LECTURE NOTES

ON

WEB TECHNOLOGIES

B.Tech V Sem (IARE-R16)

By

Mr A KRISHNA CHAITANYA
Assistant Professor

DEPARTEMENT OF INFORMATION TECHNOLOGY

INSTITUTE OF AERONAUTICAL ENGINEERING
(Autonomous)
DUNDIGAL, HYDERABAD-500 043

UNIT-I
Introduction to HTML and Java Script

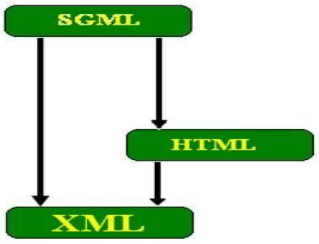
Introduction to html, fundamentals of HTML elements, Document body, text, hyperlink, lists, tables, color and images, frames; Cascading Style Sheets: Introduction, defining your own styles, properties and values in styles, style sheets, formatting blocks, and layers;
JavaScript: JavaScript basics, variables, string manipulation, mathematical functions, statements, operators, arrays and functions.

What is HTML?

Markup Language

- A Markup language defines the rules that help to add meaning to the content and structure of documents
- **They are classified as :**
  - **Stylistic Markup** – It determines the presentation of the document
  - **Structure Markup** – It defines the structure of the document
  - **Semantic Markup** – It determines the contents of the document

Hierarchy of Markup Language



Tools Used for ManualMarkup

HTML and Organization

1. HTML is a subset of Standard Generalized Markup Language (SGML)
2. HTML is specified by the **World Wide Web Consortium** (W3C).

HTML describes Web Page -

- HTML stands for **H**yper **T**ext **M**arkup **L**anguage
- HTML is not a programming language.
- HTML is a **markup language**
- Markup Language Consists of set of Tags called –**markup tags**–.
- Markup Tags are used **to describe Web Page using Markup Tags**.
- HTML elements are basic building blocks for the construction of web pages.

What we can do using HTML ?

- We can **Create Web Page** using HTML.
- We can **Format Specific Part of Web Page** using HTML.
- We can **Embed JavaScript** in order to provide dynamic feature.
- We can **Apply Styles to Webpage Using CSS** to look Web Page more interactive and Fresh.
- We can **Provide Server Side Execution Facility** by embedding Server Side Scriptinginside HTML.
- **Markup languages** are widely used in everyday computing. For instance, **word processors** use codes that indicate the **structure** and **presentation** of a **document.**
- The **word processors** write, behind the scenes, the necessary markup to produce a document that you see on the screen.
- A **web document** is viewed in a web browser, like the one you are using to read this document.

- Just like in a word processor you can specify the appearance of text, you can write HTML code to specify how the text or content of a web page should look.

Versions of HTML

- The **set of rules** under which **HTML** operates is known as **syntax.**
  The **syntax** or the rules for creating webpages is developed by **World Wide Consortium (or W3C)**. Access the **versions of HTML page** for more information.

HTML editors

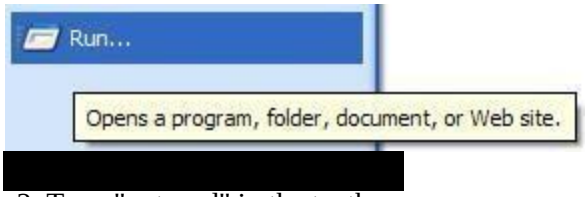
- There are so many software packages available to develop HTML. The software packages can be grouped into two main categories:

1. **text-based** (or code-based) editors
2. **WYSIWYG** (*what you see is what you get*) editors

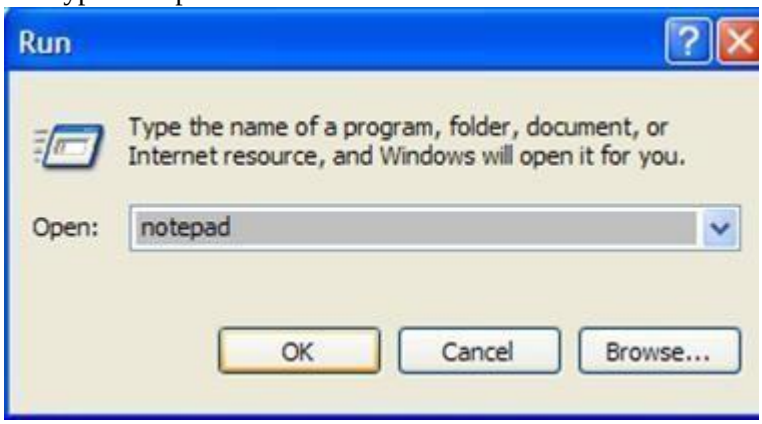
Text-based (or code-based) editors

- To start creating **web pages**, you do not need an expensive software package but you do need some knowledge of HTML. You can create web pages with a basic text editor like Windows Notepad. However, as you master scripting HTML, you will learn that using Notepad or some other basic text editor is not sufficient. In Windows, *Notepad* can be started from the Start Menu:
- Select: Programs | Accessories | Notepad
- Type notepad.exe or just notepad in the Run command from the Start Menu:

1. Start the Run Command



2. Type "notepad" in the textbox.



- Next, click on the OK button to start Notepad.
- ***Macromedia Homesite*** **i**s a popular **text-based HTML editor**. The editor provides many HTML-specific options that are not available in the Windows Notepad.

WYSIWYG editors

- Because WYSIWYG (pronounced *wuzzywig* or *wizzywig*) HTML editors do not require much HTML knowledge, they tend to be expensive. These editors allows you to directly work in the "design" or "preview" view instead of the code view.
- The main **advantage** of working with the ***design* view** is that you can design the layout of your page by dragging-and-dropping pieces of your page layout.

Thus a **web page** can be developed more **quickly** than by hard-coding it by hand using a text-based editor.

- There are several popular **WYSIWYG editors** available:
  1. ***Macromedia Dreamweaver***
  2. ***Microsoft FrontPage***
  3. ***Adobe GoLive***
- Remember these editors can help you **create web pages** but they cannot teach you HTML. If you first master HTML, you will utilize these software packages more efficiently.

Text-based editors versus WYSIWYG editors

Text-based (or code-based) editors	WYSIWYG editors
1Better control: Because text-based editors require knowledge of HTML, the developer has more control over what is written to produce a web page. In some cases, a software package (like FrontPage) may write proprietary code that may not be interpreted by all the brwosers. 2. ***Faster editing:*** We can quickly change your code unlike WYSIWYG editors. WYSIWYG editors require, for example, a lot of computer-resources to start-up or load/open a page. 3. ***More flexibility***: we can edit your code directly at the desired location. This cannot be always done with WYSIWYG editors.	1. ***Support for other scripting languages:*** WYSIWYG editors provide advanced features to code in other programming/scripting languages such as JavaScript, XHTML, ASP/ASP.NET, PHP, and JSP. 2. ***Faster/simplified development***:WYSIWYG editors allows you to develop pages quickly as the software writes the necessary code in response to the layout you design for your web page. 3. ***Better organization***: Dreamweaver, for instance, allows you to define a site folder that contains all the files that make up the website. By defining a local site folder, you have many advantages including moving of files (without breaking links), searching of a particular term or tag in the entire site (without having the file open!), and FTP support to move only changed or new files to the server.

Creating an HTML document

Before you start writing code to write a web page, it is a good practice to plan ahead the appearance of the web page.

An HTML document has two elements:

- Document Content
- Tags

1. **Document content** is the information on a **web page** that the user **will see**. That information could be text or graphics, for example. As you start creating your own web pages, try finding out first what information you want to display and how you want to present it.
2. Tags are the **HTML** codes that control **how the document** content will appear. The tags, in other words, will determine whether the text will be bold, black or blue, or of font type Time New Roman or Airal.

Naming conventions

HTML files names are very important as they are used to locate or open the files. Remember these points when naming HTML files:

- Save your web page files with the **.htm** or **.html** file extension. (Both of these files represent HTML files, older systems such Windows 3.1 and DOS cannot recognize four-letter file extensions. Because the first three letters of .html and .htm are the same, those systems ignore the "l" and just recognize ".htm".)

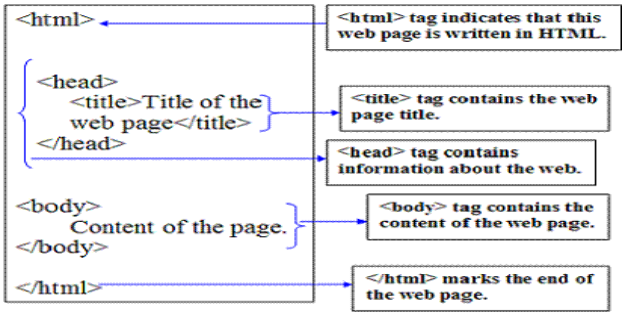
- Some web servers are **case-sensitive**. That means those web servers would consider page1.htm and Page1.htm as two different files. To avoid case sensitivity problems, use only lowercase or uppercase letters to name your files.
- Filenames should consists only of **letters** and **numbers**. Avoid using spaces, punctuation, or special characters. If you need to separate words, use dashes (-) and underscores (_), for example, creating-an-HTML-document.htm or creating_an_HTML_document.htm.

Basic structure of an HTML document

An HTML document has **two** main parts:

1. ***head.*** The head element contains title and meta data of a web document.
2. ***body.*** The body element contains the information that you want to display on a web page.

- In a **web page**, the first tag (specifically, <html>) indicates the markup language that is beingused for the document.
- The **<head>** tag contains information about the web page.
- Lastly, the content appears in the **<body>** tag.
- The following illustration provides a summary.



Understanding elements

To markup your web pages, you will need to learn the syntax (rules of a computing language) of HTML.

HTML syntax is very simple to follow. Remember the syntax only controls the presentation or structure of a web page. The most fundamental piece of building block of HTML is the *elements*.

In HTML, an element refers to **two** different things:

1. *Element of structure*of a document (for example, paragraph, web page title, etc.).
2. Element in the sense of a *tag* (for example, <p>, <title>)

Because of the different meanings of the word "element", it can be confusing what the word "element" means in a particular context.

Examples of *elements of structure* of a document	head	body	p
Examples of elements as tags	<head>	<body>	<p>

NOTE:

1. An **element** becomes a tag when we use the **angled brackets** around it. To create **a webpage**, we use tags. A tag instructs the browser what specific instruction to execute
2. Every element has a name such as head, title, p, i, and b.
3. A tag is the element name surrounded by the angled brackets. This refers to a start tag such as <p>, <title>, and <i>. A start tag starts a particular HTML instruction.
4. An **end** tag is the same as a start tag except the end tag has a forward slash between the < andthe element name. An end tag stops a particular HTML instruction.

5. Most elements have content, which is placed between the start and the end tags. Example, this is bold.
6. Some elements have no content. Such elements/tags are known as empty tags.
7. Some elements have no end tags. These are referred to as ***one-sided*** tags.
8. A tag that has an **opening** and **closing tag** is referred to as ***two-sided*** **tag**.

Understanding attributes

- In **HTML,elements** (or tags) have ***attributes*** or ***properties***. As an HTML writer, attributes allow you to add extra instruction to your tags. Because each tag has its own unique attributes, we have to learn which attribute(s) belongs which tag.
- An **attribute** has **two** parts:
    - **attribute name** and **attribute value**. Because of these two-parts, they are also referred to as *pairs*. The *attribute name* identifies (or defines) what special instruction you want to add to a particular tag.
    - The ***attribute value***, on the other hand, indicates (usually predefined) option for that attribute.
    - align="right" is an example of **attribute-value** pair.
- The word ***align*** is the attribute. The value of this attribute is *right*. A **value of an attribute** is enclosed in double quotation marks. Notice the value is on the right-hand side of the equal sign and the attribute name is on the left of equal sign.
- some **attributes** have **predefined values**. For example, for the *align* attribute, possible values include, left, center, justify and right. So if you use the *align* attribute, you should use one of these acceptable values.
- some **attributes** accept **numerical values**. For instance, for the *width* attribute, you can specify a numerical value such as 5 (which indicates 5 pixels), or 20% (which indicates 20% of the screen width).

NOTE:

- Attributes are specific to **tag names**.

For **example**, for the <p> tag, you can use the *align* attribute but not the *width* attribute. The width attribute can only be used with tags such as <table>, <td>, and <img>.

- Attributes have **values.** Make sure to use the correct value for the correct attribute. For instance, you should not use color="20", or align="brown"; instead use, color="red", and align="justify".
- Attribute values needs to be **enclosed** in double quotation marks. This is true especially if the value contains one or more spaces, for example, face="Times New Roman".
- Attribute values could come from a predefined list (such as color names red, green, blue, etc.) or (width of a table 50% or 800 pixels.)

Displaying an HTML document

After creating or as you create your web page, we should view your page at least one browser to detect any errors. If your webpage does not look the way you expect, then, you should make the necessary changes to correct any problem.

In this section, we will display a basic web page:

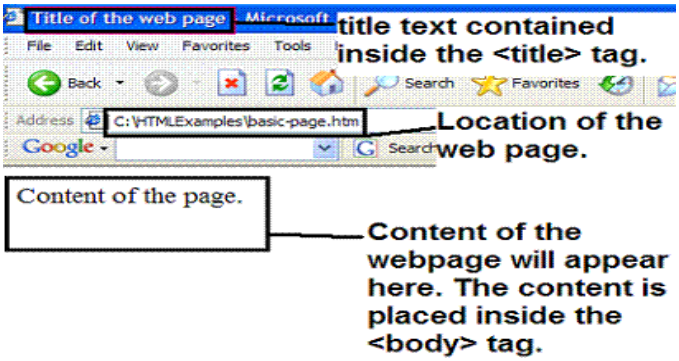


Figure 1 A simple web page with basic tags

Adding comments to HTML

As we create more pages, it may become difficult for you to understand your code in the future unless you use comments. By adding comments to our code, it could be a valuable reference guide for you when you look at your code in the future. Ideally, comments should be added to indicate what the code does. Comments are personal; so add anything that you think is impotent about your code. Remember, as you add comments to your pages, the file size of the web page will increase. That, in turn, may increase web page download time. So avoid use of excessive comments.

```
<table width="800" border="0" cellpadding="0">
  <tr>
    <!--cell 1: navigational links-->
    <td width="100"> </td>
    <!--cell 2: main page content-->
    <td width="600">
        <table width="100%" border="0" cellpadding="0">
            <tr>
                <td> </td>
            </tr>
        </table>
    </td>
    <!--cell 3: side bar cell-->
    <td width="100"> </td>
  </tr>
</table>
```

Examples of comments in HTML

Examples of comments in HTML

- In **HTML**, a comment begins with <!-- and ends with -->. Any text you place after <!-- iscomment. Browsers ignore comment text. Again, comment is for your reference; it does not get displayed on the web page.
- In a comment, you can freely include special characters, such as ampersands, quotation marks, and angle brackets. Your comments can also span multiple lines. The browser will stop ignoring text once it reads -->.
- **Example of a single line comment:**

<!--This is a small comment-->

- **Example of a multi-line comment:**

<!--This comment is long. It is displayed on more than one line. Adding multi-line comments in HTML is easy as adding a single line comment. Whether the comment is single line or multi-line, it starts with <-- and ends with -->-->.

Important points about comments:

- Use comments as a reference guide. Avoid use of excessive comments.
- Add comments to major parts of your code; or for parts, that you are unsure of.

- Use of comments is a great way to communicate with other people working on the same webpage.
- Start your comment with <-- and end with -->
- Browsers and search engines ignore comments.

Making your HTML code readable

If you do not use space effectively in your HTML code, your code may look as:

```
<html> <head> <title>Make your HTML code readable by using
white space</title> </head> <body> <p>Do you want an easy way
to understand your code? If yes, use white space to organize your
code. You can use white space as much as possible to make your
code readable. You will be surprised how easy it is to follow your
code when you use appropriate amount of white space. If you do not
use white space, your HTML code will look messy. So: </p> <ul>
<li>group related tags and separate them by indenting</li> <li>use
white space</li> <li>use comments when necessary to explain what
your code does</li> </ul> </body> </html>
```

Reading and understanding this code is difficult because it is so disorganized. Because extra white space is ignored by the browsers, you can use that to your advantage to organize your code. Use of white space to separate tags helps you better understand and maintain your pages. The following shows the updated code with the appropriate amount of space:

```
<html>
  <head>
    <title>
      Make your HTML code readable by using white space
    </title>
  </head>
  <body>
    <p>
      Do you want an easy way to understand your code? If yes, use
      white space to organize your code. You can use white space
      as much as possible to make your code readable. You will be
      surprised how easy it is to follow your code when you use
      appropriate amount of white space. If you do not use white
      space, your HTML code will look messy. So:
    </p>
    <ul>
      <li>group related tags and separate them by indenting</li>
      <li>use white space</li>
      <li>use comments when necessary to explain what your code
      does</li>
    </ul>
  </body>
</html>
```

Spacing and breaks

The amount of space you use for your HTML code can really affect how quickly and easily you understand your code. Although there are no hard rules about how much space you can and cannot use, the following identifies areas of HTML code for proper amount of spacing:

1. ***breaks between tags***
2. ***spacing between tags***
3. ***spacing inside the brackets***

1. Breaks between tags:

The first place where you can add breaks is to the following basic tags: <html>, <head>, <title></title>, </head><body>, </body> and </html>.

The following shows an example of breaks between these basic and other tags:

Breaks between tags

2. Spacing between tags:

In addition to adding breaks, you can also add space (or tabs) to enhance the understanding of your code. Because the <html> tag is the root tag of an HTML document, you may not need to use any extra spacing for this tag.

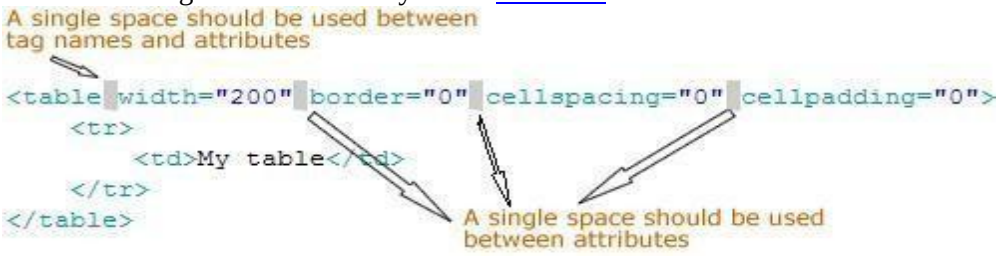
All other tags, however, inside of this root tag should be indented:

Spacing between tags

As indicated above, the <head>, and <body> tags and their corresponding closing tags are indented once. The inner tags (<title>, <h6>, and <p>) inside these main tags are indented twice.

3. Spacing inside the brackets:

Lastly, you can add space inside the angle brackets of a tag to enhance the understanding of your code. You can add space inside the angle brackets when you use attributes:



Spacing inside the brackets

As the image above shows, the first attribute is separated from the tag by one space. Similarly, subsequent attributes are separated by one space.

Avoiding unnecessary space in HTML code

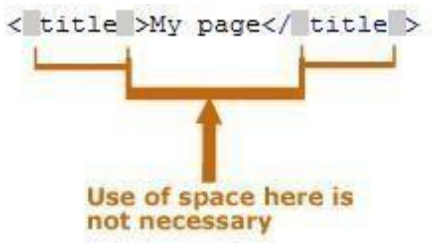
As we try to help yourself to make your code more readable, we may start using space where it is not needed. Remember browsers ignore any extra space you may have in your code. That does not mean you should use extra space when it is not needed, however.

There are two areas where the use of space should be avoided:

1. space between the tag name and brackets
2. space between the tag name and the text it affects

1. **space between the tag name and brackets**:

There should be no space between a tag name and its angle brackets. (This page explains when to use space inside the brackets when you are using attributes.) As an example, avoid:



Avoiding space between tag brackets

There is no reason to use space between a tag name and the angle brackets. So the title tag above should be rewritten as:

<title>My page</title>

2. **Space between the tag name and the text it affects:**

Space should also be avoided when you use a tag that affects some text (or some other web object):

The code shown above should instead be rewritten as:

This is a paragraph text.

Creating a paragraph

- In HTML, creating a paragraph is simple as entering text.
- To create a paragraph, move the cursor to the location where you want the paragraph in the HTML editor you are using. Next, type the tag to begin the paragraph. (Yes, if you guessed, the "p" inside the tag stands for paragraph.) Once you are done typing or pasting text for the paragraph, close the tag with .
- Closing the tag instructs the browser that the paragraph has ended.
- As an example, we will create the following paragraph with the tag:

To create the above paragraph, you would write the following code:

```
<html>
<title>How To Create Paragraph</title>
<body>
<p>To learn HTML and other web programming languages, come to ScriptingMaster.com regularly. The web site is updated daily with new content. The web site is succeeding because visitors like you keep coming back and find the site simple to use, easy to understand, and resourceful.</p>
</body>
</html>
```

The output of the code shown above would look like:

Showing how a paragaph may look in a browser

Text alignment with the align attribute

- The normal text alignment depends on how the text is read in the browser's default language. If the text is read, for instance, from left to right, the normal alignment is *left*. On the other hand, if the text is read from right to left, the normal alignment is *right*.

If we want to change the normal alignment of the text, you can use the *align*attribute. With the attribute, we could use one of these values:

1. left
2. right
3. center
4. justify

1. Left alignment:

To align text to the left margin, set the *align* attribute to *left*:

<p **align="left"**>This paragraph uses the *left* alignment. If a language is read from left to right and the browser is set to that language, the default alignment is left. When the text is aligned automatically to the left, use of the align attribute with the value *left* is not necessary.</p>

Output of the above code:

This paragraph uses the *left* alignment. If a language is read from left to right and the browser is set to that language, the default alignment is left. When the text is aligned automatically to the left, use of the align attribute with the value *left* is not necessary.

2. Right alignment:

To align text the right margin, set the *align* attribute to *right*:

<p **align="right"**>This paragraph uses the *right* alignment. If a browser's default language is read from right to left, the default alignment is right. When the text is aligned automatically to the right, use of the align attribute with the value *right* is not necessary.</p>

Output of the above code:

This paragraph uses the *right* alignment. If a browser's default language is read from right to left, the default alignment is right. When the text is aligned automatically to the right, use of the align attribute with the value *right* is not necessary.

3. Center alignment:

To change the alignment of the text to the center of the page, set the *align* attribute to *center*:

<p **align="center"**>This paragraph uses the *center* alignment. Aligning text center is easy: simply set the align attribute to center or use the tag.</p>

Output of the above code:

This paragraph uses the *center* alignment. Aligning text center is easy: simply set the align attribute to center or use the tag.

4. Justified alignment:

When you want to continue the text to the right and left margins, set the *align* attribute to *justify*:

<p **align="justify"**>This paragraph uses the *justify* alignment. The paragraph is justified: the text continues to both margins.</p>

Output of the above code:

This paragraph uses the *justify* alignment. The paragraph is justified: the text continues to both margins

Character tags

1. **logical**
2. **physical**

- A **tag** that is applied to an individual character is known as a **character tag**.
- A **character tag** can be grouped into two categories: **physical** and **logical**. (Note: *physical styles* are associated with physical character tags; *logical styles* are associated with logical character tags.)

1. Physical styles:

A *physical character tag* controls how the characters are formatted. For instance, you might display some characters as bold or italic.

Listing 1 displays some common physical character tags.

Listing 1 common physical character tags		
Tag	**Description**	**Renders as**
	bold	displays text as bold
<big>	big	displays text in a big font
	italics	displays text as italic
<s> or <strike> *	strike	displays text with a line through it
<small>	small	displays text in a small font
<sub>	subscript	displays the text as subscript — text that displays below the baseline of the text
<sup>	superscript	displays the text as superscript — text that has baseline above the baseline of the rest of the text
<tt>	teletype	displays the text with fixed-width font
<u>	underline	underlines the text
* Both <s> and <strike> tags are deprecated in HTML 4.0 so instead use the <del> tag.		

Listing 2 displays some examples of the common physical character tags.

Listing 2 examples of common physical character tags	
HTML code	**Output**
This is bold	This is **bold**
This is <big>big font</big>	This is big font
This is italic	This is *italic*
Was <s>$50</s>; now $40	Was ~~$50~~; now $40
This is small	This is small
H₂O	H_2O
May 5th 2005	May 5^{th} 2005
<tt>fixed-width font</tt>	fixed-width font
This is <u>underlined</u>	This is underlined

2. Logical styles:

A *logical character tag* describes how the text is being used, not necessarily how it is formatted. Listing 3 displays common examples logical character tags.

Listing 3 common logical character tags		
Tag	**Description**	**Renders as**
<cite>	citation	emphasizes the text in italics.
<code>	code sample	Displays some characters as code usually in Courier font (i.e., fixed-width font)
<del>	deleted text	displays text with a line through it; renders differently in Netscape and Internet Explorer
<dfn>	definition	italics; renders differently in Netscape and Internet Explorer
	emphasis	emphasizes the text in some way usually as italic.
<ins>	inserted text	underlined; renders differently in Netscape and Internet Explorer
<kbd>	code sample	fixed-width font
<samp>	code sample	fixed-width font

	strong	Text is emphasized more strongly than usually as bold.
<var>	program variable	italics

Listing 4 provides examples of how logical tags may display on a browser.

Listing 4 examples of logical tags	
HTML code	**Output**
This is used for a short <cite>quote</cite>.	This is used for a short *quote*.
<code>y = m * x + b</code>	y = m * x + b
~~Deleted~~ text	~~Deleted~~ text
<dfn>definition</dfn> text	*definition* text
This is emphasized .	This is *emphasized* .
<ins>inserted</ins> text	inserted text
<kbd>code</kbd> sample	code sample
<samp>code</samp> sample	code sample
This is strong.	This is **strong**.
<var>program</var> variable	*program* variable

Note that both logical and physical tags are two-sided tags (requiring an opening and closing tag). Also, the character tags can also be nested to format some text as

this is bold and italicized

which will result in :

this is bold and italicized

When you nest tags, make sure to closed them properly. Although the output of the following code is same, the tags are not closed properly:

this is bold and italicized

In summary, use the **characters tags** to either format text with physical tags or to describe some text with predefined logical tags. Also, remember to properly close nested tags.

Creating headings

1. There are six headings in HTML.
2. Each heading can be created with an HTML header tag, one header tag for each level of heading.
3. Header tags are range from <h1> to <h6>, where <h1> is the largest and most prominent and <h6> is the smallest.

To use a header tag, use the following syntax:

<hn>Some text</hn>

where n is a number between 1 and 6.

Figure 1 shows how to use all of the six headings as an HTML code on the left and the output from a browser is shown on the right.

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

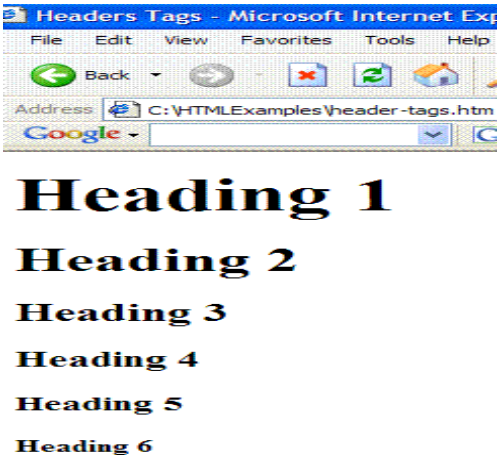


Figure 1 Six Headers Styles Displayed in Internet Explorer

HTML lists

1. **ordered,**
2. **unordered,**
3. **definition**

Lists are very important to any document as they allow you to make your key points stand out from the rest of the text. HTML supports three lists:

- unordered (bulleted)
- ordered (numbered)
- definition

1. Unordered (bulleted) list:

The most commonly used list is the *unordered list*.
An unordered list can be used whenever the order of items you want to list is unimportant.
HTML offers three different default characters to use with an unordered list:

1.a bullet (•),
2.a circle (◦),
3.a square (▪).

The following shows examples of an unordered list:

Unordered list — type bullet (default)

- List item 1
- List item 2
- List item 3

Unordered list — type bullet (default)

Unordered list — type circle

- List item 1
- List item 2
- List item 3

Unordered list — type circle

Unordered list — type square

- List item 1
- List item 2
- List item 3

Unordered list — type square

2. Ordered (numbered) list:

- The other popular kind of list is the ordered list. This type of list can be used when the order of items to list is important. It could be that you want to list steps to how to cook a recipe; for this kind of list you could use an ordered list because each step can be emphasized numerically.
- An ordered list can be created with different styles:
    a. Arabic numbers,
    b. lowercase **or**
    c. uppercase letters, **or**
    d. lowercase or uppercase Roman numerals.

The following shows examples for each of **these styles**:

Ordered list style: Arabic numbers (default)

1. List item 1
2. List item 2
3. List item 3

Ordered list style: Arabic numbers (default)

Ordered list style: lowercase letters

a. List item 1
b. List item 2
c. List item 3

Ordered list style: lowercase letters

Ordered list style: uppercase letters

A. List item 1
B. List item 2
C. List item 3

Ordered list style: uppercase letters

Ordered list style: lowercase Roman numerals

i. List item 1
ii. List item 2
iii. List item 3

Ordered list style: lowercase Roman numerals

Ordered list style: uppercase Roman numerals

I. List item 1
II. List item 2
III. List item 3

Ordered list style: uppercase Roman numerals

3. Definition list:

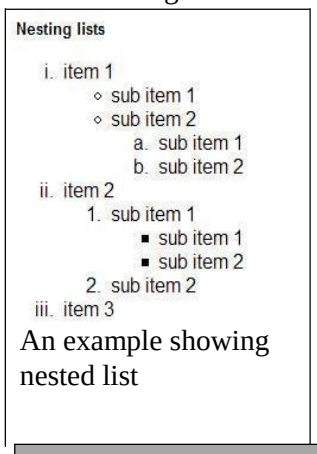
- The *definition list* is used far less frequently than the other two kinds of lists mentioned above.
- A definition list, as the name implies, is used for listing definitions:

HTML
HTML stands for Hypertext Markup Language. It is a popular markup language used today for creating web pages.
Internet
A network of computers operating world-wide using common set of communication protocols.

Nested lists:

- Sometimes, a list is long and may require subordinate lists.
- Mastering multiple lists within a single list is easy once you learn how to create a single list.

The following shows an example of a nested list:



An example showing nested list

Unordered list

- An ***unordered list*** is used when list items do not have any particular order.
- An unordered list is created with the **
** tag. tag also is a two sided tag and closed with tag.
- Like an ordered-list, an item for unordered list is listed with the tag

Suppose now we want to list the five classes in which Tom has received an "A" on mid-term for each class as:

Biology
Calculus
Statistics
History
English

To create this list, we will list start with the <ul> tag. Next, we will use the <li> tag to list the first item and to end the first item. Similarly, we will create the rest of the list items with the <li> and tag.
The following listing shows the code for creating the entire unordered list:

```
<ul>
<li>Biology</li>
<li>Calculus</li>
<li>Statistics</li>
<li>History</li>
<li>English</li>
</ul>
```

Change bullet style for an unordered list:

we can change the numbering style for an ordered list, we can change the default bullet style for an unordered list with the *type* attribute.
The three possible values for an ordered list include:

1. ***disc*** - the default type, represented by a solid circle.
2. ***square*** - a solid square.
3. ***circle*** - a ring

The following shows examples for each of the *type* attribute value for an ordered list:

HTML code	**Output**
<ul **type="disc"**> - Ordered list - Unordered list - Definition list	Ordered list Unordered list Definition list

<ul **type="square"**> - Ordered list - Unordered list - Definition list 	Ordered list Unordered list Definition list
<ul **type="circle"**> - Ordered list - Unordered list - Definition list 	Ordered list Unordered list Definition list

You can also change the bullet type for individual list items by setting the type attribute with the <li> tag:

```
<ul>
<li type="disc">Ordered list</li>
<li type="square">Unordered list</li>
<li type="circle">Definition list</li>
</ul>
```

That will produce the following:

Ordered list
Unordered list
Definition list

Ordered list

- An ordered list is a numbered list.
- An ordered list can be used whenever a list requires sequence.
- Let's assume we want to create an ordered list that displays how Tom likes six colors, the most important color is listed first and the next most important color is listed second, etc.
- To create such list, we would use an ordered list as:

```
<ol>
<li>Gold</li>
<li>Blue</li>
<li>Silver</li>
<li>White</li>
<li>Red</li>
<li>Yellow</li>
</ol>
```

The <ol> tag starts an ordered list. The <ol> tag is a two-sided tag, which means that it also requires a closing tag . The tag instructs the browser that the ordered list has ended. The <li> tag is used to list each item in the list. The <li> tag is also a two-sided tag. Thus each item that needs to be placed in a list, should be placed inside the <li> and tag.

In the example above, the first item for our ordered list is "Gold" because it is placed inside the tag and it is the first tag after the tag. The second item for our list is "Blue" because it is placed inside the second tag. By the same logic, you can deduce that "Silver" is third most important, and so on. Figure 1 shows the output of this example.	Gold Blue Silver White Red Yellow Figure 1

Change numbering style for an ordered list:

- The <ol> tag includes the *type* attribute that can be used to change numbering style for anordered list.
- This attribute can be set to one of the following choices:

1. ***Arabic numbers*** (1) - to create list using *Arabic numbers*. This is the default type for an ordered list.
2. ***uppercase letters*** (A) - to create list using *uppercase alphabet letters*
3. ***lowercase letters*** (a) - to create list using *lowercase alphabet letters*
4. ***uppercase Roman numerals*** (I) - to create list using *uppercase Roman Numerals*
5. ***lowercase Roman numerals*** (i) - to create list using *lowercase Roman Numerals*

The following shows examples with the type attribute set to different possible values:

HTML code	Output
<ol **type="1"**> - HTML - ASP - JavaScript 	HTML ASP JavaScript
<ol **type="A"**> - HTML - ASP - JavaScript 	HTML ASP JavaScript
<ol **type="a"**> - HTML - ASP - JavaScript 	HTML ASP JavaScript
<ol **type="I"**> - HTML - ASP - JavaScript 	HTML ASP JavaScript
<ol **type="i"**> - HTML - ASP - JavaScript 	HTML ASP JavaScript

Changing the list value for an ordered list:

- With the *start* attribute you can set the value of the first element in the list.
  For example, if you wanted to start your list with the number 5, set the *start* attribute to 5 and the *type* attribute to 1, if necessary.
- we can also use the *start* attribute to change the list value of the first element for alphabetical or Roman numerical values.
  If, for example, you wanted to start your ordered list with the letter C, set the *type* attribute to A and the *start* attribute to 3. See some example below:

HTML code	Output	Comments
<ol type="A" **start="3"**> - HTML - ASP - JavaScript 	HTML ASP JavaScript	Starts the list at letter C because the starting value is set to 3.
<ol type="i" **start="5"**> - HTML - ASP - JavaScript 	HTML ASP JavaScript	Starts the list with Roman numerical v because the *start* attribute is set to 5.

While the *start* attribute changes the number for the first item in the list, the *value* attribute can be used to change the numbering for all other list items. For instance,

```
<ol>
<li value="4">HTML</li>
<li value="8">ASP</li>
<li value="10">JavaScript</li>
</ol>
```

will produce:

```
HTML
ASP
JavaScript
```

Definition list

1. A definition list is used for displaying terms and each followed by a definition list.
2. A definition list starts with the <dl> tag.
3. Each term in the definition list is listed with the <dt> tag.
4. Finally, the definition for the term is listed with the <dd> tag.
5. Example that will produce the following output:

Network
A number of computers or other devices connected to share information and resources.
Multitasking
The ability of CPU to run two or more programs independently at the same time.
File Management
Organization and tracking of files.

To create **this list**, we will first use the **<dl> tag** to start the definition list.
Next, to list the term we will use the <dt> tag.
The first term we want to define is "Network" thus it will contained in the <dt> tag as <dt>Network</dt>.
To list the definition for the term, we will use the <dd> tag, as shown below:

```
<dl>
<dt>Network</dt>
<dd>A number of computers or other devices connected to share information and resources.</dd>
<dt>Multitasking</dt>
<dd>The ability of CPU to run two or more programs independently at the same time.</dd>
<dt>File Management</dt>
<dd>Organization and tracking of files.</dd>
</dl>
```

Nested list:

1. Lists can be nested, meaning one list can be placed inside of another.

2. A nested list, for example, can be used to create an outline.

The following shows an example of a nested list:

HTML
Meta tags
anchor tag
ASP
Arithmetic operators
Relational oeprators
JavaScript

In this nested list, we have total of three lists. The main list starts with "HTML". The second list, the first nested list, starts with "Meta tags". The third list, the second nested list, starts with "Arithmetic operators". The main list ends with "JavaScript."

To create that nested list, start by creating each seperate list. First, create the main list:

```
<ul>
<li>HTML</li>
<li>ASP</li>
<li>JavaScript</li>
</ul>
```

Now, create the second list:

```
<ol>
<li>Meta tags</li>
<li>anchor tag</li>
</ol>
```

Finally, create the third list:

```
<ul>
<li>Arithmetic operators</li>
<li>Relational oeprators</li>
</ul>
```

Now, we have created all three lists. Next, place each of the nested list in the main list item. For example, place the second list in the first list item, HTML, of the main list:

```
<ul>
<li>HTML
<ol>
<li>Meta tags</li>
<li>anchor tag</li>
</ol>
</li>
<li>ASP</li>
<li>JavaScript</li>
</ul>
```

That will result in:

HTML
Meta tags
anchor tag
ASP

JavaScript

Finally, place the second nested list in second item, ASP, of the main list:

```
<ul>
<li>HTML
<ol>
```

```
<li>Meta tags</li>
<li>anchor tag</li>
</ol>
</li>
<li>ASP
<ul>
<li>Arithmetic operators</li>
<li>Relational oeprators</li>
</ul>
</li>
<li>JavaScript</li>
</ul>
```

Line breaks

1. To add a line break in a web page, you can use the
 tag.
2. Typing a
 tag in your HTML code is similar to pressing the ENTER or RETURN key in a word processor.
3. The effect of
 tag is that the browser stops printing text on that line and drops down to thenext line on that page.
4. The
 tag does not require a closing tag.

The following shows an example:

**Line breaks
 in HTML** :

instructs the browser to start a new line after the word "breaks" because after this word the tag
 is placed. Thus the output of the above code is:

Line breaks
in HTML

To add multiple line breaks in a row, simply use the
 tag for each line you want to stop and start. For instance,

Line breaks

 in HTML

inserts three line breaks after the word "breaks".

The following shows the output of the above code:

Line breaksin HTML

Preformat text

1. With the <pre> tag, the text renders in the browser exactly as you type it.
2. The <pre> tag is short for preformat. So if you press the ENTER or RETURN key several timesto create line breaks within a <pre> tag, you can expect to see all those line breaks in a browser.

The following shows the <pre> tag in use:

```
<pre>
This    is an

example of preformatted text.  The

text appears in the browsers exactly as

it is   typed.
</pre>
```

<pre> tag in use to preformat text

That prints:

This is anexample of preformatted text. Thetext appears in the browsers exactly asit is typed.

This example creates a tubular data:

```
<pre>
A   |   B
C   |   DD
</pre>
```

Tabular data with pre tag

```
A   |   B
C   |   DD
```

Quotation blocks:

1. Quoting a large body of text and making it stand out from the rest of text can be accomplished with the <blockquote> tag.
2. The tag indents the quotation block on both the left and right, and also adds a blank line above and below. The amount of indentation used on both sides may vary from browser.
3. The following code shows the <blockquote> tag in use:

```
<p>This is some text before the quotation.</p>
    <blockquote>This is a long blockquote created with the <blockquote> tag.</blockquote>
```

4. The following shows the **output** of the above code:

This is some text before the quotation.

This is a long blockquote created with the <blockquote> tag.

Controlling the font size with the tag:

- Occasionally, you may change the appearance of text for extra emphasis.

For example, you may make the text bold, a heading, or just increase its size than the surronding text. This page explains how to use the tag to control the font size.

- To control font size, use a font size value of 1 - 7, where font 1 is the smallest and 7 is the largest.
- Listing 1 shows the HTML code and the corresponding 7 different font sizes.

Listing 1 HTML code and different font sizes

HTML code	Output
size 1	size 1
size 2	size 2
size 3	size 3
size 4	size 4
size 5	size 5
size 6	size 6
size 7	size 7

Notice to control the size of the font, we used the *size*attribute. An attribute adds an extra instruction to the tag with the specified value. Thus an attribute consists of the name of the attribute and a value. For instance, in the first example in listing 1, size is an attribute and the value is 1.

Changing the font color with the tag:

1. To control the color of a font, we use the color attribute with a color value.
2. For the value, you can specify either the name of the color or a RGB value. A name of the color could be like "red", "green", "blue", etc.
3. To select colors with RGB values, set the color attribute to "RRGGBB", in which the first two letters RR stands for red.
4. Each of the 2 R's can be substituted with a number value 0 - 9 or the letters A - F. "0" represents absence of the color red, while the letter "F" represents the most amount of color red.

5. The next two letters GG stands for green and can be substituted with the same values as for the color red.
6. The last two letters BB stands for blue and can be substituted with the same values as for the previous two colors.

Listing 2 examples shows examples of changing color of a font with using a name or RGB value of color.

Listing 2 Changing the font color		
Color Name	**Color value**	**Output**
Red	Red	Red
Green	Green	Green
Blue	Blue	Blue

Changing the font face with the tag:

The last property of the font that we can control with a tag is the font face.
we can use the face attribute to specify the type of font that should be used to display some text.
For instance,
this text is displayed using the verdana font face.
outputs
this text is displayed using the verdana font face.
Because not all browsers support all fonts, you can supply more than one similar font face value but up to three by separating each with a comma as:
Some text
When you specify more than one font face value, the browser checks if the first font face value is available to the browser; if it is, then, the browser uses the first font face value. If the first font face value is not available, then, the browser tries the second one and so forth until it tries all three font face values. If all of the three font face values are not available, the browser uses the default browser font face value.

Combining tag attributes:

When we have to control size, color or face of a font at the same time, we could use separate font tags as:

Some text

Specifying individual attribute with separate tags is insufficient as the browser has to process these extra tags. Additionally, it is confusing and long. Instead of individually specifying the size, color, or face of a font, you can combine all three into one tag:
Some text.
That outputs:
Some text

Horizontal rules:

To break up an HTML document into separate sections, we can insert a horizontal line (rule).
A horizontal rule is inserted by the <hr> tag.
The <hr> tag is one-sided, meaning it does not require a closing tag.
example:
Inserting horizontal lines
<hr>
As several sections are added to a web page, they can be separated into visually distinct regions by a horizontal rule.
Inserting Horizontal lines

As several sections are added to a web page, they can be separated into visually distinct regions by a horizontal rule.

<hr> attributes:

- The <hr> tag has several attributes that control the size and the appearance of the horizontal rule.
- For instance, the **size** attribute controls the rule's thickness (height) in pixels.
- The **width** attribute controls the rule's width in percentages or pixels.
- The **align** attribute sets the alignment of the rule to left, right, justify, or center. (Note the default rule's alignment is center.) Finally, the **noshade** attribute renders the rule without a surrounding shadow.

The following HTML code for each of these attributes in use:

Horizontal rule of size 12:	**<hr size="12" />**
Horizontal rule of size 12 and width 20%:	**<hr size="12" width="60%" />**
Horizontal rule of size 12, width 300 pixels, and aligned left:	**<hr size="12" width="60%" align="left" />**
Horizontal rule of size 12, width 300 pixels, and aligned right:	**<hr size="12" width="60%" align="right" />**
Horizontal rule of size 12, width 300 pixels, and aligned justify:	**<hr size="12" width="60%" align="justify" />**
Horizontal rule of size 12, width 300 pixels, and no shading:	**<hr size="12" width="60%" noshade="noshade" />**

The following shows the output of the above code:

Horizontal rule of size 12:

Horizontal rule of size 12 and width 20%:

Horizontal rule of size 12, width 300 pixels, and aligned left:

Horizontal rule of size 12, width 300 pixels, and aligned right:

Horizontal rule of size 12, width 300 pixels, and aligned justify:

Horizontal rule of size 12, width 300 pixels, and no shading:

Divisions (with the tag):

1. To structure HTML documents into divisions or sections, the tag is used.
2. The tag specifies a logical block without predefined meaning or rendering infomration.
3. Orginally, the tag was mostly used to align sections of content in a document with the *align* attribute.

The followings shows an example:

```
<div align="right">
<h4>Division heading</h4>
<p>Paragraph 1. This text is right-aligned. The div affects paragraph and other block elements.</p>
<p>Paragraph 2. This text is also right-aligned.</p>
</div>
```

The following shows the output of the above code:

Division heading

Paragraph 1. This text is right-aligned. The div affects paragraph and other block elements.

Paragraph 2. This text is also right-aligned.

Although it is not apparent from the above the example, the tag produces a line break after the tag is closed ( ). The is a block element.

Styling with tag:

The tag is today commonly used to apply style sheet fomatting properties to a desired part of a document. n example:

```
<div style="color:#CC3333; text-align:right;">
<p>Paragraph 1 inside div tag 1.</p>
<p>Paragraph 2 inside div tag 1.</p>
</div>
<div style="color:#00CC66">
<p>Paragraph 1 inside div tag 2.</p>
<p>Paragraph 2 inside div tag 2.</p>
</div>
```

That produces

Paragraph 1 inside div tag 1.
Paragraph 2 inside div tag 1.
Paragraph 1 inside div tag 2.
Paragraph 2 inside div tag 2.

In the first tag we change the color of the text and right-align the text. In the second tag we only change the color of the text and keep the default text alignment (left).

Specifying Metadata:

Metadata helps search engines track and index web pages.

There are number of *meta tags* that needs to be included in any web page for it to be indexed by search engines.

1. Meta tag attributes:

Meta tags are used to provide information about a web page.

The meta tags are optional and easy to use.

There are three main attributes of meta tags:

- Name
- content,
- http-equiv.

1. The ***name*** attribute specifies the type of information.
2. The ***content***attribute includes the meta-information. (Access this page to learn how to use thesetwo attributes to optimize your web page for search engines.)
3. Lastly, the ***http-euiv*** attribute specifies a particular header type.

Name attributes:

1. There are over a dozen of *name* attributes.
2. The purpose of using *name* attributes is to provide information to the search engines.

The following table summarizes the most common name attributes:

Attribute name	Explanation
description	Description of the web page.
keywords	Used to list keywords that describe the content of the web page.
creator or author	The organization or person who responsible for creating the webpage.
date	The date of publication in yyyy-mm-dd format.

identifier	A unique number identifying a web page.
language	Language of the page. Use a two-character language code.
rights	Used for adding copyright statement.

http-euiv attributes:

- The http-euiv attributes are equivalent to HTTP headers and control the action of a browser on a requested page. When displaying, the browser uses the instructions specified by the http-euiv attributes.
- The instructions may contain information about when the content of the page will expire or when the page should be refreshed.

Hyperlinks:

1. To make other documents easily accessible, a **hyperlink** is used.
2. A hyperlink is a part of the web page that user clicks on to view the destination document.
3. Thus a *link* basically instructs a browser to load a new URL (page).
4. A document that contains hyperlinks is referred to as **hypertext document**.

There are four main types of hyperlinks:

- relative — linking local pages using relative file names
- absolute — linking to local or extrenal pages using absolute file names
- reference — linking to specific locations or sections of a page
- email — linking to an email address

A link is created with an **<a>** tag. The "a" in <a> stands for anchor thus it is called an "anchor tag" or simply **link tag**. An anchor tag is a two-sided tag and uses a *href*attribute to specify the destination of the link.

The href attribute stands for Hypertext Reference.

Customizing links:

- Changing link colors
- Setting link title
- Setting tab order of links
- Adding keyboard shortcuts to links

Understanding URLs:

To be able to create links, we need to understand URLs.

A URL stands for uniform resource locator (URL) and it has several parts:

- Protocol
- Domain name
- Directory (folder)
- Filename and file extension

The following example shows these parts in a URL:

Parts of a URL:

1. The **first part**, as the graphic shows above, of a URL is the name of the protocol: *http*.
   The *http* protocol is a set of rules that enables web browsing. Specifically, we want to use the ***http*** protocol to access http://www.scriptingmaster.com/html/creating-links.asp.
2. The **second part** of the URL is the domain name. The ***domain name*** simply is a text name that corresponds to the IP address of the server that serves that web site. Because a text name is easier to

remember and recognize, a domain name is used rather than an IP address. (The IP address that corresponds to scriptingmaster.com is 64.40.96.247 thus we could have used http://64.40.96.247/html/creating-links.asp. However, again a domain name is much easier to remember and recognize than using IP address.) You may have seen domains that end with .edu (for education), .org (for organization), .info (for information), .mobi (for mobile), etc. When creating links, use a valid domain name (including .com, edu, or .info) and avoid use of IP addresses.

3. The **third part** of the URL specifies the directory or the folder name. The directory simply is the actual location of the file we want to access/link: creating-links.asp. To avoid broken links, ensure the directory exists and contains the file that you want to link to.
4. The **last part** of the URL is the file name: creating-links.asp. When creating links, make sure thefile exists and that you use the appropriate file extension (such as .htm, .html, .asp, or .php).

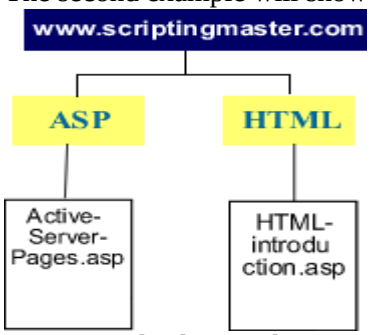
1. Relative link:

- When creating links, we need to reference the files with each file's location, known as file's **path**.
- HTML supports two kinds of file paths: relative and absolute.
- A relative path refers to a file's location that is relative to the folder in which the file is in.
- A relative link is the easiest kind of link to create since you only need to reference thefile name (because the file is in the same directory).

2. Absolute link:

- An absolute link shows the entire destination address in the link tag.
- This exact destination address is referred to as an absolute path. To link to files which are outside of the current folder, an absolute path needs to be used. Thus an absolute link is used for linking to files that are outside of the current folder; a relative link is used for linking to files that are within the same folder.
- An absolute link is also used for linking to other websites.

The second example will show you how to create a link to an external website.



In this example, we have two folders: ASP and HTML. If we link Active-Server-Pages.asp file to the HTML-introduction.asp file, we would need to reference the folder name because the HTML-introduction.asp file is outside of the ASP folder.

Figure 2 Absolute Link

Table 1	
URL	http://www.scriptingmaster.com/site-map.htm
Code	Site Map
Description	Loads the site map of ScriptingMaster.com's website. The file site-map.htm is listed after the domain name and forward slash.
Output	Site Map
URL	http://www.scriptingmaster.com/scripting-tools/scripting-tools.asp
Code	Scripting Tools
Description	In this example, we use the folder name "scripting-tools" after the

	domain name. Then, we use the forward slash and list the specific web page: scripting-tools.asp.
Output	Scripting Tools

3. **Reference link:**

A reference link points to a specific part of a web page. How is this different from other links? Both relative links and absolute links are used for linking to other webpages. But what if you wanted to link to a special part of a webpage? What do you do? You use a reference link.

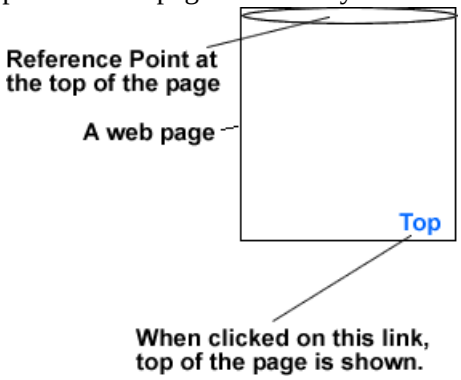


Figure 3 - Reference Point Link

Creating an email link:

When a person clicks on an email link, the person can send email to the specified email address in the HREF property. Note simply clicking on an email link does not send the email; it rather copies the recipient's email address into the user's default email editor program, if any, for user's convenience. To create an email link, we use the <a> tag with the **HREF** property set to the desired email address. For instance,

info@scriptingmaster.com

creates an email link with the destination email address as info@scriptingmaster.com. Notice because this is an email address, we specified "*mailto*:" before the email address. The link text for this link is simply the same as the email address. Output: info@scriptingmaster.com.

Changing link colors

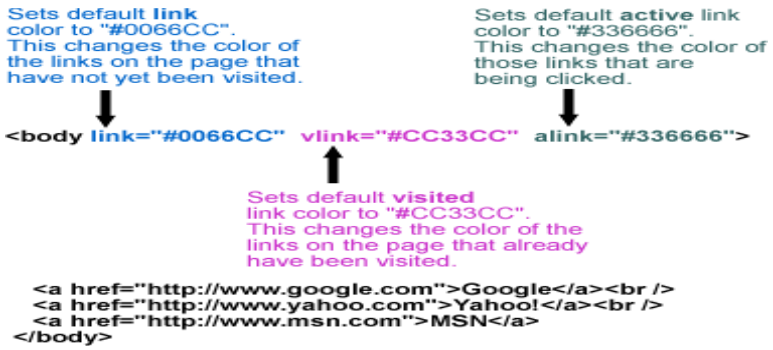
As you may have noticed while browsing websites, the default link color for browsers is blue, the default visited link color is purple, and the default active link color is red. What do you do if these default colors do not work for you because of the regular color of other text on your webpage and/or because of the background of your webpage? Well, you can change the default link colors. To change these default link colors, you can use three attributes in the body tag or define link colors in a style sheet.

Changing link colors with link attributes of the <body> tag

The three link color attributes of a webpage are:

- **link** — this changes the *normal* link color. In other words, this attribute changes the default *blue* color of the links to some other color of your choosing.
- **vlink** — this attribute changes the *visited* link color. With this attribute, you can change the default visited *purple* link color to some other color of your choice.
- **alink** — This attribute changes the *active* link color. An active link color simply meansthe displayed link color when the mouse button is pressed over a link.

These attributes should be added to the <body> tag with a choice color for each attribute:



Output:

Google Yahoo! MSN Normal links color (#0066CC)	Google Yahoo! MSN After a link is clicked, link color changes to the visited link color (#CC33CC).	Google Yahoo! MSN Showing the active link color (#336666) while the mouse button is pressed over the first link.

Changing link colors with style sheet declaration:

Because the three attributes discussed above (link, vlink, and alink) are deprecated in favor of style sheets, you should change the link color using style sheets. This, however, does not mean your HTML document would be invalid if you use those attributes in the body tag to change link colors; rather, consider using the style sheet declaration to change link colors in case if in the future browsers stop supporting these deprecated attributes.

The following shows an example of style sheet declaration to change link colors:

```
<style>
a:link {color:#0066CC;} /* sets normal link color */
a:visited {color:#CC33CC;} /* sets visited link color */
a:active {color:#336666;} /* sets active link color */
</style>
```

Setting link title:

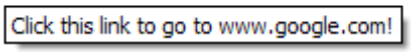
To help users navigate a website, you can add the optional **title**attribute to the link tag: <a>. With the *title* attribute, you can provide additional information to the user about the link. Although the *title* attribute causes different browsers to do different things, generally speaking the title text appears as a "tool tip" when the visitor moves the mouse over the link.

In the following example, we add some additional information about the link using the *title* attribute:

Visit www.google.com

The following shows the output:

Visit www.google.com

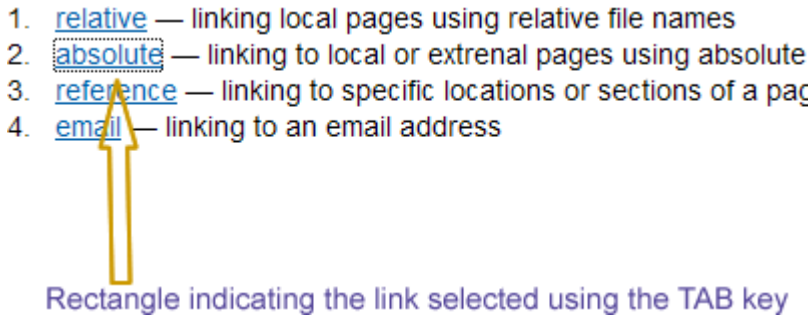


Output showing the title text when the mouse is moved over the link

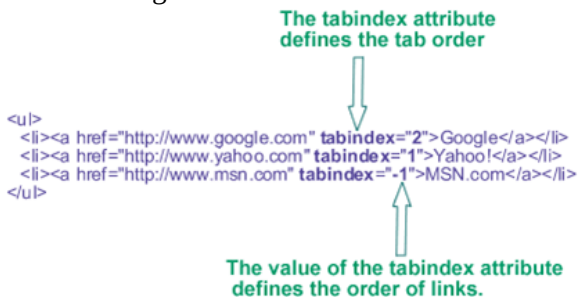
Setting tab order for links:

To help users who cannot use mouse, the *tabindex* attribute can be added to the anchor tag to use specific tab order of links. When you customize the order of links, browsers generally highlight each link, from top to bottom of the web page, each time the tab key is pressed. Once a link is selected and highlighted, the user

can press ENTER or RETURN key to visit the selected page. The following shows a link selected with a TAB key:



The following code shows how to customize the tab order with tabindex:



In the top example, we set tab order for our three links as follows:
- First, Yahoo! because it has the tabindex value set to 1
- Second, Google because it has the tabindex value set to 2
- By assigning a negative value to tabindex, we are excluding MSN link from the tab order

The following shows the output of the code; notice if you use the tab key, the link Yahoo! will be highlighted first, Google second, and MSN link is highlighted after all other links above this link, from top to bottom of this page, are highlighted:

Google
Yahoo!
MSN.com

Adding keyboard shortcuts to links:

- With the accesskey attribute, you can assign keyboard shortcuts to links.
- The idea behind creating keyboard shortcuts is to simply allow the user to quickly visit a pageby using the access keys value corresponding to the link.
- In Windows, use ALT-X (where X represents a numerical value assigned with accesskeyattribute) keys from the keyboard to open the desiredlink.
- When defining keyboard shortcuts for links, make sure you include the keyboard shortcuts next to each link. If, however, you do not display the keyboard shortcuts, user won't know you have defined keyboard shortcuts. Thus users cannot use those shortcuts for links!

The following shows how to use the accsskey to define keyboard shortcuts to links:

Start learning ASP (ALT or COMMAND 1)
Start learning HTML (ALT or COMMAND 2)
Learn JavaScript (ALT or COMMAND 3)

For the first link above, we assign the accesskey 1 to designate the shortcut ALT 1 in Windows (and COMMAND 1 under MAC) to use this link. For the second link, we set the accesskey to 2 and thus to use this link ALT 2 can be used. Because accesskey attribute is set to 3 for the third link, ALT 3 can be used to activate this link. The following shows the output of the code:

Cascading Style Sheets (CSS)

1. Introduction:

- **Content and Style**
- **HTML Structure**
- **CSS Attachment to Pages**
- **CSS Main Facilities**
- **Browser Support**
- **CSS 1 Specification**

Content and Style:

Two aspects of any document are **content** and **style.**
The **content** gives the **information** to be presented
And the **style** defines how the information is presented. Most publishers have a House Style that is a consistent way of presenting information.
HTML's main role is to **define content**. What is needed is the ability for publishers to have control of style. In the context of the Web, the publisher can be the organisation that owns the Web site but it can also be the person viewing the information. It is only by having a clear separation between content and style that this can be achieved. So in this Primer we shall use HTML in most of the examples as a means of defining the **content** and CSS to define the **style**.
CSS is a **World Wide Web Consortium (W3C**) Recommendation. That means that all the W3C Members(which includes all the main browser manufacturers) have agreed to support it in their products. That does not necessarily mean immediately as each has its own production schedule for enhancements to its products but long term all should support all of the features. CSS first became a W3CRecommendation in 1996 (called CSS 1) and there was a significant update in May 1998 .**HTML** allows a Web page to be laid out and there is some guidance to browsers as to how to represent each element. Thus, **h1** should be larger than **h2** and **em** should be emphasised by some method but
that is as far as it goes.

The aim of Cascading Style Sheets (CSS) is to give the page developer much more control on how a page should be displayed by all browsers.

A **style sheet** is a set of **rules** that controls the formatting of HTML elements on one or more Web pages. Thus, the appearance of a Web page can be changed by changing the style sheet associated with it. There is no need to make detailed changes within the Web page to change how it looks.
advantages of using style sheets :

1. **accessibility**, different styling can be provided for different users dependent on their requirements.
2. Separating style and content is **good design** and will normally produce a better and more consistent web site. As one style sheet can be used for a whole web site, it normally means that the overall **size** of the web site is smaller and the downloads required for each page can be decreased by up to 40%.
   Allwoing browsers to make overall decisions on styling often means that the rendering time by the browser is also shorter.
3. A Web page may have its own style sheet that refines the information in the common style sheet. Readers may define their own style sheet indicating their preferences. Thus style sheets **cascade** and decisions need to be made as to which style sheet is in control when there is a conflict.

A style sheetruleconsists of **two parts**. A **selector** that defines which HTML elements are controlled by the rule and a **declaration** that says what the required effect is.
Thus a simple rule is:

```
h1 { color: blue }
```

This says that all **h1** elements in a page should be displayed in blue. The selector is **h1**. **color** is the **property** that is to be changed, **blue** is the **value** that the property is changed to and **color: blue** is the declaration.

HTML Structure

HTML elements in the body of the document fall into two main classes:

1. Block-level
2. Inline (sometimes called character-level)

Figure 1.1: Block level and Inline Elements

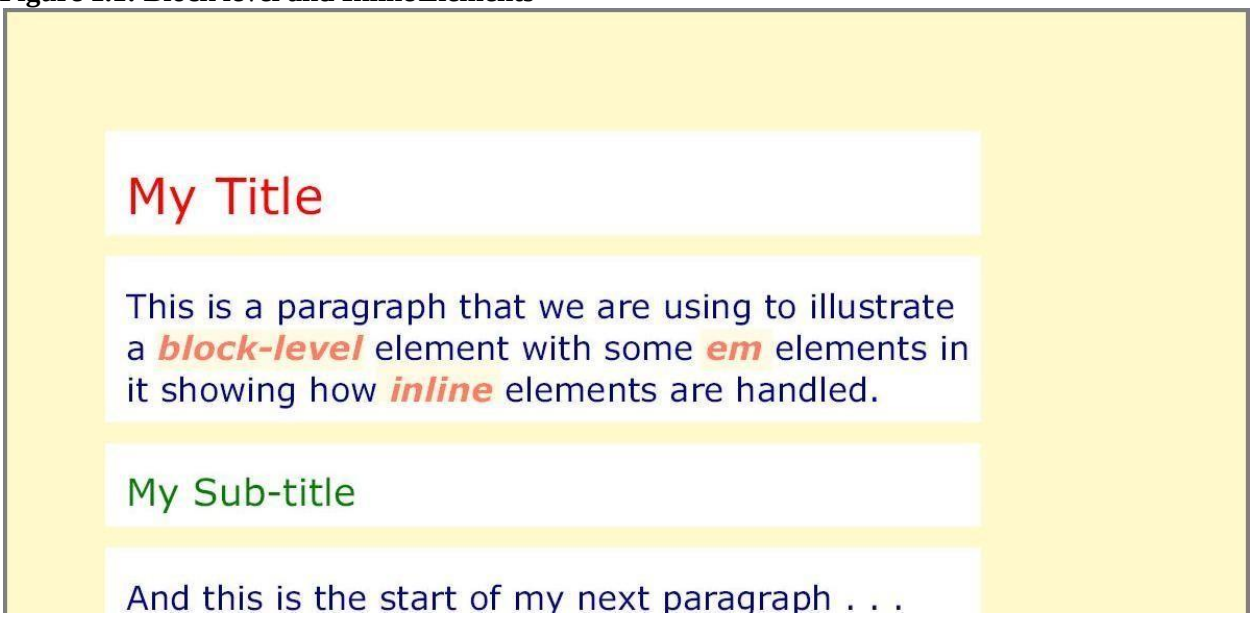


Figure 1.1: Block level and Inline Elements

Lists and tables are special types of elements that have their own unique styling.
When a **block-level element** is inserted into a document, it terminates the previous element and effectively starts a new line. Some examples are **<p>**and **<h1>**. Inline elements do not terminate the previous element and form part of the previous element; **em** is an example.
The **body** of an HTML page contains text marked-up using block-level elements (**h1, p, ul, div** etc). Each block-level element has an area which contains the content. For a **p** element, this is the area that contains the whole paragraph while for **h1** it is often an area consisting of a single line of text. CSS provides facilities for controlling where that area is placed relative to other areas.
The **inline** elements often inherit the style of the block-level element that they are part of but can be given special styling appropriate to the inline element.
For example:

```
<h1>My Title</h1>
<p>This is a paragraph that I am using to illustrate
a <em>block-level</em> element
with some <em>em</em> elements in it showing
how <em>inline</em> elements are handled.</p>
<h2>My Sub-title</h2>
<p>And this is the start of my next paragraph . . .</p>
```

Figure 1.1 shows the areas covered by the block level elements and the sub-areas covered by the inline elements. The inline elements have been given their own background colour, colour, font-weight (bold) and font style (italic) while inheriting the text size and font from the enclosing paragraph.

CSS Attachment to Pages:

There are **four ways** of linking a style sheet to an HTML page:

1. By defining a link from the HTML page to the style sheet (normally stored in a **xxx.css** file). This allows you to have a single style sheet that changes the appearance of many Webpages.

2. By specifying an HTML **style** element in the **head** of the page. This allows you to define the appearance of a single page.

3. By adding inline styles to specific elements in the HTML file by the **style** attribute. This allows you to change a single element or set of elements. This should only be used when absolutely necessary as it negates some of the advantages of having style sheets. It effectively over-rides the overall style for the page. Also, it is likely that it will be removed from CSS 3.

4. By importing a style sheet stored externally into the current style. The **style** element for thepage can consist of a set of imports plus some rules specific to the page.
Initially, while you are experimenting, the simplest method is to use the HTML **style** element to adda style sheet to an HTML page. The **style** element, consisting of a set of rules, is placed in the document **head**.When you have decided on your house style, it is likely that you will move to having the style sheet external to an individual page and either linked in or imported.

Main Facilities

The facilities provided by style sheets are much as you would expect:

1. Better control of fonts including colour
2. Better control of block-level layout (indents, margins, alignment etc)
3. Better control of inline layout, particularly with regard to diagrams and related text

Browser Support

To achieve these effects, the Web browser has to know what to do with style sheets. Really old browsers will not have this capability but the recent offerings from Microsoft, Opera and Netscape have good support now and are committed to full support in the future. HTML Editors are beginning to have Style Sheet additions and separate Style Sheet Editors are appearing. Visit the W3C CSS Web Page [1] for all the latest information.

Given the way CSS has been implemented, it is possible to design your web pages so that they still present the HTML information even when CSS support is not there. So there is little excuse for not adding style to your web pages now even if you expect your pages are to be viewed by really old browsers.

CSS Specification

This Primer will not tell you everything about CSS. If you get to the position where you really need to know precisely what happens in some subtle situation, you need to consult the formal specification that can be found on the World-Wide Web Consortium's web site [2] [3].

To make it as easy as possible to relate the specification to the Primer, the order of presentation here is similar to the order in the Specification.

2. CSS Rules:

Introduction

Syntax for CSS Properties

Introduction

Let us assume initially that we wish to define a style sheet by adding a **style** element to a page. The page will look something like:

```
<html>
<style type="text/css">
h1 {font: 12pt/14pt "Palatino"; font-weight: bold; color: red}
h2 {font: 10pt/12pt "Palatino"; font-weight: bold; color: blue}
p {font: 10pt/12pt "Times"; color: black}
```

```
</style>
<body>
...:
</body>
</html>
```

The **set of rules** are contained between **<style>**and **</style>**. Ignore the **type** attribute, for now. The reason for this will be come clearer later. For now let us concentrate on the rules. As stated in the introduction, each rule consists **of two parts**:
a **selector** that defines what HTML elements are controlled by the rule and a **declaration** that says what the required effect is. The selectors above define three rules that apply to all **h1** elements, all **h2** elements and all **p** elements respectively. We shall show how we can be both more selective (control a subset of **h1** elements) or control a set of elements (a block of different HTML elements) later. Initially let us look at the declarations. Each declaration consists of the name of a **propert**y and the value to be assigned to it. So the simplest style rule is:

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

To set several properties in a single rule requires the individual declarations to be separated by semicolons:

```
selector { property1: value1; property2: value2; property3: value3 }
```

Syntax for CSS Properties:

The general format for each sub-section is for
the title to give a descriptive name followed immediately by the syntax definition (in BNF style) for the property declaration.
For example:

```
font-weight: <value>
```

This says that the name of the property is **font-weight** and **<value>**is assigned to it. In the CSS specification, the possible values of **<value>**are defined. In this Primer, sometimes only a sample of the possible values are given to give an idea of what is possible rather than exhaustively list them all.
In BNF, the notation could be written:
<font-weight> ::= font-weight: <value>
This style is used if, for example, the definition of <value> needs to be given separately.
Some of the syntax definitions can be quite complicated. The notation used is asfollows:
Notation Meaning:

Notation	Meaning
<Abc>	A value of type Abc. Some of the more common types are discussed later.
abc	A keyword that must appear exactly as written.
X Y Z	X must occur then Y then Z in that order.
X | Y	Xor Y must occur.
X || Y || Z	X, Y or Z or some combination in any order.
[Abc]	The square brackets are used to group items together.
ST*	ST is repeated zero or more times. ST can be a bracketed set of items.
ST+	ST is repeated one or more times.
ST?	ST is optional. It can either appear once or not at all.
ST {A,B}	ST must occur at least A times and at most B times.

3. Length, Percentage, Color and URLs:

- 3.1 Introduction
- 3.2 Length Values
- 3.3 Percentage values

- 3.4 Color Values
- 3.5 URLs
- 3.6 CSS Specification

Introduction

Four values (<length>, <percentage> , <color>, <url>) are used by many of the rules. They are explained in this section.

Length Values

<length> ::= [+ | -]? <number><unit>

A length value is formed by an optional + or -, followed by a number, followed by a two-letter abbreviation that indicates the unit (this can be left out if the value is 0). There are no spaces in the middle of a length value.

Both relative and absolute length units are supported in CSS. Relative units give a length relative to another length property. The **relative length units** are:

Unit	Meaning
em	Historically width of letter **M**, but in CSS it equals the font height (if font is 12pt then 1em=12pt)
ex	x-height: the height of the letter **x** which varies between fonts (a 12pt Times Roman 'ex' will be bigger than a 12pt Baskerville 'ex')
px	pixels: in CSS the pixel is equal to one pixel on a normal computer display. So if the output is to go to a 1200dpi laser printer, the browser will take 1 pixel to mean about 12 laserprinter pixels.

The absolute length units are:

Notation	Meaning
in	inches
cm	centimetres
mm	millimetres
pt	Printing industry point; originally there was 72pt to a French inch which was larger than an English inch! Postscript and CSS define 1pt=1/72in
pc	picas; 1pc=12pt

Percentage Values:

<percentage> ::= [+ | -]? <number> %

A percentage value is formed by an optional + or -, followed by a number, followed by %. There are no spaces in a percentage value.

Percentage values are relative to other values, as defined for each property. Often the percentage value is relative to the element's font size but it might be the width of the page.

Color Values:

<color> ::= <keyword> | # <hex><hex><hex> | # <hex><hex><hex><hex><hex><hex>
<hex> |
rgb (<int><int><int>) | rgb (<percentage. <percentage ><percentage>)

A color value is a keyword or a numerical RGB specification. The original set of keyword colours was aqua, black, blue, fuchsia, gray, green, lime, maroon, navy, olive, purple, red, silver, teal, white and yellow. These have been enhanced as a result of the requirements from other specifications. The set of keywords with the corresponding RGB values given in Appendix C are likely to be supported by a browser.

RGB colours can be defined in one of four ways:

1. #rrggbb(e.g., #00ff00)
2. #rgb (e.g., #0f0)
3. rgb(x,x,x) where x is an integer between 0 and 255
4. rgb(y%,y%,y%) where y is a number between 0.0 and 100.0

The following examples all specify the same colour:

pre { color: red }
h1 { color: #foo }
p { color: #ff0000 }
h2 { color: rgb(255,0,0) }
h3 { color: rgb(100%, 0%, 0%) }

Values outside the range are clipped. So, for example, integer values have a maximum of 255 and everything above that will be changed to 255. Similarly 120% is interpreted as 100%.

URLs:

<url> ::= url(<whitespace>? [' | "]? <characters in url> [' | "]? <whitespace>)

A URL value is of the form:

url(xyz)

where xyz is the URL. The URL may be quoted with either single (') or double (") quotes and may have whitespace before the initial quote or after the final quote.
Parentheses, commas, spaces, single quotes, or double quotes in the URL must be escaped with a backslash. Partial URLs are interpreted relative to the style sheet source, not to the HTML source.
Some examples are:

body { background: url("circle.gif") }
body { background: url("http://www.clrc.images/monkey.gif") }

4. Font Properties:

- 4.1 Font Family (Sets typeface)
- 4.2 Font Style (Italicises text)
- 4.3 Font Variant (Mostly upper case)
- 4.4 Font Weight (Sets thickness of type)
- 4.5 Font Size (Sets size of text)
- 4.6 Font (Shorthand)

Font Family (set typeface)

font-family: [<family-name> | <generic-family>] [, [<family-name> | <generic-family>]]*
<family-name> ::= serif | sans-serif | cursive | fantasy | monospace
The font to use is set by the font-family property. Instead of defining a single font, a sequence of fonts should be listed. The browser will use the first if it is available but try the second and so on. The value can either be a specific font name or a generic font family.

Ex:p { font-family: "New Century Schoolbook", Times, serif }

Any font name containing spaces (multiple words) must be quoted, with either single or double quotes. That is why the first font name is quoted but Times is not.
The first two requests are for specific fonts **New Century Schoolbook** and **Times**. As both are serif fonts, the generic font family listed as a backup is the **serif** font family. In this case, Times will be used if New Century Schoolbook is not available, and a serif font if Times is not available.
Some example fonts are:

serif
Times New Roman, Bodini, Garamond
sans-serif
Trebuchet, Arial, Verdana, Futura, Gill Sans, Helvetica
cursive
Poetica, Zapf-Chancery, Roundhand, Script
fantasy
Critter, Cottonwood

monospace
Courier, Courier New, Prestige, Everson Mono
See also the **font** property which lets the user define a set of font properties in a single command..

Font Style (italicise text):

```
font-style: normal | italic | oblique
```

The font-style property has one of three values: **normal**, **italic** or **oblique** (slanted). A sample style sheet might be:

```
h3 { font-style: italic }
p { font-style: normal }
```

Italic and oblique are similar. ***Italic*** is normally a separate font while ***oblique*** often bends the normal font.

Font Variant (size of upper case):

```
font-variant: normal | small-caps
```

Normally, **upper case characters** are much larger than the lower-case characters. **Small-caps are** often used when all the letters of the word are in capitals. In this case the upper case characters are only slightly larger than the lowercase ones. An example is:

```
p { font-variant: small-caps }
```

For example, the text **Elephant** with font-variant set to **small-caps** would appear as ELEPHANT.

Font Weight (set thickness of type):

```
font-weight: normal | bold | bolder | lighter | 100 | 200 | 300 | 400 | 500 | 600 | 700 | 800 |
900
```

The font-weight property sets the weight of the font to values like **normal** and **bold**. It also has values **bolder** and **lighter** which are relative to any inherited font -weight. Block-level elements are displayed with default values inherited from the **body** or **div** element that surrounds it. It is also possible to define absolute font-weights. Values range from 100 to 900 with **normal** being 400 and bold being **700** approximately. If that much control is not possible, some of the weights may be grouped together (for example, 100, 200 and 300 may all look the same) and if the specified weight is not available, an alternative value will be chosen. Some examples are:

```
h1 { font-weight: 800 }
h2 { font-weight: bold }
h3 { font-weight: 500 }
p { font-weight: normal }
```

Font Size (set size of text):

```
font-size: <absolute-size> | <relative-size> | <length> | <percentage>
<absolute-size> ::= xx-small | x-small | small | medium | large | x-large | xx-large
<relative-size> ::= larger | smaller
```

The **font-size property sets** the size of the displayed characters. The absolute-size values are **small**, **medium** and **large** with additional possibilities like **x-small** (extra small) and **xx-small**. Relative sizes are **smaller** and **larger**. They are relative to any inherited value for the property. The length value defines the precise height in units like **12pt** or **1in**. Percentage values are relative to the inherited value.

Some examples are:

```
h1 { font-size: large }
h2 { font-size: 12pt }
h3 { font-size: 5cm }
p { font-size: 10pt }
li { font-size: 150% }
em { font-size: larger }
```

The guidance is that the medium font should be around 10pt so, in this case **li** and **em** might be similar in size.

Font (Shorthand):

```
font: [ <font-style> || <font-variant> ||
<font-weight>]? || <font-size> [ /<line-height> ]? || <font-family> ]
```

The **font property may** be used as a shorthand for the various font properties, as well as the line-height. For example:

```
p { font: italic bold 12pt/14pt Times, serif }
```

specifies paragraphs with a bold and italic Times or serif font with a size of 12 points and a line height of 14 points. Note: The order of attributes is significant: the font weight and style must be specified before the others.

5. Color and Background Properties:

- 5.1 Color
- 5.2 Background Color
- 5.3 Background Image
- 5.4 Background Repeat
- 5.5 Background Attachment
- 5.6 Background Position
- 5.7 Background (Sets background images or color)

Color

```
color: <color>
```

The color property sets the foreground colour of a text element. For example:

```
h1 {color: blue }
h2 {color: rgb(255,0,0) }
h3 {color: #0F0 }
```

Background Color:

```
background-color: <color> | transparent
```

The **background-color property** sets the background colour of an element. For example:

```
body { background-color: white }
h1 { background-color: #000080 }
```

The value **transparent** indicates that whatever is behind the element can be seen. For example if the background to the **body** element was specified as being red, a block-level element with **backgroundcolor** set to **transparent** would appear with a red background.

Background Image:

```
background-image: <url> | none
```

The **background-image** property sets the background image of an element.

For example:

```
body { background-image: url("/images/monkey.gif") }
p { background-image: url("http://www.cclrc.com/pretty.png") }
h1 { background-image: none }
```

When a background image is defined, a compatible background colour should also be defined for those users who do not load the image or to cover the case when it is unavailable. If parts of the image are transparent, the background colour will fill these parts.

For example:

```
<ul style="background-image:url('wall.png')">
<li>An item</li>
<li>Another</li>
<li>And third</li>
</ul>
```

Figure 5.1 shows what might be the result depending on the font-size and other properties specified for the list.

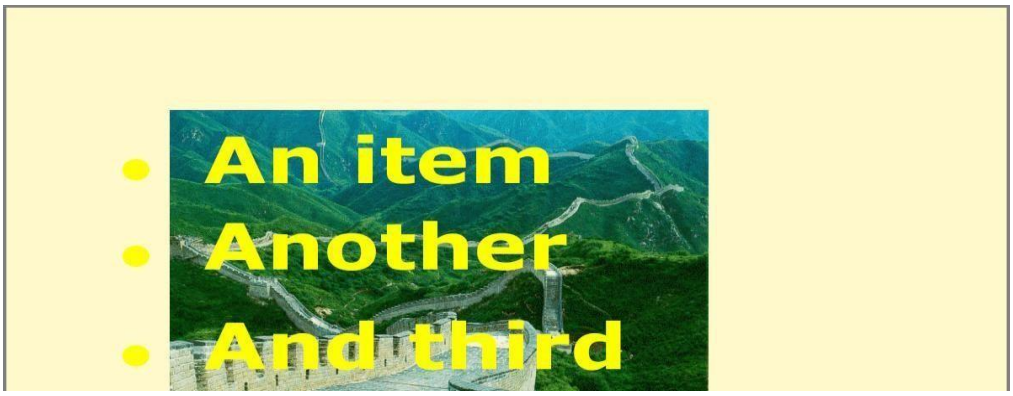


Figure 5.1: List with background-image

Background Repeat:

```
background-repeat: repeat | repeat-x | repeat-y | no-repeat
```

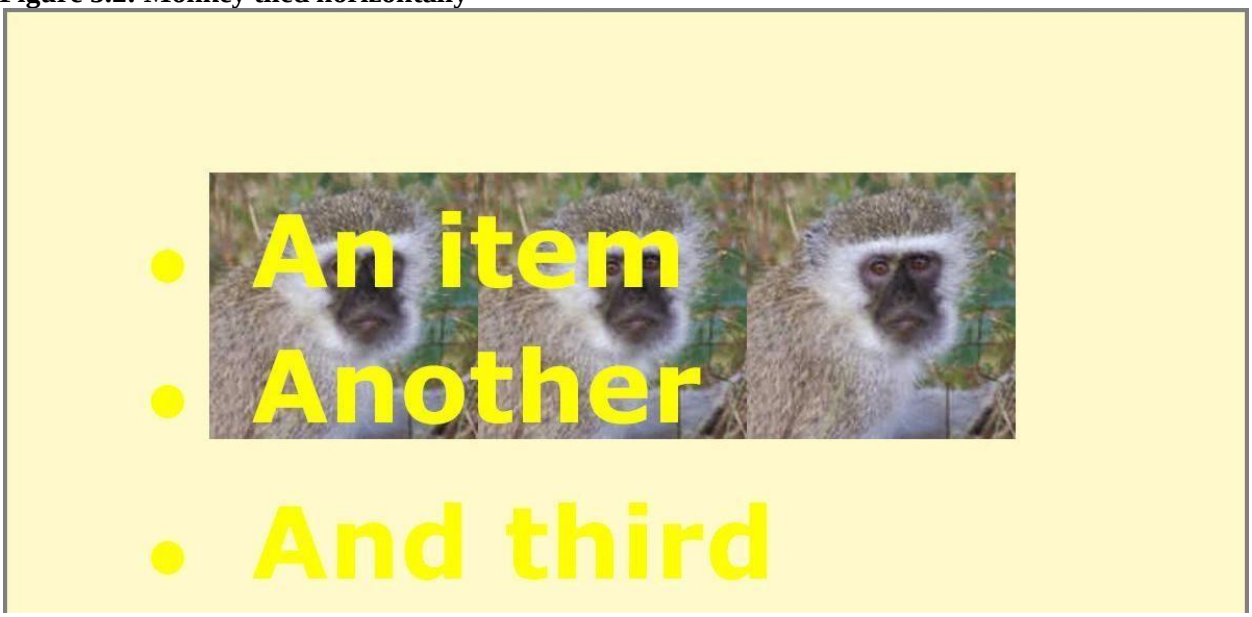
If the background image does not fill the whole element area, this description specifies how it is repeated. The **repeat-x** value will repeat the image horizontally while the **repeat-y** value will repeat the image vertically. Setting **background-image** to **repeat** will repeat the image in both x and y directions.

For example:

```
body { background-image: url(/images/monkey.gif) }
body { background-repeat: repeat-x }
```

In the above example, the monkey will only be tiled horizontallya as shown in Figure 5.2.

Figure 5.2: Monkey tiled horizontally



Background

Attachment: background-attachment: scroll | fixed

What is JavaScript?

- JavaScript is used in millions of Web pages to improve the **design, validate forms, detect browsers, create cookies, and much more.**
- JavaScript is the most popular scripting language on the internet, and works in all major browsers, such as **Internet Explorer, Mozilla, Firefox, Netscape, and Opera**.
- JavaScript does not provide the range of capabilities of standard programming languages. The Web developer is most likely to need to make Web pages interactive and dynamic.
- It is important to know JavaScript codes are **case sensitive;** you cannot indiscriminately use upper- and lower-case notation.

Your JavaScript code is open to the public.

Web Scripting Language

About Web Scripting Language:

- **Server-Side comprises the following languages:**
  - Active Server Pages (ASP)
  - Java Server Pages (JSP)
  - Cold Fusion
- **Client Side comprises the following languages:**
  - JavaScript
  - JScript
  - VBScript

The Real Name is ECMAScript

- **The Real Name is ECMAScript**
- JavaScript's official name is "ECMAScript". The standard is developed and maintained bythe ECMA organisation.

- ECMA-262 is the official JavaScript standard. The standard is based on **JavaScript (Netscape)** and **JScript (Microsoft).**
- The language was invented by **Brendan Eich** at Netscape (with Navigator 2.0), and has appearedin all Netscape and Microsoft browsers since 1996.
- The development of ECMA-262 started in 1996, and the first edition of was adopted by the ECMA General Assembly in June 1997.
- The standard was approved as an international ISO (ISO/IEC 16262) standard in 1998.
- The development of the standard is still in progress.

More on JavaScript

- JavaScript was designed to add interactivity to HTML pages since HTML is FLAT Language, No Programming Tools available.
- JavaScript is a scripting language
- A scripting language is a lightweight programming language
- A JavaScript consists of lines of executable computer code
- A 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 is Obeject Oriented Programming Language while JavaScript is Object based Scripting Language.
- Java and JavaScript are two completely different languages in both concept and design.
- Java (developed by Sun Microsystems) is a powerful and much more complexprogramming language - in the same category as C++ and C#.

What can a JavaScript Do?

JavaScript code can do any of the following: -

- **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 anyonecan put small "snippets" of code into their HTML pages
- **JavaScript can put dynamic text into an HTML page :-**A JavaScript statement like this: document.write("<h1>" + name + "</h1>") can write a variable text into an HTML page
- **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.

What can't a JavaScript Do?

JavaScript code cannot do any of the following: -

- use printers or other devices on the user's system or the client-side LAN

- *directly* access files on the user's system or the client-side LAN ; the only exception is the access to the browser's cookie files.
- *directly* access files on the Web server.
- implement multiprocessing or multithreading.
- If you do need to access files or perform other "privileged" operations, you can use JavaScript in combination with a *Java applet*. Signed Java applets are allowed to do "privileged" things, and your JavaScript programs can exchange information with applets. However, you have to bear in mind ***the biggest JavaScript/Java limitation***: the user can always disable Java or JavaScript or both!
- access any database.

How to embed JavaScript with HTML page?

Different ways to include JavaScript in HTML Code:-

- JavaScripts in a page will be executed immediately while the page loads into the browser. This is not always what we want. Sometimes we want to execute a script when a page loads, other times when a user triggers an event.
- **Scripts in the head section:** Scripts to be executed when they are called, or when an event is triggered, go in the head section. When you place a script in the head section, you will ensure that the script is loaded before anyone uses it.

```
<html><head>
<script type="text/javascript">
        //statements...
</script>
</head></html>
```

- **Scripts in the body section:** Scripts to be executed when the page loads go in the body section. When you place a script in the body section it generates the content of the page.

```
<html><head></head>
<body>
<script type="text/javascript">
        //statements...
</script>
</body></html>
```

- **Scripts in both the body and the head section:** You can place an unlimited number of scripts in your document, so you can have scripts in both the body and the head section.

```
<html><head>
<script type="text/javascript">
        //statements...
</script>
</head><body>
<script type="text/javascript">
        //statements...
</script>
</body>
```

- **Using an External JavaScript**
  - Sometimes you might want to run the same JavaScript on several pages, without having to write the same script on every page.
  - To simplify this, you can write a JavaScript in an external file. **Save the external JavaScript file with a .js file extension.**
  - **Note:** The external script cannot contain the <script> tag!
  - To use the external script, point to the .js file in the "src" attribute of the <script> tag:
  - **Note:** Remember to place the script exactly where you normally would write the script!

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

<script src="tom.js"></script>
</head>
<body></body></html>

JavaScript Statements

- JavaScript is a sequence of statements to be executed by the browser.
- The word **document.write** is a standard JavaScript command for writing output to a page.

JavaScript Statements

- JavaScript is a sequence of statements to be executed by the browser.
- The word **document.write** is a standard JavaScript command for writing output to a page.

Example :-

document.write("Hi! I am james Bond!");

- Note:- The semicolon is optional (according to the JavaScript standard), and thebrowser is supposed to interpret the
  end of the line as the end of the statement. Because of this you will often see examples without the semicolon at the end.
- **Note:** Using semicolons makes it possible to write multiple statements on one line.

JavaScript Code Vs JavaScript Blocks

JavaScript Code

- JavaScript code (or just JavaScript) is a sequence of JavaScript statements.
- Each statement is executed by the browser in the sequence they are written.

Example :-

```
<script type="text/javascript">
        document.write("<h1>This is a first header</h1>");
        document.write("<p>This is a first paragraph</p>");
        document.write("<p>This is another paragraph</p>");
    </script>
```

JavaScript Blocks

- JavaScript statements can be grouped together in blocks.
- Blocks start with a left curly bracket {, and ends with a right curly bracket }.
- The purpose of a block is to make the sequence of statements execute together.

Example :-

```
<script type="text/javascript">
        {
        document.write("<h1>This is a first header</h1>");
        document.write("<p>This is a first paragraph</p>");
        document.write("<p>This is another paragraph</p>");
        }
    </script>
```

- The example above is not very useful. It just demonstrates the use of a block. Normally a block is used to group statements together in a function or in a condition (where a group of statements should be executed if a condition is met).

Comments in JavaScript

- JavaScript comments can be used to make the code more readable.
- Comments are simply lines of text that describe or explain the script; they are not treated as executable code.

Single line comments start with //

Example :-

```
<script type="text/javascript">
// This will write a header:
document.write("<h1>This is first header</h1>");
// This will write two paragraphs:
document.write("<p>This is a first paragraph</p>");
</script>
```

Example :-

```
<script type="text/javascript">
/*
The code below will write
one header and two paragraphs
*/
document.write("<h1>This is first header</h1>");
document.write("<p>This is first paragraph</p>");
</script>
```

Example :-

```
<script type="text/javascript">
document.write("<h1>This is a header</h1>");
document.write("<p>This is a paragraph</p>");
//document.write("<p>This is another paragraph</p>");</script>

<script type="text/javascript">
/*
document.write("<h1>This is a header</h1>");
document.write("<p>This is a paragraph</p>");
document.write("<p>This is another paragraph</p>");
*/
</script>
```

Example :-

```
<script type="text/javascript">
document.write("Hello"); // This will write "Hello"
document.write("World");        // This will write "World"
</script>
```

Variables declaration in JavaScript

- A variable is a "container" for information you want to store. A variable's value can change during the script. You can refer to a variable by name to see its value or to change its value.
  Rules for variable names:
  - Variable names are case sensitive
  - They must begin with a letter or the underscore character
- JavaScript does not have **explicit data types**
- There is no way of specifying that a particular variable represents an integer, a string or a real.
- The same variable may be interpreted differently in different contexts.
- All JavaScript variables are declared applying the **keyword var.**

For example:

var x,y=7

- **JavaScript recognizes the following types of values:**
  - **Numbers**, like 989 or 3.1415
  - **Logical** (Boolean) values, either true or false
  - **Strings**, like "javaskool!"
  - **null**, a special keyword denoting a null value

Life of Variables :-

- When you declare a variable within a function, the variable can only be accessed within that function. When you exit the function, the variable is destroyed. These variables are called local variables.
- If you declare a variable outside a function, all the functions on your page can access it. The lifetime of these variables starts when they are declared, and ends when the page is closed.

Keywords:-

var, true, false, new, return

Construct block with JavaScript [if..else,switch..case]

- **JavaScript If...Else Statements or Conditional Statements**

Conditional statements in JavaScript are used to perform different actions based on different conditions.

Followings are the 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

If Statement

You should use the if statement if you want to execute some code only if a specified condition is true.

Syntax

```
if (condition)
{
//code to be executed if condition is true
}
```

If...else Statement

If you want to execute some code if a condition is true and another code if the condition is not true, use the if... else statement.

Syntax

```
if (condition)
{
//code to be executed if condition is true
}
else
{
//code to be executed if condition is not true
}
```

If...else if...else Statement

You should use the if....else if...else statement if you want to select one of many sets of lines to execute.

Syntax

```
if (condition1)
{
//code to be executed if condition1 is true
}
else if (condition2)
{
//code to be executed if condition2 is true
}
else
{
//code to be executed if condition1 and
//condition2 are not true
}
```

JavaScript Switch Statement

Conditional statements in JavaScript are used to perform different actions based on different conditions.

Syntax

```
switch(n)
{
case 1:
  //execute code block 1
  break;
case 2:
  //execute code block 2
  break;
default:
  //code to be executed if n is
  //different from case 1 and 2
}
```

Example:

```
<script type="text/javascript">
//Note that Sunday=0, Monday=1,..... Saturday=6.
var d=new Date();
theDay=d.getDay();
switch (theDay)
{
case 5: document.write("Finally Friday");
  break;
case 6: document.write("Super Saturday");
  break;
case 0: document.write("Sleepy Sunday");
  break;
default:
  document.write("I'm looking forward to this weekend!");
}
</script>
```

Operators with JavaScript

Arithmetic Operators :-

Operator	Description	Example	Result
+	Addition	x=5; y=5; document.write(x+y);	10
-	Subtraction	x=5; y=5; document.write(x-y);	0
*	Multiplication	x=5; y=5; document.write(x*y);	25
/	Division	x=15; y=5; document.write(x/y);	3
%	Modulus (division remainder)	x=16; y=5; document.write(x%y);	1
++	Increment	x=5;x++; document.write(x);	6
--	Decrement	x=5;x--; document.write(x);	4

Assignment Operators :-

Operator	Example	Is The Same As
=	x=y	x=y
+=	x+=y	x=x+y
-=	x-=y	x=x-y
*=	x*=y	x=x*y
/=	x/=y	x=x/y
%=	x%=y	x=x%y

Comparison Operators :-

Operator	Description	Example
==	is equal to	5==8 returns false
===	is equal to (checks for both value and type)	x=5, y="5"; x==y returns true x===y returns false
!=	is not equal	6!=8 returns true
>	is greater than	6>8 returns false
<	is less than	6<8 returns true
>=	is greater than or equal to	6>=8 returns false
<=	is less than or equal to	6<=8 returns true

Logical Operators:-

O erator	Description	Example
&&	and	x=6; y=3 ; **(x < 10 && y > 1) returns true**
||	or	x=6; y=3 ; **(x==5 || y==5) returns false**

!	not	x=6; y=3; **!(x==y) returns true**

String Operator :-

- A string is most often text, for example "Hello India!".
- To stick two or more string variables together, use the + operator.

Example :-

```
txt1="What";
txt2="a nice baby!";
txt3=txt1+txt2;
document.write(txt3);
```

The variable txt3 now contains "What a nice baby!".

Conditional Operator :- [?:]

- JavaScript also contains a conditional operator that assigns a value to a variable based on some condition.

Syntax :

variablename=(condition)?value1:value2

Example:-

```
gender="Male";
str= (gender=="Male") ? "Mr." : "Ms.";
document.write(str);
```

output :- Mr.

For Loop in JavaScript

Loops in JavaScript are used to execute the same block of code a specified number of times or while a specified condition is true.

In JavaScript there are two different kind of loops:-

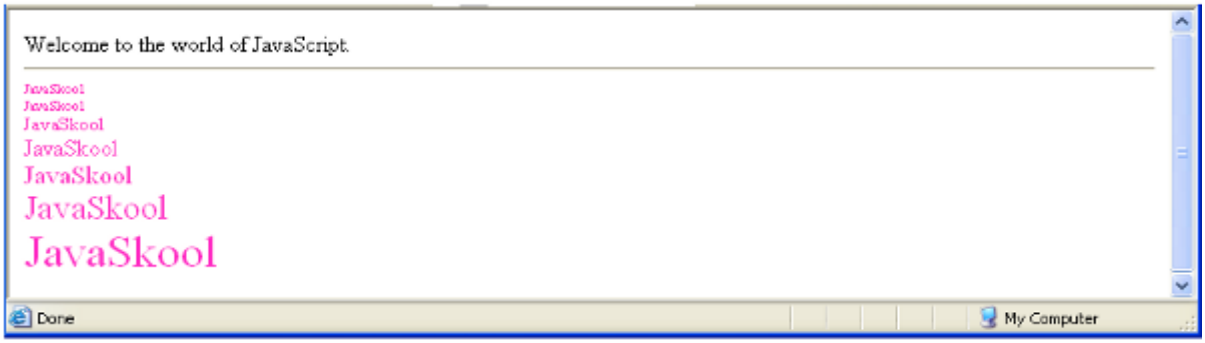
- **for** - loops through a block of code a specified number of times
- **while** - loops through a block of code while a specified condition is true

The for loop

Syntax:-

```
for (var=startvalue;var<=endvalue;var=var+increment)
{
   code to be executed
}
```

```
<html>
<head>
<title>my Page Title</title>
<head>
<Body>
Welcome to the world of JavaScript.
<hr>
<script language='javascript'>

    for(i=0;i<7;i++)
    {
    document.write("<font size='"+i+" color=#ff33c"+i+'>JavaSkool</font><br>");
    }
</script>
</body>
<html>
```



The while loop

The while loop is used when you want the loop to execute and continue executing while the specified condition is true.

while (var<=endvalue)
{
code to be executed
}

The do...while Loop

The do...while loop is a variant of the while loop. This loop will always execute a block of code ONCE, and then it will repeat the loop as long as the specified condition is true. This loop will always be executed at least once, even if the condition is false, because the code is executed before the condition is tested.

do
{
code to be executed
}while (var<=endvalue)

JavaScript Break and Continue

There are two special statements that can be used inside loops: break and continue thru which loop can be controlled.

- **Break:**
  - The break command will break the loop and continue executing the code that follows after the loop (if any).
- **Continue:**
  - The continue command will break the current loop and continue with the next value.

JavaScript For...In Statement

The for...in statement is used to loop (iterate) through the elements of an array or through the properties of an object.

Syntax:-

for (variable in object)
{
code to be executed

}

Example :-

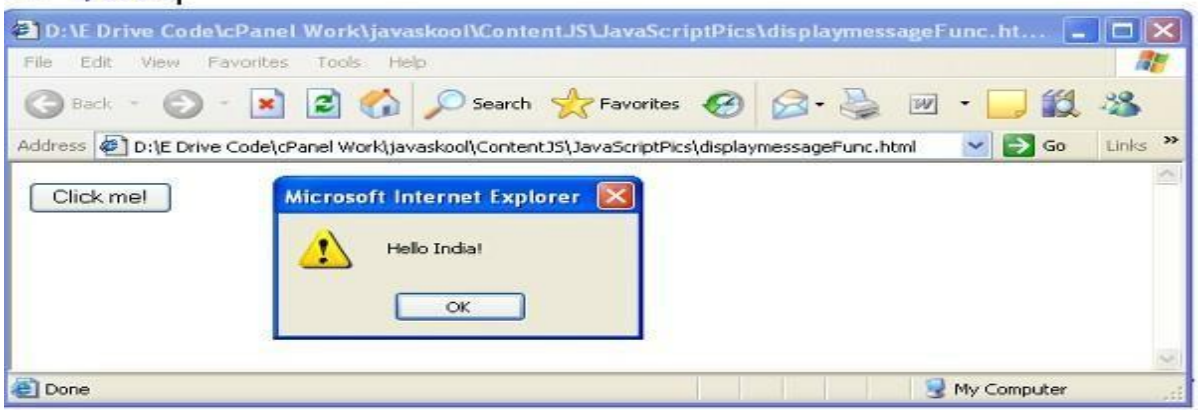
```
<html>
<body>
<script type="text/javascript">
var x
var mycars = new Array()
mycars[0] = "Saab"
mycars[1] = "Volvo"
mycars[2] = "BMW"

for (x in mycars)
{
document.write(mycars[x] + "<br />")
}
</script>
</body>
</html>
```

How to make Functions in JavaScript?

- A function is a **reusable code-block** that will be executed by an event, or when the functionis called.
- To keep the browser from executing a script when the page loads, you can put your script into a function.
- A function contains code that will be executed by an event or by a call to that function.
- You may call a function from anywhere within the page (or even from other pages if the function is embedded in an external .js file).
- Functions can be defined both in the <head> and in the <body> section of a document. However, to assure that the function is read/loaded by the browser before it is called, it could be wise to put it in the <head> section.

Example:-

```
<html>
<head>
<script type="text/javascript">
function displaymessage()
{
alert("Hello India!")
}
</script>
</head>
<body>
<form>
<input type="button" value="Click me!" onclick="displaymessage()" >
</form>
</body>
</html>
```



The syntax for creating a function is:-

function *functionname*(*var1*,*var2*,...,*varX*)
{
//statement .....
}

Here var1, var2, etc are variables or values passed into the function. The { and the } defines the start and end of the function.

Note: A function with no parameters must include the parentheses () after the function name:

function *functionname*()
{
//statement .....
}

Note: The word function must be written in lowercase letters, otherwise a JavaScript error occurs!.

The return Statement :-

- The return statement is used to specify the value that is returned from thefunction.
- So, functions that are going to return a value must use the return statement.

Example:-

```
function add(a , b)
{
x=a+b;
return x;
}
```

When you call the function above, you must pass along two parameters:

```
x=add(5,10);
or
document.write(add(5,10));
```

JavaScript Popup Boxes

- Alert Box
- **Confirm Box**
- **Prompt Box**

JavaScript Popup Boxes

- **In JavaScript we can create three kinds 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");

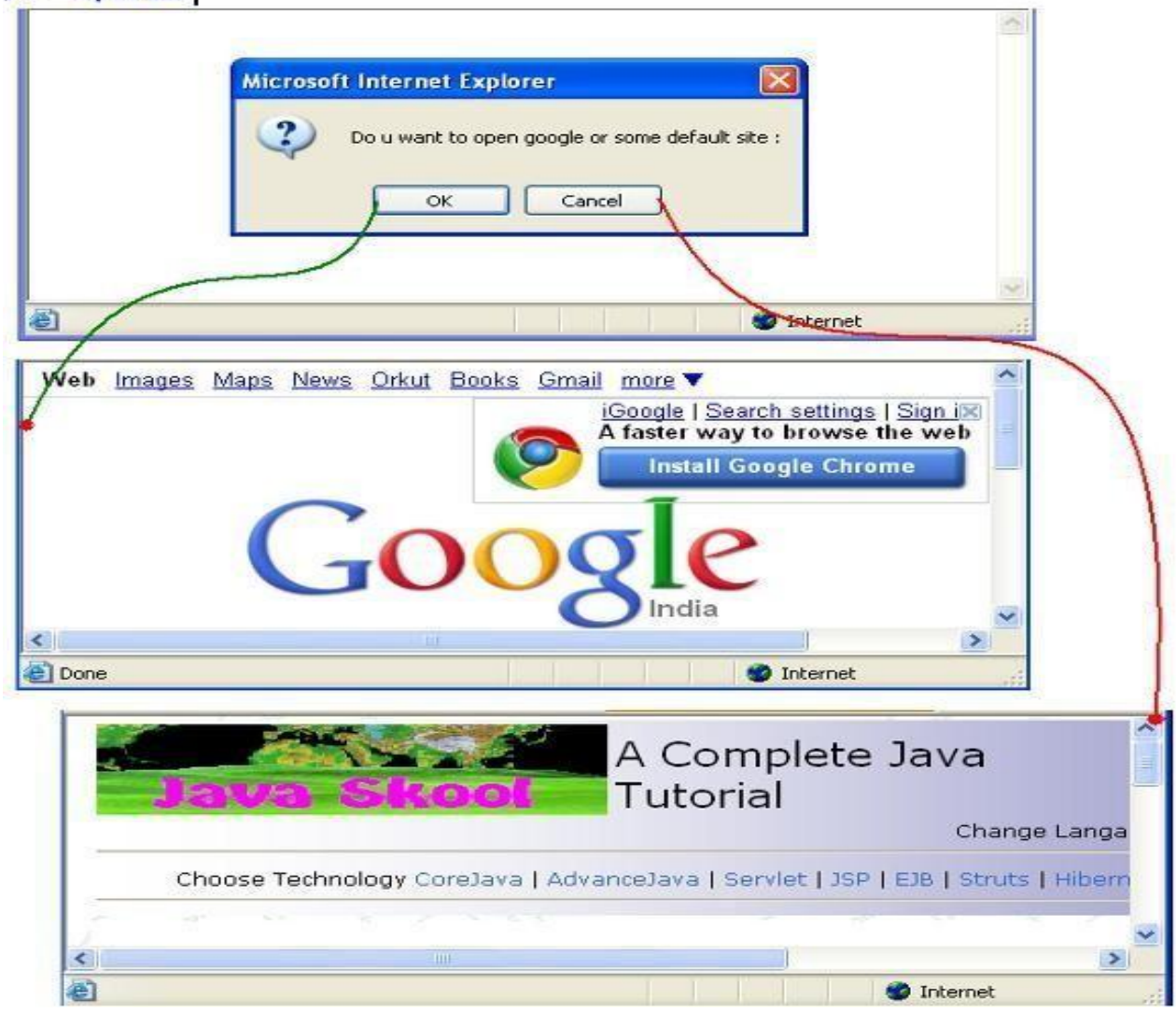
```
1 <html>
2 <head><title>Alert Page</title></head>
3 <Body>
4     <script language="javascript">
5         alert("Welcome to javaskool.com");
6     </script>
7 </body>
8 <html>
```

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");

```
1  <html>
2  <head><title>Confirm Page</title></head>
3  <Body>
4
5  <script language="javascript">
6      x=confirm("Do u want to open google or some default site : ");
7      if(x)
8          window.location.href="http://google.co.in";
9      else
10         window.location.href="http://www.javaskool.com";
11
12 //name=confirm("Do you want to confirm");
13 //document.write(name);
14 </script>
15 </body>
16 </html>
```

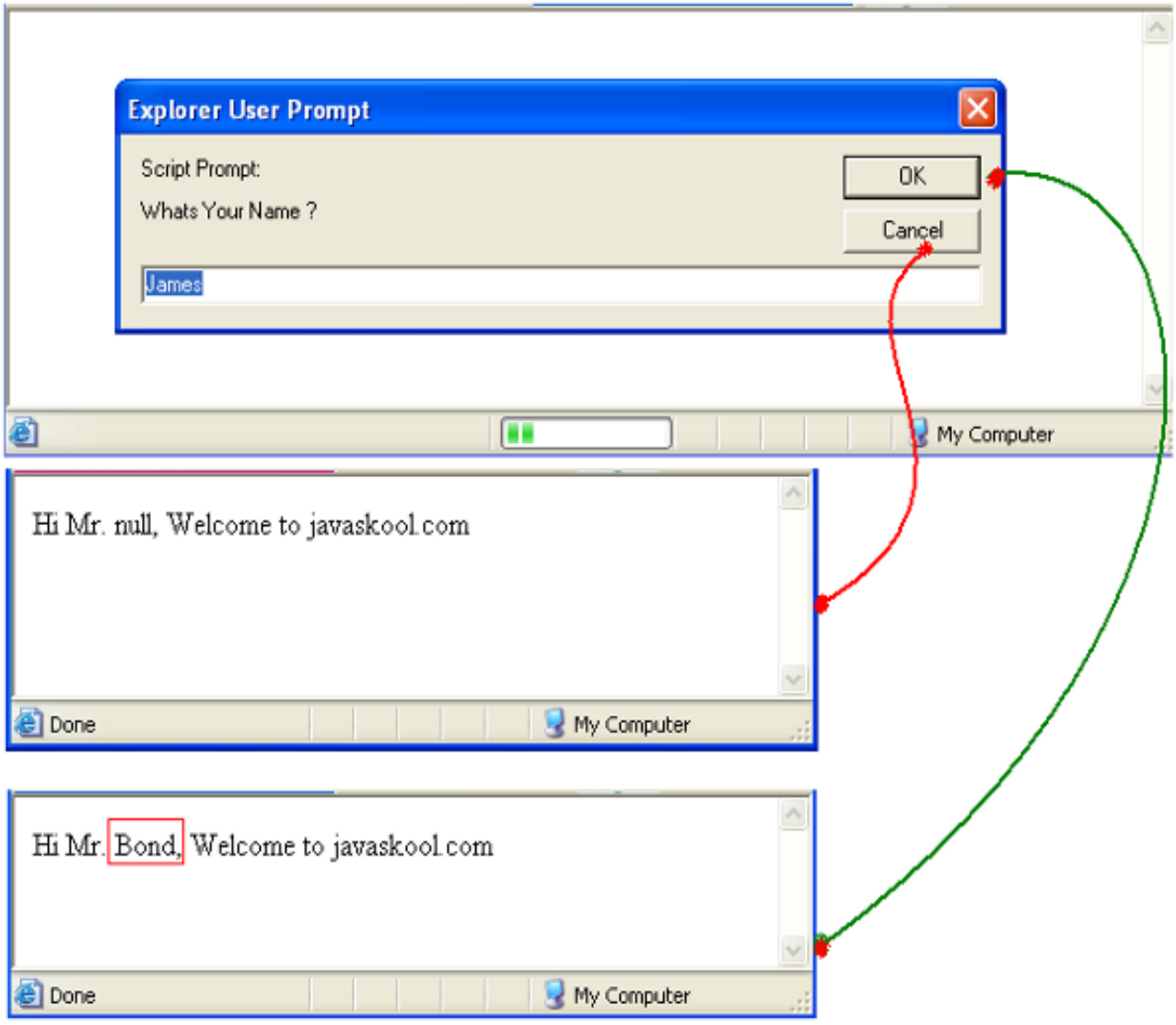


Prompt Box

- 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");

```
1  <html>
2  <head><title>Prompt Page</title><head>
3  <Body>
4      <script language="javascript">
5          name=prompt("Whats Your Name ?","James");
6          document.write("Hi Mr. "+name+", Welcome to javaskool.com");
7      </script>
8  </body>
9  <html>
```



UNIT-II

Objects in JAVASCRIPT and XML

Objects in JavaScript: Data and objects in JavaScript, regular expressions, exception handling, built-in objects, events; Dynamic HTML with JavaScript: Data validation, opening a new window, Rollover buttons, moving images, multiple pages in a single download, floating logos.
XML: Basics XML, document type definition, xml schemas, Document Object Model, presenting XML.

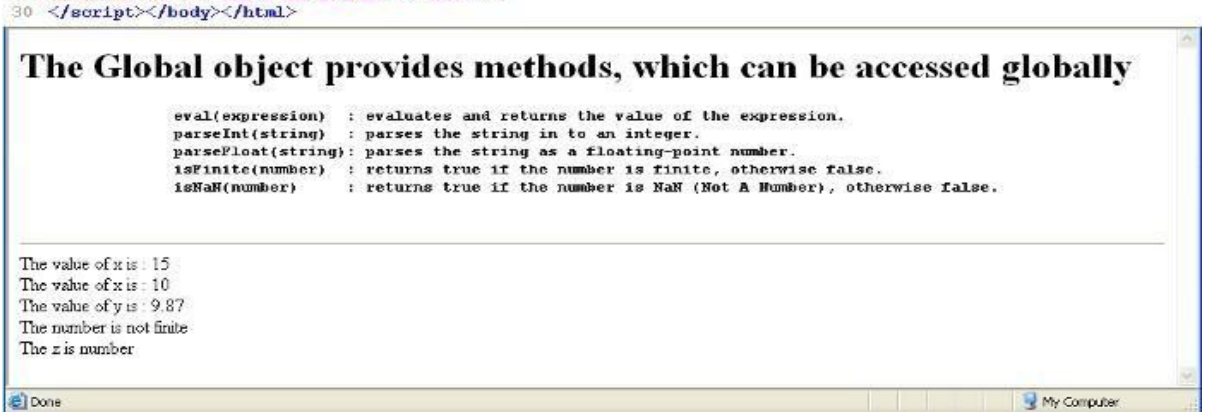
Methods from Global Object in JavaScript

- eval
- **parseInt**
- **parseFloat**
- isFinite
- isNaN

Methods from Global Object

- **The Global object provides methods, which can be accessed globally**
  - **eval(expression)** : evaluates and returns the value of the expression.
  - **parseInt(string)** : parses the string in to an integer.
  - **parseFloat(string)** : parses the string as a floating-point number.
  - **isFinite(number)** : returns true if the number is finite, otherwise false.
  - **isNaN(number)** : returns true if the number is NaN (Not A Number), otherwise false.

```
<html>
<head><title>Global Object</title></head>
<body>
    <h1>The Global object provides methods, which can be accessed globally </h1>
    <h4><pre>
        eval(expression)   : evaluates and returns the value of the expression.
        parseInt(string)   : parses the string in to an integer.
        parseFloat(string): parses the string as a floating-point number.
        isFinite(number)   : returns true if the number is finite, otherwise false.
        isNaN(number)      : returns true if the number is NaN (Not A Number), otherwise false.
    </pre></h4>
<hr>
<script language='javascript'>
var x ='10+5';
var z="9.87";

document.write('The value of x is : " +eval(x) +'<br>');
document.write('The value of x is : " +parseInt(x)+'<br>");
document.write('The value of y is : " +parseFloat(z)+'<br>');

if ( isFinite(x/0))
document.write('The number is finite" +"<br>');
else
document.write('The number is not finite' +'<br>');

if (isNaN(z))
document.write('The z is not a number' +'<br>');
else
document.write('The z is number' +'<br>");
</script></body></html>
```

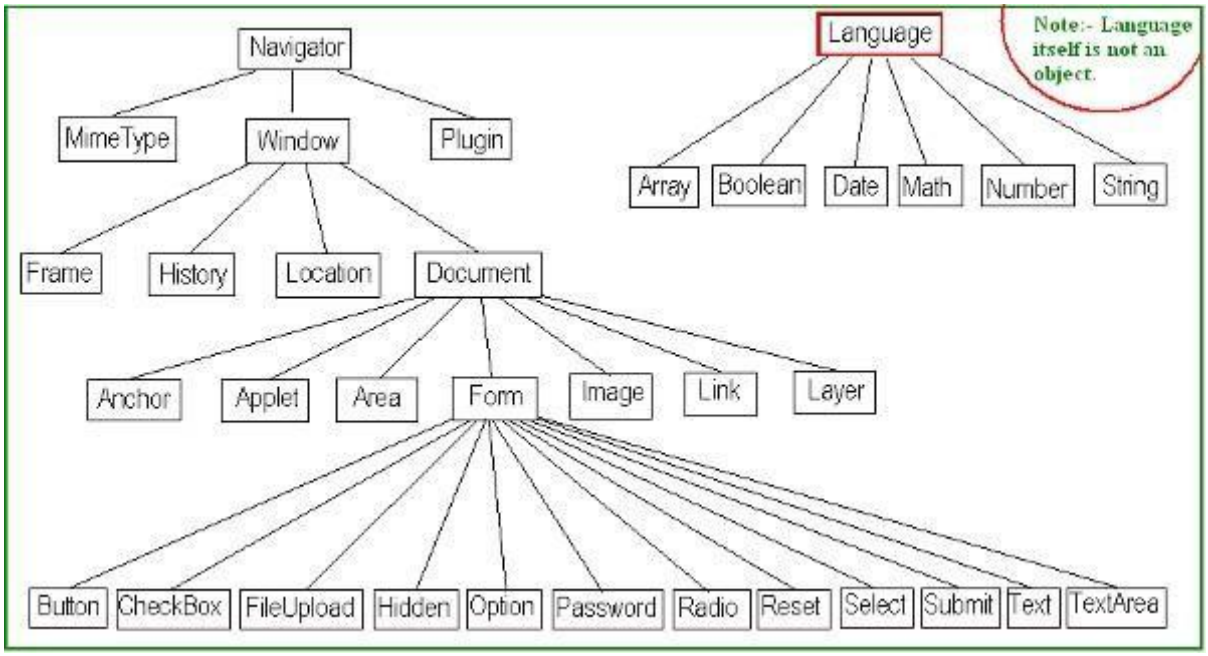


Objects Hierarchy With JavaScript

- Language Object
- **DOM Hierarchy**

JavaScript Object Hierarchy

- Many JavaScript objects are contained within each other.
- JavaScript objects have a container to contained object relationship rather than a class and subclass relationship.
- Properties are not inherited from one type of object to another.
- **There are two main types of JavaScript objects.**
  - **Language Objects :** Objects provided by the language and are not dependent on other objects.
  - **Navigator Objects:** Objects provided by the client browser. These objects are all sub objects to the navigator object.



DOM Hierarchy

1. The programming interface to the HTML objects appearing on a Web page is known as the Document Object Model (DOM).
2. The DOM supplies the properties and methods associated with HTML elements anddefines how these properties are set and how these methods are activated to effect changes in the elements.
3. More broadly, the DOM is the full collection of browser components both surrounding and comprising a Web page, including the browser itself and its component windows, frames, documents, forms, and HTML elements.
4. All of these browser components are programmable through the DOM.

Identifying DOM Objects

- In order to program DOM objects, they must be identified to the scripts that manipulate their properties and methods.
- The following table summarizes several of the references used within scripts to identify common DOM objects.

Refernces	**Object**
navigator	The browser itself.
window	The main browser window.
window.framename	A frame that occupies the browser window and identified by its assigned name.
window.document	The document appearing in the main browser window.
window.framename.document	The document appearing in a frame identified by its assigned name.
document.getElementById("id")	An XHTML element appearing in a document and identified by its assigned id value.

DOM Components, Properties, and Methods

- There are literally hundreds of property settings that can be applied to elements on a Webpage; there are dozens of methods.
- In addition, there are properties and methods associated with the browser itself, with its windows and frames, with the documents appearing inside windows and frames, and with other components of the Document Object Model.

Note:- that an object is just a special kind of data. An object has properties and methods.

- **Properties :- Properties are the values associated with an object.**

```
<script type="text/javascript">
var txt="Hello World!"
document.write( txt.length )
</script>
```

- **Methods :- Methods are the actions that can be performed on objects.**

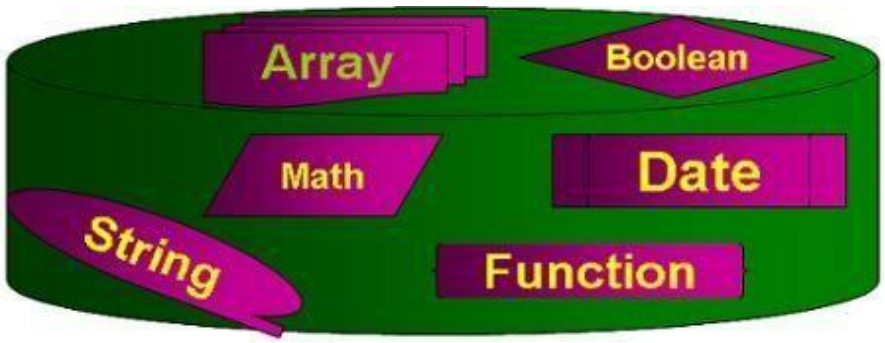
```
<script type="text/javascript">
var str="Hello India!"
document.write( str.toUpperCase() )
</script>
```

Objects Available with JavaScript

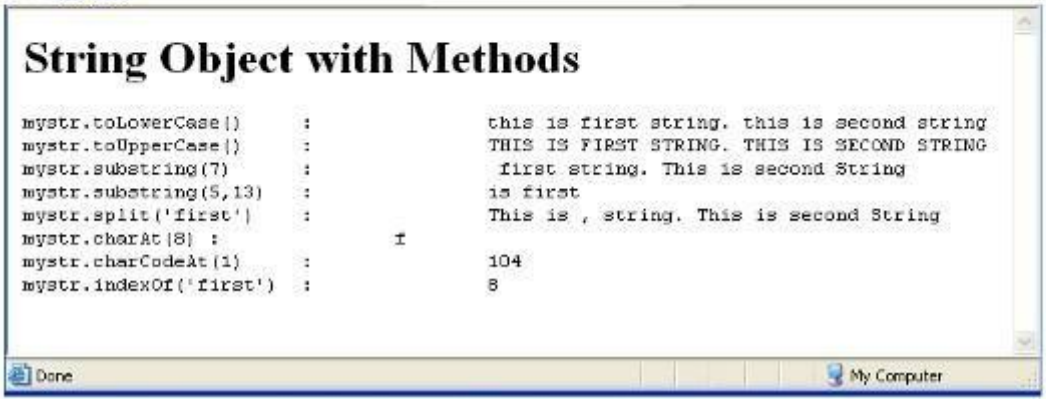
- String Object
- **Array Object**
- **Math Object**
- Date Object
- Function Object
- Boolean Object

String Object

Predefined Object supported by JavaScript:-

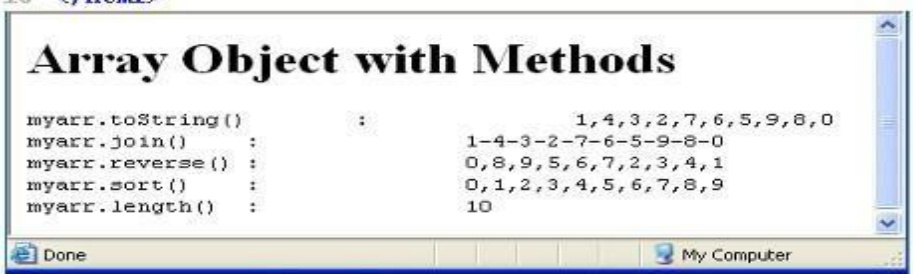


```
<html>
<head><title>String Object</title></head>
<body>
    <h1>String Object with Methods </h1>
        <script language='javascript'>
        mystr=new String('This is first string. This is second String');

        document.write("<PRE>");
        document.write("mystr.toLowerCase()\t:\t\t\t"  + mystr.toLowerCase()+'<BR>');
        document.write("mystr.toUpperCase()\t:\t\t\t"  + mystr.toUpperCase()+'<BR>');
        document.write("mystr.substring(7)\t:\t\t'     + mystr.substring(7)+"<BR>');
        document.write("mystr.substring(5,13)\t:\t\t\t" + mystr.substring(5,13)+'<BR>');
        document.write("mystr.split('first')\t:\t\t'   + mystr.split('first')+"<BR>');
        document.write("mystr.charAt(8)\t:\t\t\t"      + mystr.charAt(8)+'<BR>');
        document.write("mystr.charCodeAt(1)\t:\t\t\t"  + mystr.charCodeAt(1)+'<BR>');
        document.write("mystr.indexOf('first')\t:\t\t'+ mystr.indexOf('first')+"<BR>');
        document.write("</PRE>');
        </script>
</body>
</html>
```



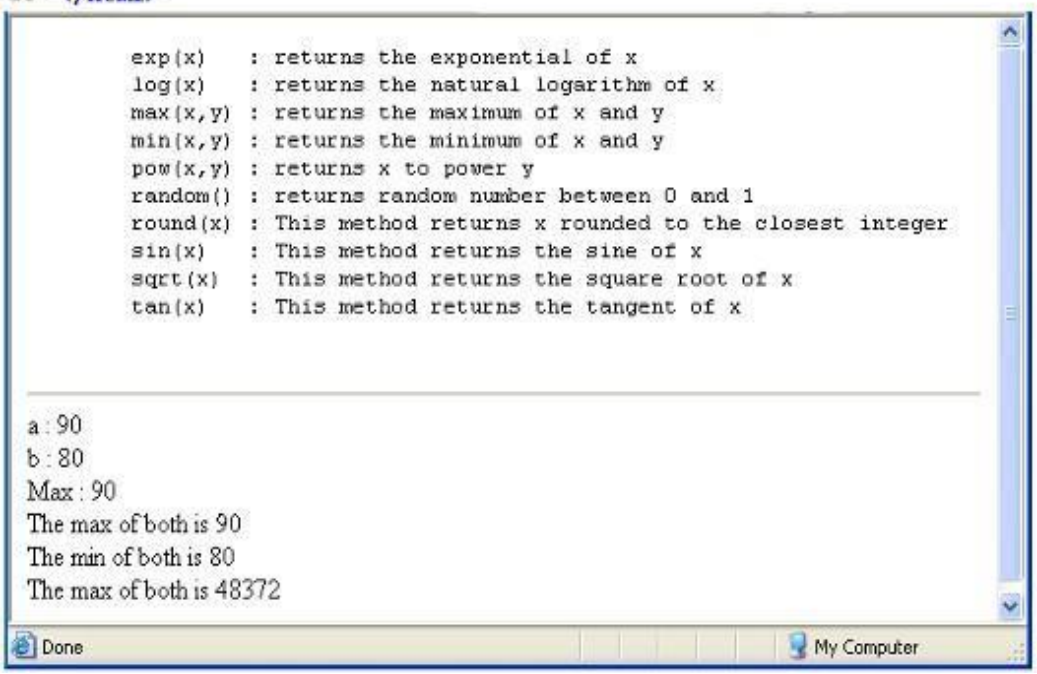
Array Object

```
<html>
<head><title>Array Object</title></head>
<body>
    <h1>Array Object with Methods </h1>
    <script language='javascript">
    myarr=new Array(1,4,3,2,7,6,5,9,8,0);
    document.write("<PRE>");
    document.write("myarr.toString()\t:\t\t"  + myarr.toString()+"<BR>");
    document.write("myarr.join()\t:\t\t"      + myarr.join('-')+"<BR>");
    document.write("myarr.reverse()\t:\t\t'   + myarr.reverse()+"<BR>");
    document.write("myarr.sort()\t:\t\t"      + myarr.sort()+"<BR>");
    document.write("myarr.length()\t:\t\t"    + myarr.length+"<BR>");
    document.write("</PRE>");
    </script>
</body>
</html>
```



Math Object

```
<html>
<head><title>Math Object</title></head>
<body>
    <pre>
    exp(x)    : returns the exponential of x
    log(x)    : returns the natural logarithm of x
    max(x,y)  : returns the maximum of x and y
    min(x,y)  : returns the minimum of x and y
    pow(x,y)  : returns x to power y
    random()  : returns random number between 0 and 1
    round(x)  : This method returns x rounded to the closest integer
    sin(x)    : This method returns the sine of x
    sqrt(x)   : This method returns the square root of x
    tan(x)    : This method returns the tangent of x
    </pre>
<hr>
    <script language='javascript">
    var x="90";
    var y="80";
    a=parseInt(x);
    b=parseInt(y);
    document.write("a : "+a + '<br>');
    document.write("b : "+b + '<br>');
    document.write("Max : "+Math.max(a,b) + "<br>");
    document.write("The max of both is " + Math.max(x,y) +"<br>");
    document.write("The min of both is " + Math.min(x,y) +"<br>");
    document.write("The max of both is " + Math.round('48372.483') +'<br>');
    </script>
</body>
</html>
```



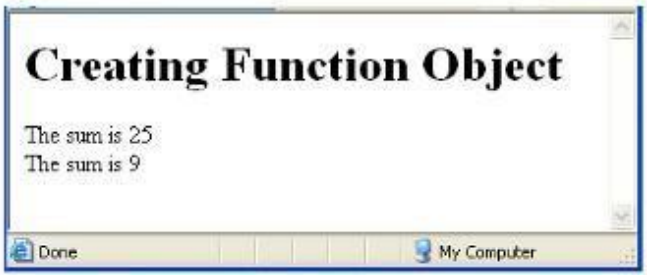
Date Object

```
<html>
<head><title>Date Object</title></head>
<body>
    <h1>Date Object with Methods </h1>
    <h3>Note:- Date Object only supports after 1/1/1970 </h3>
    <script language="javascript">
    mydate=new Date();
    document.write('<PRE>');
    document.write("Today's Date : \t");
    document.write(mydate.getDate()+'/'+mydate.getMonth()+'/'+mydate.getYear());
    document.write("<BR>");
    document.write("System Time :\t');
    document.write(mydate.getHours()+":"+mydate.getMinutes()+':'+mydate.getSeconds());
    document.write("<BR>");
    document.write("</PRE>');
    </script>
</body>
</html>
```

Function Object

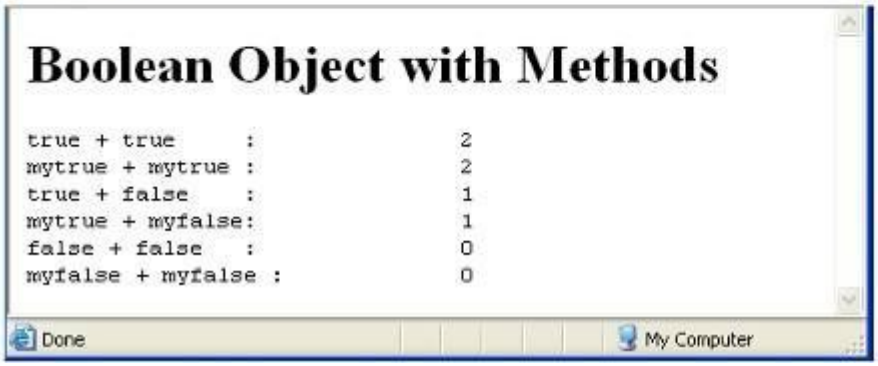
Function object can be used for both, it could be without return or with return

```
<html>
<head><title>Function</title></head>

<body>

<h1>Creating Function Object</h1>

    <script language='javascript'>
        func1=new Function('x','y',"tot=x+y; document.write('The sum is '+tot)');
        func1(12,13);
        document.write('<br>');

        func2=new Function('x','y',"tot=x+y; return tot');
        x=func2(4,5);
        document.write('The sum is "+x);
    </script>

</body>
</html>
```



Boolean Object

```
<html>
<head><title>Boolean Object</title></head>
<body>
    <h1>Boolean Object with Methods </h1>
    <script language='javascript'>
    mytrue =new Boolean(true);
    myfalse =new Boolean(false);
    document.write("<PRE>");
    document.write('true + true      : \t\t'   + (true+true) +"<br>");
    document.write('mytrue + mytrue : \t\t'   + (mytrue+mytrue)+"<br>");
    document.write('true + false     : \t\t'   + (true+false)+"<br>");
    document.write('mytrue + myfalse: \t\t'   + (mytrue+myfalse)+"<br>");
    document.write('false + false    : \t\t'   + (false+false)+"<br>");
    document.write('myfalse + myfalse : \t\t'+ (myfalse+myfalse)+"<br>");
    document.write("</PRE>");
    </script>
</body>
</html>
```



Navigator Object in JavaScript [Top Object]

- **Navigator Object**
- **Examples**

Navigator Object

■

The navigator object provides browser specific information : version, type , MIME types supported.

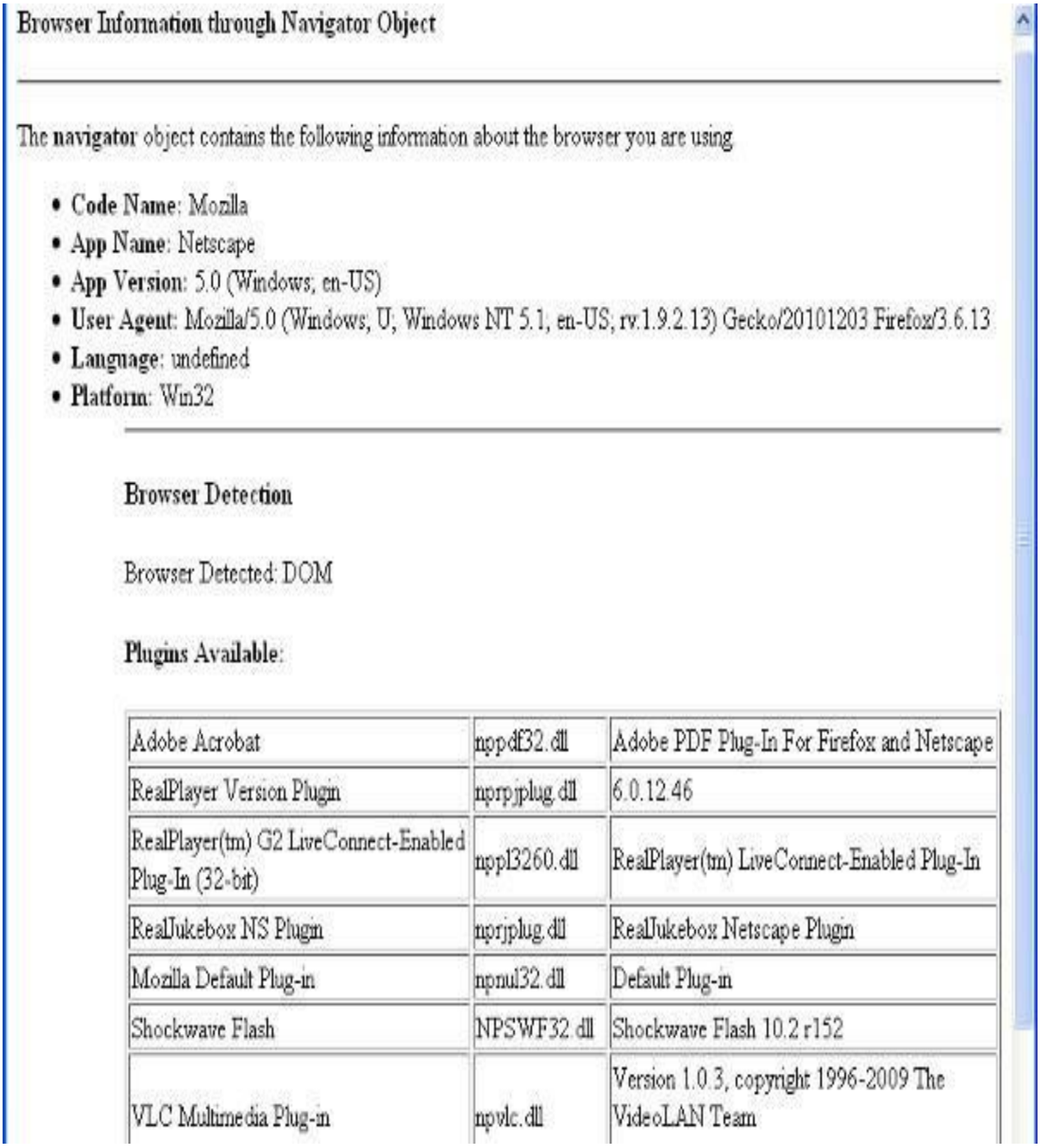
- **appVersion :- version information of the browser.**
- **appCodeName :– the code name of the browser.**
- **appName :- the name of the browser.**
- **userAgent :- the user-agent header sent in the HTTP protocol from the browser to the web server.**
- **plugins :- array of all the plug-ins installed in the browser.**
- **mimeTypes :- array of all mime types currently supported by the browser.**

Reference	Object
navigator	The browser itself.
window	The main browser window.
window.framename	A frame that occupies the browser window and identified by its assigned name.

window.document	The document appearing in the main browser window.
window.framename.document	The document appearing in a frame identified by its assigned name.
document.getElementById("id")	An XHTML element appearing in a document and identified by its assigned id value.

Examples

```
<html>
<head><title>Browser Information</title></head>
<body>
<h4>Browser Information through Navigator Object</h4><hr>
<p>The <b>navigator</b> object contains the following information
about the browser you are using.</p>
<ul>
<script language="javascript">
document.write("<li><b>Code Name:</b> " + navigator.appCodeName);
document.write("<li><b>App Name:</b> " + navigator.appName);
document.write("<li><b>App Version:</b> " + navigator.appVersion);
document.write("<li><b>User Agent:</b> " + navigator.userAgent);
document.write("<li><b>Language:</b> " + navigator.userLanguage);
document.write("<li><b>Platform:</b> " + navigator.platform);
</script>
<ul><hr>

<h4>Browser Detection</h4>
<script language="Javascript">
// check for 5.0 or later browsers
var browser=null
if (parseInt(navigator.appVersion) >= 5 ||
navigator.appVersion.indexOf("MSIE 5.0") != -1)
{
   browser="DOM";
}
else if (navigator.userAgent.indexOf("Mozilla/4") != -1)
      {
         if (navigator.appName.indexOf("Netscape") != -1)
             browser="NS4";
         if (navigator.appVersion.indexOf("MSIE 7") != -1)
             browser="IE7";
      } else browser="Other";
document.write("Browser Detected: " + browser + "<br>");
</script>
<h4>Plugins Available:  </h4>
<script language="Javascript">
     document.write("<table border=1>");
          for (i=0; i<navigator.plugins.length; i++) {
        document.write("<tr><td>");
        document.write(navigator.plugins[i].name);
        document.write("</td><td>");
        document.write(navigator.plugins[i].filename);
        document.write("</td><td>");
        document.write(navigator.plugins[i].description);
        document.write("</td></tr>");}
        document.write("</table>");
</script></body></html>
```



Browser Information through Navigator Object

The navigator object contains the following information about the browser you are using

- Code Name: Mozilla
- App Name: Netscape
- App Version: 5.0 (Windows; en-US)
- User Agent: Mozilla/5.0 (Windows; U; Windows NT 5.1; en-US; rv:1.9.2.13) Gecko/20101203 Firefox/3.6.13
- Language: undefined
- Platform: Win32

Browser Detection

Browser Detected: DOM

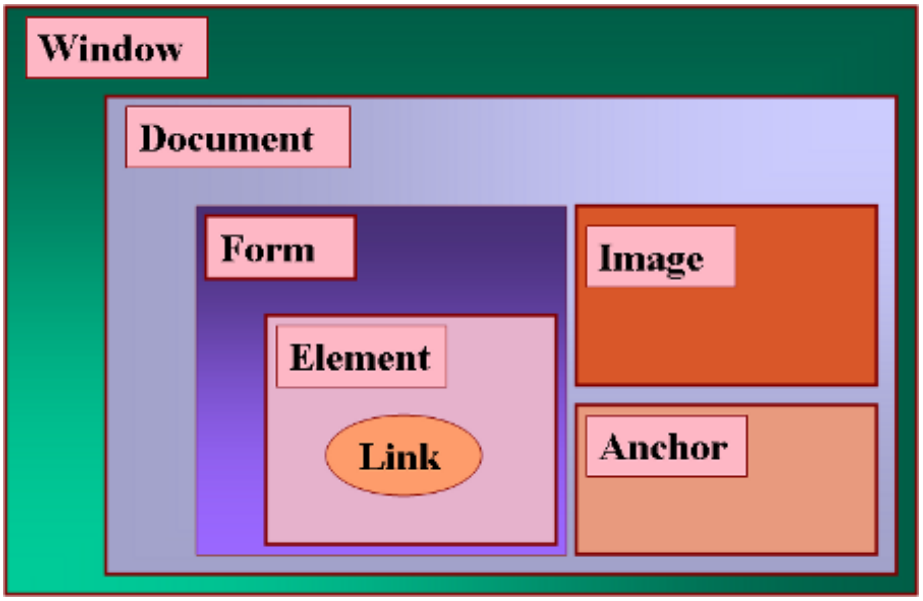
Plugins Available:

Adobe Acrobat	nppdf32.dll	Adobe PDF Plug-In For Firefox and Netscape
RealPlayer Version Plugin	nprpjplug.dll	6.0.12.46
RealPlayer(tm) G2 LiveConnect-Enabled Plug-In (32-bit)	npp13260.dll	RealPlayer(tm) LiveConnect-Enabled Plug-In
RealJukebox NS Plugin	nprjplug.dll	RealJukebox Netscape Plugin
Mozilla Default Plug-in	npnul32.dll	Default Plug-in
Shockwave Flash	NPSWF32.dll	Shockwave Flash 10.2 r152
VLC Multimedia Plug-in	npvlc.dll	Version 1.0.3, copyright 1996-2009 The VideoLAN Team

Window Object

- Window Object
- **Properties and methods with windows**
- **setTimeout and setInterval method**
- Managing History with Window
- Document Object Properties

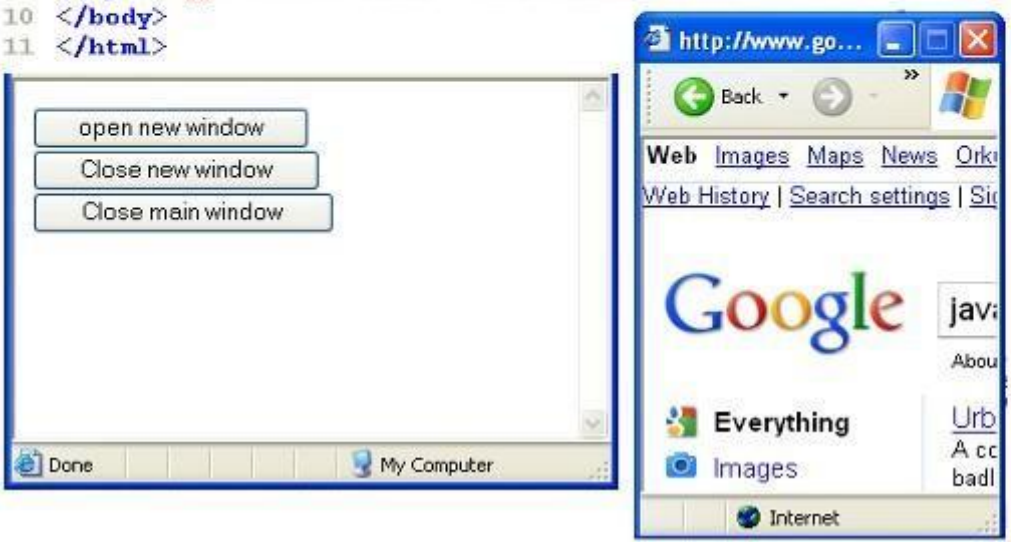
Window Object



Properties and methods with windows

- **Properties**
  - closed
  - defaultstatus
  - status
  - frame
  - location
  - name
  - self, window
  - parent
- **Methods**
  - alert(message)
  - prompt(message, default_text)
  - confirm(message)
  - blur()
  - focus()
  - close()
  - open(url, window name)

```
1  <html>
2  <head><title>Window with Child Window</title>
3  </head>
4  <body>
5  <input type=button Value="open new window"
   onClick="xyz=window.open('http://google.co.in','child','width=200
   height=200,toolbar=1,status=1');">
6  <br>
7  <input type=button Value="Close new window" onClick="xyz.close()">
8  <br>
9  <input type=button Value="Close main window" onClick="window.close()">
10 </body>
11 </html>
```



setTimeout and setInterval method

Window Object - Timeout and Time Interval :-

- **setInterval() methods preforms specified job repeatedly after specified time-interval**
- **setInterval(_alert(—4 seconds over‖)‘,4000)**
  displays alert dialog box after every 4 seconds, for an infinite number of times. This returns interval reference to a variable.
  ref = setInterval(–myfunc()‖, 4000)
- **To terminate clearInterval() method can be used as, clearInterval(ref).**

- **setTimeout() methods performs specified job once only after specified time-interval.**

ref = setTimeout(–myfunc()‖, 4000)
Calls myfunc() after 4 seconds for once only.

```
<html>
<head><title>window Object</title>
    <script language="javascript">
        function f1()
        {
            alert("Display repeatedly...")
        }
        </script>
<head>
<Body>
Welcome to the world of JavaScript
<hr>
This window will close after 5 seconds automatically.....
    <script language="javascript">
        ref = setInterval("f1()", 1000);
    //  clearInterval(ref);
        i=window.setTimeout("window.close()",5000);
    </script>
</body>
<html>
```

Managing History with Window

Managing History:

- **Back()** - **Load the previous URL in the history list.**
- **Forward()** - **Load the next URL in the history list.**
- **Length** - **Length of history list.**
- **Go()** - **Loads an URL, specified by an offset from the current place in the history list.**

Example :-

Window.history.back()
Window.history.go(-1)

Window.history.forward()
Window.history.go(1)

FirstPage.html

```
<html>
<head><title>History Object</title></head>
<body>
<form>
<a href="History_go_to_specific_page.html">Next page</a>
</form>
</body>
</html>
```

History_go_to_specific_page.html

```
<!--  History object go to specific page -->
<html>
<head>
    <script type="text/javascript">
    function goBack(){
        window.history.go(-1);
    }
    </script>
</head>
<body>
<form>
    <input type="button" value="Back" onclick="goBack()" />
</form>
</body>
</html>
```

Document Object Properties

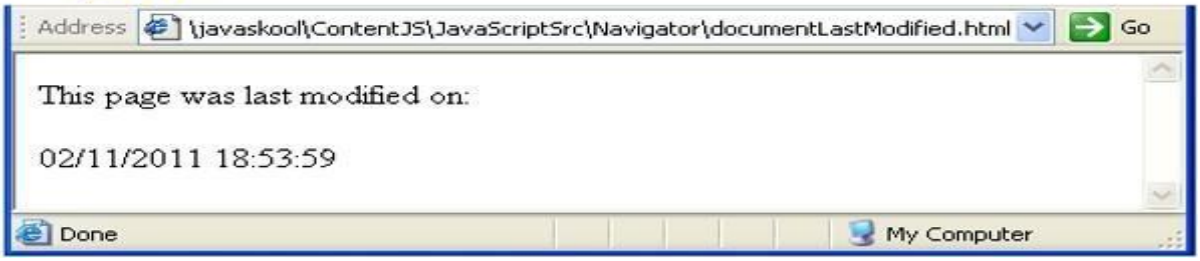
Document Object Properties :-

- **Forms[] :- An array of form objects, one for each <FORM> and </FORM> in the document. The number of forms in the document can be determined using the statement forms.length.**
- **Images[] :–An array of image objects, one for each embedded image. The number of images can be determined using images.length statement.**
- **links[] :–An array of link objects, one for each hypertext link (<A HREF>) in the document. The number of link objects can be determined using links.length statement.**

Other Properties :

- **lastModified :- A string contains date of the most recent changes made to the web document.**
- **referrer :–A string (read-only) that contains the URL that contained the link to the present document.**
- **title :–A string (read-only) that contains the title of the document.**
- **URL :–The URL of the document**
- **domain :–The domain name of the server from which the document is loaded**

```
1  <html>
2  <head><title>Document Object</title></head>
3  <body>
4  <p>This page was last modified on:</p>
5  <script language="javascript">
6      document.write(document.lastModified);
7  </script>
8  </body>
9  </html>
```



Location Object

- Location Object
- **Example**

Location Object

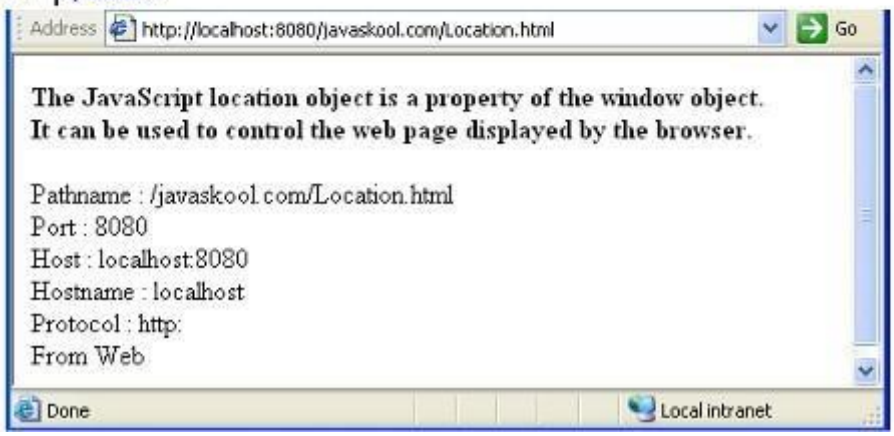
- **The following statements display the host name, the protocol name and the pathname of the URL displayed in the window,**

document.write(–File Path :‖ + location.pathname)
document.write(–Protocol Used :‖ + location.protocol)
document.write(–Host Name :‖ + location.hostname)

window.location.href = "http://www.javaskool.com/contact.html"
or
location.href = –http://www.javaskool.com/contact.html –

Examples

```
<html>
<head><title>Location</title></head>
<body>
<h1>The JavaScript location object is a property of the window object.
<br>It can be used to control the web page displayed by the browser. </h1>

    <script language="javascript">
        document.write('Pathname : "+location.pathname +'<BR>');
        document.write('Port     : "+location.port +"<BR>");
        document.write('Host     : "+location.host +"<BR>");
        document.write('Hostname : "+location.hostname +'<BR>');
        document.write('Protocol : "+location.protocol +'<BR>');

        switch (window.location.protocol)
        {
           case "http:":
                  document.write("From Web<BR>\n')
                  break
           case "file:":
                  document.write("From Local computer<BR>\n')
                  break
           default:
                  document.write("Unknown Source<BR>\n")
                  break
        }
//location.href = "http://google.co.in" ;
//document.write('href : "+location.href +'<BR>");
    </script>
</body>
</html>
```



What is JavaScript?

Javascript is a dynamic computer programming language. It is lightweight and most commonly used as a part of web pages, whose implementations allow client-side script to interact with the user and make dynamic pages.

It is an interpreted programming language with object-oriented capabilities.

JavaScript was first known as LiveScript, but Netscape changed itsnameto JavaScript, possibly because of the excitement being generated by Java. JavaScript made its first appearance in Netscape 2.0 in 1995 with the name LiveScript.

The general-purpose core of the language has been embedded in Netscape, Internet Explorer, and oth er web browsers.The
ECMA-262 Specification defined a standard version of the core JavaScript language.

- JavaScript is a lightweight, interpreted programming language.
- Designed for creating network-centric applications.
- Complementary to and integrated with Java.
- Complementary to and integrated with HTML.
- Open and cross-platform. Client-Side JavaScript

Client-side JavaScript is the most common form of the language. The script should be included in or referenced by an HTML document for the code to be interpreted by the browser.It means that a web page need not be a static HTML, but can include programs that interact with the user, control the browser, and dynamically create HTML content.
The JavaScript client-side mechanism provides many advantages over traditional CGI server-side scripts. For example, you might use JavaScript to check if the user has entered a valid e-mail address in a form field.The JavaScript code is executed when the user submits the form, and only if all the entries are valid,they would be submitted to the Web Server.JavaScript can be used to trap user- initiated events such as button clicks, link navigation, and other actions that the user initiates explicitly or implicitly
Advantages of JavaScript
The merits of using JavaScript are:

- Less server interaction:You can validate user input before sending the page off to the server. This saves server traffic, which means less load on your server.
- Immediate feedback to the visitors:They don't have to wait for a page reload to see if they have forgotten to enter something.
- Increased interactivity:You can create interfaces that react when the user hovers over them with a mouse or activates them via the keyboard.
- Richer interfaces:You can use JavaScript to include such items as drag- and-drop components and sliders to give a Rich Interface to your site visitors.

Limitations of JavaScript

- We cannot 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 cannot be used for networking applications because there is no such support available.
- JavaScript doesn't have any multithreading or multiprocessor capabilities.Once again, JavaScript is a lightweight, interpreted programming language that allows you to build interactivity into otherwise static HTMLpages.

JavaScript Development
Tools One of major strengths of JavaScript is that it does not require expensive development tools. 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.
Some 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 interactive websites.
- 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
- Javascript components, integrates well with databases, and conforms to new standards such as XHTML and XML.

- Macromedia HomeSite 5:HomeSite 5 is a well-liked HTML and JavaScript editor from Macromedia that can be used to manage personal websiteseffectively.

Where is JavaScript Today?

The ECMAScript Edition 5 standard will be the first update to be released in over four years. JavaScript 2.0 conforms to Edition 5 of the ECMAScript standard, and the difference between the two is extremely minor.The specification for JavaScript 2.0 can be found on the following site: http://www.ecmascript.org/Today, Netscape's JavaScript and Microsoft's JScript conform to the ECMAScript standard, although both thelanguages still supportthe features that are not a part of the 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 at a later point in time when you need that variable.

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 at 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 isuntypedlanguage. 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. Thevalue type of a variable can change during the execution of a program and JavaScript takes care of it automatically.

JavaScript Variable ScopeThe scope of a variable is the region of your program in which it is defined. JavaScript variables have only two scopes.

- Global Variables:A global variable has global scope which means it can be defined Any wherein 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.Javascript 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

. Take a look into the following example.

```
<script type="text/javascript">
<!--var myVar = "global"; // Declare a global variable
function checkscope( ) {
var myVar = "local"; // Declare a local variable
document.write(myVar);
}//--></script>
```

It will produce the following result: Local

JavaScript Variable Names While naming your variables in JavaScript, keep the following rules in mind.

- You should not use any of the JavaScript reserved keywords as a 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 an underscore character. For example,123test is an invalid variable name but_123test is a validone.
- JavaScript variable names are case-sensitive. For example, Name and nameare twodifferent variables. Javascript

What is an Event?

JavaScript's interaction with HTML is handled through events that occur when the user or the browser manipulates a page.When the page loads, it is called an event. When the user clicks a button, that click too is an event.

OtherExamples include events like pressing any key, closing a window, resizing a 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.

Events are a part of the Document Object Model (DOM) Level 3 and every HTML element containsa set of events which can trigger JavaScript Code.Here we will see a few examples to understand the relation between Event and JavaScript. onclick Event TypeThis is the most frequently used event type which occurs when a user clicks the left buttonof his mouse. You can put your validation, warning etc.,against this event type.

Example

Try the following example.

```
<html>
<head>
<script type="text
/javascript">
<!--
function sayHello() { document.write
("Hello World")
}
//
-->
```

XML

XML stands for **E**xtensible **M**arkup **L**anguage. It is a text-based markup language derived from Standard Generalized Markup Language (SGML).

XML tags identify the data and are used to store and organize the data, rather than specifying how to display it like HTML tags, which are used to display the data. XML is not going to replace HTML in the near future, but it introduces new possibilities by adopting many successful features of HTML.

There are three important characteristics of XML that make it useful in a variety of systems andsolutions:

- **XML is extensible:** XML allows you to create your own self-descriptive tags, or language, that suits your application.
- **XML carries the data, does not present it:** XML allows youtostorethedatairrespectiveofhowitwillbepresented.
- **XML is a public standard:** XML was developed by an organization called the World Wide Web Consortium (W3C) and is available as an openstandard.

XMLUsage

A short list of XML usage says it all:

- XML can work behind the scene to simplify the creation of HTML documents for large websites.
- XML can be used to exchange the information between organizations and systems.
- XML can be used for offloading and reloading of databases.
- XML can be used to store and arrange the data, which can customize your data handling needs.
- XML can easily be merged with style sheets to create almost any desired output.
- Virtually, any type of data can be expressed as an XML document.

What is Markup?

XML is a markup language that defines set of rules for encoding documents in a format that is both human-readable and machine-readable. So what exactly is a markup language? Markup is information added to a document that enhances its meaning in certain ways, in that it identifies the parts and how they relate to each other. More specifically, a markup language is a set of symbols that can be placed in the text of a document to demarcate and label the parts of that document.

Following example shows how XML markup looks, when embedded in a piece of text:

```
<message>
<text>Hello, world!</text>
</message>
```

This snippet includes the markup symbols, or the tags such as <message>...</message> and <text>...</text>. The tags <message> and </message> mark the start and the end of the XML code fragment. The tags <text> and </text> surround the text Hello, world!.

Is XML a Programming Language?

A programming language consists of grammar rules and its own vocabulary which is used to create computer programs. These programs instructs computer to perform specific tasks. perform any computation or algorithms. It is usually stored in a simple text file and is processed by special software that is capable of interpreting XML.

Tags and Elements

An XML file is structured by several XML-elements, also called XML-nodes or XML- tags. XML-elements' names are enclosed by triangular brackets <> as shown below:

```
<element>
```

Syntax Rules for Tags and Elements

Element Syntax: Each XML-element needs to be closed either with start or with end elements as shown below:

```
<element>... </element>
```

or in simple-cases, just this way:

```
<element/>
```

Nesting of elements: An XML-element can contain multiple XML-elements as its children, but the children elements must not overlap. i.e., an end tag of an element must have the same name as that of the most recent unmatched start tag.

Following example shows incorrect nested tags:

```
<?xml version="1.0"?>
<contact-info>
<company>IARE
<contact-info>
</company>
```

Following example shows correct nested tags:

```
<?xml version="1.0"?>
<contact-info>
<company>IARE</company>
<contact-info>
```

Let us learn about one of the most important part of XML, the XML tags. XML tags form the foundation of XML. They define the scope of an element in theXML. They can also be used to insert comments, declare settings required for parsing the environment and to insert special instructions.We can broadly categorize XML tags as follows:

StartTag

EndTag

</address>

The beginning of every non-empty XML element is marked by a start-tag. An example of start-tag is:

<address>

Every element that has a start tag should end with an end-tag. An example of end- tag is:

Note that the end tags include a solidus ("/") before the name of an element.

EmptyTag

The text that appears between start-tag and end-tag is called content. An element which has no content is termed as **empty**. An **empty** element can be represented in two ways as below:

(1) A start-tag immediately followed by an end-tag as shownbelow:

```
<hr></hr>
```

(2) A complete empty-element tag is as shownbelow:

```
<hr />
```

Empty-element tags may be used for any element which has no content.

XML Tags Rules

Following are the rules that need to be followed to use XML tags:

Rule 1

XML tags are case-sensitive.Followinglineof code is an example of wrongsyntax</Address>,becauseofthecasedifferenceintwotags,whichistreatedaserroneoussyntaxin XML.

<address>This is wrong syntax</Address>

Following code shows a correct way, where we use the same case to name the start and theend tag.

<address>This is correct syntax</address>

Rule 2

XML tags must be closed in an appropriate order, i.e., an XML tag opened inside another element must be closed before the outer element is closed. For example:

```
<outer_element>
<internal_element>
    This tag is closed before the outer_element
</internal_element>
</outer_element>
```

XML Elements

XML elements can be defined as building blocks of an XML. Elements can behave as

containers to hold text, elements, attributes, media objects or all of these.

Each XML document contains one or more elements, the scope of which are either delimited by start and end tags, or for empty elements, by an empty- element tag.

Syntax

Following is the syntax to write an XML element:

```
<element-name attribute1 attribute2>
....content
</element-name>
```

where

- **element-name** is the name of the element. The name its case in the start andend tags mustmatch.
- **attribute1, attribute2** are attributes of the element separated by white spaces. An attribute defines a property of the element. It associates a name with a value, which is a string of characters. An attribute is writtenas:

  name = "value"

The name is followed by an = sign and a string value inside double(" ") or single(' ') quotes.

Empty Element

An empty element (element with no content) has following syntax:

```
<name attribute1 attribute2.../>
```

Example of an XML document using various XML element:

```
<?xml version="1.0"?>
<contact-info>
<address category="residence">
<name>Tanmay Patil</name>
<company>TutorialsPoint</company>
<phone>(011) 123-4567</phone>
<address/>
</contact-info>
```

XML Elements Rules

Following rules are required to be followed for XML elements:

- An element name can contain any alphanumeric characters. The only punctuationmarks allowed in names are the hyphen (-), under-score (_) and period(.).
- Names are case sensitive. For example, Address, address, and ADDRESS are differentnames.
- Start and end tags of an element must beidentical.
- An element, which is a container, can contain text or elements as seen in the above example.

Root element: An XML document can have only one root element. For example, following is not a correct XML document, because both the x and y elements occur at the top level without a root element:

```
<x>...</x>
<y>...</y>
```

The following example shows a correctly formed XML document:

```
<root>
   <x>...</x>
   <y>...</y>
</root>
```

Case sensitivity: The names of XML-elements are case-sensitive. That means the name of the start and the end elements need to be exactly in the same case.

For example, **<contact-info>**is different from **<Contact-Info>**.

XML Document Type Declaration, commonly known as DTD, is a way to describe XML language precisely. DTDs check vocabulary and validity of the structure of XML documents against grammatical rules of appropriate XMLlanguage.

An XML DTD can be either specified inside the document, or it can be kept in a separate document and then liked separately.

Syntax

Basic syntax of a DTD is as follows:

<!DOCTYPE element DTD identifier

- **DTD identifier** is an identifier for the document type definition, which may be the path to a file on the system or URL to a file on the internet. If the DTD is pointing to external path, it is called **ExternalSubset.**
- **The square brackets [ ]** enclose an optional list of entity declarations called InternalSubset.

InternalDTD

A DTD is referred to as an internal DTD if elements are declared within the XML files. To refer it as internal DTD, standalone attribute in XML declaration must be set to **yes**. This means, the declaration works independent of external source.

Syntax

The syntax of internal DTD is as shown:

<!DOCTYPE root-element [element-declarations]>

where root-element is the name of root element and element-declarations is where you declare the elements.Example

Following is a simple example of internal DTD:

```
<?xml version="1.0" encoding="UTF-8" standalone="yes" ?>
<!DOCTYPE address [
<!ELEMENT address (name,company,phone)>
<!ELEMENT name (#PCDATA)>
<!ELEMENT company (#PCDATA)>
<!ELEMENT phone (#PCDATA)>
]>
<address>
<name>Tanmay Patil</name>
<company>iare</company>
<phone>(011) 123-4567</phone>
    </address>
```

Let us go through the above code:

<?xml version="1.0" encoding="UTF-8" standalone="yes" ?>

Start Declaration- Begin the XML declaration with following statement

DTD- Immediately after the XML header, the document type declaration follows, commonly referred to as theDOCTYPE:

<!DOCTYPE address [

The DOCTYPE declaration has an exclamation mark (!) at the start of the element name. The DOCTYPE informs the parser that a DTD is associated with this XML document.

DTD Body- The DOCTYPE declaration is followed by body of the DTD, where you declare elements, attributes, entities, andnotations:

```
<!ELEMENT address (name,company,phone)>
<!ELEMENT name (#PCDATA)>
<!ELEMENT company (#PCDATA)>
<!ELEMENT phone_no (#PCDATA)>
```

Several elements are declared here that make up the vocabulary of the <name> document.<!ELEMENT

name (#PCDATA)> defines the element name to be of type "#PCDATA". Here #PCDATA means parse-able text data.

EndDeclaration-Finally,thedeclarationsectionoftheDTDisclosedusingaclosingbracket and a closing angle bracket (]>). This effectively ends the definition, and thereafter, the XML document followsimmediately.

Rules

- The document type declaration must appear at the start of the document (preceded onlybytheXMLheader)—itisnotpermittedanywhereelsewithin thedocument.
    - Similar to the DOCTYPE declaration, the element declarations must start with an exclamationmark.
    - The Name in the document type declaration must match the element type of the root element.

External DTD

In external DTD elements are declared outside the XML file. They are accessed by specifyingthesystemattributeswhichmaybeeitherthelegal.dtdfileoravalidURL.Torefer it as external DTD, standalone attribute in the XML declaration must be set as **no**. This means, declaration includes information from the externalsource.

Syntax

Following is the syntax for external DTD:

<!DOCTYPE root-element SYSTEM "file-name"> where

file-name is the file with .dtd extension.

Example

The following example shows external DTD usage:

```
<?xml version="1.0" encoding="UTF-8" standalone="no" ?>
<!DOCTYPE address SYSTEM "address.dtd">
<address>
<name>Tanmay Patil</name>
<company>IARE</company>
<phone>(011) 123-4567</phone>
</address>
```

The content of the DTD file **address.dtd** are as shown:

```
<!ELEMENT address (name,company,phone)>
<!ELEMENT name (#PCDATA)>
<!ELEMENT company (#PCDATA)>
    <!ELEMENTphone(#PCDATA)>
```

Types

You can refer to an external DTD by using either **system identifiers** or **public identifiers**.

SYSTEM IDENTIFIERS

A system identifier enables you to specify the location of an external file containing DTD declarations. Syntax is as follows:

```
<!DOCTYPE name SYSTEM "address.dtd" [...]>
```

As you can see, it contains keyword SYSTEM and a URI reference pointing to the location of the document.

PUBLIC IDENTIFIERS

Public identifiers provide a mechanism to locate DTD resources and are written as below:

```
<!DOCTYPE name PUBLIC "-//Beginning XML//DTD AddressExample//EN">
```

As you can see, it begins with keyword PUBLIC, followed by a specialized identifier. Public identifiers are used to identifyan entry in a catalog. Public identifiers can follow any format, however, a commonly used format is called Formal Public Identifiers, orFPIs.

XML Schema is commonly known as XML Schema Definition (XSD). It is used to describe and validate the structure and the content of XML data. XML schema defines the elements, attributes and data types. Schema element supports Namespaces. It is similar to a database schema that describes the data in a database.

Syntax

You need to declare a schema in your XML document as follows:

<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"> Example

The following example shows how to use schema:

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
<xs:element name="contact">
<xs:complexType>
<xs:sequence>
<xs:element name="name" type="xs:string" />
<xs:element name="company" type="xs:string" />
<xs:element name="phone" type="xs:int" />
</xs:sequence>
</xs:complexType>
</xs:element>
</xs:schema>
```

The basic idea behind XML Schemas is that they describe the legitimate format that an XML document can take.

Elements

As we saw in the chapter XML - Elements, elements are the building blocks of XML document. An element can be defined within an XSD as follows:

```
<xs:element name="x" type="y"/>
```

Definition Types

You can define XML schema elements in following ways:

```
<xs:element name="phone_number" type="xs:int" />
```

Simple Type - Simple type element is used only in the context of the text. Some of predefined simple types are: xs:integer, xs:boolean, xs:string, xs:date. For example:

Complex Type - A complex type is a container for other element definitions. This allows you to specify which child elements an element can contain and to provide some structure within your XML documents. For example:

```
<xs:element name="Address">
<xs:complexType>
<xs:sequence>
<xs:element name="name" type="xs:string" />
<xs:element name="company" type="xs:string" />
<xs:element name="phone" type="xs:int" />
</xs:sequence>
</xs:complexType>
</xs:element>
```

In the above example, Address element consists of child elements. This is a container for other <xs:element> definitions, that allows to build a simple hierarchy of elements in the XML document.

Global Types - With global type, you can define a single type in your document, which can be used by all other references. For example, suppose you want to generalize the person and company for different addresses of the company. In such case, you can define a general type as below:

```
<xs:element name="AddressType">
<xs:complexType>
<xs:sequence>
<xs:element name="name" type="xs:string" />
```

```
<xs:element name="company" type="xs:string" />
</xs:sequence>
</xs:complexType>
</xs:element>

<xs:element name="Address1">
<xs:complexType>
<xs:sequence>
<xs:element name="address" type="AddressType" />
<xs:element name="phone1" type="xs:int" />
</xs:sequence>
</xs:complexType>
</xs:element>
<xs:element name="Address2">
<xs:complexType>
<xs:sequence>
<xs:element name="address" type="AddressType" />
<xs:element name="phone2" type="xs:int" />
</xs:sequence>
</xs:complexType>
    </xs:element>
```

Instead of having to define the name and the company twice (once for Address1 and once for Address2), we now have a single definition. This makes maintenance simpler, i.e., if you decide to add "Postcode" elements to the address, you need to add them at just one place.

Attributes

Attributes in XSD provide extra information within an element. Attributes have name and type property as shown below:

```
<xs:attribute name="x" type="y"/>
```

UNIT-III
Servlets and JSP

Servlet: Lifecycle of a Servlet, a simple Servlet, the servlet API, the Javax.servlet package, reading Servlet parameters, the javax.servlet. HTTP package, Handling HTTP requests and responses, using cookies and sessions.

JSP: The anatomy of a JSP page, JSP processing, declarations, directives, expressions, code snippets, implicit objects, using beans in JSP pages, connecting to database in JSP.

Servlets:

Java Servlets are programs that run on a Web or Application server and act as a middle layer between a request coming from a Web browser or other HTTP client and databases or applications on the HTTP server.

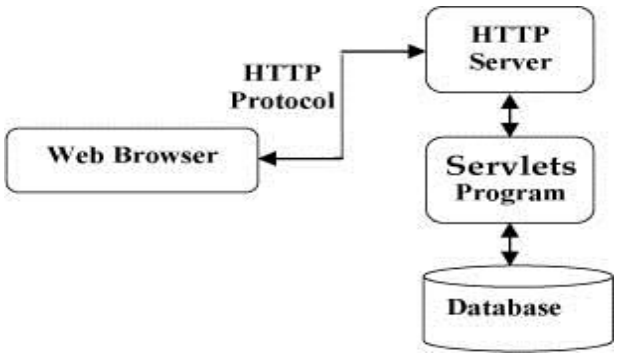
Using Servlets, you can collect input from users through web page forms, present records from a database or another source, and create web pages dynamically.

Java Servlets often serve the same purpose as programs implemented using the Common Gateway Interface (CGI). But Servlets offer several advantages in comparison with the CGI.

- Performance is significantlybetter.
- Servlets execute within the address space of a Web server. It is not necessary to create a separate process to handle each clientrequest.
- Servlets are platform-independent because they are written inJava.
- Java security manager on the server enforces a set of restrictions to protect the resources on aserver machine. So servlets aretrusted.
- The full functionality of the Java class libraries is available to a servlet. It can communicate with applets, databases, or other software via the sockets and RMI mechanisms that you have seenalready.

Servlets Architecture:

Following diagram shows the position of Servelts in a Web Application.



Servlets Tasks:

Servlets perform the following major tasks:

- Read the explicit data sent by the clients (browsers). This includes an HTML form on a Web page or it could also come from an applet or a custom HTTP clientprogram.
- Read the implicit HTTP request data sent by the clients (browsers). This includes cookies, media types and compression schemes the browser understands, and soforth.
- Process the data and generate the results. This process may require talking to a database, executing an RMI or CORBA call, invoking a Web service, or computing the response directly.
- Send the explicit data (i.e., the document) to the clients (browsers). This document can be sent in a variety of formats, including text (HTML or XML), binary (GIF images), Excel, etc.

- Send the implicit HTTP response to the clients (browsers). This includes telling the browsers or other clients what type of document is being returned (e.g., HTML), setting cookies and caching parameters, and other suchtasks.

Servlets Packages:

Java Servlets are Java classes run by a web server that has an interpreter that supports the Java Servlet specification.

Servlets can be created using the **javax.servlet** and **javax.servlet.http** packages, which are a standard part of the Java's enterprise edition, an expanded version of the Java class library that supports large-scale development projects.

These classes implement the Java Servlet and JSP specifications. At the time of writing this tutorial, the versions are Java Servlet 2.5 and JSP 2.1.

Java servlets have been created and compiled just like any other Java class. After you install the servlet packages and add them to your computer's Classpath, you can compile servlets with the JDK's Java compiler or any other current compiler.

Life cycle ofservlet

A Servlet life cycle can be defined as the entire process from its creation till the destruction. The following are the paths followed by a servlet

The servlet is initialized by calling the **init ()** method.

The servlet calls **service()** method to process a client's request. The servlet is terminated by calling the **destroy()** method.

Finally, servlet is garbage collected by the garbage collector of the JVM. Now let us discuss the life cycle methods in details.

The init() method:

The init method is designed to be called only once. It is called when the servlet is first created,and notcalledagainforeachuserrequest.So,itisusedforone-timeinitializations,justaswiththeinit method ofapplets.

The servlet is normally created when a user first invokes a URL corresponding to the servlet, but you can also specify that the servlet be loaded when the server is first started.

When a user invokes a servlet, a single instance of each servlet gets created, with each user request resulting in a new thread that is handed off to doGet or doPost as appropriate. The init() method simply creates or loads some data that will be used throughout the life of the servlet.

The init method definition looks like this:

```
publicvoid init()throwsServletException{
// Initialization code...
}
```

The service() method:

The service() method is the main method to perform the actual task. The servlet container (i.e. web server) calls the service() method to handle requests coming from the client(browsers) and to write the formatted response back to the client.

Each time the server receives a request for a servlet, the server spawns a new thread and calls service. The service() method checks the HTTP request type (GET, POST, PUT, DELETE, etc.) and calls doGet, doPost, doPut, doDelete, etc. methods as appropriate.

Here is the signature of this method:

```
publicvoid service(ServletRequest request, ServletResponse response) throwsServletException,IOException{
}
```

The service () method is called by the container and service method invokes doGe, doPost, doPut, doDelete, etc. methods as appropriate. So you have nothing to do with service() method but you override either doGet() or doPost() depending on what type of request you receive from the client.

The doGet() and doPost() are most frequently used methods with in each service request. Here is the signature of these two methods.

The doGet() Method

A GET request results from a normal request for a URL or from an HTML form that has no METHOD specified and it should be handled by doGet() method.

```
publicvoid doGet(HttpServletRequest request, HttpServletResponse response)
throwsServletException,IOException{
// Servlet code
}
```

The doPost() Method

A POST request results from an HTML form that specifically lists POST as the METHOD and it should be handled by doPost() method.

```
publicvoid doPost(HttpServletRequest request, HttpServletResponse response)
throwsServletException,IOException{
// Servlet code
}
```

The destroy() method:

The destroy() method is called only once at the end of the life cycle of a servlet. This method gives your servlet a chance to close database connections, halt background threads, write cookie lists or hit counts to disk, and perform other such cleanup activities.

After the destroy() method is called, the servlet object is marked for garbage collection. The destroy method definition looks like this:

```
publicvoid destroy(){
// Finalization code...
}
```

Servlet Deployment:

By default, a servlet application is located at the path <Tomcat-installation- directory>/webapps/ROOT and the class file would reside in <Tomcat-installation- directory>/webapps/ROOT/WEB-INF/classes.

If you have a fully qualified class name of **com.myorg.MyServlet**, then this servlet class must be located in WEB- INF/classes/com/myorg/MyServlet.class.

For now, let us copy HelloWorld.class into <Tomcat-installation-directory>/webapps/ROOT/WEB-INF/classes and create following entries in **web.xml** file located in <Tomcat-installation-directory>/webapps/ROOT/WEB-INF/

```
<servlet>
<servlet-name>HelloWorld</servlet-name>
<servlet-class>HelloWorld</servlet-class>
</servlet>
<servlet-mapping>
<servlet-name>HelloWorld</servlet-name>
<url-pattern>/HelloWorld</url-pattern>
</servlet-mapping>
```

Above entries to be created inside <web-app>...</web-app> tags available in web.xml file. There could be

various entries in this table already available, but never mind.

You are almost done, now let us start tomcat server using <Tomcat-installation- directory>\bin\startup.bat (on windows) or <Tomcat-installation-directory>/bin/startup.sh (on Linux/Solaris etc.) and finally type **http://localhost:8080/HelloWorld** in browser's address box.

JSP Overview

Java Server Pages (JSP) is a technology for developing web pages that support dynamic content which helps developers insert java code in HTML pages by making use of special JSP tags, most of which start with <% and end with %>.

A JavaServer Pages component is a type of Java servlet that is designed to fulfill the role of a user interface for a

Java web application. Web developers write JSPs as text files that combine HTML or XHTML code, XML elements, and embedded JSP actions and commands.

Using JSP, you can collect input from users through web page forms, present records from a database or another source, and create web pages dynamically.

JSP tags can be used for a variety of purposes, such as retrieving information from a database or registering user preferences, accessing JavaBeans components, passing control between pages and sharing information between requests, pages etc.

Why Use JSP?

JavaServer Pages often serve the same purpose as programs implemented using the Common Gateway Interface (CGI). But JSP offer several advantages in comparison with the CGI.

- Performance is significantly better because JSP allows embedding Dynamic Elements in HTML Pages itself instead of having a separate CGI files.
- JSP are always compiled before it's processed by the server unlike CGI/Perl which requires the server to load an interpreter and the target script each time the page is requested.
- JavaServer Pages are built on top of the Java Servlets API, so like Servlets, JSP also has access to all the powerful Enterprise Java APIs, including JDBC, JNDI, EJB, JAXP etc.
- JSP pages can be used in combination with servlets that handle the business logic, the model supported by Java servlet template engines.

Finally, JSP is an integral part of J2EE, a complete platform for enterprise class applications. This means that JSP can play a part in the simplest applications to the most complex and demanding.C **Advantages of JSP:**

Following is the list of other advantages of using JSP over other technologies:

- vs. Active Server Pages (ASP):The advantages of JSP are twofold. First, the dynamic part is written in Java, not Visual Basic or other MS specific language, so it is more powerful and easier to use.
- Second, it is portable to other operating systems and non-Microsoft Web servers.
- vs. Pure Servlets:It is more convenient to write (and to modify!) regular HTML than to have plenty of println statements that generate the HTML.
- vs. Server-Side Includes (SSI):SSI is really only intended for simple inclusions, not for "real" programs that use form data, make database connections, and the like.
- vs. JavaScript:JavaScript can generate HTML dynamically on the client but can hardly interact with the web server to perform complex tasks like database access and image processing etc. vs. Static HTML:Regular HTML, of course, cannot contain dynamic information

JSP Declarations:

A declaration declares one or more variables or methods that you can use in Java code later in the JSP file. You must declare the variable or method before you use it in the JSP file. Following is the syntax of JSP Declarations:

<%!declaration;[declaration;]+...%>

You can write XML equivalent of the above syntax as follows:

<jsp:declaration> code fragment

</jsp:declaration>

Following is the simple example for JSP Declarations:

<%!int i =0;%>

<%!int a,b,c;%>
<%!Circle a =newCircle(2.0);%>
<\%Represents static <% literal.%\> Represents static %>
literal.\'A single quote in an attribute that uses single quotes.\"A double quote in an attribute that uses double quotes.

JSP Directives:

A JSP directive affects the overall structure of the servlet class. It usually has the following form:
<%@directive attribute="value"%> There are three types of directive tag: Directive
Description
<%@ page ... %> Defines page
-
dependent attributes, such as scripting language, error page, and buffering requirements.
<%@ include ... %>
Includes a file during the translation phase.
<%@ tag lib ... %>
Declares a tag library, containing custom actions, used in the page The <jsp:forward> Action
The forward action terminates the action of the current page and forwards the request to another resource such as a static page, another JSP page, or a Java Servlet.
The simple syntax of this action is as follows:
<jsp:forward page="Relative URL"/>
Following is the list of required attributes associated with forward action: Attribute
Description page
Should consist of a relative URL of another resource such as a static page, another JSP page, or a Java Servlet
Example:
Let us reuse following two files (a) date.jps and (b) main.jsp as follows:
Following is the content of date.jsp file:

```
<p>
Today's date:
<%=(new java.util.Date()).toLocaleString()%>
</p>
```

Here is the content of main.jsp file:

```
<html>
<head>
<title>
The include Action Example
</title>
</head>
<body><center>
<h2>
The include action Example
</h2>
<jsp:forward page="date.jsp"/>
</center>
</body>
</html>
```

Now let us keep all these files in root directory and try to access main.jsp. This would display r esult something like as below. Here it discarded content from main page and displayed contentfrom forwarded pageonly.
Today's date: 12-Sep-2010 14:54:22 The <jsp:plugin> Action
The plugin action is used to insert Java components into a JSP page. It dete rmines the type of browser and inserts the <object> or <embed> tags as needed.If the needed plugin is not present, it downloads the plugin and then executes the Java component. The Java component can be either an Applet or a JavaBean.
The plugin action has several attributes that correspond to common HTML tags used to format Java

components.
The <param> element can also be used to send parameters to the Applet or Bean. Following is the typical syntax of using plugin action:

```
<jsp:plugin type="applet"code base="dirname"code="MyApplet.class" width="60",height="80">
<jsp:paramname="fontcolor" value="red"/>
<jsp:param name="background"value="black"/>
<jsp:fallback>
Unable to initialize Java Plugin
</jsp:fallback>
</jsp:plugin>
```

You can try this action using some applet if you are interested. A new element, the <fallback> element, can be used to specify an error string to be sent to the user in case the component fails. The <jsp:element> Action
The <jsp:attribute> Action The <jsp:body> Action
The <jsp:element>, lt;jsp:attribute> and <jsp:body> actions are used to define XML elements dynamically. The word dynamically is important, because it means that the XML elements can be generated at request time rather than statically at compile time.
Following is a simple example to define XML elements dynamically:

```
<%@page language ="java" contentType="text/html"%>
<html xmlns=http://www.w3c.org/1999/xhtml xmlns:jsp="http://java.sun.com/JSP/Page">
<head><title>
Genera
te XML Element
</title></head>
<body>
<jsp:element name="xmlElement">
<jsp:attribute name="xmlElementAttr"> Value for the attribute
</jsp:attribute>
<jsp:body>
Body for XML element
</jsp:body>
</jsp:element>
</body>
</html>
```

This would produce followin g HTML code at run time:

```
<html xmlns=http://www.w3c.org/1999/xhtml xmlns:jsp="http://java.sun.com/JSP/Page">
<head><title>
Generate XML Element
</title></head>
<body>
<xmlElement xmlElementAttr= "Value for the attribute"> Body for XML element
</xmlElement>
</body>
</html>
```

The <jsp:text> Action
The <jsp:text> action can be used to write template text in JSP pages and documents. Following is the simple syntax for this action:

```
<jsp:text> Template data
</jsp:text>
```

The body fo the template cannot contain other elements; it can only contain text and EL expressions (Note: EL expressions are explained in subsequent chapter). Note that in XML files, you cannot use expressions such as ${whatever > 0}, because the greater than signs are illegal. Instead, use the gt form, such as ${whatever gt 0} or an alternative is to embed the value in a CDATA section.

UNIT-IV

Introduction to PHP

Basics of PHP, downloading, installing, configuring PHP, programming in a web environment and the anatomy of a PHP page; Overview of PHP data types and concepts: Variables and data types, operators, expressions and statements, strings, arrays and functions.

PHP

The PHP Hypertext Preprocessor (PHP) is a programming language that allows web developers to create dynamic content that interacts with databases. PHP is basically used for developing web based software applications.

- PHP is a recursive acronym for "PHP: HypertextPreprocessor".
- PHP is a server side scripting language that is embedded in HTML. It is used to manage dynamic content, databases, session tracking, even build entire e-commercesites.

Common usesof PHP

- PHP performs system functions, i.e. from files on a system it can create, open, read, write, and close them.
- PHP can handle forms, i.e. gather data from files, save data to a file, thru email you can send data, return data to theuser.
- You add, delete, modify elements within your database thruPHP.
- Access cookies variables and setcookies.
- Using PHP, you can restrict users to access some pages of yourwebsite.
- It can encryptdata.

PHP Installation

What Do I Need?

To start using PHP, you can:

- Find a web host with PHP and MySQL support
- Install a web server on your own PC, and then install PHP and MySQL

Use a Web Host With PHP Support

If your server has activated support for PHP you do not need to do anything.

Just create some .php files, place them in your web directory, and the server will automatically parse them for you.

You do not need to compile anything or install any extra tools.

Because PHP is free, most web hosts offer PHP support.

Set Up PHP on Your Own PC

However, if your server does not support PHP, you must:

- install a web server
- install PHP
- install a database, such as MySQL

The official PHP website (PHP.net) has installation instructions for PHP: http://php.net/manual/en/install.php

PHP Installation on Windows NT/2000/XP with Apache

To install Apache with PHP 5 on Windows follow the following steps. If your PHP and Apache versions are different, then please take care accordingly.

Download Apache server from www.apache.org/dist/httpd/binaries/win32. You want the current stable release version with the no_src.msi extension. Double-click the installer file to install; C:\Program Files is a common location. The installer will also ask you whether you want to run Apache as a service or from the command line or DOS prompt. We recommend you do not install as a service, as this may cause problems with startup.

Extract the PHP binary archive using your unzip utility; C:\PHP is a common location.

Copy some .dll files from your PHP directory to your system directory (usually C:\Windows). You need php5ts.dll for every case. You will also probably need to copy the file corresponding to your Web server module - C:\PHP\Sapi\php5apache.dll. to your Apache modules directory. It's possible that you will also need others from the dlls subfolder, but start with the two mentioned previously and add more if you need them.

Copy either php.ini-dist or php.ini-recommended (preferably the latter) to your Windows directory, and rename it php.ini. Open this file in a text editor (for example, Notepad). Edit this file to get configuration directives; At this point, we highly recommend that new users set error reporting to E_ALL on their development machines.

Tell your Apache server where you want to serve files from and what extension(s) you want to identify PHP files (.php is the standard, but you can use .html, .phtml, or whatever you want). Go to your HTTP configuration files (C:\Program Files\Apache Group\Apache\conf or whatever your path is), and open httpd.conf with a text editor. Search for the word DocumentRoot (which should appear twice) and change both paths to the directory you want to serve files out of. (The default is C:\Program Files\Apache Group\Apache\htdocs.). Add at least one PHP extension directive as shown in the first line of the following code:

LoadModule php5_module modules/php5apache.dll AddType application/x-httpd-php .php .phtml

You may also need to add the following line:

AddModule mod_php5.c

Stop and restart the WWW service. Go to the Start menu -> Settings -> Control Panel -> Services. Scroll down the list to IIS Admin Service. Select it and click Stop. After it stops, select World Wide Web Publishing Service and click Start. Stopping and restarting the service from within Internet Service Manager will not suffice. Since this is Windows, you may also wish to reboot.

Open a text editor. Type: <?php phpinfo(); ?>. Save this file in your Web server's document root as info.php.

Start any Web browser and browse the file.you must always use an HTTP request

(http://www.testdomain.com/info.php or http://localhost/info.php or http://127.0.0.1/info.php) rather than a filename (/home/httpd/info.php) for the file to be parsed correctly

You should see a long table of information about your new PHP installation message Congratulations!

Apache Configuration

If you are using Apache as a Web Server, then this section will guide you to edit Apache Configuration Files.

PHP.INI File Configuration

The PHP configuration file, php.ini, is the final and most immediate way to affect PHP's functionality.

Just Check it here: PHP.INI File Configuration

Windows IIS Configuration

To configure IIS on your Windows machine you can refer your IIS Reference Manual shipped along with IIS.

Apache Configuration for PHP

Apache uses httpd.conf file for global settings, and the .htaccess file for per-directory access settings. Older versions of Apache split up httpd.conf into three files (access.conf, httpd.conf, and srm.conf), and some users still prefer this arrangement.

Apache server has a very powerful, but slightly complex, configuration system of its own. Learn more about it at the Apache Web site: www.apache.org

The following section describes settings in httpd.conf that affect PHP directly and cannot be set elsewhere. If you have standard installation, then httpd.conf will be found at /etc/httpd/conf:

Timeout

This value sets the default number of seconds before any HTTP request will time out. If you set PHP's max_execution_time to longer than this value, PHP will keep grinding away but the user may see a 404 error. In safe mode, this value will be ignored; you must use the timeout value in php.ini instead

DocumentRoot

DocumentRoot designates the root directory for all HTTP processes on that server. It looks something like this on Unix:

DocumentRoot ./usr/local/apache_1.3.6/htdocs.

You can choose any directory as document root.

AddType

The PHP MIME type needs to be set here for PHP files to be parsed. Remember that you can associate any file extension with PHP like .php3, .php5 or .htm.

AddType application/x-httpd-php .php

AddType application/x-httpd-phps .phps

AddType application/x-httpd-php3 .php3 .phtml

AddType application/x-httpd-php .html

Action

You must uncomment this line for the Windows apxs module version of Apache with shared object support:

LoadModule php4_module modules/php4apache.dll

or on Unix flavors:

LoadModule php4_module modules/mod_php.so

AddModule

You must uncomment this line for the static module version of Apache.

AddModule mod_php4.c

PHP.INI file Configuration

The PHP configuration file, php.ini, is the final and most immediate way to affect PHP's functionality. The php.ini file is read each time PHP is initialized.in other words, whenever httpd is restarted for the module version or with each script execution for the CGI version. If your change isn‘t showing up, remember to stop and restart httpd. If it still isn‘t showing up, use phpinfo() to check the path to php.ini.

The configuration file is well commented and thorough. Keys are case sensitive, keyword values are not; whitespace, and lines beginning with semicolons are ignored. Booleans can be represented by 1/0, Yes/No, On/Off, or True/False. The default values in php.ini-dist will result in a reasonable PHP installation that can be tweaked later.

Here we are explaining the important settings in php.ini which you may need for your PHP Parser.

short_open_tag = Off

Short open tags look like this: <? ?>. This option must be set to Off if you want to use XML functions.

safe_mode = Off

If this is set to On, you probably compiled PHP with the --enable-safe-mode flag. Safe mode is most relevant to CGI use. See the explanation in the section "CGI compile-time options". earlier in this chapter.

safe_mode_exec_dir = [DIR]

This option is relevant only if safe mode is on; it can also be set with the --with-exec-dir flag during the Unix build process. PHP in safe mode only executes external binaries out of this directory. The default is /usr/local/bin. This has nothing to do with serving up a normal PHP/HTML Web page.

safe_mode_allowed_env_vars = [PHP_]

This option sets which environment variables users can change in safe mode. The default is only those variables prepended with "PHP_". If this directive is empty, most variables are alterable.

safe_mode_protected_env_vars = [LD_LIBRARY_PATH]

This option sets which environment variables users can't change in safe mode, even if safe_mode_allowed_env_vars is set permissively.

disable_functions = [function1, function2...]

A welcome addition to PHP4 configuration and one perpetuated in PHP5 is the ability to disable selected functions for security reasons. Previously, this necessitated hand-editing the C code from which PHP was made. Filesystem, system, and network functions should probably be the first to go because allowing the capability to write files and alter the system over HTTP is never such a safe idea.

max_execution_time = 30

The function set_time_limit() won.t work in safe mode, so this is the main way to make a script time out in safe mode. In Windows, you have to abort based on maximum memory consumed rather than time. You can also use the Apache timeout setting to timeout if you use Apache, but that will apply to non-PHP files on the site too.

error_reporting = E_ALL & ~E_NOTICE

The default value is E_ALL & ~E_NOTICE, all errors except notices. Development servers should be set to at least the default; only production servers should even consider a lesser value

error_prepend_string = [""]

With its bookend, error_append_string, this setting allows you to make error messages a different color than other text, or what you have.

warn_plus_overloading = Off

This setting issues a warning if the + operator is used with strings, as in a form value.

variables_order = EGPCS

This configuration setting supersedes gpc_order. Both are now deprecated along with register_globals. It sets the order of the different variables: Environment, GET, POST, COOKIE, and SERVER (aka Built-in).

You can change this order around. Variables will be overwritten successively in left-to-right order, with the rightmost one winning the hand every time. This means if you left the default setting and happened to use the same name for an environment variable, a POST variable, and a COOKIE variable, the COOKIE variable would own that name at the end of the process. In real life, this doesn't happen much.

register_globals = Off

This setting allows you to decide whether you wish to register EGPCS variables as global. This is now deprecated, and as of PHP4.2, this flag is set to Off by default. Use superglobal arrays instead. All the major code listings in this book use superglobal arrays.

gpc_order = GPC

This setting has been GPC Deprecated.

magic_quotes_gpc = On

This setting escapes quotes in incoming GET/POST/COOKIE data. If you use a lot of forms which possibly submit to themselves or other forms and display form values, you may need to set this directive to On or prepare to use addslashes() on string-type data.

magic_quotes_runtime = Off

This setting escapes quotes in incoming database and text strings. Remember that SQL adds slashes to single quotes and apostrophes when storing strings and does not strip them off when returning them. If this setting is Off, you will need to use stripslashes() when outputting any type of string data from a SQL database. If magic_quotes_sybase is set to On, this must be Off.

magic_quotes_sybase = Off

This setting escapes single quotes in incoming database and text strings with Sybase-style single quotes rather than backslashes. If magic_quotes_runtime is set to On, this must be Off.

auto-prepend-file = [path/to/file]

If a path is specified here, PHP must automatically include() it at the beginning of every PHP file. Include path restrictions do apply.

auto-append-file = [path/to/file]

If a path is specified here, PHP must automatically include() it at the end of every PHP file.unless you escape by using the exit() function. Include path restrictions do apply.

include_path = [DIR]

If you set this value, you will only be allowed to include or require files from these directories. The include directory is generally under your document root; this is mandatory if you.re running in safe mode. Set this to . in order to include files from the same directory your script is in. Multiple directories are separated by colons: .:/usr/local/apache/htdocs:/usr/local/lib.

doc_root = [DIR]

If you.re using Apache, you.ve already set a document root for this server or virtual host in httpd.conf. Set this value here if you.re using safe mode or if you want to enable PHP only on a portion of your site (for example, only in one subdirectory of your Web root).

file_uploads = [on/off]

Turn on this flag if you will upload files using PHP script.

upload_tmp_dir = [DIR]

Do not uncomment this line unless you understand the implications of HTTP uploads!

session.save-handler = files

Except in rare circumstances, you will not want to change this setting. So don't touch it.

ignore_user_abort = [On/Off]

This setting controls what happens if a site visitor clicks the browser.s Stop button. The default is On, which means that the script continues to run to completion or timeout. If the setting is changed to Off, the script will abort. This setting only works in module mode, not CGI.

mysql.default_host = hostname

The default server host to use when connecting to the database server if no other host isspecified.

mysql.default_user = username

The default user name to use when connecting to the database server if no other name is specified.

mysql.default_password = password

The default password to use when connecting to the database server if no other password is specified.

Declaring Variables

In PHP, a variable starts with the $ sign, followed by the name of the variable:

```
<?php
$txt = "Hello world!";
$x = 5;
$y = 10.5;
?>
```

After the execution of the statements above, the variable **$txt** will hold the value **Hello world!**, the variable **$x** will hold the value **5**, and the variable **$y** will hold the value **10.5**.

PHP Variables

A variable can have a short name (like x and y) or a more descriptive name (age, carname, total_volume).
Rules for PHP variables:

- A variable starts with the $ sign, followed by the name of thevariable
- A variable name must start with a letter or the underscorecharacter
- A variable name cannot start with anumber
- A variable name can only contain alpha-numeric characters and underscores (A-z, 0-9, and_)
- Variable names are case-sensitive ($age and $AGE are twodifferentvariables).

PHP is a Loosely Typed Language

In the example above, notice that we did not have to tell PHP which data type the variable is. PHP automatically converts the variable to the correct data type, depending on its value.In other languages such as C, C++, and Java, the programmer must declare the name and type of the variable before using it.

PHP Variables Scope

In PHP, variables can be declared anywhere in the script.
The scope of a variable is the part of the script where the variable can be referenced/used. PHP has three different variable scopes:

- local
- global
- static

Global And Local Scope

A variable declared **outside** a function has a GLOBAL SCOPE and can only be accessed outside a function:

```
<?php
$x = 5; // global scope


function myTest() {
   // using x inside this function will generate an error
   echo "<p>Variable x inside function is: $x</p>";
}
myTest();

echo "<p>Variable x outside function is: $x</p>";
?>
```

A variable declared **within** a function has a LOCAL SCOPE and can only be accessed within that function:

```
<?php
function myTest() {
   $x = 5; // local scope
   echo "<p>Variable x inside function is: $x</p>";
}
myTest();


// using x outside the function will generate an error echo
"<p>Variable x outside function is: $x</p>";
?>
```

PHP Data Types

Variables can store data of different types, and different data types can do different things. PHP supports the following data types:

- String
- Integer
- Float (floating point numbers - also calleddouble)
- Boolean
- Array
- Object
- NULL
- Resource

PHP String

A string is a sequence of characters, like "Hello world!".

A string can be any text inside quotes. You can use single or double quotes:

```
<?php
$x = "Hello world!";
$y = 'Hello world!';

echo      $x;      echo
"<br>"; echo $y;
?>
```

PHP Integer

An integer is a whole number (without decimals). It is a number between -2,147,483,648 and +2,147,483,647.

Rules for integers:

- An integer must have at least one digit(0-9)
- An integer cannot contain comma orblanks
- An integer must not have a decimalpoint
- An integer can be either positive ornegative
- Integers can be specified in three formats: decimal (10-based), hexadecimal (16-based - prefixed with 0x) or octal (8-based - prefixed with0)

In the following example $x is an integer. The PHP var_dump() function returns the data type and value:

```
<?php
$x = 5985;
var_dump($x);
?>
```

PHP Float

A float (floating point number) is a number with a decimal point or a number in exponential form.
In the following example $x is a float. The PHP var_dump() function returns the data type and value:

```
<?php
$x = 10.365;
var_dump($x);
?>
```

PHP Boolean

A Boolean represents two possible states: TRUE or FALSE.

```
$x = true;
$y = false;
```

PHP Array

An array stores multiple values in one single variable.
In the following example $cars is an array. The PHP var_dump() function returns the data type and value:

```
<?php
$cars = array("Volvo","BMW","Toyota"); var_dump($cars);
?>
```

PHP NULL Value

Null is a special data type which can have only one value: NULL.
A variable of data type NULL is a variable that has no value assigned to it.
Array
An array is a special variable, which can hold more than one value at a time.
If you have a list of items (a list of car names, for example), storing the cars in single variables could look like this:

```
$cars1 = "Volvo";
$cars2 = "BMW";
$cars3 = "Toyota";
```

However, what if you want to loop through the cars and find a specific one? And what if you had not 3 cars, but 300?
The solution is to create an array!
An array can hold many values under a single name, and you can access the values by referring to an index number.

Create an Array in PHP

In PHP, the array() function is used to create an array: In PHP,
there are three types of arrays:

- **Indexed arrays** - Arrays with a numericindex
- **Associative arrays** - Arrays with namedkeys
- **Multidimensional arrays** - Arrays containing one or morearrays

PHP Indexed Arrays

There are two ways to create indexed arrays:
The index can be assigned automatically (index always starts at 0), like this:
$cars = array("Volvo", "BMW", "Toyota");
The following example creates an indexed array named $cars, assigns three elements to it, and then prints a text containing the array values:

```
<?php
$cars = array("Volvo", "BMW", "Toyota");
echo "I like " . $cars[0] . ", " . $cars[1] . " and " . $cars[2] . ".";
?>
```

PHP Associative Arrays

Associative arrays are arrays that use named keys that you assign to them. There are two ways to create an associative array:

```
$age = array("Peter"=>"35", "Ben"=>"37", "Joe"=>"43");
<?php
$age = array("Peter"=>"35", "Ben"=>"37", "Joe"=>"43"); echo "Peter is " .
$age['Peter'] . " years old.";
?>
```

Strings

A string is a sequence of characters, like "Hello world!".

Get The Length of a String

The PHP strlen() function returns the length of a string.
The example below returns the length of the string "Hello world!":

```
<?php
echo strlen("Hello world!"); // outputs 12
?>
```

Count The Number of Words in a String

The PHP str_word_count() function counts the number of words in a string:

```
<?php
echo str_word_count("Hello world!"); // outputs 2
?>
```

The output of the code above will be: 2.

Reverse a String

The PHP strrev() function reverses a string:

```
<?php
echo strrev("Hello world!"); // outputs !dlrowolleH
?>
```

String Concatenation Operator

To concatenate two string variables together, use the dot (.) operator −

```
<?php
$string1="Hello World";
   $string2="1234";

echo $string1 . " " . $string2;
?>
```

This will produce the following result –

```
Hello World 1234
```

Using the strpos() function

The strpos() function is used to search for a string or character within a string.
If a match is found in the string, this function will return the position of the first match. If no match is found, it will return FALSE.
Let's see if we can find the string "world" in our string −

```
<?php
echostrpos("Hello world!","world");
?>
```

This will produce the following result −6

(2) **operators:**An operator is a symbol that specifies a particular action in an expression.operators are classified into differenttypes

(a) Arithmetic operators
(b)Assignment operators
(c)stringoperators
(d)Increment and decrement operators
(e)Logical operators
(f)Equality operators
(g)Comparision operators
(h)Bitwise operators

a) Arithmetic operators :The arthimetic operatorsare

operators	Label	Example
+	addition	$a+$b
-	Subtraction	$a-$b
*	Multiplication	$a*$b
/	Division	$a/$b
%	modulus	$a%$b

b) Assignment operators: Assignment operators mainly used for to assign a data value to a variable. The simplest form of assignment operator just assigns somevalue.

Operator	Label	Example
=	Assignment	$a=5
+=	Addition-assignment	$a+=5
*=	multiplication-assignment	$a*=5
/=	division-assignment	$a/=5
.=	concatenation-assignment	$a.=5

(c) **String operators:**php‘s string operators provide two operators that two operatorsusefullfor concatenation the twostrings.

Operator	label	example	output
.concatenation	$a= –abc‖.–def‖;	abcdef	
.=	concatenation-assighment	$a.=–ghijkl‖	ghijkl

(d) increment and decrementoperators:

Increment(++) and decrement operators increment by 1 and decrement by 1 from the current value of a variable

Operatorlabelexampleoutput			
++	increment	++$a,$a++	increment $a by1
--	Decrement	--$a,$a--	decrement $a by1

(e) logicaloperators:

Logical operators make it possible to direct the flow of a program and used frequently with control structures such as the if conditional and while and loops.

Operatorslabelexampleoutput			
&&	AND	$a&&$b	true if both $a and $baretrue AND
	AND	$a AND$b	true if both $a and $b aretrue
||	OR	$a \|\|$b	true if either $a or $bistrue OR

OR $a OR$b true if both $a and $b aretrue
! NOT !$a true if $a isnottrue NOT
NOTNOT$a true if $a is nottrue
XOR exclusive $aXOR$b true if only $a (or) only $b istrue

(f) equalityoperators:

Equality operators are used to compare two values, testing for equivalence

Operatorslabelexample

<less than $a<$b
> Greater than $a>$b
<= less than or equal to $a<=$b
>= greator than or equal to $a=>$b

(g) bitwiseoperators:

Bitwise operators are used for variations on some of the logical operators

Operatorslabelexampleoutput

& AND $a&$b and together each bit contained in $a and$b
| OR $a|$b or together each bit contained in $a and$b
^ XOR $a^$b exclusive-or togethereachbit contained in $a and$b
~ NOT ~$b negate each bit in$b
<<shiftleft $a<<$b $a will receive the valueof$b shifted left twovalues.
>> shiftright $a>>$b $a will receive the valueof$b shifted right twovalues.

(3) expressions andstatements:-

Expressions:-an expression is a phrase representing a particular action in a program.all expressions consists of a least one **operand** ana one (or) more operations

Ex:-

```
$a=5; //assign inter value 5 to the variable $a
$a=‖5‖; //assign string value –5‖ to the variable $a
$a=‖abit‖; //assign –abit‖ to the variable $a
```

Here operends are the input expressions

Ex:-$a++; //$a is the operand

$sum=$val1+val2; //$sum,$val1,$val2 are operends

Statements:-php supports different types of statements like (1)if statement (2)else statement (3)switch statement (4)while statement (5)do…while statement (6)for statement (7)for each statement (8)break and goto statement (9)continue statement.

(1)switch statement:-

Syntax:-if (expression)

```
{
    Statement
}
```

->An example,supposeuou want a cougratutory message displayed if the user guesses apredefimindservet number

```
<?php
```

```
$sescretnumber=143;
If($_POST[_guess‘]==$secretnumber)
{
Echo –<p>congratulation!</p>‖;
}
```

(2)**else statement:** if the condition si true statement followed if will be execute other else statement will execute

```
<?php
$sescretnumber=143;
If($_POST[_guess‘]==$secretnumber)
{
Echo –<p>congratulation!</p>‖;
}
Else
{
Echon –<p>sorry!</p>‖;
}
?>
```

(4) **switch statement:-** switch statement can compare — = — operations only ex:-

```
<?php
        $x=1;
Switch($x)
        {
case 1:
                Echo –number1‖;
                Break;
              Case    2:echo    –number2‖;
                        Break;
               Case 3:
                      Echo        –number3‖;
                       Break;
               Default:
                     Echo –no number b/w 1 and 3‖;
            }
           ?>
```

(5) **while statement:-** while loop check the condition then only excute the statement when the condition is true.

Syntax:-

```
While(expression)
          {
              Statements
          }
```

Ex:-

```
<?php
      $count=1;
While($count <5)
      {
Printf(–%d sqared=%d <br>‖,$count,pow($count,2));
         $count++;
     }
     ?>
```

o/p : 1 squared=1 2
squared=4
3 squared=9
4 squared=16

(6) dowhile:

It will execute the statement atleast once even condition false(or)true.

Syntax:

```
<?php
    $count=11;
   Do
  {
Printf(-%d squared=%d<br/>‖,$count,pow($count,2));
}
While($count<10);
?>
```

(7) **for statement:** By using this loop we can run number of iteration.

Syntax:for(exp1;exp2;exp3)

```
{
        Statements;
      }
```

There are afew rules to keep in mind when using php's for loops.

- The first expression,exp1,is evaluated by default at the first iteration of the loop
- The second expression,exp2 is evaluated at the beginning of each iteration.this expression determines wherher looping will continue.
- The third expression,exp3,is evaluated at conclusion of even loop.

Ex:

```
<?php
For($kilometers=1;$kilometers<=3;$kilometers++)
{
printf(-%kilometers=%f miles<br/>‖,$kilometers,$kilometers*0.62140);
}
?>
```

o/p:

1 kilometers= 0.6214miles
2 kilometers=1.2428miles
3 kilometers =1.8642 miles.

(a)**Break and gotostatement:-**

Break statement:- break statement is end execution of a do while,for,foreach,switch,while block.

Goto statement:- In php goto statement,‖BREAK‖ features was extend to support labels.

This means we can suddenly jump to a specific location outside of a looping or conditional construct.

(10) continue statement:-

Continue statement execute the current loop iteration to the end.

(4) **string:-** string variable can hold collection of characters . in php wecanassign valuesintot he string variables _3'ways.

-> using single quotation
-> using double quotation
->heradoc style.

->inphp offers approximately 100 function collectively.
->we introduce each function but we have to implement some of the functions of a string.

(1) determining stringlength
(2) comparing twostrings
(3) manipulating stringcase
(4) alternatives for regular expression functions s(5) converting string to and formHTML
(6) padding and stripping astring
(7) counting characters andworks

(1) **determining string length**:- here we are using strlen() function and this function returns the lengthof thestring.

Ex:-intstrlen(string str)

(2) **comparing two string**:- in php provides four functions for performing thistask.

1.strcmp ()
2.strcasecmp ()
3. strlen()
4.strcmp()

(1) strcmp():- the strcmp() function performs a binary numbers andcomparetwo strings.

Syntax:-(case-sensitive) Intstrcmp(string str1,string str2)

- 0, if str1 and str2 are equal.(s1==s2)
- -1, if str1 is less than str2(s1<s2)
- 1, if str2 is less than str1(s2<s1)

Ex:-<?php
$pwd=‖abitcse‖;
$pwd2=‖abitcse2‖;
if

Ex:-<?php
$pwd=‖abc123‖;
If(strcspn($pwd,‖1234567890‖)==0) Echo‖pwd can't consists solely of numbers! }
?>

(2) strspn:- calculating the similarities between twostrings.

Syntax: int strspn(string str1,string str2[, int start[, int length]])

Ex:-<?php
$pwd=‖abc123‖;
If(strspn($pwd,‖1234567890‖)==strlen($pwd))
Echo‖thepwd cannot consist solely of numbers!‖;
?>

(3) mani

```
Ex:-<?php
        $pwd=‖abitcse‖;
        $pwd2=‖abitcse2‖;
If(strcmp($pwd,$pwd2)!=0)
        {
Echo‖pwd do not match‘;
        }
        Else
        {
Echo‖pwd match‖;
        }
       ?>
```

(2) strcasecmp():-(case-insensitive0

The strcasecmp() function operates exactly like strcmp().

Syntax: intstrcasecmp(string str1,string str2) Ex:-

```
<?php
        $gmail1= ‖abit @gmail .com ‖;

        $gmail2=  ‖   ABIT@ gmail.com ‖;
If(!strcasecmp($gmail1,gmail20)
          Echo‖ the gmail addresses are identical! ‖;
       ?>
```

(3) strspn():- calculating the simirality between twostrings.

Syntax;-intstrspn(string str1,string str2 [, int start [, int length]]] Ex:-

```
<?php
        $pwd=‖abc123‖;
If(strspn($pwd,‖1234567890‖)==strlen($pwd))
         Echo – thepwd cannot consist solely of numbers! –;
         ?>
```

(4) strcspn():- calculating difference between twostrings.

Syntax:-intstrcspn(string str1,string str2 [, int start [, int length]]) Ex:-

```
   <?php
         $pwd=  –  abc123  –;  If(strcspn($pwd,
–1234567890 –)==0)
         {
          Echo – pwd can‘t consist solely of numbers! –;
         }
         ?>
```

(3) manipulating stringcase:-

In this we have to mainely concentrated on four function.

1. Strtolower()
2. Strtoupper()
3. Ucfirst()
4. Ucwords()

(1) Strtolower:- (converting a string to all lowercase) Ex:-

```
<?php
        $name=  –  bangaram –;  Echo  str
```

to lower($name);
?

(2) Strtoupper():- (converting a string to alluppercase) Ex:-

```
<?php
    $name= – JAMALBASHA –;
     Echo strtoupper
    ?>
```

(3) Ucfirst():- (capitalizing the first letter of a string) Ex:-

```
<?php
$name=‖ abit college —;
     Echo strucfirst ($name);
   ?>
```

(4) Ucwords():- (capitalizing the first letter of each words of a string). Ex;-

```
<?php
    $name – abitengg college –;
      Echo ucwords($name)
     ?>
```

(4) **Alternatives for regular expression function**:- in this we have to describe different types of functions. Thereare

(a) strtok()	(e)strrpos()	(i)substr_count()
(b) explodc()	(f)str_replace()	(j)substr_replace()
(c) implodc()	(g)strstr()	
(d) strops()	(h)substr()	

(a) **strtok()**:- this function parses the string based on a predefindlistof characters.

Syntax:- string strtok(string str,stringtokens)

Ex:-

```
<?php
    $into= — abit: abit@gmail.com\siddavatam,kdp —;
   $tokens= — :\ , ‖;
   $tokenized=    strtok($into,$tokens);
While( $tokenized)
   {
     Echo — elements = $tokenized<br>‖;
      $tokenized=strtok($tokens);
   }
  ?>
```

(b) **explode()**:-this function devides the string str into an array of substrings,in this we have to mainly concentrated areas are size of ( ) and stip-tags( ) to determine the total number of words. **Ex:** <?Php

```
    $summary==<<<summery
Php is a server side scripting language.
    Summary;
    $words=size    of(explode(_   _,strip-tags($summary)));
    Echo‖ total words in summary;$words‖;
     ?>
```

(c) **implode( ):**-we concatenate array elements to form a single delimited string using the implode( )function.

Ex: <? Php

```
    $cites=array(–kdp‖,‖antpr‖,‖tirupathi‖.);    Echo
    implode(–\‖,$cities);
   ?>
```

o/pkdp\antpr\tirupathi

(d) **strops():** in this function finds the position of the first case_sensitiveoccurance of a substring in

astring.
(e) **strrpos():-** in this function finds the last occurance of a string returningit‘snumerical position.
(f) **str_replace**():-this function case sensitively replaces all instance of a string with another. Ex:-

```
<?php
      $gmail= –abit@gmail.com –;
      $gmail=str_replace(–@‖,  ‖(is)‖,  $gmail);  Echo
       –college mail is $gmail;
      ?>
```

(g) **strstr():-** this function returns the remainder of a string beginning with the first occurance of a predefinedstring.
Ex:-

```
<?php
       $gmail= — abit@gmail.com —;
       Echo ltrim (strstr ($gmail, –@‖), –@‖);
      ?>
```

(h) **substr():-** this function returns the part of a string located between a predefined stringoffset and lengthpositions.
Ex:-

```
<?php
      $car= – 1994 ford –; Echo
      substr( $car,5);
      ?>
```

Ex2:-

```
<?php
        $car= – 1944 ford –; Echo
        substr( $car,0,4);
       ?>
```

(i) **substr-count() :** thisfunctionreturns the no.of times one string excussanother
Ex:

```
<?php
      $intu=array(–php‖, –XAMPP‖);
      $talk=<<<talk
      PHP is a scriptin language and php is server side
    Programming language.XAMPP is a web server Talk:
Foreach($info as $it)
   {
      Echo –the word $it appears‖.substr_count($talk,$it). –time(s)<br>/>‖;
}
  ?>
```

(j) **substr-replace();** replace the portion of a string with another string
Ex:

```
<?php
     $name=‖Abitcollege‖;
     Echo substr_replace($name, –engg‖,0,4);?>
```

(5) **converting string to HTML form:-** in this we are using different types of converting function .thereare
->Converting newline characters to HTML break tags. Ex:-

```
<?php
       $info=‖aaaaaaaaaaaaaaaaaaaaaa
Bbbbbbbbbbbbbbbbbbbbb
ccccccccccccccccccccccc
ddddddddddddddddddd‖;
echo n12br($into);
       ?>
```

Here we are not use
 statement.

->using special HTML characters for other purpose. In this we using htmlspecialchars() function.
->&->&
-> — ->&quoit;
-> _->'
->< ->&1t;
->> ->>
Ex:-<?php
$input= –<php is –scripting‖ language>‖; Echo
html\specialchars($input);
?>

(6) **padding and stripping a string**:- php provides no. of functions thereare

(a) ltrim()
(b) rtrim()
(c) trim()
(d) str_pad()

(a) **ltrim()**:- this function removes various characters from the beginning of a string including white space, horizontal tab(\t), newline(\n), carriage return(\v),null(\o).

String ltrim(string str [, string [, string charlist])

(b) **rtrim**():- this function removes various characters from the end of the string and except designatedcharacters.

String rtrim(string str [, string charlist])

(c) **trim**():-both l trim and rtrim

(d) **str_pad**():- this function pads a string with a specified number ofcharacters. Ex:-
<?php
Echo str_pad(–salad‖,10). –is good.‖;
?>**output**:- salad is good

(7) **counting characters and words:-** it‘s mainely used for to determine the total number of characters or words in a given string. Php provides two functions. there are count_chars()and str_word_count.

(6) arrays and functions:- array is a collection of heterogeneous(different elements) data types in php. Because php is a loosely typedlanguage.

Ex:-<?php
$arr=array(10,20,30);
Print_r($str);
?>**output**:- [0]=10,[1]=20,[2]=30

Ex2:- <?php
$arr=array(100->10, 101->20, ->102=>30);
Print_r($arr);
?>**output**:-[100]=10, [101]=20, [102]=30

Ex3:- <?php
$arr=array(100=>10,20,30, 106=>30);
Print_r($arr);
?>**output**:- [100]=10,[101]=20,[102]=30,[106]=30

Ex:-<?php
$arr=array(100=>10, _city‘=> ‘hyd‘, 105=>30, 50=>40,70); Print_r($arr);
?>**output**:-[100]=40,[city]=hyd,[105]=30,[50]=40,[106]=70

Array functions:-

Count**:- it returns total no. of elements Ex:-<?php**
$arr=array(10,20,30);
Echo count($arr);

?>**output**:-3

Sort :- it returns the elements of an array in assending order. Ex:-<?php
$arr=array(60,20,30);
Sort($arr);
Print_r($arr);
?>**output**:- 20,30,60

rsort :- it returns the elements of an array in descending order. Ex:-<?php
$arr=array(101,104,102);
rsort_r($arr);
print_r($arr);
?> output:-104,102,101

asort:- it returns the original keys with assending order. Ex**:-**<?php
$arr=array(104=>40, 101=>20, 108=>50, 102=>80);
assort($arr);
print_r($arr);
?> output:-101=20,104=40,108=50,102=80

arsort:-it returns the original key values with descending order. Ex:-<?php
$arr=array(104=>40, 101=>20, 108=>50, 102=>80);
arsort($arr);
Print_r($arr);
?>ouput:-[102]=80,[108]=50,[104]=40,[101=20

ksort:-it returns the array in assending order with based on the‒keys‖**.** Ex:-<?php
$arr=array(104=>40,101=>20,108=>50,102=>80);
Ksort($arr);
Print_r($arr);
?> output:-[101]=20,[102]=80,[104]=40,[108]=50

krsort:- it returns the array in dessending order with based on ‒keys‖. Ex:-<?php
$arr=array(104=>40,101=>20,108=>50,102=>80);
Krsort($arr);
Print_r($arr);
?> output:-[108]=50,[104]=40,[102]=80,[101]=20

array push():- this function adds an elements into the end of an array and returns the total no. of elements in that array.
Ex:-<?php
$arr=array(10,20,30); Echo
array_push($arr,40);
Print_r($arr);
?> output:-

array_pop():- remove the last element & return the value of that element. Ex:-<?php
$arr=array(10,20,30); Echo
array_pop($arr);
Print_r($arr);
?> output:-

array_shift():-it removes the first element of an array and returns the value of that element.
Ex:-<?php
$arr=array(10,20,30); Echo
array_shift($arr);
Print_r($arr);
?> output:-

array_unshift():- add an element at the beginning of an array and return size of an array.
Ex:-<?php
$arr=array(10,20,30);

Echo array_unshift($arr);
Print_r($arr);
?> output:-

array_change_key_case():- it converts all keys of an array into lower case. Ex:-<?php
$arr=array(_ABC‘=>10,20,30);
Print_r(array_change_key_case($arr));
?> output:-

array_chunk():- splits an array into chunk of an array. Ex:-<?php
$arr=array(10,20,30,40,50,60);
Print_r(array_chunk($arr,2));
?> output:-

array_combine():- creats an array by using one array one array for keys and another for it‘s value. Ex:-<?php
$arr=array(_abc‘=>10,20,30,40,50);
$arr1=array(100,200,300,400,500);
Print_r(array_combine($arr,$arr1));
?> output:-

array_keys:- it returns new array with keys as value of another array. Ex:-<?php
$arr=array(_abc‘=>10,20,30,40);
Print_r(array_keys($arr));
?> output:-

array_count_values():- returns an array with no of occurance for each value. Ex:-<?php
$arr=array(_ABC‘=>10,20,30,40,50,10);
Printf_r(array_count_values($arr));
?>

array_values():=return array with the values of an array
ex:- <?php
$arr=array(_ABC‘=10,20,30,40,50,60);
Printf_r(array_values($arr));
?>

array_flip():-exchanges all keys with their associated values in array ex:-<?php
$arr=array(_ABC‘->10,20,30,40);
//$arr=array(10,200,400);
Printf_r(array-flip($arr));
?>

array_interest():-compaves array values and returns the matches ex:-<?php
$arr=array(10,20,30,40);
$arr=array(100,200,300,400,10);
Printf_r(array_interest($arr)); //($arr,$arr1)
?>**array_interest_assoc():-**compaves array key and values and returns the matches ex:-<?php
$arr=array(10,20,30,40);
$arr1=array(100,200,300,400,0=>10);
Printf_r(array_interest_assoc($arr,$arr1));
?>

array_merge():-merges one or more arrays into one array
ex:-<?php
$arr=array(_ABC‘=>10,20,30,40);
$arr=array(100,200,300); Printf_r(array-merge($arr,$arr1));
?>

array_product():-returns the product of all array element values ex:-<?php

```
$arr=array(_ABC‘=>10,20,30,40);
Echo print_r(array_product($arr));
?>
```

array_sum():-returns the sum of all elements of an array ex:-

```
<?php
$arr=array(10,20,30,40);
Echo print_r(array_sum($arr));
?>
```

array_reverse():-it revers the elements of an array ex:-

```
<?php
$arr=array(10,20,30,40);
Print_r(array_reverse($arr));
?>
```

array_unique():-removes the duplicate values and returns the values of an array ex:-

```
<?php
$arr=(_ABC‘=>10,20,30,40);
Print_r(array_unique($arr));
?>
```

shuffle():-shuffle the elements of an array ex:-

```
<?php
$arr=array(_ABC‘=>10,20,30,40);
Shuffle($arr);
Print_r($arr);
?>
```

extract():-divides the elements of an array as individual variables ex:-

```
<?php
$arr=array(_ABC‘=>10,20,30,40);
Extract($arr);
Echo $ABC;
?>
```

list():-assign variables as it they were an array means that,we can assign the values of an array into variables

Ex:-

```
<?php
List($x,$y,$z)=array(10,20,30);
        Echo    $x;
        Echo    $y;
        Echo $z;
```

UNIT-V
PHP and Database Access:

PHP and database access: Basic database concepts, connecting to a MySQL database, retrieving and displaying results, modifying, updating and deleting data; MVC architecture: PHP andother web technologies: PHP and XML.

Connecting to MySQL Database

Opening a Database Connection

PHP provides **mysql_connect** function to open a database connection. This function takes five parameters and returns a MySQL link identifier on success, or FALSE on failure.

Syntax

connection mysql_connect(server,user,passwd,new_link,client_flag);

Sr.No	Parameter & Description
1	**server** Optional − The host name running database server. If not specified, then default value is **localhost:3306**.
2	**user** Optional − The username accessing the database. If not specified, then default is the name of the user that owns the server process.
3	**passwd** Optional − The password of the user accessing the database. If not specified then default is an empty password.
4	**new_link** Optional − If a second call is made to mysql_connect() with the same arguments, no new connection will be established; instead, the identifier of the already opened connection will be returned.
	client_flags

5	Optional − A combination of the following constants − • **MYSQL_CLIENT_SSL** − Use SSL encryption • **MYSQL_CLIENT_COMPRESS** − Use compression protocol ▪ **MYSQL_CLIENT_IGNORE_SPACE** − Allow space after functionnames ▪ **MYSQL_CLIENT_INTERACTIVE** − Allow interactive timeoutsecondsof inactivity before closing the connection

NOTE − You can specify server, user, passwd in **php.ini** file instead of using them againand again in your every PHP scripts. Check php.ini fileconfiguration.

Closing Database Connection

Its simplest function **mysql_close** PHP provides to close a database connection. This function takes connection resource returned by mysql_connect function. It returns TRUE on success or FALSE on failure.

Syntax

```
bool mysql_close ( resource $link_identifier );
```

If a resource is not specified, then the last opened database is closed.

Example

Try the following example to open and close a database connection −

```
<?php
    $dbhost = 'localhost:3036';

    $dbuser = 'guest';

    $dbpass = 'guest123';

    $conn = mysql_connect($dbhost, $dbuser, $dbpass);


    if(! $conn ) {

        die('Could not connect: ' . mysql_error());
```

```
    }

    echo 'Connected successfully';

    mysql_close($conn);

?>
```

Create MySQL Database Using PHP

Creating a Database

To create and delete a database, you should have admin privilege. It's very easy to create a new MySQL database. PHP uses **mysql_query** function to create a MySQL database. This function takes two parameters and returns TRUE on success or FALSE on failure.

Syntax

```
bool mysql_query( sql, connection );
```

Sr.No	Parameter & Description
1	**sql** Required - SQL query to create a database
2	**connection** Optional - if not specified, then the last opened connection by mysql_connect will be used.

Example

Try the following example to create a database −

```
<?php

    $dbhost = 'localhost:3036';

    $dbuser = 'root';

    $dbpass = 'rootpassword';
```

```
$conn = mysql_connect($dbhost, $dbuser, $dbpass);
if(! $conn ) {

    die('Could not connect: ' . mysql_error());

}

echo 'Connected successfully';
$sql = 'CREATE Database test_db';

$retval = mysql_query( $sql, $conn );

if(! $retval ) {

    die('Could not create database: ' . mysql_error());


}

echo "Database test_db created successfully\n";
mysql_close($conn);

?>
```

Selecting a Database

Once you establish a connection with a database server, then it is required to select a particular database with which all your tables are associated.

This is required because there may be multiple databases residing on a single server and you can do work with a single database at a time.

PHP provides function **mysql_select_db** to select a database. It returns TRUE on success or FALSE on failure.

Syntax

```
bool mysql_select_db( db_name, connection );
```

Sr.No	Parameter & Description
1	**db_name** Required - Database name to be selected
2	**connection** Optional - if not specified, then the last opened connection by mysql_connect will be used.

Example

Here is the example showing you how to select a database.

```
<?php
    $dbhost = 'localhost:3036';
    $dbuser = 'guest';
    $dbpass = 'guest123';
    $conn = mysql_connect($dbhost, $dbuser, $dbpass);

    if(! $conn ) {
        die('Could not connect: ' . mysql_error());
    }
    echo 'Connected successfully';
    mysql_select_db( 'test_db' );
    mysql_close($conn);
?>
```

Creating Database Tables

To create tables in the new database, you need to do the same thing as creating the database. First create the SQL query to create the tables, then execute the query using mysql_query() function.

Example

Try the following example to create a table −

```
<?php
    $dbhost = 'localhost:3036';
    $dbuser = 'root';
    $dbpass = 'rootpassword';
    $conn = mysql_connect($dbhost, $dbuser, $dbpass);
```

```
if(! $conn ) {
    die('Could not connect: ' . mysql_error());
}
echo 'Connected successfully';
$sql = 'CREATE TABLE employee( '.
    'emp_id INT NOT NULL AUTO_INCREMENT, '.
    'emp_name VARCHAR(20) NOT NULL, '.
    'emp_address    VARCHAR(20) NOT NULL, '.
    'emp_salary     INT NOT NULL, '.
    'join_date      timestamp(14) NOT NULL, '.
    'primary key ( emp_id ))';

mysql_select_db('test_db');
$retval = mysql_query( $sql, $conn );
if(! $retval ) {
    die('Could not create table: ' . mysql_error());
}
echo "Table employee created successfully\n";
mysql_close($conn);
?>
```

In case you need to create many tables, then it‘s better to create a text file first and put all the SQL commands in that text file and then load that file into $sql variable and execute those commands.

Consider the following content in sql_query.txt file

```
CREATE TABLE employee(
    emp_id INT NOT NULL AUTO_INCREMENT,
    emp_name VARCHAR(20) NOT NULL,
    emp_address    VARCHAR(20) NOT NULL,
```

```
    emp_salary      INT NOT NULL,
    join_date       timestamp(14) NOT NULL,
    primary key ( emp_id ));
```

```
<?php
    $dbhost = 'localhost:3036';
    $dbuser = 'root';
    $dbpass = 'rootpassword';
    $conn = mysql_connect($dbhost, $dbuser, $dbpass);
    if(! $conn ) {
        die('Could not connect: ' . mysql_error());
    }
    $query_file = 'sql_query.txt';
    $fp = fopen($query_file, 'r');
    $sql = fread($fp, filesize($query_file));
    fclose($fp);

    mysql_select_db('test_db');
    $retval = mysql_query( $sql, $conn );
    if(! $retval ) {
        die('Could not create table: ' . mysql_error());
    }

    echo "Table employee created successfully\n";
    mysql_close($conn);
?>
```

Delete MySQL Database Using PHP

Deleting a Database

If a database is no longer required, then it can be deleted forever. You can use pass an SQL command to **mysql_query** to delete a database.

Example

Try the following example to drop a database.

```
<?php
    $dbhost = 'localhost:3036';
    $dbuser = 'root';
    $dbpass = 'rootpassword';
    $conn = mysql_connect($dbhost, $dbuser, $dbpass);

    if(! $conn ) {
        die('Could not connect: ' . mysql_error());
    }

    $sql = 'DROP DATABASE test_db';
    $retval = mysql_query( $sql, $conn );

    if(! $retval ) {
        die('Could not delete database db_test: ' . mysql_error());
    }

    echo "Database deleted successfully\n";

    mysql_close($conn);
?>
```

WARNING – It‘svery dangerous to delete a database and any table. So before deletingany table or database you should make sure you are doing everything intentionally.

Deleting a Table

It‘s again a matter of issuing one SQL command through **mysql_query** function to delete any database

table. But be very careful while using this command because by doing so you can delete some important information you have in your table.

Example

Try the following example to drop a table −

```
<?php
    $dbhost = 'localhost:3036';
    $dbuser = 'root';
    $dbpass = 'rootpassword';
    $conn = mysql_connect($dbhost, $dbuser, $dbpass);

    if(! $conn ) {
        die('Could not connect: ' . mysql_error());
    }

    $sql = 'DROP TABLE employee';
    $retval = mysql_query( $sql, $conn );

    if(! $retval ) {
        die('Could not delete table employee: ' . mysql_error());
    }

    echo "Table deleted successfully\n";

    mysql_close($conn);
?>
```

Insert Data to MySQL Database

Data can be entered into MySQL tables by executing SQL INSERT statement through PHP function **mysql_query**. Below a simple example to insert a record into **employee** table.

Example

Try the following example to insert record into employee table.

```
<?php
    $dbhost = 'localhost:3036';
    $dbuser = 'root';
    $dbpass = 'rootpassword';
    $conn = mysql_connect($dbhost, $dbuser, $dbpass);

    if(! $conn ) {
        die('Could not connect: ' . mysql_error());
    }

    $sql = 'INSERT INTO employee '.
        '(emp_name,emp_address, emp_salary, join_date) '.
        'VALUES ( "guest", "XYZ", 2000, NOW() )';

    mysql_select_db('test_db');
    $retval = mysql_query( $sql, $conn );

    if(! $retval ) {
        die('Could not enter data: ' . mysql_error());
    }

    echo "Entered data successfully\n";

    mysql_close($conn);
?>
```

In real application, all the values will be taken using HTML form and then those values will be captured using PHP script and finally they will be inserted into MySQL tables.

While doing data insert its best practice to use function **get_magic_quotes_gpc()** to check if current configuration for magic quote is set or not. If this function returns false, then use function **addslashes()** to add slashes before quotes.

Example

Try this example by putting this code into add_employee.php, this will take input using HTML Form and then it will create records into database.

```
<html>
   <head>
      <title>Add New Record in MySQL
   Database</title></head>

   <body>

      <?php

         if(isset($_POST['add'])) {

            $dbhost = 'localhost:3036';

            $dbuser = 'root';

            $dbpass = 'rootpassword';

            $conn = mysql_connect($dbhost, $dbuser, $dbpass);


            if(! $conn ) {

               die('Could not connect: ' . mysql_error());

            }


            if(! get_magic_quotes_gpc() ) {

               $emp_name = addslashes ($_POST['emp_name']);

               $emp_address = addslashes ($_POST['emp_address']); }else
            {

               $emp_name = $_POST['emp_name'];

               $emp_address = $_POST['emp_address'];

            }


            $emp_salary = $_POST['emp_salary'];
```

```
    $sql = "INSERT INTO employee ". "(emp_name,emp_address, emp_salary,
        join_date) ". "VALUES('$emp_name','$emp_address',$emp_salary, NOW())";

    mysql_select_db('test_db');
    $retval = mysql_query( $sql, $conn );

    if(! $retval ) {
        die('Could not enter data: ' . mysql_error());
    }

    echo "Entered data successfully\n";

    mysql_close($conn);
}else {

    ?>

        <form method = "post" action = "<?php $_PHP_SELF ?>"><table
            width = "400" border = "0" cellspacing = "1"
                cellpadding = "2">

                <tr>
                    <td width = "100">Employee Name</td>
                    <td><input name = "emp_name" type = "text"
                        id = "emp_name"></td>
                </tr>

                <tr>
                    <td width = "100">Employee Address</td>
                    <td><input name = "emp_address" type = "text"
```

```
                    id = "emp_address"></td>
                </tr>

                <tr>
                    <td width = "100">Employee Salary</td>
                    <td><input name = "emp_salary" type = "text"
                        id = "emp_salary"></td>
                </tr>

                <tr>
                    <td width = "100"></td>
                    <td></td>
                </tr>

                <tr>
                    <td width = "100"></td>
                    <td>
                        <input name = "add" type = "submit" id = "add"
                            value = "Add Employee">
                    </td>
                </tr>

            </table>
        </form>
    ?php
}
?>
```

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

Retrieving Data from MySQL Database

Data can be fetched from MySQL tables by executing SQL SELECT statement through PHP function mysql_query. You have several options to fetch data from MySQL.

The most frequently used option is to use function **mysql_fetch_array()**. This function returns row as an associative array, a numeric array, or both. This function returns FALSE if there are no more rows.

Following is a simple example to fetch records from **employee** table.

Example

Try the following example to display all the records from employee table.

```
<?php
    $dbhost = 'localhost:3036';
    $dbuser = 'root';
    $dbpass = 'rootpassword';

    $conn = mysql_connect($dbhost, $dbuser, $dbpass);

    if(! $conn ) {
        die('Could not connect: ' . mysql_error());
    }

    $sql = 'SELECT emp_id, emp_name, emp_salary FROM employee';
    mysql_select_db('test_db');
    $retval = mysql_query( $sql, $conn );

    if(! $retval ) {
        die('Could not get data: ' . mysql_error());
    }
```

```
    while($row = mysql_fetch_array($retval, MYSQL_ASSOC)) {
        echo "EMP ID :{$row['emp_id']}<br> ". "EMP NAME :
            {$row['emp_name']} <br> ". "EMP SALARY :
            {$row['emp_salary']} <br> ". "--------------------------------
            <br>";

    }
    echo "Fetched data successfully\n";
    mysql_close($conn);

?>
```

The content of the rows are assigned to the variable $row and the values in row are then printed.

NOTE − Always remember to put curly brackets when you want to insert an arrayvaluedirectly into a string.

In the above example, the constant **MYSQL_ASSOC** is used as the second argument to mysql_fetch_array(), so that it returns the row as an associative array. With an associative array, you can access the field by using their name instead of using the index.

PHP provides another function called **mysql_fetch_assoc()** which also returns the row as an associative array.

Example

Try the following example to display all the records from employee table using mysql_fetch_assoc() function.

```
<?php

    $dbhost = 'localhost:3036';

    $dbuser = 'root';

    $dbpass = 'rootpassword';


    $conn = mysql_connect($dbhost, $dbuser, $dbpass);


    if(! $conn ) {

        die('Could not connect: ' . mysql_error());

    }


    $sql = 'SELECT emp_id, emp_name, emp_salary FROM employee';
```

```
    mysql_select_db('test_db');

    $retval = mysql_query( $sql, $conn );

    if(! $retval ) {

        die('Could not get data: ' . mysql_error());

    }


    while($row = mysql_fetch_assoc($retval)) {

        echo "EMP ID :{$row['emp_id']}<br> ". "EMP NAME :
            {$row['emp_name']} <br> ". "EMP SALARY :
            {$row['emp_salary']} <br> ". "----------------------------------
            <br>";

    }


    echo "Fetched data successfully\n";


    mysql_close($conn);

?>
```

You can also use the constant **MYSQL_NUM**, as the second argument to mysql_fetch_array(). This will cause the function to return an array with numeric index.

Example

Try the following example to display all the records from employee table using MYSQL_NUM argument.

```
<?php

    $dbhost = 'localhost:3036';

    $dbuser = 'root';

    $dbpass = 'rootpassword';


    $conn = mysql_connect($dbhost, $dbuser, $dbpass);


    if(! $conn ) {

        die('Could not connect: ' . mysql_error());
```

```
    }

    $sql = 'SELECT emp_id, emp_name, emp_salary FROM employee';
    mysql_select_db('test_db');

    $retval = mysql_query( $sql, $conn );
    if(! $retval ) {

        die('Could not get data: ' . mysql_error());

    }

    while($row = mysql_fetch_array($retval, MYSQL_NUM)) {

        echo "EMP ID :{$row[0]}       <br> ".

            "EMP NAME : {$row[1]} <br> ".

            "EMP SALARY : {$row[2]} <br> ".

            "--------------------------------- <br>";

    }

    echo "Fetched data successfully\n";

    mysql_close($conn);

?>
```

All the above three examples will produce the same result.

Releasing Memory

It is a good practice to release cursor memory at the end of each SELECT statement. This can be done by using PHP function **mysql_free_result()**. Below is the example to show how it has to be used.

Example

Try the following example.

```
<?php

    $dbhost = 'localhost:3036';

    $dbuser = 'root';
```

```
$dbpass = 'rootpassword';

$conn = mysql_connect($dbhost, $dbuser, $dbpass);

if(! $conn ) {
    die('Could not connect: ' . mysql_error());
}

$sql = 'SELECT emp_id, emp_name, emp_salary FROM employee';
mysql_select_db('test_db');

$retval = mysql_query( $sql, $conn );

if(! $retval ) {
    die('Could not get data: ' . mysql_error());
}

while($row = mysql_fetch_array($retval, MYSQL_NUM)) {
    echo "EMP ID :{$row[0]}       <br> ".
        "EMP NAME : {$row[1]} <br> ".
        "EMP SALARY : {$row[2]} <br> ".
        "-------------------------------- <br>";
}

mysql_free_result($retval);

echo "Fetched data successfully\n";

mysql_close($conn);
?>
```

While fetching data, you can write as complex SQL as you like. The procedure will remain the same as mentioned above.

Using Paging through PHP

It‘s always possible that your SQL SELECT statement query may result into thousands of records. But it is not good idea to display all the results on one page. So we can divide this result into many pages as per requirement.

Paging means showing your query result in multiple pages instead of just put them all in one long page.

MySQL helps to generate paging by using **LIMIT** clause which will take two arguments. First argument as OFFSET and second argument how many records should be returned from the database.

Following is a simple example to fetch records using **LIMIT** clause to generatepaging.

Example

Try the following example to display 10 records per page.

```
<html>
    <head>
        <title>Paging Using PHP</title>

    </head>
    <body>
        <?php
            $dbhost = 'localhost:3036';
            $dbuser = 'root';
            $dbpass = 'rootpassword';

            $rec_limit = 10;
            $conn = mysql_connect($dbhost, $dbuser, $dbpass);

            if(! $conn ) {
                die('Could not connect: ' . mysql_error());
            }
            mysql_select_db('test_db');
```

```
/* Get total number of records */

$sql = "SELECT count(emp_id) FROM employee "; $retval
= mysql_query( $sql, $conn );

if(! $retval ) {

    die('Could not get data: ' . mysql_error());

}

$row = mysql_fetch_array($retval, MYSQL_NUM );
$rec_count = $row[0];

if( isset($_GET{'page'} ) ) {

    $page = $_GET{'page'} + 1;

    $offset = $rec_limit * $page ;

}else {

    $page = 0;

    $offset = 0;

}


$left_rec = $rec_count - ($page * $rec_limit); $sql = "SELECT
emp_id, emp_name, emp_salary ".

    "FROM employee ".

    "LIMIT $offset, $rec_limit";
$retval = mysql_query( $sql, $conn );


if(! $retval ) {

    die('Could not get data: ' . mysql_error());

}


while($row = mysql_fetch_array($retval, MYSQL_ASSOC)) { echo
    "EMP ID :{$row['emp_id']} <br> ".

        "EMP NAME : {$row['emp_name']} <br> ".

        EMP SALARY : {$row['emp_salary']} <br> ".
```

```
            " ---------------------------------<br>";

        }


        if( $page > 0 ) {

            $last = $page - 2;

            echo "<a href = \"$_PHP_SELF?page = $last\">Last 10 Records</a> |"; echo "<a href =
            \"$_PHP_SELF?page = $page\">Next 10 Records</a>";

        }else if( $page == 0 ) {

            echo "<a href = \"$_PHP_SELF?page = $page\">Next 10 Records</a>"; }else if(
        $left_rec < $rec_limit ) {

            $last = $page - 2;

            echo "<a href = \"$_PHP_SELF?page = $last\">Last 10 Records</a>";

        }


        mysql_close($conn);

    ?>


  </body>

</html>
```

Updating Data into MySQL Database

Data can be updated into MySQL tables by executing SQL UPDATE statement through PHP function **mysql_query**.

Below is a simple example to update records into **employee** table. To update a record in any table it is required to locate that record by using a conditional clause. Below example uses primary key to match a record in employee table.

Example

Try the following example to understand update operation. You need to provide an employee ID to update an employee salary.

```
<html>
```

```
<head>
    <title>Update a Record in MySQL Database</title></head>
<body>
    <?php
        if(isset($_POST['update'])) {
            $dbhost = 'localhost:3036';
            $dbuser = 'root';
            $dbpass = 'rootpassword';

            $conn = mysql_connect($dbhost, $dbuser, $dbpass);

            if(! $conn ) {
                die('Could not connect: ' . mysql_error());
            }

            $emp_id = $_POST['emp_id'];
            $emp_salary = $_POST['emp_salary'];

            $sql = "UPDATE employee ". "SET emp_salary = $emp_salary ".
                "WHERE emp_id = $emp_id" ;
            mysql_select_db('test_db');
            $retval = mysql_query( $sql, $conn );

            if(! $retval ) {
                die('Could not update data: ' . mysql_error());
            }
            echo "Updated data successfully\n";
```

```
    mysql_close($conn);
}else {
    ?>
        <form method = "post" action = "<?php $_PHP_SELF ?>"><table width
            = "400" border =" 0" cellspacing = "1"
                cellpadding = "2">
                <tr>
                    <td width = "100">Employee ID</td>
                    <td><input name = "emp_id" type = "text"
                        id = "emp_id"></td>
                </tr>
                <tr>
                    <td width = "100">Employee Salary</td><td><input name
                    = "emp_salary" type = "text"
                            id = "emp_salary"></td>
                </tr>
                <tr>
                    <td width = "100"></td>
                    <td></td>
                </tr>
                <tr>
                    <td width = "100"></td>
                    <td>
                        <input name = "update" type = "submit"
                            id = "update" value = "Update">
```

```
                        </td>
                    </tr>

                </table>
            </form>
        <?php
        }
    ?>

</body>
</html>
```

Deleting Data from MySQL Database

Data can be deleted from MySQL tables by executing SQL DELETE statement through PHP function **mysql_query**.

Following is a simple example to delete records into **employee** table. To delete a record in any table it is required to locate that record by using a conditional clause. Below example uses primary key to match a record in employee table.

Example

Try the following example to understand delete operation. You need to provide an employee ID to delete an employee record from employee table.

```
<html>
    <head>
        <title>Delete a Record from MySQL Database</title></head>
    <body>
        <?php

            if(isset($_POST['delete'])) {

                $dbhost = 'localhost:3036';

                $dbuser = 'root';

                $dbpass = 'rootpassword';

                $conn = mysql_connect($dbhost, $dbuser, $dbpass);
```

```
    if(! $conn ) {

        die('Could not connect: ' . mysql_error());

    }

    $emp_id = $_POST['emp_id'];
    $sql = "DELETE employee ". "WHERE emp_id = $emp_id" ;
    mysql_select_db('test_db');

    $retval = mysql_query( $sql, $conn );


    if(! $retval ) {

        die('Could not delete data: ' . mysql_error());

    }
    echo "Deleted data successfully\n";

    mysql_close($conn);

}else {

    ?>

        <form method = "post" action = "<?php $_PHP_SELF ?>"><table width
            = "400" border = "0" cellspacing = "1"

                cellpadding = "2">


                <tr>

                    <td width = "100">Employee ID</td>

                    <td><input name = "emp_id" type = "text"

                        id = "emp_id"></td>

                </tr>


                <tr>

                    <td width = "100"></td>

                    <td></td>

                </tr>
```

```
                        <tr>
                            <td width = "100"></td>
                            <td>
                                <input name = "delete" type = "submit"
                                    id = "delete" value = "Delete">
                            </td>
                        </tr>

                    </table>
                </form>
            <?php
        }
    ?>

    </body>
</html>
```

Using PHP to Backup MySQL Database

It is always a good practice to take a regular backup of your database. There are three ways you can use to take backup of your MySQL database.

- Using SQL Command through PHP.
- Using MySQL binary mysqldump through PHP.
- Using phpMyAdmin user interface.

Using SQL Command through PHP

You can execute SQL SELECT command to take a backup of any table. To take a complete database dump you will need to write separate query for separate table. Each table will be stored into separate text file.

Example

Try the following example of using SELECT INTO OUTFILE query for creating table backup −

```
<?php
    $dbhost = 'localhost:3036';
    $dbuser = 'root';
    $dbpass = 'rootpassword';

    $conn = mysql_connect($dbhost, $dbuser, $dbpass);

    if(! $conn ) {
        die('Could not connect: ' . mysql_error());
    }

    $table_name = "employee";
    $backup_file      = "/tmp/employee.sql";
    $sql = "SELECT * INTO OUTFILE '$backup_file' FROM $table_name";

    mysql_select_db('test_db');
    $retval = mysql_query( $sql, $conn );

    if(! $retval ) {
        die('Could not take data backup: ' . mysql_error());
    }
    echo "Backedup     data successfully\n";
    mysql_close($conn);
?>
```

There may be instances when you would need to restore data which you have backed up some time ago. To restore the backup, you just need to run LOAD DATA INFILE query like this −

```
<?php
    $dbhost = 'localhost:3036';
    $dbuser = 'root';
    $dbpass = 'rootpassword';
    $conn = mysql_connect($dbhost, $dbuser, $dbpass);
```

```
    if(! $conn ) {
        die('Could not connect: ' . mysql_error());
    }
    $table_name = "employee";
    $backup_file      = "/tmp/employee.sql";
    $sql = "LOAD DATA INFILE '$backup_file' INTO TABLE $table_name";
    mysql_select_db('test_db');
    $retval = mysql_query( $sql, $conn );
    if(! $retval ) {

        die('Could not load data : ' . mysql_error());

    }
    echo "Loaded     data successfully\n";
    mysql_close($conn);
?>
```

Using MySQL binary mysqldump through PHP

MySQL provides one utility **mysqldump** to perform database backup. Using this binary you can take complete database dump in a single command.

Example

Try the following example to take your complete database dump −

```
<?php
    $dbhost = 'localhost:3036';
    $dbuser = 'root';
    $dbpass = 'rootpassword';
    $backup_file = $dbname . date("Y-m-d-H-i-s") . '.gz';
    $command = "mysqldump --opt -h $dbhost -u $dbuser -p $dbpass ". "test_db | gzip >
$backup_file";
    system($command);
?>
```

Using phpMyAdmin user interface

If you have **phpMyAdmin** user interface available, then it is very easy for you to take backup of your database.

To back up your MySQL database using phpMyAdmin click on the "export" link on phpMyAdmin main page. Choose the database you wish to backup, check the appropriate SQL options and enter the name for the backup file.

Model View Controller(MVC) in PHP

By admin in Patterns August 10, 2009 148 Comments

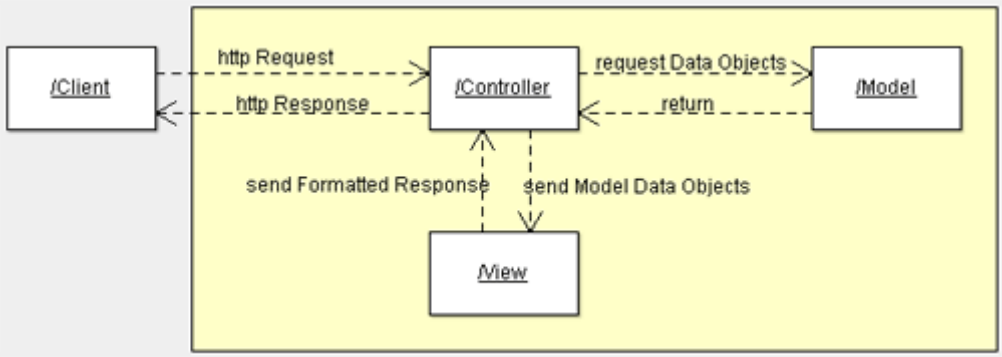
The model view controller pattern is the most used pattern for today‘s world web applications. It has been used for the first time in Smalltalk and then adopted and popularized by Java. At present there are more than a dozen PHP web frameworks based on MVC pattern.

Despite the fact that the MVC pattern is very popular in PHP, is hard to find a proper tutorial accompanied by a simple source code example. That is the purpose of this tutorial.

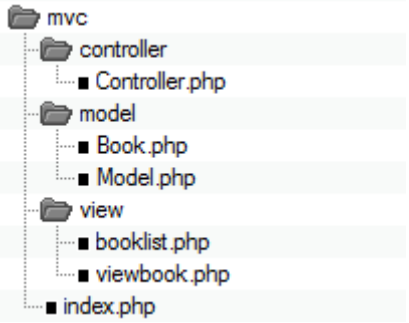
The MVC pattern separates an application in 3 modules: Model, View and Controller:

- The **model** is responsible to manage the data; it stores and retrieves entities used by an application, usually from a database, and contains the logic implemented by the application.
- The **view (presentation)** is responsible to display the data provided by the model in a specific format. It has a similar usage with the template modules present in some popular web applications, like wordpress, joomla, …
- The **controller** handles the model and view layers to work together. The controller receives a request from the client, invokes the model to perform the requested operations and sends the data to the View. The view formats the data to be presented to the user, in a web application as an html output.

The above figure contains the MVC Collaboration Diagram, where the links and dependencies between figures can be observed:



Our short php example has a simple structure, putting each MVC module in one folder:



Controller

The controller is the first thing which takes a request, parses it, initializes and invoke the model and takes the model response and sends it to the presentation layer. It‘s practically the liant between the Model and the View, a small framework where Model and View are plugged in. In our naive php implementation the controller is implemented by only one class, named unexpectedly controller. The application entry point will be index.php. The index php file will delegate all the requests to the controller:

view plaincopy to clipboardprint?

```
1. // index.php file
2. include_once("controller/Controller.php");
3.
4. $controller = new Controller();
```

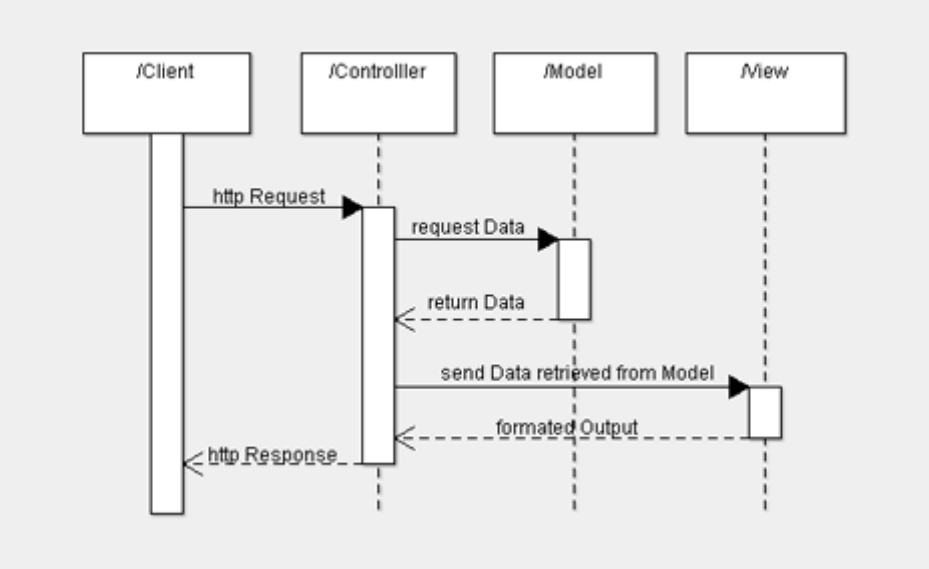
```
5.  $controller->invoke();
```

Our Controller class has only one function and the constructor. The constructor instantiates a model class and when a request is done, the controller decides which data is required from the model. Then it calls the model class to retrieve the data. After that it calls the corresponding passing the data coming from the model. The code is extremely simple. Note that the controller does not know anything about the database or about how the page is generated.

view plaincopy to clipboardprint?

```
1. include_once("model/Model.php");
2.
3. class Controller {
4.       public $model;
5.
6.       public function  construct()
7.       {
8.           $this->model = new Model();
9.       }
10.
11.      public function invoke()
12.      {
13.          if (!isset($_GET['book']))
14.          {
15.              // no special book is requested, we'll show a list of all available books
16.              $books = $this->model->getBookList();
17.              include 'view/booklist.php';
18.          }
19.          else
20.          {
21.              // show the requested book
22.              $book = $this->model->getBook($_GET['book']);
23.              include 'view/viewbook.php';
24.          }
25.      }
26. }
```

27. In the following MVC Sequence Diagram you can observe the flow during a http request:



Model and Entity Classes

The Model represents the data and the logic of an application, what many calls business logic. Usually, it‘s responsible for:

- storing, deleting, updating the application data. Generally it includes the database operations, but implementing the same operations invoking external web services or APIs is not an unusual at all.
- encapsulating the application logic. This is the layer that should implement all the logic of the application. The most common mistakes are to implement application logic operations inside the controller or the view(presentation) layer.

In our example the model is represented by 2 classes: the ‒Model‖ class and a ‒Book‖ class. The model doesn‘t need any other presentation. The ‒Book‖ class is an entity class. This class should be exposed to the View layer and represents the format exported by the Model view. In a good implementation of the MVC pattern only entity classes should be exposed by the model and they should not encapsulate any business logic. Their solely purpose is to keep data. Depending on implementation Entity objects can be replaced by xml or json chunk of data. In the above snippet you can notice how Model is returning a specific book, or a list of all available books:

view plaincopy to clipboardprint?

```
1. include_once("model/Book.php");
2.
3. class Model {
4.     public function getBookList()
5.     {
6.         // here goes some hardcoded values to simulate the database
7.         return array(
8.             "Jungle Book" => new Book("Jungle Book", "R. Kipling", "A classic book."),
9.             "Moonwalker" => new Book("Moonwalker", "J. Walker", ""),
10.            "PHP for Dummies" => new Book("PHP for Dummies", "Some Smart Guy", "")
11.        );
12.    }
13.
14.    public function getBook($title)
```

```
15.     {
16.         // we use the previous function to get all the books and then we return the requested one.
17.         // in a real life scenario this will be done through a db select command
18.         $allBooks = $this->getBookList();
19.         return $allBooks[$title];
20.     }
21.
22.
23. }
```

In our example the model layer includes the Book class. In a real scenario, the model will include all the entities and the classes to persist data into the database, and the classes encapsulating the business logic.

view plaincopy to clipboardprint?

```
1. class Book {
2.     public $title;
3.     public $author;
4.     public $description;
5.
6.     public function construct($title, $author, $description)
7.     {
8.         $this->title = $title;
9.         $this->author = $author;
10.        $this->description = $description;
11.    }
12. }
```

View (Presentation)

The view(presentation layer)is responsible for formating the data received from the model in a form accessible to the user. The data can come in different formats from the model: simple objects(sometimes called Value Objects), xml structures, json, …

The view should not be confused to the template mechanism sometimes they work in the same manner and address similar issues. Both will reduce the dependency of the presentation layer of from rest of the system and separates the presentation elements(html) from the code. The controller delegates the data from the model to a specific view element, usually associated to the main entity in the model. For example the operation ‑display account‖ will be associated to a ‑display account‖ view. The view layer can use a template system to render the html pages. The template mechanism can reuse specific parts of the page: header, menus, footer, lists and tables, …. Speaking in the context of the MVC pattern

In our example the view contains only 2 files one for displaying one book and the other one for displaying a list of books.

viewbook.php

view plaincopy to clipboardprint?

```
1. <html>
2. <head></head>
3.
4. <body>
5.
6.     <?php
```

```
7.
8.          echo 'Title:' . $book->title . '<br/>';
9.          echo 'Author:' . $book->author . '<br/>';
10.         echo 'Description:' . $book->description . '<br/>';
11.
12.     ?>
13.
14. </body>
15. </html>
```

booklist.php

```
view plaincopy to clipboardprint?
1. <html>
2. <head></head>
3.
4. <body>
5.
6.       <table>
7.          <tbody><tr><td>Title</td><td>Author</td><td>Description</td></tr></tbody>
8.          <?php
9.
10.             foreach ($books as $title => $book)
11.             {
12.                 echo '<tr><td><a href="index.php?book='.$book->title.'">'.$book-
    >title.'</a></td><td>'.$book->author.'</td><td>'.$book->description.'</td></tr>';
13.             }
14.
15.         ?>
16.     </table>
17.
18. </body>
19. </html>
```

The above example is a simplified implementation in PHP. Most of the PHP web frameworks based on MVC have similar implementations, in a much better shape. However, the possibility of MVC pattern are endless. For example different layers can be implemented in different languages or distributed on different machines. AJAX applications can implements the View layer directly in Javascript in the browser, invoking JSON services. The controller can be partially implemented on client, partially onserver…

This post should not be ended before enumerating the advantages of Model View Controller pattern:

- the Model and View are separated, making the application more flexible.
- the Model and view can be changed separately, or replaced. For example a web application can be transformed in a smart client application just by writing a new View module, or an application can use web services in the backend instead of a database, just replacing the model module.
- each module can be tested and debugged separately.

PHP AND AJAX:

What is AJAX ?

- AJAX stands for **A**synchronous **Ja**vaScript and **X**ML. AJAX is a new technique for creating better, faster, and more interactive web applications with the help of XML, HTML, CSS and JavaScript.
- Conventional web application transmit information to and from the sever using synchronous requests. This means you fill out a form, hit submit, and get directed to a new page with new information from the server.
- With AJAX when submit is pressed, JavaScript will make a request to the server, interpret the results and update the current screen. In the purest sense, the user would
- never know that anything was even transmitted to the server.

PHP and AJAX Example

To clearly illustrate how easy it is to access information from a database using Ajax and PHP, we are going to build MySQL queries on the fly and display the results on "ajax.html". But before we proceed, let‘s do a bit of ground work first. Create a table using the following command.

NOTE: We are assuming you have sufficient privilege to perform following MySQLoperations

```
CREATE TABLE `ajax_example` (
    `name` varchar(50) NOT NULL,
    `age` int(11) NOT NULL,
    `sex` varchar(1) NOT NULL,
    `wpm` int(11) NOT NULL,
    PRIMARY KEY     (`name`)
)
```

Now dump the following data into this table using the following SQL statements:

```
INSERT INTO `ajax_example` VALUES ('Jerry', 120, 'm', 20);
INSERT INTO `ajax_example` VALUES ('Regis', 75, 'm', 44);
INSERT INTO `ajax_example` VALUES ('Frank', 45, 'm', 87);
INSERT INTO `ajax_example` VALUES ('Jill', 22, 'f', 72);
```

```
INSERT INTO `ajax_example` VALUES ('Tracy', 27, 'f', 0);

INSERT INTO `ajax_example` VALUES ('Julie', 35, 'f', 90);
```

Client Side HTML file

Now let‘s have our client side HTML file which is ajax.html and it will have following code

```
<html>

<body>

<script language="javascript" type="text/javascript"><!--

//Browser Support Code

function ajaxFunction(){

  var ajaxRequest;        // The variable that makes Ajax possible!


  try{

    //    Opera 8.0+, Firefox, Safari ajaxRequest = new
          XMLHttpRequest();

  }catch (e){

    //    Internet Explorer Browsers try{

          ajaxRequest = new ActiveXObject("Msxml2.XMLHTTP"); }catch
      (e) {

          try{

              ajaxRequest = new ActiveXObject("Microsoft.XMLHTTP"); }catch
          (e){

              //    Something went wrong

              alert("Your browser broke!");

              return false;

          }

      }

  }
```

```
    // Create a function that will receive data

    // sent from the server and will update

    // div section in the same page.
        ajaxRequest.onreadystatechange = function(){

      if(ajaxRequest.readyState == 4){

          var ajaxDisplay = document.getElementById('ajaxDiv');
          ajaxDisplay.innerHTML = ajaxRequest.responseText;

      }

    }

     // Now get the value from user and pass it to

    // server script.

    var age = document.getElementById('age').value;

    var wpm = document.getElementById('wpm').value;

    var sex = document.getElementById('sex').value;

    var queryString = "?age=" + age ;

    queryString +=  "&wpm=" + wpm + "&sex=" + sex;

    ajaxRequest.open("GET", "ajax-example.php" +
queryString, true);

    ajaxRequest.send(null);

  }

  //-->

  </script>

  <form name='myForm'>

  Max Age: <input type='text' id='age' /><br />

  Max WPM: <input type='text' id='wpm' />

  <br />

  Sex: <select id='sex'>
```

```
<option value="m">m</option>

<option value="f">f</option>

</select>

<input type='button' onclick='ajaxFunction()' value='Query
                                                MySQL'/>

</form>

<div id='ajaxDiv'>Your result will display here</div>

</body>

</html>
```

NOTE: The way of passing variables in the Query is according to HTTP standard and thehave formA

```
URL?variable1=value1;&variable2=value2;
```

The above code will produce a screen as given below:

NOTE: This is dummy screen.

Server Side PHP file

So now your client side script is ready. Now we have to write our server side script which will fetch age, wpm and sex from the database and will send it back to the client. Put the following code into "ajax-example.php" file

```
<?php

$dbhost = "localhost";
```

```
$dbuser = "dbusername";

$dbpass = "dbpassword";

$dbname = "dbname";

        //Connect to MySQL Server

mysql_connect($dbhost, $dbuser, $dbpass);

        //Select Database

mysql_select_db($dbname) or die(mysql_error());

        //    Retrieve data from Query String $age  =
        $_GET['age'];

$sex = $_GET['sex']; $wpm =
$_GET['wpm'];

   //    Escape User Input to help prevent SQL Injection $age =
        mysql_real_escape_string($age);

$sex = mysql_real_escape_string($sex); $wpm =
mysql_real_escape_string($wpm);

        //build query

$query = "SELECT * FROM ajax_example WHERE sex = '$sex'";
if(is_numeric($age))

        $query  .=  "  AND  age  <=  $age";
if(is_numeric($wpm))

        $query .= " AND wpm <= $wpm";
        //Execute query

$qry_result = mysql_query($query) or die(mysql_error());


        //Build Result String

$display_string = "<table>";

$display_string .= "<tr>";

$display_string .= "<th>Name</th>";

$display_string .= "<th>Age</th>";

$display_string .= "<th>Sex</th>";
```

```
$display_string .= "<th>WPM</th>";

$display_string .= "</tr>";


//    Insert a new row in the table for each person returned while($row =
mysql_fetch_array($qry_result)){

        $display_string .= "<tr>";

        $display_string      .=      "<td>$row[name]</td>";
        $display_string       .=       "<td>$row[age]</td>";
        $display_string       .=       "<td>$row[sex]</td>";
        $display_string      .=      "<td>$row[wpm]</td>";
        $display_string .= "</tr>";

}

echo "Query: " . $query . "<br />";

$display_string .= "</table>";

echo $display_string;

?>
```

Now enter a valid value in "Max Age" or any other box and then click Query MySQL button.

If you have successfully completed this lesson, then you know how to use MySQL, PHP, HTML, and Javascript in tandem to write Ajax applications.

PHP AND XML:

XML is a markup language that looks a lot like HTML. An XML document is plain text and contains tags delimited by < and >. There are two big differences between XML and HTML:

- XML doesn't define a specific set of tags you must use.
- XML is extremely picky about document structure.

XML gives you a lot more freedom than HTML. HTML has a certain set of tags: the <a> tags surround a link, the <p> starts a paragraph and so on. An XML document, however, can use any tags you want. Put <rating></rating> tags around a movie rating, <height></height> tags around someone's height. Thus XML gives you option to device your own tags.

XML is very strict when it comes to document structure. HTML lets you play fast and loose with some opening and closing tags. But this is not the case with XML.

HTML list that's not valid XML

```
<ul>
<li>Braised Sea Cucumber
<li>Baked Giblets with Salt
<li>Abalone with Marrow and Duck Feet
</ul>
```

This is not a valid XML document because there are no closing tags to match up with the three opening <li> tags. Every opened tag in an XML document must be closed.

HTML list that is valid XML

```
<ul>
<li>Braised Sea Cucumber</li>
<li>Baked Giblets with Salt</li>
<li>Abalone with Marrow and Duck Feet</li>
</ul>
```

Parsing an XML Document

PHP 5's new **SimpleXML** module makes parsing an XML document, well, simple. It turns an XML document into an object that provides structured access to the XML.

To create a SimpleXML object from an XML document stored in a string, pass the string to **simplexml_load_string( )**. It returns a SimpleXML object.

Example

Try out the following example:

```
<html>
   <body>

      <?php
         $note=<<<XML

         <note>
            <to>Gopal K Verma</to>
            <from>Sairamkrishna</from>
            <heading>Project submission</heading>
            <body>Please see clearly </body>
         </note>

         XML;
         $xml=simplexml_load_string($note);
         print_r($xml);
      ?>
   </body>
</html>
```

It will produce the following result:

```
SimpleXMLElement Object ( [to] => Gopal K Verma [from] => Sairamkrishna [heading]
=> Project submission [body] => Please see clearly )
```

NOTE: You can use function **simplexml_load_file(filename)** if you have XML contentin a file.

Generating an XML Document

SimpleXML is good for parsing existing XML documents, but you can't use it to create a new one from scratch.

The easiest way to generate an XML document is to build a PHP array whose structure mirrors that of the XML document and then to iterate through the array, printing each element with appropriate formatting.

Example

Try out the following example:

```
<?php
$channel = array('title' => "What's For Dinner", 'link' => 'http://menu.example.com/',
                        'description' => 'Choose what to eat tonight.');
print "<channel>\n";
foreach ($channel as $element => $content) { print "
    <$element>";
    print htmlentities($content);
    print "</$element>\n";
}
print "</channel>";
?>
```

It will produce the following result:

```
<channel>
<title>What's For Dinner</title>
<link>http://menu.example.com/</link>
<description>Choose what to eat tonight.</description>
</channel>
</html>
```