

A jQuery Selector is a function which makes use of expressions to find out matching elements from a DOM based on the given criteria.

### How to use Selectors?

The selectors are very useful and would be required at every step while using jQuery. They get the exact element that you want from your HTML document.

Following table lists down few basic selectors and explains them with examples.

Selector	Description
Name	Selects all elements which match with the given element <b>Name</b> .
#ID	Selects a single element which matches with the given <b>ID</b>
.Class	Selects all elements which match with the given <b>Class</b> .
<u>Universal (*)</u>	Selects all elements available in a DOM.
<u>Multiple Elements E, F, G</u>	Selects the combined results of all the specified selectors <b>E, F</b> or <b>G</b> .

## jQuery - CSS Element Selector

### Description:

The element selector selects all the elements that have a tag name of T.

### Syntax:

Here is the simple syntax to use this selector:

```
$('tagname')
```

### Parameters:

Here is the description of all the parameters used by this selector:

- **tagname:** Any standard HTML tag name like div, p, em, img , li etc.

### Returns:

Like any other jQuery selector, this selector also returns an array filled with the found elements.

### Example:

- `$('p')` selects all elements with a tag name of **p** in the document.
- `$('div')` selects all elements with a tag name of **div** in the document.

Following example would select all the divisions and display them one by one:

```
<html>
<head>
<title>The Selecter Example</title>
<script type="text/javascript" src="jquery.js">
</script>

<script type="text/javascript" language="javascript">

$(document).ready(function() {
    /* This would select all the divisions */
    var divs = $("div");
    for( i=0; i<divs.length; i++ ){
        alert("Found Division: " + divs[i].innerHTML);
    }
});
```

```
</script>
</head>
<body>

<div class="div1" id="divid1">
    this is div one
</div>
<br />

<div class="div2" id="divid2">
    this is div two
</div>
<br />

<div class="div3" id="divid3">
    this is div three
</div>

</body>
</html>
```

```
<html>
<head>
<title>The Selector Example</title>
<script type="text/javascript" src="jquery.js">
</script>

<script type="text/javascript" language="javascript">

$(document).ready(function() {
    /* This would select all the divisions */
    $("button").click(function(){
        var divs = $("div");
        for(var i=0 ;i<divs.length ; i++){
            if(i==0){
                $(divs[i]).addClass("ybg");
            }
            if(i==1){
                $(divs[i]).addClass("pbg");
            }
            if(i==2){

                $(divs[i]).addClass("obg");
            }
        }
    })
}

</script>
```

```
});  
});  
  
</script>  
  
<style type="text/css" >  
  
.ybg{  
background-color : yellow;  
}  
  
.pbg{  
background-color : pink;  
}  
  
.obj{  
background-color : orange;  
}  
  
.div1{ color : red }  
.div2{ color : green }  
.div3{ color : blue }  
  
</style>  
  
</head>  
<body>  
<button> Click </button>  
  
<div class="div1" id="divid1">  
this is div one  
</div>  
<br />  
  
<div class="div2" id="divid2">  
this is div two  
</div>  
<br />  
  
<div class="div3" id="divid3">  
this is div three  
</div>  
  
</body>  
</html>
```

```
<html>  
<head>  
<title>The Selecter Example</title>
```

```
<script type="text/javascript" src="jquery.js">
</script>

<script type="text/javascript" language="javascript">
$(document).ready(function() {

    $("button").click(function(){

        var outputText = "";
        $("div.div1, div.div2, div.div3").each(function(index){
            outputText = outputText + index + "- "+$(this).text()+" <br/>";
        });
        $("div#output").html(outputText);

    });

});

</script>

<style type="text/css" >

.div1{ color : red }
.div2{ color : green }
.div3{ color : blue }

</style>

</head>
<body>
<button> Click </button>

<div id="output" style="border : 2px solid red; height : 100px; width:500px;" >
</div>

<div class="div1" id="divid1">
<p>this is div one </p>
</div>
<br />

<div class="div2" id="divid2">
    this is div two
</div>
<br />

<div class="div3" id="divid3">
    this is div three
</div>
```

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

## jQuery - CSS Element ID Selector

### Description:

The element ID selector selects a single element with the given id attribute.

### Syntax:

Here is the simple syntax to use this selector:

```
$( '#elementid' )
```

### Parameters:

Here is the description of all the parameters used by this selector:

- **elementid:** This would be an element ID. If the id contains any special characters like periods or colons you have to escape those characters with backslashes.

### Returns:

Like any other jQuery selector, this selector also returns an array filled with the found element.

### Example:

- `$('#myid')` selects a single element with the given id myid.
- `$('#div#yourid')` selects a single division with the given id yourid.

Following example would select second division and display its content:

```
<html>
<head>
<title>The Selecter Example</title>
<script type="text/javascript" src="jquery.js">
</script>

<script type="text/javascript" language="javascript">

$(document).ready(function() {
    var divs = $("#divid2");
```

```
        alert("Found Division: " + divs[0].innerHTML);
    });

</script>
</head>
<body>

<div class="div1" id="divid1">
    <p class="paral" id="pid1">This is first paragraph.</p>
    <p class="para2" id="pid2">This is second paragraph.</p>
</div>
<br />

<div class="div2" id="divid2">
    <p>This is second division of the DOM.</p>
</div>
<br />

<div class="div3" id="divid3">
    <p>This is a para inside third division</p>
</div>

</body>
</html>
```

## jQuery - CSS Element Class Selector

### Description:

The element class selector selects all the elements which match with the given class of the elements.

### Syntax:

Here is the simple syntax to use this selector:

```
$('.classid')
```

### Parameters:

Here is the description of all the parameters used by this selector:

- **classid:** This is class ID available in the document.

### Returns:

Like any other jQuery selector, this selector also returns an array filled with the found elements.

**Example:**

- `$('.big')` selects all the elements with the given class ID **big**.
- `($('p.small')` selects all the paragraphs with the given class ID **small**.
- `($('.big.small')` selects all the elements with a class of **big** and **small**.

Following example would select all divisions with class **.big** and display its content:

```
<html>
<head>
<title>The Selector Example</title>
<script type="text/javascript" src="jquery.js">
</script>

<script type="text/javascript" language="javascript">

$(document).ready(function() {
    var divs = $(".big");
    for( i=0; i<divs.length; i++ ){
        alert("Found Division: " + divs[i].innerHTML);
    }
});

</script>
</head>
<body>

<div class="big" id="divid1">
    <p class="para1" id="pid1">This is first paragraph.</p>
    <p class="para2" id="pid2">This is second paragraph.</p>
    <p class="para3" id="pid3">This is third paragraph.</p>
</div>
<br />

<div class="big" id="divid2">
    <p>This is second division of the DOM.</p>
    <p>This is second para inside second division.</p>
</div>
<br />

<div class="medium" id="divid3">
    <p>This is a para inside third division</p>
</div>

</body>
</html>
```

## jQuery - CSS Universal Selector

### Description:

The universal selector selects all the elements available in the document.

### Syntax:

Here is the simple syntax to use this selector:

```
$('*')
```

### Parameters:

Here is the description of all the parameters used by this selector:

- \*: A symbolic star.

### Returns:

Like any other jQuery selector, this selector also returns an array filled with the found elements.

### Example:

- \$('\*') selects all the elements available in the document.

Following example would select all the elements available and would display them one by one:

```
<html>
<head>
<title>The Selector Example</title>
<script type="text/javascript" src="jquery.js">
</script>

<script type="text/javascript" language="javascript">

$(document).ready(function() {
    var elements = $("*");
    for( i=0; i<elements.length; i++ ){
        alert("Found element: " + elements[i].innerHTML);
    }
});

</script>
</head>
```

```
<body>

<div class="big" id="divid1">
  <p class="para1" id="pid1">This is first paragraph.</p>
  <p class="para2" id="pid2">This is second paragraph.</p>
  <p class="para3" id="pid3">This is third paragraph.</p>
</div>
<br />

<div class="big" id="divid2">
  <p>This is second division of the DOM.</p>
  <p>This is second para inside second division.</p>
</div>
<br />

<div class="medium" id="divid3">
  <p>This is a para inside third division</p>
</div>

</body>
</html>
```

## jQuery - CSS Multiple Elements E, F, G Selector

### Description:

This Multiple Elements selector selects the combined results of all the specified selectors E, F or G.

You can specify any number of selectors to combine into a single result. Here order of the DOM elements in the jQuery object aren't necessarily identical.

### Syntax:

Here is the simple syntax to use this selector:

```
$( 'E, F, G, ....' )
```

### Parameters:

Here is the description of all the parameters used by this selector:

- **E:** Any valid selector
- **F:** Any valid selector
- **G:** Any valid selector
- ....

**Returns:**

Like any other jQuery selector, this selector also returns an array filled with the found elements.

**Example:**

- `$('.div, p')`: selects all the elements matched by **div** or **p**.
- `$('.p strong, .myclass')`: selects all elements matched by **strong** that are descendants of an element matched by **p** as well as all elements that have a class of **myclass**.
- `$('.p strong, #myid')`: selects a single elements matched by **strong** that is descendant of an element matched by **p** as well as element whose id is **myid**.

Following example would select elements with class ID **big** and element with ID **divid3**:

```
<html>
<head>
<title>The Selector Example</title>
<script type="text/javascript" src="jquery.js">
</script>

<script type="text/javascript" language="javascript">

$(document).ready(function() {
    var elements = $(".big, #divid3");
    for( i=0; i<elements.length; i++ ){
        alert("Found element: " + elements[i].innerHTML);
    }
});

</script>
</head>
<body>

<div class="big" id="divid1">
    <p class="para1" id="pid1">This is first paragraph.</p>
    <p class="para2" id="pid2">This is second paragraph.</p>
    <p class="para3" id="pid3">This is third paragraph.</p>
</div>
<br />

<div class="big" id="divid2">
    <p>This is second division of the DOM.</p>
    <p>This is second para inside second division.</p>
</div>
<br />

<div class="medium" id="divid3">
    <p>This is a para inside third division</p>
</div>

</body>
</html>
```

**jQuery Selectors**

Selector	Example	Selects
<u>*</u>	<code>\$("")</code>	All elements
<u>#id</u>	<code>\$("#lastname")</code>	The element with id=lastname
<u>.class</u>	<code>\$(".intro")</code>	All elements with class="intro"
<u>element</u>	<code> \$("p")</code>	All p elements
<u>.class.class</u>	<code> \$(".intro.demo")</code>	All elements with the classes "intro" and "demo"
<u>:first</u>	<code> \$("p:first")</code>	The first p element
<u>:last</u>	<code> \$("p:last")</code>	The last p element
<u>:even</u>	<code> \$("tr:even")</code>	All even tr elements
<u>:odd</u>	<code> \$("tr:odd")</code>	All odd tr elements
<u>:eq(index)</u>	<code> \$("ul li:eq(3)")</code>	The fourth element in a list (index starts at 0)
<u>:gt(no)</u>	<code> \$("ul li:gt(3)")</code>	List elements with an index greater than 3
<u>:lt(no)</u>	<code> \$("ul li:lt(3)")</code>	List elements with an index less than 3
<u>:not(selector)</u>	<code> \$("input:not(:empty)")</code>	All input elements that are not empty
<u>:header</u>	<code> \$(":header")</code>	All header elements h1, h2 ...
<u>:animated</u>	<code> \$(":animated")</code>	All animated elements
<u>:contains(text)</u>	<code> \$(":contains(sekharit)")</code>	All elements which contains the text
<u>:empty</u>	<code> \$(":empty")</code>	All elements with no child (elements) nodes
<u>:hidden</u>	<code> \$("p:hidden")</code>	All hidden p elements

<u>:visible</u>	<code>\$("table:visible")</code>	All visible tables
<u>s1,s2,s3</u>	<code> \$("th,td,.intro")</code>	All elements with matching selectors
<u>[attribute]</u>	<code> \$("[href]")</code>	All elements with a href attribute
<u>[attribute=value]</u>	<code> \$("[href='default.htm'])")</code>	All elements with a href attribute value equal to "default.htm"
<u>[attribute!=value]</u>	<code> \$("[href!='default.htm'])")</code>	All elements with a href attribute value not equal to "default.htm"
<u>[attribute\$=value]</u>	<code> \$("[href\$='.jpg'])")</code>	All elements with a href attribute value ending with ".jpg"
<u>:input</u>	<code> \$(":input")</code>	All input elements
<u>:text</u>	<code> \$(":text")</code>	All input elements with type="text"
<u>:password</u>	<code> \$(":password")</code>	All input elements with type="password"
<u>:radio</u>	<code> \$(":radio")</code>	All input elements with type="radio"
<u>:checkbox</u>	<code> \$(":checkbox")</code>	All input elements with type="checkbox"
<u>:submit</u>	<code> \$(":submit")</code>	All input elements with type="submit"
<u>:reset</u>	<code> \$(":reset")</code>	All input elements with type="reset"
<u>:button</u>	<code> \$(":button")</code>	All input elements with type="button"
<u>:image</u>	<code> \$(":image")</code>	All input elements with type="image"
<u>:file</u>	<code> \$(":file")</code>	All input elements with type="file"
<u>:enabled</u>	<code> \$(":enabled")</code>	All enabled input elements
<u>:disabled</u>	<code> \$(":disabled")</code>	All disabled input elements
<u>:selected</u>	<code> \$(":selected")</code>	All selected input elements

:checked

`$(":checked")`

All checked input elements

**Example: :first, :last**

```
$(document).ready(function() {
    $("button").click(function(){
        $("p:first").addClass("obg");
        $("p:last").addClass("pbг");
    });
});
```

**Example: :first, :last like example with our own logic**

```
$(document).ready(function() {
    $("button").click(function(){
        var count = $("p").size();
        $("p").each(function(index){
            if(index == 0){
                $(this).addClass("obg");
            }
            if(index == (count-1)){
                $(this).addClass("pbг");
            }
        });
    });
});
```

**Example: :even, :odd**

```
$(document).ready(function() {
    $("button").click(function(){
        $("tr:even").addClass("obg");
        $("tr:odd").addClass("pbг");
    });
});
```

Similar to above syntax and examples, following examples would give you understanding on using different type of other useful selectors:

- **\$('\*'):** This selector selects all elements in the document.
- **\$("p > \*"):** This selector selects all elements that are children of a paragraph element.
- **\$("#specialID"):** This selector function gets the element with id="specialID".
- **\$(".specialClass"):** This selector gets all the elements that have the class of *specialClass*.
- **\$("li:not(.myclass)"):** Selects all elements matched by <li> that do not have class="myclass".
- **\$("a#specialID.specialClass"):** This selector matches links with an id of *specialID* and a class of *specialClass*.
- **\$("p a.specialClass"):** This selector matches links with a class of *specialClass* declared within <p> elements.
- **\$("ul li:first"):** This selector gets only the first <li> element of the <ul>.
- **\$("#container p"):** Selects all elements matched by <p> that are descendants of an element that has an id of *container*.
- **\$("li > ul"):** Selects all elements matched by <ul> that are children of an element matched by <li>
- **\$("strong + em"):** Selects all elements matched by <em> that immediately follow a sibling element matched by <strong>.
- **\$("p ~ ul"):** Selects all elements matched by <ul> that follow a sibling element matched by <p>.
- **\$("code, em, strong"):** Selects all elements matched by <code> or <em> or <strong>.
- **\$("p strong, .myclass"):** Selects all elements matched by <strong> that are descendants of an element matched by <p> as well as all elements that have a class of *myclass*.
- **\$(":empty"):** Selects all elements that have no children.
- **\$("p:empty"):** Selects all elements matched by <p> that have no children.
- **\$("div[p]"):** Selects all elements matched by <div> that contain an element matched by <p>.
- **\$("p[.myclass]"):** Selects all elements matched by <p> that contain an element with a class of *myclass*.
- **\$("a[@rel]"):** Selects all elements matched by <a> that have a rel attribute.
- **\$("input[@name=myname]"):** Selects all elements matched by <input> that have a name value exactly equal to *myname*.
- **\$("input[@name^=myname]"):** Selects all elements matched by <input> that have a name value beginning with *myname*.
- **\$("a[@rel\$=self]"):** Selects all elements matched by <p> that have a class value ending with *bar*
- **\$("a[@href\*=domain.com]"):** Selects all elements matched by <a> that have an href value containing domain.com.
- **\$("li:even"):** Selects all elements matched by <li> that have an even index value.
- **\$("tr:odd"):** Selects all elements matched by <tr> that have an odd index value.
- **\$("li:first"):** Selects the first <li> element.
- **\$("li:last"):** Selects the last <li> element.
- **\$("li:visible"):** Selects all elements matched by <li> that are visible.
- **\$("li:hidden"):** Selects all elements matched by <li> that are hidden.
- **\$(":radio"):** Selects all radio buttons in the form.
- **\$(":checked"):** Selects all checked boxes in the form.
- **\$(":input"):** Selects only form elements (input, select, textarea, button).
- **\$(":text"):** Selects only text elements (input[type=text]).
- **\$("li:eq(2)"):** Selects the third <li> element
- **\$("li:eq(4)"):** Selects the fifth <li> element
- **\$("li:lt(2)"):** Selects all elements matched by <li> element before the third one; in other words, the first two <li> elements.

- `$("p:lt(3)")`: selects all elements matched by `<p>` elements before the fourth one; in other words the first three `<p>` elements.
- `$("li:gt(1)")`: Selects all elements matched by `<li>` after the second one.
- `$("p:gt(2)")`: Selects all elements matched by `<p>` after the third one.
- `$("div/p")`: Selects all elements matched by `<p>` that are children of an element matched by `<div>`.
- `$("div//code")`: Selects all elements matched by `<code>` that are descendants of an element matched by `<div>`.
- `$("//p//a")`: Selects all elements matched by `<a>` that are descendants of an element matched by `<p>`
- `$("li:first-child")`: Selects all elements matched by `<li>` that are the first child of their parent.
- `$("li:last-child")`: Selects all elements matched by `<li>` that are the last child of their parent.
- `$(":parent")`: Selects all elements that are the parent of another element, including text.
- `$("li:contains(second)")`: Selects all elements matched by `<li>` that contain the text second.

You can use all the above selectors with any HTML/XML element in generic way. For example if selector `$("li:first")` works for `<li>` element then `$("p:first")` would also work for `<p>` element.

## JQuery Callback Functions

A callback function is executed after the current animation is 100% finished.

### JQuery Callback Functions

JavaScript statements are executed line by line. However, with animations, the next line of code can be run even though the animation is not finished. This can create errors.

To prevent this, you can create a callback function.

A callback function is executed after the current animation (effect) is finished.

### JQuery Callback Example

Typical syntax: `$(selector).hide(speed,callback)`

The callback parameter is a function to be executed after the hide effect is completed:

#### Example with Callback

```
$( "p" ).hide(1000,function(){
  alert("The paragraph is now hidden");
});
```

Without a callback parameter, the alert box is displayed before the hide effect is completed:

#### Example without Callback

```
$( "p" ).hide(1000);
```

```
alert("The paragraph is now hidden");
```

### Example:

```
<html>
<head>
<title>the title</title>
<script type="text/javascript" src="jquery.js"></script>
<script type="text/javascript" language="javascript">
$(document).ready(function() {
    $("button").click(function(){
        $("div#abc").hide(2000, function(){
            alert("Hiding is over");
        });
    });
});
</script>
</head>
<body>
    <button>Click Me</button>
    <div id="abc" >
        <ul>
            <li>list item 1</li>
            <li>list item 2</li>
            <li>list item 3</li>
            <li>list item 4</li>
            <li>list item 5</li>
            <li>list item 6</li>
        </ul>
    </div>
</body>
</html>
```

## JQuery - DOM Attributes

Some of the most basic components we can manipulate when it comes to DOM elements are the properties and attributes assigned to those elements.

Most of these attributes are available through JavaScript as DOM node properties. Some of the more common properties are:

- className
- tagName
- id
- href

- title
- rel
- src

Consider the following HTML markup for an image element:

```

```

In this element's markup, the tag name is img, and the markup for id, src, alt, class, and title represents the element's attributes, each of which consists of a name and a value.

jQuery gives us the means to easily manipulate an element's attributes and gives us access to the element so that we can also change its properties.

## Get Attribute Value:

The **attr()** method can be used to either fetch the value of an attribute from the first element in the matched set or set attribute values onto all matched elements.

### Example: selector. attr(name)

Following is a simple example which fetches title attribute of <em> tag and set <div id="divid"> value with the same value:

```
<html>
<head>
<title>the title</title>
<script type="text/javascript"
src="jquery.js"></script>
<script type="text/javascript" language="javascript">

$(document).ready(function() {
    var title = $("em").attr("title");
    $("#divid").text(title);
});

</script>
</head>
<body>
<div>
    <em title="Bold and Brave">This is first paragraph.</em>
    <p id="myid">This is second paragraph.</p>
    <div id="divid"></div>
</div>
</body>
</html>
```

## Set Attribute Value:

The **attr(name, value)** method can be used to set the named attribute onto all elements in the wrapped set using the passed value.

### Example: selector.attr(name, value)

Following is a simple example which set **src** attribute of an image tag to a correct location:

```
<html>
<head>
<title>the title</title>
<script type="text/javascript"
src="jquery.js"></script>
<script type="text/javascript" language="javascript">

$(document).ready(function() {
    $("#myimg").attr("src", "/images/jquery.jpg");
});

</script>
</head>
<body>
<div>
    
</div>
</body>
</html>
```

### Example: selector.attr({ json })

```
<html>
<head>
<title>the title</title>
<script type="text/javascript" src="jquery.js"></script>
<script type="text/javascript" language="javascript">

$(document).ready(function() {
    $("button").click(function(){
        $("div#abc").attr(
            {
                title :"This is Title",
                style :"color : red; border: 1px solid green"
            }
        );
    });
});

</script>
</head>
<body>
```

```
<button>Give default value</button>
<div id="abc" >
    This is some div
</div>
</body>
</html>
```

## Applying Styles:

The **addClass( classes )** method can be used to apply defined style sheets onto all the matched elements. You can specify multiple classes separated by space.

### Example: selector.addClass(classs)

Following is a simple example which set **src** attribute of an image tag to a correct location:

```
<html>
<head>
<title>the title</title>
<script type="text/javascript"
src="jquery.js"></script>
<script type="text/javascript" language="javascript">

$(document).ready(function() {
    $("em").addClass("selected");
    $("#myid").addClass("highlight");
});

</script>
<style>
    .selected { color:red; }
    .highlight { background:yellow; }
</style>
</head>
<body>
    <em title="Bold and Brave">This is first paragraph.</em>
    <p id="myid">This is second paragraph.</p>
</body>
</html>
```

## Useful Attribute Methods:

Following table lists down few useful methods which you can use to manipulate attributes and properties:

Methods	Description
---------	-------------

<u>attr( properties )</u>	Set a key/value object as properties to all matched elements.
<u>attr( key, fn )</u>	Set a single property to a computed value, on all matched elements.
<u>removeAttr( name )</u>	Remove an attribute from each of the matched elements.
<u>hasClass( class )</u>	Returns true if the specified class is present on at least one of the set of matched elements.
<u>removeClass( class )</u>	Removes all or the specified class(es) from the set of matched elements.
<u>toggleClass( class )</u>	Adds the specified class if it is not present, removes the specified class if it is present.
<u>html()</u>	Get the html contents (innerHTML) of the first matched element.
<u>html( val )</u>	Set the html contents of every matched element.
<u>text()</u>	Get the combined text contents of all matched elements.
<u>text( val )</u>	Set the text contents of all matched elements.
<u>val()</u>	Get the input value of the first matched element.
<u>val( val )</u>	Set the value attribute of every matched element if it is called on <input> but if it is called on <select> with the passed <option> value then passed option would be selected, if it is called on check box or radio box then all the matching check box and radiobox would be checked.

Similar to above syntax and examples, following examples would give you understanding on using various attribute methods in different situation:

- `$("#myID").attr("custom")` : This would return value of attribute *custom* for the first element matching with ID myID.
- `$("img").attr("alt", "Sample Image")`: This sets the **alt** attribute of all the images to a new value "Sample Image".

- `$(“input”).attr({ value: “”, title: “Please enter a value” })` : Sets the value of all <input> elements to the empty string, as well as sets the title to the string *Please enter a value*.
- `$(“a[href^=http://]”).attr(“target”, “_blank”)`: Selects all links with an href attribute starting with *http://* and set its target attribute to *\_blank*
- `$(“a”).removeAttr(“target”)` : This would remove *target* attribute of all the links.
- `$(“form”).submit(function() {$(“:submit”, this).attr(“disabled”, “disabled”);})`; : This would modify the disabled attribute to the value "disabled" while clicking Submit button.
- `$(“p:last”).hasClass(“selected”)`: This return true if last <p> tag has associated class *selected*.
- `$(“p”).text()`: Returns string that contains the combined text contents of all matched <p> elements.
- `$(“p”).text(“<i>Hello World</i>”)`: This would set "<i>Hello World</i>" as text content of the matching <p> elements
- `$(“p”).html()` : This returns the HTML content of the all matching paragraphs.
- `$(“div”).html(“Hello World”)` : This would set the HTML content of all matching <div> to *Hello World*.
- `$(“input:checkbox:checked”).val()` : Get the first value from a checked checkbox
- `$(“input:radio[name=bar]:checked”).val()`: Get the first value from a set of radio buttons
- `$(“button”).val(“Hello”)` : Sets the value attribute of every matched element <button>.
- `$(“input”).val(“on”)` : This would check all the radio or check box button whose value is "on".
- `$(“select”).val(“Orange”)` : This would select Orange option in a dropdown box with options Orange, Mango and Banana.
- `$(“select”).val(“Orange”, “Mango”)` : This would select Orange and Mango options in a dropdown box with options Orange, Mango and Banana.

## jQuery - DOM Traversing

jQuery is a very powerful tool which provides a variety of DOM traversal methods to help us select elements in a document randomly as well as in sequential method.

Most of the DOM Traversal Methods do not modify the jQuery object and they are used to filter out elements from a document based on given conditions.

### Find Elements by index:

Consider a simple document with the following HTML content:

```
<html>
<head>
<title>the title</title>
</head>
<body>
  <div>
    <ul>
      <li>list item 1</li>
      <li>list item 2</li>
      <li>list item 3</li>
      <li>list item 4</li>
      <li>list item 5</li>
      <li>list item 6</li>
```

```
</ul>
</div>
</body>
</html>
```

- Above every list has its own index, and can be located directly by using **eq(index)** method as below example.
- Every child element starts its index from zero, thus, *list item 2* would be accessed by using `$( "li" ).eq(1)` and so on.

### Example: eq(index)

Following is a simple example which adds the color to second list item.

```
<html>
<head>
<title>the title</title>
<script type="text/javascript"
src="jquery.js"></script>
<script type="text/javascript" language="javascript">

$(document).ready(function() {
    $("li").eq(2).addClass("selected");
});

</script>
<style>
    .selected { color:red; }
</style>
</head>
<body>
    <div>
        <ul>
            <li>list item 1</li>
            <li>list item 2</li>
            <li>list item 3</li>
            <li>list item 4</li>
            <li>list item 5</li>
            <li>list item 6</li>
        </ul>
    </div>
</body>
</html>
```

### Filtering out Elements:

The **filter( selector )** method can be used to filter out all elements from the set of matched elements that do not match the specified selector(s). The *selector* can be written using any selector syntax.

### Example: filter( selector )

Following is a simple example which applies color to the lists associated with middle class:

```

<html>
<head>
<title>the title</title>
<script type="text/javascript"    src="jquery.js"></script>
<script type="text/javascript" language="javascript">

$(document).ready(function() {

    $("button").click(function() {
        $("li").filter(".middle").addClass("selected");

    });
});

</script>
<style>
    .selected { color:red; }
</style>
</head>
<body>
<button>Click Me </button>
<div>
<ul>
    <li class="top">list item 1</li>
    <li class="top">list item 2</li>
    <li class="middle">list item 3</li>
    <li class="middle">list item 4</li>
    <li class="bottom">list item 5</li>
    <li class="bottom">list item 6</li>
</ul>
</div>
</body>
</html>

```

### Locating Descendent Elements :

The **find( selector )** method can be used to locate all the descendent elements of a particular type of elements. The *selector* can be written using any selector syntax.

### Example: find( selector )

Following is an example which selects all the `<span>` elements available inside different `<p>` elements:

```

<html>
<head>
<title>the title</title>
<script type="text/javascript"    src="jquery.js"></script>

```

```

<script type="text/javascript" language="javascript">
$(document).ready(function() {
    $("button").click(function() {
        $("p").find("span").addClass("selected");
    });
});

</script>
<style>
    .selected { color:red; }
</style>
</head>
<body>
<button>Click Me </button>
<p>This is 1st paragraph and <span>THIS IS RED</span> </p>
<p>This is 2nd paragraph and <span>THIS IS ALSO RED</span> </p>
<p> this is 3rd paragraph <strong> I am strong </strong> </p>
</body>
</html>

```

## JQuery DOM Traversing Methods:

Following table lists down useful methods which you can use to filter out various elements from a list of DOM elements:

Selector	Description
<u>eq( index )</u>	Reduce the set of matched elements to a single element.
<u>filter( selector )</u>	Removes all elements from the set of matched elements that do not match the specified selector(s).
<u>filter( fn )</u>	Removes all elements from the set of matched elements that do not match the specified function.
<u>is( selector )</u>	Checks the current selection against an expression and returns true, if at least one element of the selection fits the given selector.
<u>map( callback )</u>	Translate a set of elements in the jQuery object into another set of values in a jQuery array (which may, or may not contain elements).

<u>not( selector )</u>	Removes elements matching the specified selector from the set of matched elements.
<u>slice( start, [end] )</u>	Selects a subset of the matched elements.

Following table lists down other useful methods which you can use to locate various elements in a DOM:

Selector	Description
<u>add( selector )</u>	Adds more elements, matched by the given selector, to the set of matched elements.
<u>andSelf( )</u>	Add the previous selection to the current selection.
<u>children( [selector] )</u>	Get a set of elements containing all of the unique immediate children of each of the matched set of elements.
<u>closest( selector )</u>	Get a set of elements containing the closest parent element that matches the specified selector, the starting element included.
<u>contents( )</u>	Find all the child nodes inside the matched elements (including text nodes), or the content document, if the element is an iframe.
<u>end( )</u>	Revert the most recent 'destructive' operation, changing the set of matched elements to its previous state .
<u>find( selector )</u>	Searches for descendent elements that match the specified selectors.
<u>next( [selector] )</u>	Get a set of elements containing the unique next siblings of each of the given set of elements.
<u>nextAll( [selector] )</u>	Find all sibling elements after the current element.
<u>offsetParent( )</u>	Returns a jQuery collection with the positioned parent of the first matched element.

<u>parent( [selector] )</u>	Get the direct parent of an element. If called on a set of elements, parent returns a set of their unique direct parent elements.
<u>parents( [selector] )</u>	Get a set of elements containing the unique ancestors of the matched set of elements (except for the root element).
<u>prev( [selector] )</u>	Get a set of elements containing the unique previous siblings of each of the matched set of elements.
<u>prevAll( [selector] )</u>	Find all sibling elements in front of the current element.
<u>siblings( [selector] )</u>	Get a set of elements containing all of the unique siblings of each of the matched set of elements.

## jQuery - CSS Methods

The jQuery library supports nearly all of the selectors included in Cascading Style Sheet (CSS) specifications 1 through 3, as outlined on the World Wide Web Consortium's site.

Using JQuery library developers can enhance their websites without worrying about browsers and their versions as long as the browsers have JavaScript enabled.

Most of the JQuery CSS Methods do not modify the content of the jQuery object and they are used to apply CSS properties on DOM elements.

### Apply CSS Properties:

This is very simple to apply any CSS property using JQuery method **css( PropertyName, PropertyValue )**.

Here is the syntax for the method:

```
selector.css( PropertyName, PropertyValue );
```

Here you can pass *PropertyName* as a javascript string and based on its value, *PropertyValue* could be string or integer.

### Example: selector.css(propertyName, propertyValue)

Following is an example which adds font color to the second list item.

```

<html>
<head>
<title>the title</title>
<script type="text/javascript"    src="jquery.js"></script>
<script type="text/javascript" language="javascript">

$(document).ready(function() {
    $("button").click(function(){
        $("div#abc").css("border", "1px solid green");
    });
});

</script>
<style type="text/css" >
    .pbg { background-color : pink }
</style>
</head>
<body>

    <button>Click Me</button>
    <div id="abc" >
        This is some div
    </div>
</body>
</html>

```

## Apply Multiple CSS Properties:

You can apply multiple CSS properties using a single JQuery method **CSS( {key1:val1, key2:val2....} )**. You can apply as many properties as you like in a single call.

Here is the syntax for the method:

```
selector.css( {key1:val1, key2:val2....keyN:valN})
```

Here you can pass key as property and val as its value as described above.

### Example: selector.css({ json object } )

Following is an example which adds font color as well as background color to the second list item.

```

<html>
<head>
<title>the title</title>
<script type="text/javascript"    src="jquery.js"></script>
<script type="text/javascript" language="javascript">

$(document).ready(function() {
    $("button").click(function(){
        $("div#abc").css(

```

```

        {
            border : "1px solid green",
            color : "red",
            width : "400px",
            height : "300px"
        }
    );
} );
});

</script>
<style type="text/css" >
    .pbg { background-color : pink }
</style>
</head>
<body>

    <button>Click Me</button>
    <div id="abc" >
        This is some div
    </div>
</body>
</html>

```

## Setting Element Width & Height:

The **width( val )** and **height( val )** method can be used to set the width and height respectively of any element.

### Example: selector.width(val), selector.height(val)

Following is a simple example which sets the width of first division element where as rest of the elements have width set by style sheet:

```

<html>
<head>
<title>the title</title>
<script type="text/javascript"    src="jquery.js"></script>
<script type="text/javascript" language="javascript">

$(document).ready(function() {
    $("div").click(function(){
        $(this).width(100);
        $(this).height(150);
        $(this).css("background-color", "blue");

    });
});

</script>
<style type="text/css" >

div{

```

```
        width:70px;
        height:50px;
        float:left;
        margin:5px;
        background:red;
        cursor:pointer;
    }

</style>
</head>
<body>

<div>ONE</div>
<div>TWO</div>
<div>THREE</div>
<div>FOUR</div>
<div>FIVE</div>

</body>
</html>
```

## Example : Function chaining

```
<html>
<head>
<title>the title</title>
<script type="text/javascript" src="jquery.js"></script>
<script type="text/javascript" language="javascript">

$(document).ready(function() {
    $("button").click(function(){

        $("div#abc").css("border", "1px solid green")
                    .css("color", "red")
                    .width("400px")
                    .height("500px")
                    .addClass("pbg");

    });
});

</script>
<style type="text/css" >
    .pbг { background-color : pink }
</style>
</head>
<body>

<button>Click Me</button>
```

```
<div id="abc" >
    This is some div
</div>
</body>
</html>
```

### Example:selector.toggleClass(css-class)

```
<html>
<head>
<title>the title</title>
<script type="text/javascript" src="jquery.js"></script>
<script type="text/javascript" language="javascript">

function MyFocusBlur(tb){
    $(tb).toggleClass("ybg");
}

</script>
<style type="text/css" >
.ybg{ background-color : yellow; }

</style>
</head>
<body>

    First Name : <input type="text" name="fname" onblur="MyFocusBlur(this)" onfocus="MyFocusBlur(this)" />
<br/>
    Last Name : <input type="text" name="lname" onblur="MyFocusBlur(this)" onfocus="MyFocusBlur(this)" />
<br/>
</body>
</html>
```

### JQuery CSS Methods:

Following table lists down all the methods which you can use to play with CSS properties:

Method	Description
<u>css( name )</u>	Return a style property on the first matched element.
<u>css( name, value )</u>	Set a single style property to a value on all matched elements.
<u>css( properties )</u>	Set a key/value object as style properties to all matched elements.

height( val )

Set the CSS height of every matched element.

height( )

Get the current computed, pixel, height of the first matched element.

innerHeight( )

Gets the inner height (excludes the border and includes the padding) for the first matched element.

innerWidth( )

Gets the inner width (excludes the border and includes the padding) for the first matched element.

offset( )

Get the current offset of the first matched element, in pixels, relative to the document

offsetParent( )

Returns a jQuery collection with the positioned parent of the first matched element.

outerHeight( [margin] )

Gets the outer height (includes the border and padding by default) for the first matched element.

outerWidth( [margin] )

Get the outer width (includes the border and padding by default) for the first matched element.

position( )

Gets the top and left position of an element relative to its offset parent.

scrollLeft( val )

When a value is passed in, the scroll left offset is set to that value on all matched elements.

scrollLeft( )

Gets the scroll left offset of the first matched element.

scrollTop( val )

When a value is passed in, the scroll top offset is set to that value on all matched elements.

scrollTop( )

Gets the scroll top offset of the first matched element.

width( val )

Set the CSS width of every matched element.

width()

Get the current computed, pixel, width of the first matched element.

## jQuery - DOM Manipulation Methods

JQuery provides methods to manipulate DOM in efficient way. You do not need to write big code to modify the value of any element's attribute or to extract HTML code from a paragraph or division.

JQuery provides methods such as .attr(), .html(), and .val() which act as getters, retrieving information from DOM elements for later use.

### Content Manipulation:

The **html( )** method gets the html contents (innerHTML) of the first matched element.

Here is the syntax for the method:

**selector.html()**

#### Example:selector.html()

Following is an example which makes use of .html() and .text(val) methods. Here .html() retrieves HTML content from the object and then .text( val ) method sets value of the object using passed parameter:

```
<html>
<head>
<title>the title</title>
<script type="text/javascript"    src="jquery.js"></script>
<script type="text/javascript" language="javascript">

$(document).ready(function() {

    $("div").click(function () {
        var content = $(this).html();
        $("#result").text( content );
    });

});

</script>
<style>
#division{
    margin:10px;
    padding:12px;
    border:2px solid #666;
    width:60px;
    background-color:blue;
}
```

```
        }
    </style>
</head>
<body>
    <p>Click on the square below:</p>
    <span id="result"> </span>
    <div id="division" >
        This is Blue Square!!
    </div>
</body>
</html>
```

## DOM Element Replacement:

You can replace a complete DOM element with the specified HTML or DOM elements. The **replaceWith( content )** method serves this purpose very well.

Here is the syntax for the method:

```
selector.replaceWith( content )
```

Here content is what you want to have instead of original element. This could be HTML or simple text.

### Example: **selector.replaceWith( content )**

Following is an example which would replace division element with "<h1>JQuery is Great</h1>":

```
<html>
<head>
<title>the title</title>
<script type="text/javascript" src="jquery.js"></script>
<script type="text/javascript" language="javascript">

$(document).ready(function() {

    $("div").click(function () {
        $(this).replaceWith("<h1>JQuery is Great</h1>");
    });

});

</script>
<style>
#division{
    margin:10px;
    padding:12px;
}
```

```

        border:2px solid #666;
        width:60px;
        background-color:blue;
    }
</style>
</head>
<body>
<p>Click on the square below:</p>
<div id="division" >
    This is Blue Square!!
</div>
</body>
</html>

```

## Removing DOM Elements:

There may be a situation when you would like to remove one or more DOM elements from the document. JQuery provides two methods to handle the situation.

The **empty( )** method remove all child nodes from the set of matched elements where as the method **remove( expr )** method removes all matched elements from the DOM.

Here is the syntax for the method:

```

selector.remove( [ expr ] )
or
selector.empty( )

```

You can pass optional parameter *expr* to filter the set of elements to be removed.

### Example: **selector.remove()**

Following is an example where elements are being removed as soon as they are clicked:

```

<html>
<head>
<title>the title</title>
<script type="text/javascript" src="jquery.js"></script>
<script type="text/javascript" language="javascript">

$(document).ready(function() {

    $("div").click(function () {
        $(this).remove();
    });
});

```

```
});  
});  
  
</script>  
<style>  
.myclass{  
    margin:10px;  
    padding:12px;  
    border:2px solid #666;  
    width:60px;  
}  
</style>  
</head>  
<body>  
    <p>Click on any square below:</p>  
    <span id="result"></span>  
    <div class="myclass" style="background-color:blue;"></div>  
    <div class="myclass" style="background-color:green;"></div>  
    <div class="myclass" style="background-color:red;"></div>  
</body>  
</html>
```

## Inserting DOM elements:

There may be a situation when you would like to insert new one or more DOM elements in your existing document. JQuery provides various methods to insert elements at various locations.

The **after( content )** method insert content after each of the matched elements where as the method **before( content )** method inserts content before each of the matched elements.

Here is the syntax for the method:

```
selector.after( content )  
  
or  
  
selector.before( content )
```

Here content is what you want to insert. This could be HTML or simple text.

### Example: **selector.before( content )**

Following is an example where <div> elements are being inserted just before the clicked element:

```
<html>
<head>
<title>the title</title>
<script type="text/javascript"
src="jquery.js"></script>
<script type="text/javascript" language="javascript">

$(document).ready(function() {

    $("div").click(function () {
        $(this).before('<div class="div"></div>');
    });

});

</script>
<style>
.myclass{ margin:10px;padding:12px;
    border:2px solid #666;
    width:60px;
}
</style>
</head>
<body>
<p>Click on any square below:</p>
<span id="result"> </span>
<div class="myclass" style="background-color:blue;"></div>
<div class="myclass" style="background-color:green;"></div>
<div class="myclass" style="background-color:red;"></div>
</body>
</html>
```

### Example: **selector.append( content )**

```
<html>
<head>
<title>the title</title>
<script type="text/javascript"
src="jquery.js"></script>
<script type="text/javascript" language="javascript">

$(document).ready(function() {
    $("div").click(function () {
        $(this).append('<p> this is next paragraph</p>');
    });
});

</script>
<style type="text/css" >
.myclass{
    height:150px;
    width:200px;
    border:2px solid red;
    overflow:auto;
```

```
<style>
.sample{
    margin:10px;
    padding:12px;
    border:2px solid #666;
    width:150px;
}
</style>

</head>
<body>

<p> this is first paragraph </p>

</body>
</html>
```

### Example: *selector.clone( boolean )*

```
<html>
<head>
<title>the title</title>
<script type="text/javascript"
src="jquery.js"></script>
<script type="text/javascript" language="javascript">

$(document).ready(function() {
    $("div").click(function () {
        var divElement = $(this).clone(true);
        $(this).after(divElement);
    });
});

</script>
<style>
.sample{
    margin:10px;
    padding:12px;
    border:2px solid #666;
    width:150px;
    background-color:pink;
}
</style>

</head>
<body>

<div class="sample">
    this is DIV

```

```
</div>

</body>
</html>
```

## DOM Manipulation Methods:

Following table lists down all the methods which you can use to manipulate DOM elements:

Method	Description
<u><a href="#">after( content )</a></u>	Insert content after each of the matched elements.
<u><a href="#">append( content )</a></u>	Append content to the inside of every matched element.
<u><a href="#">appendTo( selector )</a></u>	Append all of the matched elements to another, specified, set of elements.
<u><a href="#">before( content )</a></u>	Insert content before each of the matched elements.
<u><a href="#">clone( bool )</a></u>	Clone matched DOM Elements, and all their event handlers, and select the clones.
<u><a href="#">clone()</a></u>	Clone matched DOM Elements and select the clones.
<u><a href="#">empty()</a></u>	Remove all child nodes from the set of matched elements.
<u><a href="#">html( val )</a></u>	Set the html contents of every matched element.
<u><a href="#">html()</a></u>	Get the html contents (innerHTML) of the first matched element.
<u><a href="#">insertAfter( selector )</a></u>	Insert all of the matched elements after another, specified, set of elements.
<u><a href="#">insertBefore( selector )</a></u>	Insert all of the matched elements before another, specified, set of elements.

<u>prepend( content )</u>	Prepend content to the inside of every matched element.
<u>prependTo( selector )</u>	Prepend all of the matched elements to another, specified, set of elements.
<u>remove( expr )</u>	Removes all matched elements from the DOM.
<u>replaceAll( selector )</u>	Replaces the elements matched by the specified selector with the matched elements.
<u>replaceWith( content )</u>	Replaces all matched elements with the specified HTML or DOM elements.
<u>text( val )</u>	Set the text contents of all matched elements.
<u>text()</u>	Get the combined text contents of all matched elements.
<u>wrap( elem )</u>	Wrap each matched element with the specified element.
<u>wrap( html )</u>	Wrap each matched element with the specified HTML content.
<u>wrapAll( elem )</u>	Wrap all the elements in the matched set into a single wrapper element.
<u>wrapAll( html )</u>	Wrap all the elements in the matched set into a single wrapper element.
<u>wrapInner( elem )</u>	Wrap the inner child contents of each matched element (including text nodes) with a DOM element.
<u>wrapInner( html )</u>	Wrap the inner child contents of each matched element (including text nodes) with an HTML structure.

## JQuery - Events Handling

We have the ability to create dynamic web pages by using events. Events are actions that can be detected by your Web Application.

Following are the examples events:

- A mouse click
- A web page loading
- Taking mouse over an element
- Submitting an HTML form
- A keystroke on your keyboard
- etc.

When these events are triggered you can then use a custom function to do pretty much whatever you want with the event. These custom functions call Event Handlers.

## Binding event handlers:

Using the jQuery Event Model, we can establish event handlers on DOM elements with the **bind()** method as follows:

```
$('div').bind('click', function( event ){
    alert('Hi there!');
});
```

This code will cause the division element to respond to the click event; when a user clicks inside this division thereafter, the alert will be shown.

The full syntax of the bind() command is as follows:

```
selector.bind( eventType[, eventData], handler)
```

Following is the description of the parameters:

- **eventType:** A string containing a JavaScript event type, such as **click** or **submit**. Refer to the next section for a complete list of event types.
- **eventData:** This is optional parameter is a map of data that will be passed to the event handler.
- **handler:** A function to execute each time the event is triggered.

## Removing event handlers:

Typically, once an event handler is established, it remains in effect for the remainder of the life of the page. There may be a need when you would like to remove event handler.

jQuery provides the **unbind()** command to remove an exiting event handler. The syntax of unbind() is as follows:

```
selector.unbind(eventType, handler)
or
```

```
selector.unbind(eventType)
```

Following is the description of the parameters:

- **eventType:** A string containing a JavaScript event type, such as **click** or **submit**. Refer to the next section for a complete list of event types.
- **handler:** If provided, identifies the specific listener that is to be removed.

## Event Types:

The following are cross platform and recommended event types which you can bind using JQuery:

Event Type	Description
blur	Occurs when the element loses focus
change	Occurs when the element changes
click	Occurs when a mouse click
dblclick	Occurs when a mouse double-click
error	Occurs when there is an error in loading or unloading etc.
focus	Occurs when the element gets focus
keydown	Occurs when key is pressed
keypress	Occurs when key is pressed and released
keyup	Occurs when key is released
load	Occurs when document is loaded
mousedown	Occurs when mouse button is pressed
mouseenter	Occurs when mouse enters in an element region

mouseleave	Occurs when mouse leaves an element region
mousemove	Occurs when mouse pointer moves
mouseout	Occurs when mouse pointer moves out of an element
mouseover	Occurs when mouse pointer moves over an element
mouseup	Occurs when mouse button is released
resize	Occurs when window is resized
scroll	Occurs when window is scrolled
select	Occurs when a text is selected
submit	Occurs when form is submitted
unload	Occurs when documents is unloaded

## The Event Object:

The callback function takes a single parameter; when the handler is called the JavaScript event object will be passed through it.

The event object is often unnecessary and the parameter is omitted, as sufficient context is usually available when the handler is bound to know exactly what needs to be done when the handler is triggered, however there are certain attributes which you would need to be accessed.

## The Event Attributes:

The following event properties/attributes are available and safe to access in a platform independent manner:

Property	Description
altKey	Set to true if the Alt key was pressed when the event was triggered, false if not. The Alt key is labeled Option on most Mac keyboards.

ctrlKey	Set to true if the Ctrl key was pressed when the event was triggered, false if not.
data	The value, if any, passed as the second parameter to the bind() command when the handler was established.
keyCode	For keyup and keydown events, this returns the key that was pressed.
metaKey	Set to true if the Meta key was pressed when the event was triggered, false if not. The Meta key is the Ctrl key on PCs and the Command key on Macs.
pageX	For mouse events, specifies the horizontal coordinate of the event relative from the page origin.
pageY	For mouse events, specifies the vertical coordinate of the event relative from the page origin.
relatedTarget	For some mouse events, identifies the element that the cursor left or entered when the event was triggered.
screenX	For mouse events, specifies the horizontal coordinate of the event relative from the screen origin.
screenY	For mouse events, specifies the vertical coordinate of the event relative from the screen origin.
shiftKey	Set to true if the Shift key was pressed when the event was triggered, false if not.
target	Identifies the element for which the event was triggered.
timeStamp	The timestamp (in milliseconds) when the event was created.
Type	For all events, specifies the type of event that was triggered (for example, click).
Which	For keyboard events, specifies the numeric code for the key that caused the event, and for mouse events, specifies which button was pressed (1 for left, 2 for middle, 3 for right)

## Event Helper Methods:

jQuery also provides a set of event helper functions which can be used either to trigger an event or bind any event types mentioned above.

### Trigger Methods:

Following is an example which triggers the blur event on all paragraphs:

```
$("p").blur();
```

### Binding Methods:

Following is an example which binds a click event on all the <div>:

```
$("div").click( function () {  
    // do something here  
});
```

Here is a complete list of all the support methods provided by jQuery:

Method	Description
blur()	Triggers the blur event of each matched element.
blur( fn )	Bind a function to the blur event of each matched element.
change()	Triggers the change event of each matched element.
change( fn )	Binds a function to the change event of each matched element.
click()	Triggers the click event of each matched element.
click( fn )	Binds a function to the click event of each matched element.
dblclick()	Triggers the dblclick event of each matched element.
dblclick( fn )	Binds a function to the dblclick event of each matched element.

error()	Triggers the error event of each matched element.
error( fn )	Binds a function to the error event of each matched element.
focus()	Triggers the focus event of each matched element.
focus( fn )	Binds a function to the focus event of each matched element.
keydown()	Triggers the keydown event of each matched element.
keydown( fn )	Bind a function to the keydown event of each matched element.
keypress()	Triggers the keypress event of each matched element.
keypress( fn )	Binds a function to the keypress event of each matched element.
keyup()	Triggers the keyup event of each matched element.
keyup( fn )	Bind a function to the keyup event of each matched element.
load( fn )	Binds a function to the load event of each matched element.
mousedown( fn )	Binds a function to the mousedown event of each matched element.
mouseenter( fn )	Bind a function to the mouseenter event of each matched element.
mouseleave( fn )	Bind a function to the mouseleave event of each matched element.
mousemove( fn )	Bind a function to the mousemove event of each matched element.
mouseout( fn )	Bind a function to the mouseout event of each matched element.
mouseover( fn )	Bind a function to the mouseover event of each matched element.

mouseup( fn )	Bind a function to the mouseup event of each matched element.
resize( fn )	Bind a function to the resize event of each matched element.
scroll( fn )	Bind a function to the scroll event of each matched element.
select( )	Trigger the select event of each matched element.
select( fn )	Bind a function to the select event of each matched element.
submit( )	Trigger the submit event of each matched element.
submit( fn )	Bind a function to the submit event of each matched element.
unload( fn )	Binds a function to the unload event of each matched element.

## Handling Events

### Handling Events using JavaScript

Question:

What type of JavaScript code do you write to handle a button click event?

Answer:

It depends on the browser!



## Handling Events

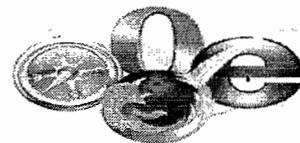
### Event Attachment Techniques

#### Most Browsers:

```
myButton.addEventListener('click', function() { }, false);
```

#### Internet Explorer:

```
myButton.attachEvent('onclick', function() { });
```



## Handling Events

### jQuery Event Model Benefits

- Events notify a program that a user performed some type of action
- jQuery provides a cross-browser event model that works in IE, Chrome, Opera, FireFox, Safari and more
- jQuery event model is simple to use and provides a compact syntax

### Example : Event Handling complexity in Javascript

```
<html>
<head>
<script type="text/javascript" src="jquery.js"></script>
</head>
<body>

<button id="buttonId" >Click</button>

<script type="text/javascript">
    var myButton = document.getElementById("buttonId");
    if(document.addEventListener){
        myButton.addEventListener("click",function(){alert("Click Button");}, false);
    }else{
        myButton.attachEvent("onclick",function(){alert("Click IE Button");});
    }
</script>
```

```

        }
</script>

</body>
</html>
```

**Example : blur(fn) , focus(fn)**

```

<html>
<head>
<script type="text/javascript" src="jquery.js"></script>
<script type="text/javascript">

$(document).ready( function(){

    $("input[type='text']").blur(function(){
        $(this).toggleClass("ybg");
    });

    $("input[type='text']").focus(function(){
        $(this).toggleClass("ybg");
    });
});

</script>
<style type="text/css" >
.ybg{
    background-color:yellow;
}

</style>
</head>
<body>

First Name : <input type="text" name="fname" /> <br/>
Last Name : <input type="text" name="lname" /> <br/>

</body>
</html>
```

**Example : click() , click(fn)**

```

<html>
<head>
<script type="text/javascript" src="jquery.js"></script>
<script type="text/javascript">
```

```

$(document).ready( function(){

    $('#buttonId1').click(function(){
        // raising the event using jquery function
        alert("button1 is clicked");
        $('#buttonId2').click();
    });

    $('#buttonId2').click(function(){
        alert("button2 is clicked");
    });

});

</script>
</head>
<body>

<button id="buttonId1" >Click1</button>
<button id="buttonId2" >Click2</button>

</body>
</html>

```

### Example : change(fn)

```

<html>
<head>
<script type="text/javascript" src="jquery.js"></script>
<script type="text/javascript">
$(document).ready( function(){

    $('#cityId').change(function(){
        $('#result').text($(this).val());
    });

});

</script>
</head>
<body>

<select id="cityId" >
    <option value="hyd">Hyderabad</option>
    <option value="bang">Banglore</option>
    <option value="chennai">Chennai</option>
    <option value="mumbai">Mumbai</option>
</select>
<span id="result" style="color:red"> <span>

```

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

**Example : change(fn)**

```
<html>
<head>
<script type="text/javascript" src="jquery.js"></script>
<script type="text/javascript">
$(document).ready( function(){

    $("#cityId1").change(function(){
        $("#cityId2").val($(this).val());
    });

});

</script>
</head>
<body>

<select id="cityId1" >
    <option value="hyd">Hyderabad</option>
    <option value="bang">Banglore</option>
    <option value="chennai">Chennai</option>
    <option value="mumbai">Mumbai</option>
</select>

<select id="cityId2" >
    <option value="hyd">Hyderabad</option>
    <option value="bang">Banglore</option>
    <option value="chennai">Chennai</option>
    <option value="mumbai">Mumbai</option>
</select>

</body>
</html>
```

**Example : \$(window).unload(fn)**

```
<html>
<head>
<script type="text/javascript" src="jquery.js"></script>
<script type="text/javascript">
$(document).ready( function(){

    $(window).unload(function(){
        alert("Do you want to leave this page");
    });

});
```

```
});  
  
</script>  
</head>  
<body>  
  
<h1> Refresh the page, to call unload functionality</h1>  
  
</body>  
</html>
```

### Example : error(fn)

```
<html>  
<head>  
<script type="text/javascript" src="jquery.js"></script>  
<script type="text/javascript">  
$(document).ready( function(){  
  
    $("img").error(function(){  
        $(this).attr("src", "images/noimage.jpg");  
  
    });  
});  
  
</script>  
</head>  
<body>  
  
 </img>  
  
</body>  
</html>
```

### Example: click(function(event){ })

```
html>  
head>  
style type="text/css">  
    .Hilight{  
        background-color:pink;  
    }  
style>  
<script type="text/javascript" src="jquery.js"></script>  
<script type="text/javascript">  
$(document).ready( function(){
```

```

$( "div#myDiv" ).click(function(event){
    $(this).addClass("Hilight");

    var result="";
    result= result+" X:"+event.pageX+"<br/>";
    result= result+" Y:"+event.pageY+"<br/>";
    result= result+" keyCode:"+event.keyCode+"<br/>";
    result= result+" TimeStamp:"+event.timeStamp+"<br/>";
    result= result+" Target element Id :"+$(event.target).attr("id")+"<br/>";
    result= result+" Event type :" +event.type+"<br/>";
    $(this).html(result);
});

});

</script>
</head>
<body>

<div id="myDiv" style="width:400px;height:200px;border:2px solid black;">
This is my div
This is my div
This is my div
This is my div
</div>

</body>
</html>

```

### Example: mouseenter(fn), mouseleave(fn)

```

<html>
<head>
<style type="text/css">
.Hilight{
    background-color:pink;
}
</style>
<script type="text/javascript" src="jquery.js"></script>
<script type="text/javascript">
$(document).ready( function(){

    $("div#myDiv").mouseenter(function(){
        $(this).toggleClass("Hilight");
    }).mouseleave(function(){
        $(this).toggleClass("Hilight");
    }).click(function(event){
        $(this).text("X : "+event.pageX + " Y:"+event.pageY);
    });
});

```

```
});  
});  
  
</script>  
</head>  
<body>  
  
<div id="myDiv" style="width:100px;height:100px;border:2px solid black;">  
    This is my div  
    This is my div  
    This is my div  
    This is my div  
</div>  
  
</body>  
</html>
```

## Handling Events

### Using bind()

- **.bind(eventType, handler(eventObject)) attaches a handler to an event for the selected element(s)**

```
$('#MyDiv').bind('click', function() {  
    //Handle click event  
});
```

**.click() is the same as .bind('click')**

## Handling Events

### Using unbind()

- **.unbind(event) is used to remove a handler previously bound to an element:**

```
$('#test').click(handler); can be unbound using  
$("#test").unbind();
```

- **Specific events can also be targeted using unbind():**

```
$('#test').unbind('click');
```

## Handling Events

### Binding Multiple Events

- **bind() allows multiple events to be bound to one or more elements**
- **Event names to bind are separated with a space:**

```
$('#MyDiv').bind('mouseenter mouseleave',  
    function() {  
        $(this).toggleClass('entered');  
    }  
);
```

- **Example: bind(), unbind()**

```
<html>  
<head>  
<style type="text/css">  
    .Hilight{  
        background-color:pink;  
    }  
</style>  
<script type="text/javascript" src="jquery.js"></script>  
<script type="text/javascript">  
$(document).ready( function(){  
  
    $("div#myDiv").bind("mouseenter mouseleave click",function(event){  
        $(this).html(event.type);  
    })  
});
```

```

});
```

```

$("button").click(function() {
    $("div#myDiv").unbind("click");
});
```

```

});
```

```
</script>
```

```
</head>
```

```
<body>
```

```
<button> Retire Click event </button>
```

```
<div id="myDiv" style="width:400px;height:200px;border:2px solid black;">
```

```
This is my div
```

```
</div>
```

```
</body>
```

```
</html>
```

### Example: bind(), unbind()

```
<html>
```

```
<head>
```

```
<style type="text/css">
```

```
    .Hilight{
        background-color:pink;
    }

```

```
</style>
```

```
<script type="text/javascript" src="jquery.js"></script>
```

```
<script type="text/javascript">
```

```
$(document).ready( function(){
```

```

    $("div#myDiv").bind("mouseenter mouseleave click", function(event){
        if(event.type=='click'){
            $(this).text("X : "+event.pageX + " Y:"+event.pageY);
        }

        if(event.type=='mouseenter' || event.type=='mouseleave'){
            $(this).toggleClass("Hilight");
        }
    });
}
```

```
    $("button").click(function(){
```

```

        //$("#myDiv").unbind();
        $("#myDiv").unbind("click");
    });

});

</script>
</head>
<body>

<div id="myDiv" style="width:100px;height:100px;border:2px solid black;">
This is my div
</div>
<button>Unbind event</button>
</body>
</html>

```

Key events occur in the following order:

1. KeyDown
2. KeyPress
3. KeyUp

**NOTE:** In order to understand the difference between keydown and keypress, it is useful to understand the difference between a "character" and a "key". A "key" is a physical button on the computer's keyboard while a "character" is a symbol typed by pressing a button. In theory, the keydown and keyup events represent keys being pressed or released, while the keypress event represents a character being typed. The implementation of the theory is not same in all browsers.

```

<html>
<head>
<script type="text/javascript" src="jquery.js"></script>
<script type="text/javascript">
$(document).ready( function(){

    var down=0;
    var up=0;
    var press=0;

    $("input#phonelId").bind("keydown keypress keyup", function(event){
        if(event.type=="keydown"){
            $("#downId").text("Keydown : "+(++down));
        }

        if(event.type=="keyup"){
            $("#upId").text("keyup : "+(--up));
        }

        if(event.type=="keypress"){
            $("#pressId").text("Press : "+(press++));
        }
    });
});

```

```

        $("#upId").text("keyup : "+(++up));
    }

    if(event.type=="keypress"){
        $("#pressId").text("keypress : "+(++press));
    }
});

});

</script>
</head>
<body>
    Phone : <input type="text" id="phoneId" name="phone" />
    <br/>
    <span id="downId" style="color:red" ></span> <br/>
    <span id="pressId" style="color:red" ></span> <br/>
    <span id="upId" style="color:red" ></span>
</body>
</html>

```

**Example: mousedown(fn), mouseup(fn)**

```

<html>
<head>
    <script src="http://code.jquery.com/jquery-latest.js"></script>
</head>
<body>
    <p>Press mouse and release here.</p>

    <script>
        $("p").mouseup(function(){
            $(this).append('<span style="color:blue;">Mouse up.</span>');
        }).mousedown(function(){
            $(this).append('<span style="color:red;">Mouse down.</span>');
        });
    </script>

</body>
</html>

```

**Example: mouseenter(fn), mouseleave(fn)**

```

<html>
<head>
<script type="text/javascript" src="jquery.js"></script>
<script type="text/javascript">
$(document).ready(function(){
    $("div").mouseenter(function(){

```

```
$this.css("background-color","yellow");
});
});
$(div).mouseleave(function(){
 $(this).css("background-color","pink");
});
});
</script>
<style>
.myclass{
 height:50px;
 width:200px;
 border:2px solid red;
}
</style>
</head>
<body>
<div class="myclass">
Hover the mouse pointer over this paragraph.
</div>
</body>
</html>
```

### Example: mouseover(fn), mouseout(fn)

```
<html>
<head>
<script type="text/javascript" src="jquery.js"></script>
<script type="text/javascript">
$(document).ready(function(){
 $("div").mouseover(function(){
 $("div").css("background-color","yellow");
});
 $("div").mouseout(function(){
 $("div").css("background-color","pink");
});
});
</script>
<style>
.myclass{
 height:50px;
 width:200px;
 border:2px solid red;
}
</style>
</head>
<body>
<div class="myclass">
Hover the mouse pointer over this paragraph.
</div>
</body>
```

```
</html>
```

**Example:Difference among mouseover(fn), mouseout(fn), mouseenter(fn),mouseleave(fn)**

```
<html>
<head>
<script type="text/javascript" src="jquery.js"></script>
<script type="text/javascript">
var overCount=0;
var outCount=0;
var enterCount=0;
var leaveCount=0;
$(document).ready(function(){

    $("div#onelId").mouseover(function(){
        $("span#overResult").text(++overCount);

    });

    $("div#onelId").mouseout(function(){
        $("span#outResult").text(++outCount);

    });

    $("div#twolId").mouseenter(function(){
        $("span#enterResult").text(++enterCount);
    });

    $("div#twolId").mouseleave(function(){
        $("span#leaveResult").text(++leaveCount);
    });

});
</script>
<style>
div.out {
    width:45%;
    height:120px;
    margin:0 15px;
    background-color:#D6EDFC;
    float:left;
}
div.in {
    width:60%;
    height:60%;
    background-color:#FFCC00;
    margin:10px auto;
    font-size:25px;
}
```

```

</style>
</head>
<body>
    <h1>The mouseover() event triggers if a mouse pointer enters the child elements as well as the selected element.</h1>
    <h1>The mouseenter() event is only triggered when the the mouse pointer enters the selected element. </h1>

    <div class="out" id="onelid" >
        <div class="in" >
            Mouseover: <span id="overResult" ></span> <br/>
            Mouseout: <span id="outResult" ></span>
        </div>
    </div>

    <div class="out" id="twolid" >
        <div class="in" >
            Mouseenter : <span id="enterResult" ></span> <br/>
            Mouseleave : <span id="leaveResult" ></span>
        </div>
    </div>

</body>
</html>

```

**Example: mousemove(fn)**

```

<html>
<head>
<script type="text/javascript" src="jquery.js"></script>
<script type="text/javascript">
$(document).ready(function(){
    $(document).mousemove(function(e){
        $("span").text(e.pageX + ", " + e.pageY);
    });
});
</script>
</head>
<body>
<h1>Mouse is at coordinates: <span></span>.</h1>
</body>
</html>

```

**Example: Selector of some other selector**

```

<html>
<head>
<script type="text/javascript" src="jquery.js"></script>
<script type="text/javascript">
$(document).ready(function(){

```

```
$("button").click(function(){
    //var oneSelector = $("div#oneId");
    //$("#div.myclass", oneSelector).css("background-color", "pink");

    var twoSelector = $("div#twold");
    $("div.myclass", twoSelector).css("background-color", "pink");
});

});
</script>
<style type="text/css" >
.myclass{
    width: 200px;
    height: 100px;
    border: 2px solid green;
    margin : 20px;
}
</style>
</head>
<body>

<div id="oneId" >
<div class="myclass" >
    DIV ONE
</div>
</div>

<div id="twold" >
<div class="myclass" >
    DIV TWO
</div>
</div>

<button> Click </button>

</body>
</html>
```

## Handling Events

### live() and delegate() Functions

- live() and delegate() allow new elements added into the DOM to automatically be "attached" to an event handler

live() – Allows binding of event handlers to elements that match a selector, including future elements. Events bubble up to the document object.

delegate() – Replacement for live() in jQuery 1.4. Attaches an event handler directly to the selector context.

## Handling Events

### Using live()

- Event handlers can be set using live()
- The document object handles events by default
- Works even when new objects are added into the DOM:

```
$('.someClass').live('click',  
    someFunction);
```

Any element added  
with .someClass will  
have a click event

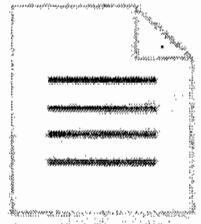
- Stop live event handling using die():

```
$('.someClass').die('click', someFunction);
```

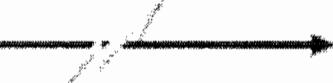
## Handling Events

### How live() Works

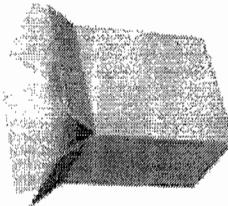
Document Object



Click Event



Event Handler



Click Event

```
<span class="someClass"/>
  <p class="someClass"/> ← $(.someClass).live('click',function() {});
    <div class="someClass"/>
```

## Handling Events

### Using delegate()

- Newer version of live() added in jQuery 1.4
- A context object (#Divs in the sample below) handles events by default rather than the document object
- Works even when new objects are added into the DOM:

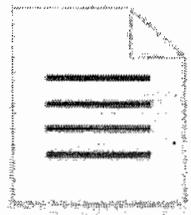
```
$('#Divs').delegate('div','click',someFunction);
```

- Stop delegate event handling using undelegate()

## Handling Events

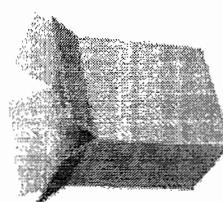
### How delegate() Works

Context Object: #Divs



ClickEvent

Event Handler



ClickEvent

```
<span class="someClass"/>
<p class="someClass"/>
<div class="someClass"/>
```

←      `$( '#Divs' ).delegate( '.someClass', 'click', func() {} );`

### Example: live(fn), die(fn), delegate(fn), undelegate(fn)

```
<html>
<head>
<script type="text/javascript" src="jquery.js"></script>
<script type="text/javascript">
$(document).ready( function(){
    var PhoneDiv = $("div#PhoneDiv");

    $("#AddPhoneButton").click(function(){
        var PhoneTypeDivClone= $("div:eq(0)",PhoneDiv).clone();
        var PhoneNumberDivClone= $("div:eq(1)",PhoneDiv).clone();
        $("select", PhoneTypeDivClone).val("");
        $("input", PhoneNumberDivClone).val("");
        PhoneDiv.append("<div style='clear:both' />").append(PhoneTypeDivClone)
            .append(PhoneNumberDivClone);
    });

    $("input", PhoneDiv).live("keypress", function(event){
        if(event.keyCode<48 || event.keyCode>57){
            //event.stopPropagation();
            return false;
        }
    });
});
```

```
<div id="PhoneTypeDiv" style="float:left" >
    <select name="PhoneType" >
        <option value="">Select One</option>
        <option value="Mobile">Mobile</option>
        <option value="Office">Office</option>
        <option value="Home">Home</option>
    </select>
</div>
<div id="PhoneNumberDiv" style="float:left" >
    <input type="text" name="PhoneNumber"/>
</div>
</div>
</td>
</tr>
<tr>
    <td>City</td>
    <td>
        <select id="cityId" >
            <option value="hyd">Hyderabad</option>
            <option value="bang">Banglore</option>
            <option value="chennai">Chennai</option>
            <option value="mumbai">Mumbai</option>
        </select>
    </td>
</tr>
<tr>
    <td colspan="2" align="center" >
        <input type="submit" value="Submit"/>
    </td>
</tr>

</table>
<button>Release Event Handlers</button>
</body>
</html>
```

## Handling Events

### Handling Hover Events

- Hover events can be handled using `hover()`:

```
$(selector).hover(handlerIn, handlerOut)
```

- `handlerIn` is equivalent to `mouseenter` and `handlerOut` is equivalent to `mouseleave`

## Handling Events

### Using `hover()`

- This example highlights `#target` on `mouseenter` and sets it back to white on `mouseleave`

```
$('#target').hover(
    function(){
        $(this).css('background-color', '#00FF99');
    },
    function(){
        $(this).css('background-color', '#FFFFFF');
    }
);
```

## Handling Events

### Alternate Hover Example

- Another option is `$(selector).hover(handlerInOut)`
- Fires the same handler for `mouseenter` and `mouseleave` events
- Used with jQuery's toggle methods:

```
$('p').hover(function() {
    $(this).toggleClass('over');
});
```

This code will toggle the class applied to a paragraph element

### Example: hover(fn)

```
<html>
<head>
<style type="text/css" >
.Hilight{
    background-color:yellow;
}

table, td, th{
    border-collapse:collapse;
    border : 2px solid green;
    width : 200px;
}

</style>
<script type="text/javascript" src="jquery.js"></script>
<script type="text/javascript">
$(document).ready( function(){
/*
    $("table#MyTable tr").hover(
        function(){
            // mouseenter
            $(this).css("background-color","#efefef");
        },
        function(){
            // mouseleave
            $(this).css("background-color","#ffffff");
        }
    );
*/
})
```

```
$("table#MyTable tr").hover(function(){
    $(this).toggleClass("Hilight");
});
</script>
<style type="text/css" >
</style>
</head>
<body>

<table id="MyTable" >
    <tr>
        <th>ENO</th>
        <th>Name</th>
        <th>Sal</th>
    </tr>
    <tr>
        <td>1001</td>
        <td>sekhar1</td>
        <td>459.09</td>
    </tr>
    <tr>
        <td>1002</td>
        <td>sekhar2</td>
        <td>459.09</td>
    </tr>
    <tr>
        <td>1003</td>
        <td>sekhar3</td>
        <td>459.09</td>
    </tr>
    <tr>
        <td>1004</td>
        <td>sekhar4</td>
        <td>459.09</td>
    </tr>
    <tr>
        <td>1005</td>
        <td>sekhar5</td>
        <td>459.09</td>
    </tr>
</table>

</body>
</html>
```

**Example: hover(fn, fn, fn, fn)**

```
<html>
<head>
<style type="text/css" >
.Hilight{
    background-color:#efefef;
}
table, td, th{
    border-collapse:collapse;
    border : 2px solid green;
    width : 200px;
}

</style>
<script type="text/javascript" src="jquery.js"></script>
<script type="text/javascript">
$(document).ready( function(){

$("table#MyTable tr").toggle(
    function(){
        // click
        $(this).css("background-color","#efefef");
    },
    function(){
        // click
        $(this).css("background-color","purple");
    },
    function(){
        // click
        $(this).css("background-color","yellow");
    },
    function(){
        // click
        $(this).css("background-color","pink");
    }
);

});
</script>
</head>
<body>

<table id="MyTable" border="1" >
<tr>
    <th>ENO</th>
    <th>Name</th>
    <th>Sal</th>
</tr>
```

```
<tr>
    <td>1001</td>
    <td>sekhar1</td>
    <td>459.09</td>
</tr>
<tr>
    <td>1002</td>
    <td>sekhar2</td>
    <td>459.09</td>
</tr>
<tr>
    <td>1003</td>
    <td>sekhar3</td>
    <td>459.09</td>
</tr>
</table>
</body>
</html>
```

## JQuery - Effects

JQuery provides a trivially simple interface for doing various kind of amazing effects. JQuery methods allow us to quickly apply commonly used effects with a minimum configuration.

### JQuery Effect Methods:

You have seen basic concept of jQuery Effects. Following table lists down all the important methods to create different kind of effects:

#### JQuery Hide and Show

With jQuery, you can hide and show HTML elements with the hide() and show() methods:

#### Example: hide(), show()

```
<html>
<head>
<script type="text/javascript" src="jquery.js"></script>
<script type="text/javascript">
$(document).ready(function(){
    $("#hide").click(function(){
        $("h1").hide();
    });
    $("#show").click(function(){
        $("h1").show();
    });
});
```

```
});  
});  
</script>  
</head>  
<body>  
<h1>If you click on the "Hide" button, I will disappear. <br/>  
If you click on the "Show" button, I will appear </h1>  
<button id="hide">Hide</button>  
<button id="show">Show</button>  
</body>  
</html>
```

Both hide() and show() can take the two optional parameters: speed and callback.

Syntax:

**\$(selector).hide(speed,callback)**

**\$(selector).show(speed,callback)**

The speed parameter specifies the speed of the hiding/showing, and can take the following values: "slow", "fast", "normal", or milliseconds:

### Example: hide(time)

```
<html>  
<head>  
<script type="text/javascript" src="jquery.js"></script>  
<script type="text/javascript">  
$(document).ready(function(){  
    $("button").click(function(){  
        $("p").hide(1000);  
    });  
});  
</script>  
</head>  
<body>  
<button>Hide</button>  
<p>This is a paragraph with little content.</p>  
<p>This is another small paragraph.</p>  
</body>  
</html>
```

**Example: hide("slow")**

```
<html>
<head>
<script type="text/javascript" src="jquery.js"></script>
<script type="text/javascript">
$(document).ready(function(){
    $(".ex .hide").click(function(){
        $(this).parents(".ex").hide("slow");
    });
});
</script>
<style type="text/css">
div.ex
{
    background-color:#e5eecc;
    padding:7px;
    border:solid 1px #c3c3c3;
}
</style>
</head>
<body>

<h3>Island Trading</h3>
<div class="ex">
    <button class="hide">Hide me</button>
    <p>Contact: Helen Bennett<br />
       Garden House Crowther Way<br />
       London</p>
</div>

<h3>Paris spécialités</h3>
<div class="ex">
    <button class="hide">Hide me</button>
    <p>Contact: Marie Bertrand<br />
       265, Boulevard Charonne<br />
       Paris</p>
</div>

</body>
</html>
```

The callback parameter is the name of a function to be executed after the hide (or show) function completes.

**jQuery Toggle**

The jQuery toggle() method toggles the visibility of HTML elements using the show() or hide() methods.

Shown elements are hidden and hidden elements are shown.

Syntax:

**`$(selector).toggle(speed,callback)`**

The speed parameter can take the following values: "slow", "fast", "normal", or milliseconds.

### **Example: toggle()**

```
<html>
<head>
<script type="text/javascript" src="jquery.js"></script>
<script type="text/javascript">
$(document).ready(function(){
    $("button").click(function(){
        $("p").toggle();
    });
});
</script>
</head>
<body>

<button>Toggle</button>
<p>This is a paragraph with little content.</p>
<p>This is another small paragraph.</p>
</body>
</html>
```

The callback parameter is the name of a function to be executed after the hide (or show) method completes.

### **jQuery Slide - slideDown, slideUp, slideToggle**

The jQuery slide methods gradually change the height for selected elements.

jQuery has the following slide methods:

**`$(selector).slideDown(speed,callback)`**

**`$(selector).slideUp(speed,callback)`**

**`$(selector).slideToggle(speed,callback)`**

The speed parameter can take the following values: "slow", "fast", "normal", or milliseconds.

The callback parameter is the name of a function to be executed after the function completes.

### **Example : slideDown(), slideUp(), slideToggle()**

```
<html>
```

```
<head>

<script type="text/javascript" src="jquery.js"></script>
<script type="text/javascript">
$(document).ready(function(){

    $(".flip").click(function(){
        $(".panel").slideDown("slow");
    });
/*
    $(".flip").click(function(){
        $(".panel").slideUp("slow");
    });
*/
/*
    $(".flip").click(function(){
        $(".panel").slideToggle("slow");
    });
*/
});
</script>

<style type="text/css">
    div.panel,p.flip
    {
        margin:0px;
        padding:5px;
        text-align:center;
        background:#e5eecc;
        border:solid 1px #c3c3c3;
    }
    div.panel
    {
        height:120px;
        display:none;
    }
</style>
</head>

<body>

<div class="panel">
    <p>Because time is valuable, we deliver quick and easy learning.</p>
    <p>At Sekharit, you can study everything you need to learn, in an accessible and handy format.</p>
</div>

<p class="flip">Show/Hide Panel</p>

</body>
```

```
</html>
```

## jQuery Fade - fadeIn, fadeOut, fadeTo

The jQuery fade methods gradually change the opacity for selected elements.

jQuery has the following fade methods:

**`$(selector).fadeIn(speed,callback)`**

**`$(selector).fadeOut(speed,callback)`**

**`$(selector).fadeTo(speed,opacity,callback)`**

The speed parameter can take the following values: "slow", "fast", "normal", or milliseconds.

The opacity parameter in the fadeTo() method allows fading to a given opacity.

The callback parameter is the name of a function to be executed after the function completes.

### Example : fadeIn(), fadeOut(), fadeTo()

```
<html>
<head>
<script type="text/javascript" src="jquery.js"></script>
<script type="text/javascript">
$(document).ready(function(){
    $("button#fadeOutId").click(function(){
        $("div#myId").fadeOut(4000);
    });

    $("button#fadeInId").click(function(){
        $("div#myId").fadeIn(4000);
    });

    $("button#fadeTold").click(function(){
        $("div#myId").fadeTo("slow",0.25);
    });
});
</script>
</head>

<body>
<div id="myId" style="background:green;width:300px;height:300px">ME AWAY!</div>
<button id="fadeOutId" > Fade Out </button>
<button id="fadeInId" > Fade In </button>
```

```
<button id="fadeToId" > Fade To </button>
</body>

</html>
```

## jQuery Custom Animations

The syntax of jQuery's method for making custom animations is:

```
$(selector).animate({params},[duration],[easing],[callback])
```

The key parameter is **params**. It defines the CSS properties that will be animated. Many properties can be animated at the same time:

```
animate({width:"70%",opacity:0.4,marginLeft:"0.6in",fontSize:"3em"});
```

The second parameter is **duration**. It specifies the speed of the animation. Possible values are "fast", "slow", "normal", or milliseconds.

**NOTE:** Only numeric values can be animated (like "margin:30px"). String values cannot be animated (like "background-color:red").

**NOTE:** The styles are set with DOM names (like "fontSize"), not CSS names (like "font-size").

### Example : animate()

```
<html>
<head>
<script type="text/javascript" src="jquery.js"></script>
<script type="text/javascript">
$(document).ready(function(){
  $("button").click(function(){
    $("div").animate({height:300}, "slow");
    $("div").animate({width:300}, "slow");
    $("div").animate({height:100}, "slow");
    $("div").animate({width:100}, "slow");
  });
});
</script>
</head>

<body>
<button>Start Animation</button>
<br /><br />
<div style="background:#98bf21;height:100px;width:100px;position:relative">
</div>
</body>
</html>
```

### Example : animate()

```
<html>
<head>
<script type="text/javascript" src="jquery.js"></script>
<script type="text/javascript">
$(document).ready(function(){
    $("button").click(function(){
        $("div").animate({left:"100px"}, "slow");
        $("div").animate({fontSize:"3em", borderWidth: "10px" }, "slow", function() {
            $(this).css("background-color", "yellow");
        });
    });
});
</script>
<style>
div {
    background-color:#98bf21;
    width:200px;
    height:100px;
    border:1px solid green;
    position:relative;
}
</style>
</head>
<body>

<button>Start Animation</button>
<br /><br />
<div>HELLO</div>
</body>
</html>
```

**NOTE:** HTML elements are positioned static by default and cannot be moved.

To make elements moveable, set the CSS position property to fixed, relative or absolute.

### Methods and Description

#### animate( params, [duration, easing, callback] )

A function for making custom animations.

#### fadeIn( speed, [callback] )

Fade in all matched elements by adjusting their opacity and firing an optional callback after completion.

[fadeOut\( speed, \[callback\] \)](#)

Fade out all matched elements by adjusting their opacity to 0, then setting display to "none" and firing an optional callback after completion.

[fadeTo\( speed, opacity, callback \)](#)

Fade the opacity of all matched elements to a specified opacity and firing an optional callback after completion.

[hide\(\)](#)

Hides each of the set of matched elements if they are shown.

[hide\( speed, \[callback\] \)](#)

Hide all matched elements using a graceful animation and firing an optional callback after completion.

[show\(\)](#)

Displays each of the set of matched elements if they are hidden.

[show\( speed, \[callback\] \)](#)

Show all matched elements using a graceful animation and firing an optional callback after completion.

[slideDown\( speed, \[callback\] \)](#)

Reveal all matched elements by adjusting their height and firing an optional callback after completion.

[slideToggle\( speed, \[callback\] \)](#)

Toggle the visibility of all matched elements by adjusting their height and firing an optional callback after completion.

[slideUp\( speed, \[callback\] \)](#)

Hide all matched elements by adjusting their height and firing an optional callback after completion.

[stop\( \[clearQueue, gotoEnd \] \)](#)

Stops all the currently running animations on all the specified elements.

[toggle\(\)](#)

Toggle displaying each of the set of matched elements.

[toggle\( speed, \[callback\] \)](#)

Toggle displaying each of the set of matched elements using a graceful animation and firing an optional callback after

completion.

#### toggle( switch )

Toggle displaying each of the set of matched elements based upon the switch (true shows all elements, false hides all elements).

#### jQuery.fx.off

Globally disable all animations.

## **jQuery - AJAX**

AJAX is an acronym standing for Asynchronous JavaScript and XML and this technology help us to load data from the server without a browser page refresh.

JQuery is a great tool which provides a rich set of AJAX methods to develop next generation web application.

### **Loading simple data:**

This is very easy to load any static or dynamic data using JQuery AJAX. JQuery provides **load()** method to do the job:

#### **Syntax:**

Here is the simple syntax for **load()** method:

```
[selector].load( URL, [data], [callback] );
```

Here is the description of all the parameters:

- **URL:** The URL of the server-side resource to which the request is sent. It could be a CGI, ASP, JSP, or PHP script which generates data dynamically or out of a database.
- **data:** This optional parameter represents an object whose properties are serialized into properly encoded parameters to be passed to the request. If specified, the request is made using the **POST** method. If omitted, the **GET** method is used.
- **callback:** A callback function invoked after the response data has been loaded into the elements of the matched set. The first parameter passed to this function is the response text received from the server and second parameter is the status code.

#### **Example:**

Consider the following HTML file with a small JQuery coding:

```
<html>
<head>
<title>the title</title>
<script type="text/javascript" src="jquery.js"></script>
<script type="text/javascript" language="javascript">
$(document).ready(function() {
    $("#driver").click(function(event){
        $('#stage').load('/jquery/result.html');
    });
});
</script>
</head>
<body>
<p>Click on the button to load result.html file:</p>
<div id="stage" style="background-color:blue;">
    STAGE
</div>
<input type="button" id="driver" value="Load Data" />
</body>
</html>
```

Here **load()** initiates an Ajax request to the specified URL **/jquery/result.html** file. After loading this file, all the content would be populated inside **<div>** tagged with ID *stage*. Assuming, our **/jquery/result.html** file has just one HTML line:

```
<h1>THIS IS RESULT...</h1>
```

When you click the given button, then **result.html** file gets loaded.

## Getting JSON data:

There would be a situation when server would return JSON string against your request. JQuery utility function **getJSON()** parses the returned JSON string and makes the resulting string available to the callback function as first parameter to take further action.

### Syntax:

Here is the simple syntax for **getJSON()** method:

```
[selector].getJSON( URL, [data], [callback] );
```

Here is the description of all the parameters:

- **URL:** The URL of the server-side resource contacted via the GET method.
- **data:** An object whose properties serve as the name/value pairs used to construct a query string to be appended to the URL, or a preformatted and encoded query string.

- **callback:** A function invoked when the request completes. The data value resulting from digesting the response body as a JSON string is passed as the first parameter to this callback, and the status as the second.

### Example:

Consider the following HTML file with a small JQuery coding:

```
<html>
<head>
<title>the title</title>
<script type="text/javascript"
src="jquery.js"></script>
<script type="text/javascript" language="javascript">
$(document).ready(function() {
    $("#driver").click(function(event){
        $.getJSON('/jquery/result.json', function(jd) {
            $('#stage').html('<p> Name: ' + jd.name + '</p>');
            $('#stage').append('<p>Age : ' + jd.age+ '</p>');
            $('#stage').append('<p> Sex: ' + jd.sex+ '</p>');
        });
    });
    </script>
</head>
<body>
<p>Click on the button to load result.html file:</p>
<div id="stage" style="background-color:blue;">
    STAGE
</div>
<input type="button" id="driver" value="Load Data" />
</body>
</html>
```

Here JQuery utility method **getJSON()** initiates an Ajax request to the specified URL **/jquery/result.json** file. After loading this file, all the content would be passed to the callback function which finally would be populated inside **<div>** tagged with ID *stage*. Assuming, our **/jquery/result.json** file has following json formatted content:

```
{
"name": "Zara Ali",
"age" : "67",
"sex": "female"
}
```

When you click the given button, then result.json file gets loaded.

## Passing data to the Server:

Many times you collect input from the user and you pass that input to the server for further processing. JQuery AJAX made it easy enough to pass collected data to the server using **data** parameter of any available Ajax method.

### Example:

This example demonstrate how can pass user input to a web server script which would send the same result back and we would print it:

```
<html>
<head>
<title>the title</title>
<script type="text/javascript">
src="jquery.js"></script>
<script type="text/javascript" language="javascript">
$(document).ready(function() {
    $("#driver").click(function(event){
        var name = $("#name").val();
        $("#stage").load('/jquery/result.php', {"name":name} );
    });
});
</script>
</head>
<body>
<p>Enter your name and click on the button:</p>
<input type="input" id="name" size="40" /><br />
<div id="stage" style="background-color:blue;">
    STAGE
</div>
<input type="button" id="driver" value="Show Result" />
</body>
</html>
```

Here is the code written in **result.php** script:

```
<?php
if( $_REQUEST["name"] )
{
    $name = $_REQUEST['name'];
    echo "Welcome ". $name;
}
?>
```

Now you can enter any text in the given input box and then click "Show Result" button to see what you have entered in the input box.

## JQuery AJAX Methods:

You have seen basic concept of AJAX using JQuery. Following table lists down all important JQuery AJAX methods which you can use based your programming need:

### Methods and Description

#### [jQuery.ajax\( options \)](#)

Load a remote page using an HTTP request.

#### [jQuery.ajaxSetup\( options \)](#)

Setup global settings for AJAX requests.

#### [jQuery.get\( url, \[data\], \[callback\], \[type\] \)](#)

Load a remote page using an HTTP GET request.

#### [jQuery.getJSON\( url, \[data\], \[callback\] \)](#)

Load JSON data using an HTTP GET request.

#### [jQuery.getScript\( url, \[callback\] \)](#)

Loads and executes a JavaScript file using an HTTP GET request.

#### [jQuery.post\( url, \[data\], \[callback\], \[type\] \)](#)

Load a remote page using an HTTP POST request.

#### [load\( url, \[data\], \[callback\] \)](#)

Load HTML from a remote file and inject it into the DOM.

#### [serialize\(\)](#)

Serializes a set of input elements into a string of data.

#### [serializeArray\(\)](#)

Serializes all forms and form elements like the .serialize() method but returns a JSON data structure for you to work with.

## JQuery AJAX Events:

You can call various JQuery methods during the life cycle of AJAX call progress. Based on different events/stages following methods are available:

You can go through all the AJAX Events.

Methods and Description
<b><u>ajaxComplete( callback )</u></b>
Attach a function to be executed whenever an AJAX request completes.
<b><u>ajaxStart( callback )</u></b>
Attach a function to be executed whenever an AJAX request begins and there is none already active.
<b><u>ajaxError( callback )</u></b>
Attach a function to be executed whenever an AJAX request fails.
<b><u>ajaxSend( callback )</u></b>
Attach a function to be executed before an AJAX request is sent.
<b><u>ajaxStop( callback )</u></b>
Attach a function to be executed whenever all AJAX requests have ended.
<b><u>ajaxSuccess( callback )</u></b>
Attach a function to be executed whenever an AJAX request completes successfully.

## Working with Ajax Features

### Agenda

- **jQuery AJAX Functions**
- Loading HTML Content from the Server
- Making GET Requests
- Making POST Requests
- Introduction to the ajax() Function

## Working with Ajax Features

### jQuery Ajax Features

- **jQuery allows Ajax requests to be made from a page:**
  - Allows parts of a page to be updated
  - Cross-Browser Support
  - Simple API
  - GET and POST supported
  - Load JSON, XML, HTML or even scripts



NOTE: In the case of javascript, to implement ajax functionality first we need to identify the browser as follows. So javascript ajax functionality is browser dependent.

```
var xmlhttp = null;
If (window.XMLHttpRequest) {
    // If IE7, Mozilla, Safari, and so on: Use native object.
    xmlhttp = new XMLHttpRequest();
}
else
{
    if (window.ActiveXObject) {
        // ...otherwise, use the ActiveX control for IE5.x and IE6.
        xmlhttp = new ActiveXObject('MSXML2.XMLHTTP.3.0');
    }
}
```

## **jQuery Ajax Functions**

- **jQuery provides several functions that can be used to send and receive data:**
  - `$(selector).load()`: Loads HTML data from the server
  - `$.get()` and `$.post()`: Get raw data from the server
  - `$.getJSON()`: Get/Post and return JSON
  - `$.ajax()`: Provides core functionality
- **jQuery Ajax functions work with REST APIs, Web Services and more**

## Using load()

- `$(selector).load(url,data,callback)` allows HTML content to be loaded from a server and added into a DOM object:

```
$(document).ready(function(){
    $('#HelpButton').click(function(){
        $('#MyDiv').load('HelpDetails.html');
    });
});
```

## Using load() With a Selector

- A selector can be added after the URL to filter the content that is returned from calling `load()`:

```
$('#MyDiv').load('HelpDetails.html #MainTOC');
```

## Passing Data using load()

- Data can be passed to the server using `load(url,data)`:

```
$('#MyDiv').load('GetCustomers.aspx',
    {PageSize:25});
```

## Using a Callback Function with load()

- **load() can be passed a callback function:**

```
$('#OutputDiv').load('NotFound.html',
    function (response, status, xhr) {
        if (status == "error") {
            alert(xhr.statusText);
        }
    });
});
```

```
<script type="text/javascript">
$(document).ready(function () {
    $('#HelpButton').click(function () {
        $('#OutputDiv').load('NotFound.html',
            function (response, status, xhr) {
                if (status == "error") {
                    alert('Error: ' + xhr.statusText);
                }
            }));
    });
});
</script>
```

## Using get()

- `$.get(url,data,callback,datatype)` can retrieve data from a server:

```
$.get('HelpDetails.html', function (data) {  
    $('#OutputDiv').html(data);  
});  
  
<script type="text/javascript">  
$(document).ready(function () {  
    $('#MyButton').click(function () {  
        $.get('../HelpDetails.html', function (data) {  
            $('#OutputDiv').html(data);  
        });  
  
        $.get('../CustomerJson.aspx', { id: 5 }, function (data) {  
            alert('ID: ' + data.ID + ' ' +  
                  data.FirstName + ' ' + data.LastName);  
        }, 'json');  
    });  
});  
</script>
```

```
<script type="text/javascript">
$(document).ready(function () {
    $('#MyButton').click(function () {
        $.getJSON('../CustomerJson.aspx', { id: 5 },
            function (data) {
                alert('ID: ' + data.ID + ' ' +
                    data.FirstName + ' ' + data.LastName);
            });
    });
});
</script>
```

## Using post()

- **\$.post(url,data,callback,datatype)** can post data to a server and retrieve results:

```
$.post('GetCustomers.aspx', { PageSize:15 },
    function (data) {
        $('#OutputDiv').html(data);
    }
);
```

## Using post() to Call a WCF Service

- post() can also be used to interact with an Ajax-enabled WCF service:

```
$.post('CustomerService.svc/GetCustomers',
    null, function (data) {
        var cust = data.d[0];
        alert(cust.FirstName + ' ' +
        cust.LastName);
    }, 'json');
```

```
<script type="text/javascript">
$(document).ready(function () {
    $('#myButton').click(function () {
        $.post('../GetCustomers.aspx', { PageSize: 15 },
            function (data) {
                $('#OutputDiv').html(data);
            });
    }));
});
</script>
```

```
<script type="text/javascript">
$(document).ready(function () {
    $('#MyButton').click(function () {
        $.post('../CustomerService.svc/GetCustomers', null,
            function (data) {
                var cust = data.d[0];
                $('#OutputDiv').html(cust.FirstName + " " + cust.LastName);
            }, 'json');
    });
});
</script>
```

## The ajax() Function

- **The ajax() function provides extra control over making Ajax calls to a server**
- **Configure using JSON properties:**
  - contentType
  - data
  - dataType
  - error
  - success
  - type (GET or POST)

## Using the ajax() Function

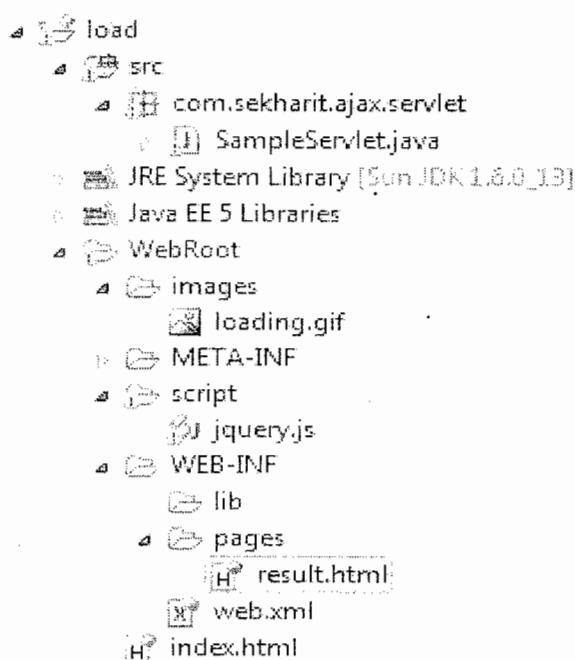
The ajax() function is configured by assigning values to JSON properties:

```
$.ajax({  
    url: '../CustomerService.svc/InsertCustomer',  
    data: customer,  
    dataType: 'json',  
    success: function (data, status, xhr) {  
        alert("Insert status: " + data.d.Status + '\n' +  
              data.d.Message);  
    },  
    error: function (xhr, status, error) {  
        alert('Error occurred: ' + status);  
    }  
}
```

```

<script type="text/javascript">
$(document).ready(function () {
    $('#MyButton').click(function () {
        var customer = 'cust=' +
            JSON.stringify({
                FirstName: $('#FirstNameTB').val(),
                LastName: $('#LastNameTB').val()
            });
        $.ajax({
            url: '../CustomerService.svc/InsertCustomer',
            data: customer,
            dataType: 'json',
            success: function (data, status, xhr) {
                $('#OutputDiv').html('Insert status: ' + data.d.Status);
            },
            error: function (xhr, status, error) {
                alert('Error occurred: ' + status);
            }
        });
    });
}

```



### SampleServlet.java

```
1. package com.sekharit.ajax.servlet;
```

```

2.
3. import java.io.IOException;
4.
5. import javax.servlet.ServletException;
6. import javax.servlet.http.HttpServlet;
7. import javax.servlet.http.HttpServletRequest;
8. import javax.servlet.http.HttpServletResponse;
9.
10. public class SampleServlet extends HttpServlet {
11.
12.     public void doGet(HttpServletRequest request, HttpServletResponse response)
13.             throws ServletException, IOException {
14.         try {
15.             Thread.sleep(5000);
16.         } catch (InterruptedException e) {
17.             e.printStackTrace();
18.         }
19.         request.getRequestDispatcher(
20.                 "/WEB-INF/pages/result.html").forward(request, response);
21.     }
22.
23. }
```

web.xml

```

1. <web-app>
2.   <servlet>
3.     <servlet-name>SampleServlet</servlet-name>
4.     <servlet-class>com.sekharit.ajax.servlet.SampleServlet</servlet-class>
5.   </servlet>
6.
7.   <servlet-mapping>
8.     <servlet-name>SampleServlet</servlet-name>
9.     <url-pattern>/SampleServlet</url-pattern>
10.    </servlet-mapping>
11.    <welcome-file-list>
12.      <welcome-file>index.html</welcome-file>
13.    </welcome-file-list>
14. </web-app>
```

index.html

```

1. <html>
2. <head>
3. <title>the title</title>
4. <script type="text/javascript" src=".//script/jquery.js"></script>
5. <script type="text/javascript" language="javascript">
6.   $(document).ready(function() {
7.
8.     $("button").click(function(event) {
9.       $('#resultId').html(' <br/> Please Wait...');
10.      $('#resultId').load('../SampleServlet');
11.
12.    });
13.
14.  
```

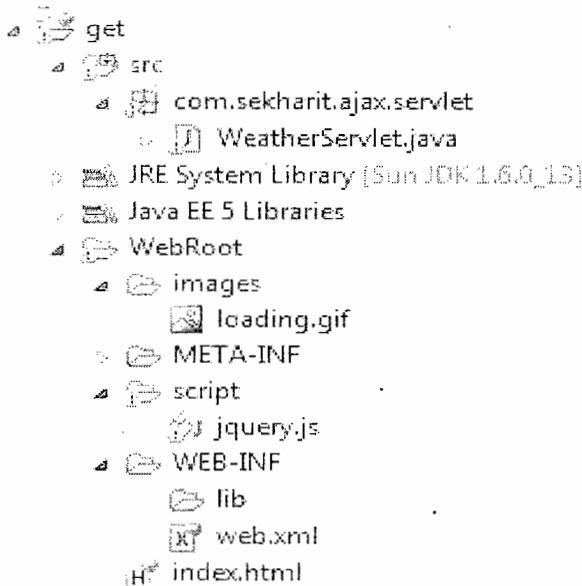
```

15.      });
16.  </script>
17.  <style type="text/css">
18.  .myclass{
19.      background-color: pink;
20.      font-size: 50px;
21.      border: 2px solid green;
22.      height: 300px;
23.      width: 400px;
24.  }
25. </style>
26. </head>
27. <body bgcolor="#c0f9fc">
28. <center>
29.     <h1>Ajax Web Application</h1>
30.     <h3>By using <i>selector.load(...)</i></h3>
31.     <hr/> <hr/>
32.
33.     <div id="resultId" class="myclass">
34.         Initial Content
35.     </div>
36.     <button>Load Data</button>
37. </center>
38. </body>
39. </html>

```

**result.html**

<div>This is the data from result.html</div>

**WeatherServlet.java**

```

1. package com.sekharit.ajax.servlet;
2.
3. import java.io.IOException;
4. import java.io.PrintWriter;

```

```

5.
6. import javax.servlet.ServletException;
7. import javax.servlet.http.HttpServlet;
8. import javax.servlet.http.HttpServletRequest;
9. import javax.servlet.http.HttpServletResponse;
10.
11. public class WeatherServlet extends HttpServlet {
12.
13.     public void doGet(HttpServletRequest request, HttpServletResponse response)
14.             throws ServletException, IOException {
15.         try {
16.             Thread.sleep(5000);
17.         } catch (InterruptedException e) {
18.             e.printStackTrace();
19.         }
20.         String city = request.getParameter("cityName");
21.         String report = getWeather(city);
22.         response.setContentType("text/xml");
23.         PrintWriter out = response.getWriter();
24.         out.println("<weather>");
25.         out.println("<city>" + city + "</city>");
26.         out.println("<report>" + report + "</report>");
27.         out.println("</weather>");
28.         out.flush();
29.         out.close();
30.     }
31.
32.     private String getWeather(String city) {
33.         String report;
34.
35.         if (city.equalsIgnoreCase("hyderabad")) {
36.             report = "Currently it is not raining in hyderabad.  
Average temperature is 20";
37.         } else if (city.equalsIgnoreCase("chennai")) {
38.             report = "It's a rainy season in Chennai now.  
Better get a umbrella before going out.";
39.         } else if (city.equalsIgnoreCase("bangalore")) {
40.             report = "It's mostly cloudy in Bangalore. Good weather  
for a cricket match.";
41.         } else {
42.             report = "The City you have entered is not present in our system.  
May be it has been destroyed in last World War or not  
yet built by the mankind";
43.         }
44.     }
45.     return report;
46. }
47.
48. }
49. }
50. }
51. }
52. }

```

web.xml

```

1. <web-app>
2.   <servlet>
3.     <servlet-name>WeatherServlet</servlet-name>
4.     <servlet-class>com.sekharit.ajax.servlet.WeatherServlet</servlet-class>
5.   </servlet>

```

```
6.    <servlet-mapping>
7.        <servlet-name>WeatherServlet</servlet-name>
8.        <url-pattern>/WeatherServlet</url-pattern>
9.    </servlet-mapping>
10.   <welcome-file-list>
11.       <welcome-file>index.html</welcome-file>
12.   </welcome-file-list>
13. </web-app>
```

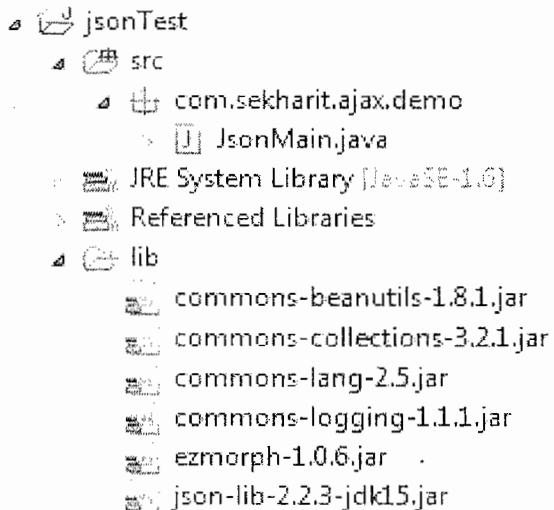
### index.html

```
1. <html>
2. <head>
3. <script type="text/javascript" src="script/jquery.js" ></script>
4. <script type="text/javascript">
5. $(document).ready(function() {
6.
7.     $("button").click(function() {
8.
9.         $('#reportResultId').html(' <br/> Please Wait...');

11.         var cityName = $("input#cityName").val();
12.
13.         $.get("./WeatherServlet", {"cityName": cityName}, function(xml) {
14.             var city = $("city", xml).text();
15.             var report = $("report", xml).text();
16.             var result = city + " : " + report;
17.             $("#reportResultId").html(result);
18.         });
19.
20.     });
21.
22.
23. });
24. </script>
25. <style type="text/css">
26. .myclass{
27.     background-color: pink;
28.     border: 2px solid green;
29.     font-size:25px;
30.     height: 300px;
31.     width: 400px;
32. }
33. </style>
34. </head>
35. <body bgcolor="#c0f9fc" >
36.     <center>
37.         <h1>Ajax Web Application</h1>
38.         <h3>By using <i>$._get(...)</i></h3>
39.         <hr/> <hr/>
40.         Enter City :
41.         <input type="text" id="cityName" size="30" />
42.         <button>Get Weather Report</button>
43.         <br/>
44.         <div id="reportResultId" class="myclass" >
45.             </div>
```

```

46.      </center>
47.    </body>
48.  </html>
```



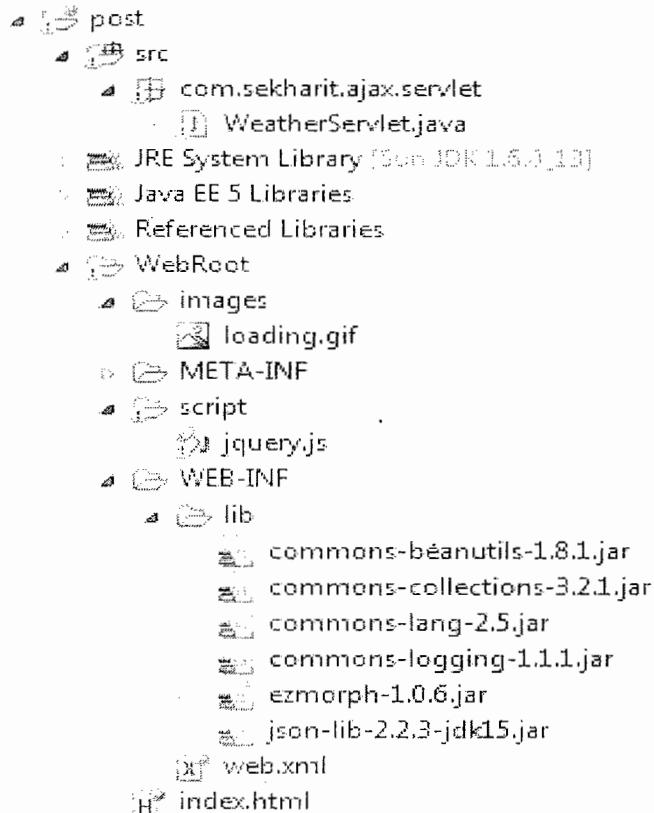
### JsonMain.java

```

1. package com.sekharit.ajax.demo;
2. import java.util.ArrayList;
3. import java.util.HashMap;
4. import java.util.List;
5. import java.util.Map;
6.
7. import net.sf.json.JSONObject;
8.
9. public class JsonMain {
10.     public static void main(String[] args) {
11.
12.         Map<String, Long> map = new HashMap<String, Long>();
13.         map.put("A", 10L);
14.         map.put("B", 20L);
15.         map.put("C", 30L);
16.
17.         List<String> list = new ArrayList<String>();
18.         list.add("Sunday");
19.         list.add("Monday");
20.         list.add("Tuesday");
21.
22.         JSONObject json = new JSONObject();
23.
24.         // adding properties to json object
25.         json.put("city", "Mumbai");
26.         json.put("country", "India");
27.
28.         // adding map to json
29.         jsonaccumulateAll(map);
30.
31.         // adding list to json
32.         jsonaccumulate("weekdays", list);
33.
```

```

34.         System.out.println(json.toString());
35.     }
36. }
```



### WeatherServlet.java

```

1. package com.sekharit.ajax.servlet;
2.
3. import java.io.IOException;
4. import java.io.PrintWriter;
5.
6. import javax.servlet.ServletException;
7. import javax.servlet.http.HttpServlet;
8. import javax.servlet.http.HttpServletRequest;
9. import javax.servlet.http.HttpServletResponse;
10.
11. import net.sf.json.JSONObject;
12.
13. public class WeatherServlet extends HttpServlet {
14.
15.     public void doPost(HttpServletRequest request, HttpServletResponse response)
16.             throws ServletException, IOException {
17.         try {
18.             Thread.sleep(5000);
19.         } catch (InterruptedException e) {
20.             e.printStackTrace();
21.         }
22.         String city = request.getParameter("cityName");
23.         String report = getWeather(city);
24.         response.setContentType("application/json");
```

```

25.         PrintWriter out = response.getWriter();
26.
27.         JSONObject json = new JSONObject();
28.         // adding properties to json object
29.         json.put("city", city);
30.         json.put("report", report);
31.         out.println(json);
32.         out.flush();
33.         out.close();
34.     }
35.
36.     private String getWeather(String city) {
37.         String report;
38.
39.         if (city.equalsIgnoreCase("hyderabad")) {
40.             report = "Currently it is not raining in hyderabad.  
Average temperature is 20";
41.         } else if (city.equalsIgnoreCase("chennai")) {
42.             report = "It's a rainy season in Chennai now.  
Better get a umbrella before going out.";
43.         } else if (city.equalsIgnoreCase("bangalore")) {
44.             report = "It's mostly cloudy in Bangalore. Good weather  
for a cricket match.";
45.         } else {
46.             report = "The City you have entered is not present in our system.  
May be it has been destroyed in last World War or not  
yet built by the mankind";
47.         }
48.     }
49. }
50. }
51. }
52. }
53. return report;
54. }
55. }
56. }
```

web.xml

```

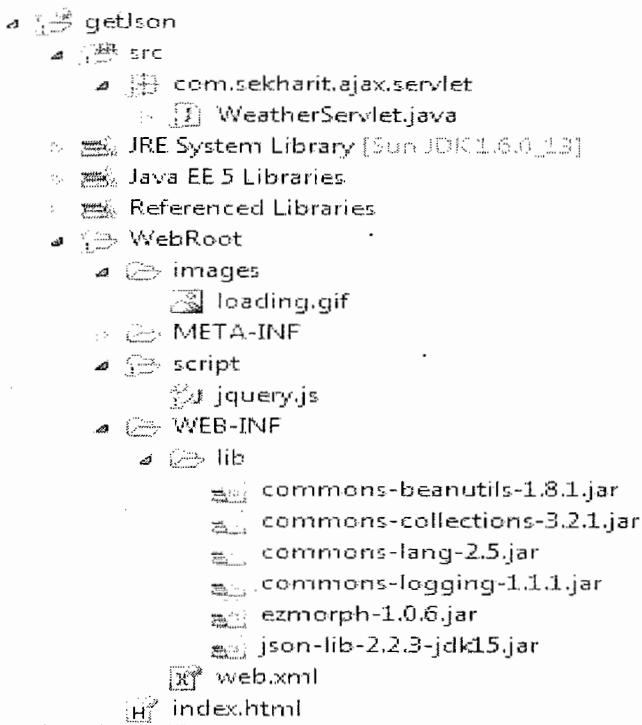
1. <web-app>
2.   <servlet>
3.     <servlet-name>WeatherServlet</servlet-name>
4.     <servlet-class>com.sekharit.ajax.servlet.WeatherServlet</servlet-class>
5.   </servlet>
6.   <servlet-mapping>
7.     <servlet-name>WeatherServlet</servlet-name>
8.     <url-pattern>/WeatherServlet</url-pattern>
9.   </servlet-mapping>
10.    <welcome-file-list>
11.      <welcome-file>index.html</welcome-file>
12.    </welcome-file-list>
13.  </web-app>
```

index.html

```

1. <html>
2. <head>
3. <script type="text/javascript" src="script/jquery.js" ></script>
4. <script type="text/javascript">
5. $(document).ready(function() {
6.
```

```
7.     $("button").click(function() {
8.
9.         $('div#reportResultId').html(' </img>
10.                                     <br/> Please Wait...') ;
11.         var cityName = $("#cityName").val();
12.
13.         $.post("./WeatherServlet", {"cityName": cityName}, function(json) {
14.             var city = json.city;
15.             var report = json.report;
16.             var result = city + " : " + report;
17.             $("div#reportResultId").html(result);
18.         });
19.
20.     });
21.
22.
23. });
24. </script>
25. <style type="text/css">
26.     .myclass{
27.         background-color: pink;
28.         border: 2px solid green;
29.         font-size:25px;
30.         height: 300px;
31.         width: 400px;
32.     }
33. </style>
34. </head>
35. <body bgcolor="#c0f9fc" >
36.     <center>
37.         <h1>Ajax Web Application</h1>
38.         <h3>By using <i>$ .post(...)</i></h3>
39.         <hr/> <hr/>
40.         Enter City :
41.         <input type="text" name="cityName" id="cityName" size="30" />
42.         <button>Get Weather Report</button>
43.         <br/>
44.         <div id="reportResultId" class="myclass" >
45.         </div>
46.     </center>
47. </body>
48. </html>
```

**WeatherServlet.java**

```

1. package com.sekharit.ajax.servlet;
2.
3. import java.io.IOException;
4. import java.io.PrintWriter;
5.
6. import javax.servlet.ServletException;
7. import javax.servlet.http.HttpServlet;
8. import javax.servlet.http.HttpServletRequest;
9. import javax.servlet.http.HttpServletResponse;
10.
11. import net.sf.json.JSONObject;
12.
13. public class WeatherServlet extends HttpServlet {
14.
15.     public void doGet(HttpServletRequest request, HttpServletResponse response)
16.             throws ServletException, IOException {
17.         try {
18.             Thread.sleep(5000);
19.         } catch (InterruptedException e) {
20.             e.printStackTrace();
21.         }
22.         String city = request.getParameter("cityName");
23.         String report = getWeather(city);
24.         response.setContentType("application/json");
25.         PrintWriter out = response.getWriter();
26.
27.         JSONObject json = new JSONObject();
28.         // adding properties to json object
29.         json.put("city", city);
30.         json.put("report", report);
31.         out.println(json);

```

```

32.         out.flush();
33.         out.close();
34.     }
35.
36.     private String getWeather(String city) {
37.         String report;
38.
39.         if (city.equalsIgnoreCase("hyderabad")) {
40.             report = "Currently it is not raining in hyderabad.  
Average temperature is 20";
41.         } else if (city.equalsIgnoreCase("chennai")) {
42.             report = "It's a rainy season in Chennai now. Better  
get a umbrella before going out.";
43.         } else if (city.equalsIgnoreCase("bangalore")) {
44.             report = "It's mostly cloudy in Bangalore. Good weather  
for a cricket match.";
45.         } else {
46.             report = "The City you have entered is not present in our system.  
May be it has been destroyed in last World War or not  
yet built by the mankind";
47.         }
48.     }
49.     return report;
50. }
51.
52. }
53. }
54. }
55. }
56. }
```

web.xml

```

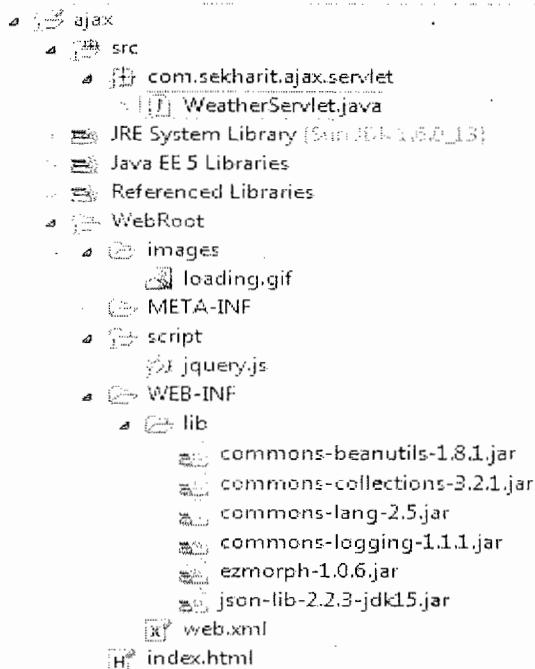
1. <web-app>
2.   <servlet>
3.     <servlet-name>WeatherServlet</servlet-name>
4.     <servlet-class>com.sekharit.ajax.servlet.WeatherServlet</servlet-class>
5.   </servlet>
6.   <servlet-mapping>
7.     <servlet-name>WeatherServlet</servlet-name>
8.     <url-pattern>/WeatherServlet</url-pattern>
9.   </servlet-mapping>
10.  <welcome-file-list>
11.    <welcome-file>index.html</welcome-file>
12.  </welcome-file-list>
13. </web-app>
```

index.html

```

1. <html>
2. <head>
3. <script type="text/javascript" src="script/jquery.js" ></script>
4. <script type="text/javascript">
5. $(document).ready(function() {
6.   $("button").click(function() {
7.
8.     $('div#reportResultId').html(' <br/>  
Please Wait...');
9.     var cityName = $("#cityName").val();
10.    $.getJSON("./WeatherServlet", {"cityName": cityName}, function(json) {
11.      var city = json.city;
```

```
12.         var report = json.report;
13.         var result = city + " : "+ report;
14.         $("div#reportResultId").html(result);
15.     });
16.
17.   });
18. });
19. </script>
20. <style type="text/css">
21. .myclass{
22.   background-color: pink;
23.   border: 2px solid green;
24.   font-size:25px;
25.   height: 300px;
26.   width: 400px;
27. }
28. </style>
29. </head>
30. <body bgcolor="#c0f9fc" >
31.   <center>
32.     <h1>Ajax Web Application</h1>
33.     <h3>By using <i>$.getJSON(...)</i></h3>
34.     <hr/> <hr/>
35.     Enter City :
36.     <input type="text" id="cityName" size="30" />
37.     <button>Get Weather Report</button>
38.     <br/>
39.     <div id="reportResultId" class="myclass" >
40.       </div>
41.     </center>
42.   </body>
43. </html>
```

**WeatherServlet.java**

```

1. package com.sekharit.ajax.servlet;
2.
3. import java.io.IOException;
4. import java.io.PrintWriter;
5.
6. import javax.servlet.ServletException;
7. import javax.servlet.http.HttpServlet;
8. import javax.servlet.http.HttpServletRequest;
9. import javax.servlet.http.HttpServletResponse;
10.
11. import net.sf.json.JSONObject;
12.
13. public class WeatherServlet extends HttpServlet {
14.
15.     public void doPost(HttpServletRequest request, HttpServletResponse response)
16.             throws ServletException, IOException {
17.         try {
18.             Thread.sleep(5000);
19.         } catch (InterruptedException e) {
20.             e.printStackTrace();
21.         }
22.         String city = request.getParameter("cityName");
23.         String report = getWeather(city);
24.         response.setContentType("application/json");
25.         PrintWriter out = response.getWriter();
26.
27.         JSONObject json = new JSONObject();
28.         // adding properties to json object
29.         json.put("city", city);
30.         json.put("report", report);
31.         System.out.println(json);
32.         out.println(json);

```

```

33.         out.flush();
34.         out.close();
35.     }
36.
37.     private String getWeather(String city) {
38.         String report;
39.
40.         if (city.equalsIgnoreCase("hyderabad")) {
41.             report = "Currently it is not raining in hyderabad.  
Average temperature is 20";
42.         } else if (city.equalsIgnoreCase("chennai")) {
43.             report = "It's a rainy season in Chennai now. Better  
get a umbrella before going out.";
44.         } else if (city.equalsIgnoreCase("bangalore")) {
45.             report = "It's mostly cloudy in Bangalore. Good weather  
for a cricket match.";
46.         } else {
47.             report = "The City you have entered is not present in our system.  
May be it has been destroyed in last World War or not  
yet built by the mankind";
48.         }
49.     }
50.     return report;
51. }
52.
53. }
54. }
55. }
56.
57. }

```

web.xml

```

1. <web-app>
2.   <servlet>
3.     <servlet-name>WeatherServlet</servlet-name>
4.     <servlet-class>com.sekharit.ajax.servlet.WeatherServlet</servlet-class>
5.   </servlet>
6.   <servlet-mapping>
7.     <servlet-name>WeatherServlet</servlet-name>
8.     <url-pattern>/WeatherServlet</url-pattern>
9.   </servlet-mapping>
10.    <welcome-file-list>
11.      <welcome-file>index.html</welcome-file>
12.    </welcome-file-list>
13.  </web-app>
14.

```

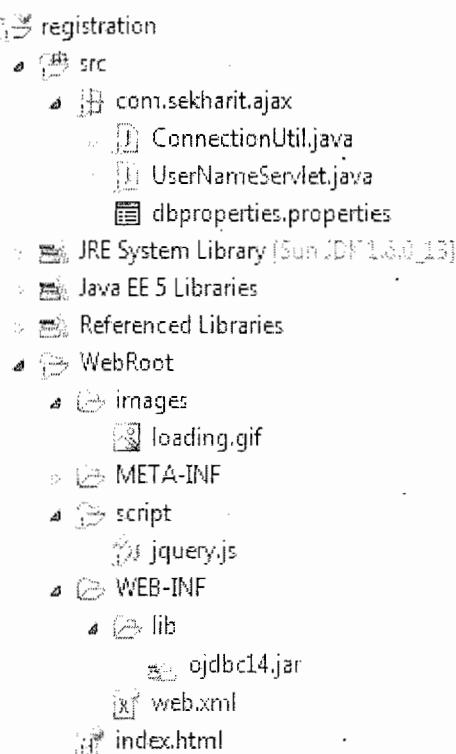
index.html

```

1. <html>
2. <head>
3. <script type="text/javascript" src="script/jquery.js"></script>
4. <script type="text/javascript">
5. $(document).ready(function() {
6.   $("button").click(function() {
7.     $('div#reportResultId').html(' <br/> Please Wait... ');
8.
9.     var cityName = $("#cityName").val();
10.    $.ajax(
11.      {
12.        url:"./WeatherServlet",

```

```
13.         data: {"cityName": cityName},
14.         dataType: 'json',
15.         type: 'POST',
16.         success: function(data, status, xhr) {
17.             var city = data.city;
18.             var report = data.report;
19.             var result = city + " : " + report;
20.             $("div#reportResultId").html(result);
21.         },
22.         error: function(xhr, status, error) {
23.             alert("success"+error);
24.             $("div#reportResultId").html("Processing Error...") ;
25.         }
26.     });
27.
28. });
29. });
30. </script>
31. <style type="text/css">
32. .myclass{
33.     background-color: pink;
34.     border: 2px solid green;
35.     font-size:25px;
36.     height: 300px;
37.     width: 400px;
38. }
39. </style>
40. </head>
41. <body bgcolor="#c0f9fc" >
42.     <center>
43.         <h1>Ajax Web Application</h1>
44.         <h3>By using <i>$.ajax(...)</i></h3>
45.         <hr/> <hr/>
46.         Enter City :
47.         <input type="text" name="cityName" id="cityName" size="30" />
48.         <button>Get Weather Report</button>
49.         <br/>
50.         <div id="reportResultId" class="myclass" >
51.         </div>
52.     </center>
53. </body>
54. </html>
```



### dbproperties.properties

```

1. driverClass=oracle.jdbc.driver.OracleDriver
2. url=jdbc:oracle:thin:@localhost:1521:XE
3. username=system
4. password=tiger

```

### ConnectionUtil.java

```

1. package com.sekharit.ajax;
2.
3. import java.io.IOException;
4. import java.sql.Connection;
5. import java.sql.DriverManager;
6. import java.sql.ResultSet;
7. import java.sql.SQLException;
8. import java.sql.Statement;
9. import java.util.HashMap;
10. import java.util.Properties;
11.
12. public class ConnectionUtil {
13.
14.     private static HashMap dbProps;
15.     static {
16.         try {
17.             dbProps = new HashMap();
18.             Properties properties = new Properties();
19.             properties
20.                 .load(ConnectionUtil.class
21.                     .getClassLoader()
22.                     .getResourceAsStream(

```

```
23.     "com/sekharit/ajax/dbproperties.properties"));
24.             dbProps.putAll(properties);
25.             Class.forName((String) dbProps.get("driverClass"));
26.
27.         } catch (ClassNotFoundException e) {
28.             e.printStackTrace();
29.         } catch (IOException e) {
30.             e.printStackTrace();
31.         }
32.     }
33.
34.     public static Connection getConnection() {
35.         Connection connection = null;
36.         try {
37.             connection = DriverManager.getConnection((String) dbProps
38.                 .get("url"), (String) dbProps.get("username"),
39.                 (String) dbProps.get("password"));
40.         } catch (SQLException e) {
41.             e.printStackTrace();
42.         }
43.         return connection;
44.     }
45.
46.     public static void closeConnection(Connection con) {
47.         if (con != null) {
48.             try {
49.                 con.close();
50.             } catch (SQLException e) {
51.                 e.printStackTrace();
52.             }
53.         }
54.     }
55.
56.
57.     public static void closeConnection(Connection con, Statement st) {
58.         closeConnection(con);
59.         if (st != null) {
60.             try {
61.                 st.close();
62.             } catch (SQLException e) {
63.                 e.printStackTrace();
64.             }
65.         }
66.     }
67.
68.
69.     public static void closeConnection(Connection con, Statement st,
70.             ResultSet rs) {
71.         closeConnection(con, st);
72.         if (rs != null) {
73.             try {
74.                 rs.close();
75.             } catch (SQLException e) {
76.                 e.printStackTrace();
```

```
77.          }
78.      }
79.  }
80.
81. }
```

### UserNameServlet.java

```
1. package com.sekharit.ajax;
2.
3. import java.io.IOException;
4. import java.io.PrintWriter;
5. import java.sql.Connection;
6. import java.sql.PreparedStatement;
7. import java.sql.ResultSet;
8.
9. import javax.servlet.ServletException;
10. import javax.servlet.http.HttpServlet;
11. import javax.servlet.http.HttpServletRequest;
12. import javax.servlet.http.HttpServletResponse;
13.
14. public class UserNameServlet extends HttpServlet {
15.     private String QUERY = "SELECT * FROM USER_TAB WHERE USER_ID=?";
16.
17.     public void doPost(HttpServletRequest request, HttpServletResponse response)
18.             throws ServletException, IOException {
19.
20.         try {
21.             Thread.sleep(5000);
22.         } catch (InterruptedException e) {
23.             e.printStackTrace();
24.         }
25.         response.setContentType("text/xml");
26.         String userId = request.getParameter("userId");
27.         PrintWriter out = response.getWriter();
28.         StringBuffer buffer = new StringBuffer();
29.         buffer.append("<resp>");
30.         buffer.append("<valid>");
31.         if (userId == null || userId.trim().length() == 0) {
32.             buffer.append("EMPTY");
33.         } else {
34.             Connection connection = null;
35.             PreparedStatement ps = null;
36.             ResultSet rs = null;
37.             try {
38.                 connection = ConnectionUtil.getConnection();
39.                 ps = connection.prepareStatement(QUERY);
40.                 ps.setString(1, userId);
41.                 rs = ps.executeQuery();
42.                 if (rs.next()) {
43.                     buffer.append("NO");
44.                 } else {
45.                     buffer.append("YES");
46.                 }
47.             } catch (Exception e) {
48.                 buffer.append("ERROR");

```

```

49.           }
50.       }
51.       buffer.append("</valid>");
52.       buffer.append("</resp>");
53.       out.println(buffer.toString());
54.       out.flush();
55.       out.close();
56.   }
57.
58. }

```

**web.xml**

```

1. <web-app >
2.   <servlet>
3.     <servlet-name>UserNameServlet</servlet-name>
4.     <servlet-class>com.sekharit.ajax.UserNameServlet</servlet-class>
5.   </servlet>
6.
7.   <servlet-mapping>
8.     <servlet-name>UserNameServlet</servlet-name>
9.     <url-pattern>/userNameServlet</url-pattern>
10.    </servlet-mapping>
11.    <welcome-file-list>
12.      <welcome-file>index.html</welcome-file>
13.    </welcome-file-list>
14.  </web-app>

```

**index.html**

```

1. <html>
2. <head>
3. <script type="text/javascript" src="script/jquery.js" ></script>
4. <script type="text/javascript">
5. $(document).ready(function(){
6.   $("#usernameId").change(function(){
7.     hideAll();
8.     $('span#loadingId').show();
9.     var username = $("#usernameId").val();
10.    $.ajax(
11.      {
12.        url:"./userNameServlet",
13.        data:{ "userId": username},
14.        dataType:'xml',
15.        type:'POST',
16.        success: function(data, status, xhr){
17.          var result = $("valid", $(data)).text();
18.          if(result == 'YES'){
19.            hideAll();
20.            $('span#rightId').show();
21.          } else if(result == 'NO'){
22.            hideAll();
23.            $('span#wrongId').show();
24.          } else if(result == 'EMPTY'){
25.            hideAll();
26.            $('span#emptyId').show();

```

```
27.          } else if(result == 'ERROR'){
28.                  hideAll();
29.                  $("span#errorId").show();
30.          }
31.
32.          },
33.          error: function(xhr, status, error){
34.                  hideAll();
35.                  $("span#errorId").show();
36.          }
37.      });
38.
39.  });
40. });
41.
42. function hideAll(){
43.     $('span#loadingId').hide();
44.     $('span#wrongId').hide();
45.     $('span#rightId').hide();
46.     $('span#emptyId').hide();
47. }
48. </script>
49. <style type="text/css">
50. .myclass{
51.     background-color: pink;
52.     border: 2px solid green;
53.     font-size:25px;
54.     height: 300px;
55.     width: 400px;
56.     float: left;
57. }
58. img{
59.     height: 20px;
60.     width: 20px;
61. }
62. </style>
63. </head>
64. <body bgcolor="#c0f9fc" >
65.     <center>
66.         <h1>Ajax Web Application</h1>
67.         <h3>By using <i>$.ajax(...)</i></h3>
68.         <hr/> <hr/>
69.             <h3> Registratoin Page </h3>
70.         <table border="1" >
71.             <tr>
72.                 <td>Name</td>
73.                 <td>
74.                     <input type="text" id="usernameId" name="name"/>
75.                     <span id="loadingId" style="display: none" >
76.                          </img>
77.                     </span>
78.                     <span id="wrongId" style="display: none; color: red" >
79.                         User Name not Available
80.                     </span>
81.                     <span id="rightId" style="display: none; color: green" >
```

```

82.                               User Name Available
83.                               </span>
84.                               <span id="emptyId" style="display: none; color: red" >
85.                                   User Name can't be empty
86.                                   </span>
87.                               <span id="errorId" style="display: none; color: red" >
88.                                   Processing Error...
89.                                   </span>
90.                               </td>
91.                           </tr>
92.                           <tr>
93.                               <td>Password</td>
94.                               <td>
95.                                   <input type="password" name="password"/>
96.                               </td>
97.                           </tr>
98.                           <tr>
99.                               <td>Address</td>
100.                              <td>
101.                                  <textarea cols="50" rows="4" > </textarea>
102.                              </td>
103.                           </tr>
104.                           <tr>
105.                               <td colspan="2" align="center" >
106.                                   <input type="submit" value="Register">
107.                               </td>
108.                           </tr>
109.                       </table>
110.                   </center>
111.               </body>
112.           </html>

```

**SQL> select \* from user\_tab;**

USER_ID	PASSWORD	COUNTRY	STATE	CITY
sekhar	*****	India	A.P	Hyd
sekharreddy	#####	India	A.P	Hyd

Reference websites:

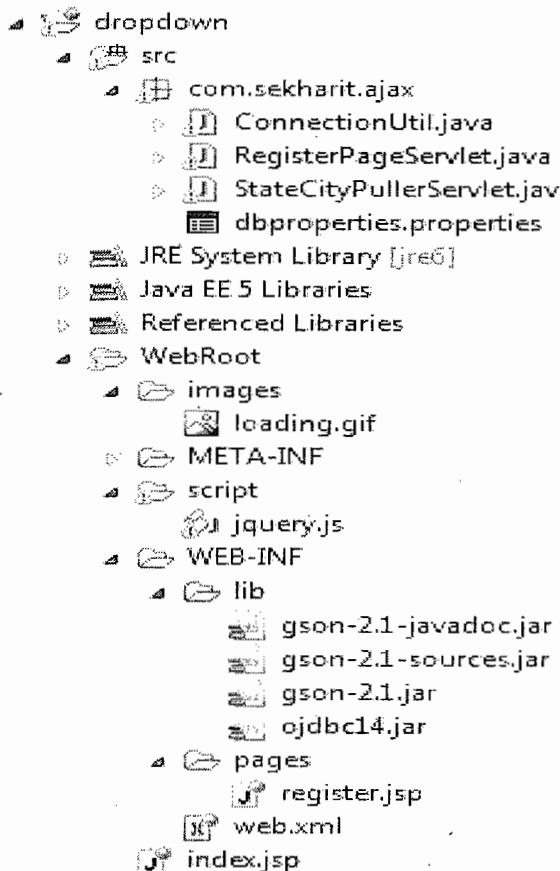
[Jquery.com](http://jquery.com)

[Jqueryui.com](http://jqueryui.com)

<http://codylindley.com/jqueryselectors/>

<http://appendto.com/community/jquery-vsdoc>

<http://james.padolsey.com/jquery/#v=1.6.2&fn=jQuery.fn.hide>



### dbproperties.properties

--SAME AS ABOVE --

### ConnectionUtil.java

--SAME AS ABOVE --

### RegisterPageServlet.java

```

1. package com.sekharit.ajax;
2.
3. import java.io.IOException;
4. import java.sql.Connection;
5. import java.sql.ResultSet;
6. import java.sql.Statement;
7. import java.util.HashMap;
8. import java.util.Map;
9.
10. import javax.servlet.ServletException;
11. import javax.servlet.http.HttpServlet;
12. import javax.servlet.http.HttpServletRequest;
13. import javax.servlet.http.HttpServletResponse;
14.
15. public class RegisterPageServlet extends HttpServlet {
16.     private String COUNTRIES_QUERY = "SELECT * FROM COUNTRIES";
17.
18.     public void doGet(HttpServletRequest request, HttpServletResponse response)
19.             throws ServletException, IOException {
20.         Map<String, String> countries = new HashMap<String, String>();
21.         Connection connection = null;
22.         Statement st = null;
```

```

23.         ResultSet rs = null;
24.         try {
25.             connection = ConnectionUtil.getConnection();
26.             st = connection.createStatement();
27.             rs = st.executeQuery(COUNTRIES_QUERY);
28.             while (rs.next()) {
29.                 countries.put(rs.getString(1), rs.getString(2));
30.             }
31.         } catch (Exception e) {
32.             e.printStackTrace();
33.         }
34.         request.setAttribute("countries", countries);
35.         request.getRequestDispatcher("/WEB-INF/pages/register.jsp").forward(
36.             request, response);
37.     }
38. }
39. }
```

**StateCityPullerServlet.java**

```

1. package com.sekharit.ajax;
2.
3. import java.io.IOException;
4. import java.sql.Connection;
5. import java.sql.PreparedStatement;
6. import java.sql.ResultSet;
7. import java.util.HashMap;
8. import java.util.Map;
9.
10. import javax.servlet.ServletException;
11. import javax.servlet.http.HttpServlet;
12. import javax.servlet.http.HttpServletRequest;
13. import javax.servlet.http.HttpServletResponse;
14.
15. import com.google.gson.Gson;
16.
17. public class StateCityPullerServlet extends HttpServlet {
18.     private String STATES_QUERY = "SELECT * FROM STATES WHERE CID=?";
19.     private String CITIES_QUERY = "SELECT * FROM CITIES WHERE SID=?";
20.
21.     public void doGet(HttpServletRequest request, HttpServletResponse response)
22.             throws ServletException, IOException {
23.
24.         try {
25.             Thread.sleep(5000);
26.         } catch (InterruptedException e) {
27.             e.printStackTrace();
28.         }
29.         String type = request.getParameter("type");
30.         // Returns "country" or "state".
31.         String value = request.getParameter("value");
32.         // Value of selected country or state.
33.         Map<String, String> options = null;
34.         if (type.equalsIgnoreCase("state")) {
35.             options = getOptions(STATES_QUERY, value);
36.         } else if (type.equalsIgnoreCase("city")) {
```

```

37.             options = getOptions(CITIES_QUERY, value);
38.         }
39.         String json = new Gson().toJson(options);
40.         // Convert Java object to JSON string.
41.         response.setContentType("application/json");
42.         // Inform client that
43.         response.setCharacterEncoding("UTF-8");
44.         // Important if you want world
45.         response.getWriter().write(json);
46.         // Write JSON string to response.
47.     }
48.
49.     private Map<String, String> getOptions(String query, String value) {
50.         Map<String, String> options = new HashMap<String, String>();
51.         Connection connection = null;
52.         PreparedStatement ps = null;
53.         ResultSet rs = null;
54.         try {
55.             connection = ConnectionUtil.getConnection();
56.             ps = connection.prepareStatement(query);
57.             ps.setInt(1, Integer.parseInt(value));
58.             rs = ps.executeQuery();
59.             while (rs.next()) {
60.                 options.put(rs.getString(1), rs.getString(2));
61.             }
62.         } catch (Exception e) {
63.             e.printStackTrace();
64.         }
65.         return options;
66.     }
67.
68. }

```

**web.xml**

```

1. <?xml version="1.0" encoding="UTF-8"?>
2. <web-app version="2.5" xmlns="http://java.sun.com/xml/ns/javaee"
3.   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
4.   xsi:schemaLocation="http://java.sun.com/xml/ns/javaee
5.   http://java.sun.com/xml/ns/javaee/web-app_2_5.xsd">
6.   <servlet>
7.     <servlet-name>RegisterPageServlet</servlet-name>
8.     <servlet-class>com.sekharit.ajax.RegisterPageServlet</servlet-class>
9.   </servlet>
10.    <servlet>
11.      <servlet-name>StateCityPullerServlet</servlet-name>
12.      <servlet-class>com.sekharit.ajax.StateCityPullerServlet</servlet-class>
13.    </servlet>
14.
15.    <servlet-mapping>
16.      <servlet-name>RegisterPageServlet</servlet-name>
17.      <url-pattern>/registerPage</url-pattern>
18.    </servlet-mapping>
19.    <servlet-mapping>
20.      <servlet-name>StateCityPullerServlet</servlet-name>
21.      <url-pattern>/stateCityPullerServlet</url-pattern>

```

```

22.      </servlet-mapping>
23.
24.      <welcome-file-list>
25.          <welcome-file>index.jsp</welcome-file>
26.      </welcome-file-list>
27. </web-app>

```

**index.jsp**

```

1. <html>
2. <body bgcolor="#c0f9fc" >
3.   <center>
4.     <h1>Ajax Web Application</h1>
5.     <hr/> <hr/>
6.     <a href=". /registerPage" >Registration</a>
7.   </center>
8. </body>
9. </html>

```

**register.jsp**

```

1. <%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core" %>
2. <html>
3. <head>
4. <script type="text/javascript" src="script/jquery.js" ></script>
5. <script type="text/javascript">
6. $(document).ready(function(){
7.   $('#country').change(function() {
8.     var value = $(this).val();
9.     var type = 'state';
10.    removeAllPrevOptions();
11.    fillOptions(type, value);
12.  });
13.
14.   $('#state').change(function() {
15.     var value = $(this).val();
16.     var type = 'city';
17.     removeCityPrevOptions()
18.     fillOptions(type, value);
19.   });
20.
21. });
22.
23. function fillOptions(type, value) {
24.   var loadingId = 'span#+type+LoadingId';
25.   var dropdown = $('#'+type);
26.
27.   $(loadingId).show();
28.   $.getJSON('. /sateCityPullerServlet', { "type" : type, "value": value },
29.             function(opts) {
30.               if (opts) {
31.                 $.each(opts, function(key, value) {
32.                   dropdown.append($('').val(key).text(value));
33.                 });
34.               } else {
35.                 alert("No options are available");

```

```
36.          }
37.          $(loadingId).hide();
38.
39.      });
40.  }
41.
42.
43.  function removeAllPrevOptions(){
44.      removeStatePrevOptions();
45.      removeCityPrevOptions();
46.  }
47.
48.  function removeStatePrevOptions(){
49.      var stateDropdown = $('#state');
50.      $('>option', stateDropdown).remove();
51.      stateDropdown.append($('<option/>').text('Please select '));
52.  }
53.
54.  function removeCityPrevOptions(){
55.      var cityDropdown = $('#city');
56.      $('>option', cityDropdown).remove();
57.      cityDropdown.append($('<option/>').text('Please select '));
58.  }
59.
60.
61.  function hideAllLoading(){
62.      $('#stateLoadingId').hide();
63.      $('#cityLoadingId').hide();
64.  }
65. </script>
66. <style type="text/css" >
67.
68.     table, td, th{
69.         border-collapse:collapse;
70.         border : 2px solid red;
71.         width : 400px;
72.     }
73.     img{
74.         height: 20px;
75.         width: 20px;
76.     }
77. </style>
78.
79. </head>
80. <body bgcolor="#c0f9fc" >
81.   <center>
82.       <h1>Ajax Web Application</h1>
83.       <hr/> <hr/>
84.       <form>
85.           <table >
86.             <tr>
87.                 <td> Country </td>
88.                 <td>
89.                     <select id="country" name="country">
90.                         <option selected="selected" >Please select</opt-
```

```
91.          <c:forEach items="${countries}" var="country">
92.              <option value="${country.key}" >
93.                  ${country.value}
94.              </option>
95.          </c:forEach>
96.      </select>
97.  </td>
98. </tr>
99. <tr>
100.     <td> State </td>
101.     <td>
102.         <select id="state" name="state">
103.             <option>Please select</option>
104.         </select>
105.         <span id="stateLoadingId" style="display: none" >
106.              </img>
107.         </span>
108.     </td>
109. </tr>
110. <tr>
111.     <td> City </td>
112.     <td>
113.         <select id="city" name="city">
114.             <option>Please select</option>
115.         </select>
116.         <span id="cityLoadingId" style="display: none" >
117.              </img>
118.         </span>
119.     </td>
120. </tr>
121. </table>
122.
123.     </form>
124. </center>
125. </body>
126. </html>
```

**Reference websites:**

[Jquery.com](http://jquery.com)

[Jqueryui.com](http://jqueryui.com)

<http://codylindley.com/jqueryselectors/>

<http://appendto.com/community/jquery-vsdoc>

<http://james.padolsey.com/jquery/#v=1.6.2&fn=jQuery.fn.hide>