Web Technologies Notes

Contributor: Abhishek

[KMV (DU)]

Computer Science Notes

UNIT-I

Introduction to HTML:-

HTML means Hypertext Markup Language. In 1960 Ted Nelson introduced Hypertext. HTML is a scripting language which is used to create web pages. If you are thinking of creating your own web pages, you need to know at least basic HTML. These HTML documents are plain text files, user can create these documents using text editor like Notepad or Edit.

HTML is a hypertext Language because it supports font styled text, pictures, graphics and animations and also it provides hyper links that used to browse the Internet easily. Text becomes hypertext with the addition of links that connects other hypertext documents. Hypertext is a text augmented with links-pointers to other pieces of text, possible else where in the same document (internal linking) or in another document (external linking).

Rules to write HTML Code:-

- Every HTML document begins with start tag is <HTML> terminates with an ending tag is </HTML>
- HTML documents should be saved with the extension .html or .htm.
- A tag is made up of left operator(<), a right operator(>) and a tag name between these two operators.
- If you forget to mention the right operator(>) or if you give any space between left operator and tag name browser will not consider it as tag.
- At the same time if browser not understands the tag name it just ignores it, browser won‟t generate any errors.
- HTML language is not case sensitive, hence user can write the code in either upper case or lower case. No difference between <HTML> and <html>

Syntax of a tag:

<Tagname [parameters=value]>

Ex: HR is a tag name that displays a horizontal ruler line.
<HR> ----------(No parameters, no value)
<HR ALIGN=CENTER>----------(Tag with parameter and value for the parameter)
<HR WIDTH=”30%” SIZE=100 ALIGN=RIGHT>------(Tag with more parameters with their values)

Different types of Tags:

1. Singleton tags
2. Paired tags

Singleton tag does not require an ending tag. (Ex: <HR>
Paired tag required an ending tag, which is similar to opening tag except backslash before the tag name (Ex: <HTML> is opening tag, then ending tag is </HTML>

Comments in HTML:

An HTML comment begins with “<!--“ and ends wit<h “-->”.There should not be a space between angular bracket and exclamation mark.

Creating HTML Page:

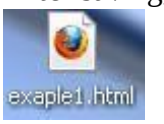
The Following steps are needed to create a HTML page
Step 1: Open any text editor like Notepad, Edit, Word etc.

Step 2: Use the file menu to create a new document (File New) and type the following code

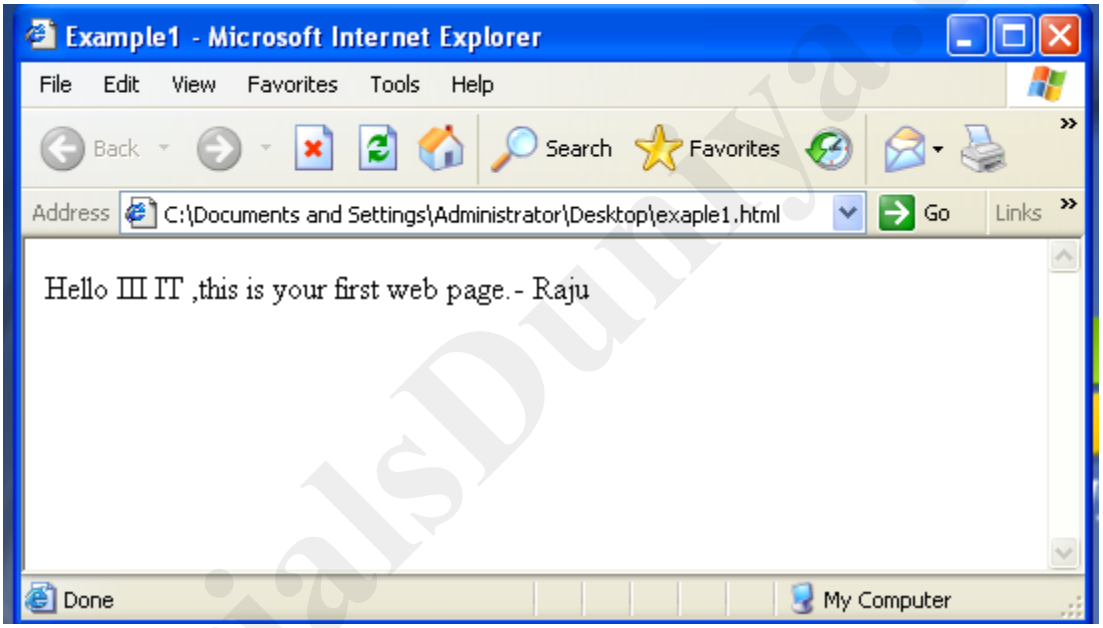
```
<HTML>
<HEAD>
<TITLE>Example1 </TITLE>
<BODY>
        Hello III IT ,this is your first web page.- Raju
</BODY>
</HTML>
```

Step 3: Go to the file menu and choose saveas option (File->saveas) and give the name of the file as "example1.html" under root directory(C:)(or any valid path)

Step 4: After saving, an internet explorer icon will be displayed as shown below



Step 5: Double click to execute it. The output displayed following



Basic HTML tags

1. Body tag:-

Body tag contain some attributes such as bgcolor, background etc. bgcolor is used for background color, which takes background color name or hexadecimal number and #FFFFFF and background attribute will take the path of the image which you can place as the background image in the browser.

<body bgcolor="#F2F3F4" background= "c:\amer\imag1.gif">

2. Paragraph tag:-

Most text is part of a paragraph of information. Each paragraph is aligned to the left, right or center of the page by using an attribute called as align.

<p align="left" | "right" | "center">

3. Heading tag:-

HTML is having six levels of heading that are commonly used. The largest heading tag is

<h1>. The different levels of heading tag besides <h1> are <h2>,<h3>, <h4>, <h5> and <h6>. These heading tags also contain attribute called as align.

<h1 align=”left” | “right” | “center”> .....<h2>

4. hr tag:-
This tag places a horizontal line across the system. These lines are used to break the page. This tag also contains attribute i.e., width which draws the horizontal line with the screen size of the browser. This tag does not require an end tag.

<hr width=”50%”>.

5. base font:-
This specify format for the basic text but not the headings.

<basefont size=”10”>

6. font tag:-
This sets font size, color and relative values for a particular text.

<font size=”10” color=”#f1f2f3”>

7. bold tag:-
This tag is used for implement bold effect on the text

 .....

8. Italic tag:-
This implements italic effects on the text.

… ....

9. strong tag:-
This tag is used to always emphasized the text

.....

10. tt tag:-
This tag is used to give typewriting effect on the text

<tt> .....</tt>

11. sub and sup tag:-
These tags are used for subscript and superscript effects on the text.

_{.....}
^{.....}

12. Break tag:-
This tag is used to the break the line and start from the next line.

13. &amp &lt &gt   &quot:-

These are character escape sequence which are required if you want to display characters that HTML uses as control sequences.

Example: < can be represented as &lt.

14. Anchor tag:-
This tag is used to link two HTML pages, this is represented by <a>
 some text
href is an attribute which is used for giving the path of a file which you want to link.

Example 1: HTML code to implement common tags.
mypage.html

```
<html>
<head> <! -- This page implements common html tags -->
<title> My Home page </title>
</head>
<body >
<h1 align="center"> ST ANN"S COLLEGE OF ENGINEERING & TECHNOLOGY</h1>
<h2 align="center"> Chirala</h2>
<basefont size=4>
<p> This college runs under the <tt>management</tt> of <font size=5> <b><i>&quot
SACET&quot &amp </i></b></font><br>
  it is affiliated to <strong> JNTUK</strong>
<hr size=5 width=80%>
<h3> <u>&lt Some common tags &gt</u> </h3><br>
</body>
</html>
```

Text Styles or Cosmetic tags:- HTML provides a numerous range of tags for formatting the text. If you want to format the text with different styles, just you include these tags one by one before text.

<B>……</B>	Bold Text
<U>……...</U>	Underline Text
<I>……..</I>	Displays as Italics
<EM>……</EM>	For Emphasis (New Standard for Italics)
<STRONG>……</STRONG>	Strong or Bold text (New Standard for Bold)
<S>……..</S> or <DEL>…….</DEL>	Strikes the text
<SAMP>……</SAMP>	Code sample text
<VAR>……..</VAR>	Small fonts, fixed width
<ADDRESS>……..</ADDRESS>	Like address model (Looks like italics)
<PRE>……..</PRE>	Considers spaces, new lines etc. As it is prints the information

Scrolling Text Tag:-

<marquee> ..... </marquee> Displays scrolling text in a marquee style.

Marquee tag attributes:-

a) align: sets the alignment of the text relative to the marquee. Set to top(default), middle or bottom.

b) behaviour: Sets how the text in the marquee should move, It can be scroll(default), slide(text enters from one side and stops at the other end), or alternative(text seems to bounce from one side to other)
c) bgcolor: sets the background color for the marquee box
d) direction: sets the direction of the text for scrolling. It can be left(default), right, down or up.

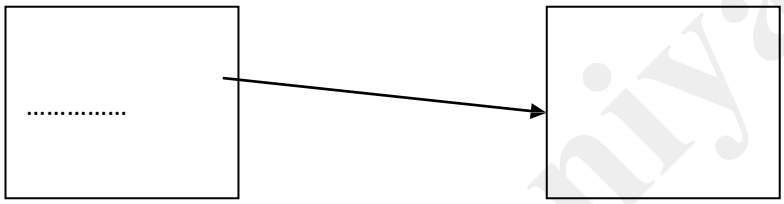
Example:-
<marquee align=”middle” behavior=”slide”bgcolr=”red”direction=”down”>Raju</marquee>

Blinking text Tag:-

<blink>……</blank> displays enclosed text as blinking on and off approximately once a second.

Linking in HTML:-

Text becomes hypertext with the addition of links which connect separate locations with in a collection of hypertext documents.



Hyperlinks can be applied for either text or images.Links may connect several web pages of a web site.Links can connect web pages on the same or different servers. Navigation between pages became easier because of links. Information in the same page also connected through links(internal links).

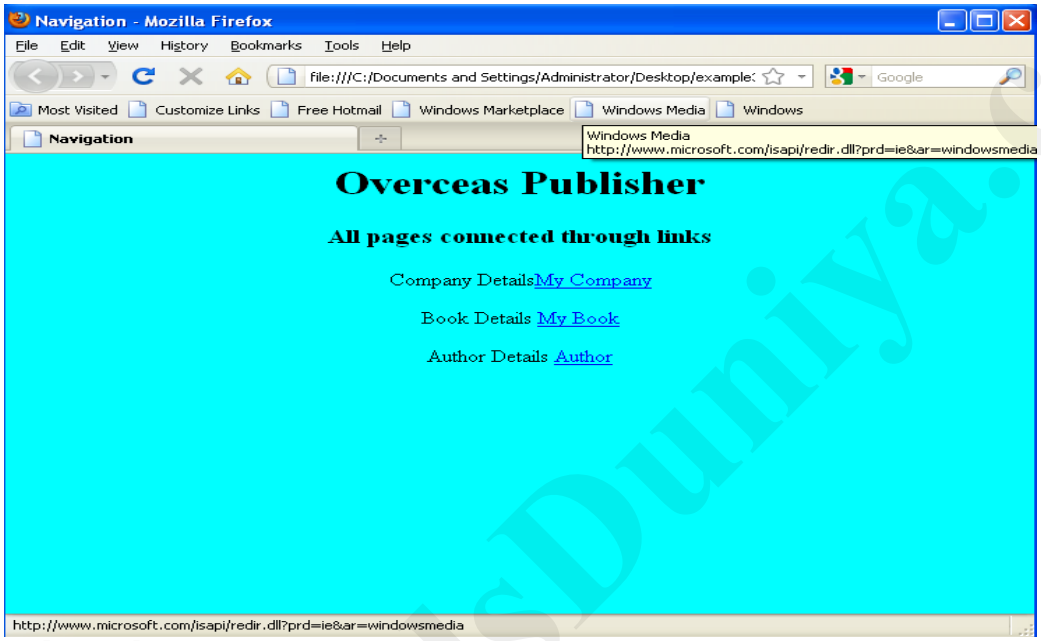
<A> ..... </A>

Anchor tag is used for creating links. Minimum it requires a parameter i.e., HREF, which indicates the destination document. Other parameters Name and target can be useful for identification for anchor tag and the location of a frame where target page is to be displayed respectively. Name and target tag are optional.
Syntax: <A HREF=”address” NAME=”id“ TARGET=”target window”>
…….
…….. id=Text that displays as link
</A>

HREF Parameter:-

If HREF is included, the text between the opening and closing anchor element that between <A> and </A> becomes hyper text. If users clicks on this text, they are moved to specified document.
Ex:-
<A HREF=www.vaneshdoc.com>Publishers</A>
Result displayed as Publishers
When the user click on this text, Publishers web site is displayed on the browser.
Example :
Create a HTML web page that connect web pages created through hyperlinks.
<html>

```
<head>
<title> Navigation </title>
</head>
<body bgcolor=cyan>
<h1 align=center>Overceas Publisher </h1>
<h3 align=center>All pages connected through links</h3>
<center>
<p>Company Details<A HREF=”1.html”>My Company</A>
<p>Book Details <A HREF=”2.html”>My Book</A>
<p> Author Details <A HREF=”3.html”> Author</A>
</center>
</body>
</html>
```



Color and Image:

Color can be used for background, elements and links. To change the color of links or of the page background hexadecimal values are placed in the <body> tag.

<body bgcolor = “#nnnnnn” text = “#nnnnnn” link= “#nnnnnn” vlink= “#nnnnnn” alink = “#nnnnnn”>

The vlink attribute sets the color of links visited recently, alink the color of a currently active link. The six figure hexadecimal values must be enclosed in double quotes and preceded by a hash(#).

Images are one of the aspect of web pages. Loading of images is a slow process, and if too many images are used, then download time becomes intolerable. Browsers display a limited range of image types.

<body background = “URL”>

This tag will set a background image present in the URL.

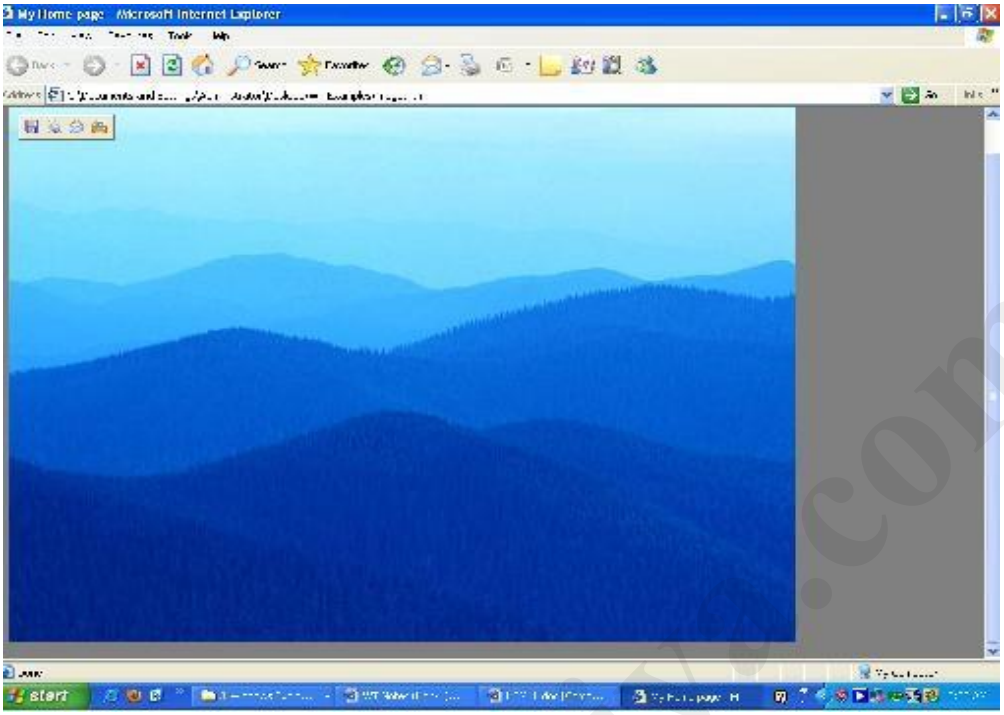
Another tag that displays the image in the web page, which appears in the body of the text rather than on the whole page is given below

<img src=”URL” height=”n” width=”n” align = “top” | “center” | “bottom” >

Example 4: HTML code that implements color and image

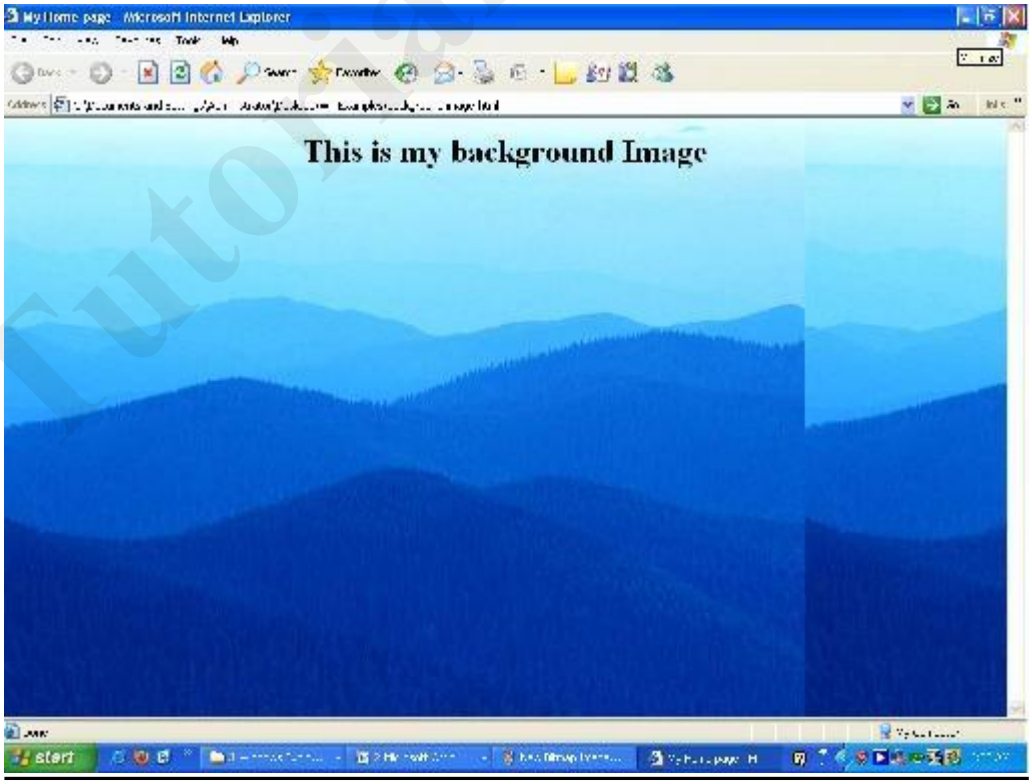
```
<html>
<head> <! -- This page implements color and image -->
<title> My Home page </title>
</head>
<body bgcolor="gray" text="magneta" vlink="yellow" alink="brown">
<img src= ” C:\Documents and Settings\All Users\Documents\My
```

Pictures\Sample Pictures\Blue hills.jpg”>
</body>
</html>



Example 5: HTML code that implements background image

```
<html>
<head> <! -- This page implements background image -->
<title> My Home page </title>
</head>
<body background="C:\Documents and Settings\All Users\Documents\My Pictures\Sample Pictures\Blue hills.jpg">
<h1 align="center"> This is my background Image</h1>
</body>
</html>
```



Lists:

One of the most effective ways of structuring a web site is to use lists. Lists provides straight forward index in the web site. HTML provides three types of list i.e., bulleted list, numbered list and a definition list. Lists can be easily embedded easily in another list to provide a complex but readable structures. The different tags used in lists are explained below.

Unordered Lists:-

Unordered lists are also called unnumbered .lists. The Unordered list elements are used to represent a list of items, which are typically separated by white space and/or marked by bullets. Using <UL> tag does creation of unordered lists in HTML. Which is paired tag, so it requires ending tag that is </UL>. The list of items are included in between <UL>…..</UL>. The TYPE attribute can also be added to the <UL> tag that indicates the displayed bullet along with list of item is square, disc or circle. By default it is disc.

Syntax:-

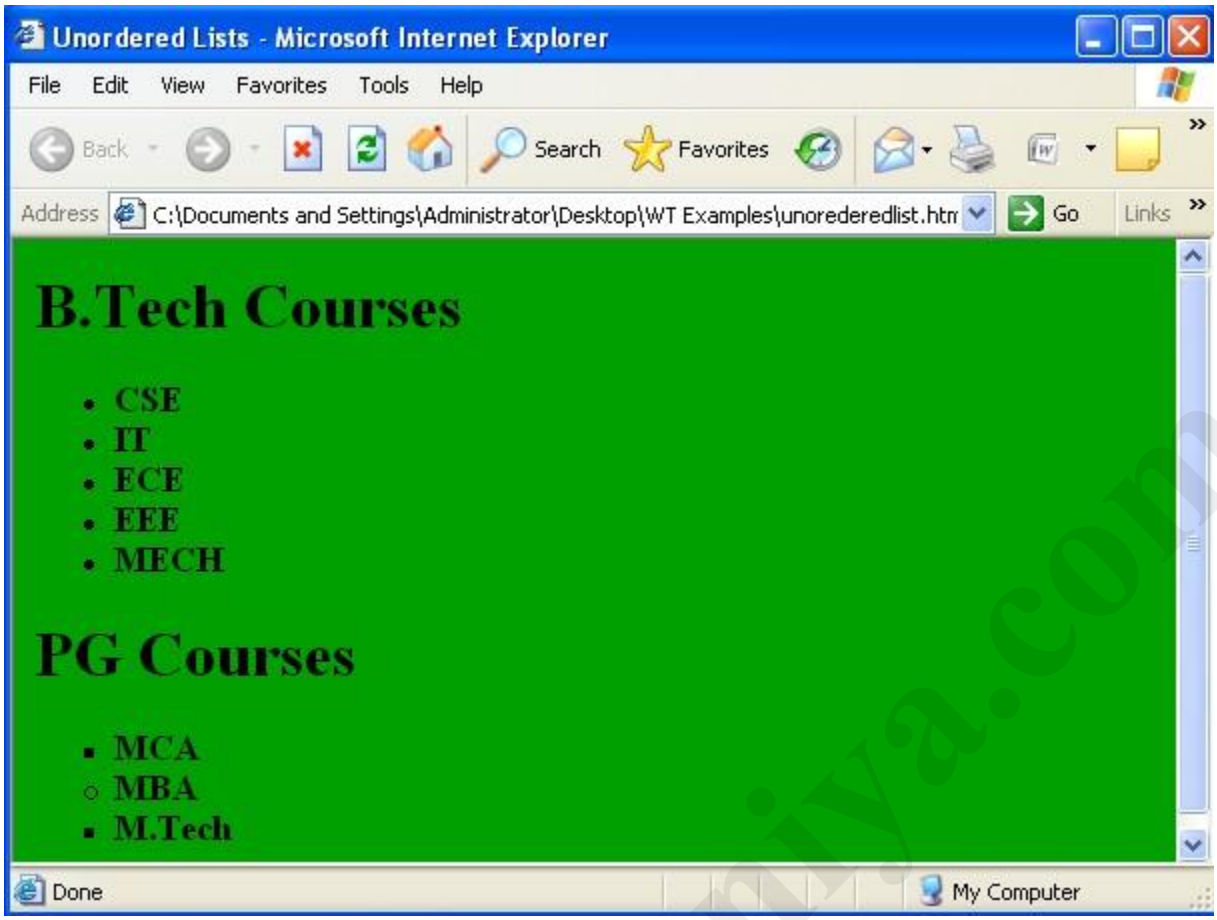
```
<UL [TYPE={square/disc/circle}]>
    <LI>item name1
    <LI>item name2
    ……..
    ----------
    <LI>item namen
</UL>
```

Example:

Write a HTML program for displaying names of B.Tech Courses with default bullets and names of PG Courses with square bullets.

```
<html>
    <head>
        <title>Unordered Lists</title>
    </head>
<body bgcolor=”tan”>

    <h1>B.Tech Courses
        <h3>
<ul>
    <li>CSE
    <li>IT
    <li>ECE
    <li>EEE
    <li>MECH
</ul>
    </h3>
<h1>PG Courses
    <h3>
        <ul type=”square”>
            <li>MCA
            <li type=”circle”>MBA
            <li>M.Tech
        </ul>
    </h3>
  </body>
</html>
```



Ordered Lists:-

Ordered lists are also called sequenced or **numbered lists**. In the ordered list the list of item have an order that is signified by numbers, hence it some times called as number lists. Elements used to present a list of items, which are typically separated by white space and/or marked by numbers or alphabets. An orders list should start with the <OL> element, which is immediately followed by a <LI> element which is same as <LI> in unordered list. End of ordered lists is specified with ending tag </OL>.

Different Ordered list types like roman numeral list, alphabet list etc. can be specified with TYPE tag. Another optional parameter with <OL> tag is START attribute, which indicates the starting number or alphabet of the ordered list. For example TYPE=”A” and START=5 will give list start with letter E. The TYPE attribute used in <LI>, changes the list type for particular item. To give more flexibility to list, we can use VALUE parameter with <LI>tag that helps us to change the count for the list item and subsequence items.

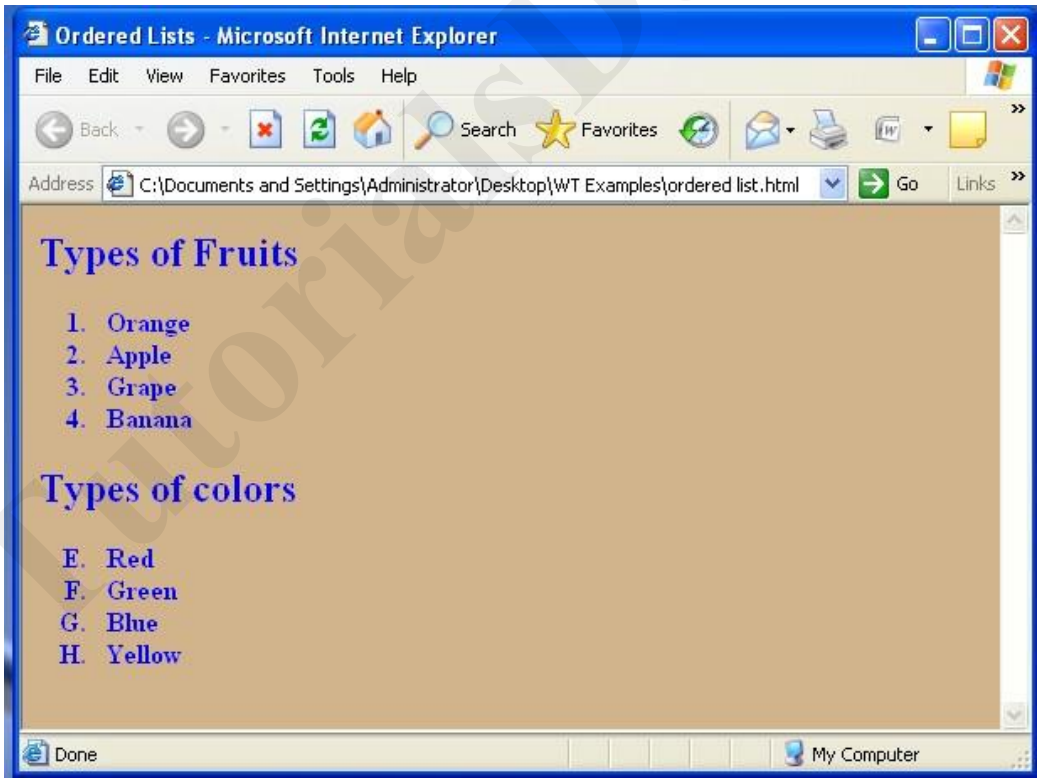
Syntax:-
<OL [type={“1” or “I” or “a” or”A” or “i”}] START=n>
<LI>item name1
<LI>item name2

<LI>item namen
</OL>
Different Ordered list types
Type=”1” (default) e.g.1,2,3,4…..
Type=”A” Capital letters e.g.A,B,C…
Type=”a” Small letters e.g. a,b,c……

Type=”I” Large roman letters e.g. I, II, III,…

Example:-

```
<html>
<head>
<title>Ordered Lists</title>
</head>
<body bgcolor=”tan” text=”blue”>
<h2> Types of Fruits
<h4>
        <OL>
                <LI>Orange
                <LI>Apple
                <LI>Grape
                <LI>Banana
        </OL>
</h4>
<h2>Types of colors
<h4>
        <OL type=”A” START=5>
                <LI>Red
                <LI>Green
                <LI>Blue
                <LI>Yellow
        </OL>
</h4>
</body>
</html>
```



Other Lists:-

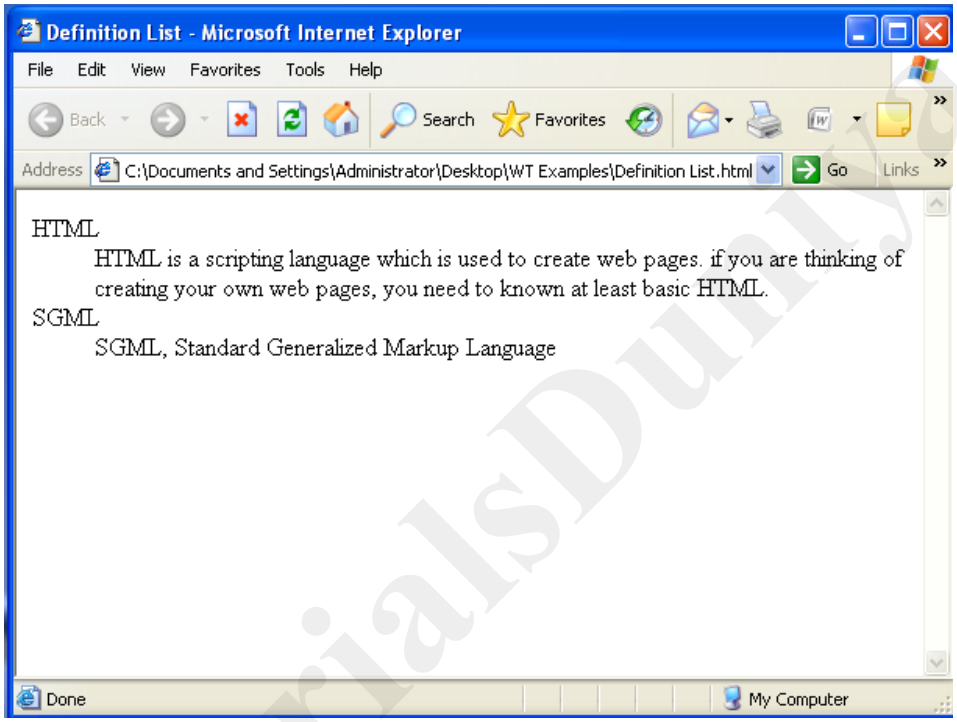
There are several lists in HTML, some of them are definition list and Directory List.

Definition List:- <DL>……</DL>

A Definition list is alist of definition terms <DT> and corresponding Difinition Description<DD> on anew line. To create a definition it must start with <DL> and immediately followed by the first definition term <DT>

Example:-

```
<html>
    <head>
        <title>Definition List</title>
    <body>
        <DL>
        <DT>HTML
        <DD> HTML is a scripting language which is used to create web pages. if you are thinking of creating your own web pages, you need to known at least basic HTML.
        <DT>SGML
        <DD>SGML, Standard Generalized Markup Language
        </DL>
    </body>
</html>
```



Directory List:-

A Directory list element is used to present a list of items containing up to 20 characters each. Items in a Directory List may be arranged in columns, typically 24 characters wide. A Directory List being with <DIR>element,which is immediately followed by a <LI>element. This tag is a deprecated tag, so it is not preferable to use. Hence,use <UL> instead of <DIR>

```
        Other information
            <DIR>
                <LI>Contacts-2043240
                <LI>Business-4123412
                <LI>Personal-3123122
            </DIR>
```

Nested Lists:- Lists can be nested that is Nested Lists is list with in another list.

Tables:

Table is one of the most useful HTML constructs. Tables are find all over the web application. The main use of table is that they are used to structure the pieces of information and to structure the whole web page. Below are some of the tags used in
table.

<table align=”center” | “left” | “right” border=”n” width=”n%” cellpadding=”n” cellspacing=”n”>
……
</table>

Every thing that we write between these two tags will be within a table. The attributes of the table will control in formatting of the table. Cell padding determines how much space there is between the contents of a cell and its border, cell spacing sets the amount of white space between cells. Width attribute sets the amount of screen that table will use.

<tr> ….. </tr>

This is the sub tag of <table> tag, each row of the table has to be delimited by these tags.

<th>……</th>

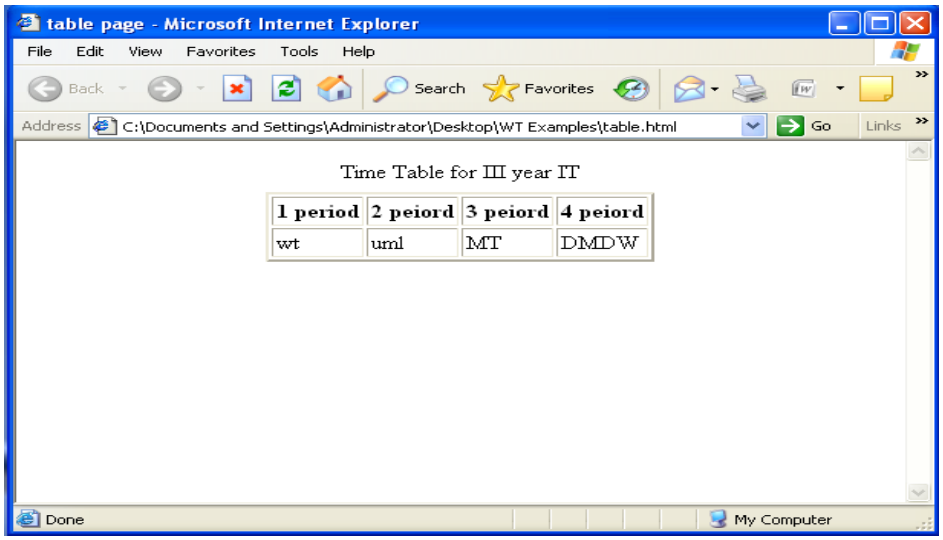
This is again a sub tag of the <tr> tag. This tag is used to show the table heading .

<td>…..</td>

This tag is used to give the content of the table.

Example 3: HTML code showing the use of table tag

```
<html>
<head>
<title> table page</title>
</head>
<body>
<table align="center" cellpadding="2" cellspacing="2" border="2">
<caption> Time Table for III year IT </caption>
<tr><th> 1 period </th>
<th> 2 peiord </th>
<th> 3 peiord </th>
<th> 4 peiord </th>
</tr>
<tr>
<td> wt </td>
<td> uml</td>
<td> MT</td>
<td> DMDW</td>
</tr>
</table>
</body>
</html>
```



Complex HTML Tables and Formatting:-

You can add background color and background images by using bgcolor and background attributes respectively. Spanning of cells is possible that is you can merge some sequence of rows or columns with the help of ROWSPAN or COLSPAN attributes respectively. For example <th COLSPAN=”2”>widened to span two cells. VALIGN attribute is used for vertical alignment formats and it accepts the values “top”, “middle”, “bottom” and “baseline”.

Example:

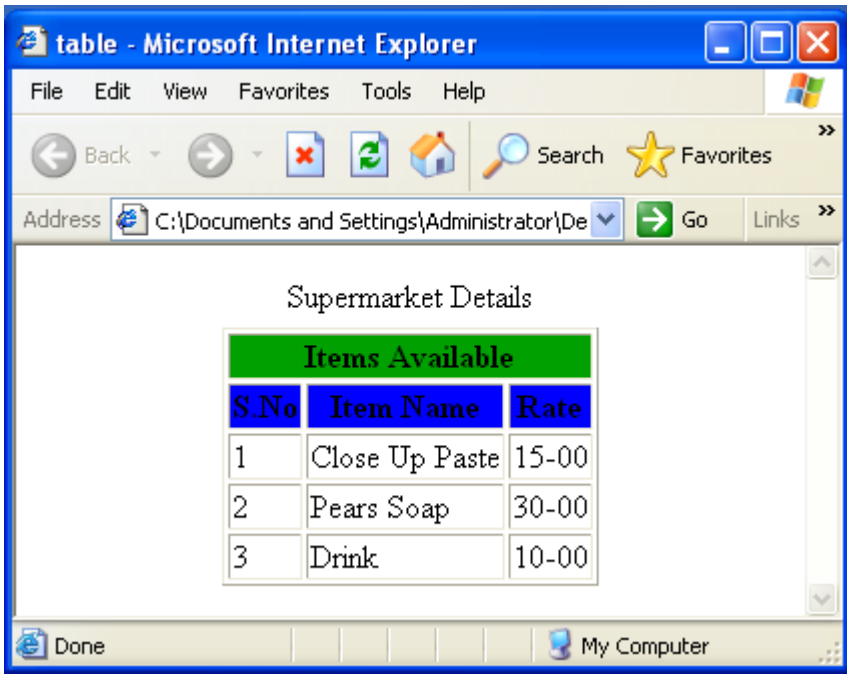
```
<html>
    <head>
        <title> table</title>
    </head>
    <body>
        <center>
            <table border=”2”>
            <caption>Supermarket Details</caption>
            <tr>
            <th colspan=3 bgcolor=”tan” align=”center”>Items Available</th>
            </tr>
            <tr><th>S.No<th>Item Name<th>Rate</tr>
            <tr><td>1<td>Close Up Paste<td>15-00</tr>
            <tr><td>2<td>Pears Soap<td>30-00</tr>
            <tr><td>3<td>Drink<td>10-00</tr>
            </table>
        </center>
    </body>
</html
```

TutorialsDuniya.com

Download FREE Computer Science Notes, Programs, Projects, Books PDF for any university student of BCA, MCA, B.Sc, B.Tech CSE, M.Sc, M.Tech at https://www.tutorialsduniya.com

- Algorithms Notes
- Artificial Intelligence
- Android Programming
- C & C++ Programming
- Combinatorial Optimization
- Computer Graphics
- Computer Networks
- Computer System Architecture
- DBMS & SQL Notes
- Data Analysis & Visualization
- Data Mining
- Data Science
- Data Structures
- Deep Learning
- Digital Image Processing
- Discrete Mathematics
- Information Security
- Internet Technologies
- Java Programming
- JavaScript & jQuery
- Machine Learning
- Microprocessor
- Operating System
- Operational Research
- PHP Notes
- Python Programming
- R Programming
- Software Engineering
- System Programming
- Theory of Computation
- Unix Network Programming
- Web Design & Development

Please Share these Notes with your Friends as well



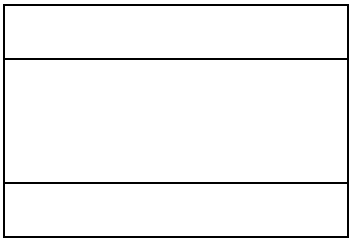
Frames:

Frames provide a pleasing interface which makes your web site easy to navigate. When we talk about frames actually we are referring to frameset, which is a special type of web page.

Simply frameset is nothing but collection of frames. Web page that contains frame element is called framed page. Framed page begins with <frameset> tag and ends with </frameset>. Each individual frame is identified through <frame> tag. Creation of framed page is very simple. You can nest the framesets. First you decide how you want to divide your webpage and accordingly define frame elements.

Consider the following diagrams, first form divides into two columns and the second form divides into three rows.

Two columns frameset



Three rows frameset

In order to divide into two columns we can use the following syntax

<frameset cols=”25%,75%>
<frame name=”disp” src=”1.html”>
<frame name=”res” src=”2.html”>
</frameset>

In the second diagram we have three rows so by using rows parameter of frameset, we can divide logically the window into three rows.

<frameset rows=”20%,*,10%>
<frame name=”first” src=”1.html”>

```
<frame name="second" src="2.html">
<frame name="third" src="3.html">
</frameset>
```

According to above code ,first row occupies 20% of the window, third row occupies 10% of the window, second row * represents remaining area that is 70% of the window.

Nested Framesets:-

Some times it is required to divide your window into rows and columns, then there is requirement of nested framesets. Frameset with in another frameset is known as nested frameset.

The purpose of NAME parameter in frame tag in the above example is nothing but main importance is if we have some links in left side and you want to display respective pages in the right side frame, then name is essential. Using target parameter of Anchor(A) tag as follows users can specify name of frame.

Example:

First.html

```
<frameset rows="20%,*">
    <frame name="fr1" src="frame1.html">
    <frameset cols="25%,*">
        <frame name="fr2" src="frame2.html">
        <frame name="fr3" src="frame3.html">
    </frameset>
</frameset>
```

Frame1.html

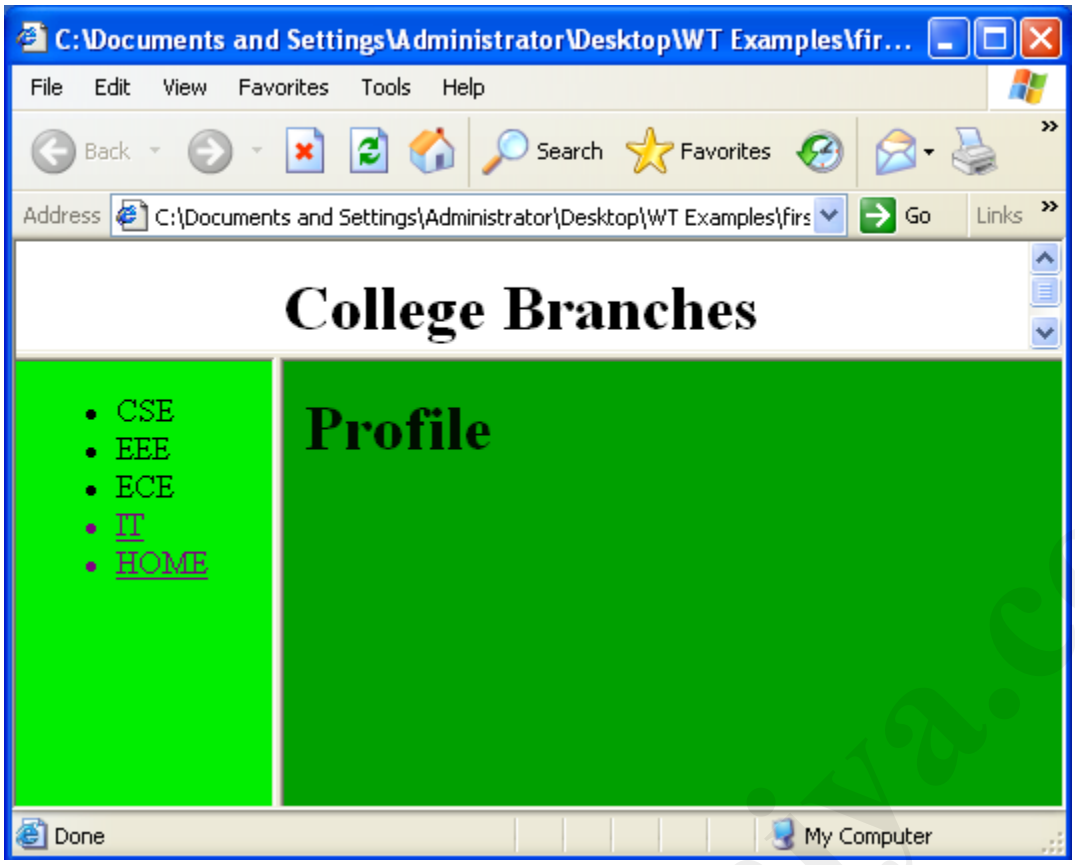
```
<html>
    <body>
        <center><h1> College branches</h1></center>
    </body>
</html>
```

Frame2.html

```
<html>
    <body bgcolor="green">
        <ul>
            <li>CSE
            <li>EEE
            <li>ECE
            <A href="example2.html" target="fr3"><li>IT</A>
        </ul>
    </body>
</html>
```

Frame3.html

```
<html>
    <body text="white" bgcolor="tan">
    <h1>Profile</h1>
    </body>
</html>
```



Forms:

Forms are the best way of adding interactivity of element in a web page. They are usually used to let the user to send information back to the server but can also be used to simplify navigation on complex web sites. The tags that use to implement forms are as follows.

<forms action="URL" method = "post" | "get">…….</form>

When get is used, the data is included as part of the URL. The post method encodes the data within the body of the message. Post can be used to send large amount of data, and it is more secure than get. The tags used inside the form tag are:

<input type = "text" | "password" | "checkbox" | "radio" | "submit" name="string" value="string" size="n">

In the above tag, the attribute type is used to implement text, password, checkbox, radio and submit button.

Text: It is used to input the characters of the size n and if the value is given than it is used as a default value. It uses single line of text. Each component can be given a separate name using the name attribute.

Password: It works exactly as text, but the content is not displayed to the screen, instead an * is used.

Radio: This creates a radio button. They are always grouped together with a same name but different values.

Checkbox: It provides a simple checkbox, where all the values can be selected unlike radio button.

Submit: This creates a button which displays the value attribute as its text. It is used to send the data to the server.

<select name=”string”>…..</select>

This tag helps to have a list of item from which a user can choose. The name of the particular select tag and the name of the chosen option are returned.

<option value=”string” selected>……</option>

The select statement will have several options from which the user can choose. The values will be displayed as the user moves through the list and the chosen one returned to the server.

<textarea name=”string” rows=”n” cols=”n”>…….</textarea>

This creates a free format of plain text into which the user can enter anything they like. The area will be sized at rows by cols but supports automatic scrolling.

Example 6: HTML code that implements forms

```
<html>
<head>
<title>form</title>
</head>
<body>
<p align="left">Name:<input type="text" maxlength=30 size=15>
<p align="left">Password:<input type="password" maxlenght=10 size=15>
<p align="left">Qualification: <br><input type="checkbox" name="q"
value="be">B.E
<input type="checkbox" name="q" value="me">M.E
<p align="left">Gender:<br> <input type="radio" name="g" value="m">Male
<input type="radio" name="g" value="f">Female
<p align="left">course:<select name="course" size=1>
<option value=cse selected>CSE
<option value=it>CSIT
</select>
<p align="left">Address:<br><textarea name="addr" rows=4 cols=5
scrolling=yes></textarea>
<p align="center"><input type="submit" name="s" value="Accept">
<p align="center"><input type="reset" name="c" value="Ignore">
</body>
</html>
```

Cascading Style Sheet:-

CSS stands for Cascading Style Sheets. It not only extends its features in controlling colors and sizes of fonts, but also controls spaces between various elements, the color and width of a given line etc.

Syntax: p {font-size:30pt;}
p =selection
Font-size =property
30pt =value

Types of Style Sheets:-

1. Inline Style sheet
2. Embedded Style Sheet
3. External Style Sheet

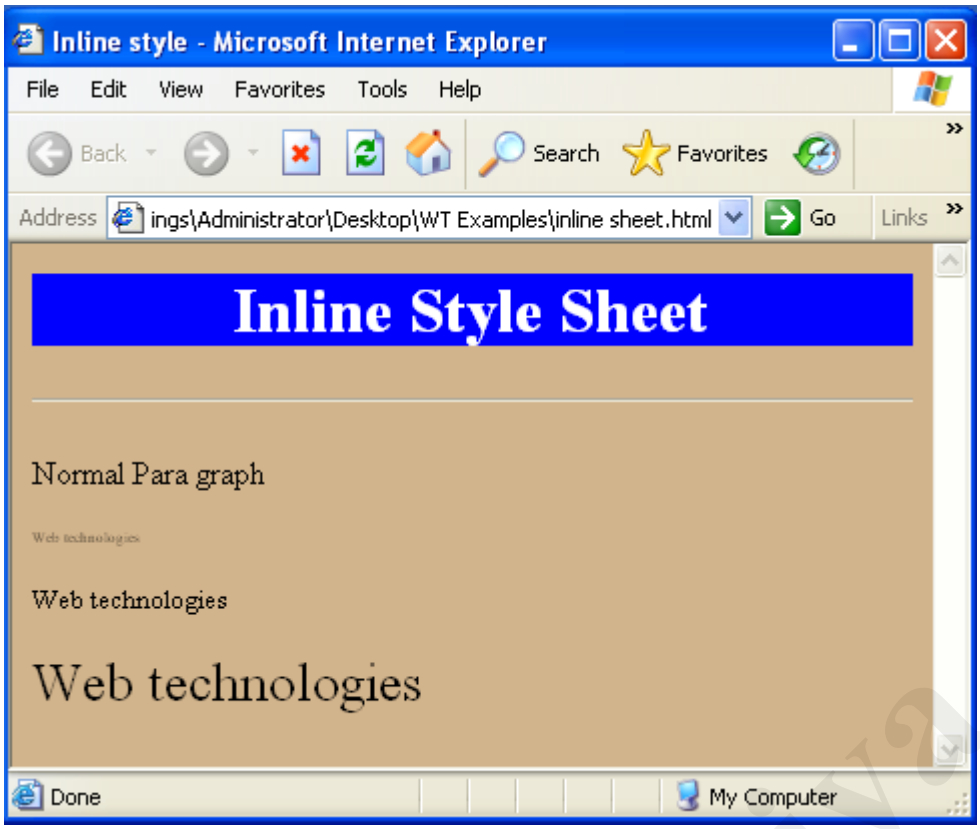
An inline sheet applies style to a particular element in a webpage
Embedding a style sheet is for defining styles to a single webpage. Linking and importing are two ways to attach external style sheet to html documents.
An external style sheet is a file created separately, saved with the .css extension and attached to an HTML document by means of the link element. Linking an external style sheet can apply styles to multiple web pages.

Inline Style Sheet:-

Inclusion of style in a tag is called inline styles. Operator colon(:) Is followed by style property. To separate multiple properties we have to use operator semicolon(;)

Example:-

```
<html>
    <head>
        <title>Inline style</title>
</head>
<body bgcolor=”tan”>
<h1 align=”center” style=”color:white;background-color:blue”>
Inline Style Sheet
</h1>
<hr>
<p>Normal Para graph</p>
<p style=”font-size:5pt”>Web technologies</p>
<p style=”font-size:10pt”>Web technologies</p>
<p style=”font-size:20pt”>Web technologies</p>
</body>
</html>
```



Embedded Style Sheet:-

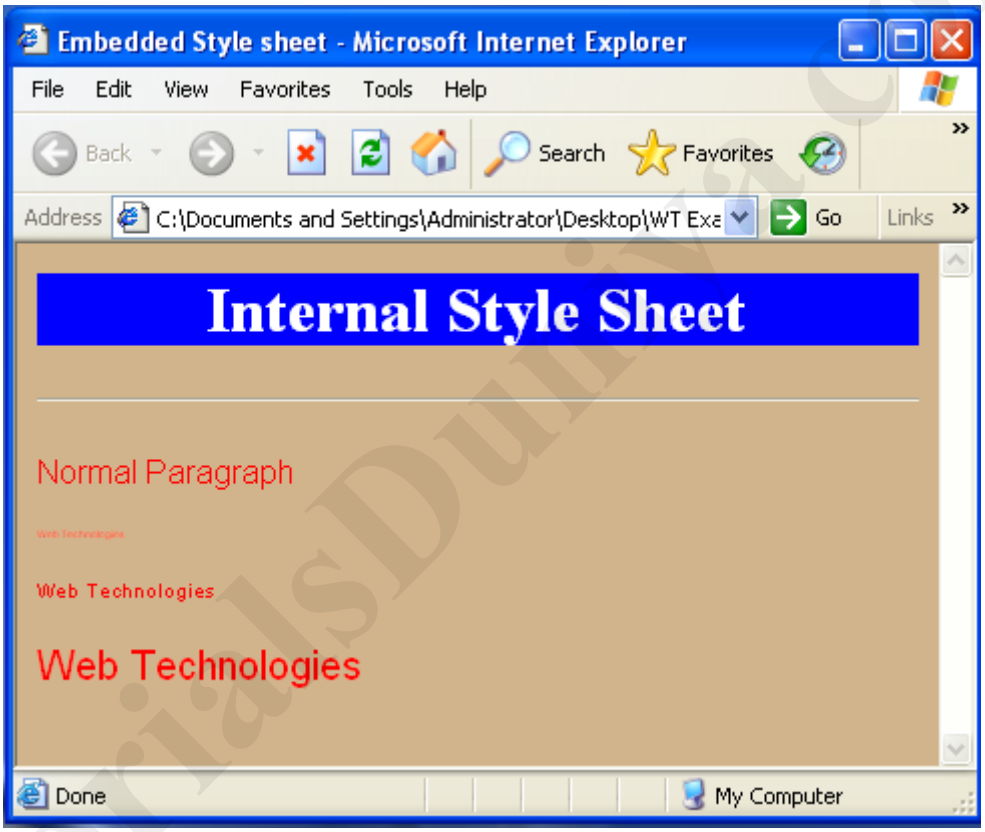
If Style is used as tag, in Header Section then that style sheet is known as internal style sheet. If you include all the formatting parameters in between <style> and </style>, then this is called as internal style sheets or embedded style sheets. Advantage of Internal Style Sheet comparing with inline styles, at a time several tags can be formatted with internal style sheets, where as in inline styles only one tag at a time formatted.

```
<style>
        P       {color:red;font-family:arial}
        .s5     {font-size:5;}
        .s10    {font-size:10;}
        .s15    {font-size:20;}
        H1      {color:white;background-color:blue}
</style>
```

Example:-

```
<html>
<head>
        <title>Embedded Style sheet</title>
        <style>
        P       {color:red;font-family:arial}
        .s5     {font-size:5;}
        .s10    {font-size:10;}
```

```
        .s15    {font-size:20;}
        H1      {color:white;background-color:blue}
        </style>
</head>
<body bgcolor=tan>
        <h1 align="center"> Internal Style Sheet</h1>
        <hr>
        <p>Normal Paragraph</p>
        <p class="s5">Web Technologies</p>
        <p class="s10">Web Technologies</p>
        <p class="s5">Web Technologies</p>
</body>
</html>
```



External Style Sheets:-

Third kind of style is external style sheet. In this total style elements are defined in a separate document, and this document is added to required web page. By using this we can use this style sheets in different web pages. As style sheet is separated from the document it gained the name External Style sheet.

Example:

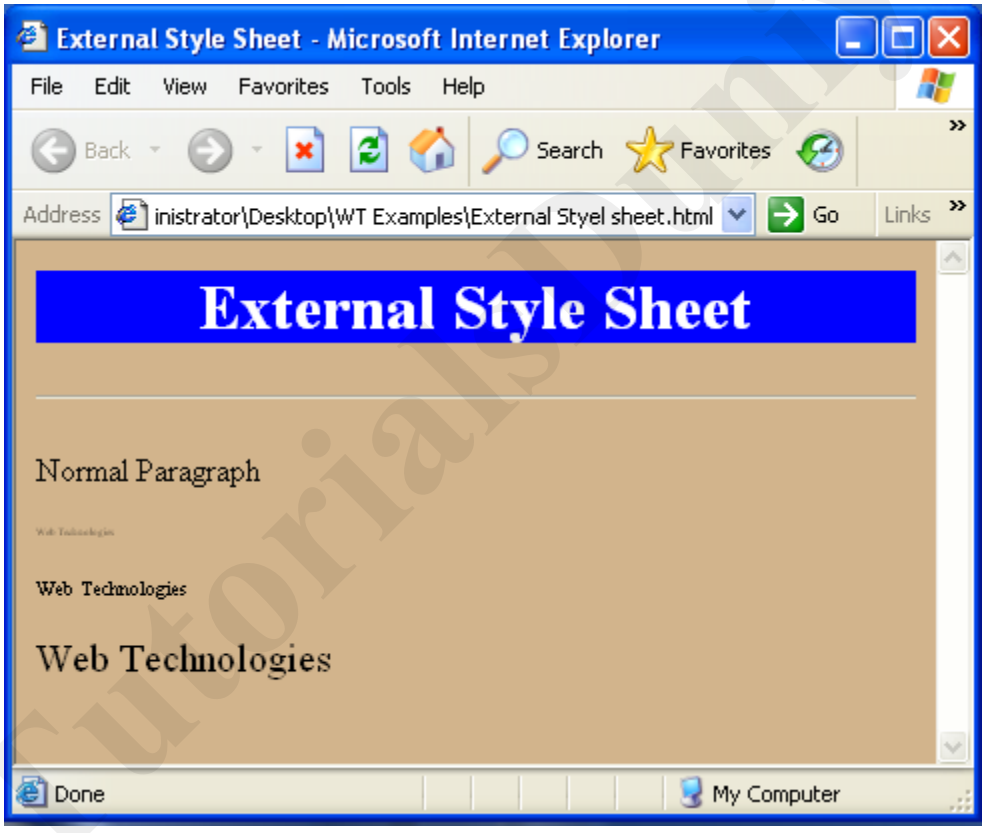
Write one html program that calls a style sheet file.

Ourstyles.css

```
<link rel="stylesheet" type="text/css" href="ourstyles.css">
        P       {color:red;font-family:arial}
        .s5     {font-size:5;}
        .s10    {font-size:10;}
        .s15    {font-size:20;}
        H1      {color:white;background-color:blue}
```

External style sheet.html

```
<html>
<head>
<title>External Style Sheet</title>
<link rel=”stylesheet” type=”text/css” href=”ourstyles.css”>
</head>
<body bgcolor=tan>
        <h1 align=”center”>External Style Sheet</h1>
        <hr>
        <p>Normal Paragraph</p>
        <p class=”s5”>Web Technologies</p>
        <p class=”s10”>Web Technologies</p>
        <p class=”s5”>Web Technologies</p>
</body>
</html>
```



CSS Links

DEFINING STYLES FOR LINKS
As mentioned in the above table, there are four different selectors with respect to links.
You can specify whatever style you'd like to each of these selectors, just like you'd do with normal text.
The four selectors are:

```
A:link
Defines the style for normal unvisited links.

A:visited
Defines the style for visited links.

A:active
Defines the style for active links.
A link becomes active once you click on it.

A:hover
Defines the style for hovered links.
A link is hovered when the mouse moves over it.
Note: Not supported by Netscape browsers prior to version 6.
```

PRACTICAL EXAMPLES
Here you can see a few examples on how CSS can be used to replace the traditional image based mouseover effects for links.
The hover style is not supported by Netscape browsers prior to version 6, but since it does no harm, you can still use it for the benefit of the +90% of visitors that arrive using MSIE).
One of the most common uses of CSS with links is to remove the underline. Typically it's done so that the underline appears only when a hover occurs. In the example below, we did just that. In addition we added a red color for hovered links.

Example: Hover

```
<style type="text/css">
A:link {text-decoration: none}
A:visited {text-decoration: none}
A:active {text-decoration: none}
A:hover {text-decoration: underline; color: red;}
</style>
```

Another example would be to create links that are both underlined and overlined.
Example: Underline/Overline

```
<style type="text/css">
A:link {text-decoration: none}
A:visited {text-decoration: none}
A:active {text-decoration: none}
A:hover {text-decoration: underline overline; color: red;}
</style>
```

A third example would be to create links that change in size
when hovered.
Example: Size changing links

```
<style type="text/css">
A:link {text-decoration: none}
A:visited {text-decoration: none}
A:active {text-decoration: none}
A:hover {font-size:24; font-weight:bold; color: red;}
</style>
```

A final example would be to create links that have a permanent background color, obviously standing out from the rest.

Example: Background colored links

```
<style type="text/css">
A:link {background: #FFCC00; text-decoration: none}
A:visited {background: #FFCC00; text-decoration: none}
A:active {background: #FFCC00; text-decoration: none}
A:hover {background: #FFCC00; font-weight:bold; color: red;}
</style>
```

MULTIPLE LINKSTYLES ON SAME PAGE

The final topic deals with how to add multiple link styles that can be used on the same page.

In the above examples we addressed the HTML selector - A:link etc - and thus redefined the overall link style.

How do we define a link style that is only active in a certain area of the page?

The answer is: context dependent selectors.
Rather than addressing the A:link selector we will address it while being dependant on a certain outer class that surrounds the area where we'd like our link style to be effective.
For example:

```
<html>
<head>
<style type="text/css">
.class1 A:link {text-decoration: none}
.class1 A:visited {text-decoration: none}
.class1 A:active {text-decoration: none}
.class1 A:hover {text-decoration: underline; color: red;}

.class2 A:link {text-decoration: underline overline}
.class2 A:visited {text-decoration: underline overline}
.class2 A:active {text-decoration: underline overline}
.class2 A:hover {text-decoration: underline; color: green;}
```

```
</style>
</head>

<body>
ONE TYPE OF LINKS
<br>
<span class="class1">
<a href="http://www.yahoo.com">YAHOO</a>
<br>
<a href="http://www.google.com">GOOGLE</a>
</span>
<br>
<br>
ANOTHER TYPE OF LINKS
<br>
<span class="class2">
<a href="http://www.yahoo.com">YAHOO</a>
<br>
<a href="http://www.google.com">GOOGLE</a>
</span>
</body>
</html>
```

Note how we use the to define the context.
This is smart for two reasons:

1) The obvious, that it allows us to use different link styles on the same page, rather than being limited to using a single overall link style.

2) We can define entire areas where a certain link style works for all links within that area. Thus, we don't have to add a style definition to each and every link in that area.

CSS Layers

With CSS, it is possible to work with layers: pieces of HTML that are placed on top of the regular page with pixel precision.

The advantages of this are obvious - but once again Netscape has very limited support of CSS layers - and to top it off: the limited support it offers is quite often executed with failures.

So the real challenge when working with layers is to make them work on Netscape browsers as well.

LAYER BASICS

look at the code:

```
LAYER 1 ON TOP:
<div style="position:relative; font-size:50px; z-index:2;">LAYER 1</div>
<div style="position:relative; top:-50; left:5; color:red; font-size:80px; z-index:1">LAYER 2</div>

LAYER 2 ON TOP:
<div style="position:relative; font-size:50px; z-index:3;">LAYER 1</div>
<div style="position:relative; top:-50; left:5; color:red; font-size:80px; z-index:4">LAYER 2</div>
```

To create a layer all you need to do is assign the position attribute to your style. The position can be either absolute or relative.

The position itself is defined with the top and left properties.

Finally, which layer is on top is defined with the z-index attribute.

RELATIVE VERSUS ABSOLUTE POSITIONING

You can either position your layer calculated from the upper left corner(absolute) or calculated from the position where the layer itself is inserted (relative).

```
position:absolute
If you define the position to be absolute it will be calculated from the upper left corner of the page - unless the layer is defined inside another layer, in which case it will be calculated from the upper left corner of the parent layer.

position:relative
If you define the position to be relative it will be relative to the position of the tag that carries the style.
That is, if you add a relatively positioned layer in the middle of the page, then the position will be calculated from that exact spot in the middle of your page where it was added.
```

DEFINING THE POSITION

While the position property indicates the out spring of our coordinate system, the left and top properties defines the exact position of our layer.

You can enter both positive and negative values for these properties - thus it is possible to place content higher up and further to the left on the page than the logical position in the HTML code where the layer itself is defined.

In other words: at the bottom of your HTML code you can enter the code for a layer that is positioned at the top of the resulting page.

Both left and top properties can be dynamically changed
with JavaScript.

This means that it is possible to move things around on the screen even after the page has finished loading.

In fact this technique can be (and has been) used to create entire games. Other uses might be menus that pop out when a mouse-over is detected on a link. The possibilities are endless - but in order to keep things simple, we will not dig into details about these dynamic HTML effects here.

POSITION IN THE STACK - THE Z-INDEX

Picture a game of 52 cards. If the ace of spades was at the bottom we'd say it had z-index:1;. If the queen of hearts was at the top we'd say she had z-index:52;.

Try looking at the code example at the top of this page again, and see how we used the z-index to put LAYER 1 on top in the first example, while we had LAYER 2 on top in the second example.

Very interesting possibilities arise from the fact that the z-index can be dynamically changed with JavaScript.

You could create several "pages" on top of each other - all on the same page. When the user clicks a link it will simply move the layer with the desired info on top rather than load a new page. The techniques to create effects like that goes beyond the scope of pure CSS however, so for now we will just refer to DHTML (Dynamic HTML - a mix between JavaScript and CSS) for further explorations into that area.

VISIBILE VERSUS HIDDEN LAYERS

A final property is the visibility property that will allow you to create invisible layers.

Why would anyone want to create an invisible layer? Well, imagine the possibilities it gives for adding pop-up menus and other cool effects on your pages.

With dynamic HTML it is possible to change the visibility of a layer according to certain

events. The most common use of this is to create menus that pop out (like the sub menus in the START menu on Windows). The trick behind these menus is to create all submenus as invisible layers. Then, when a mouse-over is detected on a link the according layer becomes visible. (Sounds pretty easy - actually is pretty easy - except when tried on Netscape browsers that seem to have only a vague idea of the logic behind CSS layers).

Valid values for the visibility property are: visible and hidden.

This example shows how to create an invisible layer:

```
<div style="position:relative; visibility:hidden;">HELLO!!!</div>
```

PRACTICAL USE OF LAYERS

It's obvious that layers offer certain possibilities for precise positioning of static elements on your pages.

In reality layers are often used in more dynamic ways:

- Flying elements/banners on the page
- Games where you move an object around
- Menus that pop out when triggered
- Menus that become visible when triggered

While all of these effects might seem pretty cool and useful - the fact is that the web is filled with dynamic effects that are much more cool than the average visitor really likes.

The more you create a unique interface for your site the more you force the visitor to forget about what she is used to. Do not underestimate the power of sticking to the elements that the average visitor is accustomed to.

What's cool about creating an effect that makes 90% of all web designers clap their hands while leaving 90% of non-web designers confused or disappointed?

In any case, judge for yourself if a certain effect is really needed - and if so: do not hesitate to use it.

TutorialsDuniya.com

Download FREE Computer Science Notes, Programs, Projects, Books PDF for any university student of BCA, MCA, B.Sc, B.Tech CSE, M.Sc, M.Tech at https://www.tutorialsduniya.com

- Algorithms Notes
- Artificial Intelligence
- Android Programming
- C & C++ Programming
- Combinatorial Optimization
- Computer Graphics
- Computer Networks
- Computer System Architecture
- DBMS & SQL Notes
- Data Analysis & Visualization
- Data Mining
- Data Science
- Data Structures
- Deep Learning
- Digital Image Processing
- Discrete Mathematics
- Information Security
- Internet Technologies
- Java Programming
- JavaScript & jQuery
- Machine Learning
- Microprocessor
- Operating System
- Operational Research
- PHP Notes
- Python Programming
- R Programming
- Software Engineering
- System Programming
- Theory of Computation
- Unix Network Programming
- Web Design & Development

Please Share these Notes with your Friends as well

CSS CURSORS

Microsoft Internet Explorer 4+ and Netscape 6+ supports customized cursors defined with CSS.

Although the cursors will not have the customized look in other browsers it usually doesn't ruin anything. These browsers will simply show the normal arrow-cursor which would be same case as if you refrained from customizing cursors at all.

So unless the page really doesn't work without the customized cursor there shouldn't be technical reasons for choosing not to.

However there might be other reasons for thinking twice before adding custom cursor to your pages. Many users are easily confused or irritated when a site breaks the standard user interface.

 default=cursor:default

 crosshair=cursor:crosshair

 hand=cursor:hand

 pointer=cursor:pointer

 Cross browser=cursor:pointer;cursor:hand

 move=cursor:move

 text=cursor:text

 wait=cursor:wait

 help=cursor:help

 n-resize=cursor:n-resize

 ne-resize=cursor:ne-resize

 e-resize=cursor:e-resize

 se-resize=cursor:se-resize

 s-resize=cursor:s-resize

 sw-resize=cursor:sw-resize

 w-resize=cursor:w-resize

 nw-resize=cursor:nw-resize

 progress=cursor:progress

 not-allowed=cursor:not-allowed

 no-drop=cursor:no-drop

 vertical-text=cursor:vertical-text

 all-scroll=cursor:all-scroll

 col-resize=cursor:col-resize

 row-resize=cursor:row-resize

 cursor:url(uri)=cursor:url(uri

)

ADDING A CUSTOMIZED CURSOR

The syntax for a customized cursor is this:
(Position the mouse over each link to see the effect)

Selector {cursor:value}

For example:

```
<html>
<head>
<style type="text/css">
.xlink {cursor:crosshair}
.hlink{cursor:help}
</style>
</head>

<body>
<b>
<a href="mypage.htm" class="xlink">CROSS LINK</a>
<br>
<a href="mypage.htm" class="hlink">HELP LINK</a>
</b>
</body>
</html>
```

REDEFINING THE CURSOR FOR ENTIRE PAGES

If you want to redefine the cursor so that it's not only showing up when moved over a link, you simply specify the desired cursor using the body-selector.
For example:

```
<html>
<head>
<style type="text/css">
body {cursor:crosshair}
</style>
</head>

<body>
<b>
SOME TEXT
<br>
<a href="mypage.htm">ONE LINK</a>
<br>
<a href="mypage.htm">ANOTHER LINK</a>
</b>
</body>
</html>
```

REDEFINING THE CURSOR FOR AREAS ON A PAGE

If you want one look of the cursor in one area of the page and another look of the cursor in another area you can do it with context dependant selectors.
This way, you create different styles for links, that are dependant on the context. Now if the context is set with a dummy tag, such as you don't have to specify the desired style each and every time there is a link in the section.
For example:

```
<html>
<head>
<style type="text/css">
.xlink A{cursor:crosshair}
.hlink A{cursor:help}
</style>
</head>

<body>
<b>
<span class="xlink">
<a href="mypage.htm">CROSS LINK 1</a><br>
<a href="mypage.htm">CROSS LINK 2</a><br>
<a href="mypage.htm">CROSS LINK 3</a><br>
</span>
<br>
<span class="hlink">
<a href="mypage.htm">HELP LINK 1</a><br>
```

```
<a href="mypage.htm">HELP LINK 2</a><br>
<a href="mypage.htm">HELP LINK 3</a><br>
</span>
</b>
</body>
</html>
```

JAVA SCRIPT

INTRODUCTION JAVA SCRIPT:

Script means small piece of code. Java script you can easily create interactive web pages. It is designed to add interactivity to HTML pages. Scripting languages are two kinds one is client-side other one is servers-side scripting. In general client-side scripting is used for verifying simple validation at client side, server-side scripting is used for database verifications. VBScript, java script and J script are examples for client-side scripting and ASP,JSP, servlets etc.are examples of server-side scripting.

Web pages are two types

1. Static web page

2. Dynamic webpage

- ❖ Static web page where there is no specific interaction with the client
- ❖ Dynamic web page which is having interactions with client and as well as validations can be added.
  Simple HTML script is called static web page, if you add script to HTML page it is called dynamic page. Netscape navigator developed java script.
  Microsoft‟s version of java script is J script.
- ➢ Java script code as written between <script> ---- </script>tags
- ➢ All java script statements end with a semicolon
- ➢ Java script ignores white space
- ➢ Java script is case sensitive language
- ➢ Script program can save as either. Js or. html

Similarities between java script and java:

1. Both java script and java having same kind of operators

2. Java script uses similar control structures of java

3. Nowadays both are used as languages for use on internet.

4. Labeled break and labeled continue both are similar

Difference between java script and java:

1. Java is object –oriented programming language where as java script is object-based programming language.
2. Java is full –featured programming language but java script is not.
3. Java source code is first compiled and the client interprets the code. Java script code is not compiled, only interpreted.

4. Inheritance, polymorphism, threads are not available in JavaScript.

The syntax of the script tag is as follows:

<script language=""scripting language name"">

</script>

The language attribute specifies the scripting language used in the script. Both Microsoft internet explorer and Netscape navigator use java script as the default scripting language. The script tag may be placed in either the head or the body or the body of an HTML document.

Ex: <script language=""java script"">

</script>

Variable declaration in java script:

Variables are declared with the var key word

Var variable name=value

Ex: var x=20

The variable is preceded by the var, it is treated as local variable otherwise variable is treated as global variable.

web technologies

Comments in java script:

Single line comment- //

Multi-line comment- /**/

operators in java script:

Arithmetic operators

Relational operators

Logical operators

Assignment operator

Increment decrement operators

Conditional operator (ternary)

Bitwise operators

Control structures:

- If statement
- Switch
- While
- Do-while
- For
- Break
- Continue

Control structures syntax and working as same as java language.

BASICS OF OBJECTS IN JAVA SCRIPT:

Basic objects are document object and window object

Document object:

To display some information on the screen.

Syntax: document. Write („"any message"");

Document. Writeln („"any message"");

In this document is object name and write () or writeln () are methods. Symbol period is used as connector between object name and method name. The difference between these two methods is carriage form feed character that is new line character automatically added into the document. Write ln(),but it is not included in document. Write ().

Window objects:

Window object there are 3 types of popup boxes.

1. Alert box
2. Confirm box
3. Prompt box

1. Alert box is used error message to user

Syntax: window. Alert („"string message"");

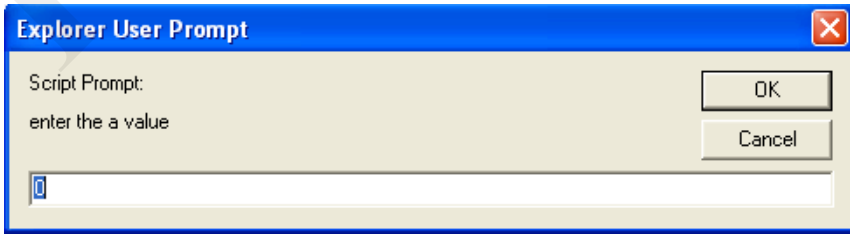
2. Conform syntax

Window.confirm("you want to save?")

(figure)

3. Prompt box for accepting data syntax

Variable=window.prompt("string message","default values");

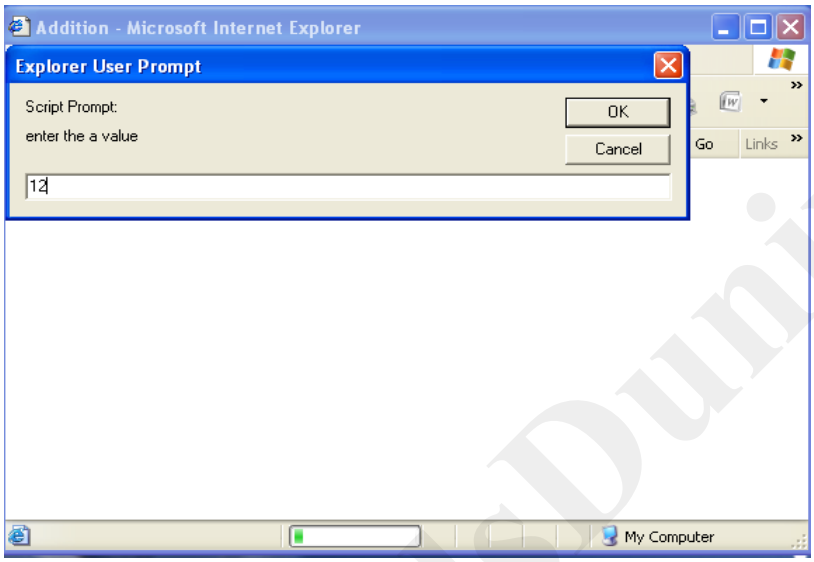


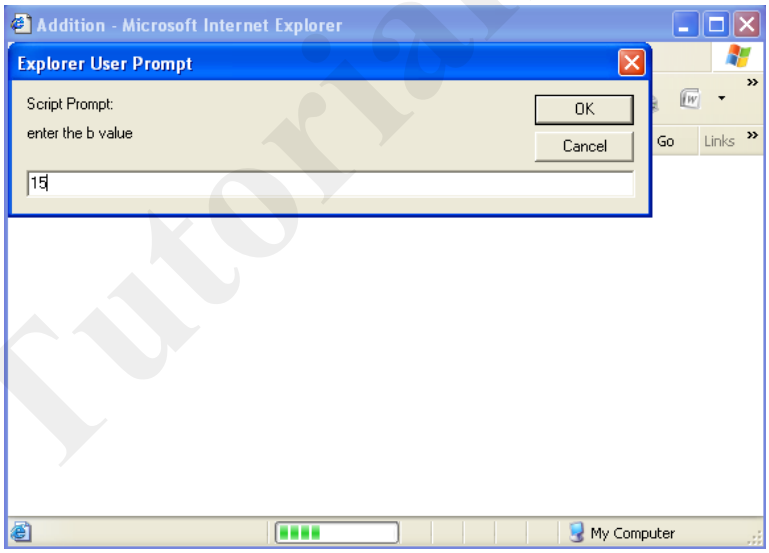
Example 1:

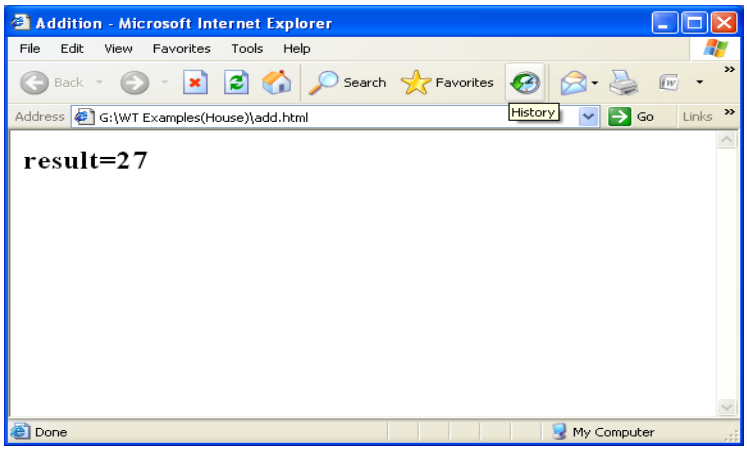
Write a java script addition of two numbers.

<html>

```
<head>
<title>addition</title>
<script language=”java script”>
Var s1,s2,a,b,c;
S1=window.prompt(“enter the a value”,”0”);
S2=window.prompt(“enter the b value”,”0”);
A=parseInt(s1);
b=parseInt(s2);
c=a+b;
document.write(“<h2>result=”+c+</h2>”);
</script>
</head>
</html>
```



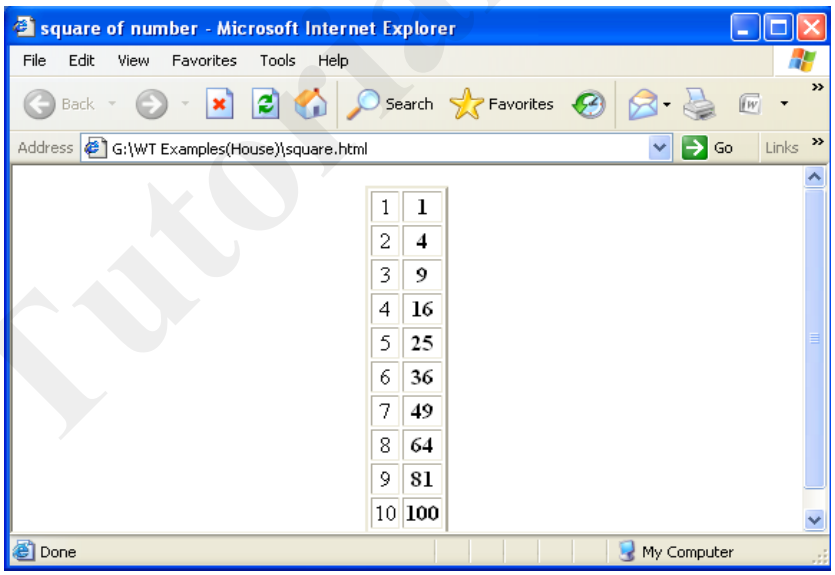




Example2:

Write a program to display 1 to 10 and square of number.

```
<html>
<head>
<title>square of number</title>
<script language=”java script”>
Var i=1,n;
Document .writeln(“<center><table border=1>”);
While(i<=10)
{
Document.writeln(“<tr><td>”+i+”<td>”+(i*i)+”</tr>”);
I++;
}
Document.writeln(“</table></center>”);
</script>
</head>
</html>
```



Example 3:

Write a program to display table for given number

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

```
<title>nth table </title>
<script language=”java script”>
Var i=i,n;
n=parseInt(window.prompt(“enter n value”,”0”));
Document .writeln(“<center><table border=1>”);
For(i=1;i<=10;i++)
{
Document.writeln(“<tr><td>”+n+”<td>X<td>”+i+”<td>=”+(n*i)+”</tr>”);

}
Document.writeln(“</table></center>”);
</script>
</head>
</html>
```

(figure)

EVENTS:

ONCLICK event:

ONCLICK event is used for responding for mouse click action.when the user presses a button, it can call any function through ONCLICK event. For example:

```
<INPUT TYPE=”BUTTON”VALUE=”OK”ONCLICK=”fun()”>
```

When the button OK is clicked it calls the functioin display and according to action given in fun() function it will work.

Example6:

Write a program to display a button OK and display your name when it is clicked.

```
<html>
<head>
<title> onclick example</title>
<script language=”java script”>
Function fun()
{
Window .alert(“hai,this is Raju”);
}
</script>
</head>
<body>
<form>
<INPUT TYPE=”BUTTON” VALUE=”OK“ ONCLICK=”fun()”>
</form>
</body>
```

```
</html>
```

onsubmit event:

Example 7:

```
<html>
<head>
<title> onsubmit example</title>
script language=”java script”>
Function fun()
{
Window .alert(“hai,this is raju”);
}
</script>
</head>
<body>
<form name =”f1” onsubmit=”fun()”>
<INPUT TYPE=”BUTTON” VALUE=”submit” >
</form>
</body>
</html>
```

onload event:

```
<html>
<head>
<title> onload example</title>
<script language=”java script”>
Function fun()
{
Window .alert(“hai,this is ramesh”);
}
</script>
</head>
<body onload=”fun()”>
</body>
</html>
```

Accepting values from text box:

In javascript we can refer the entered values from the textbox which is defined in HTML forms. By taking the name parameter we can perform this task. Simply we can give as follows

Variable=document. Formname.textboxname.value;

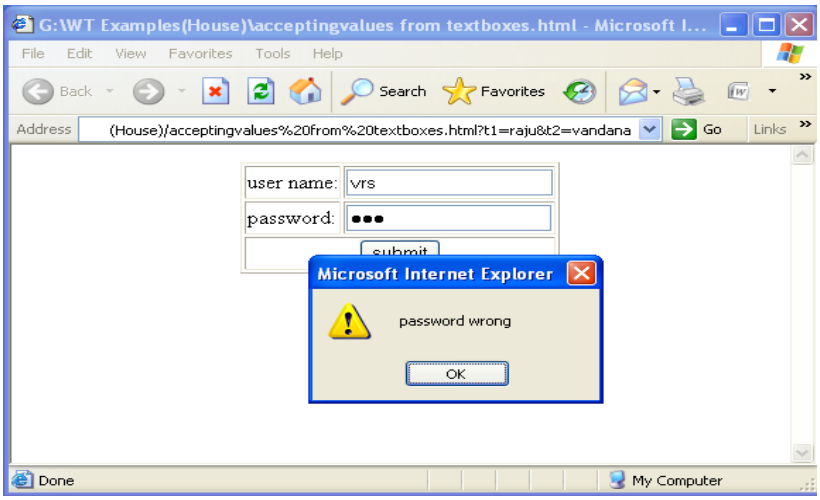
For example, form name is “form1” and text box name is “user” then you can assign its values in java script variable as follows:

Str=document .form1.user.value;

Example 9:

Write a program to login form

```
<html>
<head>
<script language="javascript">
function fun()
{
var s1="vrs";
var s2="yrn";
var s3,s4;
s3=document.f1.t1.value;
s4=document.f1.t2.value;
if((s1==s3)&&(s2==s4))
window.alert("user name and password correct");
else if(!(s1==s3)&&(s2==s4))
window.alert("user name wrong");
else
if((s1==s3)&&!(s2==s4))
window.alert("password wrong");
else
if(!(s1==s3)&&!(s2==s4))
window.alert("both user name and password wrong");
}
</script>
</head>
<body>
<center>
<form name="f1" onsubmit="fun()">
<table border="1">
<tr><td>user name:<td><input type="text" name="t1" size=20></tr>
<tr><td>password:<td><input type="password" name="t2" size=20></tr>
<tr><td colspan=2 align="center"><input type="submit" value="submit"></tr>
</table>
</form>
</center>
</body>
</html>
```

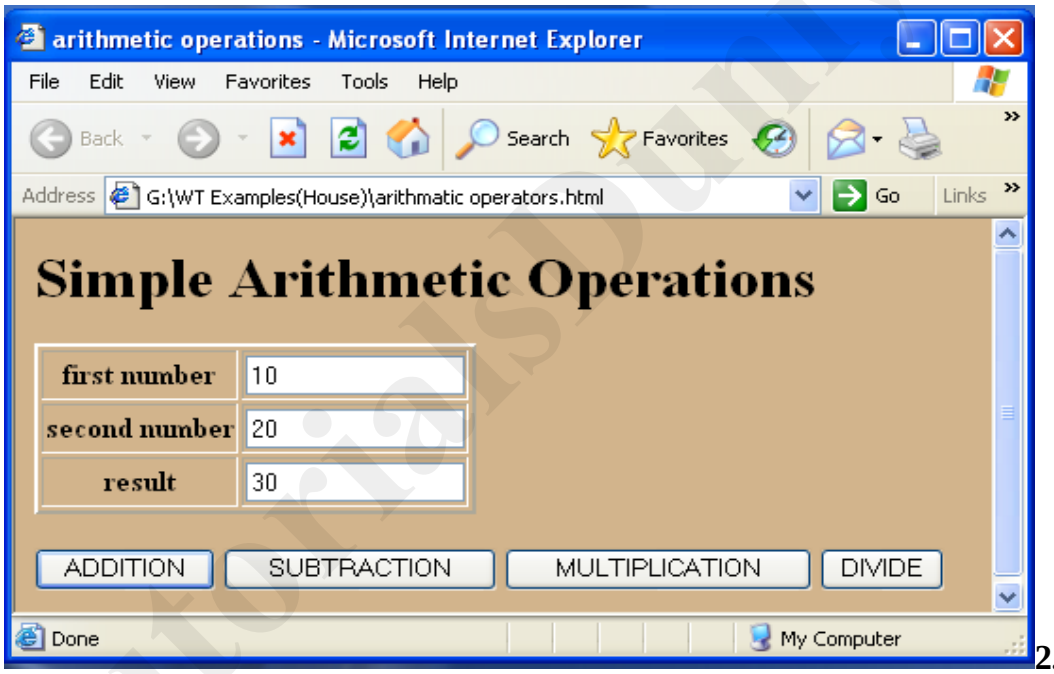


Example 10:

Write a program to display a form with three text fields,two for accepting numbers and thirdone for displaying result.form should contain four buttons with labels ADD,SUBTRACT,MULTIPLY and DIVIDE .perform the respective arthimetic operations and display the result in the third text box.

```
<html>
<head>
<title>arithmetic operations</title>
<script language ="javascript">
function cal(op)
{
var a,b,c=0;
a=parseInt(document.form1.first.value);
b=parseInt(document.form1.second.value);
switch(op)
{
case '+':c=a+b;break;
case '-':c=a-b;break;
case '*':c=a*b;break;
case '/':c=a/b;break;
}
document.form1.result.value=c;
}
</script>
</head>
<body bgcolor=tan>
<form name="form1">
<h1>Simple Arithmetic Operations</h1>
<TABLE border='2'>
<TR>
<TH> first number
<TD><input type="text" name="first" size=15>
```

```
</TR>
<TR>
<TH>second number
<TD><input type="text" name="second" size=15>
</TR>
<TR>
<TH>result
<TD><input type="text" name="result" size=15>
</TR>
</TABLE>
<p>
<INPUT TYPE =BUTTON VALUE="ADDITION" ONCLICK=cal("+")>
<INPUT TYPE =BUTTON VALUE="SUBTRACTION" ONCLICK=cal("-")>
<INPUT TYPE =BUTTON VALUE="MULTIPLICATION"
ONCLICK=cal("*")>
<INPUT TYPE =BUTTON VALUE="DIVIDE" ONCLICK=cal("/")>
</body>
</html>
```



2.6. RECURSION IN JAVASCRIPT:

A recursive function is a function that calls itself either directly or on directly through another function.

The following recursion example in java script shows the evaluation for the factorial of n. a recursive definition of the factorial function is arrived by observation that is N!=N*(N-1)!

Example 11:

Write a program to display the factorial of a given number.

TutorialsDuniya.com

Download FREE Computer Science Notes, Programs, Projects, Books PDF for any university student of BCA, MCA, B.Sc, B.Tech CSE, M.Sc, M.Tech at https://www.tutorialsduniya.com

- Algorithms Notes
- Artificial Intelligence
- Android Programming
- C & C++ Programming
- Combinatorial Optimization
- Computer Graphics
- Computer Networks
- Computer System Architecture
- DBMS & SQL Notes
- Data Analysis & Visualization
- Data Mining
- Data Science
- Data Structures
- Deep Learning
- Digital Image Processing
- Discrete Mathematics
- Information Security
- Internet Technologies
- Java Programming
- JavaScript & jQuery
- Machine Learning
- Microprocessor
- Operating System
- Operational Research
- PHP Notes
- Python Programming
- R Programming
- Software Engineering
- System Programming
- Theory of Computation
- Unix Network Programming
- Web Design & Development

Please Share these Notes with your Friends as well

```
<html>
<head>
<title>recursion for factorial</title>
<script language="javascript">
  var a =window.prompt("enter number for factorial:","0");
var num=parseInt(a);
document.writeln("factorial of "+num+ "is" +factorial(num));
function factorial(num)
{
if(num<=1)
return 1;
else
return num*factorial(num-1);
}
</script>
</head>
</html>
```

ITERATION: if a function or structure executes by following a repetition structure mechanism it is called iterative process. It terminates when a loop continuation condition fails.

Common features of recursion and iteration:

- Both are based on a control structure.
- Both involve repetition.
- Both involve a termination test.
- Both can occur infinitely.

2.6.1. Recursion VS iteration:

S.NO	RECURSION	ITERATION
1	Recursion uses selection structure(such as if, if/else or switch)	Iteration uses a repetition structure(such as for, while or do-while)
2	Recursion achieves repetition through repeated function calls	Iteration explicitily uses the repetition structure.
3	Recursion terminates when a base case is recognized	Iteration terminates when the loop continuation condition fails.
4.	Recursion keeps producing simpler versions of original problem until	Iteration keeps modifying the counter until the counter assumes

	the base case is reached	a value that makes the loop continuation condition fail.
5.	Infinite recursion occurs if the recursion step does not reduce the problem each time.	An infinite loop occurs if the loop continuation test never becomes false.
6.	Recursion can be expensive in both processor time and memory space.	Iteration is less expensive in both processor time and memory space.
7	Recursion consumes considerable memory	Iteration consumes considerable memory

Example 12:

Write java script that three integers from the user and outputs their sum,average,largest.use alert dialog box to display results.

Problem in detail:

Use four different functions for accepting data, finding sum, average caluculation and to find largest among them.every time display the data along with output. Use four buttons to call the respective functions.

```
<html>
<head>
<title> numbers</title>
<script language=”javascript”>
Var a,b,c,data;
Var aNUM,bNUM,cNUM;
Function acc()
{
A=window.prompt(“enter first number”);
B=window.prompt(“enter second number”);
C=window.prompt(“enter third number”);
aNUM=parseInt(a);
bNUM=parseInt(b);
cNUM=parseInt(c);
data=a+” ,”+b+” ,”+c;
}
Function sum()
{
Result=aNUM+bNUM+cNUM;
Window.alert(“sum of “+data+”numbers is “+result);
}
Function avg()
{
Avg=(aNUM+bNUM+cNUM)/3;
Window.alert(“average of “+data+”number is “+avg);
```

```
}
Function largest()
{
Larg=Math.max(math.max(a.NUM,bNUM),cNUM);
Window.alert("largest of "+data+"numbers "+larg);
}
</script>
</head>
<body bg color="tan" text="black">
<form>
<table border="2">
<caption><h3>sum,average and largest</h3></caption>
<colgroup>
<col span="2"ALIGN="right">
</colgroup>
<tr><th>to accept numbers
<td><input type="button" value="accept" onclick="acc()">
</tr>
<tr><th> to sum of those numbers
<td><input type="button" value="sum" onclick="sum()"></tr>
<tr><th>to get average of those numbers
<td><input type="button" value="average" onclick="average()"></tr>
<tr><th>to get largest among those numbers
<td><input type="button" value="largest" onclick="largest()"></tr>
</table>
</form>
</body>
</html>
```

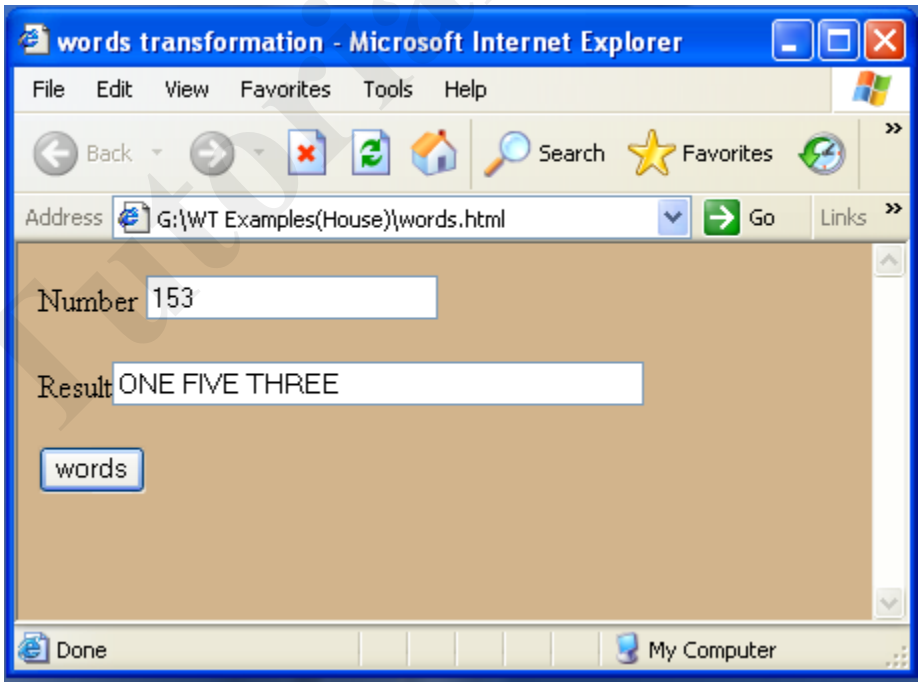
(figure)

Example 13:

Write a program for word equivalent of given number.

```
<html>
<head>
<title>words transformation</title>
<script language="javascript">
function dispwords()
{
var n=0,k=0,i=0,temp=0,r=" ";
var ar=new Array();
n=parseInt(document.f1.num.value);
temp=n;
while(temp>0)
{
k=temp%10;
ar[i++]=words(k);
temp=Math.floor(temp/10);
}
ar.reverse();
```

```
r=ar.join(" ");
document.f1.res.value=r;
}
function words(k)
{
switch(k)
{
case 0:return"ZERO";
case 1:return"ONE";
case 2:return"TWO";
case 3:return"THREE";
case 4:return"FOUR";
case 5:return"FIVE";
case 6:return"SIX";
case 7:return"SEVEN";
case 8:return"EIGHT";
case 9:return"NINE";
}
}
</script>
</head>
<body bgcolor=TAN>
<form name=f1>
Number <input type=text name=num>
<p>
Result<input type=text size=40 name=res>
<p>
<input type=button value="words" onclick="dispwords()">
</form>
</body>
</html>
```



Functions in Java script:

Function is a piece of code that performs specific talk.

Functions are two types 1. Library Functions2.User defined Functions

Library Functions in Java script:

1. **Eval:** This function takes a string representing java script code to execute. The interpreter evaluates the code and executes dynamically.
2. **isFinite:** This function takes a numeric value and returns true if the argument results a finite numeric. Otherwise it returns false.
3. **isNaN:** This function takes a numeric argume nt and returns true if the argument is not a number otherwise returns false.
4. **parseFloat:** It accepts a string as argument and converts into its equivalent float value. If the conversion fails then a value NaN is returned.
5. **parseInt:** It accepts a string as argument and converts into its equivalent numeric. If the conversion fails then a value NaN is returned.

Example:

Write a program to accept a number from user and display whether it is Armstrong or not. Before testing for Armstrong, check user has entered number or not.

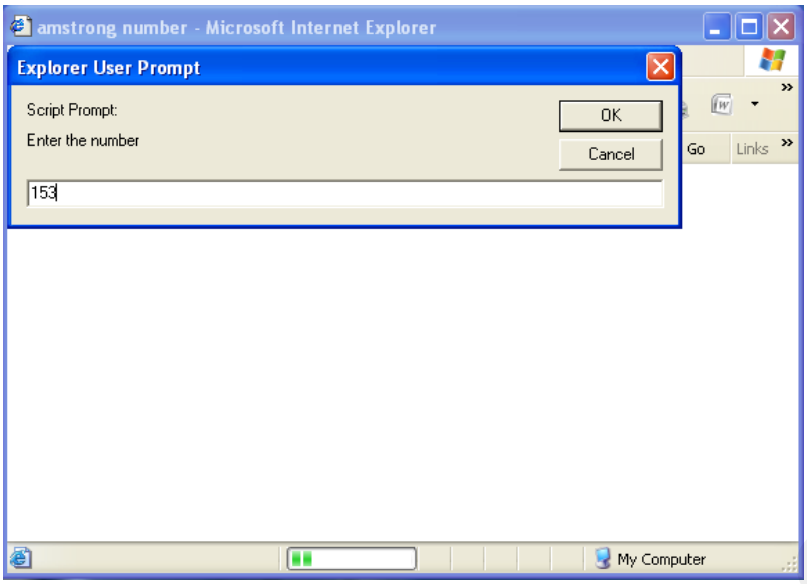
For Example n=153

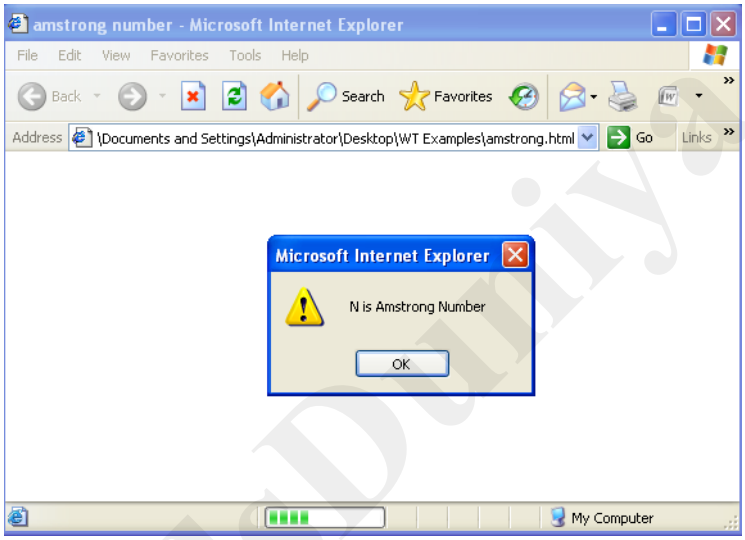
$1^3+5^3+3^3=1+125+27=153=x$

x=n Armstrong

x!=n not Armstrong

```
<html>
<head>
<title>amstrong number</title>
<script language="javascript">
var n,r,x,sum=0;
do
{
 n=parseInt(window.prompt("Enter the number","0"));
 }
while(isNaN(n));
x=n;
while(n!=0)
{
 r=n%10;
 sum=sum+r*r*r;
 n=parseInt(n/10);
}
if(x==sum)
        window.alert("N is Amstrong Number")
        else
        window.alert("N is Not amstrong Number");
</script>
</head>
</html>
```





User-Defined Functions:

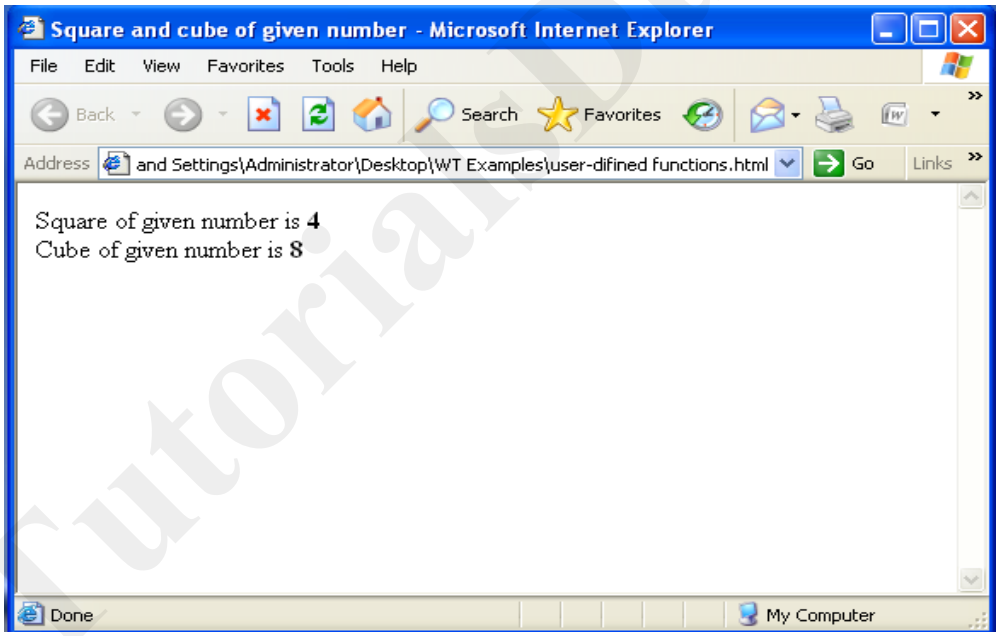
Divide and conquer can be achieved in Javascript by using functions. Functions that you create start with the command "function" and are followed by the name for the function. A function can be defined as follows:

```
Function function-name(parameter-list)
{
    Declerations;
        Statements;
    ----------
    ----------
    return[expression];
}
```

Example:

Write a program to display square and cube of a given number by using user-defined functions.

```
<html>
<head>
        <title>Square and cube of given number</title>
        <script language="JavaScript">
        var a;
        a=parseInt(window.prompt("Enter number","0"));
        document.writeln("Square of given number is
<b>"+square(a)+"</b><br>");
        document.writeln("Cube of given number is <b>"+cube(a)+"</b>");
        function square(k)
        {
           return k*k;
        }
        function cube(k)
        {
           return k*k*k;
        }
        </script>
</head>
</html>
```



Arrays:

Array is a collection of similar type of elements which can be referred by a common name. Any element in an array is referred by an array name followed by [position of the element]. The particular position of element in an array is called array index or subscript.

Arrays are two types1. Single dimensional arrays

2. Multi dimensional arrays

Single dimensional array:

Array Declaration: "new" operator is used to declare and allocate memory for array. Operator new is also known as dynamic memory allocation operator.

Syntax: var variablename=new array(size);

Ex: var a=new array(10);

Initialization of array elements:

If you want to initialize the array elements with zeros then you can use for loop for that purpose, observe the following example:

```
var num=new array(10);
for(i=0;i<num.length;i++)
{
   num[i]=0;
}
```

The same thing can be achieved through for/in control structure, that enables to process each element in an array.

For example,

```
var num =new array(10);
for(var i in num)
{
     num[i]=0;
}
```

The above statements show the way of usage for/in control structure.

Two Dimensional Array:

For example we want required 3 rows 2 cols.

First create 3 rows

```
var a =new array(3);
```

a[0]
a[1]
a[2]

Then create 2 cols each row

```
for(i=0;i<3;i++)
{
   a[i]=new array(2);
}
```

a[0][0]	a[0][1]
a[1][0]	a[1][1]
a[2][0]	a[2][1]

Objects in JavaScript:-

An object is a collection of variables and functions(methods).

Syntax: object.method()

There are several objects in javascript programming languages

Name of the Object	Purpose of the Object
Document	To get complete control on document along with displaying contents
Window	To get dialog control and provision for accepting data from user
Math	Used for mathematical functions and standard constants in mathematics
String	String manipulation can be done and also include to generate HTML
Date	For date manipulations, mainly to get current date and time
Number	To get number constants
Boolean	Wrapper class for Boolean type

Methods in MATH Object:-

Method	Meaning	Example
Math.abs(x)	Returns the absolute value	Math.abs(-20) is 20
Math.ceil(x)	Returns the ceil value	Math.ceil(5.8) is 6 Math.ceil(2.2) is 3
Math.floor(x)	Returns the floor value	Math.floor(5.8) is 5 Math.floor(2.2) is 2
Math.round(x)	Returns the round value, nearest integer value	Math.round(5.8) is 6 Math.round(2.2) is 2
Math.trunc(x)	Removes the decimal places it returns only integer value	Math.trunc(5.8) is 5 Math.trunc(2.2) is 2
Math.max(x,y)	Returns the maximum value	Math.max(2,3) is 3 Math.max(5,2) is 5
Math.min(x,y)	Returns the minimum value	Math.min(2,3) is 2 Math.min(5,2) is 2
Math.sqrt(x)	Returns the square root of x	Math.sqrt(4) is 2
Math.pow(a,b)	This method will compute the a^b	Math.pow(2,4) is 16
Math.sin(x)	Returns the sine value of x	Math.sin(0.0) is 0.0
Math.cos(x)	Returns cosine value of x	Math.cos(0.0) is 1.0
Math.tan(x)	Returns tangent value of x	Math.tan(0.0) is 0
Math.exp(x)	Returns exponential value i.e e^x	Math.exp(0) is 1
Math.random(x)	Generates a random number in beween 0 and 1	Math.random()
Math.log(x)	Display logarithmic value	Math.log(2.7) is 1

String Object:-

A String is a collection of characters; these may be including any kind of special characters, digits, normal characters and other characters.Strings may be written with help of single quotation or double quotation marks.

Methods of STRING Object:-

Method	Meaning	Example
toLowerCase()	It converts the given string into lowercase	var s=”VRSYRN” s.toLowerCase() output: vrsyrn
toUpperCase()	It converts the given string into upper case	var s=” vrsyrn” s.toLowerCase() output: VRSYRN
substring(n)	It returns starting of n th position	Var s=” vrsyrn” s.substring(4) ouput: rn
substring(m,n)	It returns starting from mth character up to nth character, not include nth character	var s=” vrsyrn” s.substring(0,4) ouput: vrsy
Substr(m,n)	It return starting from mth character upto n characters	var s=” vrsyrn” s.substr(1,3) ouput: rsy
charAt(index)	It gives the ith character of string	Var s=”vrsyrn” s.charAt(2) output: s
charCodeAt(index)	It returns ASCII value of the character present at the given index	Var s=”teamwork” s.charCodeAt(2) output:97
indexOf(character)	It gives the position of first occurrence of x to a given string	Var s=”teamwork” s.indexOf(„m“) output:3
indexOf(character,n)	It gives the position of x that occurs after nth position in the string	Var s=”teamwork” s.indexOf(„m“,2) output:3
contact(string)	It returns concatenation of s1and s2. Java script strings can be concatenated using „+“ operator	Var s1=”vrs” Var s2=”yrn” s1.concat(s2) output:vrsyrn
split(string)	It specifies the split string from the given string	Var s=”vrs,yrn” s.split(„„“) output: vrs ,yrn

Date object:

This object is used for obtaining the date and time. This date and time value is based on computer“s local time(system“s time).first we have to create a new date object.for this purpose we are using the new operator with out any operator(to current date): var d1=new Date()

With date information:var d2=new Date(yyyy,mm,dd)
With date and time information: var d3=new d3=new
Date(yyyy,mm,dd,hh,mm,ss,millis)
Methods in date object:

Java script date object provides several methods, they can be classified is string form, get methods and set methods. All these methods are provided in the following table.

METHOD	MEANING
getDate()	Returns 1 to 31 ,day of the month
Get Day()	Returns 0 to 6, Sunday to Saturday respectively.
getMonth()	Returns 0 to 11, jan to dec respectively.
getFullYear()	Returns four-digit year number
getHours()	Returns 0 to 23
get Minitues()	Returns 0 to 59
Getseconds()	Returns 0 to 59
setDate(v)	To set the date,day,month and full year
setDay(v)	
Setmonth(v)	
Setfullyear(Y,m,d)	
Sethours(v)	To set the hours,minutes,seconds,time
Setminutes(v)	
Setseconds(v)	
Settime(v)	

Boolean object:

Wrapper object for Boolean data type is Boolean object. To manipulate Boolean data types in java script program these objects are provided. Constructor to create Boolean object is
Var n=new Boolean(Boolean value)
For example,
Var n=new Boolean(true);
It accepts parameters, true, false, 0, 1 number.NaN or empty string. If it empty string or numbers.NaN it treats as false values.

Methods in Boolean object:
Tostring()-to convert to string for true or false values.
Valueof()-to get value of Boolean object.

2.7.5. NUMBER OBJECT:

Object wrapper in JavaScript is number object, for any number data type, this act as wrapper class. Same methods are applied for number object also. Constructor is as follows:

Var n=new number (any value)
For example
Var n=number(432,123);
Properties of number object:
Number.MAX_VALUE.number.MIN_VALUE,
number.NaN,number.NEGITIVE_INFINITY and number.POSITIVE_INFINITY.
Number.NaN is for not a number.
Object:
Window.open(): syntax:window.open(file name,name,properties)
Window.close(): syntax:window.close()

Properties:

Directories -yes or no
Height -number of pixels
Width -number of pixels.
Location -yes or no
Menubar -yes or no
Resizable -yes or no
Scrollobars -yes or no
Status -yes or no
Toolbar -yes or no
Always lowered -yes or no
Alwaysraised -yes or no
Dependent -yes or no
Hotkeys -yes or no

Example:
open("ram.html","window","width=300,height=300,status=no,toolbar=no,menubar=no");

2.8. DYNAMIC HTML:

2.8.1 What is the difference between HTML and DHTML

HTML	DHTML
HTML is used to create static web pages.	DHTML is used to create dynamic web pages.
Html is consists of simple html tags.	DHTML is made up to html

	tags+cascading style sheets+javascript.
Creation of html web pages is simplest but less interactive.	Creation of DHTML is complex but more interactive.

EVENT HANDLING:

vents are triggers that call one of your function. An event could be action Such as clicking on a button or placing your mouse over an image. for example we will use the onclick event for starting our form validation scripts, and the on mouse overevent for creating graphics images that change when you place your cursor over them.

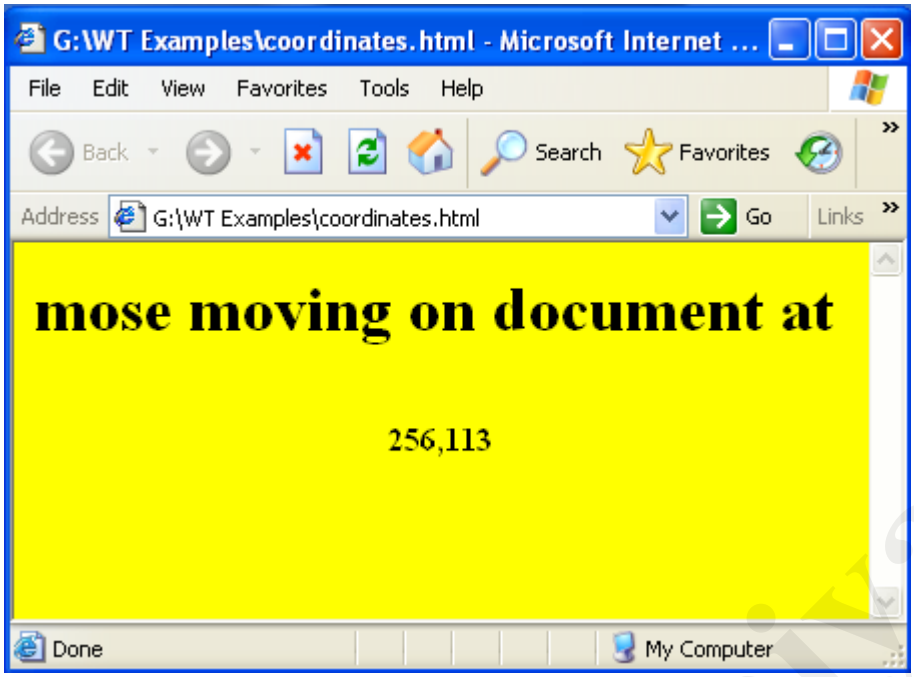
Advantages of events:

1. You can create user –interactive forms.
2. Validate the forms immediately when elements lost their focus
3. Display messages when some components receive focus
4. You can handle errors
5. Process the forms with help of submit button
6. You can write events when objects are selected
7. Also you can handle resize of windows or frames
8. Scripts can respond to user inter actions
9. Change the page according i.e. add dynamism to the page
10. It makes web applications more responsive and user-friendly]

Mouse Events:

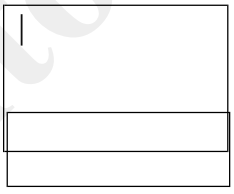
```
<HTML>
<head>
<script language="javascript">
function disp()
{
 c.innerText=event.offsetX+","+event.offsetY;
}
</script>
</head>
<body bgcolor="yellow" onmousemove="disp()">
<h1>mose moving on document at</h1>
<center>
<br>
```

```
<b id=c></b>
</center>
</body>
</html>
```

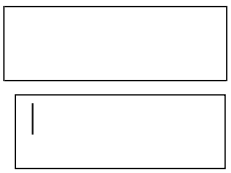


FOCUSING EVENTS:

Focus is nothing but the position of the cursor where it is placed .In forms, focus is moved between various elements on the form. For example, you have simple form as follows:



In the first field cursor is blinking, if you press tab key then the cursor is moved to second field and they appears as follows:



That means first field is loosing its focus .second field is gaining its focus. The events that fired during this are ONFOCUS and ONBLUR events. ONFOCUS is fired when the element gained focus and ONBLUR event is fired when

TutorialsDuniya.com

Download FREE Computer Science Notes, Programs, Projects, Books PDF for any university student of BCA, MCA, B.Sc, B.Tech CSE, M.Sc, M.Tech at https://www.tutorialsduniya.com

- Algorithms Notes
- Artificial Intelligence
- Android Programming
- C & C++ Programming
- Combinatorial Optimization
- Computer Graphics
- Computer Networks
- Computer System Architecture
- DBMS & SQL Notes
- Data Analysis & Visualization
- Data Mining
- Data Science
- Data Structures
- Deep Learning
- Digital Image Processing
- Discrete Mathematics
- Information Security
- Internet Technologies
- Java Programming
- JavaScript & jQuery
- Machine Learning
- Microprocessor
- Operating System
- Operational Research
- PHP Notes
- Python Programming
- R Programming
- Software Engineering
- System Programming
- Theory of Computation
- Unix Network Programming
- Web Design & Development

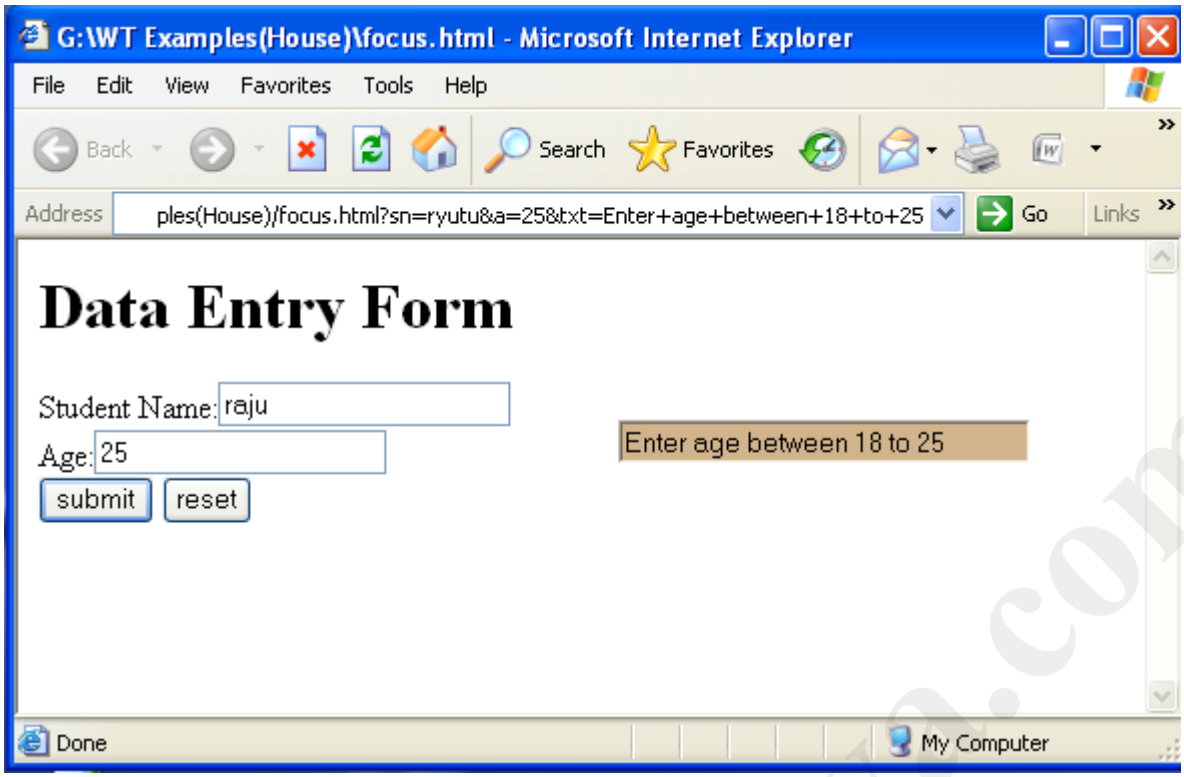
Please Share these Notes with your Friends as well

the element looses its focus. These events are very important if you want to handle the form elements. In general ONFOCUS event is used if user wants to display some messages when some particular graphical element receives focus. ONBLUR is used to verify given condition is satisfied after the user entered some data and pressed tab button. Following program is used to explain the concept of ONFOCUS and ONBLUR events.

Example 16:

Write a program to display a form that accepts student name, age, father name. When age field receives its focus display message that age should be below 18 to 25. After losing its focus from age field verify user entered in between given values or not display respective message.

```
<html>
<head>
<script language="javascript">
function disp()
{
 document.f1.txt.value="Enter age between 18 to 25";
}
</script>
</head>
<body>
<form name="f1">
<h1>Data Entry Form</h1>
Student Name:<input type="text" name="sn">
<br>
Age:<input type="text" name="a" onfocus="disp()">
<br>
<input type="submit" value="submit">
<input type="reset" value="reset">
<input type=text name="txt" size=30 onkeyup="clear()"
style="position:absolute;left:300;top:90;background-color:tan;color:black;">
</form>
</body>
 </html>
```



Key events:

Sometimes you want to check keys that user pressed and according to that you want add dynamism to your web page. Java script provides many key events; with the help of them you can easily get these effects. Important
Key events that are supported by java script are

1. ONKEYDOWN
2. ONKEYPRESS
3. ONKEYUP

Key up event is used to restrict the user entry to a text field. Now the one more example is here. Numerical text fields, these fields accept only numerical values in to restrict the user to enter only numerical values into a text then he should depend on the key events. When a key is pressed, he should verify that the key is numeric or not. If is numeric display the value into the text field otherwise restrict that key.
For this we used the logic as follows

```
If (s.length=1)
{
        If (event.keycode<48|event. Keycode>58)
        Document. Form 1. Txt. Value=s. substring (0, s. length-1)
}
Else
        If (event. Keycode<49|event. Keycode>58)
         Document. Form . txt. Value=``””;
```

First time when user presses other then number, then we clear the text field, after entering some numeric values and if he presses any alphabet we display up to

previous length that is only numeric value. This can be achieved through parseInt() instead of substring().

Form processing and on change event:

ONSUBMIT is invoked when the user submits the form. ONRESET is invoked when the user resets the form. If you want to cancel default action of the event then you can use window. Event. Return value=false.

ONCHANGE event is invoked when the user changed a value in the SELECT element, or text changed in a text field, it can be called as follows.

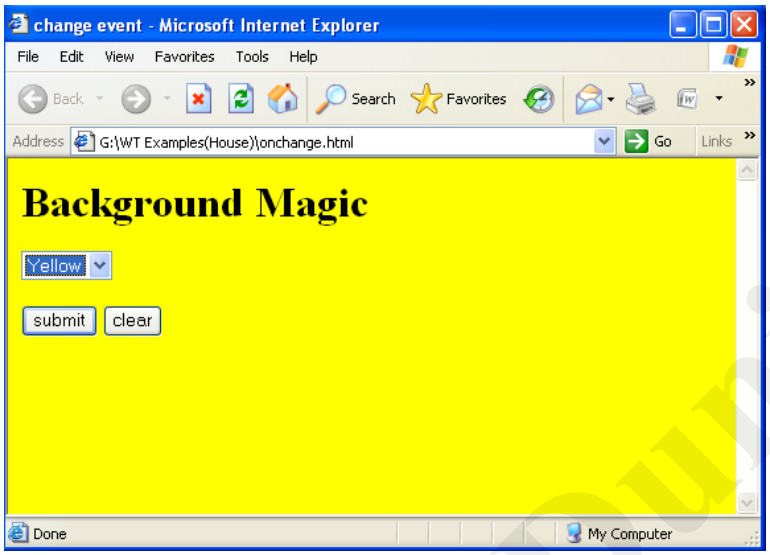
<SELECTID=``sel"""" onchange=change colar()""">

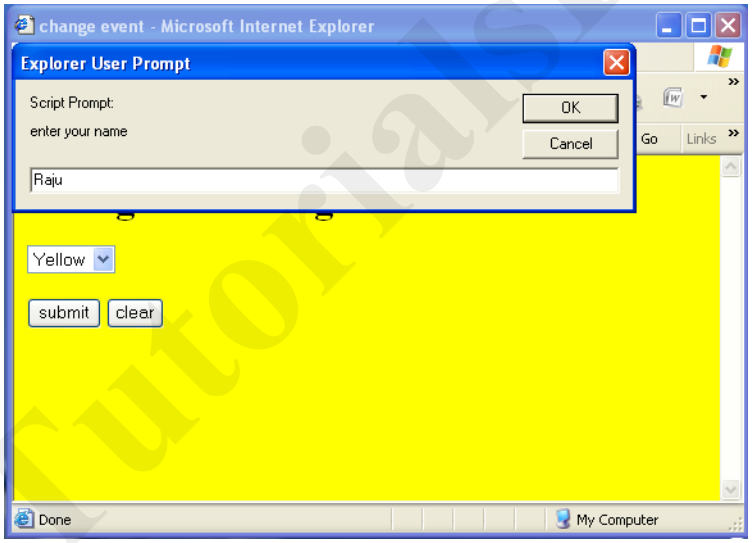
If the options in the combo box are changed then this change color() method is called is called.

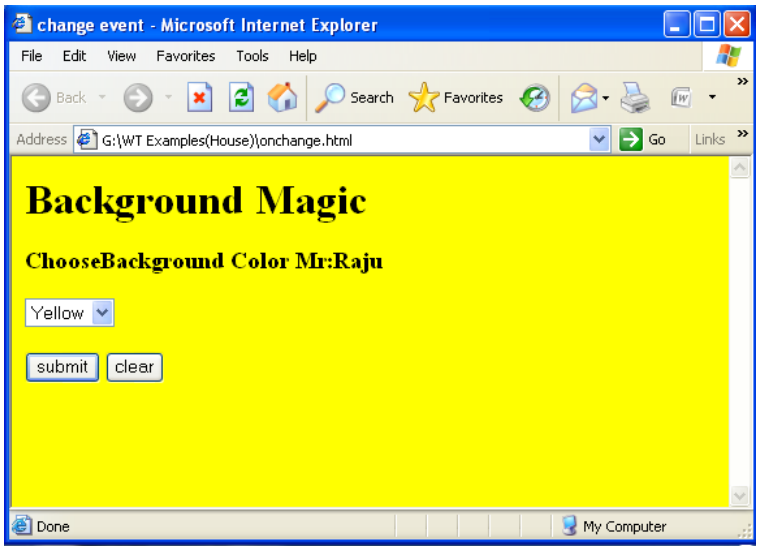
Example 17:write a program to change the background color when the user selected one of the colors from the combo box. When the user press submit button accept user name and display it.if he clicks on rest button display an alert message that he pressed resetting option.

```
<html>
<head>
<title>change of colors</title>
<script language=”java script”>
Function change color()
{
Document.body.style.backgroundcolor=document.form1.se1.value;
}
Function change text()
{
Var nam=window.prompt(“enter your name”,”name”);
s.inner text=”choose background color Mr/Ms :”+nam;
window.event.return value=false;
}
function say bye()
{
Window.alert(“now you are resetting “);
}
</script>
</head>
<boy bgcolor=tan>
<h1>BACKGROUND MAGIC</h1>
<form name=form1 onsubmit=change Text()  onreset=”saybye()”>
<h3 id=s>choose background color</h3>
```

```
<select id=”se1” onchange=”changecolor()”>
<option value=”red”>Red
<option value=”blue”>Blue
<option value=”yellow”>Yellow
<option value=”green”>Green
</select>
<input type=submit value=”submit”>
<input type=reset value=”reset”>
</form>
</body>
</html>
```







2.9.5 Timer methods:

HTML elements can be modified according to code. JavaScript is having setInterval()method in window object that is used for running a function according to given time interval. For example window.setInterval("disp()",1000),the disp() method calls for every 1000 milliseconds. There is another method setTiout(),the specified function called after waiting number of milliseconds specified along with setTimeout() and clearInterval().,these are used for stopping the timeout timer or interval timer. These functions are needed because window.setInterval()program will run continuously ,to stop it we need clearInterval()method.

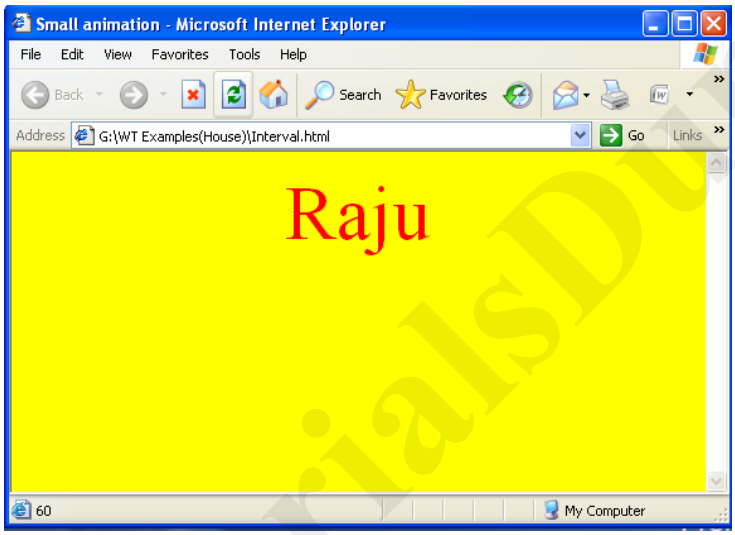
These two methods are used as follows.
To start timer
T1.window.setInterval("disp()",100);
To stop Timer
Window.clearInterval(t1);;

Example 18:
write a program to display your name by increasing font sizes according to time intervals.

```
<html>
<head>
<title>Small animation</title>
<script language="javascript">
var t1;
var c=0;
function start()
{
        t1=window.setInterval("incr()",100);
}
```

```
function incr()
{
        c=c+10;
        t.style.fontSize=c;
        window.status=c;
        if(c==200)
                window.clearInterval(t1);
        //t.style.color=c;

}
</script>
</head>
<body bgcolor="yellow" onload="start()">
<center>
<p id="t" style="color:red">Raju</p>
</body>
</html>
```



2.10. COLLECTIONS:

Arrays of related object on a page can be called as collections.dynamic HTML object model is contains various collections.
Some of the collections are all, children, frames etc.
Collection all"contains all the HTML elements of the current web page.as all is an array of elements ,it contains the property length to get the number of tags on the web page;other property of this collection is tag name that is used to get the specified tag name. to display all the tag names of the current page,just run a for loop and add individual tag to a string.

There is a property inner text for html element;it refers to the text contained in that element.property inner HTML is similar to inner Text, but it can include HTML formatting. The outerHTML property is similar to innerHTML property;it includes the enclosing HTML tags as well as the content inside them.

SUMMMARY OF COLLECTIONS:

Collection name	usage
all	All elements on the web page
children	Children elements for individual element
anchors	Collection<A name>tags
links	Collection<AHREF>tags
applets	Collection pf <APPLET>tags
embeds	Collection of <EMBED>tags
images	Collection of <IMG>tags
forms	Collection of FORMS
frames	Representing each frame
scripts	Collection contains all the <script>tags
stylesheets	Represent each style element

FILTERS AND TRANSITIONS:
Horizontal and vertical filters:

Fundamental filters supported by dynamic HTML are flipv and fliph popularly known as flip filters. These filters create mirror effects for the images or text. These are vertical in nature if filters is flipv and horizontal in the case of fliph. Simple effects are as follows:

In dynamic HTML creation of filter is very simple.only thing you require is style sheets. In styles, you have a special style that is filter style,which is used for all the

kinds of filters.the value passed to filter is fliph for horizontal flip and flipv for vertical flip. If you require both filters just apply both of them.

Example:

Style=”filter:fliph”for horizontal flip
Style=”filter:flipV” for vertical flip
Style =”filter:fliph flipv” for both flips.

Example 19:

Write a program to display vertical flip for ramesh

```
<html>
<body bgcolor=tan>
<table>
<tr><th  size=40px;filter:flipv”>Ramesh</tr>
</table>
</body>
</html>
```

Example 20:

Write a DHTML and java script program to accept username and display the given name along with four buttons that contains horizontal flip,vertical flip, both flips and normal buttons with actions.

```
<html>
<head>
<script language="javascript">
var k;
function change(k)
{
var k;
        switch(k)
        {
                case 1:a.style.filter='fliph';break;
                case 2:a.style.filter='flipv';break;
                case 3:a.style.filter='fliph flipv';break;
                case 4:a.style.filter='';
        }
}
</script>
</head>
<body onload="a.innerText=window.prompt('Your Name')">
<form name="f1">
```

```
<p id="a" align="center" style="background-color:tan;font-size:80px;">Your
Name</p>
<input type=button onClick="change(1)" value="horizontal">
<input type=button onclick="change(2)" value="vertical">
<input type=button onclick="change(3)" value="Horizontal&vertical">
<input type=button onclick="change(4)" value="Normal">
</form>
</body>
</html>
```

filtering colors:chroma filter:

One of the option available in graphics software is you can turn off the color that you don‟t want. For image enhancements these kinds of options are useful. One such kind of filter is chroma filter. Without using any graphics software you can achieve these effects on your web page with the help chroma filter. Dynamically transparent effects can be obtained.

In style argument if you pass filter:chroma, then chroma filter starts its processing. By using dynamic features, if you can any type of graphical user components to apply this feature. By using identification of an image you can call filters collection and change its properties are follows.

chImg. Filters(„‟chroma‟‟‟). Color=color code;

example 21:

chroma an image by using paint brush with some black and white colors. Write dynamic html program to apply black and white filters to that image. (used white and black color for understanding purpose)

```
<HTML>
<HEAD>
        <TITLE>chroma black and write filter</TITLE>
        <SCRIPTLANGUAGE=‟‟‟JAVASCRIPT‟‟‟>
        Function sopcolor(c)
{
          If (c){
            Chimg. Filters(„‟chroma‟‟‟). Color=parseint(c,16);
            Chimg.filters(``chroma‟‟‟). Endable=true;
            }
                Else
                        Chimg. Filters(``chroma‟‟‟). Enabled=false;
          }
        </SCRIPT>
</HEAD>
<BODY BGCOLOR=TAN>
<H1>CHROMA FILTER:</H1>
        <img id=‟‟‟chimg‟‟‟sic=‟‟‟cir. Jpg‟‟‟style=‟‟‟filter:chroma‟‟‟>
        <FORM>
```

```
<INPUT TYPE=BUTTON VALUE=""ALL
COLORS""onclick=stopcolor(0)>
<INPUT TYPE=BUTTON VALUE=""BLACK""
Onclick=stopcolor(000000")>
<INPUT TYPE=BUTTON VALUE=""WHITE"" onclick=stopcplor(FFFFFF")>
</FORM>
</BIDY>
</HTML>
```

Masking:

Masking effect is obtained by using mask filters. These are created as image masks, with the help of these filters you can get an effect of background image colors applied on the specified text of foreground. Image mask means adding text to image with image colors. Foreground text is in transparent color so that background image can be displayed on the text.

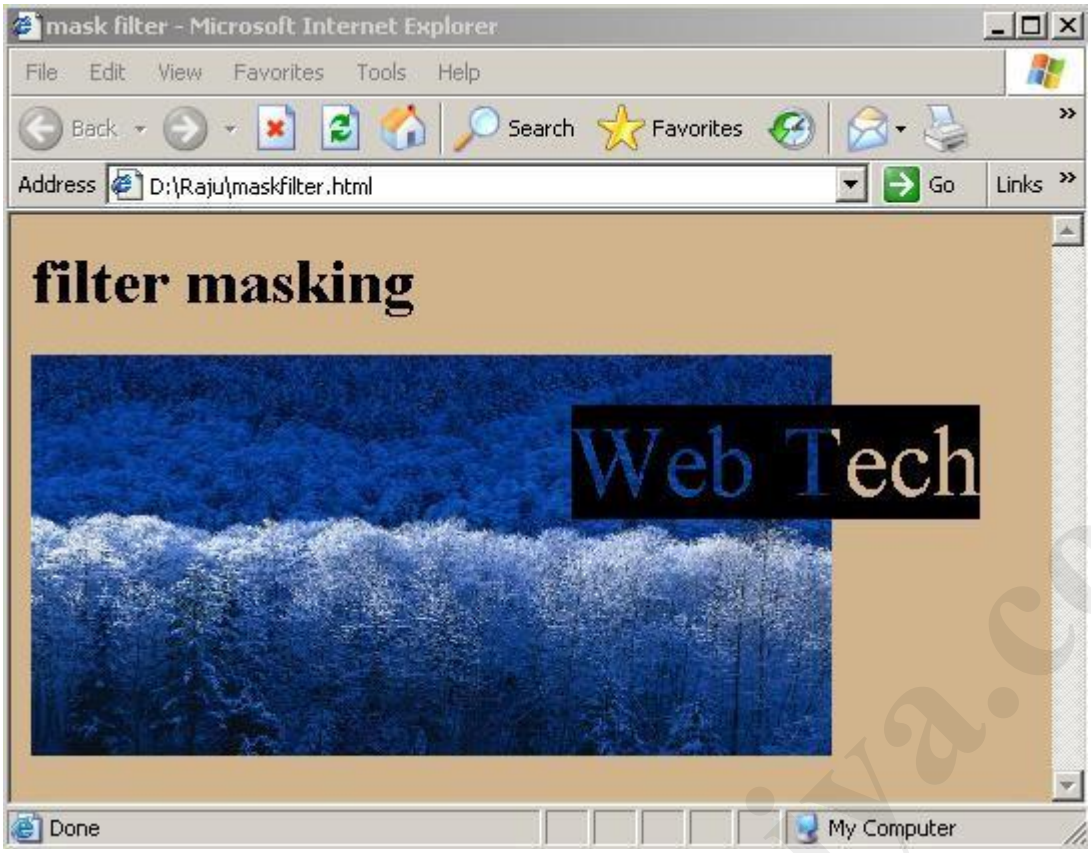
These effects can be achieved in DHTML by using statements.

<h1 style=``position: absolute: top:95 ; left:280;filter:mask(color=#000000)"">
Followed by an image behind this header so that you can get the mask feel.
<img src=``anyimg. Jpg""width=""400""height=``200"">
Better to get both are placed at one location.

Example 22:

Write a DHTML program to display a text on given image. That text should start form the image and moved outwards of the image, so that background color of webpage should apply to text.

```
<html>
<head>
<title>mask filter</title>
</head>
<body bgcolor=tan>
<h1>filter masking</h1>
<div style="position:absolute;top:95;left:280;filter:mask(color= #000000)">
<p style="font-size:50px">Web Tech</div>
<img src="D:\Raju\WT\winter.jpg" width=400 height=200>
</body>
</html>
```



Other filters:

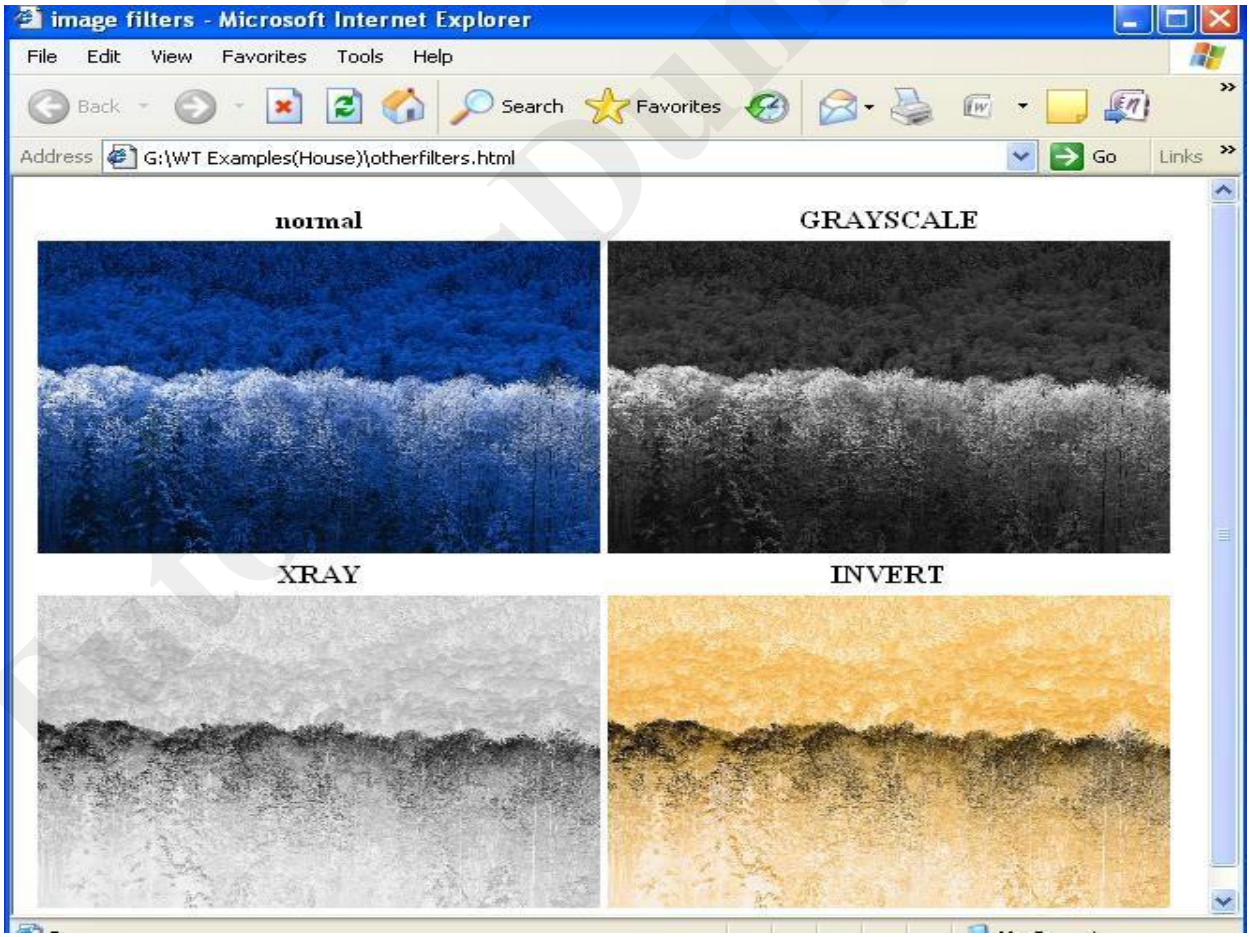
Graphics software or photo editors contain an important feature for image enhancement. Those features are applying some negative image or gray scale effects to the picture. Automatically by clicking one of the menu options. DHTML also provides this feature by using filters. There are three important filters of that kind.

1. **Invert filter:** this applies a negative image effect to the given image, which means that drak areas became light and area became dark.
   Example: <img src=”could06.jpg” style=”filter: invert”>
2. **Gray filter:** this generates a grayscale image effect; in that all colors are stripped from the image and all that remains in brightness data.
   Example:<img src=”could06.jpg” style=”filter:gray”>
3. **X-ray filter:** this generates inversion of grayscale image effect that is called as x-ray effect.
   Example:<img src=”could06.jpg” style=”filter:xray”>

Example 23:
Write a program that displays grayscale,invert and x-ray effects for the given image.

<html>

```
<head>
<title>image filters</title>
</head>
<body>
<table>
<tr><th>normal<th>GRAYSCALE</tr>
<tr>
<td><img src="C:\Documents and Settings\All Users\Documents\My Pictures\Sample Pictures\Winter.jpg" width=300 height=200></td>
<td><img src="C:\Documents and Settings\All Users\Documents\My Pictures\Sample Pictures\Winter.jpg" width=300 height=200 style="filter:gray"></td></tr>
<tr><th>XRAY <th>INVERT</tr>
<tr>
<td><img src="C:\Documents and Settings\All Users\Documents\My Pictures\Sample Pictures\Winter.jpg" width=300 height=200 style="filter:xray"></td>
<td><img src="C:\Documents and Settings\All Users\Documents\My Pictures\Sample Pictures\Winter.jpg" width=300 height=200 style="filter:invert"></td>
</tr>
</table>
</body>
</html>
```



Shadow effects:

Now a day‟s normal word packages are also providing facilities to add shadows to your text. For example a message “welcome” with shadow effect observes here.

But this slight difference we can‟t observe, but if this shadow is larger then we can easily identify it.to add some depth to your text then you can apply shadow filter to your text. A three –dimensional appearance can be observed to your text if you apply shadow effects shadow filters in DHTML takes two important parameters. One is direction that specifies in which direction you want to display your shadow and other parameters is color.

Tells the color of the shadow. For example, for letter I that shadow appears left side as follows.

As exactly may be you cannot get with simple shadoweffects in DHTML, but you can decided which color you want and which directionthat shadow can be displayed you can fix it.

<p style=”color:blue; font-size:75;position:absolute;

Filter:shadow(direction=50,color=black)”>ramesh

Then the shadow of text ramesh appears 50 degrees, means above right side and display in black color.actual text color is blue.

The shadow directions are 0-top, 45-above right,90-right,135-below right,180-below,225-below left,270-left,315-above left.

Example 24:

Write a DHTML program that displays shadow for a text.

```
<HTML>
<head>
<title>shadow filter</title>
</head>
<body bgcolor=tan>
<p style="color:white;font-
size:75;position:absolute;top:25;left:25;padding:10;filter:shadow(direction=315,color
=red)">SACET
<p style="color:green;font-
size:75;position:absolute;top:25;left:300;padding:10;filter:shadow(direction=150,colo
r=black)">SACET
<p style="color:blue;font-
size:75;position:absolute;top:150;left:25;padding:10;filter:shadow(direction=150,colo
r=black)">SACET
<p style="color:yellow;font-
size:75;position:absolute;top:150;left:300;padding:10;filter:shadow(direction=150,col
or=blue)">SACET
</body>
</html>
```

Note: But the above code will be supported for the lower versions of the browsers; higher version of the browsers will support the following code

```
<HTML>
<head>
<title>shadow filter</title>
</head>
<body bgcolor=tan>
<p style="color:white;font-size:75;position:absolute;top:25;left:25;padding:10;text-shadow:2px 2px 0 red">SACET</p>
<p style="color:green;font-size:75;position:absolute;top:25;left:300;padding:10;box-shadow: 5px 5px 10 black">SACET</p>
<p style="color:blue;font-size:75;position:absolute;top:150;left:25;padding:10;text-shadow:5px 3px 2 white">SACET</p>
<p style="color:yellow;font-size:75;position:absolute;top:150;left:300;padding:10;text-shadow:4px 5px 3 red">SACET</p>

</body>
</html>
```

TutorialsDuniya.com

Download FREE Computer Science Notes, Programs, Projects, Books PDF for any university student of BCA, MCA, B.Sc, B.Tech CSE, M.Sc, M.Tech at https://www.tutorialsduniya.com

- Algorithms Notes
- Artificial Intelligence
- Android Programming
- C & C++ Programming
- Combinatorial Optimization
- Computer Graphics
- Computer Networks
- Computer System Architecture
- DBMS & SQL Notes
- Data Analysis & Visualization
- Data Mining
- Data Science
- Data Structures
- Deep Learning
- Digital Image Processing
- Discrete Mathematics
- Information Security
- Internet Technologies
- Java Programming
- JavaScript & jQuery
- Machine Learning
- Microprocessor
- Operating System
- Operational Research
- PHP Notes
- Python Programming
- R Programming
- Software Engineering
- System Programming
- Theory of Computation
- Unix Network Programming
- Web Design & Development

Please Share these Notes with your Friends as well

Gradient effects

One of the beautiful effect of image graphics is to display an image starting with a color and ending with the complete picture. This can be called as gradient effects. The alpha filter is used to create the above said gradient effect. Alpha filter is as follows

Filter: alpha (style=2, opacity=100, finishopacity=0)”>

It takes three important parameters indicates how the gradient transforms, there are four options are available for this.

0-uniform
1-linear
2-circular
3-rectangular

Other two parameters are opacity indicates the percentage at what percent at what opacity the specified gradient starts and other one finish opacity indicates where it finishes

Pic.filters(“alpha”)opacity=0;

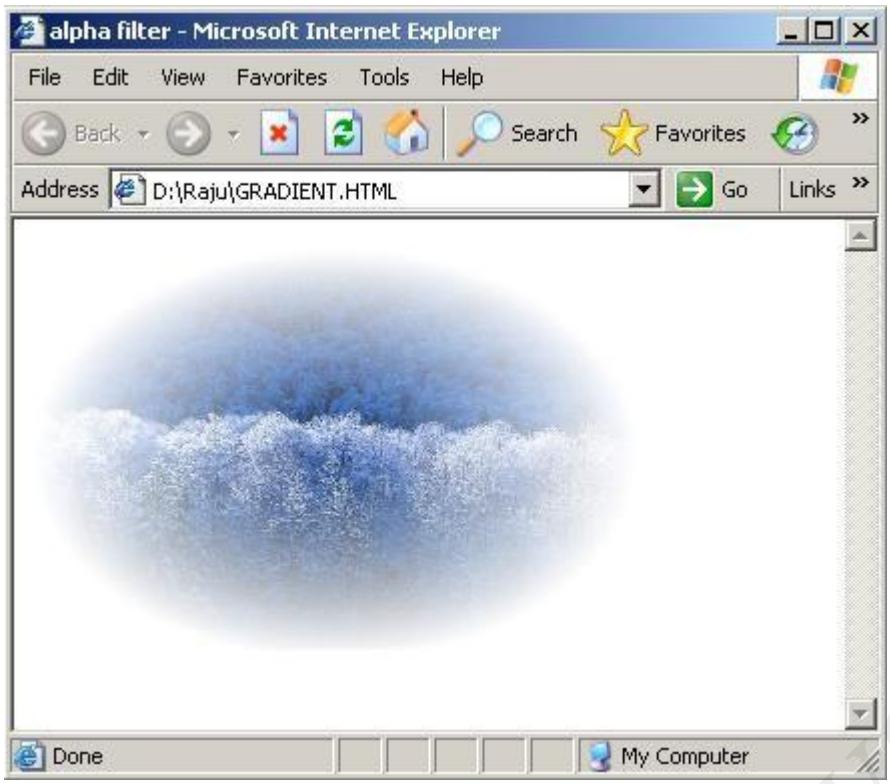
Pic.filters(“alpha”).finish opacity=100;

Indicates that first initial percent of opacity starts with 0 and finished it after achieving complete opacity.

Example 25:

Write a simple program that explains the concept of alpha gradient.

```
<html>
<head>
<title>alpha filter</title>
</head>
<body>
<div id="pic"
style="position:absolute;filter:alpha(style=2,opacity=100,finishopacity=0)">
<img width=300 height=200 src="D:\Raju\WT\winter.jpg">
</div>
</body>
</html>
```



Note: Present versions of the browsers supports following syntax

All Filters

A demonstration of all filter functions:

```
.blur {
   -webkit-filter: blur(4px);
   filter: blur(4px);
}

.brightness {
   -webkit-filter: brightness(0.30);
   filter: brightness(0.30);
}

.contrast {
   -webkit-filter: contrast(180%);
   filter: contrast(180%);
}

.grayscale {
   -webkit-filter: grayscale(100%);
   filter: grayscale(100%);
}

.huerotate {
   -webkit-filter: hue-rotate(180deg);
   filter: hue-rotate(180deg);
}
```

```
.invert {
   -webkit-filter: invert(100%);
   filter: invert(100%);
}

.opacity {
   -webkit-filter: opacity(50%);
   filter: opacity(50%);
}

.saturate {
   -webkit-filter: saturate(7);
   filter: saturate(7);
}

.sepia {
   -webkit-filter: sepia(100%);
   filter: sepia(100%);
}

.shadow {
   -webkit-filter: drop-shadow(8px 8px 10px green);
   filter: drop-shadow(8px 8px 10px green);
}
```

Unit 3
Working with XML

Introduction to XML:

XML stands for e**X**tensible **M**arkup **L**anguage and is a text-based markup language derived from Standard Generalized Markup Language (SGML). The primary purpose of this standard is to provide way to store self describing data easily. Self-describing data are those that describe both their structure and their content. But, HTML documents describe how data should appear on the browsers screen and no information about the data. XML documents, on the other hand describe the meaning of data. The content and structure of XML documents are accessed by software module called XML processor.

XML Characteristics:

1. **XML is extensible :** XML essentially allows you to create your own language, or tags, that suits your application.
2. **XML separates data from presentation :** XML allows you to store content with regard to how it will be presented.
3. **XML is a public standard :** XML was developed by an organization called the World Wide Web Consortium (W3C) and available as an open standard.

XML Usage:

A short list of XML's usage says it all

- XML can work behind the scene to simplify the creation of HTML documents for large web sites.
- XML can be used to exchange of information between organizations and systems.
- XML can be used for offloading and reloading of databases.
- XML can be used to store and arrange data in a way that is customizable for your needs.
- XML can easily be mixed with stylesheets to create almost any output desired.

XML features:

- XML allows the user to define his own tags and his own document structure.
- XML document is pure information wrapped in XML tags.
- XML is a text based language, plain text files can be used to share data.
- XML provides a software and hardware independent way of sharing data.

XML document structure

An XML document consists of following parts: 1) Prolog 2) Body

1. Prolog:

This part of XML document may contain following parts: XML declaration, Optional processing instructions, Comments and Document Type Declaration

XML Declaration:

Every XML document should start with one-line XML declaration which describes document itself. The XML declaration is written as below:

Syn: <?xml version="1.0" encoding="UTF-8"?>

Where *version* is the XML version and *encoding* specify the character encoding used in the document. UTF-8 stands for Unicode Transformation Format is used for set of ASCII characters. It also have *standalone* attribute indicates whether the document can be processed as standalone document or is dependent on other document like Document Type Declaration(DTD).

Syn: <?xml version="1.0" encoding="UTF-8" standalone="yes|no"?>

Processing Instruction:

Processing Instructions starts with left angular bracket along with question mark(<?),ending with question mark followed by the right angular bracket(?>). These parameters instruct the application about how to interpret XML document. XML parser's do not take care of processing instructions and are not text portion of XML document.

Ex: <?xsl-stylesheet href="simple.xsl" type="text/xsl"?>

Comments:
Like HTML, comments may use anywhere in XML documents. An XML comments starts with <!—and ends with -->. Everything with in these will be ignored by the parsers and will not be parsed.

Syn: <!-- this is comments -->

Following points should be remembered while using comments: do not use double hyphens, never place inside entity declaration or within any tag, never place before XML declaration

Document Type Declaration(DTD):

XML allows to create new tags and have meaning if it has some logical structure created using set of related tags. <!DOCTYPE > is used to specify the logical structure of XML document by imposing constraints on what tags can be used and where. DTD may contain Name of root element, reference to external DTD, element and entity declarations.

2. Body:

This portion of XML document contains textual data marked up by tags. It must have one element called Document or Root element, which defines content in the XML document. Root element must be the top-level element in the document hierarchy and there can be one and only one root element.

Ex: <?xml version="1.0"?>

```
<book>
    <title>WT</title>
    <author>Uttam Roy</author>
    <price>500</price>
</book>
```

In this document, the name of root element id <book> which contains sub tags <title>, <author> and <price>. Each of these tags contains text "WT", "Uttam Roy" and "500" respectively.

XML Elements

An XML element consists of starting tag, an ending tag and its contents and attributes. The contents may be simple text or other element or both. XML tags are very much similar to that of HTML tags. A tag begins with less than(<) and ends with greater than(>) character. It takes the form <tag-name> and must have corresponding ending tag(</tag-name>). An element consists of opening tag, closing tag and contents. Few tag may not contain any content and hence know as Empty elements. According to the well-formedness constraint, every XML element must have closing tag. XML provides two ways for XML empty elements as follows:

*Syn:
</br> or
*

Following are the rules that need to be followed for XML elements:

- An element *name* can contain any alphanumeric characters. The only punctuation allowed in names are the hyphen (-), under-score (_) and period (.)
- Names are case sensitive. For example Address, address, ADDRESS are different names
- Element start and end tag should be identical
- An element which is a container can contain text or elements as seen in the above example

Attributes: Attributes are used to describe elements or to provide more information about elements. They appear in the starting tag of element. The syntax of specifying an attribute in element is:

Syn: <element-name attribute-name="value">...</elment-name>

Ex: <employee gener="male">ABCD</employee>

There is no strict rules that describes when to use elements and when to use attributes. However, it is recommended not to use attributes as far as possible due to following reasons:

- Too many attributes reduce readability of XML document
- Attributes cannot contain multiple values, but elements can
- Attributes are not easily extendable
- Attributes cannot represent logical structure, but elements together with their child elements can
- Attributes are difficult to access by parsers

- Attribute values are not easy to check against DTD

Well-formed XML:

An XML document is said to be well-formed if it contains text and tags that conform with the basic XML well-formedness constraints. XML can extend existing documents by creating new elements that fit their applications. The only thing is to remember the well-formedness constraints. The following rules must be followed by XML documents:

- An XML document must have one and only one root element
- All tags must be closed
- All tags must be properly nested
- XML tags are case-sensitive
- Attributes must always be quoted
- Certain characters are reserved for processing like pre-defined entities

Pre-defined Entities: W3C specification defined few entities each of which represents a special character that cannot be used in XML document directly. All XML processors must recognize those entities, whether they are declared or not.

Entity Name	Entity Number	Description	Character
&lt;	&#60;	Less than	<
&gt;	&#62;	Greater than	>
&amp;	&#38;	Amprersand	&
&quot;	&#34;	Quotation mark	“
&apos;	&#39;	Apostrophe	‘

Valid XML

Well-formed XML documents obey only basic well formedness constraints. So, valid XML documents are those that are well formed and comply with rules specified in DTC or Schema.

Name Space

XML was developed to be used by many applications. If many applications want to communicate using XML documents, problems may occur. In XML document, element and attribute names are selected by developers. In some cases two different documents may have same root element. For example, both client.xml and server.xml contains same root tag <config> as shown below.

Client.xml	**Server.xml**
<config> <version>1.0</version> </config>	<config> <version>1.0</version> </config>

XML namespace provides simple, straightforward way to distinguish between element names in XML document. Namespace suggests to use prefix with every element as follows:

Client.xml	**Server.xml**
<c:config> <c:version>1.0</c:version> </c:config>	<s:config> <s:version>1.0</s:version> </s:config>

Uniform Resource Identifier(URI) is used to guarantee the prefixes used by different developers. In general URL are used to choose unique name. But, URL must be prefixed for each tag instead of them we use prefix. Prefixes are just shorthand placeholders of URLs. Association of prefix and URL is done in the starting tag using reserved XML attribute *xmlns*.

Syn: xmlns:prefix=”URI”

Name Space Rules: The *xmlns* attribute identifies namespace and makes association between prefix and created namespace. Many prefixes may be associated with one namespace.

Default Namespace: Namespaces may not have their associated prefixes and are called default namespace. In such cases, a blank prefix is assumed for element and all of its descendants.

Document Type Declaration (DTD)

XML Schema languages:

Schema is an abstract representation of object characteristics and its relationship to other objects. An XML schema represents internal relationship between elements and attributes in XML document. It defines structure of XML documents by specifying list of valid elements and attributes. XML schema language is a formal language to express XML schemas. Most popular and primary schema languages are: DTD and W3C Schema.

1. Document Type Declaration(DTD):

It is one of the several XML schema languages and was introduced as part of XML 1.0. Even though DTD is not mandatory for an application to read and understand XML document, many developers recommend writing DTDs for XML applications. Using DTD we can specify various elements types, attributes and their relationship with in another. Basically DTD is used to specify set of rules for structuring data in any XML file.

1.1 Using DTD in XML document:

To validate XML document against DTD, we must tell validator where to find DTD so that it knows rules to be verified during validation. A Document Type Declaration is used to make such link and DOCTYPE keyword is used for this purpose. There are three ways to make this link: Internal DTD, External DTD and Combined internal and external.

1. Internal DTD:

When we embed DTD in XML document, DTD information is included within XML document itself. Specifically, DTD information is placed between square brackets in DOCTYPE declaration. The general syntax of internal DTD is

Syn: *<!DOCTYPE root-element [*
element-declarations
]>

Where *root-element* is the name of root element and *element-declarations* is where we declare the elements. Since every XML document must have one and only one root element, this is also structure definition of the entire document. Here, DOCTYPE must be in uppercase, document type declaration must appear before first element and name following word DOCTYPE i.e. root-element must match with name of root element.

Advantage of internal DTD is that we have to handle only single xml document instead of many which is useful for debugging and editing. It is a good idea to use with smaller sized documents. Problem of internal DTD is that it makes documents difficult to read for big sized document.

Ex: <?xml version="1.0" ?>

```
<!DOCTYPE bookstore [
        <!ELEMENT bookstore (book*)>
        <!ELEMENT book (title,author,price)>
        <!ELEMENT title (#PCDATA)>
        <!ELEMENT author (#PCDATA)>
        <!ELEMENT publisher (#PCDATA)>
        <!ELEMENT price (#PCDATA)>
]>
<bookstore>
    <book>
    <title>WT</title>
```

```
<author>Uttam Roy</author>
<publisher>Oxford</publisher>
<price>500</price>
</book>
<book>
<title>AJ</title>
<author>Schildt</author>
<publisher>TMH</publisher>
<price>200</price>
</book>
</bookstore>
```

2. External DTD:

Another way of connection DTD to XML document is to reference it with in XML document i.e. create separate document, put DTD information there and point to it from XML document. The general syntax for external DTD is.

Syn: *<!DOCTYPE root-element SYSTEM | PUBLIC "uri">*

Where *uri* is the Uniform Resource Identifier of the *.dtd* file. This declaration states that we are going to define structure of root-element of XML document and its definition can be found from *uri* specified like *book.dtd*. both *xml* and *dtd* files should be kept in same directory.

Ex:

book.xml

```
<?xml version="1.0" ?>
<!DOCTYPE book SYSTEM "book.dtd">
<bookstore>
<book>
        <title>WT</title>
        <author>Uttam Roy</author>
        <publisher>Oxford</publisher>
        <price>500</price>
</book>
<book>
        <title>AJ</title>
        <author>Schildt</author>
        <publisher>TMH</publisher>
        <price>200</price>
</book>
</bookstore>
```

book.dtd

```
<!ELEMENT bookstore (book*)>
<!ELEMENT book (title,author,price)>
<!ELEMENT title (#PCDATA)>
<!ELEMENT author (#PCDATA)>
<!ELEMENT publisher (#PCDATA)>
<!ELEMENT price (#PCDATA)>
```

Location of DTD need not always be local file, it can be any valid URL. Following declaration for XHTML uses PUBLIC DTD:

Syn: *<!DOCTYPE HTML PUBLIC '-//W3C//DTD HTML 4.0 Transitional//EN'>*

Disadvantage of using separate DTD is we have to deal with two documents.

3. Combining Internal and External DTD:

External DTD are useful for common rules for set of XML documents, whereas internal DTDs are beneficial for defining customized rules for specific document. XML allows to combine both internal and external DTD for complete collection of rules for given document. The general form of such DTD is:

Syn: *<!DOCTYPE root-element SYSTEM | PUBLIC "uri" [DTD declarations…] >*

Ex:

```
<?xml version="1.0" ?>
<!DOCTYPE book SYSTEM "book.dtd"
[       <!ELEMENT excl '&#21;'>
]>
<msg>Hello, World&excl; </msg>
```

1.2 DTD validation:

We'll use Java based DTD validator to validate the *bookstore.xml* against the *books.dtd*

DTDValidator.java

```
import java.io.*;
import javax.xml.parsers.*;
import org.w3c.dom.*;
public class DTDValidator
{
  public static void main(String[] args) {
    try {
                DocumentBuilderFactory f = DocumentBuilderFactory.newInstance();
                f.setValidating(true); // Default is false
                Document d = f.newDocumentBuilder().parse(arg[0]);
    }
        catch (Exception e) {       System.out.println(e.toString());          }
 }
```

1.3 Element Type Declaration:

Elements are primary building blocks in XML document. Element type declaration set the rules for type and number of elements that may appear in XML document, what order they may appear in.

Syn: *<!ELEMENT element-name type>*

Or

<!ELEMENT element-name (content)>

Here, *element-name* is name of element to be defined. The *content* could include specific rule, data or another element(s). The keyword ELEMENT must be in upper case, element names are case sensitive, all elements used in XML must be declared and same name cannot be used in multiple declarations.

In DTD, elements are classified depending upon their content as follows:

- **Standalone/Empty elements**: these elements cannot have any content and may have attributes. They can be declared using type keyword EMPTY as follows:
  Syn: *<!ELEMENT element-name EMPTY>* *Ex:* *<!ELEMENT br EMPTY>*
- **Unrestricted elements**: element with content can be created using content type as ANY. Keyword ANY indicates that *element-name* can be anything including text and other elements in any order and any number of times.
  Syn: *<!ELEMENT element-name ANY>* *Ex:* *<!ELEMENT msg ANY>*
- **Simple elements**: simple element cannot contain other elements, but contains only text.
  Syn: *<!ELEMENT element-name (#PCDATA)>* *Ex:* *<!ELEMENT author (#PCDATA)>*
  This interprets that element *element-name* can have only text content. The type of text id PCDATA means Parsed Character DATA and the text will be parsed by parser and will examined for entities and markups and expanded as and when necessary. Sometimes we can use CDATA means Character DATA in place of PCDATA.
- **Compound elements**: compound elements ca contains other elements known as child elements.
  Syn: *<!ELEMENT element-name (child-elements-names)>*
  Ex: <!ELEMENT book (title, author, price)>

Occurrence Indicator: sometimes it is necessary to specify how many times element may occur in document which is done by Occurrence Indicator. When no occurrence indicator is specified, child element must occur exactly once in XML document

Operator	Syntax	Description
None	a	Exactly one occurrence of a
* (Astrisck)	a*	Zero or more occurrences of a i.e. any number of times
+ (Plus)	a+	One or more occurrences of a i.e. at least once

? (Question mark)	a?	Zero or one occurrences of a i.e. at most once

Declaring multiple children: elements with multiple children are declared with names of the child elements inside parenthesis. The child elements must also be declared.

Operator	Syntax	Description
, (Sequence)	a , b	a followed by b
| (Choice)	a | b	a or b
() (Singleton)	(expression)	Expression is treated as a unit

1.4 Attribute Declaration:

Attributes are used to associate name, value pairs with elements. They are useful when we want to provide some additional information about elements content. The declaration starts with ATTLIST followed by name of the element the attributes are associated with and declaration of individual declarations:

Syn: <!ATTLIST element-name attribute-name attribute-type default-vale>

Ex: <!ATTLIST employee geneder CDATA 'male'/>

Here, ATTLIST must be in upper case. The *default-value* can be any of the following:

- **Default**: in this case, attribute is optional and developer may or may not provide this attribute. When attribute is declared with default value, the value of attribute is whatever value appears as attributes content in instance document.
  Ex: <!ATTLIST line width CDATA '100'/>
- **#REQUIRED**: attribute must be specified with value every time enclosing element is listed
  Ex: <!ATTLIST line width CDATA #RQUIRED />
- **#FIXED**: attribute is optional and is used to ensure that the attributes are set to particular values.
  Ex: <!ATTLIST line width CDATA #FIXED '50'/>
- **#IMPLIED**: similar to that of default attribute except that no default value is provided by XML
  Ex: <!ATTLIST line width CDATA '#IMPLIED' />

Attribute types:

The *attribute-type* can be one among string or CDATA, tokenized and enumerated types.

- **String type**: may take any literal string as value and can be declared using keyword CDATA. An attribute of CDATA type can contain any character if it conforms to well formedness constraints. Some it can contains escape characters like <, > etc.
- **Tokenized type**: following tokenized types are available
  - **ID**: it is globally unique identifier of attribute, this means value of ID attribute must not appear more than once throughout the XML document and resembles primary key concept of data base.
    <!ATTLIST question no ID #REQUIRED>
  - **IDREF**: similar to that of foreign key concept in databases and is used to establish connections between elements. IDREF value of the attribute must refer to ID value declared
    <!ATTLIST answer qno IDREF #REQUIRED>
  - **IDREFS**: it allows a list of ID values separated by white spaces
    <!ATTLIST answer qno IDREFS #REQUIRED>
  - **NMTOKEN**: it restricts attributes value to one that is valid XML name means allows punctuation marks and white spaces.
    <!ATTLIST car serial NMTOKEN #REQUIRED>
  - **NMTOKENS**: can contains same characters and white spaces as NMTOKEN. White space includes one or more characters, carriage returns, line feeds, tabs
    <!ATTLIST car serial NMTOKENS #REQUIRED>
  - **ENTITY**: refers to external non parsed entities
    <!ATTLIST car serial ENTITY #REQUIRED>
  - **ENTITIES**: values of ENTITIES attribute may contain multiple entity names separated by one or more white spaces
    <!ATTLIST car serial ENTITIES #REQUIRED>

- **Enumerated type**: enumerated attribute values are used when we want attribute value to be one of fixed set of values. There are two kinds of enumerated types:
  - **Enumeration**: attributes are defined by a list of acceptable values from which document author must choose a value. The values are explicitly specified in declaration, separated by pipe(|)
    <!ATTLIST employee gender (male|female) #REQUIRED>
  - **Notation**: it allows to use value that has been declared a NOTATION in DTD. Notation is used to specify format of non-XML data and common used is to describe MIME types like image/gif, image/jpeg etc.
    <!NOTATION jpg SYSTEM 'image/gif'>
    <!ENTITY logo SYSTEM 'logo.jpg' NDATA jpg>
    <!ATTLIST photo format NOTATION (jpg) #IMPLIED>

1.5 Entity Declaration:

Entities are variables that represent other values. If a text contains entities, the value of entity is substituted by its actual value whenever the text is parsed. Entity must be defined in DTD declaration to use custom entites in XML document. Built-in entities and character entities do not require any declaration. There are two types of entity declarations: General entity and Paramter entity. Each type can be again Parsed or Unparsed.

- **General and Parameter entities**: General entities are used with in the document content. Parameter entities are parsed entites used with in DTD. These two types of entites use different forms of references and are recognized in different contexts. They occupy different namespaces
- **Parsed and Unparsed entities**: Parsed entity is an entity whose content is parsed and checked for well formedness constraint during parsing procedure. Unparsed entity is resource whose contents may or may not be text and if text may not be XML. It means there are no constrainst on conetents of unparsed entities. Each unparsed entity has associated notation, identified by name.

1.5.1 General Entity Declaration:

There are three kinds of general entitity declarations:

- **Internal parsed**: an internal entity declaration has following form
  Syn: *<!ENTITY entity-name "entity-value">*
  Ex: *<!ENTITY UKR "Uttam Kumar Roy">*
  The entity UKR can be referred in XML document as follows:
  <author>&UKR;</author>
  This will be interpreted as : <author>Uttam Kumar Roy</author>
- **External parsed**: external entities allow an XML document to refer to external resource.Parse external entites refer to data that an XML parser has to parse and used for long replacement text which is kept in another file. There are two type of external parsed entities: Public and Private. Public external enties are identified by PUBLIC keyword and intended for general use. Private external entites are identified by SYSTEM keyword and are intended for use by single author or group of authors.
  Syn: *<!ENTITY entity-name SYSTRM | PUBLIC "URI">*
  Ex: *<!ENTITY author SYSTEM "author.xml">*
- **External unparsed**: refer to data that an XML processor does not have to parse. For example, there are numerious ASCII text files, JPRG photographs etc. None of theseare well formed XML. Mechanism that XML suggests for embedding these files is enternal unparsed entity. They can be either private or public.
  Syn: <!ENTITY logo SYSTEM "logo.jpg" NDATA jpeg>

1.5.2 Parameter Entity Declaration:

DTD supports another kind of entity called parameter entity. It is used within DTD which allows to assign collection of elements, attributes and attribute values to name and refer them using name instead of explicitly listimg them every time they are used.

- **Internal parsed entity**: it has following form
  Syn: <!ENTITY % entity-name entity-definition>
  Ex: <!ENTITY % name "firstname, middlename, lastname">
  This parameter entity describes portion of content model that can be referenced with in elements ind DTD. They can be referenced using entity name between precent sign(%) and semicolor(;).
  Syn: %Entity-name; Ex: %name;
- **External parsed entity**: These are used to link external DTDs. It may be private or public and is identified by keywords SYSTEM and PUBLIC. Private entites are intended for use by single author where as public entities can be used by anyone.
  Syn: <!ENTITY % entity-name SYSTEM | PUBLIC "URI">

2. XML Schema:

XML Schema Definition commonly known as XSD, is a way to describe precisely the XML language. XSD check the validity of structure and vocabulary of an XML document against the grammatical rules of the appropriate XML language.An XML document can be defined as:

- **Well-formed**: If the XML document adheres to all the general XML rules such as tags must be properly nested, opening and closing tags must be balanced and empty tags must end with '/>', then it is called as *well-formed*.
- **Valid**: An XML document said to be valid when it is not only *well-formed*, but it also conforms to available XSD that specifies which tags it uses, what attributes those tags can contain and which tags can occur inside other tags, among other properties.

Limitaions of Document Type Declaration (DTD)

- There is no bult-in data type in DTDs
- No new data types can be created in DTDs
- The use of cardinatlity in DTDs is limited
- Namespaces are not supported
- DTDs provide very limited support for modularity and reuse
- We can not put any restrictions on text content
- Defaults for elements can not be specified
- We have very little control over mixed content
- DTDs are written in strange format and are difficult to validate

Strengths of XML Schema(XSD)

- XML schema provided much grater specification than DTDs
- They support large number of built-in data types
- They support namespaces
- They are extensible to future additions
- They support uniqueness and referencial integrity constraints in much better way
- It is easier to define data restrictions

2.1 XSD Structure:

An XML XSD is kept in a separate document and then the document having extension *.xsd* and is be linked to the XML document to use it. Schema is the root element of XSD and it is always required.

Syn: *<?xml version="1.0"?>*
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
...
</xs:schema>

Above fragement specifies that elements and datatypes used in the schema are defined in "http://www.w3.org/2001/XMLSchema" namespace and these elements/data types should be prefixed with *xs*. Similarly, XSD can be linked to xml file with following syntax:

Syn: <roo-tag xmlns:xsi=" http://www.w3.org/2001/XMLSchema-instance" xsi:schemaLocation="uri">
Above fragement specifies default namespace declaration i.e. "http://www.w3.org/2001/XMLSchema-instance". This namespace is used by schema validator check that all the elements are part of this namespace. It is optional. Use schemaLocation attribute to specify the location of the *xsd* file.

Ex: book.xml

```
<?xml version="1.0" ?>
<bookstore xsi:schemaLocation="book.xsd" xmlns:xsi=" http://www.w3.org/2001/XMLSchema-instance"
>
<book>
        <title>WT</title>
        <author>Uttam Roy</author>
        <publisher>Oxford</publisher>
        <price>500</price>
</book>
<book>
        <title>AJ</title>
        <author>Schildt</author>
        <publisher>TMH</publisher>
        <price>200</price>
</book>
</bookstore>
```

book.xsd

```
<?xml version="1.0"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
 <xs:element name="bookstore">
<xs:complexType>
        <xs:sequence>
                <xs:element name="book">
                <xs:complexType>
                        <xs:sequence>
                        <xs:element name="title" type="xs:string"/>
                        <xs:element name="author" type="xs:string"/>
                        <xs:element name="publisher" type="xs:string"/>
                        <xs:element name="price" type="xs:integer"/>
                        </xs:sequence>
                </xs:complexType>
                </xs:element>
        </xs:sequence>
</xs:complexType>
</xs:element>
```

2.2 XSD Validation:

We'll use Java based XSD validator to validate the *bookstore.xml* against the *books.xsd.*

XSDValidator.java

```
import java.io.*;
import javax.xml.*;
import javax.xml.transform.dom.*;
import   javax.xml.parsers.*;
import javax.xml.validation.*;
import org.w3c.dom.*;
```

```
public class XSDValidator {
  public static void main(String[] args) {
    try {
      SchemaFactory factory =
        SchemaFactory.newInstance(XMLConstants.W3C_XML_SCHEMA_NS_URI);
          Schema schema = factory.newSchema(new File(args[1]));
           Validator validator = schema.newValidator();
          DocumentBuilder parser=DocumentBuilderFactory.newInstance().newDocumentBuilder();
          Document document=parser.parse(new File(args[0]));
           validator.validate(new DOMSource(document));
    }
    catch (Exception e)
   {          System.out.println("Exception: "+e.getMessage());        }
}
```

2.3 Element declaration:

Elements are primary building blocks in XML document. Element type declaration can be done using <xs:element> tag with following syntax.

Syn: <xs:element name="element-name" type="element-type">

Ex: <xs:element name="title" type="xs:string">

Each element declaration within the XSD has mandatory attribute *name*. the value of this name attribute is the element name attribute is the element name that will appear in the XML document. Element definition may also have optional *type* attribute, which provides description of what can be containted within the element when it appears in XML document. Every XML document must have root element. This root element must be declared first in schema for conforming XML documents.

```
Ex: <!xml version="1.0"?>
        <xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">
            <xs:element name="bookstore">
            </xs:element>
        </xsd:schema>
```

2.3.1 Declarting simple elements:

Simple type elements can contain only text and/or data. They can not have child elements or attributes, and can not be empty. Simple elements are defined as follows:

Syn: <xs:element name="element-name" type="element-type">

Ex: <xs:element name="title" type="xs:string"/>

The value of *type* attributes specifies an elements content type and can be any simple type. This attribute can be any complex type.

- **Default Value:** Simple Element can have default value that specifies the default content to be used when not content is specified. When an element is declared with default value, the value of the element is whatever value appears as elements content in instance document. Following example illustrates this:

  <xs:element name="gender" type="xs:boolean" default="true" />

- **Fixed Value:** Simple Element can also have optional fixed value. Fixed attribute is used to ensure that elements content is always set to particular value. Consider the following syntax:

  <xs:element name="branch" type="xs:string" fixed="IT" />

- **Occurance indicators:** an element have two optional attributes : minOccurs and maxOccurs. They are used to specify the number of times an element can occur in XML document.
  - **minOccurs**: this attribute specifies minimum number of times an element can occur. The following is example of usage of this attribute:

    <xs:element name="option" type="xs:string" minOccurs="0"/>

TutorialsDuniya.com

Download FREE Computer Science Notes, Programs, Projects, Books PDF for any university student of BCA, MCA, B.Sc, B.Tech CSE, M.Sc, M.Tech at https://www.tutorialsduniya.com

- Algorithms Notes
- Artificial Intelligence
- Android Programming
- C & C++ Programming
- Combinatorial Optimization
- Computer Graphics
- Computer Networks
- Computer System Architecture
- DBMS & SQL Notes
- Data Analysis & Visualization
- Data Mining
- Data Science
- Data Structures
- Deep Learning
- Digital Image Processing
- Discrete Mathematics
- Information Security
- Internet Technologies
- Java Programming
- JavaScript & jQuery
- Machine Learning
- Microprocessor
- Operating System
- Operational Research
- PHP Notes
- Python Programming
- R Programming
- Software Engineering
- System Programming
- Theory of Computation
- Unix Network Programming
- Web Design & Development

Please Share these Notes with your Friends as well

- **maxOccurs**: this attribute specifies maximum number of times an element can occur. The declaration of element will be as follows:
  <xs:element name="option" type="xs:string" maxOccurs="10"/>

Schema	DTD	Meaning
minOccurs='0', maxOccurs='unbounded'	*	Zero or more
minOccurs='1', maxOccurs='unbounded'	+	One or more
Minoccurs='0'	?	Optional
None	None	Exactly one

2.3.2 Declarting complex elements:

Complex types can be named or can be anonymous. They are associated with complex elements in the same manner, typically using a type definition and an element declaration. By default, complex type elements have complex content i.e. they have child elements. Complex type elements can be limited to having simple content i.e. they contain only text. General form of element declaration is:

Syn: <xs:complexType name="complex-type-name"><xs:sequence>
</xs:sequence></xs:complexType>

Ex: *<xs:complexType name="sName"><xs:sequence>*
<xs:element name="first" type="xs:string"/>
<xs:element name="middle" type="xs:string"/>
<xs:element name="lase" type="xs:string"/>
</xs:sequence></xs:complexType>

2.3 Atribute declaration:

Attrbibutes are used to describe properties of an element. Attributes themselves are always declared as simple types as follows:

Syn: *<xs:attribute name"attribute-name" type="attribute-type">*

Ex: *<xs:attribute name="id" type="xs:string'/>*

Simple types can not have attributs. Element that have attributes are complex types. So, attributes declaration always occurs as part of complex type declaration, immediately after its content model.

Ex: *<xs:complexType name="sName"><xs:sequence>*
<xs:element name="first" type="xs:string"/>
<xs:element name="middle" type="xs:string"/>
<xs:element name="lase" type="xs:string"/>
</xs:sequence>
<xs:attribute name="id" type="xs:string"/>
</xs:complexType>

A part from this simple definition, there can be additional specifications for attributes:

- **Attribute element properties:**
  - **use**: possible values are optional, required and prohibited.
    <xs:attribute name="id" type="xs:string" use="required"/>
  - **default**: this specifies the value to be used if attribute is not specified
    <xs:attribute name="gender" type="xs:boolean" default="false"/>
  - **fixed**: it specifies that attribute, if it appears must always have fixed value specified. If the attribute does not appear, the schema processor will provide attribute with value specified here.
    <xs:attribute name="unit" type="xs:boolean" default="rpm"/>
- **Order Indicators**
  - **All**: Child elements can occur in any order.
    <xs:all>
    <xs:element name="first" type="xs:string"/>
    <xs:element name="middle" type="xs:string"/>
    <xs:element name="last" type="xs:string"/>

```
</xs:all>
```

- **Choice**: Only one of the child element can occur.

```
<xs:choice>
        <xs:element name="first" type="xs:string"/>
        <xs:element name="middle" type="xs:string"/>
        <xs:element name="last" type="xs:string"/>
</xs:choice>
```

- **Sequence**: Child element can occur only in specified order.

```
<xs:sequence>
        <xs:element name="first" type="xs:string"/>
        <xs:element name="middle" type="xs:string"/>
        <xs:element name="last" type="xs:string"/>
</xs:sequence>
```

- **Occurence Indicators**
  - **maxOccurs** - Child element can occur only maxOccurs number of times.
  - **minOccurs** - Child element must occur minOccurs number of times.
- **Group Indicators**
  - **Group**: a set of related elements can be created using this indicator. the general form for creating an element group is as follows:

```
Syn:  <xs:group name="group-name">  ...  </ xs:group>
Ex:   <xs:group name="personInfo">
          <xs:element name="first" type="xs:string"/>
          <xs:element name="middle" type="xs:string"/>
          <xs:element name="last" type="xs:string"/>
      </ xs:group>
```

  - **attributeGroup**: XML Schema provides this element, which is used to group a set of attributes declarations so that they can be incorporated into complex types definitions with syntax:

```
Syn:  <xs:attributeGroup name="group-name">  ...  </ xs:attributeGroup>
Ex:   <xs: attributeGroup name="personInfo">
          <xs:element name="first" type="xs:string"/>
          <xs:element name="middle" type="xs:string"/>
          <xs:element name="last" type="xs:string"/>
      </ xs: attributeGroup>
```

2.4 Annotations declaration:

XML schema provides three annotation elements for documentation purposes in XML schema instance. They provide a way to write realistic ans structured comments for the benefit of applications. An annotation is represented by <annotation> element which typically appears at the beginning of most schemas. However, it can appear inside any complex element definition. It can contain only two elements <appinfo> and <documentation> any number of times. Following is an example:

```
<xs:annotation>
        <xs:documentation> <author>Uttam Roy</author></xs:documentation>
        <xs:appinfo><version>2.1</version></xs:appinfo>
</xs.annotation>
```

2.5 XML Scheme data types:

An element is limited by its type. Depending upon content model, elements are categorized as Simple or Complex type. A simple type can further be divided into three types: Atomic, List and Union. XML schema 1.0 specification provides about 46 built in data types. All built-in data types except anyType are considered as simple types. Some of the built-in data types are as follows:

2.5.1 XSD String Data Types:

String data types are used to represent characters in the XML documents.

- **<xs:string>:** The <xs:string> data type can take characters, line feeds, carriage returns, and tab characters. XML processor do not replace line feeds, carriage returns, and tab characters in the content with space and keep them intact. For example, multiple spaces or tabs are preserved during display.
  Syn: <xs:element name="elment-name" type="xs:string"/>
  Ex: < xs:element name="sname" type="xs:string"/>
- **<xs:token>:** The <xs:token> data type is derived from <string> data type and can take characters, line feeds, carriage returns, and tab characters. XML processor removes line feeds, carriage returns, and tab characters in the content and keep them intact. For example, multiple spaces or tabs are removed during display.
  Syn: <xs:element name="element-name" type="xs:token"/>

Following is the list of commonly used data types which are derived from <string> data type.

- **ID:** Represents the ID attribute in XML and is used in schema attributes.
- **IDREF:** Represents the IDREF attribute in XML and is used in schema attributes.
- **Language:**Represents a valid language id
- **Name:** Represents a valid XML name
- **NMTOKEN:** Represents a NMTOKEN attribute in XML and is used in schema attributes.
- **normalizedString:** Represents a string that does not contain line feeds, carriage returns, or tabs.
- **String:** Represents a string that can contain line feeds, carriage returns, or tabs.
- **Token:** Represents a string that does not contain line feeds, carriage returns, tabs, leading or trailing spaces, or multiple spaces

2.5.2 XSD Date & Time Data Types:

Date and Time data types are used to represent date and time in the XML documents.

- **<xs:date> data type:** The <xs:date> data type is used to represent date in YYYY-MM-DD format. Each component is required. **YYYY**- represents year, **MM**- represents month, **DD**- represents day
  Syn: <xs:element name="birthdate" type="xs:date"/>
  Ex: <birthdate>1980-03-23</birthdate>
- **<xs:time> data type**: The <xs:time> data type is used to represent time in hh:mm:ss format. Each component is required. **hh**- represents hours, **mm**- represents minutes, **ss**- represents seconds
  Syn: <xs:element name="startTime" type="xs:time"/>
  Ex: <startTime>10:20:15</startTime>
- **<xs:datetime> data type:** The <xs:datetime> data type is used to represent date and time in YYYY-MM-DDThh:mm:ss format. Each component is required. **YYYY**- represents year, **MM**- represents month, **DD**- represents day, **T**- represents start of time section, **hh**- represents hours, **mm**- represents minutes, **ss**- represents seconds
  Syn: <xs:element name="startTime" type="xs:datetime"/>
  Ex: <startTime>1980-03-23T10:20:15</startTime>
- **<xs:duration> data type:** The <xs:duration> data type is used to represent time interval in PnYnMnDTnHnMnS format. Each component is optional except P. **P**- represents year, **nY**- represents month, **nM**- represents day, **nD**- represents day, **T**- represents start of time section, **nH**- represents hours, **nM**- represents minutes, **nS**- represents seconds
  Syn: <xs:element name="period" type="xs:duration"/>
  Element usage in xml to represent period of 6 years, 3 months, 10 days and 15 hours.
  Ex: <period>P6Y3M10DT15H</period>

Following is the list of commonly used date data types .

- **Date:** Represents a date value
- **dateTime:** Represents a date and time value
- **duration:** Represents a time interval
- **gDay:** Represents a part of a date as the day (DD)

- **gMonth:** Represents a part of a date as the month (MM)
- **gMonthDay:** Represents a part of a date as the month and day (MM-DD)
- **gYear:** Represents a part of a date as the year (YYYY)
- **gYearMonth:** Represents a part of a date as the year and month (YYYY-MM)
- **time:** Represents a time value

2.5.3 XSD Numeric Data Types:

Numeric data types are used to represent numbers in the XML documents.

- **<xs:decimal> data type:** The <xs:decimal> data type is used to represent numeric values. It support decimal numbers upto 18 digits.
  Syn: *<xs:element name="score" type="xs:decimal"/>*
  Ex: *<score>9.12</score>*
- **<xs:integer> data type:** The <xs:integer> data type is used to represent integer values.
  Syn: *<xs:element name="score" type="xs:integer"/>*
  Ex: *<score>9</score>*

Following is the list of commonly used numeric data types .

- **Byte:** A signed 8 bit integer
- **Decimal:** A decimal value
- **Int:** A signed 32 bit integer
- **Integer:** An integer value
- **Long:** A signed 64 bit integer
- **negativeInteger:** An integer having only negative values (..,-2,-1)
- **nonNegativeInteger:** An integer having only non-negative values (0,1,2,..)
- **nonPositiveInteger:** An integer having only non-positive values (..,-2,-1,0)
- **positiveInteger:** An integer having only positive values (1,2,..)
- **short:** A signed 16 bit integer
- **unsignedLong:** An unsigned 64 bit integer
- **unsignedInt:** An unsigned 32 bit integer
- **unsignedShort:** An unsigned 16 bit integer
- **unsignedByte:** An unsigned 8 bit integer

2.5.4 XSD Miscalleneous Data Types

Other Important Miscellaneous data types used are boolean, binary and anyURI.

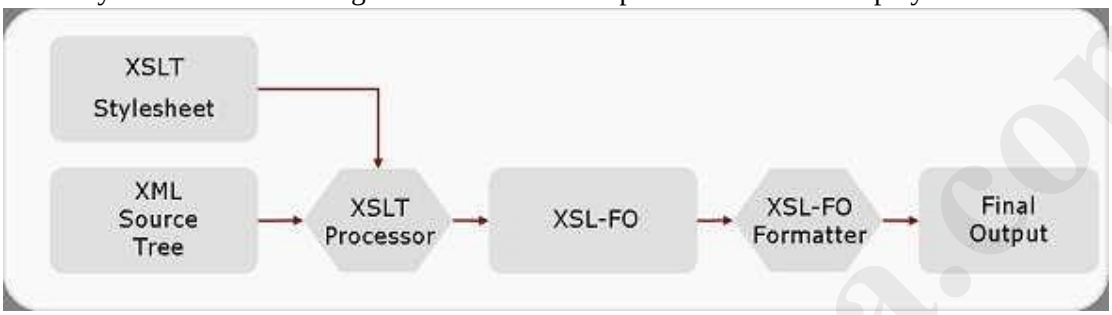
- **<xs:boolean> data type:** The <xs:boolean> data type is used to represent true, false, 1 (for true) or 0 (for false) value.
  Syn: *<xs:element name="pass" type="xs:boolean"/>*
  Ex: *<pass>false</pass>*
- **Binary data types:** The Binary data types are used to represent binary values. Two binary types are common in use. **base64Binary**- represents base64 encoded binary data, **hexBinary** - represents hexadecimal encoded binary data
  Syn: *<xs:element name="blob" type="xs:hexBinary"/>*
  Ex: *<blob>9FEEF</blob>*
- **<xs:anyURI> data type**: The <xs:anyURI> data type is used to represent URI.
  Syn: *<xs:attribute name="resource" type="xs:anyURI"/>*
  Ex: *<image resource="http://www.tutorialspoint.com/images/smiley.jpg" />*

eXtensible Stylesheet Language Transformation(XSLT):

XML documents contain self-describing and structured data. The set of tags and their structure varies widely in different applications. Web browsers can not display such non-HTML files as they have no prior knowledge about the meaning of set of tags used in different XML documents. Users may also want to

generate new XML documents from one or more existing XML documents for processing or sharing of data between different applications. One possible solution is to generate separate XML document such that the former contains only insensitive data. XSLT comes into play in this scenario.

XSLT, Extensible Stylesheet Language Transformations provides the ability to transform XML data from one format to another automatically. An XSLT stylesheet is used to define the transformation rules to be applied on the target XML document. XSLT stylesheet is written in XML format. XSLT Processor takes the XSLT stylesheet and apply the transformation rules on the target XML document and then it generates a formatted document in form of XML, HTML or text format. This formatted document then is utilized by XSLT formatter to generate the actual output which is to be displayed to the end user.



Following are the main parts of XSL.

- **XSLT** - used to transform XML document into various other types of document.
- **XPath** - used to navigate XML document.
- **XSL-FO** - used to format XML document.

In general following tasks can be performed using XSLT: Constant text generation, Reformatting of information, sensitive information suppression, adding new information, copying information and Sorting document with respect to a criteria.

Advantages

- Independent of programming. Transformations are written in a seperate xsl file which is again an XML document.
- Output can be altered by simply modifing the transformations in xsl file. No need to change in any code. So Web designers can edit stylesheet and can see the change in output quickly.

1. Stylesheet strcture:

XSLT files are themselves XML documents and hence must follow the well-formedness constraints. The W3C defined the exact syntax of an XSLT 2.0 document by XML schema. XSLT file starts with XML declaration. Every XSLT file must have either <stylesheet> or <transform> as root element. Following are simple structure of XSLT document:

```
<?xml version="1.0"?>
<xsl:stylesheet version="2.0" xmlns:xsl="http://www.w3.org/1999/XSLT/Transform">
...
</ xsl:stylesheet>
```

Or

```
<?xml version="1.0"?>
<xsl:transform version="2.0" xmlns:xsl="http://www.w3.org/1999/XSLT/Transform">
...
</ xsl: transform >
```

These elements must have the attribute *version* and namespace attribute *xmlns*. Version attribute indicate version of XSLT being used. Namespace attribute distinguishes XSLT elements from other elements. There are different ways to apply XSLT document to XML document. One way to add link to XML document which points to actual XSLT files and lets the browsers do transformation with following declaration:

```
<?xml version ="1.0"?>
```

<?xml-stylesheet type="text/xsl" href="URI">
<root> ... </root>

students.xml	*students.xsl*
<?xml version="1.0"?> *<?xml-stylesheet type="text/xsl" href="students.xsl">* *<class>* *...* *</class>*	*<?xml version="1.0"?>* *<xsl:stylesheet version="2.0" xmlns:xsl="http://www.w3.org/1999/XSLT/Transform">* *...* *</ xsl:stylesheet>*

2. XSLT Elements:

An XSLT file contains elements, which instruct processor how an XML document is to be transformed. It may contain elements that are not defined by XSLT. In such cases, XSLT processor does not process these non-XSLT elements and add them to the ouput in the same order they occurred in the source XSLT document. This means that the transformed XML document may use original mark-ups as well as new mark-ups.

Element	**Description**
stylesheet	Defines the root element of a style sheet
transform	Defines the root element of a style sheet
template	Rules to apply when a specified node is matched
apply-templates	Applies a template rule to the current element or to the current element's child nodes
call-template	Calls a named template
element	Creates an element node in the output document
variable	Declares a local or global variable
param	Declares a local or global parameter
value-of	Extracts the value of a selected node
attribute	Adds an attribute
attribute-set	Defines a named set of attributes
if	Contains a template that will be applied only if a specified condition is true
choose	Used in conjunction with <when> and <otherwise> to express multiple conditional tests
when	Specifies an action for the <choose> element
for-each	Loops through each node in a specified node set
import	Imports the contents of one style sheet into another. **Note:** An imported style sheet has lower precedence than the importing style sheet
include	Includes the contents of one style sheet into another. **Note:** An included style sheet has the same precedence as the including style sheet
sort	Sorts the output
processing-instruction	Writes a processing instruction to the output
comment	Creates a comment node in the result tree
copy	Creates a copy of the current node (without child nodes and attributes)
copy-of	Creates a copy of the current node (with child nodes and attributes)

3. XSLT templates:

An XSLT document is all about template rules. A template specifies rule and instruction, which is executed when rule matches. The rule is specified by XSLT <template> element. It has attribute *match*, which specifies pattern. The value of match attribute is subset of expression.

Syn: *<xsl:template match="expression">*
...
</xsl:template>

Ex: *<xsl:template match="/">*
<h1>Hello! World.</h1>
</xsl:template>

XSLT document contain single template rule. It has match attribute with expressin "/", which means the document root of any XML document. Ths instruction with in this template specifies the string *Hello! World* has to be added to the output and the resulting document obtained is as follows:

<html> <body><h1>Hello! World.</h1></body> </html>

Applying templates:

In general, if a node matches with template pattern, the templates action part is processed. It is also possible to instruct XSLT processor to process other template rules if any. This is done using <apply-templates> element with following syntax:

```
<xsl:template match="/">
  <xsl:apply-templates/>
</xsl:template>
```

This example states that whenever document root is encountered, XSLT processor has to process all templates that match with document roots children roots. The XSLT engine in turn, compares each child element of document root aginst templates in style sheet and if match is found, it processes the corresponding template.

Processing Sequence and default templates:

When XSLT processor is supplied XML document for transformation using XSLT document, it first creates document tree. Processing always starts from document root of this tree. So, XSLT processor looks for template for it. If no template is found for document root, XSLT processor provides default templates. This default template for document root looks like this:

```
<xsl:template match="/">
  <xsl:apply-templates/>
</xsl:template>
```

The behaviour of default template for any element node looks as follows:

```
<xsl:template match="*">
        <xsl:apply-templates/>
</xsl:template>
```

Default template for text nodes as follows:

```
<xsl:template match="text()">
        <xsl:apply-templates/>
</xsl:template>
```

Default templates and their behavior:

- **Root**: process template for its children
- **Element**: process templates for its children
- **Attribute**: output attribute name and value
- **Text**: output text value
- **Processing instruction**: do nothing
- **Comment**: do nothing

Named templates:

XSLT named templates resemble the functions in any procedural programming language. The <template> element has *name* attribute, which can be used to give name to template. Once template is created this way, it can be called by using <call-template> element and specifying its name.

```
<xsl:template match="/">
        <xsl:call-template name="header"/>
</xsl:template>
```

```
<xsl:template name="header">
        <title>XSLT</title>
</xsl:template>
```

4. Selecting values:

The value of a node can also be added using <value-of> element. Value of node depends on type of the node. For example, the value of text node is the text itself, whereas the value of element node is concatenation of values of all text descendents. If multiple nodes are selected by select attribute, value is concatenation of values of those selected attributes. Consider simple XML document:

```
<book>
        <title>Web Technologies</title>
</book>
```

One can now extract the value of title element using <value-of> elemement as follows:

```
<xsl:template match="/">
  Title:    <xsl:value-of select="book/title"/>
</xsl:template>
```

This XSLT file, on applying previous XML document produces following result:

Title: Web Technologies

Values of different node types:

- **Text**: text of node
- **Element**: concatenation of values of all text descendants
- **Attribute**: attribute value without quotation marks
- **Namespace**: the URI of the namespace
- **Comment:** anything between <!--and -->
- **Processing instruction**: anything between <? and ?>

XSLT has another element <copy-of>, which returnas all selected elements including nested elements and text. Consider the following XSLT document.

```
<xsl:template match="/">
  <xsl:copy-of select="."/>
</xsl:template>
```

When we apply this XSLT document to any XML document, it produces the same XML document. This is because, when root element (/) is selected, <copy-of> copies root element together with all child elements recursively.

5. Varaibale and Parameters:

Named templates resembles the functions in any procedural programming language. Like function, named templates may accept argument. Formal parameters are declard with in template using <param> element as follows:

```
<xsl:template name="add">
        <xsl:param name="a"/>
        <xsl:param name="b"/>
        <xsl:value-of select="$a+$b"/>
</xsl:template>
```

This example defines named template *add*, which takes two parameters *a* and *b*. The purspose of this template is to add two arguments taken and produce the result to output. Arguments can then be passed to template using <with-param> element during template call.

```
<xsl:call-template name="add">
        <xsl:with-param name="a" select="2"/>
        <xsl:with-param name="b" select="4"/>
</xsl:call-template>
```

This code calls template add with parameters 2 and 4. If this XSLT applied to XML document the output will be 6. The scope of formal is with in the template only. XSLT allows to declare anduse variable. Consider the following code:

```
<xsl:template>
        <xsl:variable name="a">4</xsl:variable>
        <xsl:variable name="b"> 6</xsl:variable>
        <xsl:value-of select="$a+$b"/>
</xsl:template>
```

6. Conditional Processing:

There are two types of branching constructs in XSLT: <if> and <choose>

6.1 Using if:

XSLT <if> element has attribute *test*, which takes Boolean expression. If the effective Boolean value of this expression is evaluated to true, the action under <if> construct is followed. The general syntax of <if> construct is as follows:

```
<xsl:if test="condition">
  ...
</xsl:if>
```

The following extracts information about only that book having title as "Web Technologies":

```
<xsl:template match="//book">
  <xsl:if test="@title='Web Technologies'">
        Author: <xsl:value-of select="@author"/>
        Price:  <xsl:value-of select="@price"/>
  </xsl:if>
</xsl:template>
```

6.2 Using choose:

XSLT <choose> element allows us to select particular condition among set of conditions specified by <when> element. The general format of <choose> construct is:

```
<xsl:choose>
  <xsl:when  text="expression1">..</xsl:when>
  <xsl:when  text="expression2">..</xsl:when>
  ...
  <xsl:when  text="expressionN">..</xsl:when>
  <xsl:otherwise>...</xsl:otherwise>
</xsl:choose>
```

Consider the following XML file *result.xml*, containing marks of different students:

```
<result>
        <student><rollno>01</rollno><marks>80</marks></student>
        <student><rollno>02</rollno><marks>70</marks></student>
        <student><rollno>03</rollno><marks>60</marks></student>
        <student><rollno>04</rollno><marks>55</marks></student>
        <student><rollno>05</rollno><marks>77</marks></student>
</result>
```

The following XSLT document displays results of the students:

```
<xsl:choose>
        <xsl:when test="marks &gt; 80 and marks &lt;= 100">A Grade</xsl:when>
        <xsl:when test="marks &gt;  70 and marks  &lt;=  80">B Grade</xsl:when>
        <xsl:when test="marks &gt; 60 and marks  &lt;=  70">C  Grade</xsl:when>
        <xsl:when test="marks &lt;=60">D Grade</xsl:when>
```

```
</xsl:choose>
```

6. Repetition:

XSLT allows <for-each> construct, which can be used to process set of instructions repetedly for different items in sequence. The attribute *select* evaluates sequence of nodes. For each of telements in this sequence, instruction under <for-each> element are processed. Consider the following XML file *result.xml*, containing marks of different students:

```
<result>
        <student><rollno>01</rollno><marks>80</marks></student>
        <student><rollno>02</rollno><marks>70</marks></student>
        <student><rollno>03</rollno><marks>60</marks></student>
        <student><rollno>04</rollno><marks>55</marks></student>
        <student><rollno>05</rollno><marks>77</marks></student>
</result>
```

The following XSLT document displays results of the students:

```
<xsl:for-each select="result">
        Roll No: <xsl:value-of select ="rollno"/><br>
        Marks: <xsl:value-of select ="marks"/><br>
</xsl:for-each>
```

7. Creating nodes and Sequences:

XSLT allows to directly create custom nodes such as element node, text nodes etc. or sequences of nodes and atomic values that appear in output.

7.1 Creating element nodes:

An element node is created using <element> tag. The content of created element is whatever is generated between the starting and closing of <element> tag. If an element has attributes, they are declared using <attribute> tag described in the next section.

```
<xsl:element name="msg">
  Hello world!
</xsl:element>
```

7.2 Create attribute node:

An attributes of an element is created using enclosed <attribute> tag. The mandatory attribute *name* specifies name of the generated attributes. The value is indicated by contnent of <attribute> element.

```
<xsl:element name="msg">
        <xsl:attribute name="lang">en</xsl:attribute>
        Hello world!
</xsl:element>
```

This code segment creates the element *msg* with attribute *lang* as follows:

```
<msg lang="en">Hellow World!</msg>
```

7.3 Create text nodes:

Generally, XSLT processor outputs text that appears in the stylesheet. However, extra white spaces are not provided in such case. Secondaly, special characters such as < and & are represented in text by escape character sequence < and & respectively. For this reason, it provides <text> element to add literal text to result with following syntax:

```
<xsl:text> Hello World &amp;</xsl:text>
```

7.4 Creating document node:

XSLT allows to create new document node using <document> element. For example, following code create temporary document node, which is stored in varaibel named "tempTree".

```
<xsl:variable name="tempTree" as ="document-node()">
  <xsl:document> <xsl:apply-templates/> </xsl:document>
</xsl:variable>
```

7.5 Creating processing instructions:

Processing instruction is added in the result using <processing-instruction> element. The most popular use of this element is to insert the <stylesheet> element in output HTML/XML document with syntax.

```
<xsl:processing-instruction name="xml-stylesheet">
  <xsl:text> href="sort.xsl" type="text/xsl"</xsl:text>
</xsl:processing-instruction>
```

7.5 Creating comments:

Comment is added using <comment> element as follows:

```
<xsl:comment>This is XSLT document</xsl:document>
```

8. Grouping nodes:

XSLT allows us to group related items based on common values. Consider the following XML document.

```
<result>
        <student><rollno>01</rollno><marks>80</marks><dept>IT</dept></student>
        <student><rollno>02</rollno><marks>70</marks><dept>IT</dept></student>
        <student><rollno>03</rollno><marks>60</marks><dept>CSE</dept></student>
        <student><rollno>04</rollno><marks>55</marks><dept>IT</dept></student>
        <student><rollno>05</rollno><marks>77</marks><dept>CSE</dept></student>
</result>
```

The following XSLT document displays results of the students as groups by *dept*:

```
<xsl:template match="/result">
        <xsl:for-each-group select="student" group-by="@dept">
        <xsl:value-of select="current-grouping-key()" />
        <xsl:for-each select="current-group()">
                <xsl:value-of select="@rollno"/>
        </xsl:for-each>
</xsl:template>
```

This enumerates group items based either on common value of grouping key or pattern specified by group-by attribute. The current-group() function returns the current group item in the iteration and current-grouping-key() returns commn key of current group.

9. Sorting nodes:

We can sort group of similar elements using <sort> element. The attributes of the <sort> element describe how to perform sorting. For example, sorting can be doen alphabetically or numerically or in increasing or decreating order. The attribute select is used to specify sorting key. The order attribute specifies order and can have values accending or decending. The type of data to be sorted can be specified using attribute data-type. Following example sorts list of student respect to their marks.

```
<table><xsl:for-each select="/result">
  <xsl:sort select="marks" data-type="number"/>
        <tr><td><xsl:value-of select="rollno"/></td>
        <td><xsl:value-of select="marks"/></td>
        <td><xsl:value-of select="dept"/></td></tr>
</xsl:for-each></table>
```

10. Functions:

XSLT also allows custom functions to be defined in stylesheet. A function is defined using <function> element. It has attribute name, which specifies the name of the function. Once function is defined, it can be called from any expressin. The function name must have prefix. This is required to avoid conflict with any function from default namespace. A prefix can not be bound to reserved namespace.

```
<xsl:function name="f:fact">
        <xsl:param name="n">
```

```
        <xsl:value-of select="if ($n le 1) then 1 else $n*f:fact($n-1)"/>
    </xsl:function>
    <xsl:template match="/">
        <xsl:value-of select="f:fact(3)"/>
    < xsl:template>
```

11. Copying nodes:

The <copy> element copies the current node to the output. If the node is an element node, its namespace nodes are copied automatically, but attributes and children of element nodes are not copied automatically. Consider the simple XML document:

```
<result>
        <student><rollno>01</rollno><marks>80</marks><dept>IT</dept></student>
        <student><rollno>02</rollno><marks>70</marks><dept>IT</dept></student>
        <student><rollno>03</rollno><marks>60</marks><dept>CSE</dept></student>
        <student><rollno>04</rollno><marks>55</marks><dept>IT</dept></student>
        <student><rollno>05</rollno><marks>77</marks><dept>CSE</dept></student>
</result>
```

Now consider following XSLT document:

```
<xsl:template match="/student">
  <xsl:copy />
</xsl:template>
```

12. Numbering:

The <number> element allows to insert and format number into the result tree.

```
<xsl:template match="/result">
        <xsl:for-each-group select="student" group-by="@dept">
        <xsl:number value="position()"/>
        <xsl:value-of select="current-grouping-key()" />
        <xsl:for-each select="current-group()">
                <xsl:number value="position()"/>
                <xsl:value-of select="@rollno"/>
        </xsl:for-each>
</xsl:template>
```

Document Object Model (DOM):

The Document Object Model(DOM) is an application programming interface(API) for HTML and XML documents. It defines the logical structure of documents and the way a document is accesses and manipulated. DOM is a set of platform independent and language neutral application programming interface(API) which describes how to access and manipulate the information stored in XML or in HTML documents. Main objectives of DOM are Accessing the elements of document, deleting the elements of documents and changing the elements of document.

DOM models document as hierarchical structure consisting of different kinds of nodes. Each of these nodes represents specific portion of the document. Some kind of nodes may have children of different types. Some nodes cannot have anything below it in the hierarchical structure and are leaf nodes. With the Document Object Model, programmers can build documents, navigate their structure, and add, modify, or delete elements and content. Anything found in an HTML or XML document can be accessed, changed, deleted, or added using the document object model. The DOM is separated into 3 different parts/levels:

1. Core DOM: standard model for any structured document.
2. HTML DOM:standard model for HTML documents.
3. XML DOM: standard model for XML documents.

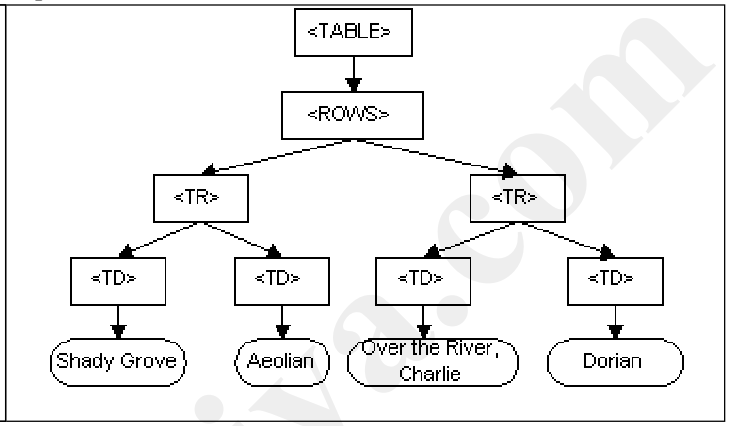
1. Core DOM:

This portion defines the basic set of interfaces and objtects for any structured document.

2. HTML DOM:

The HTML Document Object Model (DOM) is a programming API for HTML documents. It defines the logical structure of documents and the way a document is accessed and manipulated. With the Document Object Model, programmers can create and build documents, navigate their structure, and add, modify, or delete elements and content. Anything found in an HTML document can be accessed, changed, deleted, or added using the Document Object Model, with a few exceptions - in particular, the DOM interfaces for the internal subset and external subset have not yet been specified.

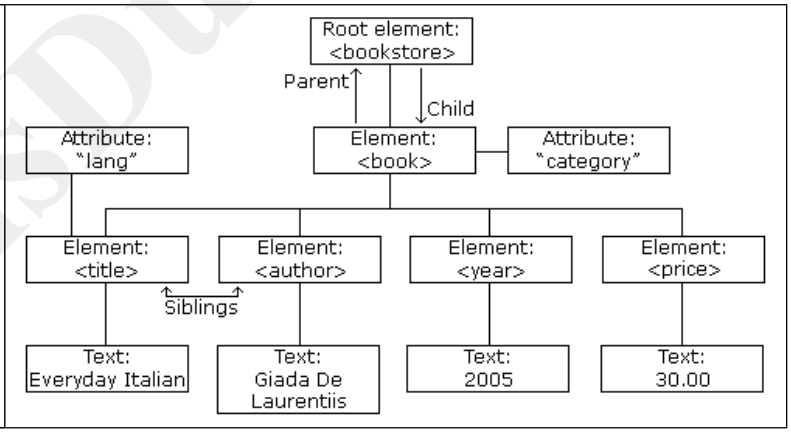
```
<TABLE>
 <ROWS>
 <TR>
    <TD>Shady Grove</TD>
    <TD>Aeolian</TD>
 </TR>
 <TR>
    <TD>Over the River, Charlie</TD>
    <TD>Dorian</TD>
 </TR>
 </ROWS>
 </TABLE>
```



3. XML DOM:

According to the DOM, everything in an XML document is a Node. The DOM says: The entire document is a document node, Every XML element is an element node, The text in the XML elements are text nodes, Every attribute is an attribute node, Comments are comment nodes.

```
<bookstore>
<book category="cooking">
<title lang="en">Everday Italian</title>
<author>Giada De Laurentiis</author
<year>2005</year>
<price>30.00</price>
</book>
</bookstore>
```



The root node in the XML above is named <bookstore>. All other nodes in the document are contained within <bookstore>. The root node <bookstore> holds four <book> nodes. The first <book> node holds four nodes: <title>, <author>, <year>, and <price>, which contains one text node each, "Everyday Italian", "Giada De Laurentiis", "2005", and "30.00". The XML DOM views an XML document as a tree-structure. The tree structure is called a node-tree. All nodes can be accessed through the tree. Their contents can be modified or deleted, and new elements can be created.

The node tree shows the set of nodes, and the connections between them. The tree starts at the root node and branches out to the text nodes at the lowest level of the tree. The nodes in the node tree have a hierarchical relationship to each other. The terms parent, child, and sibling are used to describe the relationships. Parent nodes have children. Children on the same level are called siblings (brothers or sisters). In a node tree, the top node is called the root, Every node except the root has exactly one parent node, A

node can have any number of children, A leaf is a node with no children, Siblings are nodes with the same parent.

Using XML processors:

Parsing XML refers to going through XML document to access data or to modify data in one or other way. XML Parser provides way how to access or modify data present in an XML document. Java provides multiple options to parse XML document. Following are various types of parsers which are commonly used to parse XML documents.

- **Dom Parser**: Parses the document by loading the complete contents of the document and creating its complete hiearchical tree in memory.
- **SAX Parser**: Parses the document on event based triggers. Does not load the complete document into the memory.

Difference between DOM and SAX:

DOM	SAX
DOM is a tree based parsing method	SAX is an event based parsing method
We can insert or delete a node	We can insert or delete a node
Traverse in any direction	Top to bottom traversing
Stores the entire XML document in to memory before processing	Parses node by node
Occupies more memory	Doesn't store the XML in memory
DOM preserves comments	SAX doesn't preserve comments.
import javax.xml.parsers.*; import org.w3c.dom.*;	import javax.xml.sax.*; import org.xml.sax.helpers.*;

1. Java DOM Parser:

The Document Object Model is an official recommendation of the World Wide Web Consortium (W3C). It defines an interface that enables programs to access and update the style, structure,and contents of XML documents. XML parsers that support the DOM implement that interface. In order to use this, we need to know a lot about the structure of a document, need to move parts of the document around and need to use the information in the document more than once. When we parse an XML document with a DOM parser, we get back a tree structure that contains all of the elements of your document. The DOM provides a variety of functions you can use to examine the contents and structure of the document.

DOM interfaces:

The DOM defines several Java interfaces. Here are the most common interfaces:

- **Node**: The base datatype of the DOM.
- **Element**: The vast majority of the objects you'll deal with are Elements.
- **Attr:** Represents an attribute of an element.
- **Text:** The actual content of an Element or Attr.
- **Document:** Represents entire XML document, a Document object is often referred to as a DOM tree.

Common DOM methods:

When you are working with the DOM, there are several methods you'll use often:

- **Document.getDocumentElement()** - Returns the root element of the document.
- **Node.getFirstChild()** - Returns the first child of a given Node.
- **Node.getLastChild()** - Returns the last child of a given Node.
- **Node.getNextSibling()** - These methods return the next sibling of a given Node.
- **Node.getPreviousSibling()** - These methods return the previous sibling of a given Node.
- **Node.getAttribute(attrName)** - For a given Node, returns the attribute with the requested name.

Steps to Use DOM parser:

Following are the steps used while parsing a document using DOM Parser.

1. Import XML-related packages

import org.w3c.dom.;*
import javax.xml.parsers.;*
import java.io.;*

2. Create a DocumentBuilder

DocumentBuilderFactory factory =DocumentBuilderFactory.newInstance();
DocumentBuilder builder = factory.newDocumentBuilder();

3. Create a Document from a file or stream

StringBuilder strb = new StringBuilder();
strb.append("<?xml version="1.0"?> <bookstore> </bookstore>");
ByteArrayInputStream input = new ByteArrayInputStream(strb.toString().getBytes("UTF-8"));
Document doc = builder.parse(input);

4. Extract the root element

Element root = document.getDocumentElement();

5. Examine attributes

getAttribute("attributeName");
getAttributes();

6. Examine sub-elements

getElementsByTagName("subelementName");
getChildNodes();

Example for using of DOM parser:

class.xml

```
<?xml version="1.0"?>
<class>
        <student rollno="393">
                <firstname>dinkar</firstname>
                <lastname>kad</lastname>
                <nickname>dinkar</nickname>
                <marks>85</marks>
        </student>
        <student rollno="493">
                <firstname>Vaneet</firstname>
                <lastname>Gupta</lastname>
                <nickname>vinni</nickname>
                <marks>95</marks>
         </student>
</class>
```

DomParserDemo.java

```
import java.io.File;
import javax.xml.parsers.DocumentBuilderFactory;
import javax.xml.parsers.DocumentBuilder;
import org.w3c.dom.Document;
import org.w3c.dom.NodeList;
import org.w3c.dom.Node;
import org.w3c.dom.Element;
public class DomParserDemo
{
  public static void main(String[] args){
 try {
      File inputFile = new File("input.txt");
```

```
        DocumentBuilderFactory dbFactory = DocumentBuilderFactory.newInstance();
        DocumentBuilder dBuilder = dbFactory.newDocumentBuilder();
        Document doc = dBuilder.parse(inputFile);
        doc.getDocumentElement().normalize();
        System.out.println("Root element :" + doc.getDocumentElement().getNodeName());
        NodeList nList = doc.getElementsByTagName("student");
        System.out.println(" ---------------------------- ");
        for (int temp = 0; temp < nList.getLength(); temp++)
            {           Node nNode = nList.item(temp);
                        System.out.println("\nCurrent Element  :"  + nNode.getNodeName());
                        if (nNode.getNodeType() == Node.ELEMENT_NODE)
                       {            Element eElement = (Element) nNode;
                                    System.out.println("Student roll no : " + eElement.getAttribute("rollno"));
System.out.println("First                                   Name                                    :
"+eElement.getElementsByTagName("firstname").item(0).getTextContent());
System.out.println("Last                    Name                    :                   "                   +
eElement.getElementsByTagName("lastname").item(0).getTextContent());
System.out.println("Nick                                    Name                                    :
"+eElement.getElementsByTagName("nickname").item(0).getTextContent());
System.out.println("Marks : " +eElement.getElementsByTagName("marks").item(0).getTextContent());
            }
        }
    } catch (Exception e) {         e.printStackTrace();        }
  }
}
```

2. Java SAX Parser:

SAX (the Simple API for XML) is an event-based parser for xml documents.Unlike a DOM parser, a SAX parser creates no parse tree. SAX is a streaming interface for XML, which means that applications using SAX receive event notifications about the XML document being processed an element, and attribute, at a time in sequential order starting at the top of the document, and ending with the closing of the ROOT element. Reads an XML document from top to bottom, recognizing the tokens that make up a well-formed XML document. Tokens are processed in the same order that they appear in the document. Reports the application program the nature of tokens that the parser has encountered as they occur. The application program provides an "event" handler that must be registered with the parser. As the tokens are identified, callback methods in the handler are invoked with the relevant information

ContentHandler Interface

This interface specifies the callback methods that the SAX parser uses to notify an application program of the components of the XML document that it has seen.

- **void startDocument()** - Called at the beginning of a document.
- **void endDocument()** - Called at the end of a document.
- **void startElement(String uri, String localName, String qName, Attributes atts)** - Called at the beginning of an element.
- **void endElement(String uri, String localName,String qName)** - Called at the end of an element.
- **void characters(char[] ch, int start, int length)** - Called when character data is encountered.
- **void ignorableWhitespace(char[] ch, int start, int length)** - Called when a DTD is present and ignorable whitespace is encountered.
- **void processingInstruction(String target, String data)** - Called when a processing instruction is recognized.

- **void setDocumentLocator(Locator locator))** - Provides a Locator that can be used to identify positions in the document.
- **void skippedEntity(String name)** - Called when an unresolved entity is encountered.
- **void startPrefixMapping(String prefix, String uri)** - Called when a new namespace mapping is defined.
- **void endPrefixMapping(String prefix)** - Called when a namespace definition ends its scope.

Attributes Interface

This interface specifies methods for processing the attributes connected to an element.

- **int getLength()** - Returns number of attributes.
- **String getQName(int index)**
- **String getValue(int index)**
- **String getValue(String qname)**

Example for using of SAX parser:

class.xml

```
<?xml version="1.0"?>
<class>
        <student rollno="393">
                <firstname>dinkar</firstname>
                <lastname>kad</lastname>
                <nickname>dinkar</nickname>
                <marks>85</marks>
        </student>
        <student rollno="493">
                <firstname>Vaneet</firstname>
                <lastname>Gupta</lastname>
                <nickname>vinni</nickname>
                <marks>95</marks>
         </student>
</class>
```

SAXParserDemo.java

```
import java.io.File;
import javax.xml.parsers.SAXParser;
import javax.xml.parsers.SAXParserFactory;
import org.xml.sax.Attributes;
import org.xml.sax.SAXException;
import org.xml.sax.helpers.DefaultHandler;

public class SAXParserDemo {
  public static void main(String[] args){
    try {
        File inputFile = new File("input.txt");
        SAXParserFactory factory = SAXParserFactory.newInstance();
        SAXParser saxParser = factory.newSAXParser();
        UserHandler userhandler = new UserHandler();
        saxParser.parse(inputFile, userhandler);
    } catch (Exception e) {      e.printStackTrace();      }
  }
}
class UserHandler extends DefaultHandler {
  boolean bFirstName = false;
```

```
boolean bLastName = false;
boolean bNickName = false;
boolean bMarks = false;
public void startElement(String uri, String localName, String qName, Attributes attributes)
    throws SAXException {
        if (qName.equalsIgnoreCase("student"))
        {           String rollNo = attributes.getValue("rollno");
                    System.out.println("Roll No : " + rollNo);
        }
        else if (qName.equalsIgnoreCase("firstname"))
        {       bFirstName = true;      }
        else if (qName.equalsIgnoreCase("lastname"))
        {       bLastName = true;       }
        else if (qName.equalsIgnoreCase("nickname"))
        {       bNickName = true;       }
         else if (qName.equalsIgnoreCase("marks"))
        {       bMarks = true;          }
}
public void endElement(String uri,  String localName, String qName) throws SAXException {
   if (qName.equalsIgnoreCase("student"))
        {       System.out.println("End  Element :" + qName);       }
}
public void characters(char ch[],   int start, int length) throws SAXException {
   if (bFirstName) {
      System.out.println("First Name: " + new String(ch, start, length));
      bFirstName = false;
   } else if (bLastName) {
      System.out.println("Last Name: " + new String(ch, start, length));
      bLastName = false;
   } else if (bNickName) {
      System.out.println("Nick Name: " + new String(ch, start, length));
      bNickName = false;
   } else if (bMarks) {
      System.out.println("Marks: "     + new String(ch, start, length));
      bMarks = false;
   }
  }
}
```

AJAX (Asynchronous JavaScript and XML)

AJAX is an acronym for **Asynchronous JavaScript and XML**. It is a group of inter-related technologies like javascript, dom, xml, html, css etc. AJAX allows you to send and receive data asynchronously without reloading the entire web page. So it is fast.

AJAX allows you to send only important information to the server not the entire page. So only valuable data from the client side is routed to the server side. It makes your application interactive and faster.

Where it is used?

There are too many web applications running on the web that are using AJAX Technology. Some are:

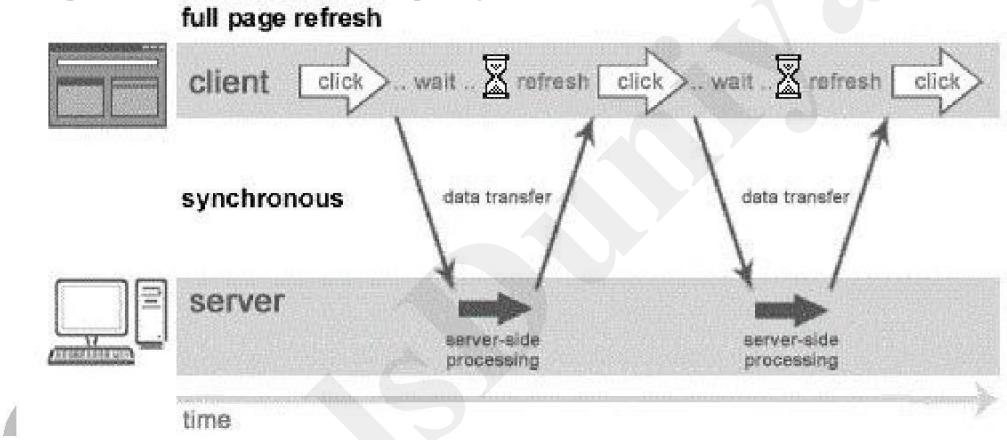
1. Gmail
2. Facebook
3. Twitter
4. Google maps
5. YouTube etc.,

Synchronous Vs. Asynchronous Application

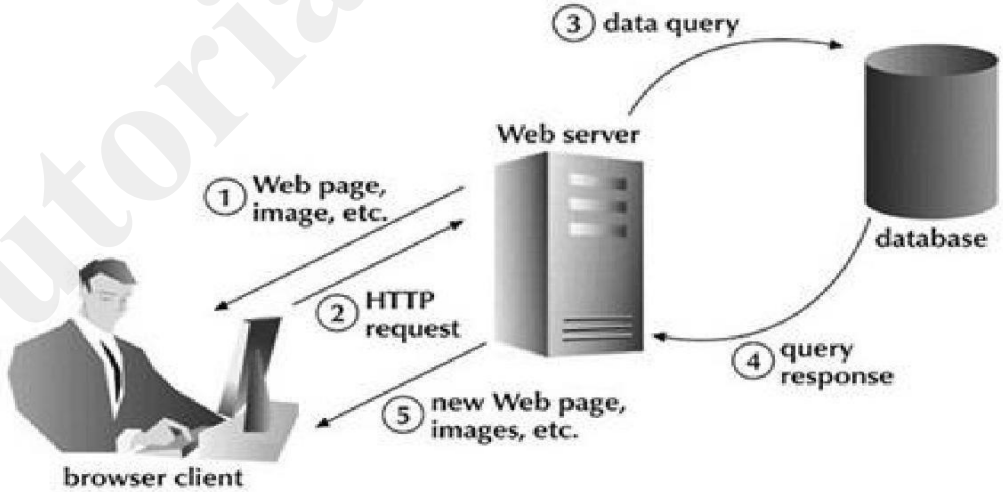
Before understanding AJAX, let's understand classic web application model and AJAX Web application model.

- **Synchronous (Classic Web-Application Model)**

A synchronous request blocks the client until operation completes i.e. browser is not unresponsive. In such case, JavaScript Engine of the browser is blocked.



As you can see in the above image, full page is refreshed at request time and user is blocked until request completes. Let's understand it another way.



- **Asynchronous (AJAX Web-Application Model)**

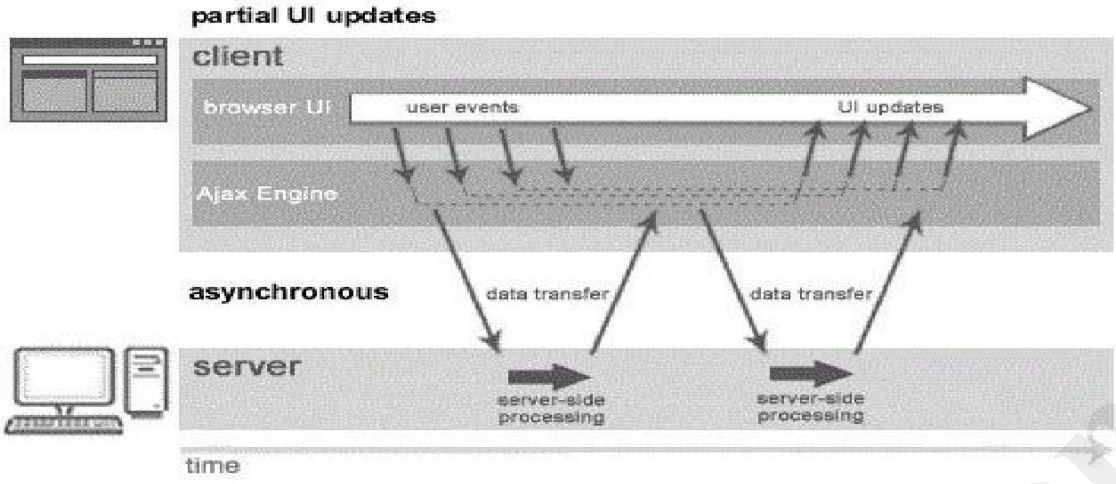
An asynchronous request doesn't block the client i.e. browser is responsive. At that time, user can perform other operations also. In such case, JavaScript Engine of the browser is not blocked.

TutorialsDuniya.com

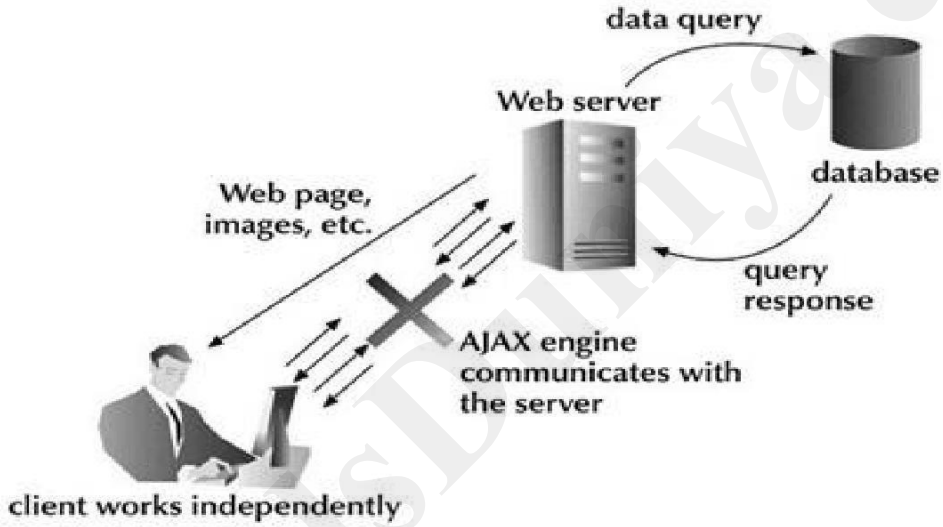
Download FREE Computer Science Notes, Programs, Projects, Books PDF for any university student of BCA, MCA, B.Sc, B.Tech CSE, M.Sc, M.Tech at https://www.tutorialsduniya.com

- Algorithms Notes
- Artificial Intelligence
- Android Programming
- C & C++ Programming
- Combinatorial Optimization
- Computer Graphics
- Computer Networks
- Computer System Architecture
- DBMS & SQL Notes
- Data Analysis & Visualization
- Data Mining
- Data Science
- Data Structures
- Deep Learning
- Digital Image Processing
- Discrete Mathematics
- Information Security
- Internet Technologies
- Java Programming
- JavaScript & jQuery
- Machine Learning
- Microprocessor
- Operating System
- Operational Research
- PHP Notes
- Python Programming
- R Programming
- Software Engineering
- System Programming
- Theory of Computation
- Unix Network Programming
- Web Design & Development

Please Share these Notes with your Friends as well



As you can see in the above image, full page is not refreshed at request time and user gets response from the AJAX Engine. Let's try to understand asynchronous communication by the image given below.



AJAX Technologies

AJAX is not a Technology but group of inter-related technologies. AJAX Technologies includes:

- HTML/XHTML and CSS
- DOM
- XML or JSON(JavaScript Object Notation)
- XMLHttpRequest
- JavaScript

- **HTML/XHTML and CSS**
  These technologies are used for displaying content and style. It is mainly used for presentation.

- **DOM**
  It is used for dynamic display and interaction with data.

- **XML or JSON**
  For carrying data to and from server. JSON is like XML but short and faster than XML.

- XMLHttpRequest
  For asynchronous communication between client and server.

- JavaScript
  It is used to bring above technologies together. Independently, it is used mainly for client-side validation.

Understanding XMLHttpRequest

An object of XMLHttpRequest is used for asynchronous communication between client and server. It performs following operations:

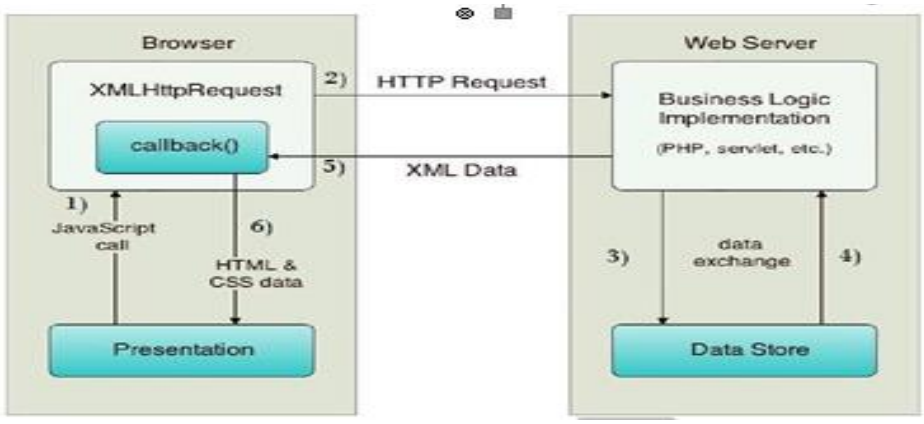
1. Sends data from the client in the background
2. Receives the data from the server
3. Updates the webpage without reloading it.

- **Properties of XMLHttpRequest object:**

Property	Description
onReadyStateChange	It is called whenever readystate attribute changes. It must not be used with synchronous requests.
readyState	Represents the state of the request. It ranges from 0 to 4. **0** UNOPENED open() is not called. **1** OPENED open is called but send() is not called. **2** HEADERS_RECEIVED send() is called, and headers and status are available. **3** LOADING Downloading data; responseText holds the data. **4** DONE The operation is completed fully.
reponseText	Returns response as TEXT.
responseXML	Returns response as XML

Method	Description
void open(method, URL)	Opens the request specifying get or post method and url.
void open(method, URL, async)	Same as above but specifies asynchronous or not.
void open(method, URL, async, username, password)	Same as above but specifies username and password.
void send()	Sends GET request.
void send(string)	Sends POST request.
setRequestHeader(header,value)	It adds request headers.

1. User sends a request from the UI and a javascript call goes to XMLHttpRequest object.
2. HTTP Request is sent to the server by XMLHttpRequest object.
3. Server interacts with the database using JSP, PHP, Servlet, ASP.net etc.
4. Data is retrieved.
5. Server sends XML data or JSON data to the XMLHttpRequest callback function.
6. HTML and CSS data is displayed on the browser.



Integrating PHP and AJAX:-

The following example will demonstrate how a web page can communicate with a web server while a user type characters in an input field:

Example

Start typing a name in the input field below:

First name:

Suggestions:

Example Explained

In the example above, when a user types a character in the input field, a function called "showHint()" is executed.

The function is triggered by the onkeyup event.

Here is the HTML code:

Example

```
<html>
<head>
<script>
function showHint(str) {
   if (str.length == 0) {
      document.getElementById("txtHint").innerHTML = "";
      return;
   } else {
```

```
        var xmlhttp = new XMLHttpRequest();
        xmlhttp.onreadystatechange = function() {
            if (this.readyState == 4 && this.status == 200) {
                document.getElementById("txtHint").innerHTML = this.responseText;
            }
        };
        xmlhttp.open("GET", "gethint.php?q=" + str, true);
        xmlhttp.send();
    }
}
</script>
</head>
<body>

<p><b>Start typing a name in the input field below:</b></p>
<form>
First name: <input type="text" onkeyup="showHint(this.value)">
</form>
<p>Suggestions: <span id="txtHint"></span></p>
</body>
</html>
```

Code explanation:

First, check if the input field is empty (str.length == 0). If it is, clear the content of the txtHint placeholder and exit the function.

However, if the input field is not empty, do the following:

- Create an XMLHttpRequest object
- Create the function to be executed when the server response is ready
- Send the request off to a PHP file (gethint.php) on the server
- Notice that q parameter is added to the url (gethint.php?q="+str)
- And the str variable holds the content of the input field

The PHP File - "gethint.php"

The PHP file checks an array of names, and returns the corresponding name(s) to the browser:

```
<?php
// Array with names
$a[] = "Anna";
$a[] = "Brittany";
$a[] = "Cinderella";
$a[] = "Diana";
$a[] = "Eva";
$a[] = "Fiona";
$a[] = "Gunda";
$a[] = "Hege";
$a[] = "Inga";
$a[] = "Johanna";
```

```
$a[] = "Kitty";
$a[] = "Linda";
$a[] = "Nina";
$a[] = "Ophelia";
$a[] = "Petunia";
$a[] = "Amanda";
$a[] = "Raquel";
$a[] = "Cindy";
$a[] = "Doris";
$a[] = "Eve";
$a[] = "Evita";
$a[] = "Sunniva";
$a[] = "Tove";
$a[] = "Unni";
$a[] = "Violet";
$a[] = "Liza";
$a[] = "Elizabeth";
$a[] = "Ellen";
$a[] = "Wenche";
$a[] = "Vicky";

// get the q parameter from URL
$q = $_REQUEST["q"];

$hint = "";

// lookup all hints from array if $q is different from ""
if ($q !== "") {
    $q = strtolower($q);
    $len=strlen($q);
    foreach($a as $name) {
        if (stristr($q, substr($name, 0, $len))) {
            if ($hint === "") {
                $hint = $name;
            } else {
                $hint .= ", $name";
            }
        }
    }
}

// Output "no suggestion" if no hint was found or output correct values
echo $hint === "" ? "no suggestion" : $hint;
?>
```

Introduction to Web Services

Technology keep on changing, users were forces to learn new application on continuous basis. With internet, focus is shifting to-wards services based software. Users may access these services using wide range of devices such as PDAs, mobile phones, desktop computers etc. Service oriented software development is possible using man known techniques such as COM, CORBA, RMI, JINI, RPC etc. some of them are capable of delivering services over web & some or not. Most of these technologies uses particular protocols for communication & with no standardization. **Web service** is the concept of creating services that can be accessed over web. Most of these

What are Web Services?

A web services may be defines as: An application component accessible via standard web protocols. It is like unit of application logic. It provides services & data to remote clients & other applications. Remote clients & application access web services with internet protocols. They use XML for data transport & SOAP for using services. Accessing service is independent of implementation. With component development model, web service must have following characteristics:

- Registration with lookup service
- Public interface for client to invoke service
- It should use standard web protocols for communication
- It should be accessible over web
- It should support loose coupling between uncoupled distributed systems

Web services receive information from clients as messages, containing instructions about what client wants, similar to method calls with parameters. These message delivered by web services are encoded using XML.XML enabled web services are interoperable with other web services.

Web Service Technologies:

Wide variety of technologies supports web services. Following technologies are available for creation of web services. These are vendor neutral technologies. They are:

- Simple Object Access Protocol(SOAP)
- Web Services Description Language(WSDL)
- UDDI(Universal Description Discovery and Integration)

Simple Object Access Protocol (SOAP):

SOAP is a light weight & simple XML based protocol. It enables exchange of structured & typed information on web by describing messaging format for machine to machine communication. It also enables creation of web services based on open infrastructure. SOAP consists of three parts:

- **SOAP Envelope**: defines what is in message, who is the recipient, whether message is optional or mandatory

- **SOAP Encoding Rules**: defines set of rules for exchanging instances of application defined data types
- **SOAP RPC Representation**: defines convention for representing remote procedure calls & response

SOAP can be used in combination with variety of existing internet protocols & formats including HTTP, SMTP etc. Typical SOAP message is shown below:

```
<IVORY:Envelope xmlns:IVORY="http://schemas.xmlsoap.org/soap/envelope"
                IVORY:encodingStyle="http://schemas.xmlsoap.org/soap/encoding">
<IVORY:Body>
    <m:GetLastTradePrice xmlns:m="Some-URI">
    <symbol>DIS</symbol>
    </m:GetLastTradePrice>
</IVORY:Body>
</IVORY:Envelope>
```

The consumer of web service creates SOAP message as above, embeds it in HTTP POST request & sends it to web service for processing:

```
POST /StockQuote HTTP/1.1
Host: www.stockquoteserver.com
Content-Type: text/xml;
charset="utf-8"
Content-Length: nnnn
SOAPAction: "Some-URI"
    ....
SOAP Message
....
```

The message now contains requested stock price. A typical returned SOAP message may look like following:

```
<SOAP-ENV:Envelope xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope"
SOAP-ENV:encodingStyle=" http://schemas.xmlsoap.org/soap/encoding" />
    <SOAP-ENV:Body>
        <m:GetLastTradePrice xmlns:m="Some-URI">
            <Price>34.5</Price>
        </m:GetLastTradePrice>
    </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

Interoperability:

The major goal in design of SOAP was to allow for easy creation of interoperable distributed web services. Few details of SOAP specifications are open for interpretation; implementation may differ across different vendors. SOAP message though it is conformant XML message, may not strictly follow SOAP specification.

Implementations:

SOAP technology was developed by DevelopMentor, IBM, Lotus, Microsoft etc. More than 50 vendors have currently implemented SOAP. Most popular implementations are by Apache which is open source java based implementation & by Microsoft in .NET platform. SOAP specification has been submitted to W3C, which is now working on new specifications called XMLP (XML Protocol)

SOAP Messages with Attachments (SwA)

SOAP can send message with an attachment containing of another document or image etc. On Internet, GIF, JPEG data formats are treated as standards for image transmission. Second iteration of SOAP specification allowed for attachments to be combined with SOAP message by using multipart

MIME structure. This multi part structure is called as **SOAP Message Package**. This new specification was developed by HP & Microsoft. Sample SOAP message attachment is shown here:

```
MIME-Version: 1.0
Content-Type: Multipart/Related; boundary=MIME_boundary;
type=text/xml; start=”<myimagedoc.xml@mystie.com>”
Content-Description: This is the optional message description.
--MIME_boundary
Content-Type: text/xml; charset=UTF-8
Content-Transfer-Encoding: 8bit
Content-ID: <myimagedoc.xml@mysite.com>
<?xml version= "1.0"?>
<SOAP-ENV: Envelope xmlns:SOAP-ENV=”http://schemas.xmlsoap.org/soap/envelope”
<SOAP-ENV:Body>
        ...
        <theSignedForm href=”cid:myimage.tiff@mysite.com” />
        ...
</SOAP-ENV:Body>
</SOAP-ENV:Envelope>
        --MIME_boundary
        Content-Type: image/tiff
        Content-Transfer-Encoding: binary
        Content-ID: <myimagedoc.xml@mysite.com>
        ...binary TIFF image…
        --MIME_boundary--
```

Web Services Description Language (WSDL)

WSDL is an XML format for describing web service interface. WSDL file defines set of operations permitted on the server & format that client must follow while requesting service. WSDL file acts like contract between client & service for effective communication between two parties. Client has to request service by sending well formed & conformant SOAP request.

If we are creating web service that offered latest stock quotes, we need to create WSDL file on server that describes service. Client obtains copy of this file, understand contract, create SOAP request based on contract & dispatch request to server using HTTP post. Server validates the request, if found valid executes request. The result which is latest stock price for requested symbol is then returned to client as SOAP response.

WSDL Document:

WSDL document is an XML document that contains of set of definitions. First we declare name spaces required by schema definition:

```
<schema xmlns=”http://www.w3.org/2000/10/XMLSchema”
xmlns:wsdl=”http://schemas.xmlsoap.org/wsdl/”
targetNameSpace=http://schemas.xmlsoap.org/wsdl/ elementFormDefault=”qualified”>
```

The root element is definitions as shown below:

```
<wsdl:definitions name=”nmtoken”? targetNameSpace=”uri”?>
        <import namespace=”uri” location=”uri”/>
<wsdl:documentation ..... />?
        ...
</wsdl:definitions>
```

The *name* attribute is optional & can serve as light weight form of documentation. The *nmtoken* represents name token that are qualified strings similar to CDATA, but character usage is limited to letters, digits, underscores, colons, periods & dashes. A *targetNamespace* may be specified by providing uri. The *import* tag may be used to associate namespace with document locations. Following code segment shows how declared namespace is associated with document location specified in *import* statement:

```
<definitions                                    name=”StockQuote”
        targetNameSpace=”http://example.com/stockquote/definitions”
```

```
        xmlns:tns="http://example.com/stockquote/definitions"
        xmlns:xsdl="http://example.com/stockquote/schemas"
        xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
        xmlns="http://schemas.xmlsoap.org/wsdl/">
<import namespace="http://example.com/stockquote/schemas"
        Location="http://example.com/stockquote/stockquote.xsd"/>
```

Finally, optional *wsdl:documentation* element is used for declaring human readable documentation. The element may contain any arbitrary text. There are six major elements in document structure that describes service. These are as follows:

- **Types Element:** it provides definitions for data types used to describe how messages will exchange data. Syntax for types element is as follows:

```
<wsdl:types> ?
        <wsdl:documentation
        .../> <xsd:schema .../>
        <-- extensibility element -->
</wsdl:types>
```

The *wsdl:documentation* tag is optional as in case of *definitions*. The *xsd* type system may be used to define types in message. WSDL allows type systems to be added via extensibility element.

- **Message Element:** It represents abstract definition of data begin transmitted. Syntax for message element:

```
<wsdl:message name="nktoken"> *
        <wsdl;documentation .../>
            <part name="nmtoken" element="qname"? type="qname"? /> *
</wsdl:message>
```

The *message name* attribute is used for defining unique name for message with in document scope. The *wsdl:documentation* is optional & may be used for declaring human readable documentation. The message consists of one or more logical parts. The *part* describes logical abstract content of message. Each part consists of name & optional element & type attributes.\

- **Port Type Element:** It defines set of abstract operations. An operation consists of both input & output messages. The *operation* tag defines name of operation, *input* defines input for operation & *output* defines output format for result. The *fault* element is used for describing contents of SOAP fault details element. It specifies abstract message format for error messages that may be output as result of operation:

```
<wsdl:portType name="nmtoken">*
        <wsdl:documentation ..../>?
    <wsdl:operation name="nmtoken">*
        <wsdl:documentation ..../>?
            <wsdl:input name="nmtoken"? message="qname">?
        <wsdl:documentation ..../>?
        </wsdl:input>
            <wsdl:output name="nmtoken"? message="qname">?
        <wsdl:documentation ..../>?
        </wsdl:output>
            <wsdl:fault name="nmtoken"? message="qname">?
            <wsdl:documentation ..../>?
        </wsdl:fault>
    </wsdl:operation>
</wsdl:portType>
```

- **Binding Element:** It defines protocol to be used & specifies data format for operations & messages defined by particular *portType*. The full syntax for binding is given below:

```
<wsdl:binding name="nmtoken" type="qname"> *
        <wsdl:documentation ..../>?
        <--Extensibility element -->*
    <wsdl:operation name="nmtoken">*
        <wsdl:documentation ..../>?
        <--Extensibility element -->*
    <wsdl:input> ?
        <wsdl:documentation ..../>?
        <--Extensibility element -->*
    </wsdl:input>
    <wsdl:output> ?
        <wsdl:documentation ..../>?
        <--Extensibility element -->*
    </wsdl:output>
    <wsdl:fault name="nmtoken"> *
        <wsdl:documentation ..../>?
        <--Extensibility element -->*
    </wsdl:fault>
    </wsdl:operation>
</wsdl:binding>
```

The operation in WSDL file can be document oriented or remote procedure call (RPC) oriented. The style attribute of <*soap:binding*> element defines type of operation. If operation is document oriented, input & output messages will consist of XML documents. If operation is RPC oriented, input message contains operations input parameters & output message contains result of operation.

❖ **Port Element:** It defines individual end point by specifying single address for binding:

```
<wsdl:port name="nmtoken" binding="qname"> *
    <--Extensibility element (1) -->
</wsdl:port>
```

The *name* attribute defines unique name for port with current WSDL document. The *binding* attribute refers to binding & extensibility element is used to specify address information for port.

❖ **Service Element:** it aggregates set of related ports. Each port specifies address for binding:

```
<wsdl:service name="nmtoken"> *
        <wsdl:documentation ..../>?
    <wsdl:port name="nktoken" binding="qname"> *
        <wsdl:documentation .../> ?
        <--Extensibility element -->
    </wsdl:port>
        <--Extensibility element -->
</wsdl:service>
```

Universal Description, Discovery & Integration (UDDI)

We need to publish web services so that customers & business partners can use the services. It requires common registry to register web service for clients to find it. For this several vendors including IBM, HP, Oracle, Sun Microsystem etc. formed an industry consortium known as UDDI. Today more than 250 companies have joined UDDI project. The main task of this project is to develop specifications for web based business registry. The registry should be able to describe web service & allow others to discover registered web services.

UDDI allows any organization to publish information about its web services. The framework defines standard for businesses to share information, describe their services & their business & to decide what information is made public & what information is kept private. The interface is based on XML & SOAP, uses HTTP to interact with registry.

Registry itself holds information about business such as company name, contact etc. it holds both descriptive & technical information about web service. It provides search facilities that allow to search specific industry segment or geographic location.

Implementation:

This is global, public registry called UDDI business registry. It is possible for individuals to set up private UDDI registries. The implementations for creating private registries are available from IBM, Idoox etc. Microsoft has developed UDDI SDK that allows visual basic programmer to write program code to interact with UDDI registry. The use of SDK greatly simplifies interaction with registry & shields programmer from local level details of XML & SOAP.

Electronic Business XML (ebXML):

ebXML is set of specifications that allows businesses to collaborate. It enables global electronic market place where business can meet & tranasact with help of XML based messages. Business may be geographically located anywhere in world & could be of any size to participate in global marketplace. The framework defines specifications for sharing of web based business services. It includes specifications for message service, collaborative partner agreements, core components, business process methodology, registry & repository.

It defines registry & repository where business can register themselves by providing their contact information, address & so on. Such information is called Core Component. After business has registered with ebXML registry, other partners can look up registry to locate that business. Once business partner is located, the core components of located business are downloaded. Once buyer is satisfied with fact that seller service can meet its requirements, it negotiates contract with seller. Such collaborative partner agreements are defined in ebXML. Once both parties agree on contract terms, sign agreements & collaborative business transaction by exchanging their private documents. ebXML provides marketplace & defines several XML based documents for business to join & transact in such marketplace.

Web Technologies

UNIT-IV

What You Should Already Know

Before you continue you should have a basic understanding of the following:

- HTML
- JavaScript

What is PHP?

- PHP stands for **P**HP: **H**ypertext **P**reprocessor
- PHP is a widely-used, open source scripting language
- PHP scripts are executed on the server
- PHP is free to download and use

What is a PHP File?

- PHP files can contain text, HTML, JavaScript code, and PHP code
- PHP code are executed on the server, and the result is returned to the browser as plain HTML
- PHP files have a default file extension of ".php"

What Can PHP Do?

- PHP can generate dynamic page content
- PHP can create, open, read, write, and close files on the server
- PHP can collect form data
- PHP can send and receive cookies
- PHP can add, delete, modify data in your database
- PHP can restrict users to access some pages on your website
- PHP can encrypt data

With PHP you are not limited to output HTML. You can output images, PDF files, and even Flash movies. You can also output any text, such as XHTML and XML.

Why PHP?

- PHP runs on different platforms (Windows, Linux, Unix, Mac OS X, etc.)
- PHP is compatible with almost all servers used today (Apache, IIS, etc.)
- PHP has support for a wide range of databases
- PHP is free. Download it from the official PHP resource: www.php.net
- PHP is easy to learn and runs efficiently on the server side

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

The PHP script is executed on the server, and the plain HTML result is sent back to the browser.

Basic PHP Syntax

A PHP script always starts with **<?php** and ends with **?>**. A PHP script can be placed anywhere in the document.

On servers with shorthand-support, you can start a PHP script with <? and end with ?>.

For maximum compatibility, we recommend that you use the standard form (<?php) rather than the shorthand form.

```
<?php
// PHP code goes here
?>
```

The default file extension for PHP files is ".php".

A PHP file normally contains HTML tags, and some PHP scripting code.

Below, we have an example of a simple PHP script that sends the text "Hello World!" back to the browser:

Example

```
<!DOCTYPE html>
<html>
<body>

<?php
echo "Hello World!";
?>

</body>
</html>
```

Show example »

Each code line in PHP must end with a semicolon. The semicolon is a separator and is used to distinguish one set of instructions from another.

There are two basic statements to output text with PHP: **echo** and **print**.

In the example above we have used the echo statement to output the text "Hello World".

Comments in PHP

In PHP, we use // to make a one-line comment, or **/*** and ***/** to make a comment block:

Example

```
<!DOCTYPE html>
<html>
<body>

<?php
//This is a comment

/*
This is
a comment
block
*/
?>
```

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

Variables are "containers" for storing information.

Do You Remember Algebra From School?

Do you remember algebra from school? x=5, y=6, z=x+y

Do you remember that a letter (like x) could be used to hold a value (like 5), and that you could use the information above to calculate the value of z to be 11?

These letters are called **variables**, and variables can be used to hold values (x=5) or expressions (z=x+y).

PHP Variables

As with algebra, PHP variables are used to hold values or expressions.

A variable can have a short name, like x, or a more descriptive name, like carName.

Rules for PHP variable names:

- Variables in PHP starts with a $ sign, followed by the name of the variable
- The variable name must begin with a letter or the underscore character
- A variable name can only contain alpha-numeric characters and underscores (A-z, 0-9, and _)
- A variable name should not contain spaces
- Variable names are case sensitive (y and Y are two different variables)

Creating (Declaring) PHP Variables

PHP has no command for declaring a variable.

A variable is created the moment you first assign a value to it:

```
$myCar="Volvo";
```

After the execution of the statement above, the variable **myCar** will hold the value **Volvo**.

Tip: If you want to create a variable without assigning it a value, then you assign it the value of *null*.

Let's create a variable containing a string, and a variable containing a number:

```
<?php
$txt="Hello World!";
$x=16;
?>
```

Note: When you assign a text value to a variable, put quotes around the value.

PHP is a Loosely Typed Language

In PHP, a variable does not need to be declared before adding a value to it.

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 a strongly typed programming language, you have to declare (define) the type and name of the variable before using it.

PHP Variable Scope

The scope of a variable is the portion of the script in which the variable can be referenced.

PHP has four different variable scopes:

- local
- global
- static
- parameter

Local Scope

A variable declared **within** a PHP function is local and can only be accessed within that function. (the variable has local scope):

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

function myTest()
{
echo $a; // local scope
}

myTest();
?>
```

The script above will not produce any output because the echo statement refers to the local scope variable $a, which has not been assigned a value within this scope.

You can have local variables with the same name in different functions, because local variables are only recognized by the function in which they are declared.

Local variables are deleted as soon as the function is completed.

Global Scope

Global scope refers to any variable that is defined outside of any function.
Global variables can be accessed from any part of the script that is not inside a function.
To access a global variable from within a function, use the **global** keyword:

```
<?php
$a = 5;
$b = 10;

function myTest()
{
global $a, $b;
$b = $a + $b;
}

myTest();
echo $b;
?>
```

The script above will output 15.

PHP also stores all global variables in an array called $GLOBALS[*index*]. Its index is the name of the variable. This array is also accessible from within functions and can be used to update global variables directly.

The example above can be rewritten as this:

```
<?php
$a = 5;
$b = 10;

function myTest()
{
$GLOBALS['b'] = $GLOBALS['a'] + $GLOBALS['b'];
}

myTest();
echo $b;
?>
```

Static Scope

When a function is completed, all of its variables are normally deleted. However, sometimes you want a local variable to not be deleted.

To do this, use the **static** keyword when you first declare the variable:

static $rememberMe;

Then, each time the function is called, that variable will still have the information it contained from the last time the function was called.

Note: The variable is still local to the function.

Parameters

A parameter is a local variable whose value is passed to the function by the calling code.

Parameters are declared in a parameter list as part of the function declaration:

```
function myTest($para1,$para2,...)
{
// function code
}
```

Parameters are also called arguments. We will discuss them in more detail when we talk about functions.

A string variable is used to store and manipulate text.

String Variables in PHP

String variables are used for values that contain characters.

In this chapter we are going to look at the most common functions and operators used to manipulate strings in PHP.

After we create a string we can manipulate it. A string can be used directly in a function or it can be stored in a variable.

Below, the PHP script assigns the text "Hello World" to a string variable called $txt:

```
<?php
$txt="Hello World";
echo $txt;
?>
```

The output of the code above will be:

Hello World

Now, lets try to use some different functions and operators to manipulate the string.

The Concatenation Operator

There is only one string operator in PHP.

The concatenation operator (.) is used to put two string values together.

To concatenate two string variables together, use the concatenation operator:

```
<?php
$txt1="Hello World!";
$txt2="What a nice day!";
echo $txt1 . " " . $txt2;
?>
```

The output of the code above will be:

Hello World! What a nice day!

If we look at the code above you see that we used the concatenation operator two times. This is because we had to insert a third string (a space character), to separate the two strings.

The strlen() function

The strlen() function is used to return the length of a string.

Let's find the length of a string:

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

The output of the code above will be:

12

The length of a string is often used in loops or other functions, when it is important to know when the string ends. (i.e. in a loop, we would want to stop the loop after the last character in the string).

TutorialsDuniya.com

Download FREE Computer Science Notes, Programs, Projects, Books PDF for any university student of BCA, MCA, B.Sc, B.Tech CSE, M.Sc, M.Tech at https://www.tutorialsduniya.com

- Algorithms Notes
- Artificial Intelligence
- Android Programming
- C & C++ Programming
- Combinatorial Optimization
- Computer Graphics
- Computer Networks
- Computer System Architecture
- DBMS & SQL Notes
- Data Analysis & Visualization
- Data Mining
- Data Science
- Data Structures
- Deep Learning
- Digital Image Processing
- Discrete Mathematics
- Information Security
- Internet Technologies
- Java Programming
- JavaScript & jQuery
- Machine Learning
- Microprocessor
- Operating System
- Operational Research
- PHP Notes
- Python Programming
- R Programming
- Software Engineering
- System Programming
- Theory of Computation
- Unix Network Programming
- Web Design & Development

Please Share these Notes with your Friends as well

The strpos() function

The strpos() function is used to search for a character/text within a string.

If a match is found, this function will return the character 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
echo strpos("Hello world!","world");
?>
```

The output of the code above will be:

6

The position of the string "world" in the example above is 6. The reason that it is 6 (and not 7), is that the first character position in the string is 0, and not 1.

The assignment operator = is used to assign values to variables in PHP.

The arithmetic operator + is used to add values together.

Arithmetic Operators

The table below lists the arithmetic operators in PHP:

Operator	Name	Description	Example	Result
x + y	Addition	Sum of x and y	2 + 2	4
x - y	Subtraction	Difference of x and y	5 - 2	3
x * y	Multiplication	Product of x and y	5 * 2	10
x / y	Division	Quotient of x and y	15 / 5	3
x % y	Modulus	Remainder of x divided by y	5 % 2 10 % 8 10 % 2	1 2 0
- x	Negation	Opposite of x	- 2	
a . b	Concatenation	Concatenate two strings	"Hi" . "Ha"	HiHa

Assignment Operators

The basic assignment operator in PHP is "=". It means that the left operand gets set to the value of the expression on the right. That is, the value of "$x = 5" is 5.

Assignment	Same as...	Description
x = y	x = y	The left operand gets set to the value of the expression on the right
x += y	x = x + y	Addition
x -= y	x = x - y	Subtraction
x *= y	x = x * y	Multiplication
x /= y	x = x / y	Division
x %= y	x = x % y	Modulus
a .= b	a = a . b	Concatenate two strings

Incrementing/Decrementing Operators

Operator	Name	Description
++ x	Pre-increment	Increments x by one, then returns x
x ++	Post-increment	Returns x, then increments x by one
-- x	Pre-decrement	Decrements x by one, then returns x
x --	Post-decrement	Returns x, then decrements x by one

Comparison Operators

Comparison operators allows you to compare two values:

Operator	Name	Description	Example
x == y	Equal	True if x is equal to y	5==8 returns false
x === y	Identical	True if x is equal to y, and they are of same type	5==="5" returns false
x != y	Not equal	True if x is not equal to y	5!=8 returns true
x <> y	Not equal	True if x is not equal to y	5<>8 returns true
x !== y	Not identical	True if x is not equal to y, or they are not of same type	5!=="5" returns true
x > y	Greater than	True if x is greater than y	5>8 returns false
x < y	Less than	True if x is less than y	5<8 returns true
x >= y	Greater than or equal to	True if x is greater than or equal to y	5>=8 returns false
x <= y	Less than or equal to	True if x is less than or equal to y	5<=8 returns true

Logical Operators

Operator	Name	Description	Example
x and y	And	True if both x and y are true	x=6 y=3 (x < 10 and y > 1) returns

			true
x or y	Or	True if either or both x and y are true	x=6 y=3 (x==6 or y==5) returns true
x xor y	Xor	True if either x or y is true, but not both	x=6 y=3 (x==6 xor y==3) returns false
x && y	And	True if both x and y are true	x=6 y=3 (x < 10 && y > 1) returns true
x || y	Or	True if either or both x and y are true	x=6 y=3 (x==5 || y==5) returns false
! x	Not	True if x is not true	x=6 y=3 !(x==y) returns true

Array Operators

Operator	Name	Description
x + y	Union	Union of x and y
x == y	Equality	True if x and y have the same key/value pairs
x === y	Identity	True if x and y have the same key/value pairs in the same order and of the same types
x != y	Inequality	True if x is not equal to y
x <> y	Inequality	True if x is not equal to y
x !== y	Non-identity	True if x is not identical to y

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

Conditional Statements

Very often when you write code, you want to perform different actions for different decisions.

You can use conditional statements in your code to do this.

In PHP we have the following conditional statements:

- **if statement** - use this statement to execute some code only if a specified condition is true

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

---

The if Statement

Use the if statement to execute some code only if a specified condition is true.

Syntax
if (*condition*) *code to be executed if condition is true;*

The following example will output "Have a nice weekend!" if the current day is Friday:

```
<html>
<body>

<?php
$d=date("D");
if ($d=="Fri") echo "Have a nice weekend!";
?>

</body>
</html>
```

Notice that there is no ..else.. in this syntax. The code is executed **only if the specified condition is true**.

The if...else Statement

Use the if... else statement to execute some code if a condition is true and another code if a condition is false.

Syntax

```
if (condition)
  {
  code to be executed if condition is true;
  }
else
  {
  code to be executed if condition is false;
  }
```

Example

The following example will output "Have a nice weekend!" if the current day is Friday, otherwise it will output "Have a nice day!":

```
<html>
<body>

<?php
$d=date("D");
if ($d=="Fri")
  {
  echo "Have a nice weekend!";
  }
else
  {
  echo "Have a nice day!";
  }
?>

</body>
</html>
```

The if...elseif....else Statement

Use the if....elseif...else statement to select one of several blocks of code to be executed.

Syntax

```
if (condition)
  {
  code to be executed if condition is true;
  }
elseif (condition)
  {
  code to be executed if condition is true;
  }
else
  {
  code to be executed if condition is false;
  }
```

Example

The following example will output "Have a nice weekend!" if the current day is Friday, and "Have a nice Sunday!" if the current day is Sunday. Otherwise it will output "Have a nice day!":

```
<html>
<body>

<?php
$d=date("D");
if ($d=="Fri")
  {
  echo "Have a nice weekend!";
  }
elseif ($d=="Sun")
  {
  echo "Have a nice Sunday!";
  }
else
  {
  echo "Have a nice day!";
  }
?>

</body>
</html>
```

The PHP Switch Statement

Use the switch statement to select one of many blocks of code to be executed.

Syntax

```
switch (n)
{
case label1:
  code to be executed if n=label1;
  break;
case label2:
  code to be executed if n=label2;
  break;
default:
  code to be executed if n is different from both label1 and label2;
}
```

This is how it works: First we have a single expression *n* (most often a variable), that is evaluated once. The value of the expression is then compared with the values for each case in the structure. If there is a match, the block of code associated with that case is executed. Use

break to prevent the code from running into the next case automatically. The default statement is used if no match is found.

Example

```
<html>
<body>

<?php
$x=1;
switch
($x)
{
case 1:
  echo "Number 1";
  break;
case 2:
  echo "Number 2";
  break;
case 3:
  echo "Number 3";
  break;
default:
  echo "No number between 1 and 3";
}
?>

</body>
</html>
```

An array stores multiple values in one single variable.

What is an Array?

A variable is a storage area holding a number or text. The problem is, a variable will hold only one value.

An array is a special variable, which can store multiple values in one single variable.

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="Saab";
$cars2="Volvo";
$cars3="BMW";
```

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 best solution here is to use an array!

An array can hold all your variable values under a single name. And you can access the values by referring to the array name.

Each element in the array has its own index so that it can be easily accessed.

In PHP, there are three kind of arrays:

- **Numeric array** - An array with a numeric index
- **Associative array** - An array where each ID key is associated with a value
- **Multidimensional array** - An array containing one or more arrays

---

Numeric Arrays

A numeric array stores each array element with a numeric index.

There are two methods to create a numeric array.

1. In the following example the index are automatically assigned (the index starts at 0):

```
$cars=array("Saab","Volvo","BMW","Toyota");
```

2. In the following example we assign the index manually:

```
$cars[0]="Saab";
$cars[1]="Volvo";
$cars[2]="BMW";
$cars[3]="Toyota";
```

Example

In the following example you access the variable values by referring to the array name and index:

```
<?php
$cars[0]="Saab";
$cars[1]="Volvo";
$cars[2]="BMW";
$cars[3]="Toyota";
echo $cars[0] . " and " . $cars[1] . " are Swedish cars.";
?>
```

The code above will output:

Saab and Volvo are Swedish cars.

Associative Arrays

An associative array, each ID key is associated with a value.

When storing data about specific named values, a numerical array is not always the best way to do it.

With associative arrays we can use the values as keys and assign values to them.

Example 1

In this example we use an array to assign ages to the different persons:

```
$ages = array("Peter"=>32, "Quagmire"=>30, "Joe"=>34);
```

Example 2

This example is the same as example 1, but shows a different way of creating the array:

```
$ages['Peter'] = "32";
$ages['Quagmire'] = "30";
$ages['Joe'] = "34";
```

The ID keys can be used in a script:

```
<?php
$ages['Peter'] = "32";
$ages['Quagmire'] = "30";
$ages['Joe'] = "34";

echo "Peter is " . $ages['Peter'] . " years old.";
?>
```

The code above will output:

Peter is 32 years old.

Multidimensional Arrays

In a multidimensional array, each element in the main array can also be an array. And each element in the sub-array can be an array, and so on.

Example

In this example we create a multidimensional array, with automatically assigned ID keys:

```
$families = array
 (
 "Griffin"=>arra
 y (
 "Peter",
 "Lois",
 "Megan"
 ),
 "Quagmire"=>array
 (
 "Glenn"
 ),
 "Brown"=>arra
 y (
 "Cleveland"
 , "Loretta",
 "Junior"
 )
 );
```

The array above would look like this if written to the output:

```
Arra
y (
[Griffin] =>
  Array (
  [0] => Peter
  [1] => Lois
  [2] => Megan
  )
[Quagmire] =>
  Array (
  [0] => Glenn
  )
[Brown] =>
  Array (
  [0] => Cleveland
  [1] => Loretta
  [2] => Junior
  )
)
```

Example 2

Lets try displaying a single value from the array above:

```
echo "Is " . $families['Griffin'][2]
. " a part of the Griffin family?";
```

The code above will output:

Is Megan a part of the Griffin family?

Loops execute a block of code a specified number of times, or while a specified condition is true.

PHP Loops

Often when you write code, you want the same block of code to run over and over again in a row. Instead of adding several almost equal lines in a script we can use loops to perform a task like this.

In PHP, we have the following looping statements:

- **while** - loops through a block of code while a specified condition is true
- **do...while** - loops through a block of code once, and then repeats the loop as long as a specified condition is true
- **for** - loops through a block of code a specified number of times
- **foreach** - loops through a block of code for each element in an array

---

The while Loop

The while loop executes a block of code while a condition is true.

Syntax

while (*condition*)
 {
 code to be executed;
 }

Example

The example below first sets a variable *i* to 1 ($i=1;).

Then, the while loop will continue to run as long as *i* is less than, or equal to 5. *i* will increase by 1 each time the loop runs:

```
<html>
<body>

<?php
$i=1;
while($i<=5
)
 {
```

```
 echo "The number is " . $i . "<br>";
 $i++;
 }
?>

</body>
</html>
```

Output:

```
The number is 1
The number is 2
The number is 3
The number is 4
The number is 5
```

The do...while Statement

The do...while statement will always execute the block of code once, it will then check the condition, and repeat the loop while the condition is true.

Syntax

```
do
 {
 code to be executed;
 }
while (condition);
```

Example

The example below first sets a variable *i* to 1 ($i=1;).

Then, it starts the do...while loop. The loop will increment the variable *i* with 1, and then write some output. Then the condition is checked (is *i* less than, or equal to 5), and the loop will continue to run as long as *i* is less than, or equal to 5:

```
<html>
<body>

<?php
$i=1
; do
 {
 $i++;
 echo "The number is " . $i . "<br>";
 }
```

```
while ($i<=5);
?>

</body>
</html>
```

Output:

```
The number is 2
The number is 3
The number is 4
The number is 5
The number is 6
```

Loops execute a block of code a specified number of times, or while a specified condition is true.

The for Loop

The for loop is used when you know in advance how many times the script should run.

Syntax

```
for (init; condition; increment)
  {
  code to be executed;
  }
```

Parameters:

- *init*: Mostly used to set a counter (but can be any code to be executed once at the beginning of the loop)
- *condition*: Evaluated for each loop iteration. If it evaluates to TRUE, the loop continues. If it evaluates to FALSE, the loop ends.
- *increment*: Mostly used to increment a counter (but can be any code to be executed at the end of the iteration)

Note: The *init* and *increment* parameters above can be empty or have multiple expressions (separated by commas).

Example

The example below defines a loop that starts with i=1. The loop will continue to run as long as the variable *i* is less than, or equal to 5. The variable *i* will increase by 1 each time the loop runs:

```
<html>
<body>
```

```
<?php
for ($i=1; $i<=5; $i++)
  {
  echo "The number is " . $i . "<br>";
  }
?>

</body>
</html>
```

Output:

The number is 1
The number is 2
The number is 3
The number is 4
The number is 5

The foreach Loop

The foreach loop is used to loop through arrays.

Syntax

```
foreach ($array as $value)
  {
  code to be executed;
  }
```

For every loop iteration, the value of the current array element is assigned to $value (and the array pointer is moved by one) - so on the next loop iteration, you'll be looking at the next array value.

Example

The following example demonstrates a loop that will print the values of the given array:

```
<html>
<body>

<?php
$x=array("one","two","three");
foreach ($x as $value)
  {
  echo $value . "<br>";
  }
```

```
?>

</body>
</html>
```

Output:

```
one
two
three
```

The real power of PHP comes from its functions.

In PHP, there are more than 700 built-in functions.

PHP Built-in Functions

For a complete reference and examples of the built-in functions, please visit our PHP Reference.

PHP Functions

In this chapter we will show you how to create your own functions.

To keep the script from being executed when the page loads, you can put it into a function.

A function will be executed by a call to the function.

You may call a function from anywhere within a page.

Create a PHP Function

A function will be executed by a call to the function.

Syntax

```
function functionName()
{
code to be executed;
}
```

PHP function guidelines:

- Give the function a name that reflects what the function does
- The function name can start with a letter or underscore (not a number)

Example

A simple function that writes my name when it is called:

```
<html>
<body>

<?php
function writeName()
{
echo "Kai Jim Refsnes";
}

echo "My name is ";
writeName();
?>

</body>
</html>
```

Output:

My name is Kai Jim Refsnes

PHP Functions - Adding parameters

To add more functionality to a function, we can add parameters. A parameter is just like a variable.

Parameters are specified after the function name, inside the parentheses.

Example 1

The following example will write different first names, but equal last name:

```
<html>
<body>

<?php
function writeName($fname)
{
```

```
echo $fname . " Refsnes.<br>";
}

echo "My name is ";
writeName("Kai  Jim");
echo "My sister's name is ";
writeName("Hege");
echo "My brother's name is ";
writeName("Stale");
?>

</body>
</html>
```

Output:

My name is Kai Jim Refsnes.
My sister's name is Hege Refsnes.
My brother's name is Stale Refsnes.

Example 2

The following function has two parameters:

```
<html>
<body>

<?php
function writeName($fname,$punctuation)
{
echo $fname . " Refsnes" . $punctuation . "<br>";
}

echo "My name is ";
writeName("Kai Jim",".");
echo "My sister's name is ";
writeName("Hege","!");
echo "My brother's name is ";
writeName("Ståle","?");
?>

</body>
</html>
```

Output:

My name is Kai Jim Refsnes.
My sister's name is Hege Refsnes!
My brother's name is Ståle Refsnes?

PHP Functions - Return values

To let a function return a value, use the return statement.

Example

```
<html>
<body>

<?php
function add($x,$y)
{
$total=$x+$y
; return
$total;
}

echo "1 + 16 = " . add(1,16);
?>

</body>
</html>
```

Output:

1 + 16 = 17

The PHP $_GET and $_POST variables are used to retrieve information from forms, like user input.

PHP Form Handling

The most important thing to notice when dealing with HTML forms and PHP is that any form element in an HTML page will **automatically** be available to your PHP scripts.

Example

The example below contains an HTML form with two input fields and a submit button:

```
<html>
<body>

<form action="welcome.php" method="post">
Name: <input type="text" name="fname">
```

```
Age: <input type="text" name="age">
<input type="submit">
</form>

</body>
</html>
```

When a user fills out the form above and clicks on the submit button, the form data is sent to a PHP file, called "welcome.php":

"welcome.php" looks like this:

```
<html>
<body>

Welcome <?php echo $_POST["fname"];
?>!<br> You are <?php echo $_POST["age"]; ?>
years old.

</body>
</html>
```

Output could be something like this:

```
Welcome John!
You are 28 years old.
```

The PHP $_GET and $_POST variables will be explained in the next chapters.

Form Validation

User input should be validated on the browser whenever possible (by client scripts). Browser validation is faster and reduces the server load.

You should consider server validation if the user input will be inserted into a database. A good way to validate a form on the server is to post the form to itself, instead of jumping to a different page. The user will then get the error messages on the same page as the form. This makes it easier to discover the error.

In PHP, the predefined $_GET variable is used to collect values in a form with method="get".

The $_GET Variable

The predefined $_GET variable is used to collect values in a form with method="get"

TutorialsDuniya.com

Download FREE Computer Science Notes, Programs, Projects, Books PDF for any university student of BCA, MCA, B.Sc, B.Tech CSE, M.Sc, M.Tech at https://www.tutorialsduniya.com

- Algorithms Notes
- Artificial Intelligence
- Android Programming
- C & C++ Programming
- Combinatorial Optimization
- Computer Graphics
- Computer Networks
- Computer System Architecture
- DBMS & SQL Notes
- Data Analysis & Visualization
- Data Mining
- Data Science
- Data Structures
- Deep Learning
- Digital Image Processing
- Discrete Mathematics
- Information Security
- Internet Technologies
- Java Programming
- JavaScript & jQuery
- Machine Learning
- Microprocessor
- Operating System
- Operational Research
- PHP Notes
- Python Programming
- R Programming
- Software Engineering
- System Programming
- Theory of Computation
- Unix Network Programming
- Web Design & Development

Please Share these Notes with your Friends as well

Information sent from a form with the GET method is visible to everyone (it will be displayed in the browser's address bar) and has limits on the amount of information to send.

Example

```
<form action="welcome.php" method="get">
Name: <input type="text" name="fname">
Age: <input type="text" name="age">
<input type="submit">
</form>
```

When the user clicks the "Submit" button, the URL sent to the server could look something like this:

http://www.w3schools.com/welcome.php?fname=Peter&age=37

The "welcome.php" file can now use the $_GET variable to collect form data (the names of the form fields will automatically be the keys in the $_GET array):

```
Welcome <?php echo $_GET["fname"];
?>.<br> You are <?php echo $_GET["age"]; ?>
years old!
```

When to use method="get"?

When using method="get" in HTML forms, all variable names and values are displayed in the URL.

Note: This method should not be used when sending passwords or other sensitive information!

However, because the variables are displayed in the URL, it is possible to bookmark the page. This can be useful in some cases.

Note: The get method is not suitable for very large variable values. It should not be used with values exceeding 2000 characters.

In PHP, the predefined $_POST variable is used to collect values in a form with method="post".

The $_POST Variable

The predefined $_POST variable is used to collect values from a form sent with method="post".

Information sent from a form with the POST method is invisible to others and has no limits on the amount of information to send.

Note: However, there is an 8 MB max size for the POST method, by default (can be changed by setting the post_max_size in the php.ini file).

Example

```
<form action="welcome.php" method="post">
Name: <input type="text" name="fname">
Age: <input type="text" name="age">
<input type="submit">
</form>
```

When the user clicks the "Submit" button, the URL will look like this:

http://www.w3schools.com/welcome.php

The "welcome.php" file can now use the $_POST variable to collect form data (the names of the form fields will automatically be the keys in the $_POST array):

```
Welcome <?php echo $_POST["fname"];
?>!<br> You are <?php echo $_POST["age"]; ?>
years old.
```

When to use method="post"?

Information sent from a form with the POST method is invisible to others and has no limits on the amount of information to send.

However, because the variables are not displayed in the URL, it is not possible to bookmark the page.

The PHP $_REQUEST Variable

The predefined $_REQUEST variable contains the contents of both $_GET, $_POST, and $_COOKIE.

The $_REQUEST variable can be used to collect form data sent with both the GET and POST methods.

Example

```
Welcome <?php echo $_REQUEST["fname"];
?>!<br> You are <?php echo $_REQUEST["age"];
?> years old.
```

What is MySQL?

- MySQL is a database server
- MySQL is ideal for both small and large applications
- MySQL supports standard SQL
- MySQL compiles on a number of platforms
- MySQL is free to download and use

The data in MySQL is stored in database objects called tables.

A table is a collection of related data entries and it consists of columns and rows.

Databases are useful when storing information categorically. A company may have a database with the following tables: "Employees", "Products", "Customers" and "Orders".

PHP + MySQL

- PHP combined with MySQL are cross-platform (you can develop in Windows and serve on a Unix platform)

Database Tables

A database most often contains one or more tables. Each table is identified by a name (e.g. "Customers" or "Orders"). Tables contain records (rows) with data.

Below is an example of a table called "Persons":

LastName	FirstName	Address	City
Hansen	Ola	Timoteivn 10	Sandnes
Svendson	Tove	Borgvn 23	Sandnes
Pettersen	Kari	Storgt 20	Stavanger

The table above contains three records (one for each person) and four columns (LastName, FirstName, Address, and City).

Queries

A query is a question or a request.

With MySQL, we can query a database for specific information and have a recordset returned.

Look at the following query:

SELECT LastName FROM Persons

The query above selects all the data in the "LastName" column from the "Persons" table, and will return a recordset like this:

LastName
Hansen
Svendson
Pettersen

Download MySQL Database

If you don't have a PHP server with a MySQL Database, you can download MySQL for free here: http://www.mysql.com/downloads/

Facts About MySQL Database

One great thing about MySQL is that it can be scaled down to support embedded database applications. Perhaps it is because of this reputation that many people believe that MySQL can only handle small to medium-sized systems.

The truth is that MySQL is the de-facto standard database for web sites that support huge volumes of both data and end users (like Friendster, Yahoo, Google).

Look at http://www.mysql.com/customers/ for an overview of companies using MySQL.

The free MySQL database is very often used with PHP.

Create a Connection to a MySQL Database

Before you can access data in a database, you must create a connection to the database.

In PHP, this is done with the mysql_connect() function.

Syntax
mysql_connect(servername,username,password);

Parameter	Description
servername	Optional. Specifies the server to connect to. Default value is "localhost:3306"
username	Optional. Specifies the username to log in with. Default value is the name of the user that owns the server process
password	Optional. Specifies the password to log in with. Default is ""

Note: There are more available parameters, but the ones listed above are the most important. Visit our full PHP MySQL Reference for more details.

Example

In the following example we store the connection in a variable ($con) for later use in the script. The "die" part will be executed if the connection fails:

```
<?php
$con = mysql_connect("localhost","peter","abc123");
if (!$con)
  {
  die('Could not connect: ' . mysql_error());
  }

// some code
?>
```

Closing a Connection

The connection will be closed automatically when the script ends. To close the connection before, use the mysql_close() function:

```
<?php
$con = mysql_connect("localhost","peter","abc123");
if (!$con)
  {
  die('Could not connect: ' . mysql_error());
  }

// some code

mysql_close($con);
?>
```

A database holds one or multiple tables.

Create a Database

The CREATE DATABASE statement is used to create a database in MySQL.

Syntax
CREATE DATABASE database_name

To learn more about SQL, please visit our SQL tutorial.

To get PHP to execute the statement above we must use the mysql_query() function. This function is used to send a query or command to a MySQL connection.

Example

The following example creates a database called "my_db":

```
<?php
$con = mysql_connect("localhost","peter","abc123");
if (!$con)
  {
  die('Could not connect: ' . mysql_error());
  }

if (mysql_query("CREATE DATABASE my_db",$con))
  {
  echo "Database created";
  }
else
  {
  echo "Error creating database: " . mysql_error();
  }

mysql_close($con);
?>
```

Create a Table

The CREATE TABLE statement is used to create a table in MySQL.

Syntax

```
CREATE TABLE
table_name (
column_name1 data_type,
column_name2 data_type,
column_name3 data_type,
....
)
```

To learn more about SQL, please visit our SQL tutorial.

We must add the CREATE TABLE statement to the mysql_query() function to execute the command.

Example

The following example creates a table named "Persons", with three columns. The column names will be "FirstName", "LastName" and "Age":

```
<?php
$con = mysql_connect("localhost","peter","abc123");
if (!$con)
  {
  die('Could not connect: ' . mysql_error());
  }

// Create database
if (mysql_query("CREATE DATABASE my_db",$con))
  {
  echo "Database created";
  }
else
  {
  echo "Error creating database: " . mysql_error();
  }

// Create table
mysql_select_db("my_db", $con);
$sql = "CREATE TABLE
Persons (
FirstName
varchar(15), LastName
varchar(15), Age int
)";

// Execute query
mysql_query($sql,$con);
```

```
mysql_close($con);
?>
```

Important: A database must be selected before a table can be created. The database is selected with the mysql_select_db() function.

Note: When you create a database field of type varchar, you must specify the maximum length of the field, e.g. varchar(15).

The data type specifies what type of data the column can hold. For a complete reference of all the data types available in MySQL, go to our complete Data Types reference.

Primary Keys and Auto Increment Fields

Each table should have a primary key field.

A primary key is used to uniquely identify the rows in a table. Each primary key value must be unique within the table. Furthermore, the primary key field cannot be null because the database engine requires a value to locate the record.

The following example sets the personID field as the primary key field. The primary key field is often an ID number, and is often used with the AUTO_INCREMENT setting. AUTO_INCREMENT automatically increases the value of the field by 1 each time a new record is added. To ensure that the primary key field cannot be null, we must add the NOT NULL setting to the field.

Example

```
$sql = "CREATE TABLE
Persons (
personID int NOT NULL
AUTO_INCREMENT, PRIMARY
KEY(personID),
FirstName
varchar(15), LastName
varchar(15), Age int
)";

mysql_query($sql,$con);
```

Insert Data Into a Database Table

The INSERT INTO statement is used to add new records to a database table.

Syntax

It is possible to write the INSERT INTO statement in two forms.

The first form doesn't specify the column names where the data will be inserted, only their values:

```
INSERT INTO table_name
VALUES (value1, value2, value3,...)
```

The second form specifies both the column names and the values to be inserted:

```
INSERT INTO table_name (column1, column2,
column3,...) VALUES (value1, value2, value3,...)
```

To learn more about SQL, please visit our SQL tutorial.

To get PHP to execute the statements above we must use the mysql_query() function. This function is used to send a query or command to a MySQL connection.

Example

In the previous chapter we created a table named "Persons", with three columns; "Firstname", "Lastname" and "Age". We will use the same table in this example. The following example adds two new records to the "Persons" table:

```
<?php
$con = mysql_connect("localhost","peter","abc123");
if (!$con)
  {
  die('Could not connect: ' . mysql_error());
  }

mysql_select_db("my_db", $con);

mysql_query("INSERT INTO Persons (FirstName, LastName,
Age) VALUES ('Peter', 'Griffin',35)");

mysql_query("INSERT INTO Persons (FirstName, LastName,
Age) VALUES ('Glenn', 'Quagmire',33)");

mysql_close($con);
?>
```

Insert Data From a Form Into a Database

Now we will create an HTML form that can be used to add new records to the "Persons" table.

Here is the HTML form:

```
<html>
<body>

<form action="insert.php" method="post">
Firstname: <input type="text" name="firstname">
Lastname: <input type="text" name="lastname">
Age: <input type="text" name="age">
<input type="submit">
</form>

</body>
</html>
```

When a user clicks the submit button in the HTML form in the example above, the form data is sent to "insert.php".

The "insert.php" file connects to a database, and retrieves the values from the form with the PHP $_POST variables.

Then, the mysql_query() function executes the INSERT INTO statement, and a new record will be added to the "Persons" table.

Here is the "insert.php" page:

```
<?php
$con = mysql_connect("localhost","peter","abc123");
if (!$con)
  {
  die('Could not connect: ' . mysql_error());
  }

mysql_select_db("my_db", $con);

$sql="INSERT INTO Persons (FirstName, LastName,
Age) VALUES
('$_POST[firstname]','$_POST[lastname]','$_POST[age]')";

if (!mysql_query($sql,$con))
  {
  die('Error: ' . mysql_error());
  }
echo "1 record added";

mysql_close($con);
?>
```

Select Data From a Database Table

The SELECT statement is used to select data from a database.

Syntax
SELECT
column_name(s) FROM
table_name

To learn more about SQL, please visit our SQL tutorial.

To get PHP to execute the statement above we must use the mysql_query() function. This function is used to send a query or command to a MySQL connection.

Example

The following example selects all the data stored in the "Persons" table (The * character selects all the data in the table):

```
<?php
$con = mysql_connect("localhost","peter","abc123");
if (!$con)
  {
  die('Could not connect: ' . mysql_error());
  }

mysql_select_db("my_db", $con);

$result = mysql_query("SELECT * FROM

Persons"); while($row =

mysql_fetch_array($result))
  {
  echo $row['FirstName'] . " " . $row['LastName'];
  echo "<br />";
  }

mysql_close($con);
?>
```

The example above stores the data returned by the mysql_query() function in the $result variable.

Next, we use the mysql_fetch_array() function to return the first row from the recordset as an array. Each call to mysql_fetch_array() returns the next row in the recordset. The while loop loops through all the records in the recordset. To print the value of each row, we use the PHP $row variable ($row['FirstName'] and $row['LastName']).

The output of the code above will be:

Peter Griffin
Glenn Quagmire

Display the Result in an HTML Table

The following example selects the same data as the example above, but will display the data in an HTML table:

```
<?php
$con = mysql_connect("localhost","peter","abc123");
if (!$con)
  {
  die('Could not connect: ' . mysql_error());
  }

mysql_select_db("my_db", $con);

$result = mysql_query("SELECT * FROM

Persons"); echo "<table border='1'>
<tr>
<th>Firstname</th>
<th>Lastname</th>
</tr>";

while($row = mysql_fetch_array($result))
  {
  echo "<tr>";
  echo  "<td>"  .  $row['FirstName']  .
  "</td>"; echo "<td>" . $row['LastName'] .
  "</td>"; echo "</tr>";
  }
echo "</table>";

mysql_close($con);
?>
```

The output of the code above will be:

Firstname	Lastname
Glenn	Quagmire
Peter	Griffin

The WHERE clause

The WHERE clause is used to extract only those records that fulfill a specified criterion.

Syntax
SELECT
column_name(s) FROM
table_name
WHERE column_name operator value

To learn more about SQL, please visit our SQL tutorial.

To get PHP to execute the statement above we must use the mysql_query() function. This function is used to send a query or command to a MySQL connection.

Example

The following example selects all rows from the "Persons" table where "FirstName='Peter'":

```
<?php
$con = mysql_connect("localhost","peter","abc123");
if (!$con)
  {
  die('Could not connect: ' . mysql_error());
  }

mysql_select_db("my_db", $con);

$result = mysql_query("SELECT * FROM
Persons WHERE FirstName='Peter'");

while($row = mysql_fetch_array($result))
  {
  echo $row['FirstName'] . " " . $row['LastName'];
  echo "<br>";
  }
?>
```

The output of the code above will be:

Peter Griffin

The ORDER BY Keyword

The ORDER BY keyword is used to sort the data in a recordset.

The ORDER BY keyword sort the records in ascending order by default.

If you want to sort the records in a descending order, you can use the DESC keyword.

Syntax
SELECT
column_name(s) FROM
table_name
ORDER BY column_name(s) ASC|DESC

To learn more about SQL, please visit our SQL tutorial.

Example

The following example selects all the data stored in the "Persons" table, and sorts the result by the "Age" column:

```
<?php
$con = mysql_connect("localhost","peter","abc123");
if (!$con)
  {
  die('Could not connect: ' . mysql_error());
  }

mysql_select_db("my_db", $con);

$result = mysql_query("SELECT * FROM Persons ORDER BY

age"); while($row = mysql_fetch_array($result))
  {
  echo $row['FirstName'];
  echo " " . $row['LastName'];
  echo " " . $row['Age'];
  echo "<br>";
  }

mysql_close($con);
?>
```

The output of the code above will be:

Glenn Quagmire 33
Peter Griffin 35

Order by Two Columns

It is also possible to order by more than one column. When ordering by more than one column, the second column is only used if the values in the first column are equal:

```
SELECT
column_name(s) FROM
table_name
ORDER BY column1, column2
```

Update Data In a Database

The UPDATE statement is used to update existing records in a table.

Syntax
UPDATE table_name
SET column1=value,
column2=value2,... WHERE
some_column=some_value

Note: Notice the WHERE clause in the UPDATE syntax. The WHERE clause specifies which record or records that should be updated. If you omit the WHERE clause, all records will be updated!

To learn more about SQL, please visit our SQL tutorial.

To get PHP to execute the statement above we must use the mysql_query() function. This function is used to send a query or command to a MySQL connection.

Example

Earlier in the tutorial we created a table named "Persons". Here is how it looks:

FirstName LastName Age

Peter Griffin 35

Glenn Quagmire 33

The following example updates some data in the "Persons" table:

```
<?php
$con = mysql_connect("localhost","peter","abc123");
if (!$con)
  {
  die('Could not connect: ' . mysql_error());
  }

mysql_select_db("my_db", $con);

mysql_query("UPDATE Persons SET

Age=36
WHERE FirstName='Peter' AND LastName='Griffin'");

mysql_close($con);
?>
```

After the update, the "Persons" table will look like this:

FirstName LastName Age

Peter Griffin 36

Glenn Quagmire 33

Delete Data In a Database

The DELETE FROM statement is used to delete records from a database table.

Syntax
DELETE FROM table_name
WHERE some_column = some_value

Note: Notice the WHERE clause in the DELETE syntax. The WHERE clause specifies which record or records that should be deleted. If you omit the WHERE clause, all records will be deleted!

To learn more about SQL, please visit our SQL tutorial.

To get PHP to execute the statement above we must use the mysql_query() function. This function is used to send a query or command to a MySQL connection.

Example

Look at the following "Persons" table:

FirstName LastName Age

Peter Griffin 35

Glenn Quagmire 33

The following example deletes all the records in the "Persons" table where LastName='Griffin':

```
<?php
$con = mysql_connect("localhost","peter","abc123");
if (!$con)
  {
  die('Could not connect: ' . mysql_error());
  }

mysql_select_db("my_db", $con);

mysql_query("DELETE FROM Persons WHERE

LastName='Griffin'"); mysql_close($con);
?>
```

After the deletion, the table will look like this:

FirstName LastName Age

Glenn Quagmire 33

Create an ODBC Connection

With an ODBC connection, you can connect to any database, on any computer in your network, as long as an ODBC connection is available.

Here is how to create an ODBC connection to a MS Access Database:

1. Open the **Administrative Tools** icon in your Control Panel.
2. Double-click on the **Data Sources (ODBC)** icon inside.
3. Choose the **System DSN** tab.
4. Click on **Add** in the System DSN tab.
5. Select the **Microsoft Access Driver**. Click **Finish.**
6. In the next screen, click **Select** to locate the database.
7. Give the database a **Data Source Name (DSN)**.
8. Click **OK**.

Note that this configuration has to be done on the computer where your web site is located. If you are running Internet Information Server (IIS) on your own computer, the instructions above will work, but if your web site is located on a remote server, you have to have physical access to that server, or ask your web host to to set up a DSN for you to use.

Connecting to an ODBC

The odbc_connect() function is used to connect to an ODBC data source. The function takes four parameters: the data source name, username, password, and an optional cursor type.

The odbc_exec() function is used to execute an SQL statement.

Example

The following example creates a connection to a DSN called northwind, with no username and no password. It then creates an SQL and executes it:

```
$conn=odbc_connect('northwind',",");
$sql="SELECT * FROM customers";
$rs=odbc_exec($conn,$sql);
```

Retrieving Records

The odbc_fetch_row() function is used to return records from the result-set. This function returns true if it is able to return rows, otherwise false.

The function takes two parameters: the ODBC result identifier and an optional row number:

```
odbc_fetch_row($rs)
```

Retrieving Fields from a Record

The odbc_result() function is used to read fields from a record. This function takes two parameters: the ODBC result identifier and a field number or name.

The code line below returns the value of the first field from the record:

```
$compname=odbc_result($rs,1);
```

The code line below returns the value of a field called "CompanyName":

```
$compname=odbc_result($rs,"CompanyName");
```

Closing an ODBC Connection

The odbc_close() function is used to close an ODBC connection.

```
odbc_close($conn);
```

An ODBC Example

The following example shows how to first create a database connection, then a result-set, and then display the data in an HTML table.

```
<html>
<body>

<?php
$conn=odbc_connect('northwind','','');
if (!$conn)
  {exit("Connection Failed: " . $conn);}
$sql="SELECT * FROM customers";
```

```
$rs=odbc_exec($conn,$sql);
if (!$rs)
  {exit("Error in
SQL");} echo
"<table><tr>";
echo "<th>Companyname</th>";
echo "<th>Contactname</th></tr>";
while (odbc_fetch_row($rs))
  {
  $compname=odbc_result($rs,"CompanyName");
  $conname=odbc_result($rs,"ContactName");
  echo "<tr><td>$compname</td>";
  echo "<td>$conname</td></tr>";
  }
odbc_close($conn);
echo "</table>";
?>

</body>
</html>
```

What is Perl?

Perl is a programming language. Perl stands for Practical Report and Extraction Language. You'll notice people refer to 'perl' and "Perl". "Perl" is the programming language as a whole whereas 'perl' is the name of the core executable. There is no language called "Perl5" -- that just means "Perl version 5". Versions of Perl prior to 5 are very old and very unsupported. Some of Perl's many strengths are:

- **Speed of development.** You edit a text file, and just run it. You can develop programs very quickly like this. No separate compiler needed. I find Perl runs a program quicker than Java, let alone compare the complete modify-compile-run-oh no-forgot-that-semicolon sequence.
- **Power.** Perl's regular expressions are some of the best available. You can work with objects, sockets...everything a systems administrator could want. And that's just the standard distribution. Add the wealth of modules available on CPAN and you have it all. Don't equate scripting languages with toy languages.
- **Usuability.** All that power and capability can be learnt in easy stages. If you can write a batch file you can program Perl. You don't have to learn object oriented programming, but you can write OO programs in Perl. If autoincrementing non-existent variables scares you, make perl refuse to let you.There is always more than one way to do it in Perl. You decide your style of programming, and Perl will accommodate you.
- **Portability.** On the Superhighway to the Portability Panacea, Perl's Porsche powers past Java's jaded jalopy. Many people develop Perl scripts on NT, or Win95, then just FTP them to a Unix server where they run. No modification necessary.
- **Editing tools** You don't need the latest Integrated Development Environment for Perl. You can develop Perl scripts with any text editor. Notepad, vi, MS Word 97, or even direct off the console. Of course, you can make things easy and use one of the many freeware or shareware programmers file editors.
- **Price.** Yes, 0 guilders, pounds, dmarks, dollars or whatever. And the peer to peer support is also free, and often far better than you'd ever get by paying some company to answer the phone and tell you to do what you just tried several times already, then look up the same reference books you already own.

What can I do with Perl ?

Just two popular examples :

Go surf. Notice how many websites have dynamic pages with .pl or similar as the filename extension? That's Perl. It is the most popular language for CGI programming for many reasons, most of which are mentioned above. In fact, there are a great many more dynamic pages written with perl that may not have a *.pl* extension. If you code in Active Server Pages, then you should try using ActiveState's PerlScript. Quite frankly, coding in PerlScript rather than VBScript or JScript is like driving a car as opposed to riding a bicycle. Perl powers a good deal of the Internet.

If you are a Unix sysadmin you'll know about sed, awk and shell scripts. Perl can do everything they can do and far more besides. Furthermore, Perl does it much more efficiently and portably. Don't take my word for it, ask around.

If you are an NT sysadmin, chances are you aren't used to programming. In which case, the advantages of Perl may not be clear. Do you need it? Is it worth it? A few examples of how I use Perl to ease NT sysadmin life:

- **User account creation**. If you have a text file with the user's names in it, that is all you need. Create usernames automatically, generate a unique password for each one

and create the account, plus create and share the home directory, and set the permissions.

- **Event log munging.** NT has great Event Logging. Not so great Event Reading. You can use Perl to create reports on the event logs from multiple NT servers.
- **Anything else** that you would have used a batch file for, or wished that you could automate somehow. Now you *can*.

- single word: atom, chain
- multi word: atomName, centralAtomName
- constants: CALPHA_ATOM_NAME or ATOM_NAME_CALPHA, ATOM_NAME_CBETA

Perl is not type-safe and this can cause confusion and errors. Use a limited prefix notation for such common basic types as array, hash, FileHandle.

- array refs ('a' prefix): aAtoms, aChains
- hash refs ('h' prefix): hNames2Places, hChains
- FileHandle objects ('fh' prefix): fhIn, fhOut, fhPdb
- or ("ist"=input stream, "ost"=output stream): ostPdb, istMsa

Functions

- single word: Trim()
- multi word: OpenFilesForReading()

- single word: Assert
- multi word: FileIoHelper

Classes

As for modules but with 'C' prefix: CStopwatch, CWindowPanel, Pdb::CResidue

Instance methods

- public method: plot(), getColour(), classifyHetGroups()
- private method: _plot(), _getColour(), _classifyHetGroups()
- accessor methods same as JavaBeans: getProperty(), setProperty(), isProperty()

Perl, perl or PeRl?

There is also a certain amount of confusion regarding the capitalization of Perl. Should it be written Perl or perl? Larry Wall now uses ―Perl‖ to signify the language proper and ―perl‖ to signify the implementation of the language.

Perl History

Version	Date	Version Details
Perl 0		Introduced Perl to Larry Wall's office associates
Perl 1	Jan 1988	Introduced Perl to the world
Perl 2	Jun 1988	Introduced Harry Spencer's regular expression package
Perl 3	Oct 1989	Introduced the ability to handle binary data
Perl 4	Mar 1991	Introduced the first ―Camel‖ book (Programming Perl, by Larry Wall, Tom Christiansen, and Randal L Schwartz; O'Reilly & Associates). The book drove the name change, just so it could refer to Perl 4, instead of Perl 3.
Feb 1993	Feb 1993	The last stable release of Perl 4
Perl 5	Oct 1994	The first stable release of Perl 5, which introduced a number of new features and a complete rewrite.
Perl .005_02	Aug 1998	The next major stable release
Perl .005_03	Mar 1999	The last stable release before 5.6
Perl 5.6	Mar 2000	Introduced unified **fork** support, better threading, an updated Perl compiler, and the **our** keyword

Main Perl Features:

1. **Perl Is Free:**
   Perl's source code is open and free anybody can download the C source that constitutes a Perl interpreter. Furthermore, you can easily extend the core functionality of Perl both within the realms of the interpreted language and by modifying the Perl source code.
2. **Perl Is Simple to Learn, Concise, and Easy to Read:**
   It has a syntax similar to C and shell script, among others, but with a less restrictive format. Most programs are quicker to write in Perl because of its use of built-in functions and a huge standard and contributed library. Most programs are also quicker to execute than other languages because of Perl's internal architecture.
3. **Perl Is Fast**
   Compared to most scripting languages, this makes execution almost as fast as compiled C code. But, because the code is still interpreted, there is no compilation process, and applications can be written and edited much faster than with other languages, without any of the performance problems normally associated with an interpreted language.
4. **Perl Is Extensible**
   You can write Perl-based packages and modules that extend the functionality of the language. You can also call external C code directly from Perl to extend the functionality.
5. **Perl Has Flexible Data Types**
   You can create simple variables that contain text or numbers, and Perl will treat the variable data accordingly at the time it is used.
6. **Perl Is Object Oriented**
   Perl supports all of the object-oriented features—inheritance, polymorphism, and encapsulation.
7. **Perl Is Collaborative**
   There is a huge network of Perl programmers worldwide. Most programmers supply, and use, the modules and scripts available via CPAN, the Comprehensive Perl Archive Network

Compiler or Interpreter:

a. **Compiler:** A program that decodes instructions written in a higher order language and produces an assembly language program.
   A compiler that generates machine language for a different type of computer than the one the compiler is running in.

b. **Interpreter:** In computing, an interpreter is a computer program that reads the source code of another compute program and executes that program. *A program that translates and executes source language statements one line at a time.*

c. **Difference between Compiler and Interpreter**
   A compiler first takes in the entire program, checks for errors, compiles it and then executes it. Whereas, an interpreter does this line by line, so it takes one line, checks it for errors and then executes it.
   Example of Compiler – Java
   Example of Interpreter – PHP

d. **Perl is interpreter or compiler?**
   Neither, and both. Perl is a scripting language. There is a tool, called perl, intended to run programs written in the perl language.
   "Compiled" languages are ones like C and C++, where you have to take the source code, compile it into an executable file, and THEN run it.
   "Interpreted" languages, like Perl, PHP, and Ruby, are ones which do NOT require pre-compiling.
   They are generally compiled on-the-fly (which is what the perl command-line tool does) into *opcodes*, and then run. So, Perl is an interpreted language because a tool reads the source code and immediately runs it.

Perl is a compiler because it has to compile that source code before it can be run while it's being interpreted.

Popular "Myth conceptions"

1. **It's only for the Web**
   Probably the most famous of the myths is that Perl is a language used, designed, and created exclusively for developing web-based applications.
2. **It's Not Maintenance Friendly**
   Any good (or bad) programmer will tell you that anybody can write unmaintainable code in any language. Many companies and individuals write maintainable programs using Perl.
3. **It's Only for Hackers**
   Perl is used by a variety of companies, organizations, and individuals. Everybody from programming beginners through ―hackers‖ up to multinational corporations use Perl to solve their problems.
4. **It's a Scripting Language**
   In Perl, there is no difference between a script and program. Many large programs and projects have been written entirely in Perl.
5. **There's No Support**
   The Perl community is one of the largest on the Internet, and you should be able to find someone, somewhere, who can answer your questions or help you with your problems.
6. **All Perl Programs Are Free**
   Although you generally write and use Perl programs in their native source form, this does not mean that everything you write is free. Perl programs are your own intellectual property and can be bought, sold, and licensed just like any other program.
7. **There's No Development Environment**
   Perl programs are text based, you can use any source-code revision-control system. The most popular solution is CVS, or Concurrent Versioning System, which is now supported under Unix, MacOS and Windows.
8. **Perl Is a GNU Project**
   While the GNU project includes Perl in its distributions, there is no such thing as ―GNU Perl.‖ Perl is not produced or maintained by GNU and the Free Software Foundation. Perl is also made available on a much more open license than the GNU Public License.
9. **Perl Is Difficult to Learn**
   Because Perl is similar to a number of different languages, it is not only easy to learn but also easy to continue learning. Its structure and format is very similar to C, **awk**, shell script, and, to a greater or lesser extent, even BASIC.

Perl Overview:

Installing and using Perl

Perl was developed by Larry Wall. It started out as a scripting language to supplement *rn*, the USENET reader. It available on virtually every computer platform.

Perl is an interpreted language that is optimized for string manipulation, I/O, and system tasks. It has built in for most of the functions in section 2 of the UNIX manuals -- very popular with sys administrators. It incorporates syntax elements from the *Bourne shell*,

csh, awk, sed, grep, and C. It provides a quick and effective way to write interactive web applications

Writing a Perl Script

Perl scripts are just text files, so in order to actually "write" the script, all you need to do is create a text file using your favorite text editor. Once you've written the script, you tell Perl to execute the text file you created.

Under Unix, you would use

$ perl myscript.pl

and the same works under Windows:

TutorialsDuniya.com

Download FREE Computer Science Notes, Programs, Projects, Books PDF for any university student of BCA, MCA, B.Sc, B.Tech CSE, M.Sc, M.Tech at https://www.tutorialsduniya.com

- Algorithms Notes
- Artificial Intelligence
- Android Programming
- C & C++ Programming
- Combinatorial Optimization
- Computer Graphics
- Computer Networks
- Computer System Architecture
- DBMS & SQL Notes
- Data Analysis & Visualization
- Data Mining
- Data Science
- Data Structures
- Deep Learning
- Digital Image Processing
- Discrete Mathematics
- Information Security
- Internet Technologies
- Java Programming
- JavaScript & jQuery
- Machine Learning
- Microprocessor
- Operating System
- Operational Research
- PHP Notes
- Python Programming
- R Programming
- Software Engineering
- System Programming
- Theory of Computation
- Unix Network Programming
- Web Design & Development

Please Share these Notes with your Friends as well

C:\> perl myscript.pl

Under Mac OS, you need to drag and drop the file onto the *MacPerl* application. Perl scripts have a *.pl* extension, even under Mac OS and Unix.

Perl Under Unix

The easiest way to install Perl modules on Unix is to use the CPAN module. For example:

shell> perl -MCPAN -e shell
cpan> install DBI
cpan> install DBD::mysql

The DBD::mysql installation runs a number of tests. These tests attempt to connect to the local MySQL server using the default user name and password. (The default user name is your login name on Unix, and ODBC on Windows. The default password is "no password.") If you cannot connect to the server with those values (for example, if your account has a password), the tests fail. You can use force install DBD::mysql to ignore the failed tests.

DBI requires the Data::Dumper module. It may be installed; if not, you should install it before installing DBI.

Perl Under Windows

1. Log on to the Web server computer as an administrator.
2. Download the ActivePerl installer from the following ActiveState Web site: http://www.activestate.com/ (http://www.activestate.com/)
3. Double-click the **ActivePerl** installer.
4. After the installer confirms the version of ActivePerl that it is going to be installed, click **Next**.
5. If you agree with the terms of the license agreement, click **I accept the terms in the license agreement**, and then click **Next**. Click **Cancel** if you do not accept the license agreement. If you do so, you cannot continue the installation.
6. To install the whole ActivePerl distribution package (this step is recommended), click **Next** to continue the installation. The software is installed in the default location (typically C:\Perl).
7. To customize the individual components or to change the installation folder, follow the instructions that appears on the screen.
8. When you are prompted to confirm the addition features that you want to configure during the installation, click any of the following settings, and then click **Next**:
    a. **Add Perl to the PATH environment variable**: Click this setting if you want to use Perl in a command prompt without requiring the full path to the Perl interpreter.
    b. **Create Perl file extension association**: Click this setting if you want to allow Perl scripts to be automatically run when you use a file that has the Perl file name extension (.pl) as a command name.
    c. **Create IIS script mapping for Perl**: Click this setting to configure IIS to identify Perl scripts as executable CGI programs according to their file name extension.
    d. **Create IIS script mapping for Perl ISAPI**: Click this setting to use Perl scripts as an ISAPI filter.
9. Click **Install** to start the installation process.
10. After the installation has completed, click **Finish**.

Perl Components:

Variables

Perl Variables with the techniques of handling them are an important part of the Perl language. As a language-type script, Perl was designed to handle huge amounts of data text. Working with variables is fairly straightforward given that it is not necessary to define and allocate them, so no sophisticated techniques for the release of memory occupied by them.

As general information, to note that the names of Perl variables contain alphabetic characters, numbers and the underscore (_) character and are case sensitive.

A specific language feature is that variables have a non-alphabetical prefix that fashion somewhat cryptic the language.

a. *scalar variables – starting with $*
b. *array variables – starting with @*
c. *hashes or associative arrays indicated by %*

The $, @ and % characters actually predefine the variable type in Perl. Perl language also offers some built-in predefined variables that facilitate and shorten the programming code.

Operators

The operators work with numbers and strings and manipulate data objects called operands. We found the operators in expressions which we need to evaluate.

Statements

The statements are one of the most important topics in the Perl language, actually for any programming language. We use statements in order to process or evaluate the expressions. Perl uses the values returned by statements to evaluate or process other statements.

A Perl statement ends with the semicolon character (;) which is used to tell interpreter that the statement was complete.

Subroutines (Functions)

Definition: *Subroutine* is a block of source code which does one or some tasks with specified purpose.

Advantages:

1. It reduces the Complexity in a program by reducing the code.
2. It also reduces the Time to run a program.In other way,It's directly proportional to Complexity.
3. It's easy to find-out the errors due to the blocks made as function definition outside the main function.

Modules:

A ***Perl module*** is a discrete component of software for the Perl programming language. Technically, it is a particular set of conventions for using Perl's package mechanism that has become universally adopted.

A module defines its source code to be in a *package* (much like a Java package), the Perl mechanism for defining namespaces, e.g. *CGI* or *Net::FTP* or *XML::Parser*; the file structure mirrors the namespace structure (e.g. the source code for *Net::FTP* is in *Net/FTP.pm*). A collection of modules, with accompanying documentation, build scripts, and usually a test suite, compose a ***distribution***.

Perl Parsing Rules

The Execution Process:

The execution process of *perl* contains the following steps

- It takes raw input,
- Parses each statement and converts it into a series of opcodes,
- Builds a suitable opcode tree,
- Executes the opcodes within a Perl "virtual machine."

It classifies only two stages

- The parsing stage and the
- Execution or run-time stage

The Perl parser thinks about all of the following when it looks at a source line:

- **Basic syntax** The core layout, line termination, and so on
- **Comments** If a comment is included, ignore it
- **Component identity** Individual terms (variables, functions and numerical and textual constants) are identified
- **Bare words** Character strings that are not identified as valid terms

- **Precedence** Once the individual items are identified, the parser processes the statements according to the precedence rules, which apply to all operators and terms
- **Context** What is the context of the statement, are we expecting a list or scalar, a number or a string, and so on. This actually happens during the evaluation of individual elements of a line, which is why we can nest functions such as sort, reverse, and keys into a single statement line
- **Logic Syntax** For logic operations, the parser must treat different values, whether constant- or variable-based, as true or false values

All of these present some fairly basic and fundamental rules about how Perl looks at an entire script.

The basic rules govern such things as line termination and the treatment of white space. These basic rules are

- Lines must start with a token that does not expect a left operand
- Lines must be terminated with a semicolon, except when it's the last line of a block, where the semicolon can be omitted.
- White space is only required between tokens that would otherwise be confusing, so spaces, tabs, newlines, and comments (which Perl treats as white space) are ignored. The line sub menu *{print"menu"}* works as it would if it were more neatly spaced.
- Lines may be split at any point, providing the split is logically between two tokens.

Component Identity

When Perl fails to identify an item as one of the predefined operators, it treats the character sequence as a "term." Terms are core parts of the Perl language and include variables, functions, and quotes. The term-recognition system uses these rules:

- Variables can start with a letter, number, or underscore, providing they follow a suitable variable character, such as $, @, or %.
- Variables that start with a letter or underscore can contain any further combination of letters, numbers, and underscore characters.
- Variables that start with a number can only consist of further numbers—be wary of using variable names starting with digits. The variables such as $0 through to $9 are used for group matches in regular expressions.
- Subroutines can only start with an underscore or letter, but can then contain any combination of letters, numbers, and underscore characters.
- Case is significant—$VAR, $Var, and $var are all different variables.
- Each of the three main variable types have their own name space—$var, @var, and %var are all separate variables.
- File handles should use all uppercase characters—this is only a convention, not a rule, but it is useful for identification purposes.

Operators and Precedence

a) Arithmetic Operators:

The following are the arithmetic operators in Perl.

Operator	Description
+	Addition operator
-	Subtraction operator
*	Multiplication operator
/	Division operator
%	Modulus operator
**	Exponentiation operator

The operators +, -, *, / take two operands and return the sum, difference, product and quotient respectively. Perl does a floating point division not an integral division. To get the integral quotient one has to use *int()* function. Say if you divide *"int(5/2)"* the result will be 2, to get the exact result use the code below.

b) Assignment Operators

Operator	Description
=	Normal Assignment
+=	Add and Assign
-=	Subtract and Assign
*=	Multiply and Assign
/=	Divide and Assign
%=	Modulus and Assign
**=	Exponent and Assign

Everyone knows how to use the assignment operator (=). There are other operators, when used with "=" gives a different result.

c) Increment/Decrement Operators

The following are the auto increment, decrement operators in Perl.

Operator	Description
++	Auto-increment operator
--	Auto-decrement operator

The usage of auto increment operators are same as in C Language. In prefix decrement / increment first thevalue is increased or decreased then the new value is returned eg: "++$a", "--$a".

The vice versa of the above, is post decrement/increment operators. First the old value is returned then incremented or decremented to give the result. eg: "$a++", "$a--"

d) Comparison Operators

Operator		Function
=	eq	Equal to Operator
!=	ne	Not Equal to Operator
<	lt	Less than Operator
>	gt	Greater than Operator
<=	le	Less than or Equal to Operator
>=	ge	Greater than or Equal to operator

Interface with CGI

What is CGI ?

The Common Gateway Interface, or CGI, is a set of standards that define how information is exchanged between the web server and a custom script.

The CGI specs are currently maintained by the NCSA and NCSA defines CGI is as follows −

The Common Gateway Interface, or CGI, is a standard for external gateway programs to interface with information servers such as HTTP servers.

The current version is CGI/1.1 and CGI/1.2 is under progress.

Web Browsing

To understand the concept of CGI, lets see what happens when we click a hyper link to browse a particular web page or URL.

- Your browser contacts the HTTP web server and demand for the URL ie. filename.
- Web Server will parse the URL and will look for the filename in if it finds that file then sends back to the browser otherwise sends an error message indicating that you have requested a wrong file.
- Web browser takes response from web server and displays either the received file or error message.

However, it is possible to set up the HTTP server so that whenever a file in a certain directory is requested that file is not sent back; instead it is executed as a program, and whatever that program outputs is sent back for your browser to display. This function is called the Common Gateway Interface or CGI and the programs are called CGI scripts. These CGI programs can be a PERL Script, Shell Script, C or C++ program etc.

CGI Architecture Diagram

Web Server Support & Configuration

Before you proceed with CGI Programming, make sure that your Web Server supports CGI and it is configured to handle CGI Programs. All the CGI Programs be executed by the HTTP server are kept in a pre-configured directory. This directory is called CGI Directory and by convention it is named as /cgi-bin. By convention PERL CGI files will have extention as **.cgi**.

First CGI Program

```
#!/usr/bin/perl

print "Content-type:text/html\r\n\r\n";
print '<html>';
print '<head>';
print '<title>Hello Word - First CGI Program</title>';
print '</head>';
print '<body>';
print '<h2>Hello Word! This is my first CGI program</h2>';
print '</body>';
print '</html>';

1;
```

Output

```
Hello Word! This is my first CGI program
```

HTTP Header

The line **Content-type:text/html\r\n\r\n** is part of HTTP header which is sent to the browser to understand the content. All the HTTP header will be in the following form

```
HTTP Field Name: Field Content
```

For Example

Content-type:text/html\r\n\r\n

There are few other important HTTP headers which you will use frequently in your CGI Programming.

S.No.	Header & Description
1	**Content-type: String** A MIME string defining the format of the file being returned. Example is Content-type:text/html
2	**Expires: Date String** The date the information becomes invalid. This should be used by the browser to decide when a page needs to be refreshed. A valid date string should be in the format

01 Jan 1998 12:00:00 GMT.

Location: URL String

3 The URL that should be returned instead of the URL requested. You can use this filed to redirect a request to any file.

Last-modified: String

4 The date of last modification of the resource.

Content-length: String

5 The length, in bytes, of the data being returned. The browser uses this value to report the estimated download time for a file.

Set-Cookie: String

6 Set the cookie passed through the *string*

CGI Environment Variables

All the CGI program will have access to the following environment variables. These variables play an important role while writing any CGI program.

S.No.	Variable Name & Description
1	**CONTENT_TYPE** The data type of the content. Used when the client is sending attached content to the server. For example file upload etc.
2	**CONTENT_LENGTH** The length of the query information. It's available only for POST requests.
3	**HTTP_COOKIE** Return the set cookies in the form of key & value pair.
4	**HTTP_USER_AGENT** The User-Agent request-header field contains information about the user agent originating the request. Its name of the web browser.
5	**PATH_INFO** The path for the CGI script.
6	**QUERY_STRING** The URL-encoded information that is sent with GET method request.
7	**REMOTE_ADDR** The IP address of the remote host making the request. This can be useful for logging or for authentication purpose.

REMOTE_HOST

8 The fully qualified name of the host making the request. If this information is not available then REMOTE_ADDR can be used to get IR address.

REQUEST_METHOD

9 The method used to make the request. The most common methods are GET and POST.

SCRIPT_FILENAME

10 The full path to the CGI script.

SCRIPT_NAME

11 The name of the CGI script.

SERVER_NAME

12 The server's hostname or IP Address.

SERVER_SOFTWARE

13 The name and version of the software the server is running.

```
#!/usr/bin/perl

print "Content-type: text/html\n\n";
print "<font size=+1>Environment</font>\n";

foreach (sort keys %ENV) {
   print "<b>$_</b>: $ENV{$_}<br>\n";
}

1;
```

Output

```
Environment CONTEXT_DOCUMENT_ROOT:
CONTEXT_PREFIX:
DOCUMENT_ROOT:
GATEWAY_INTERFACE:
GEOIP_ADDR:
GEOIP_CONTINENT_CODE:
GEOIP_COUNTRY_CODE:
GEOIP_COUNTRY_NAME:
HTTP_ACCEPT:
HTTP_ACCEPT_ENCODING:
HTTP_ACCEPT_LANGUAGE:
HTTP_COOKIE:
HTTP_HOST:
HTTP_UPGRADE_INSECURE_REQUESTS:
HTTP_USER_AGENT:
HTTP_VIA:
HTTP_X_FORWARDED_FOR:
HTTP_X_FORWARDED_PROTO:
HTTP_X_HOST:
PATH:
QUERY_STRING:
REMOTE_ADDR:
```

```
REMOTE_PORT:
REQUEST_METHOD:
REQUEST_SCHEME:
REQUEST_URI:
SCRIPT_FILENAME:
SCRIPT_NAME:
SCRIPT_URI:
SCRIPT_URL:
SERVER_ADDR:
SERVER_ADMIN:
SERVER_NAME:
SERVER_PORT:
SERVER_PROTOCOL:
SERVER_SIGNATURE:
SERVER_SOFTWARE:
UNIQUE_ID:
```

How To Raise a "File Download" Dialog Box ?

Sometime it is desired that you want to give option where a use will click a link and it will pop up a "File Download" dialogue box to the user in stead of displaying actual content. This is very easy and will be achived through HTTP header.

This HTTP header will be different from the header mentioned in previous section.

For example,if you want make a **FileName** file downloadable from a given link then its syntax will be as follows.

```
#!/usr/bin/perl

# HTTP Header
print "Content-Type:application/octet-stream; name=\"FileName\"\r\n";
print "Content-Disposition: attachment; filename=\"FileName\"\r\n\n";

# Actual File Content will go hear.
open( FILE, "<FileName" );
while(read(FILE, $buffer, 100) ){
   print("$buffer");
}
```

GET and POST Methods

You must have come across many situations when you need to pass some information from your browser to web server and ultimately to your CGI Program. Most frequently browser uses two methods two pass this information to web server. These methods are GET Method and POST Method.

Passing Information using GET method

The GET method sends the encoded user information appended to the page request. The page and the encoded information are separated by the ? character as follows −
http://www.test.com/cgi-bin/hello.cgi?key1=value1&key2=value2

The GET method is the defualt method to pass information from browser to web server and it produces a long string that appears in your browser's Location:box. Never use the GET method if you have password or other sensitive information to pass to the server. The GET method has size limtation: only 1024 characters can be in a request string.

This information is passed using QUERY_STRING header and will be accessible in your CGI Program through QUERY_STRING environment variable.

You can pass information by simply concatenating key and value pairs along with any URL or you can use HTML <FORM> tags to pass information using GET method.

Simple URL Example : Get Method

Here is a simple URL which will pass two values to hello_get.cgi program using GET method.

Below is hello_get.cgi script to handle input given by web browser.

```
#!/usr/bin/perl

local ($buffer, @pairs, $pair, $name, $value, %FORM);
# Read in text
$ENV{'REQUEST_METHOD'} =~ tr/a-z/A-Z/;

if ($ENV{'REQUEST_METHOD'} eq "GET") {
   $buffer = $ENV{'QUERY_STRING'};
}

# Split information into name/value pairs
@pairs = split(/&/, $buffer);

foreach $pair (@pairs) {
   ($name, $value) = split(/=/, $pair);
   $value =~ tr/+/ /;
   $value =~ s/%(..)/pack("C", hex($1))/eg;
   $FORM{$name} = $value;
}

$first_name = $FORM{first_name};
$last_name  = $FORM{last_name};

print "Content-type:text/html\r\n\r\n";
print "<html>";
print "<head>";
print "<title>Hello - Second CGI Program</title>";
print "</head>";
print "<body>";
print "<h2>Hello $first_name $last_name - Second CGI Program</h2>";
print "</body>";
print "</html>";

1;
```

Output

```
Hello ZARA ALI .....
```

Simple FORM Example: GET Method

Here is a simple example which passes two values using HTML FORM and submit button. We are going to use same CGI script hello_get.cgi to handle this input.

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

   Last Name: <input type = "text" name = "last_name">
   <input type = "submit" value = "Submit">
</FORM>
```

Here is the actual output of the above form, You enter First and Last Name and then click submit button to see the result.

First Name:

Last Name:

Passing Information using POST method

A generally more reliable method of passing information to a CGI program is the POST method. This packages the information in exactly the same way as GET methods, but instead of sending it as a text string after a ? in the URL it sends it as a separate message. This message comes into the CGI script in the form of the standard input.

Below is hello_post.cgi script to handle input given by web browser. This script will handle GET as well as POST method.

```
#!/usr/bin/perl

local ($buffer, @pairs, $pair, $name, $value, %FORM);
# Read in text
$ENV{'REQUEST_METHOD'} =~ tr/a-z/A-Z/;

if ($ENV{'REQUEST_METHOD'} eq "POST"){
   read(STDIN, $buffer, $ENV{'CONTENT_LENGTH'});
}else {
   $buffer = $ENV{'QUERY_STRING'};
}

# Split information into name/value pairs
@pairs = split(/&/, $buffer);

foreach $pair (@pairs) {
   ($name, $value) = split(/=/, $pair);
   $value =~ tr/+/ /;
   $value =~ s/%(..)/pack("C", hex($1))/eg;
   $FORM{$name} = $value;
```

```
}

$first_name = $FORM{first_name};
$last_name  = $FORM{last_name};

print "Content-type:text/html\r\n\r\n";
print "<html>";
print "<head>";
print "<title>Hello - Second CGI Program</title>";
print "</head>";
print "<body>";
print "<h2>Hello $first_name $last_name - Second CGI Program</h2>";
print "</body>";
print "</html>";

1;
```

Let us take again same example as above, which passes two values using HTML FORM and submit button. We are going to use CGI script hello_post.cgi to handle this input.

```
<FORM action = "/cgi-bin/hello_post.cgi" method="POST">
   First Name: <input type="text" name="first_name">  <br>

   Last Name: <input type="text" name="last_name">

   <input type="submit" value="Submit">
</FORM>
```

Here is the actual output of the above form, You enter First and Last Name and then click submit button to see the result.

First Name:

Last Name:

Passing Checkbox Data to CGI Program

Checkboxes are used when more than one option is required to be selected.

Here is example HTML code for a form with two checkboxes

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

The result of this code is the following form

- Maths
- Physics

Below is checkbox.cgi script to handle input given by web browser for radio button.

```
#!/usr/bin/perl

local ($buffer, @pairs, $pair, $name, $value, %FORM);
# Read in text
$ENV{'REQUEST_METHOD'} =~ tr/a-z/A-Z/;

if ($ENV{'REQUEST_METHOD'} eq "POST"){
   read(STDIN, $buffer, $ENV{'CONTENT_LENGTH'});
}else {
   $buffer = $ENV{'QUERY_STRING'};
}

# Split information into name/value pairs
@pairs = split(/&/, $buffer);

foreach $pair (@pairs) {
   ($name, $value) = split(/=/, $pair);
   $value =~ tr/+/ /;
   $value =~ s/%(..)/pack("C", hex($1))/eg;
   $FORM{$name} = $value;
}

if( $FORM{maths} ){
   $maths_flag ="ON";
}else{
   $maths_flag ="OFF";
}

if( $FORM{physics} ){
   $physics_flag ="ON";
}else{
   $physics_flag ="OFF";
}

print "Content-type:text/html\r\n\r\n";
print "<html>";
print "<head>";
print "<title>Checkbox - Third CGI Program</title>";
print "</head>";
print "<body>";
print "<h2> CheckBox Maths is : $maths_flag</h2>";
print "<h2> CheckBox Physics is : $physics_flag</h2>";
print "</body>";
print "</html>";

1;
```

Passing Radio Button Data to CGI Program

Radio Buttons are used when only one option is required to be selected.

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

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

The result of this code is the following form −

Maths

Physics

Below is radiobutton.cgi script to handle input given by web browser for radio button.

```
#!/usr/bin/perl

local ($buffer, @pairs, $pair, $name, $value, %FORM);
# Read in text
$ENV{'REQUEST_METHOD'} =~ tr/a-z/A-Z/;

if ($ENV{'REQUEST_METHOD'} eq "POST") {
   read(STDIN, $buffer, $ENV{'CONTENT_LENGTH'});
}else {
   $buffer = $ENV{'QUERY_STRING'};
}

# Split information into name/value pairs
@pairs = split(/&/, $buffer);

foreach $pair (@pairs) {
   ($name, $value) = split(/=/, $pair);
   $value =~ tr/+/ /;
   $value =~ s/%(..)/pack("C", hex($1))/eg;
   $FORM{$name} = $value;
}
$subject = $FORM{subject};
print "Content-type:text/html\r\n\r\n";
print "<html>";
print "<head>";
print "<title>Radio - Fourth CGI Program</title>";
print "</head>";
print "<body>";
print "<h2> Selected Subject is $subject</h2>";
print "</body>";
print "</html>";

1;
```

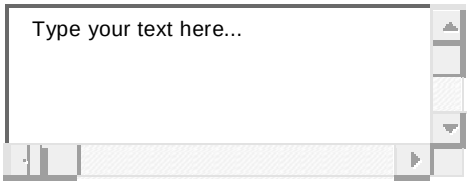
Passing Text Area Data to CGI Program

TEXTAREA element is used when multiline text has to be passed to the CGI Program.

Here is example HTML code for a form with a TEXTAREA box −

```
<form action = "/cgi-bin/textarea.cgi" method = "POST" target = "_blank">
   <textarea name = "textcontent" cols = 40 rows = 4>
      Type your text here...
   </textarea>
   <input type = "submit" value = "Submit">
</form>
```

The result of this code is the following form −



Below is textarea.cgi script to handle input given by web browser.

```
#!/usr/bin/perl

local ($buffer, @pairs, $pair, $name, $value, %FORM);
# Read in text
$ENV{'REQUEST_METHOD'} =~ tr/a-z/A-Z/;

if ($ENV{'REQUEST_METHOD'} eq "POST") {
   read(STDIN, $buffer, $ENV{'CONTENT_LENGTH'});
}else {
   $buffer = $ENV{'QUERY_STRING'};
}

# Split information into name/value pairs
@pairs = split(/&/, $buffer);

foreach $pair (@pairs) {
   ($name, $value) = split(/=/, $pair);
   $value =~ tr/+/ /;
   $value =~ s/%(..)/pack("C", hex($1))/eg;
   $FORM{$name} = $value;
}

$text_content = $FORM{textcontent};

print "Content-type:text/html\r\n\r\n";
print "<html>";
print "<head>";
print "<title>Text Area - Fifth CGI Program</title>";
print "</head>";
print "<body>";
print "<h2> Entered Text Content is $text_content</h2>";
print "</body>";
print "</html>";

1;
```

Passing Drop Down Box Data to CGI Program

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

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

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

The result of this code is the following form −

Below is dropdown.cgi script to handle input given by web browser.

```
#!/usr/bin/perl

local ($buffer, @pairs, $pair, $name, $value, %FORM);
# Read in text
$ENV{'REQUEST_METHOD'} =~ tr/a-z/A-Z/;

if ($ENV{'REQUEST_METHOD'} eq "POST") {
   read(STDIN, $buffer, $ENV{'CONTENT_LENGTH'});
}else {
   $buffer = $ENV{'QUERY_STRING'};
}

# Split information into name/value pairs
@pairs = split(/&/, $buffer);

foreach $pair (@pairs) {
   ($name, $value) = split(/=/, $pair);
   $value =~ tr/+/ /;
   $value =~ s/%(..)/pack("C", hex($1))/eg;
   $FORM{$name} = $value;
}

$subject = $FORM{dropdown};

print "Content-type:text/html\r\n\r\n";
print "<html>";
print "<head>";
print "<title>Dropdown Box - Sixth CGI Program</title>";
print "</head>";
print "<body>";
print "<h2> Selected Subject is $subject</h2>";
print "</body>";
print "</html>";

1;
```

Using Cookies in CGI

HTTP protocol is a stateless protocol. But for a commercial website it is required to maintain session information among different pages. For example one user registration ends after completing many pages. But how to maintain user's session information across all the web pages.

In many situations, using cookies is the most efficient method of remembering and tracking preferences, purchases, commissions, and other information required for better visitor experience or site statistics.

How It Works

Your server sends some data to the visitor's browser in the form of a cookie. The browser may accept the cookie. If it does, it is stored as a plain text record on the visitor's hard drive. Now, when the visitor arrives at another page on your site, the cookie is available for retrieval. Once retrieved, your server knows/remembers what was stored.

Cookies are a plain text data record of 5 variable-length fields −

- **Expires** − The date the cookie will expire. If this is blank, the cookie will expire when the visitor quits the browser.
- **Domain** − The domain name of your site.
- **Path** − The path to the directory or web page that set the cookie. This may be blank if you want to retrieve the cookie from any directory or page.
- **Secure** − If this field contains the word "secure" then the cookie may only be retrieved with a secure server. If this field is blank, no such restriction exists.
- **Name=Value** − Cookies are set and retrviewed in the form of key and value pairs.

Setting up Cookies

This is very easy to send cookies to browser. These cookies will be sent along with HTTP Header. Assuming you want to set UserID and Password as cookies. So it will be done as follows −

```
#!/usr/bin/perl

print "Set-Cookie:UserID=XYZ;\n";
print "Set-Cookie:Password=XYZ123;\n";
print "Set-Cookie:Expires=Tuesday, 31-Dec-2007 23:12:40 GMT";\n";
print "Set-Cookie:Domain=www.tutorialspoint.com;\n";
print "Set-Cookie:Path=/perl;\n";
print "Content-type:text/html\r\n\r\n";
.....Rest of the HTML Content....
```

From this example you must have understood how to set cookies. We use **Set-Cookie** HTTP header to set cookies.

Here it is optional to set cookies attributes like Expires, Domain, and Path. It is notable that cookies are set before sending magic line **"Content-type:text/html\r\n\r\n**.

Retrieving Cookies

This is very easy to retrieve all the set cookies. Cookies are stored in CGI environment variable HTTP_COOKIE and they will have following form.

```
key1=value1;key2=value2;key3=value3....
```

Here is an example of how to retrieving cookies.

```
#!/usr/bin/perl
$rcvd_cookies = $ENV{'HTTP_COOKIE'};
@cookies = split /;/, $rcvd_cookies;

foreach $cookie ( @cookies ){
   ($key, $val) = split(/=/, $cookie); # splits on the first =.
   $key =~ s/^\s+//;
   $val =~ s/^\s+//;
   $key =~ s/\s+$//;
   $val =~ s/\s+$//;

   if( $key eq "UserID" ){
      $user_id = $val;
   }elsif($key eq "Password"){
      $password = $val;
   }
}

print "User ID  = $user_id\n";
print "Password = $password\n";

This will produce following result
User ID = XYZ
Password = XYZ123
```

A form to mail Program:

Create the web form

First we need to create a simple HTML form, to start with we'll keep the form simple by just asking for the users email address and comments. Here is our HTML form:

```
<html>
<head>
<title>Simple Feedback Form</title>
<style>label{display:block;}</style>
</head>
<body>

<form action="/cgi-bin/feedback_form.cgi" method="post">

<label>Email Address</label>
<input type="text" name="email_address" size="40">

<label>Your Feedback</label>
<textarea name="feedback" cols="50" rows="10"></textarea>
```

```
<input type="submit" name="send" value="Submit">

</form>

</body>
</html>
```

This form will send two parameters to our cgi script, *email_address* and *feedback*. Save this file as *feedback_form.html* and upload it to the **web** folder on your hosting.

Create the form script

We're going to use the **CGI.pm Perl module** to help make writing our cgi script easier. At the top of the script we start with the location of the perl interpretor, then we tell Perl we want to use the CGI.pm module and create a new cgi object:

```
#!/usr/bin/perl

use CGI;

my $cgi = new CGI;
```

The CGI.pm module is object-orientated, this means all of the CGI.pm functions and data are accessed through an instance of CGI.pm, in our script this instance is called *$cgi*.

Lets use our CGI object to retrieve the information from the form the user filled in. To access the form parameters we can use the CGI objects *param* function:

```
my $email_address = $cgi->param('email_address');
my $feedback = $cgi->param('feedback');
```

We store the form data in two local Perl variables, *$email_address* and *$feedback*.

Filtering user submitted data

Whenever you write a cgi script that receives data from an unknown source you should always filter the data to make sure it doesn't contain anything harmful. For example, if we don't filter the data in our form it would be quite easy for a Hacker to use our cgi script to send out spam to thousands of people. The golden rule is never trust any data you haven't created or don't control.

To filter our user data we're going to create two filter functions:

```
sub filter_email_header
{
  my $form_field = shift;
  $form_field = filter_form_data($form_field);
  $form_field  =~ s/[\0\n\r\|\!\/\<\>\^\$\%\*\&]+/ /g;

  return $form_field;
}

sub filter_form_data
```

```
{
  my $form_field = shift;
  $form_field  =~ s/From://gi;
  $form_field  =~ s/To://gi;
  $form_field  =~ s/BCC://gi;
  $form_field  =~ s/CC://gi;
  $form_field  =~ s/Subject://gi;
  $form_field  =~ s/Content-Type://gi;

  return $form_field;
}
```

The first filter function removes special characters which could be used to trick our script into sending spam and is applied to the *$email_address* data.The second filter function removes common email headers from the data the user submitted and can be applied to both *$email_address* and *$feedback*. We'll place the two functions at the bottom of our script.

Now we'll call the two filter functions to clean up our user submitted data:

```
$email_address  = filter_email_header($email_address);
$feedback = filter_form_data($feedback);
```

Emailing the feedback

Once we have the filtered data we need to email it back to you. Our web hosting servers run a local mail server (sendmail) that your cgi script can use to send email. To send the email our cgi script opens a communication channel to the sendmail program using the pipe (|) symbol. It then prints all the information necessary to send an email across that channel:

```
open ( MAIL, "| /usr/lib/sendmail -t" );
print MAIL "From: $email_address\n";
print MAIL "To: you\@domain.com\n";
print MAIL "Subject: Feedback Form Submission\n\n";
print MAIL "$feedback\n";
print MAIL "\n.\n";
close ( MAIL );
```

Make sure you set your email address on line 3, you'll need to escape the @ symbol by putting a backslash (\) before it because Perl uses the @ symbol to denote a special type of variable. The two newline characters (\n\n) at the end of line 4 are used to mark the end of the email headers ready for the content. The \n.\n on line 6 prints a dot (.) on its own line to tell sendmail that we've finished printing the message.

Thank the user for their feedback

Finally, when a user submits your form, let's show a page thanking them for their feedback:

```
print $cgi->header(-type => 'text/html');

print <<HTML_PAGE;
 <html>
<head>
<title>Thank You</title>
</head>
<body>
```

```
<h1>Thank You</h1>
<p>Thank you for your feedback.</p>
</body>
</html>
HTML_PAGE
```

The first thing we do is print back the HTTP header, using the CGI header function, to let the web browser know what type of content to expect. Then we print out the HTML page.

The final script

This example script shows a very basic way to get form contents emailed to you, it doesn't however have the refinements of a professional script, e.g. input validation. Below is the finished script. We've added some comments (lines beginning with #) to help make it clearer.

```
#!/usr/bin/perl

use CGI;

# Create a CGI.pm object
my $cgi = new CGI;

# Get the form data
my $email_address = $cgi->param('email_address');
my $feedback = $cgi->param('feedback');

# Filter the form data
$email_address  = filter_email_header($email_address);
$feedback = filter_form_data($feedback);

# Email the form data
open ( MAIL, "| /usr/lib/sendmail -t" );
print MAIL "From: $email_address\n";
print MAIL "To: you\@domain.com\n";
print MAIL "Subject: Feedback Form Submission\n\n";
print MAIL "$feedback\n";
print MAIL "\n.\n";
close ( MAIL );

# Print the HTTP header
print $cgi->header(-type => 'text/html');

# Print the HTML thank you page
print <<HTML_PAGE;
 <html>
<head>
<title>Thank You</title>
</head>
<body>
<h1>Thank You</h1>
<p>Thank you for your feedback.</p>
</body>
</html>
HTML_PAGE

# Functions to filter the form data
```

```
sub filter_email_header
{
  my $form_field = shift;
  $form_field = filter_form_data($form_field);
  $form_field  =~ s/[\0\n\r\|\!\/\<\>\^\$\%\*\&]+/ /g;

  return $form_field ;
}

sub filter_form_data
{
  my $form_field = shift;
  $form_field  =~ s/From://gi;
  $form_field  =~ s/To://gi;
  $form_field  =~ s/BCC://gi;
  $form_field  =~ s/CC://gi;
  $form_field  =~ s/Subject://gi;
  $form_field  =~ s/Content-Type://gi;

  return $form_field ;
}
```

Simple Page Search:

```
#!/usr/bin/env perl

use strict;
use warnings;

use HTML::Parser;
use LWP::UserAgent;

my $ua = LWP::UserAgent->new;
my $response = $ua->get('http://search.cpan.org/');
if ( !$response->is_success ) {
    print "No matches\n";
    exit 1;
}

my $parser = HTML::Parser->new( 'text_h' => [ \&text_handler, 'dtext' ] );
$parser->parse( $response->decoded_content );

sub text_handler {
    chomp( my $text = shift );

    if ( $text =~ /language/i ) {
        print "Matched: $text\n";
    }
}
```

Web Technologies
UNIT-VI

Ruby is a pure object-oriented programming language. It was created in 1993 by Yukihiro Matsumoto of Japan.
You can find the name Yukihiro Matsumoto on the Ruby mailing list at www.ruby-lang.org. Matsumoto is also known as Matz in the Ruby community.
Ruby is "A Programmer's Best Friend".
Ruby has features that are similar to those of Smalltalk, Perl, and Python. Perl, Python, and Smalltalk are scripting languages. Smalltalk is a true object-oriented language. Ruby, like Smalltalk, is a perfect object-oriented language. Using Ruby syntax is much easier than using Smalltalk syntax.

Features of Ruby

- Ruby is an open-source and is freely available on the Web, but it is subject to a license.
- Ruby is a general-purpose, interpreted programming language.
- Ruby is a true object-oriented programming language.
- Ruby is a server-side scripting language similar to Python and PERL.
- Ruby can be used to write Common Gateway Interface (CGI) scripts.
- Ruby can be embedded into Hypertext Markup Language (HTML).
- Ruby has a clean and easy syntax that allows a new developer to learn Ruby very quickly and easily.
- Ruby has similar syntax to that of many programming languages such as C++ and Perl.
- Ruby is very much scalable and big programs written in Ruby are easily maintainable.
- Ruby can be used for developing Internet and intranet applications.
- Ruby can be installed in Windows and POSIX environments.
- Ruby support many GUI tools such as Tcl/Tk, GTK, and OpenGL.
- Ruby can easily be connected to DB2, MySQL, Oracle, and Sybase.
- Ruby has a rich set of built-in functions, which can be used directly into Ruby scripts.

Tools You Will Need

For performing the examples discussed in this tutorial, you will need a latest computer like Intel Core i3 or i5 with a minimum of 2GB of RAM (4GB of RAM recommended). You also will need the following software:

- Linux or Windows 95/98/2000/NT or Windows 7 operating system
- Apache 1.3.19-5 Web server
- Internet Explorer 5.0 or above Web browser
- Ruby 1.8.5

This tutorial will provide the necessary skills to create GUI, networking, and Web applications using Ruby. It also will talk about extending and embedding Ruby applications.

Popular Ruby Editors:

To write your Ruby programs, you will need an editor:

- If you are working on Windows machine, then you can use any simple text editor like Notepad or Edit plus.
- VIM (Vi IMproved) is very simple text editor. This is available on almost all Unix machines and now Windows as well. Otherwise, your can use your favorite vi editor to write Ruby programs.
- RubyWin is a Ruby Integrated Development Environment (IDE) for Windows.
- Ruby Development Environment (RDE) is also very good IDE for windows users.

Interactive Ruby (IRb):

Interactive Ruby (IRb) provides a shell for experimentation. Within the IRb shell, you can immediately view expression results, line by line.

This tool comes along with Ruby installation so you have nothing to do extra to have IRb working.

Just type **irb** at your command prompt and an Interactive Ruby Session will start as given below:

```
$irb
irb 0.6.1(99/09/16)
irb(main):001:0> def hello
irb(main):002:1> out = "Hello World"
irb(main):003:1> puts out
irb(main):004:1> end
nil
irb(main):005:0> hello
Hello World
nil
irb(main):006:0>
```

Ruby Syntax:

Let us write a simple program in ruby. All ruby files will have extension **.rb**. So, put the following source code in a test.rb file.

```
#!/usr/bin/ruby -w

puts "Hello, Ruby!";
```

Here, I assumed that you have Ruby interpreter available in /usr/bin directory. Now, try to run this program as follows:

```
$ ruby test.rb
```

This will produce the following result:

```
Hello, Ruby!
```

Whitespace in Ruby Program:

Whitespace characters such as spaces and tabs are generally ignored in Ruby code, except when they appear in strings. Sometimes, however, they are used to interpret ambiguous statements. Interpretations of this sort produce warnings when the -w option is enabled.

Example:

```
a + b is interpreted as a+b ( Here a is a local variable)
a  +b is interpreted as a(+b) ( Here a is a method call)
```

Line Endings in Ruby Program:

Ruby interprets semicolons and newline characters as the ending of a statement. However, if Ruby encounters operators, such as +, -, or backslash at the end of a line, they indicate the continuation of a statement.

Ruby Identifiers:

Identifiers are names of variables, constants, and methods. Ruby identifiers are case sensitive. It mean Ram and RAM are two different idendifiers in Ruby.

Ruby identifier names may consist of alphanumeric characters and the underscore character (_).

Reserved Words:

The following list shows the reserved words in Ruby. These reserved words may not be used as constant or variable names. They can, however, be used as method names.

BEGIN do next then

TutorialsDuniya.com

Download FREE Computer Science Notes, Programs, Projects, Books PDF for any university student of BCA, MCA, B.Sc, B.Tech CSE, M.Sc, M.Tech at https://www.tutorialsduniya.com

- Algorithms Notes
- Artificial Intelligence
- Android Programming
- C & C++ Programming
- Combinatorial Optimization
- Computer Graphics
- Computer Networks
- Computer System Architecture
- DBMS & SQL Notes
- Data Analysis & Visualization
- Data Mining
- Data Science
- Data Structures
- Deep Learning
- Digital Image Processing
- Discrete Mathematics
- Information Security
- Internet Technologies
- Java Programming
- JavaScript & jQuery
- Machine Learning
- Microprocessor
- Operating System
- Operational Research
- PHP Notes
- Python Programming
- R Programming
- Software Engineering
- System Programming
- Theory of Computation
- Unix Network Programming
- Web Design & Development

Please Share these Notes with your Friends as well

END	else	nil	true
alias	elsif	not	undef
and	end	or	unless
begin	ensure	redo	until
break	false	rescue	when
case	for	retry	while
class	if	return	yield
def	in	self	_FILE____
defined?	module	super	__LINE__

Here Document in Ruby:

"Here Document" refers to build strings from multiple lines. Following a << you can specify a string or an identifier to terminate the string literal, and all lines following the current line up to the terminator are the value of the string.

If the terminator is quoted, the type of quotes determines the type of the line-oriented string literal. Notice there must be no space between << and the terminator.

Here are different examples:

```
#!/usr/bin/ruby -w

print <<EOF
    This is the first way of creating
    here document ie. multiple line string.
EOF

print <<"EOF";                # same as above
    This is the second way of creating
    here document ie. multiple line string.
EOF

print <<`EOC`                 # execute commands
        echo hi there
        echo lo there
EOC

print <<"foo", <<"bar"  # you can stack them
        I said foo.
foo
        I said bar.
bar
```

This will produce the following result:

```
    This is the first way of creating
    her document ie. multiple line string.
    This is the second way of creating
    her document ie. multiple line string.
hi there
lo there
        I said foo.
        I said bar.
```

Ruby *BEGIN* Statement

Syntax:

```
BEGIN {
   code
}
```

Declares *code* to be called before the program is run.

Example:

```
#!/usr/bin/ruby

puts "This is main Ruby Program"

BEGIN {
   puts "Initializing Ruby Program"
}
```

This will produce the following result:

```
Initializing Ruby Program
This is main Ruby Program
```

Ruby *END* Statement

Syntax:

```
END {
   code
}
```

Declares *code* to be called at the end of the program.

Example:

```
#!/usr/bin/ruby

puts "This is main Ruby Program"

END {
   puts "Terminating Ruby Program"
}
BEGIN {
   puts "Initializing Ruby Program"
}
```

This will produce the following result:

```
Initializing Ruby Program
This is main Ruby Program
Terminating Ruby Program
```

Ruby Comments:

A comment hides a line, part of a line, or several lines from the Ruby interpreter. You can use the hash character (#) at the beginning of a line:

```
# I am a comment. Just ignore me.
```

Or, a comment may be on the same line after a statement or expression:

```
name = "Madisetti" # This is again comment
```

You can comment multiple lines as follows:

```
# This is a comment.
# This is a comment, too.
# This is a comment, too.
# I said that already.
```

Here is another form. This block comment conceals several lines from the interpreter with =begin/=end:

```
=begin
This is a  comment.
This is a comment, too.
This is a comment, too.
I said that already.
=end
```

Ruby Classes:

Ruby is a perfect Object Oriented Programming Language. The features of the object-oriented programming language include:

- Data Encapsulation:
- Data Abstraction:

- Polymorphism:
- Inheritance:

An object-oriented program involves classes and objects. A class is the blueprint from which individual objects are created. In object-oriented terms, we say that your *bicycle* is an instance of the *class of objects* known as bicycles.

Take the example of any vehicle. It comprises wheels, horsepower, and fuel or gas tank capacity. These characteristics form the data members of the class Vehicle. You can differentiate one vehicle from the other with the help of these characteristics.

A vehicle can also have certain functions, such as halting, driving, and speeding. Even these functions form the data members of the class Vehicle. You can, therefore, define a class as a combination of characteristics and functions.

A class Vehicle can be defined in Java as follows :

```
Class Vehicle
{
   Number no_of_wheels
   Number horsepower
   Characters type_of_tank
   Number Capacity
   Function speeding
   {
   }
   Function driving
   {
   }
   Function halting
   {
   }
}
```

By assigning different values to these data members, you can form several instances of the class Vehicle. For example, an airplane has three wheels, horsepower of 1,000, fuel as the type of tank, and a capacity of 100 liters. In the same way, a car has four wheels, horsepower of 200, gas as the type of tank, and a capacity of 25 litres.

Defining a Class in Ruby:

To implement object-oriented programming by using Ruby, you need to first learn how to create objects and classes in Ruby.

A class in Ruby always starts with the keyword *class* followed by the name of the class. The name should always be in initial capitals. The class *Customer* can be displayed as:

```
class Customer
end
```

You terminate a class by using the keyword *end*. All the data members in the *class* are between the class definition and the *end* keyword.

Variables in a Ruby Class:

Ruby provides four types of variables:

- **Local Variables:** Local variables are the variables that are defined in a method. Local variables are not available outside the method. You will see more details about method in subsequent chapter. Local variables begin with a lowercase letter or _.
- **Instance Variables:** Instance variables are available across methods for any particular instance or object. That means that instance variables change from object to object. Instance variables are preceded by the at sign (@) followed by the variable name.
- **Class Variables:** Class variables are available across different objects. A class variable belongs to the class and is a characteristic of a class. They are preceded by the sign @@ and are followed by the variable name.

- **Global Variables:** Class variables are not available across classes. If you want to have a single variable, which is available across classes, you need to define a global variable. The global variables are always preceded by the dollar sign ($).

Example:

Using the class variable @@no_of_customers, you can determine the number of objects that are being created. This enables in deriving the number of customers.

```
class Customer
   @@no_of_customers=0
end
```

Creating Objects in Ruby using *new* Method:

Objects are instances of the class. You will now learn how to create objects of a class in Ruby. You can create objects in Ruby by using the method *new* of the class.

The method *new* is a unique type of method, which is predefined in the Ruby library. The new method belongs to the *class* methods.

Here is the example to create two objects cust1 and cust2 of the class Customer:

```
cust1 = Customer. new
cust2 = Customer. new
```

Here, cust1 and cust2 are the names of two objects. You write the object name followed by the equal to sign (=) after which the class name will follow. Then, the dot operator and the keyword *new* will follow.

Custom Method to create Ruby Objects :

You can pass parameters to method *new* and those parameters can be used to initialize class variables.

When you plan to declare the *new* method with parameters, you need to declare the method *initialize* at the time of the class creation.

The *initialize* method is a special type of method, which will be executed when the *new* method of the class is called with parameters.

Here is the example to create initialize method:

```
class Customer
   @@no_of_customers=0
   def initialize(id, name, addr)
      @cust_id=id
      @cust_name=name
      @cust_addr=addr
   end
end
```

In this example, you declare the *initialize* method with **id, name**, and **addr** as local variables. Here, *def* and *end* are used to define a Ruby method *initialize*. You will learn more about methods in subsequent chapters.

In the *initialize* method, you pass on the values of these local variables to the instance variables @cust_id, @cust_name, and @cust_addr. Here local variables hold the values that are passed along with the new method.

Now, you can create objects as follows:

```
cust1=Customer.new("1", "John", "Wisdom Apartments, Ludhiya")
cust2=Customer.new("2", "Poul", "New Empire road, Khandala")
```

Member Functions in Ruby Class:

In Ruby, functions are called methods. Each method in a *class* starts with the keyword *def* followed by the method name.

The method name always preferred in **lowercase letters**. You end a method in Ruby by using the keyword *end*.

Here is the example to define a Ruby method:

```
class Sample
   def function
      statement 1
```

```
      statement 2
   end
end
```

Here, *statement 1* and *statement 2* are part of the body of the method *function* inside the class Sample. These statments could be any valid Ruby statement. For example we can put a method *puts* to print *Hello Ruby* as follows:

```
class Sample
   def hello
      puts "Hello Ruby!"
   end
end
```

Now in the following example, create one object of Sample class and call *hello* method and see the result:

```
#!/usr/bin/ruby

class Sample
   def hello
      puts "Hello Ruby!"
   end
end

# Now using above class to create objects
object = Sample. new
object.hello
```

This will produce the following result:

```
Hello Ruby!
```

Ruby Variables

Variables are the memory locations which hold any data to be used by any program.
There are five types of variables supported by Ruby. You already have gone through a small description of these variables in previous chapter as well. These five types of variables are explained in this chapter.

Ruby Global Variables:

Global variables begin with $. Uninitialized global variables have the value *nil* and produce warnings with the -w option.
Assignment to global variables alters global status. It is not recommended to use global variables. They make programs cryptic.
Here is an example showing usage of global variable.

```
#!/usr/bin/ruby

$global_variable = 10
class Class1
  def print_global
     puts "Global variable in Class1 is #$global_variable"
  end
end
class Class2
  def print_global
     puts "Global variable in Class2 is #$global_variable"
  end
end

class1obj = Class1.new
class1obj.print_global
class2obj = Class2.new
class2obj.print_global
```

Here $global_variable is a global variable. This will produce the following result:

NOTE: In Ruby you CAN access value of any variable or constant by putting a hash (#) character just before that variable or constant.

```
Global variable in Class1 is 10
Global variable in Class2 is 10
```

Ruby Instance Variables:

Instance variables begin with @. Uninitialized instance variables have the value *nil* and produce warnings with the -w option.

Here is an example showing usage of Instance Variables.

```
#!/usr/bin/ruby

class Customer
   def initialize(id, name, addr)
      @cust_id=id
      @cust_name=name
      @cust_addr=addr
   end
   def display_details()
      puts "Customer id #@cust_id"
      puts "Customer name #@cust_name"
      puts "Customer address #@cust_addr"
    end
end

# Create Objects
cust1=Customer.new("1", "John", "Wisdom Apartments, Ludhiya")
cust2=Customer.new("2", "Poul", "New Empire road, Khandala")

# Call Methods
cust1.display_details()
cust2.display_details()
```

Here, @cust_id, @cust_name and @cust_addr are instance variables. This will produce the following result:

```
Customer id 1
Customer name John
Customer address Wisdom Apartments, Ludhiya
Customer id 2
Customer name Poul
Customer address New Empire road, Khandala
```

Ruby Class Variables:

Class variables begin with @@ and must be initialized before they can be used in method definitions.

Referencing an uninitialized class variable produces an error. Class variables are shared among descendants of the class or module in which the class variables are defined.

Overriding class variables produce warnings with the -w option.

Here is an example showing usage of class variable:

```
#!/usr/bin/ruby

class Customer
   @@no_of_customers=0
   def initialize(id, name, addr)
      @cust_id=id
      @cust_name=name
      @cust_addr=addr
      @@no_of_customers += 1
   end
   def display_details()
      puts "Customer id #@cust_id"
      puts "Customer name #@cust_name"
      puts "Customer address #@cust_addr"
    end
```

```
    def total_no_of_customers()
       puts "Total number of customers: #@@no_of_customers"
    end
end

# Create Objects
cust1=Customer.new("1", "John", "Wisdom Apartments, Ludhiya")
cust2=Customer.new("2", "Poul", "New Empire road, Khandala")

# Call Methods
cust1.total_no_of_customers()
cust2.total_no_of_customers()
```

Here @@no_of_customers is a class variable. This will produce the following result:

```
Total number of customers: 1
Total number of customers: 2
```

Ruby Local Variables:

Local variables begin with a lowercase letter or _. The scope of a local variable ranges from class, module, def, or do to the corresponding end or from a block's opening brace to its close brace {}.

When an uninitialized local variable is referenced, it is interpreted as a call to a method that has no arguments.

Assignment to uninitialized local variables also serves as variable declaration. The variables start to exist until the end of the current scope is reached. The lifetime of local variables is determined when Ruby parses the program.

In the above example local variables are id, name and addr.

Ruby Constants:

Constants begin with an uppercase letter. Constants defined within a class or module can be accessed from within that class or module, and those defined outside a class or module can be accessed globally.

Constants may not be defined within methods. Referencing an uninitialized constant produces an error. Making an assignment to a constant that is already initialized produces a warning.

```
#!/usr/bin/ruby

class Example
   VAR1 = 100
   VAR2 = 200
   def show
       puts "Value of first Constant is #{VAR1}"
       puts "Value of second Constant is #{VAR2}"
   end
end

# Create Objects
object=Example.new()
object.show
```

Here VAR1 and VAR2 are constant. This will produce the following result:

```
Value of first Constant is 100
Value of second Constant is 200
```

Ruby Pseudo-Variables:

They are special variables that have the appearance of local variables but behave like constants. You can not assign any value to these variables.

- **self:** The receiver object of the current method.
- **true:** Value representing true.
- **false:** Value representing false.
- **nil:** Value representing undefined.
- **__FILE :** The name of the current source file.
- **__LINE :** The current line number in the source file.

Ruby Basic Literals:

The rules Ruby uses for literals are simple and intuitive. This section explains all basic Ruby Literals.

Integer Numbers:

Ruby supports integer numbers. An integer number can range from -2^{30} to 2^{30-1} or -2^{62} to 2^{62-1}. Integers with-in this range are objects of class *Fixnum* and integers outside this range are stored in objects of class *Bignum*.

You write integers using an optional leading sign, an optional base indicator (0 for octal, 0x for hex, or 0b for binary), followed by a string of digits in the appropriate base. Underscore characters are ignored in the digit string.

You can also get the integer value corresponding to an ASCII character or escape sequence by preceding it with a question mark.

Example:

```
123                  # Fixnum decimal
1_234                # Fixnum decimal with underline
-500                 # Negative Fixnum
0377                 # octal
0xff                 # hexadecimal
0b1011               # binary
?a                   # character code for 'a'
?\n                  # code for a newline (0x0a)
12345678901234567890 # Bignum
```

NOTE: Class and Objects are explained in a separate chapter of this tutorial.

Floating Numbers:

Ruby supports integer numbers. They are also numbers but with decimals. Floating-point numbers are objects of class *Float* and can be any of the following:

Example:

```
123.4                # floating point value
1.0e6                # scientific notation
4E20                 # dot not required
4e+20                # sign before exponential
```

String Literals:

Ruby strings are simply sequences of 8-bit bytes and they are objects of class String. Double-quoted strings allow substitution and backslash notation but single-quoted strings don't allow substitution and allow backslash notation only for \\ and \'

Example:

```
#!/usr/bin/ruby -w

puts 'escape using "\\"';
puts 'That\'s right';
```

This will produce the following result:

```
escape using "\"
That's right
```

You can substitute the value of any Ruby expression into a string using the sequence **#{ expr }**. Here, expr could be any ruby expression.

```
#!/usr/bin/ruby -w

puts "Multiplication Value : #{24*60*60}";
```

This will produce the following result:

```
Multiplication Value : 86400
```

Backslash Notations:

Following is the list of Backslash notations supported by Ruby:

Notation	Character represented
\n	Newline (0x0a)

\r	Carriage return (0x0d)
\f	Formfeed (0x0c)
\b	Backspace (0x08)
\a	Bell (0x07)
\e	Escape (0x1b)
\s	Space (0x20)
\nnn	Octal notation (n being 0-7)
\xnn	Hexadecimal notation (n being 0-9, a-f, or A-F)
\cx, \C-x	Control-x
\M-x	Meta-x (c | 0x80)
\M-\C-x	Meta-Control-x
\x	Character x

Ruby Ranges:

A Range represents an interval.a set of values with a start and an end. Ranges may be constructed using the s..e and s...e literals, or with Range.new.
Ranges constructed using .. run from the start to the end inclusively. Those created using ... exclude the end value. When used as an iterator, ranges return each value in the sequence.
A range (1..5) means it includes 1, 2, 3, 4, 5 values and a range (1...5) means it includes 1, 2, 3, 4 values.

Example:

```
#!/usr/bin/ruby

(10..15).each do |n|
   print n, ' '
end
```

This will produce the following result:

```
10 11 12 13 14 15
```

Ruby Conditional statements:

Ruby *if...else* Statement:

Syntax:

```
if conditional [then]
         code...
[elsif conditional [then]
         code...]...
[else
         code...]
end
```

if expressions are used for conditional execution. The values *false* and *nil* are false, and everything else are true. Notice Ruby uses elsif, not else if nor elif.
Executes *code* if the *conditional* is true. If the *conditional* is not true, *code* specified in the else clause is executed.
An if expression's *conditional* is separated from code by the reserved word *then*, a newline, or a semicolon.

Example:

```
#!/usr/bin/ruby

x=1
if x > 2
```

```
   puts "x is greater than 2"
elsif x <= 2 and x!=0
   puts "x is 1"
else
   puts "I can't guess the number"
end
x is 1
```

Ruby *if* modifier:

Syntax:

```
code if condition
```

Executes *code* if the *conditional* is true.

Example:

```
#!/usr/bin/ruby

$debug=1
print "debug\n" if $debug
```

This will produce the following result:

```
debug
```

Ruby *unless* Statement:

Syntax:

```
unless conditional [then]
   code
[else
   code ]
end
```

Executes *code* if *conditional* is false. If the *conditional* is true, code specified in the else clause is executed.

Example:

```
#!/usr/bin/ruby

x=1
unless x>2
   puts "x is less than 2"
 else
  puts "x is greater than 2"
end
```

This will produce the following result:

```
x is less than 2
```

Ruby *unless* modifier:

Syntax:

```
code unless conditional
```

Executes *code* if *conditional* is false.

Example:

```
#!/usr/bin/ruby

$var =  1
print "1 -- Value is set\n" if $var
print "2 -- Value is set\n" unless $var

$var = false
print "3 -- Value is set\n" unless $var
```

This will produce the following result:

```
1 -- Value is set
3 -- Value is set
```

Ruby *case* Statement

Syntax:

```
case expression
[when expression [, expression ...] [then]
   code ]...
[else
   code ]
end
```

Compares the *expression* specified by case and that specified by when using the === operator and executes the *code* of the when clause that matches.

The *expression* specified by the when clause is evaluated as the left operand. If no when clauses match, *case* executes the code of the *else* clause.

A when statement's expression is separated from code by the reserved word then, a newline, or a semicolon.

Thus:

```
case expr0
when expr1, expr2
   stmt1
when expr3, expr4
   stmt2
else
   stmt3
end
```

is basically similar to the following:

```
_tmp = expr0
if expr1 === _tmp || expr2 === _tmp
   stmt1
elsif expr3 === _tmp || expr4 === _tmp
   stmt2
else
   stmt3
end
```

Example:

```
#!/usr/bin/ruby

$age =  5
case $age
when 0 .. 2
    puts "baby"
when 3 .. 6
    puts "little child"
when 7 .. 12
    puts "child"
when 13 .. 18
    puts "youth"
else
    puts "adult"
end
```

This will produce the following result:

```
little child
```

Ruby Looping statements:

Loops in Ruby are used to execute the same block of code a specified number of times. This chapter details all the loop statements supported by Ruby.

Ruby *while* Statement:

Syntax:

```
while conditional [do]
   code
end
```

Executes *code* while *conditional* is true. A *while* loop's *conditional* is separated from *code* by the reserved word do, a newline, backslash \, or a semicolon ;.

Example:

```
#!/usr/bin/ruby

$i = 0
$num = 5

while $i < $num  do
   puts("Inside the loop i = #$i" )
   $i +=1
end
```

This will produce the following result:

```
Inside the loop i = 0
Inside the loop i = 1
Inside the loop i = 2
Inside the loop i = 3
Inside the loop i = 4
```

Ruby *while* modifier:

Syntax:

```
code while condition

OR

begin
  code
end while conditional
```

Executes *code* while *conditional* is true.

If a *while* modifier follows a *begin* statement with no *rescue* or ensure clauses, *code* is executed once before conditional is evaluated.

Example:

```
#!/usr/bin/ruby

$i = 0
$num = 5
begin
   puts("Inside the loop i = #$i" )
   $i +=1
end while $i < $num
```

This will produce the following result:

```
Inside the loop i = 0
Inside the loop i = 1
Inside the loop i = 2
Inside the loop i = 3
Inside the loop i = 4
```

Ruby *until* Statement:

```
until conditional [do]
   code
end
```

Executes *code* while *conditional* is false. An *until* statement's conditional is separated from *code* by the reserved word *do*, a newline, or a semicolon.

Example:

```
#!/usr/bin/ruby

$i = 0
$num = 5

until $i > $num  do
   puts("Inside the loop i = #$i" )
   $i +=1;
end
```

This will produce the following result:

```
Inside the loop i = 0
Inside the loop i = 1
Inside the loop i = 2
Inside the loop i = 3
Inside the loop i = 4
Inside the loop i = 5
```

Ruby *until* modifier:

Syntax:

```
code until conditional

OR

begin
   code
end until conditional
```

Executes *code* while *conditional* is false.

If an *until* modifier follows a *begin* statement with no *rescue* or ensure clauses, *code* is executed once before *conditional* is evaluated.

Example:

```
#!/usr/bin/ruby

$i = 0
$num = 5
begin
   puts("Inside the loop i = #$i" )
   $i +=1;
end until $i > $num
```

This will produce the following result:

```
Inside the loop i = 0
Inside the loop i = 1
Inside the loop i = 2
Inside the loop i = 3
Inside the loop i = 4
Inside the loop i = 5
```

Ruby *for* Statement:

Syntax:

```
for variable [, variable ...] in expression [do]
   code
end
```

Executes *code* once for each element in *expression*.

Example:

```
#!/usr/bin/ruby

for i in 0..5
   puts "Value of local variable is #{i}"
end
```

Here, we have defined the range 0..5. The statement for i in 0..5 will allow i to take values in the range from 0 to 5 (including 5). This will produce the following result:

```
Value of local variable is 0
Value of local variable is 1
Value of local variable is 2
Value of local variable is 3
Value of local variable is 4
Value of local variable is 5
```

A *for...in* loop is almost exactly equivalent to:

```
(expression).each do |variable[, variable...]| code end
```

except that a for loop doesn't create a new scope for local variables. A for loop's *expression* is separated from *code* by the reserved word do, a newline, or a semicolon.

Example:

```
#!/usr/bin/ruby

(0..5).each do |i|
   puts "Value of local variable is #{i}"
end
```

This will produce the following result:

```
Value of local variable is 0
Value of local variable is 1
Value of local variable is 2
Value of local variable is 3
Value of local variable is 4
Value of local variable is 5
```

Ruby *break* Statement:

Syntax:

```
break
```

Terminates the most internal loop. Terminates a method with an associated block if called within the block (with the method returning nil).

Example:

```
#!/usr/bin/ruby

for i in 0..5
   if i > 2 then
      break
   end
   puts "Value of local variable is #{i}"
end
```

This will produce the following result:

```
Value of local variable is 0
Value of local variable is 1
Value of local variable is 2
```

Ruby *next* Statement:

Syntax:

```
next
```

Jumps to next iteration of the most internal loop. Terminates execution of a block if called within a block (with *yield* or call returning nil).

Example:

```
#!/usr/bin/ruby

for i in 0..5
   if i < 2 then
      next
   end
   puts "Value of local variable is #{i}"
end
```

This will produce the following result:

```
Value of local variable is 2
Value of local variable is 3
Value of local variable is 4
Value of local variable is 5
```

Ruby *redo* Statement:

Syntax:

```
redo
```

Restarts this iteration of the most internal loop, without checking loop condition. Restarts *yield* or *call* if called within a block.

Example:

```
#!/usr/bin/ruby

for i in 0..5
   if i < 2 then
      puts "Value of local variable is #{i}"
      redo
   end
end
```

This will produce the following result and will go in an infinite loop:

```
Value of local variable is 0
Value of local variable is 0
.............................
```

Ruby *retry* Statement:

Syntax:

```
retry
```

If *retry* appears in rescue clause of begin expression, restart from the beginning of the 1begin body.

```
begin
   do_something # exception raised
rescue
   # handles error
   retry  # restart from beginning
end
```

If retry appears in the iterator, the block, or the body of the for expression, restarts the invocation of the iterator call. Arguments to the iterator is re-evaluated.

```
for i in 1..5
   retry if some_condition # restart from i == 1
end
```

Example:

```
#!/usr/bin/ruby

for i in 1..5
   retry if  i > 2
   puts "Value of local variable is #{i}"
end
```

This will produce the following result and will go in an infinite loop:

```
Value of local variable is 1
Value of local variable is 2
Value of local variable is 1
Value of local variable is 2
Value of local variable is 1
Value of local variable is 2
.............................
```

Ruby Methods

Ruby methods are very similar to functions in any other programming language. Ruby methods are used to bundle one or more repeatable statements into a single unit.
Method names should begin with a lowercase letter. If you begin a method name with an uppercase letter, Ruby might think that it is a constant and hence can parse the call incorrectly.
Methods should be defined before calling them, otherwise Ruby will raise an exception for undefined method invoking.

Syntax:

```
def method_name [( [arg [= default]]...[, * arg [, &expr ]])]
   expr..
end
```

So you can define a simple method as follows:

```
def method_name
   expr..
end
```

You can represent a method that accepts parameters like this:

```
def method_name (var1, var2)
   expr..
end
```

You can set default values for the parameters which will be used if method is called without passing required parameters:

```
def method_name (var1=value1, var2=value2)
   expr..
end
```

Whenever you call the simple method, you write only the method name as follows:

```
method_name
```

However, when you call a method with parameters, you write the method name along with the parameters, such as:

```
method_name 25, 30
```

The most important drawback to using methods with parameters is that you need to remember the number of parameters whenever you call such methods. For example, if a method accepts three parameters and you pass only two, then Ruby displays an error.

Example:

```
#!/usr/bin/ruby

def test(a1="Ruby", a2="Perl")
   puts "The programming language is #{a1}"
   puts "The programming language is #{a2}"
end
test "C", "C++"
test
```

This will produce the following result:

```
The programming language is C
The programming language is C++
The programming language is Ruby
The programming language is Perl
```

Return Values from Methods:

Every method in Ruby returns a value by default. This returned value will be the value of the last statement. For example:

```
def test
   i = 100
   j = 10
```

```
   k = 0
end
```

This method, when called, will return the last declared variable k.

Ruby *return* Statement:

The *return* statement in ruby is used to return one or more values from a Ruby Method.

Syntax:

```
return [expr[`,' expr...]]
```

If more than two expressions are given, the array containing these values will be the return value. If no expression given, nil will be the return value.

Example:

```
return

OR

return 12

OR

return 1,2,3
```

Have a look at this example:

```
#!/usr/bin/ruby

def test
   i = 100
   j = 200
   k = 300
return i, j, k
end
var = test
puts var
```

This will produce the following result:

```
100
200
300
```

Class Methods:

When a method is defined outside of the class definition, the method is marked as *private* by default. On the other hand, the methods defined in the class definition are marked as public by default. The default visibility and the *private* mark of the methods can be changed by *public* or *private* of the Module.

Whenever you want to access a method of a class, you first need to instantiate the class. Then, using the object, you can access any member of the class.

Ruby gives you a way to access a method without instantiating a class. Let us see how a class method is declared and accessed:

```
class Accounts
   def reading_charge
   end
   def Accounts.return_date
   end
end
```

See how the method return_date is declared. It is declared with the class name followed by a period, which is followed by the name of the method. You can access this class method directly as follows:

```
Accounts.return_date
```

To access this method, you need not create objects of the class Accounts.

Ruby *alias* Statement:

This gives alias to methods or global variables. Aliases can not be defined within the method body. The alias of the method keep the current definition of the method, even when methods are overridden.
Making aliases for the numbered global variables ($1, $2,...) is prohibited. Overriding the built-in global variables may cause serious problems.

Syntax:

```
alias method-name method-name
alias global-variable-name global-variable-name
```

Example:

```
alias foo bar
alias $MATCH $&
```

Here we have defined foo alias for bar and $MATCH is an alias for $&

Ruby *undef* Statement:

This cancels the method definition. An *undef* can not appear in the method body.
By using *undef* and *alias*, the interface of the class can be modified independently from the superclass, but notice it may be broke programs by the internal method call to self.

Syntax:

```
undef method-name
```

Example:

To undefine a method called *bar* do the following:

```
undef bar
```

Ruby Arrays

Ruby arrays are ordered, integer-indexed collections of any object. Each element in an array is associated with and referred to by an index.
Array indexing starts at 0, as in C or Java. A negative index is assumed relative to the end of the array --- that is, an index of -1 indicates the last element of the array, -2 is the next to last element in the array, and so on.
Ruby arrays can hold objects such as String, Integer, Fixnum, Hash, Symbol, even other Array objects. Ruby arrays are not as rigid as arrays in other languages. Ruby arrays grow automatically while adding elements to them.

Creating Arrays:

There are many ways to create or initialize an array. One way is with the *new* class method:

```
names = Array.new
```

You can set the size of an array at the time of creating array:

```
names = Array.new(20)
```

The array *names* now has a size or length of 20 elements. You can return the size of an array with either the size or length methods:

```
#!/usr/bin/ruby

names = Array.new(20)
puts names.size   # This returns 20
puts names.length # This also returns 20
```

This will produce the following result:

```
20
20
```

You can assign a value to each element in the array as follows:

```
#!/usr/bin/ruby
```

```
names = Array.new(4, "mac")

puts "#{names}"
```

This will produce the following result:

```
macmacmacmac
```

You can also use a block with new, populating each element with what the block evaluates to:

```
#!/usr/bin/ruby

nums = Array.new(10) { |e| e = e * 2 }

puts "#{nums}"
```

This will produce the following result:

```
024681012141618
```

There is another method of Array, []. It works like this:

```
nums = Array.[](1, 2, 3, 4,5)
```

One more form of array creation is as follows :

```
nums = Array[1, 2, 3, 4,5]
```

The *Kernel* module available in core Ruby has an Array method, which only accepts a single argument. Here, the method takes a range as an argument to create an array of digits:

```
#!/usr/bin/ruby

digits = Array(0..9)

puts "#{digits}"
```

This will produce the following result:

```
0123456789
```

Ruby Hashes

A Hash is a collection of key-value pairs like this: "employee" => "salary". It is similar to an Array, except that indexing is done via arbitrary keys of any object type, not an integer index. The order in which you traverse a hash by either key or value may seem arbitrary and will generally not be in the insertion order. If you attempt to access a hash with a key that does not exist, the method will return *nil*.

Creating Hashes:

As with arrays, there is a variety of ways to create hashes. You can create an empty hash with the *new* class method:

```
months = Hash.new
```

You can also use *new* to create a hash with a default value, which is otherwise just *nil*:

```
months = Hash.new( "month" )

or

months = Hash.new "month"
```

When you access any key in a hash that has a default value, if the key or value doesn't exist, accessing the hash will return the default value:

```
#!/usr/bin/ruby

months = Hash.new( "month" )

puts "#{months[0]}"
puts "#{months[72]}"
```

This will produce the following result:

```
month
month
#!/usr/bin/ruby
```

```
H = Hash["a" => 100, "b" => 200]

puts "#{H['a']}"
puts "#{H['b']}"
```

This will produce the following result:

```
100
200
```

You can use any Ruby object as a key or value, even an array, so following example is a valid one:

```
[1,"jan"] => "January"
```

Ruby Iterators - each and collect

Iterators are nothing but methods supported by *collections*. Objects that store a group of data members are called collections. In Ruby, arrays and hashes can be termed collections. Iterators return all the elements of a collection, one after the other. We will be discussing two iterators here, *each* and *collect*. Let's look at these in detail.

Ruby *each* Iterator:

The each iterator returns all the elements of an array or a hash.

Syntax:

```
collection.each do |variable|
   code
end
```

Executes *code* for each element in *collection*. Here, *collection* could be an array or a ruby hash.

Example:

```
#!/usr/bin/ruby

ary = [1,2,3,4,5]
ary.each do |i|
   puts i
end
```

This will produce the following result:

```
1
2
3
4
5
```

You always associate the *each* iterator with a block. It returns each value of the array, one by one, to the block. The value is stored in the variable **i** and then displayed on the screen.

Ruby *collect* Iterator:

The *collect* iterator returns all the elements of a collection.

Syntax:

```
collection = collection.collect
```

The *collect* method need not always be associated with a block. The *collect* method returns the entire collection, regardless of whether it is an array or a hash.

Example:

```
#!/usr/bin/ruby

a = [1,2,3,4,5]
b = Array.new
b = a.collect{ |e| e }
```

```
puts b
```
This will produce the following result:
```
1
2
3
4
5
```
NOTE: The *collect* method is not the right way to do copying between arrays. There is another method called a *clone*, which should be used to copy one array into another array. You normally use the collect method when you want to do something with each of the values to get the new array. For example, this code produces an array b containing 10 times each value in a.
```
#!/usr/bin/ruby

a = [1,2,3,4,5]
b = a.collect{|x| 10*x}
puts b
```
This will produce the following result:
```
10
20
30
40
50
```
Ruby File I/O, Directories
Ruby provides a whole set of I/O-related methods implemented in the Kernel module. All the I/O methods are derived from the class IO.
The class *IO* provides all the basic methods, such as *read*, *write*, *gets*, *puts*, *readline*, *getc*, and *printf*.
This chapter will cover all the basic I/O functions available in Ruby. For more functions, please refer to Ruby Class *IO*.
The *puts* Statement:
In previous chapters, you assigned values to variables and then printed the output using *puts* statement.
The *puts* statement instructs the program to display the value stored in the variable. This will add a new line at the end of each line it writes.
Example:
```
#!/usr/bin/ruby

val1 = "This is variable one"
val2 = "This is variable two"
puts val1
puts val2
```
This will produce the following result:
```
This is variable one
This is variable two
```
The *gets* Statement:
The *gets* statement can be used to take any input from the user from standard screen called STDIN.
Example:
The following code shows you how to use the gets statement. This code will prompt the user to enter a value, which will be stored in a variable val and finally will be printed on STDOUT.
```
#!/usr/bin/ruby

puts "Enter a value :"
val = gets
```

```
puts val
```

This will produce the following result:

```
Enter a value :
This is entered value
This is entered value
```

The *putc* Statement:

Unlike the *puts* statement, which outputs the entire string onto the screen, the *putc* statement can be used to output one character at a time.

Example:

The output of the following code is just the character H:

```
#!/usr/bin/ruby

str="Hello Ruby!"
putc str
```

This will produce the following result:

```
H
```

The *print* Statement:

The *print* statement is similar to the *puts* statement. The only difference is that the *puts* statement goes to the next line after printing the contents, whereas with the *print* statement the cursor is positioned on the same line.

Example:

```
#!/usr/bin/ruby

print "Hello World"
print "Good Morning"
```

This will produce the following result:

```
Hello WorldGood Morning
```

Opening and Closing Files:

Until now, you have been reading and writing to the standard input and output. Now, we will see how to play with actual data files.

The *File.new* Method:

You can create a *File* object using *File.new* method for reading, writing, or both, according to the mode string. Finally, you can use *File.close* method to close that file.

Syntax:

```
aFile = File.new("filename", "mode")
   # ... process the file
aFile.close
```

The *File.open* Method:

You can use *File.open* method to create a new file object and assign that file object to a file. However, there is one difference in between *File.open* and *File.new* methods. The difference is that the *File.open* method can be associated with a block, whereas you cannot do the same using the *File.new* method.

```
File.open("filename", "mode") do |aFile|
   # ... process the file
end
```

Here is a list of The Different Modes of Opening a File:

Modes	Description
r	Read-only mode. The file pointer is placed at the beginning of the file. This is the default mode.
r+	Read-write mode. The file pointer will be at the beginning of the file.

w	Write-only mode. Overwrites the file if the file exists. If the file does not exist, creates a new file for writing.
w+	Read-write mode. Overwrites the existing file if the file exists. If the file does not exist, creates a new file for reading and writing.
a	Write-only mode. The file pointer is at the end of the file if the file exists. That is, the file is in the append mode. If the file does not exist, it creates a new file for writing.
a+	Read and write mode. The file pointer is at the end of the file if the file exists. The file opens in the append mode. If the file does not exist, it creates a new file for reading and writing.

Reading and Writing Files:

The same methods that we've been using for 'simple' I/O are available for all file objects. So, gets reads a line from standard input, and *aFile.gets* reads a line from the file object aFile. However, I/O objects provides additional set of access methods to make our lives easier.

The *sysread* Method:

You can use the method *sysread* to read the contents of a file. You can open the file in any of the modes when using the method sysread. For example :

Following is the input text file:

```
This is a simple text file for testing purpose.
```

Now let's try to read this file:

```
#!/usr/bin/ruby

aFile = File.new("input.txt", "r")
if aFile
   content = aFile.sysread(20)
   puts content
else
   puts "Unable to open file!"
end
```

This statement will output the first 20 characters of the file. The file pointer will now be placed at the 21st character in the file.

The *syswrite* Method:

You can use the method syswrite to write the contents into a file. You need to open the file in write mode when using the method syswrite. For example:

```
#!/usr/bin/ruby

aFile = File.new("input.txt", "r+")
if aFile
   aFile.syswrite("ABCDEF")
else
   puts "Unable to open file!"
end
```

This statement will write "ABCDEF" into the file.

The *each_byte* Method:

This method belongs to the class *File*. The method *each_byte* is always associated with a block. Consider the following code sample:

```
#!/usr/bin/ruby

aFile = File.new("input.txt", "r+")
if aFile
   aFile.syswrite("ABCDEF")
   aFile.each_byte {|ch| putc ch; putc ?. }
else
   puts "Unable to open file!"
end
```

Characters are passed one by one to the variable ch and then displayed on the screen as follows:

```
s. .a. .s.i.m.p.l.e. .t.e.x.t. .f.i.l.e. .f.o.r. .t.e.s.t.i.n.g.
.p.u.r.p.o.s.e...
.
.
```

The *IO.readlines* Method:

The class *File* is a subclass of the class IO. The class IO also has some methods, which can be used to manipulate files.

One of the IO class methods is *IO.readlines*. This method returns the contents of the file line by line. The following code displays the use of the method *IO.readlines*:

```
#!/usr/bin/ruby

arr = IO.readlines("input.txt")
puts arr[0]
puts arr[1]
```

In this code, the variable arr is an array. Each line of the file *input.txt* will be an element in the array arr. Therefore, arr[0] will contain the first line, whereas arr[1] will contain the second line of the file.

The *IO.foreach* Method:

This method also returns output line by line. The difference between the method *foreach* and the method *readlines* is that the method *foreach* is associated with a block. However, unlike the method *readlines*, the method *foreach* does not return an array. For example:

```
#!/usr/bin/ruby

IO.foreach("input.txt"){|block| puts block}
```

This code will pass the contents of the file *test* line by line to the variable block, and then the output will be displayed on the screen.

Renaming and Deleting Files:

You can rename and delete files programmatically with Ruby with the *rename* and *delete* methods.

Following is the example to rename an existing file *test1.txt*:

```
#!/usr/bin/ruby

# Rename a file from test1.txt to test2.txt
File.rename( "test1.txt", "test2.txt" )
```

Following is the example to delete an existing file *test2.txt*:

```
#!/usr/bin/ruby

# Delete file test2.txt
File.delete("test2.txt")
```

File Modes and Ownership:

Use the *chmod* method with a mask to change the mode or permissions/access list of a file:

Following is the example to change mode of an existing file *test.txt* to a mask value:

```
#!/usr/bin/ruby

file = File.new( "test.txt", "w" )
file.chmod( 0755 )
```

Following is the table, which can help you to choose different mask for *chmod* method:

Mask	Description
0700	rwx mask for owner
0400	r for owner
0200	w for owner
0100	x for owner
0070	rwx mask for group
0040	r for group

0020	w for group
0010	x for group
0007	rwx mask for other
0004	r for other
0002	w for other
0001	x for other
4000	Set user ID on execution
2000	Set group ID on execution
1000	Save swapped text, even after use

File Inquiries:

The following command tests whether a file exists before opening it:

```
#!/usr/bin/ruby

File.open("file.rb") if File::exists?( "file.rb" )
```

The following command inquire whether the file is really a file:

```
#!/usr/bin/ruby

# This returns either true or false
File.file?( "text.txt" )
```

The following command finds out if it given file name is a directory:

```
#!/usr/bin/ruby

# a directory
File::directory?( "/usr/local/bin" ) # => true

# a file
File::directory?( "file.rb" ) # => false
```

The following command finds whether the file is readable, writable or executable:

```
#!/usr/bin/ruby

File.readable?( "test.txt" )   # => true
File.writable?( "test.txt" )   # => true
File.executable?( "test.txt" ) # => false
```

The following command finds whether the file has zero size or not:

```
#!/usr/bin/ruby

File.zero?( "test.txt" )      # => true
```

The following command returns size of the file :

```
#!/usr/bin/ruby

File.size?( "text.txt" )     # => 1002
```

The following command can be used to find out a type of file :

```
#!/usr/bin/ruby

File::ftype( "test.txt" )      # => file
```

The ftype method identifies the type of the file by returning one of the following: *file, directory, characterSpecial, blockSpecial, fifo, link, socket, or unknown.*

The following command can be used to find when a file was created, modified, or last accessed :

```
#!/usr/bin/ruby

File::ctime( "test.txt" ) # => Fri May 09 10:06:37 -0700 2008
File::mtime( "text.txt" ) # => Fri May 09 10:44:44 -0700 2008
File::atime( "text.txt" ) # => Fri May 09 10:45:01 -0700 2008
```

Directories in Ruby:

All files are contained within various directories, and Ruby has no problem handling these too. Whereas the *File* class handles files, directories are handled with the *Dir* class.

Navigating Through Directories:

To change directory within a Ruby program, use *Dir.chdir* as follows. This example changes the current directory to */usr/bin*.

```
Dir.chdir("/usr/bin")
```

You can find out what the current directory is with *Dir.pwd*:

```
puts Dir.pwd # This will return something like /usr/bin
```

You can get a list of the files and directories within a specific directory using *Dir.entries*:

```
puts Dir.entries("/usr/bin").join(' ')
```

Dir.entries returns an array with all the entries within the specified directory. *Dir.foreach* provides the same feature:

```
Dir.foreach("/usr/bin") do |entry|
   puts entry
end
```

An even more concise way of getting directory listings is by using Dir's class array method:

```
Dir["/usr/bin/*"]
```

Creating a Directory:

The *Dir.mkdir* can be used to create directories:

```
Dir.mkdir("mynewdir")
```

You can also set permissions on a new directory (not one that already exists) with mkdir:

NOTE: The mask 755 sets permissions owner, group, world [anyone] to rwxr-xr-x where r = read, w = write, and x = execute.

```
Dir.mkdir( "mynewdir", 755 )
```

Deleting a Directory:

The *Dir.delete* can be used to delete a directory. The *Dir.unlink* and *Dir.rmdir* perform exactly the same function and are provided for convenience.

```
Dir.delete("testdir")
```

Creating Files & Temporary Directories:

Temporary files are those that might be created briefly during a program's execution but aren't a permanent store of information.

Dir.tmpdir provides the path to the temporary directory on the current system, although the method is not available by default. To make *Dir.tmpdir* available it's necessary to use require 'tmpdir'.

You can use *Dir.tmpdir* with *File.join* to create a platform-independent temporary file:

```
require 'tmpdir'
   tempfilename = File.join(Dir.tmpdir, "tingtong")
   tempfile = File.new(tempfilename, "w")
   tempfile.puts "This is a temporary file"
   tempfile.close
   File.delete(tempfilename)
```

This code creates a temporary file, writes data to it, and deletes it. Ruby's standard library also includes a library called *Tempfile* that can create temporary files for you:

```
require 'tempfile'
   f = Tempfile.new('tingtong')
   f.puts "Hello"
   puts f.path
   f.close
```

Ruby Regular Expressions

A *regular expression* is a special sequence of characters that helps you match or find other strings or sets of strings using a specialized syntax held in a pattern.
A *regular expression literal* is a pattern between slashes or between arbitrary delimiters followed by %r as follows:

Syntax:

```
/pattern/
/pattern/im    # option can be specified
%r!/usr/local! # general delimited regular expression
```

Example:

```
#!/usr/bin/ruby

line1 = "Cats are smarter than dogs";
line2 = "Dogs also like meat";

if ( line1 =~ /Cats(.*)/ )
  puts "Line1 contains Cats"
end
if ( line2 =~ /Cats(.*)/ )
  puts "Line2 contains  Dogs"
end
```

This will produce the following result:

```
Line1 contains Cats
```

Regular-expression modifiers:

Regular expression literals may include an optional modifier to control various aspects of matching. The modifier is specified after the second slash character, as shown previously and may be represented by one of these characters:

Modifier	Description
i	Ignore case when matching text.
o	Perform #{} interpolations only once, the first time the regexp literal is evaluated.
x	Ignores whitespace and allows comments in regular expressions
m	Matches multiple lines, recognizing newlines as normal characters
u,e,s,n	Interpret the regexp as Unicode (UTF-8), EUC, SJIS, or ASCII. If none of these modifiers is specified, the regular expression is assumed to use the source encoding.

Like string literals delimited with %Q, Ruby allows you to begin your regular expressions with %r followed by a delimiter of your choice. This is useful when the pattern you are describing contains a lot of forward slash characters that you don't want to escape:

```
# Following matches a single slash character, no escape required
%r|/|

# Flag characters are allowed with this syntax, too
%r[</(.*)>]i
```

Regular-expression patterns:

Except for control characters, **(+ ? . * ^ $ ( ) [ ] { } | \)**, all characters match themselves. You can escape a control character by preceding it with a backslash.
Following table lists the regular expression syntax that is available in Ruby.

Pattern	Description

^	Matches beginning of line.
$	Matches end of line.
.	Matches any single character except newline. Using m option allows it to match newline as well.
[...]	Matches any single character in brackets.
[^...]	Matches any single character not in brackets
re*	Matches 0 or more occurrences of preceding expression.
re+	Matches 1 or more occurrence of preceding expression.
re?	Matches 0 or 1 occurrence of preceding expression.
re{ n}	Matches exactly n number of occurrences of preceding expression.
re{ n,}	Matches n or more occurrences of preceding expression.
re{ n, m}	Matches at least n and at most m occurrences of preceding expression.
a| b	Matches either a or b.
(re)	Groups regular expressions and remembers matched text.
(?imx)	Temporarily toggles on i, m, or x options within a regular expression. If in parentheses, only that area is affected.
(?-imx)	Temporarily toggles off i, m, or x options within a regular expression. If in parentheses, only that area is affected.
(?: re)	Groups regular expressions without remembering matched text.
(?imx: re)	Temporarily toggles on i, m, or x options within parentheses.
(?-imx: re)	Temporarily toggles off i, m, or x options within parentheses.
(?#...)	Comment.
(?= re)	Specifies position using a pattern. Doesn't have a range.
(?! re)	Specifies position using pattern negation. Doesn't have a range.
(?> re)	Matches independent pattern without backtracking.
\w	Matches word characters.
\W	Matches nonword characters.
\s	Matches whitespace. Equivalent to [\t\n\r\f].
\S	Matches nonwhitespace.
\d	Matches digits. Equivalent to [0-9].
\D	Matches nondigits.
\A	Matches beginning of string.
\Z	Matches end of string. If a newline exists, it matches just before newline.
\z	Matches end of string.
\G	Matches point where last match finished.
\b	Matches word boundaries when outside brackets. Matches backspace (0x08) when inside brackets.
\B	Matches nonword boundaries.
\n, \t, etc.	Matches newlines, carriage returns, tabs, etc.
\1...\9	Matches nth grouped subexpression.
\10	Matches nth grouped subexpression if it matched already. Otherwise refers to the octal representation of a character code.

Regular-expression Examples:

Literal characters:

Example	Description
/ruby/	Match "ruby".
¥	Matches Yen sign. Multibyte characters are supported in Ruby 1.9 and Ruby 1.8.

Character classes:

Example	Description
/[Rr]uby/	Match "Ruby" or "ruby"
/rub[ye]/	Match "ruby" or "rube"
/[aeiou]/	Match any one lowercase vowel
/[0-9]/	Match any digit; same as /[0123456789]/
/[a-z]/	Match any lowercase ASCII letter
/[A-Z]/	Match any uppercase ASCII letter
/[a-zA-Z0-9]/	Match any of the above
/[^aeiou]/	Match anything other than a lowercase vowel
/[^0-9]/	Match anything other than a digit

Special Character Classes:

Example	Description
/./	Match any character except newline
/./m	In multiline mode . matches newline, too
/\d/	Match a digit: /[0-9]/
/\D/	Match a nondigit: /[^0-9]/
/\s/	Match a whitespace character: /[\t\r\n\f]/
/\S/	Match nonwhitespace: /[^ \t\r\n\f]/
/\w/	Match a single word character: /[A-Za-z0-9_]/
/\W/	Match a nonword character: /[^A-Za-z0-9_]/

Repetition Cases:

Example	Description
/ruby?/	Match "rub" or "ruby": the y is optional
/ruby*/	Match "rub" plus 0 or more ys
/ruby+/	Match "rub" plus 1 or more ys
/\d{3}/	Match exactly 3 digits
/\d{3,}/	Match 3 or more digits
/\d{3,5}/	Match 3, 4, or 5 digits

Nongreedy repetition:

This matches the smallest number of repetitions:

Example	Description
/<.*>/	Greedy repetition: matches "<ruby>perl>"
/<.*?>/	Nongreedy: matches "<ruby>" in "<ruby>perl>"

Grouping with parentheses:

Example	Description

/\D\d+/ No group: + repeats \d

/(\D\d)+/ Grouped: + repeats \D\d pair

/([Rr]uby(,)?)+/ Match "Ruby", "Ruby, ruby, ruby", etc.

Backreferences:

This matches a previously matched group again:

Example	Description
/([Rr])uby&\1ails/	Match ruby&rails or Ruby&Rails
/(['"])(?:(?!\1).)*\1/	Single or double-quoted string. \1 matches whatever the 1st group matched . \2 matches whatever the 2nd group matched, etc.

Alternatives:

Example	Description
/ruby|rube/	Match "ruby" or "rube"
/rub(y|le))/	Match "ruby" or "ruble"
/ruby(!+|\?)/	"ruby" followed by one or more ! or one ?

Anchors:

This need to specify match position

Example	Description
/^Ruby/	Match "Ruby" at the start of a string or internal line
/Ruby$/	Match "Ruby" at the end of a string or line
/\ARuby/	Match "Ruby" at the start of a string
/Ruby\Z/	Match "Ruby" at the end of a string
/\bRuby\b/	Match "Ruby" at a word boundary
/\brub\B/	\B is nonword boundary: match "rub" in "rube" and "ruby" but not alone
/Ruby(?=!)/	Match "Ruby", if followed by an exclamation point
/Ruby(?!!)/	Match "Ruby", if not followed by an exclamation point

Special syntax with parentheses:

Example	Description
/R(?#comment)/	Matches "R". All the rest is a comment
/R(?i)uby/	Case-insensitive while matching "uby"
/R(?i:uby)/	Same as above
/rub(?:y|le))/	Group only without creating \1 backreference

Search and Replace:

Some of the most important String methods that use regular expressions are **sub** and **gsub** , and their in-place variants **sub!** and **gsub!**.

All of these methods perform a search-and-replace operation using a Regexp pattern. The **sub** & **sub!** replace the first occurrence of the pattern and **gsub** & **gsub!** replace all occurrences. The **sub** and **gsub** return a new string, leaving the original unmodified where as **sub!** and **gsub!** modify the string on which they are called.

Following is the example:

```
#!/usr/bin/ruby

phone = "2004-959-559 #This is Phone Number"

# Delete Ruby-style comments
phone = phone.sub!(/#.*$/, "")
puts "Phone Num : #{phone}"
```

```
# Remove anything other than digits
phone = phone.gsub!(/\D/, "")
puts "Phone Num : #{phone}"
```

This will produce the following result:

```
Phone Num : 2004-959-559
Phone Num : 2004959559
```

Following is another example:

```
#!/usr/bin/ruby

text = "rails are rails, really good Ruby on Rails"

# Change "rails" to "Rails" throughout
text.gsub!("rails", "Rails")

# Capitalize the word "Rails" throughout
text.gsub!(/\brails\b/, "Rails")

puts "#{text}"
```

This will produce the following result:

```
Rails are Rails, really good Ruby on Rails
```

Ruby Web Applications - CGI Programming

Ruby is a general-purpose language; it can't properly be called a *web language* at all. Even so, web applications and web tools in general are among the most common uses of Ruby. Not only can you write your own SMTP server, FTP daemon, or Web server in Ruby, but you can also use Ruby for more usual tasks such as CGI programming or as a replacement for PHP.

Please spend few minutes with CGI Programming Tutorial for more detail on CGI Programming.

Writing CGI Scripts:

The most basic Ruby CGI script looks like this:

```
#!/usr/bin/ruby

puts "HTTP/1.0 200 OK"
puts "Content-type: text/html\n\n"
puts "<html><body>This is a test</body></html>"
```

If you call this script *test.cgi* and uploaded it to a Unix-based Web hosting provider with the right permissions, you could use it as a CGI script.

For example, if you have the Web site http://www.example.com/ hosted with a Linux Web hosting provider and you upload *test.cgi* to the main directory and give it execute permissions, then visiting http://www.example.com/test.cgi should return an HTML page saying ***This is a test***.

Here when *test.cgi* is requested from a Web browser, the Web server looks for *test.cgi* on the Web site, and then executes it using the Ruby interpreter. The Ruby script returns a basic HTTP header and then returns a basic HTML document.

Using cgi.rb:

Ruby comes with a special library called **cgi** that enables more sophisticated interactions than those with the preceding CGI script.

Let's create a basic CGI script that uses cgi:

```
#!/usr/bin/ruby

require 'cgi'

cgi = CGI.new
puts cgi.header
puts "<html><body>This is a test</body></html>"
```

Here, you created a CGI object and used it to print the header line for you.

Form Processing:

Using class CGI gives you access to HTML query parameters in two ways. Suppose we are given a URL of /cgi-bin/test.cgi?FirstName=Zara&LastName=Ali.

You can access the parameters *FirstName* and *LastName* using CGI#[] directly as follows:

```
#!/usr/bin/ruby

require 'cgi'
cgi = CGI.new
cgi['FirstName'] # =>  ["Zara"]
cgi['LastName']  # =>  ["Ali"]
```

There is another way to access these form variables. This code will give you a hash of all the key and values:

```
#!/usr/bin/ruby

require 'cgi'
cgi = CGI.new
h = cgi.params  # =>  {"FirstName"=>["Zara"],"LastName"=>["Ali"]}
h['FirstName']  # =>  ["Zara"]
h['LastName']   # =>  ["Ali"]
```

Following is the code to retrieve all the keys:

```
#!/usr/bin/ruby

require 'cgi'
cgi = CGI.new
cgi.keys          # =>  ["FirstName", "LastName"]
```

If a form contains multiple fields with the same name, the corresponding values will be returned to the script as an array. The [] accessor returns just the first of these.index the result of the params method to get them all.

In this example, assume the form has three fields called "name" and we entered three names "Zara", "Huma" and "Nuha":

```
#!/usr/bin/ruby

require 'cgi'
cgi = CGI.new
cgi['name']         # => "Zara"
cgi.params['name'] # => ["Zara", "Huma", "Nuha"]
cgi.keys            # => ["name"]
cgi.params          # => {"name"=>["Zara", "Huma", "Nuha"]}
```

Note: Ruby will take care of GET and POST methods automatically. There is no separate treatment for these two different methods.

An associated, but basic, form that could send the correct data would have HTML code like so:

```
<html>
<body>
<form method="POST" action="http://www.example.com/test.cgi">
First Name :<input type="text" name="FirstName" value="" />
<br />
Last Name :<input type="text" name="LastName" value="" />

<input type="submit" value="Submit Data" />
</form>
</body>
</html>
```

Creating Forms and HTML:

CGI contains a huge number of methods used to create HTML. You will find one method per tag. In order to enable these methods, you must create a CGI object by calling CGI.new.

To make tag nesting easier, these methods take their content as code blocks. The code blocks should return a *String*, which will be used as the content for the tag. For example:

```
#!/usr/bin/ruby

require "cgi"
cgi = CGI.new("html4")
cgi.out{
   cgi.html{
      cgi.head{ "\n"+cgi.title{"This Is a Test"} } +
      cgi.body{ "\n"+
         cgi.form{"\n"+
            cgi.hr +
            cgi.h1 { "A Form: " } + "\n"+
            cgi.textarea("get_text") +"\n"+
            cgi.br +
            cgi.submit
         }
      }
   }
}
```

NOTE: The *form* method of the CGI class can accept a method parameter, which will set the HTTP method (GET, POST, and so on...) to be used on form submittal. The default, used in this example, is POST.

This will produce the following result:

```
Content-Type: text/html
Content-Length: 302
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0 Final//EN">
<HTML>
<HEAD>
<TITLE>This Is a Test</TITLE>
</HEAD>
<BODY>
<FORM METHOD="post" ENCTYPE="application/x-www-form-urlencoded">
<HR>
<H1>A Form: </H1>
<TEXTAREA COLS="70" NAME="get_text" ROWS="10"></TEXTAREA>
<BR>
<INPUT TYPE="submit">
</FORM>
</BODY>
</HTML>
```

TutorialsDuniya.com

Download FREE Computer Science Notes, Programs, Projects, Books PDF for any university student of BCA, MCA, B.Sc, B.Tech CSE, M.Sc, M.Tech at https://www.tutorialsduniya.com

- Algorithms Notes
- Artificial Intelligence
- Android Programming
- C & C++ Programming
- Combinatorial Optimization
- Computer Graphics
- Computer Networks
- Computer System Architecture
- DBMS & SQL Notes
- Data Analysis & Visualization
- Data Mining
- Data Science
- Data Structures
- Deep Learning
- Digital Image Processing
- Discrete Mathematics
- Information Security
- Internet Technologies
- Java Programming
- JavaScript & jQuery
- Machine Learning
- Microprocessor
- Operating System
- Operational Research
- PHP Notes
- Python Programming
- R Programming
- Software Engineering
- System Programming
- Theory of Computation
- Unix Network Programming
- Web Design & Development

Please Share these Notes with your Friends as well