

HTML: HYPER TEXT MARKUP LANGUAGE

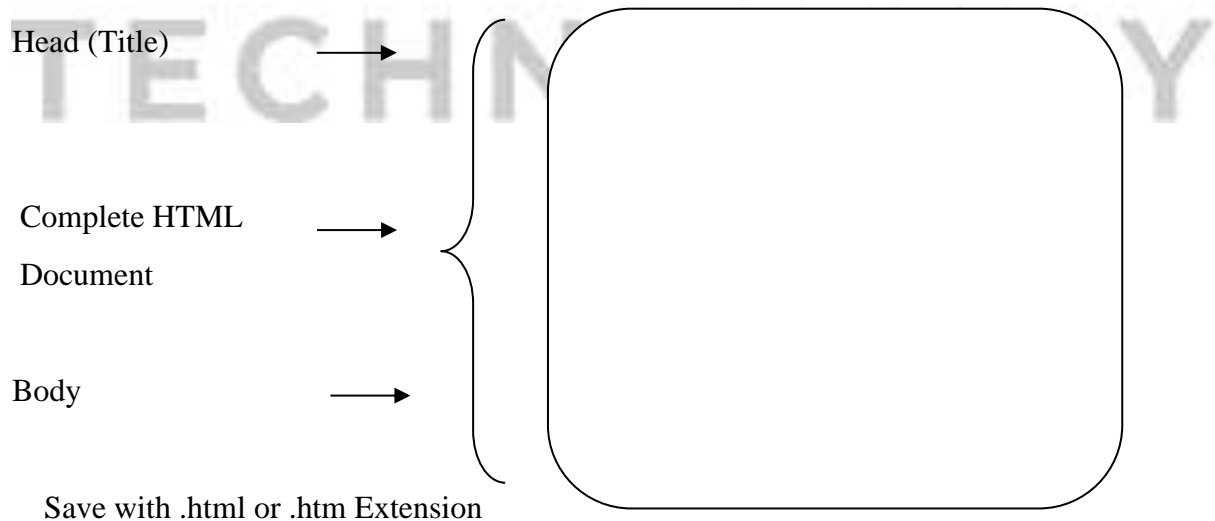
Introduction to Web terminology

- Web Page
- Web site
- Web Publishing
- Browser
- Uniform resource locator (URL)
- Static Site
- Dynamic site

Foot steps for designing web page

- Hyper Text Markup Language (HTML)
- HTML code
- HTML tags
- Head
- Title
- Body

Structure of HTML Document



Example:

```
<HTML>
  <HEAD>
    <TITLE>
      The First Example
    </TITLE>
  </HEAD>
  <BODY>
    Hello world
  </BODY>
</HTML>
```

Example Explained :

- The first tag in your HTML document is <html>. This tag tells your browser that this is the start of an HTML document.
- The last tag in your document is </html>. This tag tells your browser that this is the end of the HTML document.
- The text between the <head> tag and the </head> tag is header information.
- Header information is not displayed in the browser window.
- The text between the <title> tags is the title of your document. The title is displayed in your browser's caption.
- The text between the <body> tags is the text that will be displayed in your browser.

Stepwise procedure:

- Open any text editor (notepad).
- Open a new file.
- Write HTML code as given in example.
- Save file with .html extension.
- Open Browser (internet explorer / Netscape Navigator)
- Open recently saved file.
- Observe the effects on the browser.

HTML Tags:

- HTML tags are used to mark-up HTML **elements**
- HTML tags are surrounded by the **two characters < and >**
- The surrounding characters are called **angle brackets**
- HTML tags normally **come in pairs** like and
- The first tag in a pair is the **start tag**, the second tag is the **end tag**
- The text between the start and end tags is the **element content**
- HTML tags are **not case sensitive**, means the same as

This is an HTML element:

- This text is bold
- The HTML element starts with a **start tag**:
The **content** of the HTML element is: This text is bold
The HTML element ends with an **end tag**:
- The purpose of the tag is to define an HTML element that should be displayed as bold.

Tag Attributes:

- Tags can have attributes. Attributes provide additional information to an HTML element. The following tag defines an HTML table: <table>. With an added border attribute, you can tell the browser that the table should have no borders: <table border="0">
- Attributes always come in name/value pairs like this: name="value".
- Attributes are always specified in the start tag of an HTML element.
- Always Quote Attribute Values
- Attribute values should always be enclosed in quotes. Double style quotes are the most common, but single style quotes are also allowed.
- In some rare situations, like when the attribute value itself contains quotes, it is necessary to use single quotes:
- name='John "ShotGun" Nelson'

Creation of web page using HTML basic Tags:

- Structure Tags:-

<HTML> <HEAD> <TITLE> <BODY>

</HTML> </HEAD> </TITLE> </BODY>

- Block level tags:-

- Headings:-

<h1> to <h6> tags

<h1>This is a heading</h1>

<h2>This is a heading</h2>

- Paragraphs:

Paragraphs are defined with the <p> tag.

<p>This is a paragraph</p>

<p>This is another paragraph</p>

- Line Breaks:

The
 tag is used when you want to end a line, but don't want to start a new paragraph. The
 tag forces a line break wherever you place it.

<p>This
 is a para
graph with line breaks</p>The
 tag is an empty tag. It has no closing tag.

- Comments in HTML:-

The comment tag is used to insert a comment in the HTML source code. A comment will be ignored by the browser. You can use comments to explain your code, which can help you when you edit the source code at a later date.

<!-- This is a comment -->

Text Formatting Tags

- Tag Description:-

Defines bold text

<big>Defines big text

Defines emphasized text

<i>Defines italic text

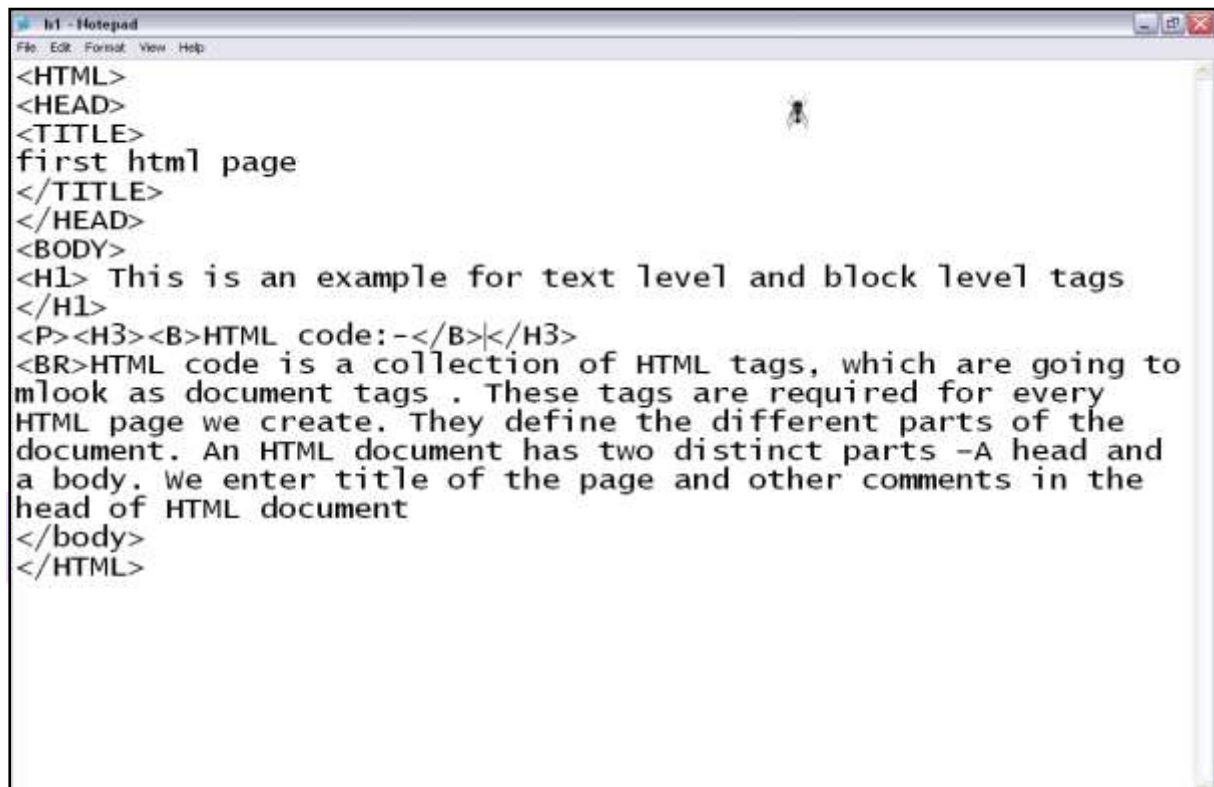
<small>Defines small text

Defines strong text

<sub>Defines subscripted text

<sup>Defines superscripted text

Example shown with Notepad :



```
<HTML>
<HEAD>
<TITLE>
first html page
</TITLE>
</HEAD>
<BODY>
<H1> This is an example for text level and block level tags
</H1>
<P><H3><B>HTML code:-</B></H3>
<BR>HTML code is a collection of HTML tags, which are going to
mlook as document tags . These tags are required for every
HTML page we create. They define the different parts of the
document. An HTML document has two distinct parts -A head and
a body. We enter title of the page and other comments in the
head of HTML document
</body>
</HTML>
```

Following is the output shown in Internet Explorer:



HTML Character Entities:-

- **Character Entities**

Some characters have a special meaning in HTML, like the less than sign (<) that defines the start of an HTML tag. If we want the browser to actually display these characters we must insert character entities in the HTML source.

- A character entity has three parts: an ampersand (&), an entity name or a # and an entity number, and finally a semicolon (;).
- To display a less than sign in an HTML document we must write: **<** or **<**;
- The advantage of using a name instead of a number is that a name is easier to remember. The disadvantage is that not all browsers support the newest entity names, while the support for entity numbers is very good in almost all browsers.
- **Note** that the entities are case sensitive.

- **The Most Common Character Entities:**

Result	Description	Entity Name	Entity Number
<	Non breaking space	 	
>	Less than	<	<
&	Greater than	>	>
“	Ampersand	&	&
‘	Quotation mark	"	"
	Apostrophe	'	'

Creation of web page using List and Links :-

- Type of lists :-
- The ordered list Tag:-

It define sequentially numbered list of items. It is used in conjunction with the tag, which is used to tag the individual list items in the list. Sometimes tags do not have end tags because it can be implied as <P> tag.

Example:-

```
<OL>
```

```
<LI>computer concepts
```

```
<LI>Ms Excel
```

```
</OL>
```

- **The Unordered list Tag:-**

The unordered list tag define a bulleted list of items. The LI is nested inside UL tag and define each item within the list. For nesting list you can list inside another list of same or different kind of list. The browser automatically indents nested list level.

Example:-

```
<UL type="square">  
    <LI> software  
    <OL type="1">  
        <LI> system software  
        <LI> application software  
    </OL>  
</UL>
```

- **The Definition list:-**

The DT tag allows you to create glossaries or list of items and definitions. A glossary consist of three tag elements tag to define list (DL), tag to define the term (DT), and to define the definition (DD).

Example:-

```
<DL><DT> hardware  
<DD> it is defined as physical or tangible equipments  
    associated with computer system.  
<DT> software  
<DD> It is a set of program  
</DL>
```

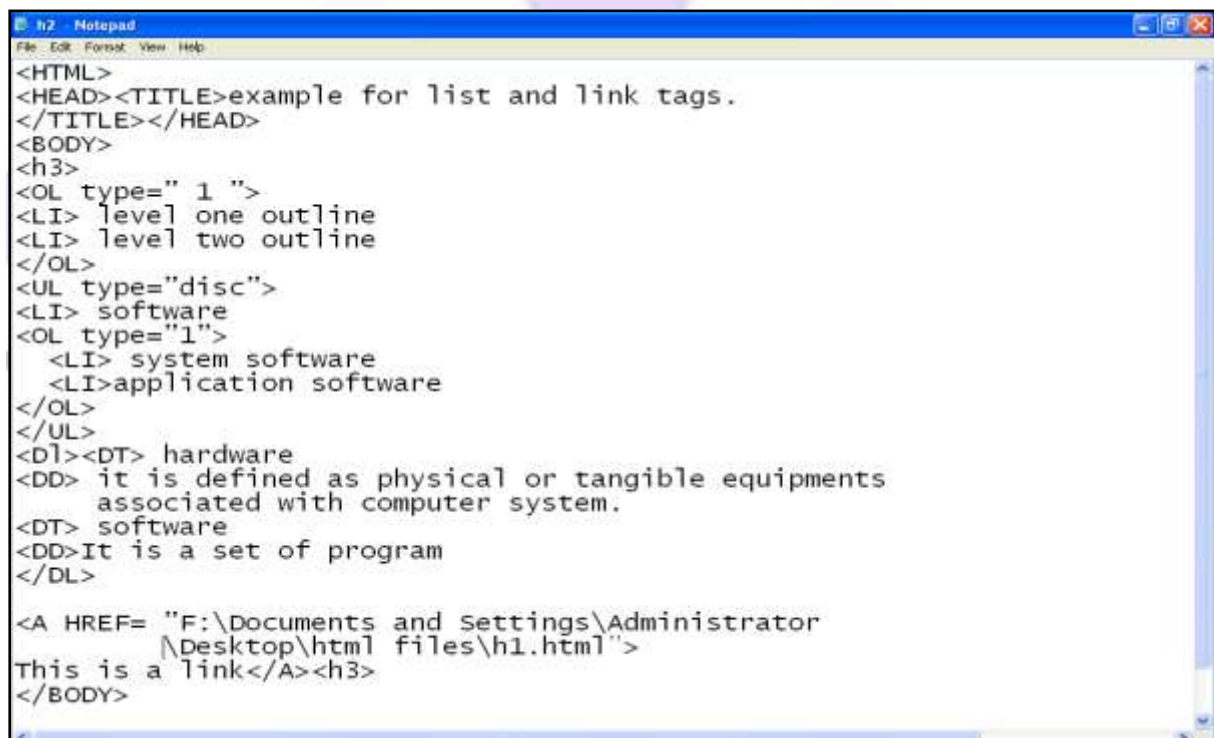
Creation of links:-

- The Anchor Tag and the href Attribute:-
 - **HTML uses the <a> (anchor) tag to create a link to another document.**
 - An anchor can point to any resource on the Web: an HTML page, an image, a sound file etc.
- The syntax of creating an anchor:
 - `Text to be displayed` The <a> tag is used to create an anchor to link from, the href attribute is used to address the document to link to, and the words between the open and close of the anchor tag will be displayed as a hyperlink.

Example:-

```
<HTML>
  <HEAD>
    <TITLE>example for link tags.
  </TITLE>
</HEAD>
<BODY>
  <A href="c:\mydocuments\example1.html">
    This is the link
  </A>
</BODY>
</HTML>
```

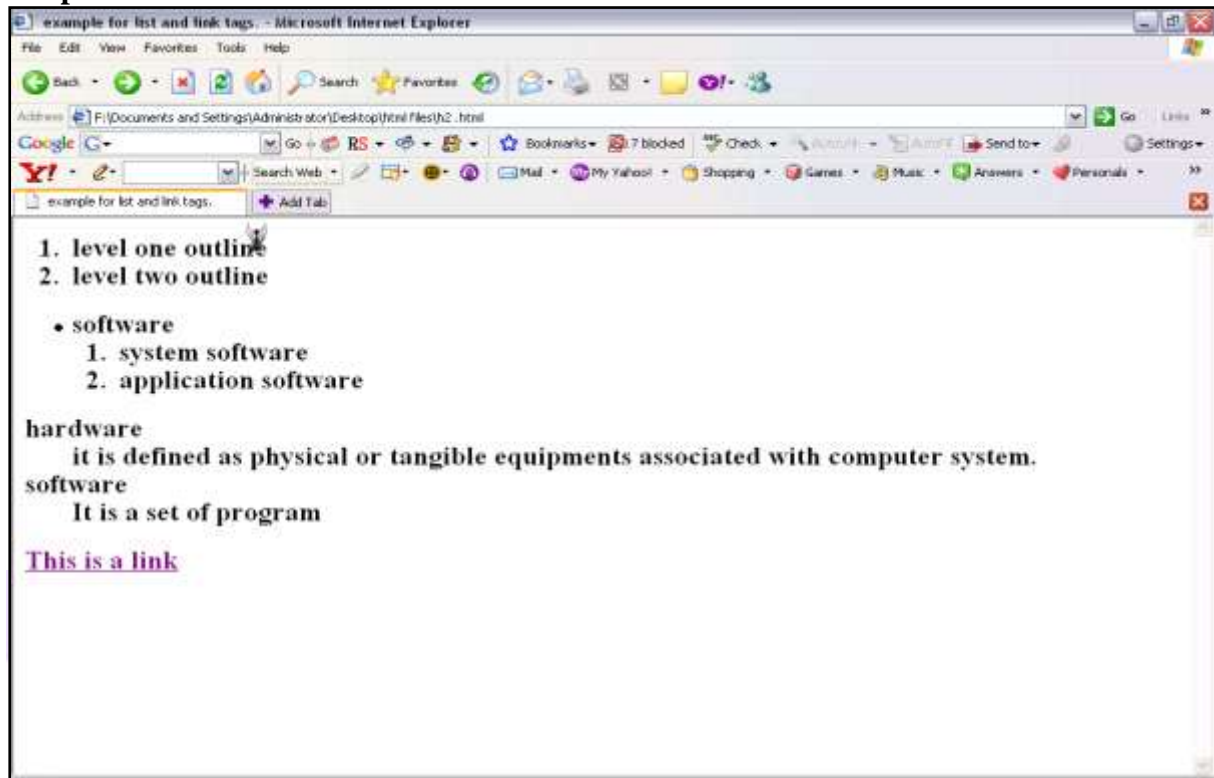
Sample run using list and link tags:-

A screenshot of a Notepad window titled 'h2 - Notepad'. The window contains the following HTML code:

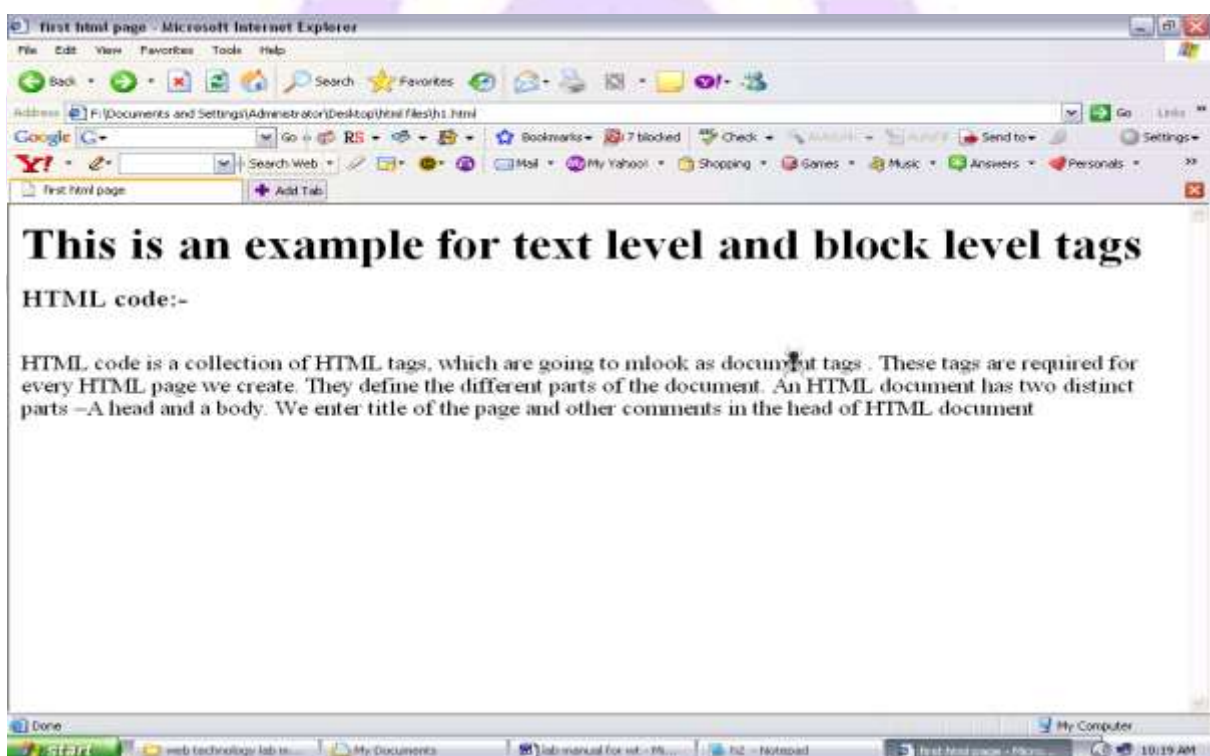
```
<HTML>
<HEAD><TITLE>example for list and link tags.
</TITLE></HEAD>
<BODY>
<h3>
<OL type="1">
<LI> level one outline
<LI> level two outline
</OL>
<UL type="disc">
<LI> software
<OL type="1">
  <LI> system software
  <LI> application software
</OL>
</UL>
<DL><DT> hardware
<DD> it is defined as physical or tangible equipments
      associated with computer system.
<DT> software
<DD>It is a set of program
</DL>

<A HREF= "F:\Documents and Settings\Administrator
      \Desktop\html files\h1.html">
This is a link</A><h3>
</BODY>
```


Output



Output after clicking on Hyperlink:-



Link to the location on same page:-

```
<html>
<body>
<p>
<a href="#C4">See also Chapter 4.</a>
</p>
<h2>Chapter 1</h2>
<p>This chapter explains ba bla bla</p>
<h2>Chapter 2</h2>
<p>This chapter explains ba bla bla</p>
<h2>Chapter 3</h2>
<p>This chapter explains ba bla bla</p>
<h2><a name="C4">Chapter 4</a></h2>
<p>This chapter explains ba bla bla</p>
</body>
</html>
```

[See also Chapter 4.](#)

Chapter 1

This chapter explains ba bla bla

Chapter 2

This chapter explains ba bla bla

Chapter 3

This chapter explains ba bla bla

Chapter 4

This chapter explains ba bla bla

Creation of web page using Tables:-

- **Tables**

Tables are defined with the <table> tag. A table is divided into rows (with the <tr> tag), and each row is divided into data cells (with the <td> tag). The letters td stands for "table data," which is the content of a data cell. A data cell can contain text, images, lists, paragraphs, forms, horizontal rules, tables, etc.

Simple Example:-

```
<table border="1">
<tr>
<td>row 1, cell 1</td>
<td>row 1, cell 2</td>
</tr>
<tr>
<td>row 2, cell 1</td>
<td>row 2, cell 2</td>
</tr>
</table>
```

Headings in a Table :-

- Headings in a table are defined with the <th> tag.

```
<table border="1">
<tr>
<th>Heading</th>
<th>Another Heading</th>
</tr>
<tr>
<td>row 1, cell 1</td>
<td>row 1, cell 2</td>
</tr>
</table>
```

Table with caption & colspan:-

```
<caption>Cell that spans two columns:</caption>
<table border="1">
<tr>
<th>Name</th>
<th colspan="2">Telephone</th>
</tr>
<tr>
<td>Bill Gates</td>
```

```
<td>555 77 854</td>
```

```
<td>555 77 855</td>
```

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

Table with rowspan,cellspacing,cellpadding:-

<caption>Cell that spans two rows: </caption>

```
<table border="1" cellspacing="10" cellpadding="5">
```

```
<tr>
```

```
<th>First Name</th>
```

```
<td>Bill Gates</td>
```

```
</tr>
```

```
<tr>
```

```
<th rowspan="2">Telephone</th>
```

```
<td>555 77 854</td>
```

```
</tr>
```

```
<tr>
```

```
<td>555 77 855</td>
```

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

Add a background color or a background image to a table:-

```
<html><body>
<h4>A background color:</h4>
<table border="1"
bgcolor="red">
<tr>
<td>First</td>
<td>Row</td>
</tr>
<tr>
<td>Second</td>
<td>Row</td>
</tr>
</table>
<h4>A background image:</h4>
<table border="1" background="bgdesert.jpg">
<tr>
<td>First</td>
<td>Row</td>
</tr>
<tr>
<td>Second</td>
<td>Row</td>
</tr></table></body></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:

The web developer must keep track of more

HTML documents.

It is difficult to print the entire page.

The Frameset Tag:-

- The <frameset> tag defines how to divide the window into frames.
- Each frameset defines a set of rows **or** columns.
- The values of the rows/columns indicate the amount of screen area each row/column will occupy.

- **Example:-**

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

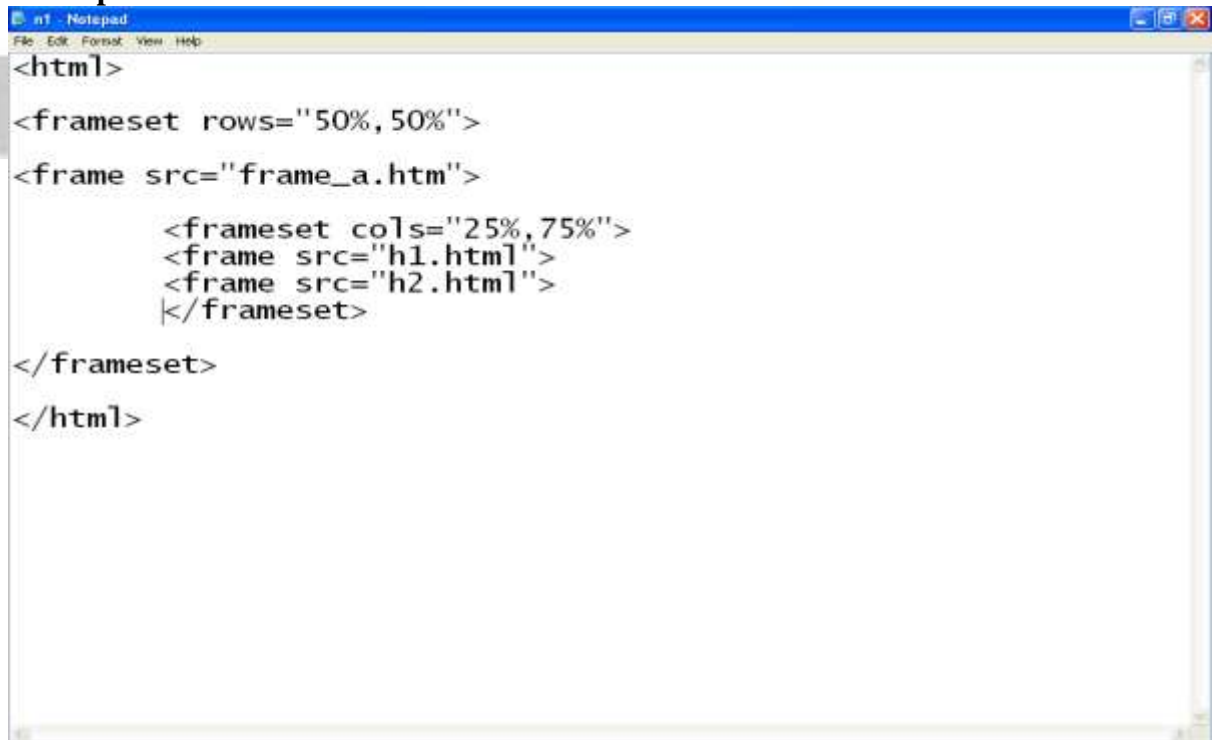
```
<frame src="frame_a.htm">
```

```
<frame src="frame_b.htm">
```

```
</frameset>
```

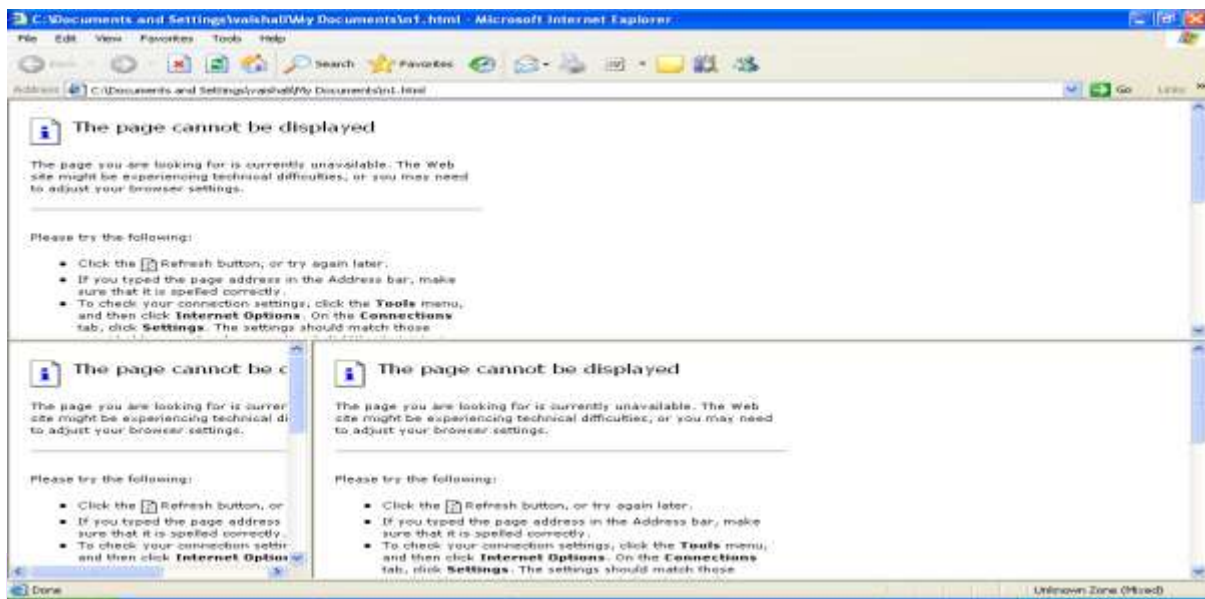
Note: The frameset column size value can also be set in pixels (cols="200,500"), and one of the columns can be set to use the remaining space (cols="25%,*").

Example with Mixed frame



```
<html>
<frameset rows="50%,50%">
<frame src="frame_a.htm">
    <frameset cols="25%,75%">
    <frame src="h1.html">
    <frame src="h2.html">
    </frameset>
</frameset>
</html>
```

Output:-



- 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.
- 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!

Frames with no resize :-

```
<html>
```

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

```
<frame noresize="noresize" src="frame_a.htm">
```

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

```
<frame noresize="noresize" src="frame_b.htm">
```

```
<frame noresize="noresize" src="frame_c.htm">
```

```
</frameset>
```

```
</frameset>
```

```
</html>
```

- This example demonstrates the no resize attribute. The frames are not resizable. Move the mouse over the borders between the frames and notice that you can not move the borders.

Navigation frames:-

- The navigation frame contains a list of links with the second frame as the target.
- The file called "tryhtml_contents.htm" contains three links.

- The source code of the links:

```
<a href="frame_a.htm" target="showframe">Frame a</a><br>  
<a href="frame_b.htm" target="showframe">Frame b</a><br>  
<a href="frame_c.htm" target="showframe">Frame c</a>  
The second frame will show the linked document.
```

```
<html>  
<frameset cols="120,*">  
<frame src="tryhtml_contents.htm">  
<frame src="frame_a.htm" name="showframe">  
</frameset>  
</html>
```

[Frame a](#)
[Frame b](#)
[Frame c](#)

Frame C

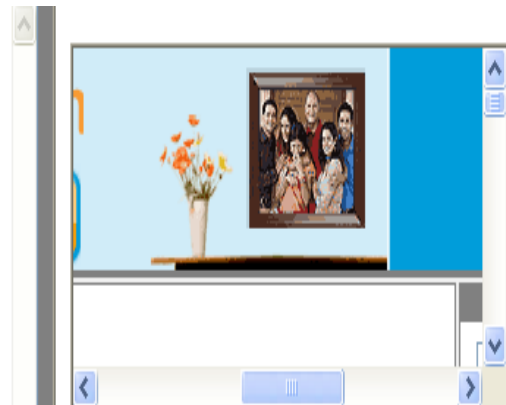
Inline frames:- (Example)

```
<html>
<body>

<iframe src="default.asp"></iframe>

<p>Some older browsers don't support iframes.</p>
<p>If they don't, the iframe will not be visible.</p>

</body>
</html>
```



Forms:-

- A form is an area that can contain form elements.
- Form elements are elements that allow the user to enter information (like text fields, textarea fields, drop-down menus, radio buttons, checkboxes, etc.) in a form.
- A form is defined with the `<form>` tag.

```
<form>
```

```
<input>
```

```
<input>
```

```
</form>
```

■ Input

The most used form tag is the `<input>` tag. The type of input is specified with the `type` attribute. The most commonly used input types are explained below.

■ Text Fields

Text fields are used when you want the user to type letters, numbers, etc. in a form.

■ `<form>` First name:

```
<input type="text" name="firstname">
```

```
<br> Last name:
```

```
<input type="text" name="lastname">
```

```
</form>
```

- How it looks in a browser:

- First name:
- Last name:

- Note that the form itself is not visible. Also note that in most browsers, the width of the text field is 20 characters by default.

Radio Buttons:

- Radio Buttons are used when you want the user to select one of a limited number of choices.

```
<form>
<input type="radio" name="abc" value="male"> Male
<br>
<input type="radio" name="abc" value="female"> Female </form>
```

- How it looks in a browser:
- Male
- Female
- Note that only one option can be chosen.

Checkboxes:-

- Checkboxes are used when you want the user to select one or more options of a limited number of choices.
- `<form>` I have a bike:

```
<input type="checkbox" name="vehicle" value="Bike">
<br> I have a car:
<input type="checkbox" name="vehicle" value="Car">
<br> I have an airplane:
<input type="checkbox" name="vehicle" value="Airplane"> </form>
```

- I have a bike: ☐
- I have a car: ☐
- I have an airplane: ☐

The Form's Action Attribute and the Submit Button:-

- When the user clicks on the "Submit" button, the content of the form is sent to another file. The form's action attribute defines the name of the file to send the content to. The file defined in the action attribute usually does something with the received input.
- `<form name="input" action="html_form_action.asp" method="get">`
 - Username:

- `<input type="text" name="user">`
- `<input type="submit" value="Submit">`
- `</form>`
- How it looks in a browser:

○ Username:

SUBMIT

Textarea:-

```
<html>
```

```
<body>
```

```
<p>
```

This example cannot be edited
because our editor uses a textarea
for input,
and your browser does not allow
a textarea inside a textarea.

```
</p>
```

```
<textarea rows="10" cols="30">
```

The cat was playing in the garden.

```
</textarea>
```

```
</body>
```

```
</html>
```

The cat was playing in the garden.

Select Tag:-

```
<html>
```

```
<body>
```

```
<form action="">
```

```
<select name="cars">
```

```
<option value="volvo">Volvo</option>
```

```
<option value="saab">Saab</option>
```

```
<option value="fiat">Fiat</option>
```

```
<option value="audi">Audi</option>
```

```
</select>
```

```
</form>
```

```
</body>
```

```
</html>
```



Html with Images :-

- In HTML, images are defined with the `` tag.
- The `` tag is empty, which means that it contains attributes only and it has no closing tag.
- To display an image on a page, you need to use the `src`
- attribute. `Src` stands for "source". The value of the `src` attribut is the URL of the image you want to display on your page.

- The syntax of defining an image:
- ``

To add image as Background:-

```
<html>
<body background="background.jpg">

<h3>Look: A background image!</h3>

<p>Both gif and jpg files can be used as HTML
backgrounds.</p>

<p>If the image is smaller than the page, the image will
repeat itself.</p>

</body>
</html>|
```

Output:-

Look: A background image!

Both gif and jpg files can be used as HTML backgrounds.

If the image is smaller than the page, the image will repeat itself.

Image alignment:-

```
<p>
```

An image

```

```

in the text

```
</p>
```

Image as a hyperlink:-

```
<html>
```

```
<body>
```

```
<p>
```

You can also use an image as a link:

```
<a href="lastpage.htm">
```

```

```

```
</a>
```

```
</p>
```

```
</body>
```

```
</html>
```

You can also use an image as a link:



CSS-CASCADING STYLE SHEET

What is CSS?

Cascading Style Sheets, fondly referred to as CSS, is a simple design language intended to simplify the process of making web pages presentable.

CSS handles the look and feel part of a web page. Using CSS, you can control the color of the text, the style of fonts, the spacing between paragraphs, how columns are sized and layout, what background images or colors are used, as well as a variety of other effects. CSS is easy to learn and understand but it provides a powerful control over the presentation of an HTML document. Most commonly, CSS is combined with the markup languages HTML or XHTML.

Advantages of CSS

- **CSS saves time** - You can write CSS once and then reuse the same sheet in multiple HTML pages. You can define a style for each HTML element and apply it to as many web pages as you want.
- **Pages load faster** - If you are using CSS, you do not need to write HTML tag attributes every time. Just write one CSS rule of a tag and apply it to all the occurrences of that tag. So, less code means faster download times.
- **Easy maintenance** - To make a global change, simply change the style, and all the elements in all the web pages will be updated automatically.
- **Superior styles to HTML** - CSS has a much wider array of attributes than HTML, so you can give a far better look to your HTML page in comparison to HTML attributes.
- **Multiple Device Compatibility** - Style sheets allow content to be optimized for more than one type of device. By using the same HTML document, different versions of a website can be presented for handheld devices such as PDAs and cellphones or for printing.
- **Global web standards** – Now HTML attributes are being deprecated and it is being recommended to use CSS. So it's a good idea to start using CSS in all the HTML pages to make them compatible with future browsers.

CSS — SYNTAX 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 a 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*, *border*, etc.

Value: Values are assigned to properties. For example, *color* property can have the 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 the 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, the style rule will apply to `` element only when it lies inside the `` 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 a 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 when those headings will lie within tags having *id* attribute set to *black*.

The Child Selectors

You have seen the descendant selectors. There is one more type of selector, 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 a direct child of the <body> element. Other paragraphs put inside other elements like <div> or <td> 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 the input elements having 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 **semicolon (;)**. You can keep them in a single line or multiple lines. For better readability, we keep them in separate lines. For a while, don't bother about the properties mentioned in the above block. These properties will

be explained in the coming chapters and you can find the complete detail about properties in CSS References.

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 the various *class* selectors together as shown below:

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

CSS – INCLUSION

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

Example

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

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

It will produce the following result:

This is inline CSS

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="..." />
</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.
Media	screen tty projection handheld print braille aural all	Specifies the device the document will be displayed on. Default value is <i>all</i> . This is an 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" 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 an HTML document:

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

CSS Rules Overriding

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

- Any inline style sheet takes the highest priority. So, it will override any rule defined in `<style>...</style>` tags or the rules defined in any external style sheet file.
- Any rule defined in `<style>...</style>` tags will override the rules defined in any external style sheet file.
- Any rule defined in the external style sheet file takes the lowest priority, and the rules defined in this file will be applied only when the 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 to 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 the style sheet. You can simply 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 — MEASUREMENT UNITS

Before we start the actual exercise, we 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 along with 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

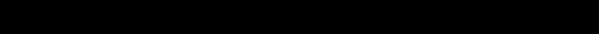
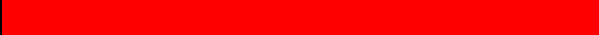







CSS uses color values to specify a color. Typically, these are used to set a color either for the foreground of an element (i.e., its text) or for the background of the element. They can also be used to affect the color of borders and other decorative effects. You can specify your color values in various formats. Following table lists all the 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;}

These formats are explained in more detail in the following sections:

CSS Colors - Hex Codes

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). A hexadecimal value can be taken from any graphics software like Adobe Photoshop, Jasc Paintshop Pro, or even using Advanced Paint Brush. Each hexadecimal code will be preceded by a pound or hash sign '#'. Following are the examples to use Hexadecimal notation.

Color	Color Hex
	#000000
	#FF0000
	#00FF00
	#0000FF
	#FFFF00
	#00FFFF
	#FF00FF
	#C0C0C0
	#FFFFFF

CSS Colors - Short Hex Codes

This is a shorter form of the six-digit notation. In this format, each digit is replicated to arrive at an equivalent six-digit value. For example: #6A7 becomes #66AA77. A hexadecimal value can be taken from any graphics software like Adobe Photoshop, Jasc Paintshop Pro or even using Advanced Paint Brush. Each hexadecimal code will be preceded by a pound or hash sign #. Following are the examples to use the Hexadecimal notation.

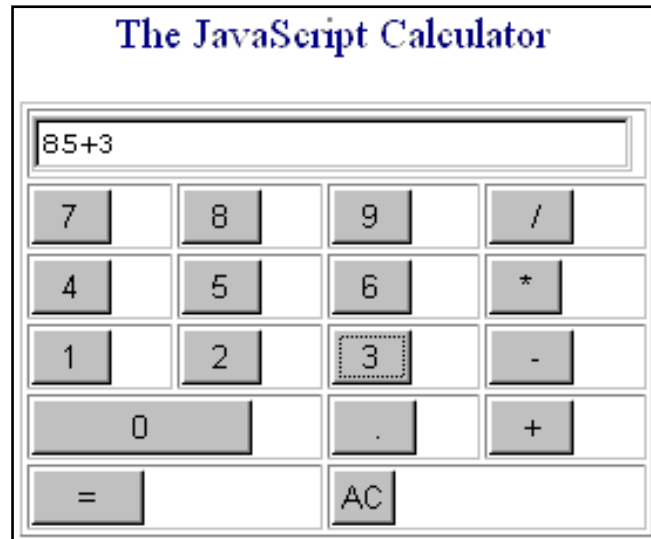
JAVASCRIPT

Client-side programming with JavaScript

- What is JavaScript
- scripts vs. programs
 - JavaScript vs. JScript vs. VBScript
 - common tasks for client-side scripts
- JavaScript
 - data types & expressions
 - control statements
 - functions & libraries
 - strings & arrays
 - Date, document, navigator, user-defined classes
- Browsers have limited functionality
 - Text, images, tables, frames
- JavaScript allows for interactivity
- Browser/page manipulation
 - Reacting to user actions
- A type of programming language
 - Easy to learn
 - Developed by Netscape
 - Now a standard exists –
www.ecma-international.org/publications/standards/ECMA-262.HTM

JavaScript Allows Interactivity

- Improve appearance
 - Especially graphics
 - Visual feedback
- Site navigation
- Perform calculations
- Validation of input
- Other technologies



How Does It Work?

- Embedded within HTML page
 - View source
- Executes on client
 - Fast, no connection needed once loaded
- Simple programming statements combined with HTML tags
- Interpreted (not compiled)
 - No special tools required

Learning JavaScript

- Special syntax to learn
- Learn the basics and then use other people's (lots of free sites)
- Write it in a text editor, view results in browser
- You need to revise your HTML
- You need patience and good eyesight!

Client-Side Programming

- HTML is good for developing *static* pages
 - can specify text/image layout, presentation, links, ...
 - Web page looks the same each time it is accessed
 - in order to develop interactive/reactive pages, must integrate programming in some form or another
- client-side programming
 - programs are written in a separate programming (or scripting) language

e.g., JavaScript, JScript, VBScript

- programs are embedded in the HTML of a Web page, with (HTML) tags to identify the program component

e.g., `<script type="text/javascript"> ... </script>`

- the browser executes the program as it loads the page, integrating the dynamic output of the program with the static content of HTML
- could also allow the user (client) to input information and process it, might be used to validate input before it's submitted to a remote server

Scripts vs Programs

- A scripting language is a simple, interpreted programming language
 - scripts are embedded as plain text, interpreted by application
 - *simpler execution model*: don't need compiler or development environment
 - *saves bandwidth*: source code is downloaded, not compiled executable
 - *platform-independence*: code interpreted by any script-enabled browser
 - *but*: slower than compiled code, not as powerful/full-featured.
- JavaScript: the first Web scripting language, developed by Netscape in 1995
- syntactic similarities to Java/C++, but simpler, more flexible in some respects,
- limited in others
- (loose typing, dynamic variables, simple objects)
- JScript: Microsoft version of JavaScript, introduced in 1996
- same core language, but some browser-specific differences
- fortunately, IE, Netscape, Firefox, etc. can (mostly) handle both
- JavaScript & JScript
- *JavaScript 1.5 & JScript 5.0 cores both conform to ECMAScript standard*
- VBScript: client-side scripting version of Microsoft Visual Basic

Common Scripting Tasks

- adding dynamic features to Web pages
 - validation of form data (probably the most commonly used application)
 - image rollovers
 - time-sensitive or random page elements
 - handling cookies
 - Ajax adds ability to update page elements without (re)loading a webpage
 - defining programs with Web interfaces

- utilize buttons, text boxes, clickable images, prompts, etc
- **limitations of client-side scripting**
 - since script code is embedded in the page, it is viewable to the world
 - for security reasons, scripts are limited in what they can do

e.g., can't access the client's hard drive

- since they are designed to run on any machine platform, scripts do not contain platform specific commands
- script languages are not full-featured

e.g., JavaScript objects are very crude, not good for large project development

JavaScript Statements

```
<html>
<head><title>My Page</title></head>
<body>
<script language="JavaScript">
document.write('This is my first →
JavaScript Page');
</script>
</body>
</html>
```

Note the symbol for line continuation

How to Put a JavaScript Into an HTML Page?

```
<html>
<body>
<script type="text/javascript">
document.write("Hello World!")
</script>
</body>
</html>
```

Ending Statements With a Semicolon?

- With traditional programming languages, like C++ and Java, each code statement has to end with a semicolon (;).
- Many programmers continue this habit when writing JavaScript, but in general, semicolons are **optional**! However, semicolons are required if you want to put more than one statement on a single line.

JavaScript Statements

```
<html>
<head><title>My Page</title></head>
<body>
<script language="JavaScript">
document.write('<b1>This is my first →
JavaScript Page</b1>');
</script>
</body>
</html>
```

HTML written inside JavaScript

JAVASCRIPT STATEMENTS

```
<html>
<head><title>My Page</title></head>
<body>
<p>
<a href="myfile.html">My Page</a>
<br />
<a href="myfile.html"
onmouseover="window.alert('Hello');">
My Page</A>
</p>
</body>
</html>
```

An Event

JS Written inside HTML



JavaScript Popup Boxes

- **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.

```
<script>  
alert("Hello World!")  
</script>
```

JavaScript Popup Boxes – 2

- **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.

JavaScript Popup Boxes – 3

- **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.

Prompt Box Example

```
<script>  
x=prompt ("Error 404", "")  
document.write("sorry<br>",+x)  
</script>
```

JS Examples -1

$Y=20x+12$ if $x=3$, What is the value of Y ?

```
<script>  
x=3  
y=20*x+12
```

```
alert(y)
</script>
```

JS Examples -2

```
<script>
s1=12
s2=28
sum= s1+s2
document.write("SUM= "+ sum)
</script>
```

Webpage Embeeding:

- JavaScript code can be embedded in a Web page using <script> tags
 - the output of JavaScript code is displayed as if directly entered in HTML
- document.write displays text in the page
- text to be displayed can include HTML tags
- the tags are interpreted by the browser as normal when the text is displayed
- as in C++/Java, statements end with ;
- but a line break might also be interpreted as the end of a statement (depends upon browser)
- JavaScript comments similar to C++/Java
 - // starts a single line comment
 - /*...*/ enclose multi-line comments

```
<html>
<head>
  <title>JavaScript Page</title>
</head>
<body>
  <script type="text/javascript">
    // silly code to demonstrate output
    document.write("<p>Hello world!</p>");
```

```

document.write(" <p>How are <br/> " +
               " <i>you</i>?</p> ");
</script>
<p>Here is some static text as well.</p>
</body>
</html>

```

Conditional Statements

- Very often when you write code, you want to perform different actions for different decisions. You can use conditional statements in your code to do this.

In JavaScript we have the following conditional statements:

- **if statement** - use this statement if you want to execute some code only if a specified condition is true
- **if...else statement** - use this statement if you want to execute some code if the condition is true and another code if the condition is false
- **if...else if...else statement** - use this statement if you want to select one of many blocks of code to be executed
- **switch statement** - use this statement if you want to select one of many blocks of code to be executed

JavaScript Data Types & Variables

- JavaScript has only three primitive data types

String : "foo" 'how do you do?' "I said 'hi'." ""

Number: 12 3.14159 1.5E6

Boolean : true false (Null, Undefined have special meaning in JavaScript)

assignments are as in C++/Java

```
message = "howdy";
```

```
pi = 3.14159;
```

variable names are sequences of letters, digits, and underscores that *start with a letter or an underscore*

variables names are case-sensitive

you don't have to declare variables, will be created the first time used, but it's better if you use var statements (because of scoping issues)

```
var message, pi=3.14159;
```

variables are loosely typed, can be assigned different types of values (Danger!)

```
<html>
```

```

<head>
  <title>Data Types and Variables</title>
</head>
<body>
  <script type="text/javascript">
    var x, y;
    x= 1024;
    y=x; x = "foobar";
    document.write("<p>x = " + y + "</p>");
    document.write("<p>x = " + x + "</p>");
  </script>
</body>
</html>

```

JavaScript Operators & Control Statements

standard C++/Java operators & control statements are provided in JavaScript

- +, -, *, /, %, ++, --, ...
- ==, !=, <, >, <=, >=
- &&, ||, !, ===, !==
- if, if-else, switch
- while, for, do-while, ...

PUZZLE: Suppose you took a piece of paper and folded it in half, then in half again, and so on.

How many folds before the thickness of the paper reaches from the earth to the sun?

```

<html>
<head>
  <title>Folding Puzzle</title>
</head>
<body>
  <script type="text/javascript">
    var distanceToSun = 93.3e6*5280*12;
    var thickness = .002;
    var foldCount = 0;

```

```

while (thickness < distanceToSun) {
    thickness *= 2;
    foldCount++;
}

document.write("<p>Number of folds = " +
    foldCount+"</p>");

</script>
</body>
</html>

```

JavaScript Math Routines

The built-in Math object contains many functions and constants

1. Math.sqrt
2. Math.pow
3. Math.abs
4. Math.max
5. Math.min
6. Math.floor
7. Math.ceil
8. Math.round
9. Math.PI
10. Math.E
11. Math.random function returns a real number in [0..1)

```

<html>
<head>
    <title>Random Dice Rolls</title>
</head>
<body>
    <div style="text-align:center">
        <script type="text/javascript">
            var roll1 = Math.floor(Math.random()*6) + 1;
            var roll2 = Math.floor(Math.random()*6) + 1;
            document.write("<img src='http://www.csc.liv.ac.uk/' +
                '~martin/teaching/comp519/Images/die' +
                roll1 + ".gif" alt='die showing " + roll1 + "'/>");
            document.write("&nbsp;&nbsp;&nbsp;");
            document.write("<img src='http://www.csc.liv.ac.uk/' +

```

```

    "~martin/teaching/comp519/Images/die" +
    roll2 + ".gif" alt='die showing ' + roll2 + "'/>");
</script>
</div>
</body>
</html>

```

Interactive Pages Using Prompt

somewhat crude user interaction can take place using prompt

1st argument: the prompt message that appears in the dialog box

2nd argument: a default value that will appear in the box (in case the user enters nothing)

the function returns the value (as a string) entered by the user in the dialog box

if value is a number, must use parseFloat (or parseInt) to convert

we will see other ways to interact with a user later

```

<html>
<head>
  <title>Interactive page</title>
</head>
<body> <p>
<script type="text/javascript">
  var userName = prompt("What is your name?", "");
  var userAge = prompt("Your age?", "");
  var userAge = parseFloat(userAge);

  document.write("Hello " + userName + ".")
  if (userAge < 18) {
    document.write(" Do your parents know " +
      "you are online?");
  }
  else {
    document.write(" Welcome friend!");
  }
</script> </p>

```

<p>The rest of the page...</p>

</body>

</html>

User-Defined Functions

- function definitions are similar to C++/Java, except:
 - no return type is specified for the function (since variables are loosely typed)
 - no variable typing for parameters (since variables are loosely typed)
 - by-value parameter passing only (parameter gets copy of argument)
- Can limit variable scope to the function.
- if the first use of a variable is preceded with var, then that variable is local to the function

for modularity, should make all variables in a function local

```
function isPrime(n)
```

```
// Assumes: n > 0 and is an integer
```

```
// Returns: true if n is prime, else false
```

```
{
```

```
  if (n < 2) {
```

```
    return false;
```

```
  }
```

```
  else if (n == 2) {
```

```
    return true;
```

```
  }
```

```
  else {
```

```
    for (var i = 2; i <= Math.sqrt(n); i++) {
```

```
      if (n % i == 0) {
```

```
        return false;
```

```
      }
```

```
    }
```

```
    return true;
```

```
  }
```

```
}
```


Function Example

Function definitions (usually) go in the <head> section

<head> section is loaded first, so then the function is defined before code in the <body> is executed (and, therefore, the function can be used later in the body of the HTML document)

```
<html>
<head>
  <title>Prime Tester</title>
  <script type="text/javascript">
    function isPrime(n)
      // Assumes: n > 0
      // Returns: true if n is prime
      {
        // CODE AS SHOWN ON PREVIOUS SLIDE
      }
  </script>
</head>
<body> <p>
  <script type="text/javascript">
    testNum = parseFloat(prompt("Enter a positive integer", "7"));

    if (isPrime(testNum)) {
      document.write(testNum + " <b>is</b> a prime number.");
    }
    else {
      document.write(testNum + " <b>is not</b> a prime number.");
    }
  </script> </p>
</body>
</html>
```

JavaScript Objects

- an object defines a new type (formally, *Abstract Data Type*)
 - encapsulates data (properties) and operations on that data (methods)
- for example, a String object encapsulates a sequence of characters, enclosed in quotes

properties of a String include

- `length` :stores the number of characters in the string

methods include

- `charAt(index)`: returns the character stored at the given index (as in C++/Java, indices start at 0)
- `substring(start, end)`: returns the part of the string between the start (inclusive) and end (exclusive) indices
- `toUpperCase()`: returns copy of string with letters uppercase
- `toLowerCase()`: returns copy of string with letters lowercase

to create a **String** in JavaScript, assign using `new` keyword or (in the case of a String) just make a direct assignment (`new` is implicit for a String assignment)

```
word = new String("foo");      word = "foo";
```

properties/methods are called exactly as in C++/Java

```
word.length      word.charAt(0)
```

String example: Palindromes

suppose we want to test whether a word or phrase is a palindrome

noon Radar

Madam, I'm Adam.

A man, a plan, a canal:

Panama!

must strip non-letters out of the word or phrase

make all chars uppercase in order to be case-insensitive

finally, traverse and compare chars from each end

function `strip(str)`

```
// Assumes: str is a string
```

```
// Returns: str with all but letters removed
```

```
{
```

```
  var copy = "";
```

```
  for (var i = 0; i < str.length; i++) {
```

```
    if ((str.charAt(i) >= "A" && str.charAt(i) <= "Z") ||
```

```

        (str.charAt(i) >= "a" && str.charAt(i) <= "z")) {
        copy += str.charAt(i);
    }
}

return copy;
}

function isPalindrome(str)
// Assumes: str is a string
// Returns: true if str is a palindrome, else false
{
    str = strip(str.toUpperCase());

    for(var i = 0; i < Math.floor(str.length/2); i++) {
        if (str.charAt(i) != str.charAt(str.length-i-1)) {
            return false;
        }
    }
    return true;
}

```

JavaScript Arrays

- arrays store a sequence of items, accessible via an index

since JavaScript is loosely typed, elements do not have to be the same type

- to create an array, allocate space using new (or can assign an array directly using the proper syntax shown below)

```
items = new Array(10); // allocates space for 10 items (but can grow)
```

```
items = new Array(); // if no size given, will adjust dynamically
```

```
items = [0,0,0,0,0,0,0,0,0,0]; // can assign size & values []
```

- to access an array element, use [] (as in C++/Java)

```
for (i = 0; i < 10; i++) {
```

```
    items[i] = i; // initializes array with values 0, ..., 9
```

```
}
```

- the length property stores the number of items in the array

```

for (i = 0; i < items.length; i++) {
    document.write(items[i] + "<br/>"); // displays elements one line at a
                                     // time
}

```

Array Example

suppose we want to simulate die rolls and verify even distribution

keep an array of counters:

initialize each count to 0

each time you roll X, increment rolls[X]

display each counter

```

<html>
<!-- COMP519 js10.html 11.10.2014 -->
<head>
<title>Die Statistics</title>
<script type="text/javascript"
src="http://www.csc.liv.ac.uk/~martin/teaching/comp519/JS/random.js">
</script>
</head>
<body> <p>
<script type="text/javascript">
    numRolls = 60000;
    dieSides = 6;
    rolls = new Array(dieSides+1);
    for (i = 1; i < rolls.length; i++) {
        rolls[i] = 0;
    }
    for(i = 1; i <= numRolls; i++) {
        rolls[randomInt(1, dieSides)]++;
    }
    for (i = 1; i < rolls.length; i++) {
        document.write("Number of " + i + "'s = " +
            rolls[i] + "<br />");
    }

```

```

    }
</script> </p>
</body>
</html>

```

- Arrays have predefined methods that allow them to be used as stacks, queues, or other common programming data structures.

```

var stack = new Array();
stack.push("blue");    // append the string "blue" to array
stack.push(12);        // stack is now the array ["blue", 12]
stack.push("green");   // stack = ["blue", 12, "green"]
var item = stack.pop(); // item is now equal to "green"
                        // and stack = ["blue", 12] again
var q = [1,2,3,4,5,6,7,8,9,10];
item = q.shift();      // item is now equal to 1, remaining
                        // elements of q move down one position
                        // in the array, e.g. q[0] equals 2
q.unshift(125); // q is now the array [125,2,3,4,5,6,7,8,9,10]
q.push(244);    // q = [125,2,3,4,5,6,7,8,9,10,244]

```

Date Object

- String & Array are the most commonly used objects in JavaScript
 - other, special purpose objects also exist
- for example, the Date object can be used to access the date and time, and to perform "date arithmetic"
 - to create a Date object, use new & supply year/month/day/... as desired

```

today = new Date();          // sets to current date & time
newYear = new Date(2002,0,1); //sets to Jan 1, 2002 12:00AM

```

- methods include:

```

newYear.getYear()           can access individual components of a date
newYear.getMonth()
newYear.getDay()
newYear.getHours()
newYear.getMinutes()
newYear.getSeconds()

```

`newYear.getMilliseconds()`

`document` Object

Internet Explorer, Firefox, Opera, etc. allow you to access information about an HTML document using the document object

`document.write(...)`

method that displays text in the page

`document.URL`

property that gives the location of the HTML document

`document.lastModified`

property that gives the date & time the HTML document was last changed

```
<html>
<head>
  <title>Documentation page</title>
</head>
<body>
  <table width="100%">
    <tr>
      <td><i>
        <script type="text/javascript">
          document.write(document.URL);
        </script>
      </i></td>
      <td style="text-align: right;"><i>
        <script type="text/javascript">
          document.write(document.lastModified);
        </script>
      </i></td>
    </tr>
  </table>
</body>
</html>
```

HTML Forms and JavaScript

- JavaScript is very good at processing user input in the web browser
- HTML `<form>` elements receive input
- Forms and form elements have unique names
 - Each unique element can be identified
 - Uses JavaScript Document Object Model (DOM)

Naming Form Elements in HTML

Name:

Phone:

Email:

```
<form name="addressform">
```

```
Name: <input name="yourname"><br />
```

```
Phone: <input name="phone"><br />
```

```
Email: <input name="email"><br />
```

```
</form>
```

Forms and JavaScript

document.formname.elementname.value

Thus:

document.**addressform.yourname**.value

document.addressform.phone.value

document.addressform.email.value

Name:

Phone:

Email:

Using Form Data

Personalising an alert box

```
<form name="alertform">
```

Enter your name:

```
<input type="text" name="yourname">
```

```
<input type="button" value="Go" onClick="window.alert('Hello ' + →  
document.alertform.yourname.value);">
```

```
</form>
```

