LECTURE NOTES

ON

WEB TECHNOLOGIES
ACADEMIC YEAR 2021-22

III B.Tech.–II SEMESTER (R19)

G.K.HAVILAH, Assistant Professor

DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

V S M COLLEGE OF ENGINEERING
RAMCHANDRAPURAM
E.G DISTRICT
533255

III Year – II Semester		**L**	**T**	**P**	**C**
		3	**0**	**0**	**3**
WEB TECHNOLOGIES					

Course Objectives:
From the course the student will learn

- Translate user requirements into the overall architecture and implementation of newsystems and Manage Project and coordinate with the Client
- Write backend code in PHP language and Writing optimized front end code HTML andJavaScript
- Understand, create and debug database related queries and Create test code to validate theapplications against client requirement
- Monitor the performance of web applications & infrastructure and Troubleshooting webapplication with a fast and accurate a resolution

Course Outcomes:

- Illustrate the basic concepts of HTML and CSS & apply those concepts to design staticweb pages
- Identify and understand various concepts related to dynamic web pages and validate themusing JavaScript
- Outline the concepts of Extensible markup language & AJAX
- Develop web Applications using Scripting Languages & Frameworks
- Create and deploy secure, usable database driven web applications using PHP and RUBY

UNIT I
HTML: Basic Syntax, Standard HTML Document Structure, Basic Text Markup, Html styles, Elements, Attributes, Heading, Layouts, Html media, Iframes Images, Hypertext Links, Lists, Tables, Forms, GET and POST method, HTML 5, Dynamic HTML.
CSS: Cascading style sheets, Levels of Style Sheets, Style Specification Formats, Selector Forms, The Box Model, Conflict Resolution, CSS3.

UNIT II
Javascript - Introduction to Javascript, Objects, Primitives Operations and Expressions, Control Statements, Arrays, Functions, Constructors, Pattern Matching using Regular Expressions, Fundamentals of Angular JS and NODE JS Angular Java Script- Introduction to Angular JS Expressions: ARRAY, Objects, Strings, Angular JS Form Validation & Form Submission.
Node.js- Introduction, Advantages, Node.js Process Model, Node JS Modules, Node JS File system, Node JS URL module, Node JS Events.

UNIT III
Working with XML: Document type Definition (DTD), XML schemas, XSLT, Document object model, Parsers - DOM and SAX.
AJAX A New Approach: Introduction to AJAX, Basics of AJAX, XML Http Request Object, AJAX UI tags, Integrating PHP and AJAX.

UNIT IV

PHP Programming: Introduction to PHP, Creating PHP script, Running PHP script. Working with variables and constants: Using variables, Using constants, Data types, Operators. Controlling program flow: Conditional statements, Control statements, Arrays, functions.

UNIT V

Web Servers- IIS (XAMPP, LAMP) and Tomcat Servers. Java Web Technologies-Introduction toServlet, Life cycle of Servlet, Servlet methods, Java Server Pages.
Database connectivity – Servlets, JSP, PHP, Practice of SQL Queries.Introduction to Mongo DB and JQuery.
Web development frameworks – Introduction to Ruby, Ruby Scripting, Ruby on rails – Design,Implementation and Maintenance aspects.

Text Books:

1) Programming the World Wide Web, 7th Edition, Robet W Sebesta, Pearson, 2013.
2) Web Technologies, 1st Edition 7th impression, Uttam K Roy, Oxford, 2012.
3) Pro Mean Stack Development, 1st Edition, ELad Elrom, Apress O'Reilly, 2016
4) Java Script & jQuery the missing manual, 2nd Edition, David sawyer mcfarland, O'Reilly,2011.
5) Web Hosting for Dummies, 1st Edition, Peter Pollock, John Wiley & Sons, 2013.
6) RESTful web services, 1st Edition, Leonard Richardson, Ruby, O'Reilly, 2007.

Reference Books:

1) Ruby on Rails Up and Running, Lightning fast Web development, 1st Edition, BruceTate, Curt Hibbs, Oreilly, 2006.
2) Programming Perl, 4th Edition, Tom Christiansen, Jonathan Orwant, O'Reilly, 2012.
3) Web Technologies, HTML, JavaScript, PHP, Java, JSP, XML and AJAX, Black book, 1stEdition, Dream Tech, 2009.
4) An Introduction to Web Design, Programming, 1st Edition, Paul S Wang, Sanda S Katila,Cengage Learning, 2003.

online- 6/20

HTML

Introduction :-

The internet is a global system of networked computers together with their users and data.

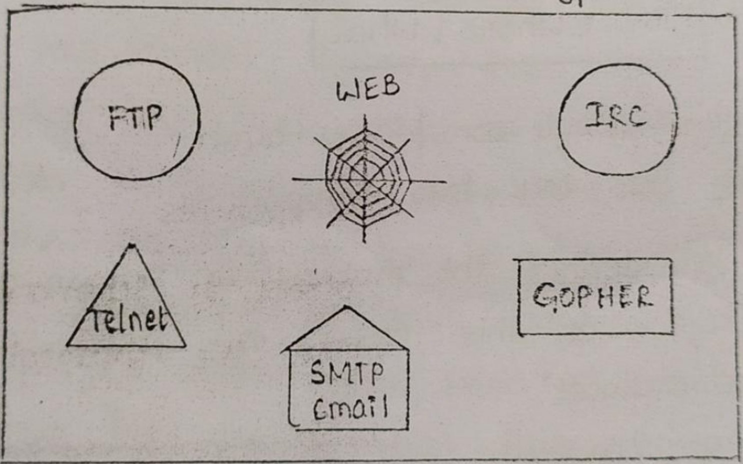
World Wide Web (WWW):

WWW was created by "Tim Berners" Lee" at CERN as a simple way to publish information and make it availa on the internet.

* Web is a software application that it makes easier possible for nearly anyone to publish and browse hyperte documents on the internet. It is called "WEB" because t interconnections between documents ressembler a spiders w

* The Web runs on the HTTP (Hyper text Transfer Protoc

Internet:-



The above figure shows the relationships between the we the internet and the number of applications.

Uniform Resource Locator (URL) :-

The central idea in the development of Web was the URL A URL is a Web address that uniquely identifies a docu on the Web such as a document can be an image, a HTML file, a program etc., Unique address makes lining anyone's web documents possible.

cause the browser to attempt to retrieve that page

If the browser is successful in trying to find the page. and the browser will display it. This high level explanation does not, however, convey any of the details of what is happening. To go from a URL to having the Webpage displayed, the browser needs to be able to answer such questions such as ....

a) How can the page be accessed?

b) Where can the page found?

c) What is the file name corresponding to the page?

→ The URL is designed to incorporate sufficient information to resolve these questions. the URL has 3 parts. so we can view format of URL as follows:

$\boxed{\text{HOW: // Where / What}}$

Eg: http: // www. google.com / index.html.

let us break this URL into components

http (How):- It defines the protocol or schema by which to access the page. In this case, the protocol is "Hyper Text" Transfer protocol".

WWW. google. com (Where):-

It defines the domain name of the computer where the page resides. The computer is web server capable of satisfying page requests. Just as a Walter - server food, a web server serves web pages.

Index. html (What):-

It provides the local name (filename) uniquely identifying the specific page.

Note: The URL consists of a protocol, a web servers name and a filename.

A Web browser is one of many [illegible] user and the interne
functions as the interface between a
popular browsers include Netscape Navigator, MICROSOFT's
Internet Explorer, Mosaic and lynx.

Hyperlink:-

A string of clickable text or a clickable graphic that points to another web page or document. When the hyperlink I is Selected, another web page II is requested., retrieved and renderred by Browser.

Hyper Text:-

Web pages that have hyperlinks to other pages are called Hyper text.

Website:-

An entity on the internet that publishes Web pages is called Website.

Web Technologies:-

The technologies Which are used for designing web pages or Websites We can call it as Web technologies.

Web page:

A hypertext document connected to the WWW. It contains text, images, audio, video (Multimedia) these are viewed by browser. With a web browser one can view Webpages that may contain text, images, video and other multimedia and navigate between them via hyperlinks.

Browsing

Searching for information Which is specific.

Surfing

To spend time visiting a lot of Websites.

Browser

An application used to access and view Websites and Webpages.
Ex: chrome, firefox, Explorer etc...

Search Engine.

It is also a program. It searches for some particular docume
When specific keyword is entered.

Domain:

A group of computers on a network that are administered as a unit with common rules and procedures.
Ex .gov, .edu, .in, .com (TLD) (1985, Jan)

Domain Name

It is the name of Website.

HTTP:

It is a protocol that web browsers and web servers used to communicate with eachother over the internet.

DNS :-

Domain Name System is a system used to convert a compute hostname into an IP address on the internet.

Server :-

It is a computer that provides data to other computers. It may serve data to systems on LAN or WAN over the internet

Database :-

It is an organised collection of data generally stored and accessed electronically from a computer system.

* HTML

HTML stands for Hyper Text Markup Language. HTML is a method of describing the format of documents which allows them to be viewed on computer screens. HTML pages can be developed to be simple text or to be complex multimedia extravaganza containing sound, moving images, virtual reality and Java applets.

→ HTML is not a programming language, you can't write an HTML program and expect anything to happen.

→ HTML is not a data description language that the HTML you write won't tell anything about the structure of your code.

→ HTML is not really very complicated - although the creator of WYSIWYG [What You See Is What You Get] authoring tools would like you to think that it is.

History

The idea of Hypertext and Hyperlink documents has been around for a While. For this practical implementation is done by a number of ~~techniques~~ technologies which began to come together. In 1980's an early example being the hyper card information management system from apple.

→ The phenomenal success of HTML as a format is due to the "Mosaic" browser developed at NCSA. The US Super computing center and the simplicity of the language itself.

→ HTML is an application of SGML [Standardized Generalized Markup Language] SGML grew from a number of pieces of Work, carried by charles Gold farb, Edward Mosher and Raymond lorie at IBM who created a General mark up Language in 1960's.

Versions:-

Version 2.0 : It was released in 1994 and remains the baseline for backwards compatibility and should be supported by all browsers and authoring tools.

Version 3.2:- It was released in 1996 with many useful additions.

Version 4.0:- It was ratified towards the end of 1997 and slightly arounded in late 1999.

In HTML, formatting is specified by using Tags. A Tag is a format name surrounded by Angular Brackets. End tags which switch a format off also contain a "forward slash".

Eg:

<H1> Text in H1 style </H1>

where <H1> is an opening tag and </H1> is closing tag. Tags are not case sensitive i.e., <HEAD>, <head> and <hEa are equivalent style must be switched off by an end tag.

→ White spaces, tabs and new lines are ignored by the browsers, If a browser does not understand a tag, It will ignore it.

Structure of HTML:

```
<html>
  <head>
     <title> Write title of document </title>
  </head>
  <body>
     <!-- This is the body section of the html code -->
  </body>
</html>
```

```
<!-- ... First program in HTML ... -->
<html>
  <head>
     <title> First program </title>
  </head>
  <body bgcolor="red" text="White" bottommargin="30" top margi
                                      = "200" scroll = "yes">
  <b><i><u>CSE </u></i></b>
  </body>
</html>
```

1) Comment tags :- This tag is used to write comments the HTML source code. These are not to executed but they are used for user understandability.

Single-line comment tag:-

<!-- This is a single-line comment tag -->

Multiple-line comment tag:-

<!-- This is a Multiple-line comment tag --
-- comment continues with two hyphens --
-- at beginning and ending of the files -->

2) <HTML> - - - </HTML> Tag :-

This tag is used to tell the browser that every thing continued between them is a HTML code for creating web page

<HTML> Shows starting of html page
</HTML> Shows ending of html page.

3) <HEAD> - - - </HEAD> Tag:-

The tag contains the head section of a HTML document. This tag holds information about the document such as title..
These are the tags that can appear only in the section like <title>, <link>; <meta>

4) <TITLE> - - - </TITLE> Tag

This tag gives the title of Webpage. This title appears in the web browsers title bars and is used by search engines to refer the document.

5) <BODY> - - - </BODY> Tag

This tag contains the body of the html document Which includes creating web pages content like text, images, links, forms etc.. that appear in the Web browser.

This tag has some attributes. They are as follows:

a) background :-

This attribute is used to set a background of a browser like some images.

b) ~~bgcol~~ bgcolor :-

This attribute specifies the color of a browser background.

c) bottom margin :-

It is used to set the bottom margin or empty space at the bottom of the document in pixels

d) top margin :-

It is used to set the top margin or empty space at the top of the document in pixels.

e) left margin :-

It is used to set the left margin or empty space at the left of the document in pixels.

f) right margin :-

It is used to set the right margin or empty space at the right of the document in pixels.

g) text :-

It is used to specify the color of the text in the document.

h) scroll :-

It specifies whether a vertical scroll bar appears to the right side of the document, can be YES or NO

i) Document Type Declaration (DTD) :-

The DTD for basic HTML is as follows:

```
<! DOCTYPE HTML public "-//w3c//dtd html 4.0//en"
"HTTP://www.w3.org/tr/pr-html 4.0/transitional.dtd">
```

Example 1-

```
<html>
    <head>
        <title> Untitled Document </title>
    </head>
    <body bgcolor="skyblue" text="red" leftmargin ="300" scroll
                    "YES" right margin ="300" > WELCOME TO JAVA A
                            WEB TECHNOLOGIES
    </body>
</html>
```

6) Header tag

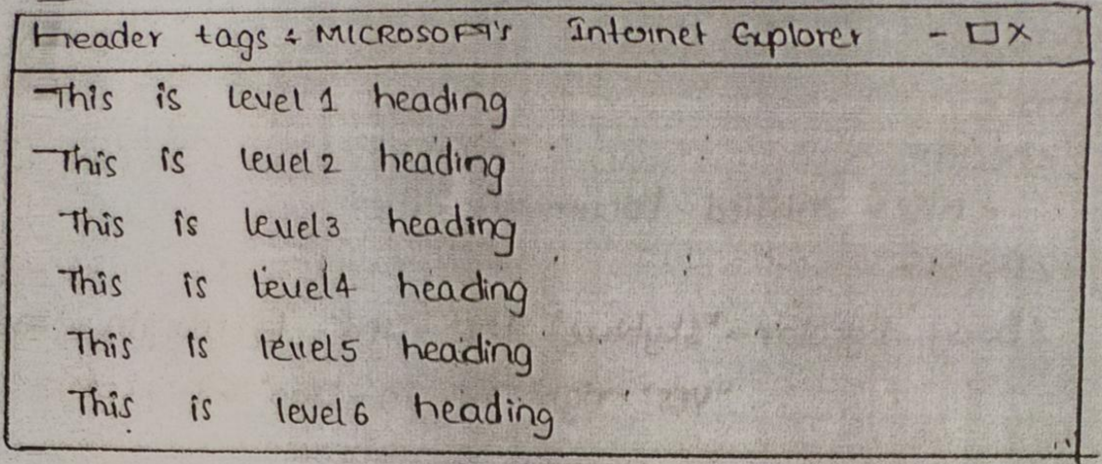
It is very important part of any HTML page. It contains lot of control information that is needed by the browsers and servers. These tags are used for printing headings. The tags (<H1>, <H2>, <H3>, <H4>, <H5> and <H6>

<H1> is the highest (or) Biggest level headings
<H6> is the lowest (or) smallest level headings

Example :-

The example for an Heading tag is as follows,

```
<html>
    <head>
        <title> Header tags. </title>
    </head>
    <body>
        <H1> This is level 1 heading </H1>
        <H2> This is level 2 heading </H2>
        <H3> This is level 3 heading </H3>
        <H4> This is level 4 heading </H4>
        <H5> This is level 5 heading </H5>
        <H6> This is level 6 heading </H6>
    </body>
</html>
```



Header tags - MICROSOFT's Internet Explorer - ☐ X

This is level 1 heading

This is level 2 heading

This is level3 heading

This is level4 heading

This is level5 heading

This is level6 heading

7) Font tag

The settings of font on Webpage are making text font bigger, smaller, bolder and have different colors. fonts are one of the most important visual elements of our webpages.

Attributes of font:-

i) font family

ii) font size

iii) font color

Eg: <font size = 40 color ="yellow" face = "georgia" > Here write your own text </font>

~~Text~~

8) Text formatting tags:-

(i) Bold tag: It sets the text style to Bold.

syntax : $\boxed{\text{<b> --- </b>}}$

(ii) Italic tag:- It displays the text in Italic form

syntax: $\boxed{\text{<i> --- </i>}}$

(iii) strik eout tag: The tag cause a line to be drawn through the text.

syntax : $\boxed{\text{<S> ---- </s>}}$

(iv) Underlined tag: This tag underlines the enclosed text

syntax: $\boxed{\text{<u> --- <u>}}$

(v) <BIG> and <SMALL> Tag: This tag makes the text to look bigger and smaller

<SUB> sets the text as subscripts. This puts the charact
down than regular text. <SUP> sets the text as super sc
which parts the characters higher up than regular text.

Eg:- The formulae is = (a+b) ² ⇒ $(a+b)^2$

The chemical is = h ₂ = h_2

(vii) Strong tag :-

This tag usually displays the text in bold which emphasizes the text strongly.

syntax : `<strong> - - - </strong>`

(viii) Center tag :-

This tag helps to align text which it contended in the center of the web page.

syntax : `<center> - - - </center>`

(ix) Deleted text tag :-

This tag usually marks text as deleted and displays as strike through text in browser.

Syntax : `<del> - - - </del>`

(x) columns Tag :-

It breaks the text into columns.

syntax : `<multicol> - - - </multicol>`

Example:

```
<html>
  <head>
      <title> Example of font attributes </title> </head>
    <body bgcolor = "skyblue">
      <center>
         <h1> <blink> Example of font </blink> </h1>
         this is : <b> Bold text </b>
         <br>
         <h3> this is : <i> Italic text </i> <br>
```

```
this is : very <big> BIG </big> & very <small> SMALL </small
<br>
& this is: <u> underline text </u>
  this is : <s> Strike out tag </s>
<br>
The formulae is = (a+b) <sup> 2 </sup> <br>
The chemical is = H<SUB> 2 </sub> O <sub> 2 </sub> <br>
this blinks : <blink> computer science </blink> <br>
</h3>
</center>
</body>
</html>
```

Output:

Example of font attributes - MICROSOFT Internet Explorer -□X

Example of font

this is : Bold text

this is : italic text

this is : very BIG & very SMALL

this is : <u>underlined text</u>

this is : Strike out

The formulae is = $(a+b)^2$

The chemical is = H_2O_2

this blinks : computer science

9) Marquee tag :-

This tag displays the scrolling text in a marquee style

syntax : $\boxed{\text{<marquee> - - - - </marquee>}}$

Attributes of marquee tag :-

a) Align :- sets the alignment of the text to top (by default) middle (or) bottom.

2020/8/13 14:0

b) Behaviour:-

It shows how the text in the marquee tag should move. can SCROLL (dy by default): the text SLIDE (text enters froi one slide and stops at another side) or ALTERNATE (text Seems to bounce from one slide to another).

c) Bgcolor:- It sets the background color for the marquee t

d) Direction:- It sets the direction of the text for scrolling. It can be LEFT (by default). RIGHT, UP (or) DOWN.

Example:

<Marquee align="middle" behavior="alternate" bgcolor="blue" direction="up"> This is an example of Marquee tag </marq

10) Line and paragraph tags:-

(i) Horizontal line tag:-

It draws a horizontal line to separate or group items horizont This tag does not have any ending (or) closing tag.

Syntax: $\boxed{\text{<HR>}}$

Attributes of Horizontal tag

a) Align: sets the alignments for line to either LEFT, RIGHT (o CENTER (by default)

b) color: sets the color of the line

c) size: sets the size of the horizontal line

(ii) BLINK tag:-

This tag causes the text to blink

syntax: $\boxed{\text{<blink> Text Blinks </blink>}}$

(iii) Line Break tag:

It intersects a line break into a page. This tag does not have the closing tag.

Syntax: $\boxed{\text{
}}$

2020/8/13 14:

(iv) No line Break tag:
This tag causes the browser not to break the text into separate line

Syntax : <Nobr> --- </Nobr>

(v) paragraph tag:
This tag indicates the start of a new paragraph, producing a single blank line in between the two paragraphs.

Syntax : <p> --- </p>

Attributes of paragraph tag :-
Align: It sets the alignment of the text in the paragraph where it sets the para either to LEFT (by default), RIGHT, CENTER, JUSTIFY.

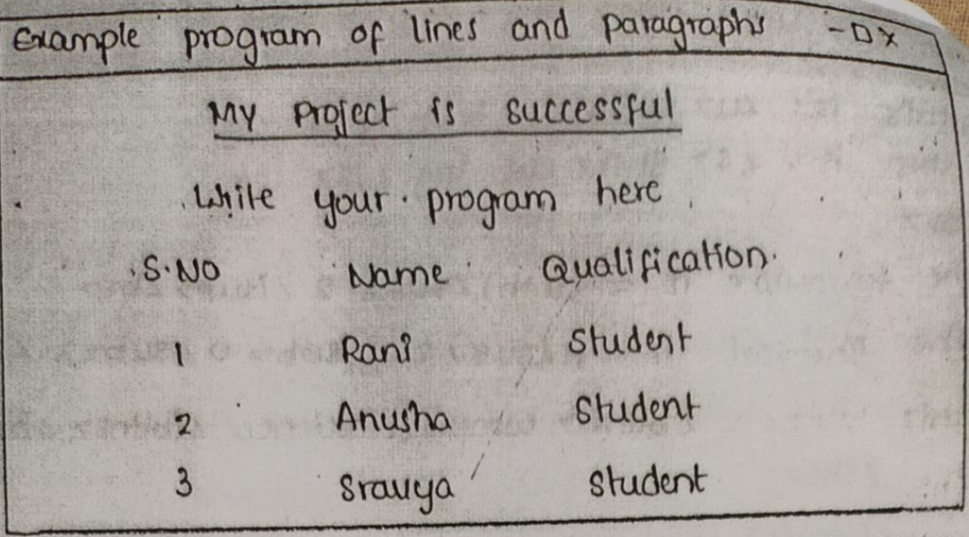
Eg. <p align = center> ---- </p>

(vi) pre-formatted tag :-
This tag is one of the handiest tags in the HTML tool box. <PRE ---> marks the text as "preformatted" all the spaces and carriage returns are rendered exactly as you type them.

Syntax : <pre> ---- </pre>

```
<!-- HTML script on line and paragraph does tags -->
<html>
  <head>
    <title> Example program of lines and paragraphs </title>
  </head>
  <body bgcolor = "cream">
  <center>
  <H3> My project is successful </H3>
  <p> Write your paragraph here </p> <br>
  <pre>
     S.NO     Name      Qualification
      1       Rani       Student
      2       Anusha     Student
      3       Sravya     Student
 </pre> </center> </body> </html>
```

2020/8/13 14:04


Example program of lines and paragraphs -□x

My Project is successful

While your program here

S.NO	Name	Qualification
1	Rani	Student
2	Anusha	Student
3	Sravya	Student

Special characters:

(i) Space (single white space): The format of inserting this special character is &nbsp;

(ii) Ampersand (&): The format of inserting this special characters is &amp. /&#60;

Linking: In web terms, a hyper link is a reference to a resource on web. Hyper link can points to any resource on the web: an HTML page, an image, a sound file, a movie etc..

1) Anchor tag:-

An anchor tag is a term used to define a hyper link destination inside a document. The HTML anchor element <a> is used to define both hyperlinks & anchors.

Syntax : $\boxed{\text{<a href = "address"> link text </a>}}$

Attributes of Anchor tag:-

a) href :- The href attributes defines the link "address". This <a> element defines a link to w3 schools

<a href = "http://www.w3 schools.com/"> visit w3 schools! </a>

b) Target:- The target attribute defines where the linked document be opened. The below code will open the document in a ~~w~~ new browser.

2020/8/13 14:04

window:-

```
<a href = "heading.html1" target = "blank"> visit w3 schools! </a>
```

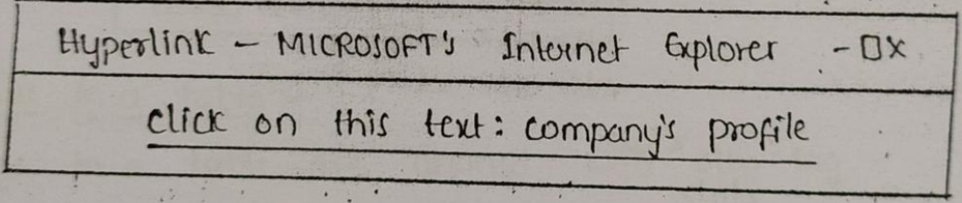
c) Name:-

When the name attribute is used, the <a> element defines a named anchor inside a HTML document. Named Anchor are not displayed in any special way. They are invisible to the reader.

Example:-

```
<html>
   <head>
      <title> Hyper link </title>
   </head>
   <body bgcolor = "#99ffcc">
   <a href = "profile.html"> click on the text: company's profile </a>
   </body>
</html>
```

Output:



```
Hyperlink - MICROSOFT's Internet Explorer   - □ x
click on this text: company's profile
```

Named Anchor Syntax:-

```
<a name = "label"> Any content </a>
```

The link syntax to a named anchor:

```
<a href = "# label"> Any content </a>
```

The # defines a linke to named anchor

```
<!--HTML script on Anchor tag-->
<html>
  <head>
    <title> Anchor tag </title>
  </head>
  <body>
```

```
<img src="D://I0/b.jpeg" width="200" height="300" Image </img
<a href="http://www.manabadi.com" target="blank"> open
                    manabadi website</a>
</body>
</html>
```

LISTS

HTML offers author several mechanisms for specifying lists of information. All lists must contain one or more lists element.

* lists may contain 3 types : (i) unordered list
  (ii) ordered list
  (iii) Definition list.

(i) unordered list :

An unordered list is a list of items. The list of items are marked with bullets (typically small black circles). An unorde list with <ul> tag. Each list item starts with <li> tag.

```
<ul>
    <li> coffee </li>
    <li> Milk </li>
</ul>
```

Here is how it looks in a Browser

o/p:

```
• coffee
• Milk
```

Inside a list item you can put paragraphs, line breaks, im links etc.,

Eg:-
```
<ul type="SQUARE">
    <li> CSE </li>
    <li> ECE </li>
</ul>
```

Here is how it looks in a Browser

o/p:

```
■ CSE
■ ECE
```

2020/8/13 1

An ordered list is also a list of items. The list items are marke
with numbers. An ordered list starts with <ol> tag. Each
list item starts with <li> tag.

Syntax:
```
<ol type = "I|a|A|i|1">
    <li> - - - </li>
    <li> - - - </li>
</ol>
```

Inside a list item you can put paragraphs, line breaks, images, links etc---

Eg:
```
<ol   type = "a"   start = "c">
        <li> B.Tech </li>
        <li> M.Tech </li>
</ol>
```

Here is how it looks like a browser.

o/p =
```
c. B.Tech
d. M.Tech
```

Definition list:

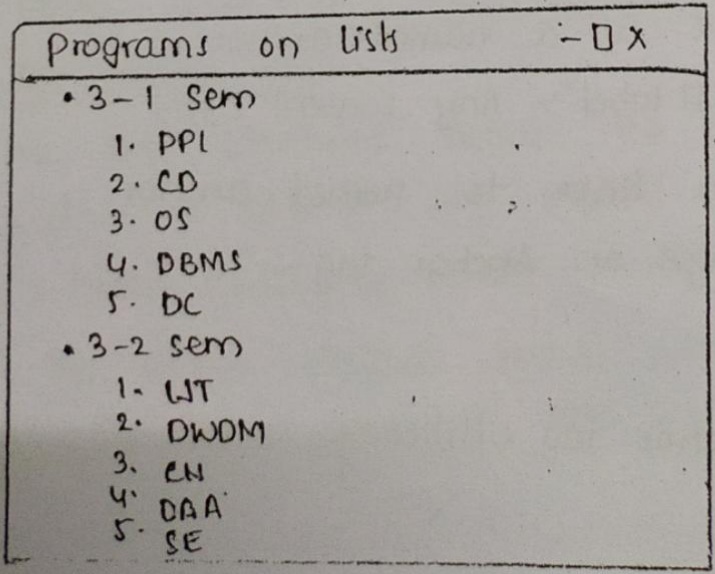
A definition list is not a list of single items. It is a list of items with a description of each item. A definition list with a <dl> tag. Each term starts with ~~<dd tag~~ <dt> tag (Definition term
Each description starts with <dd> tag (Definition Description)

Syntax:
```
<dl>
    <dt> - - - </dt>
    <dd> - - - </dd>
</dl>
```

Eg:
```
<dl>
    <dt> HTML </dt>
    <dd> It is a scripting used to represent data </dd>
</dl>
```

```
<html>
  <head>
    <title> program on list </title>
   </head>
   <body>
         <ul type="circle">
            <li> 3-1 sem </li>
               <ol* type="1">
                  <li> ppl </li>
                  <li> cd </li>
                  <li> OS </li>
                  <li> DBMS </li>
                  <li> DC </li>
               </ol>
            <li> 3-2 sem </li>
               <ol type="1">
                  <li> WT </li>
                  <li> DWDM </li>
                  <li> CN </li>
                  <li> DAA </li>
                  <li> SE </li>
               </ol>
         </ul>
      </body>
  </html>
```

Output:



```
Programs on lists                 - □ X
 • 3-1 Sem
    1. PPL
    2. CD
    3. OS
    4. DBMS
    5. DC
 • 3-2 sem
    1. WT
    2. DWDM
    3. CN
    4. DAA
    5. SE
```

Cascading Style Sheets

* Types of CSS:
  - → css1 specification was developed in 1996
  - → css2 was released on 1998.
  - → css control and changes the presentation of html documents. It is not technically HTML, but can be embedded in HTML documents.
  - → style sheets allows you to impose a standard style on a whole document, or even a whole collection of documents
  - → Style is specifed for a tag by the values of its properties.

* Types / Levels of stylesheets:-

(CSS is used to set the styles in web pages which contain HTML elements. It sets the background color, font size, font-family, color, ... etc.) property of elements in a web pages. There are three types of css which are given below:.

* Inline css
* Internal or Embedded css
* External css

* Inline css: (Inline css contains the css property in the body section attached with element is known as inline css.) This style of is specified within an HTML tag using style attributes.

<!--Example for inline-->

```
<!DOCTYPE html>
<html>
 <head>
   <title>Inline css </title>
 </head>
 <body>
      <p style="color: #009900;
             font-size: 50px;
             font-style: italic;
             text-align: center;"> CSE </p>
 </body>
</html
```

2020/8/13 14:05

CSE

*

* Internal / embedded css:
This can be used when a single HTML document must be styled uniquely. The css rule set should be within the HTML file in the head section i.e, the css is embedded within the HTML file
Example:

```
<!DOCTYPE Html>
<html>
  <head>
    <title>Internal css </title>
    <style>
       .main{
            text-align :center;
       }
       .~~GFG{~~
       .WT {
            color:#009900;
            font-size :50px;
            font-weight :bold;
       }
       .SADP
         {
           font-style :bold;
           font-size :20px;
         }
   </style>
  </head>
  <body>
     <div class="main">
     <div class="WT"> web technologies </div>
     <div class ="SADP"> Software Architecture and Design Patterns</div
     </div>
  </body>
</html>
```

Output:

* External CSS:

External css contains separate css file which contains only style property with the help of tag attributes. CSS property written in a separate file with .css extension and should be linked to the HTML document using link tag.

Example:

```
wt.css
body
 {
  background -color: powderblue;
 }
.main{
    text-align: center;
  }
.wt{
    color: #009900;
    font-size: 50px;
    font-weight: bold;
  }
.html{
     font-style: bold;
     font-size: 20px;
}
```

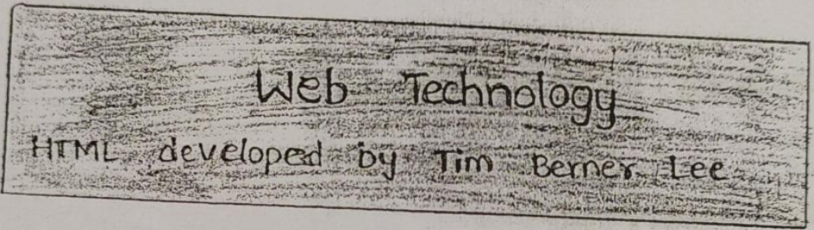
* Below is the HTML file that is making use of the created external style sheet

- link tag is used to link the external style sheet with the html webpage
- href attribute is used to specify the location of the external style sheet file.

2020/8/13 14:05

```
<!DOCTYPE html>
<html>
  <head>
    <link rel="stylesheet" href="gWt.css"/>
  </head>
  <body>
      <div   class = "main">
      <div   class = "Wt"> Web Technology </div>
      <div   id = "Html"> HTML Developed by Tim Berner Lee </div>
    </div>
  </body>
</html>
```

Output:-



Web Technology

HTML developed by Tim Berner Lee

Properties of CSS:-

Inline css has the highest priority, then comes Internal/Embedded followed by External css which has the least priority. Multiple style sheets can be defined on one page. If for an HTML tag, styles are defined in multiple style sheets then the below order will be followed.

- As Inline has the highest priority, any styles that are defined in the internal and external style sheets are overridden by Inline styles.
- Internal or Embedded stands second in the priority list and Overrides the styles in the external style sheet.
- External style sheets have the least priority. If there are no Styles defined either in ~~exte~~ inline or internal style sheet then external style sheet rules are applied for the HTML tags.

- Format depends on the level of the style sheet
- Inline:
  - Style sheet appears as the value of the style attribute
  - General form:

```
Style , "property-1: value-1;
         property-2 : value-2;
         ....
         Property_n ; value_n;
```

- Document level
  - Style sheet appears as a list of rules that are the contents of a <style> tag
  - The <style> tag must include the type attribute, set to "text/css".
  - The list of rules must be placed in an HTML comment, because it is not HTML
  - comments in the rule list must have a different form - use c comments (/*...*/)
  - General form:

```
<style form="text/css">
<!--
rule list
-->
</style>
```

- Form of the rules:

  Selector {list of property/values}

  - Each property/value pair has the form: property : value
  - pairs are separated by semicolons, just as the value of a <style> tag.

style sheets.

→ Form is a list of style rules, as in the contents of a <style> tag for document - level style sheets.

→ The CSSI specification was developed in 1996

→ CSS2 was released in 1998

→ CSSs provide the means to control and change presentation of HTML documents

→ CSS is not technically HTML, but can be embedded in HTML documents

→ Style sheets allow you to impose a standard style on a whole document, or even a whole collection of documents.

→ Style is ~~de~~ specified for a tag by the values of its properties.

Levels of Style Sheets :-

There are three levels of style sheets

- Inline:- specified for a specific occurrence of a tag and apply only to that tag. This is fine-grain style, which defeats the purpose of style sheets - uniform style.
- Document-level style sheets:- apply to the whole document in which they appear.
- External style sheets :- can be applied to any no. of documents.

Linking an external stylesheet

A <link> tag is used to specify that the browser is to fetch and use an external style sheet file.

```
<link - rel = "stylesheet" type = "text/css"
  href = "http://www.OBS.com/website.css">
</link>
```

- The user, through browser settings, may specify styles.
- Individual properties can be specified as important

2020/8/13 14:05

* CSS Selector

CSS selectors are used to select the content you want to s
selectors are the part of css rule set. CSS selectors select
HTML elements according to its id, class, type, attribute etc-
There are several different types of selectors in css.

1) CSS Element selector
2) CSS Id selector
3) CSS class selector
4) CSS Universal selector
5) CSS Group selector.

EICUG

1) CSS Element Selector.

The element selector selects the HTML element by name.

```
<!DOCTYPE html>
<html>
  <head>
   <style>
    p{
       text-align : center;
       color : blue;
     }
   </style>
  </head>
  <body>
    <p>This style will be applied on every paragraph.</p>
    <p id="para1" >cse </p>
    <p> ECE </p>
  </body>
</html>
```

o/p:

```
        This style will be applied on every paragraph

                      cse

                      ECE
```

2020/8/13 14:

The id selector selects the id attribute of an HTML element to select a specific element. An id is always unique. Within the page so it is chosen to select a single, unique element.
It is written with the (#) hash character, followed by the id of the element.

```
<!DOCTYPE html>
<html>
  <head>
    <style>
      #para1 {
            text-align : center;
             color : blue;
        }
    </style>
  </head>
<body>
   <p id="para1"> Hello cse </p>
   <p> This paragraph will not be affected. </p>
</body>
</html>
```

O/P:

```
                    Hello Csc

This Paragraph will not be affected
```

3) CSS class selector.
The class selector selects HTML elements with a specific class attribute. It is used with a period character. followed by the class name.

Note: A class name should not be started with a number

```
<html>
 <head>
  <style>
    .center {
        text-align: center;
        color : blue;
    }
  </style>
  </head>
  <body>
   <h1 class = "center"> This heading is blue and center aligned. </h1>
   <p class = "center"> This paragraph is blue and center aligned. </p>
 </body>
</html>
```

O/P.

```
          This heading is blue and Center aligned
                  This paragraph is blue and Center aligned
```

*) CSS class selector for specific element:
If you want to specify that only one specific HTML element should be affected then you should use the element name with class Selector.

```
<!DOCTYPE html>
<html>
 <head>
  <style>
   p.center {
      text-align : center;
       color: blue;
   }
  </style>
 </head>
 <body>
  <h1 class="center">This heading is not affected </h1>
   <p class="center">This paragraph is blue and center-aligned.</p>
 </body>
</html>
```

O/p :-

This heading is not affected

This paragraph is blue and Center-aligned

4) CSS Universal Selector

The universal selector is used as a wildcard character. It selects all the elements on the pages.

```
<!DOCTYPE html>
<html>
 <head>
  <style>
 * {
   color: green;
   font-size: 20px;
 }
 </style>
</head>
<body>
 <h2>This is heading </h2>
 <p>This style will be applied on every paragraph.</p>
 <p id="para1"> cse </p>
 <p>·Ece </p>
</body>
</html>
```

O/P:

This is heading

This style will be applied on every paragraph

CSE
ECE

5) CSS Group Selector

The grouping selector is used to select all the elements with th same style definitions. It is used to minimize the code. commas are used to separate each selector in grouping.

```
<!DOCTYPE html>
<html>
<head>
<style>
   h1,h2,p {
        text-align : center;
          color: blue;
      }
 </style>
 </head>
 <body>
  <h1> CSE </h1>
  <h2> ECE </h2>
  <p> This is a paragraph.</p>
 </body>
</html>
```

O/P:

```
CSE
ECE
This is a paragraph
```

→ Borders - every element has a border-style property.

- controls whether the element has a border and if so, the style of the border
- border-style values : none, dotted, dashed, and double.
- border-width - thin, medium (default), thick, or a length value in pixels.
- Border width can be specified for any of the four borders (e.g., border-top-width)
- border-color - any color
- Border color can be specified for any of the four borders (e.g., border-top-color)

→ SHOW borders.html and display

- padding - the distance between the content of an element and its border
- controlled by padding, padding-left etc.,

→ SHOW marpads.html and display.

Back-ground Images

- The background-image property.

→ SHOW back_image.html and display

- Repetition can be ~~all~~ controlled
- background-repeat property

→ possible values ; repeat (default), no-repeat, repeat-x .or repeat-y

- background-position property

→ possible values : top, center, bottom, left or right.

* When two or more ~~details~~ rules apply to the same
  rules for deciding which rule applies
- Document level
  - In-line style sheets have precedence over document styles
  - Document style sheets have precedence over external style sheets
- Within the same level there can be conflicts.
  - A tag may be used twice as a selector.
  - A tag may inherit a property and also be used as a selector.
- Style sheets can have different sources.
  - The author of a document may specify styles.

* Difference between HTML and HTML5

HTML is used to design Web pages using a markup langua This Language is used to annotate text so that a machin can understand it and manipulate text accordingly. The lang uses tags to define what manipulation has to be done on text. HTML5 is the fifth version of HTML. Many elemen are removed or modified from HTML5.
There are many differences between HTML and HTML5 whicl discussed below:

HTML	HTML5
1) It didn't support audio and video without the use of flash Player support.	1) It supports audio and video cor with the use of <audio> and <vi tags
2) It uses cookies to store tempo-rary data	2) It uses SQL databases and appl ons cache to store offline data.
3) Does not allow JavaScript to run in browser.	3) Allows Java script to run in back nd. This is possible due to JS we Worker API in HTML5
4) vector graphics is possible in HTML with the help of various technologies such as VML, Silver-light, Flash, etc..	4) vector graphics is additionally ar integral a part of HTML5 like SVG and canvas
5) It does not allow drag and drop effects	5) It allows drag and drop effec
6) Not possible to draw shapes like circle, rectangle, triangle etc.	6) HTML5 allows to draw shapes lik circle, rectangle, triangle, etc..
7) It works with all old browsers	7) It is supported by all new browse like Firefox, Mozilla, chrome, Safari etc.

8) Older Version of HTML are less mobile - friendly.	8) HTML5 Language is more mobile - friendly.
9) Doctype declaration is too long and complicated.	9) Doctype declaration is quite sim and easy.
10) Elements like nav, header were not present.	10) New element for Web structure like nav, header, footer etc--- *
11) Character encoding is long and complicated.	11) character encoding is simply and easy..
12) It is almost impossible to get true GeoLocation of user with the help of browser.	12) One can track the GeoLocation of a user easily by using JS GeoLocation API.
13) It cannot handle inaccurate syntax.	13) It is capable of handling inaccurate syntax.
14) Attributes like charset, async and ping are absent. in HTML.	14) Attributes of charset, async and ping. are a part of HTML5.

* There are many HTML elements which have been modified or removed from HTML5. Some of them are listed below:

Element	In HTML5
<applet>	changed to <object>.
<acronym>	changed to <abbr>
<dir>	changed to <ul>
<frameset>	Removed
<frame>	Removed
<noframes>	Removed
<strike>	No new tag. css is used for this.
<big>	No new tag -css is used for this
<basefont>	No new tag -css is used for this.
<font>	No new tag -css is used for this.
<center>	No new tag .css is used for this.
<tt>	No new tag .css is used for this.

Semantics :-

(10)

HTML tags are classified in two types.

* Semantics
* Non - Semantics.

Semantic elements :

Semantic elements have meaningful names Which tells about type of content. For example, header, footer, table, ... etc. HT introduces many semantic elements as mentioned below Which make the code easier to Write and understand for the developer as well as instructs the browser on how to treat it

→ Article: It contains independent content Which doesn't requir any other context.

Example, Blog Post, Newspaper Article etc.

```
<!DOCTYPE html>
<html>
    <head>
      <title> Article Tag </title>
      <style>
        h1{
          color: #006400;
          font-size: 50px;
          Text-align:left;
        }
        P{
        font-size: 25px;
        text-align: left;
        margin-top: -40px;
       }
      </style>
   </head>
   <body>
     <article>
        <h1> SIET </h1>
        <p> Srinivasa Institute of Engineering and Technology </p>
     </article>
   </body>
</html>
```

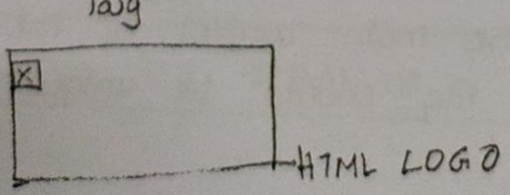
2020/8/13 14

Figure and Figcaption:-

These are used to add an image in a Web page with small description.

```
<!DOCTYPE html>
<html> <head>
      <title> Figcaption Tag </title>
      <style>
            h2 {
                  color : #006400;
                  font-size : 50px;
                  Text-align : none;
                  margin-bottom : 0px;
            }
            p {
              font-size : 25px;
               text-align : none;
               margin-top : 0px;
            }
      </style>
   </head>
   <body>
      <h2> CSE </h2>
       <figure>
           <img src="4.jpg" alt="gfg" style="width: 20%">
           <figcaption>. HTML Logo </figcaption>
       </figure>
   </body>
</html>
```

Output:



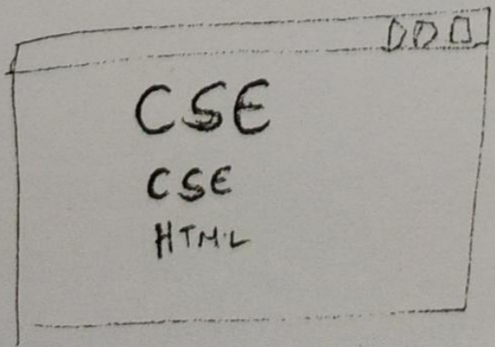
2020/8/13 14

Header:

As the name suggests, it is for the header of a section introductory of a page.. There can be mulltiple headers on a pa

```
<!DOCTYPE html>
<html>
    <head>
        <title> Header tag </title>
        <style>
            h1, h3 {
                color: #006400;
                Text-align : left;
                margin-bottom : 0px;
            }
            p{
              font-size : 25px;
               text-align : left;
               margin-top : 0px;
           }
        </style>
    </head>
    <body>
        <article>
            <header>
                <h1> CSE </h1>
                <h3> CSE </h3>
                <p> HTML </p>
            </header>
        </article>
    </body>
</html>
```

Output:



ntic Elements

<article>	Defines an article in the document
<aside>	Defines content aside from the page content
<details>	Defines additional details that the user can view or hide
<dialog>	Defines a dialog box or Window
<figcaption>	Defines a caption for a <figure> element.
<figure>	Defines self-contained content, like illustrations, diagrams, photos, code listing, etc.
<footer>	Defines a footer for the document or a section.
<header>	Defines a header, for the document or a section.
<nav>	Defines navigation links in the document.
<section>	Defines a section in the document.
<summary>	Defines a visible heading for a <details> element.
<time>	Defines a date/time.

* Form elements

<datalist>	Defines pre-defined options for input controls.
<keygen>	Defines a key-pair generator field (for forms).
<output>	Defines the result of a calculation

* Graphic Elements

<canvas>	Draw graphics, on the fly, via scripting (usually Javascript).
<svg>	Draw scalable vector graphics

* Media Elements:

<audio>	Defines sound content
<embed>	Defines containers for external applications (like plug-ins)
<source>	Defines sources for <video> and <audio>
<track>	Defines tracks for <video> and <audio>
<video>	Defines video or movie content.

Web storage

In HTML, to store user data on a local machine, we were using Javascript cookies. To avoid that HTML5 has introduced Web storage, with which websites themselves store user data on a local machine.

The advantages of Web storage, as compared to cookies, are:

- More Secure
- Faster
- stores a larger amount of data
- The stored data is not sent with every server request. It is only included when asked for. This is a big advantage of HTML5 Web storage over cookies.

There are two types of Web storage objects:

1) Local - this stores data with no expiration date.
2) Session - this stores data for one session only.

How it works:

The localStorage and sessionStorage objects create a key = value pair.

An example is: key = "Name", value = "CSE"

The syntax for using Web storage objects:

```
Storing a value:
• localStorage.setItem("key1", "value1");
• localStorage ["key1"] = "value1";
Getting a value:
• alert (localStorage.getItem ("key1");
• alert (localStorage{"key 1"]);
Remove a value:
• removeItem ("key1");
Remove all values:
• localStorage.clear();
```

2020/8/13 14:

```
    <body>
        <h1> div Tag </h1>
        <div class = "WT">
            <h1> CSE </h1>
            <p> Computer Science and Engineering </p>
        </div>
    </body>
</html>
```

Output:-

```
div Tag
WT
CSE
Computer Science Engineering
```

* Features:-

The new features of HTML5 are:

- New sets of tags such as <header> and <section>
- <canvas> element for 2D drawing.
- Local storage
- New form controls like calendar, date and time.
- New media functionality
- Geo-Location.
- HTML5 is not an official standard as yet; hence, not all browsers support it or some of its features. One of the most important reasons behind developing HTML5 was to prevent users from having to download and install multiple plugins like Silverlight and Flash.
- Semantic elements
- Form elements
- Graphic elements
- Media elements.

* Advanced features of HTML5

Geo-Location :- It is an HTML5 API that is used geographical location of a Website's use has to first permit the site to fetch his location.

- Some uses of geo-location are:
  - * public transportation Websites
  - * Taxi and other transportation Websites
  - * To calculate shopping costs on an e-commerce site
  - * Real estate Websites
  - * Online gaming

How it works :

Geo-Location works by scanning common sources of location information, which include the following:

- Global positioning system (GPS), Which is the most accurate
- Network signals - IP address, RFID, Wi-fi and Bluetooth MA addresses
- GSM / CDMA cell IDs
- User Inputs.

The API offers a function to detect geo-location support in browsers:

```
if (navigator. geolocation)
{
    // do stuff
}
```

The syntax is:

```
get_CurrentPosition (showLocation, ErrorHandler, Options);
```

- showLocation : This defines the callback method that retrieves l on information
- ErrorHandler (optional): This defines the callback method that is invoked When an error occurs in processi the asynchronous call
- (optional) options : This defines a set of options for retrieving the

1 1/N

UNIT II

JAVA SCRIPT

5.86

Introduction:-

Java Script is a quick programming languages which can be used on the client side scripting. Java Script is mainly used for performing a no.of tasks such as validating input, doing local calculations, window pop-ups etc..

Java Script originated from a language called "liveScript". It was developed at Netscape by "Brenden Eich". JavaScript was initially named as "Mocha". In late 1995, live Script became a joint venture of Netscape and Sun Micro systems and the name was changed as "JavaScript".

A Language Standard for Java Script was developed in the late 1990's by the European computer Manufacture Association (ECMA) as "ECMA - 262". It was approved by ISO as ISO - 16262. Microsoft JAvascript is also called as "JScript".

Javascript can be divided into 3 parts (i) core side
(ii) client side
(iii) Server side

Core side:- It is the heart of Language including expressions, operations, statements and subprograms.

Client side:- It is the collection of objects that controls a browser and interaction with user.

Server side:- It is the collection of objects that makes the language useful on webserver. For example, to support communication with DBMS.

* client side Javascript is a "HTML embedded scripting language and HTML document can include any no.of embedded systems script".

2020/8/13 14:42

Differences between Java and Javascript

Java	JavaScript
1) Java is strongly typed language Eg: int a=10;	1) JavaScript is dynamically typed ~~Va~~ i.e, Variables can't be determined before script is evaluated Eg: var a=10.
2) The objects in Java are static. It means the collection of data members is done at compile time	2) Javascript objects are dynamic, the no.of data members and methods of objects can change during the execution.

Benefits of JavaScript

1) JavaScript has no.of big benefits to anyone who wants to mak their website dynamic.
2) It is widely supported in Web Browser.
3) It gives easy access to the document objects and can manipulat most of them.
4) It can give interesting animation without the language download times associated with many multimedia datatypes.
5) Web surfers do not need a script special plugins to use your scripts.

Keypoints in JavaScript:-

1) Each line of code is terminated by a semi-colon.
2) Functions have parameters which are passed inside the paranthes
3) Variables are declared using "var" ~~keyboard.~~ keyword.
4) Execution of script starts with the first line of code and runs until there are no more codes
5) Java Script programs are stored with an extension of .js.
6) Java Script is case sensitive
7) Java Script is usually embedded directly into HTML pages.
8) Java script is usually an interpreted language (means that Script executes without preliminary compilation)

2020/8/13 14:42

f) Symbol:-

Symbols are new in ES6. A Symbol is an immutable primitive value that is unique. For the sake of brevity, that is the extent that this article will cover symbols.

const mySymbol = Symbol ('mySymbol');

What are objects?

Objects are not a primitive data Type.

An object is a collection of properties. These properties are stor in key/value pairs. properties can reference any type of dat including objects and/or primitive values.

```
var obj = {
            key1: 'value',
            key2: 'value',
            key3: true,
            key4: 32
            key5: {}
         }
```

Loosely Typed:-

JavaScript is a loosely Typed language. This means you do have to declare a variable's type. Javascript automatically determines it for you. It also means that a variable type can change.

We'll create a variable named car and set it equal to a String value:

var car = 'ford';

Later, we realize we want the value of car to be the ye it was made, so we change car to a "number"

car = 1998;

It works and Javascript is careless. Because JS is loosely typ we are free to change variables as we please.

2020/8/13 14:43

Example:

```
<html>
  <body>
    <script type="text/javascript">
      var a=33;
      var b=10;
      var linebreak ="<br/>";
      document.write("value of a => (a=b) =>");
      result = (a=b);
      document.write(result);
      document.write(linebreak);
      document.write("value of a=> (a+=b) =>");
      result = (a+=b);
      document.write(result);
      document.write(linebreak);
      document.write ("value of a=> (a-=b) =>");
      result =(a-=b);
      document.write(result);
      document.write(linebreak);
      document.write("value of a => (a*=b) =>");
      result = (a*=b);
      document.write(result);
      document.write(linebreak);
      document.write("value of a => (a/=b) =>");
      result = (a/=b);
      document.write(result);
      document.write(linebreak);
      document.write("value of a => (a%=b) =>");
      result = (a%=b);
      document.write(result);
      document.write(linebreak);
    </script>
  </body>
</html>
```

2020/8/13 14:44

Output:-

```
value of a => (a=b) => 10
value of a => (a+=b) => 20
value of a => (a-=b) => 10
value of a => (a*=b) => 100
value of a => (a/=b) => 10
value of a => (a%=b) => 0
```

*Miscellaneous operator:-

The two operators here that are quite useful in javascript : the conditional operator (?:) and the type_of operator.

* Conditional Operator (?:)

The conditional operator first evaluates an expression for a true or false value and then executes one of the two given statements depending upon the result of the evaluation.

S.No.	Operator	Description
1.	?: (conditional)	If condition is true? Then value X : otherwise value Y.

Example:-

```
<html>
  <body>
   <script type="text/javascript">
    var a=10;
    var b=20;
    var linebreak ="<br/>";
    document.write ("((a>b)?100:200) =>");
    result = (a>b)?100:200;
    document.write (result);
    document.write (linebreak);
    document.write(" ((a<b)?100:200)=>");
    result = (a<b)? 100:200 ;
    document.write(result);
    document.write (linebreak);
  </script>
 </body>
</html>
```

2020/8/13 14:44

Output:-

```
((a>b)?100:200) => 200
((a<b)? 100:200) => 100.
```

* typeof operator

The typeof operator is a unary operator that is placed before its single operand. which can be of any type. Its value is a string Indicating the data type of the operand.

The typeof operator evaluates to "number", "string", or "boolean" if its operand is a number, string, or boolean value and returns true or false based on the evaluation.

Here is a list of return values for the typeof operator:

Type	String returned by typeof
Number	"number"
String	"string"
Boolean	"boolean"
Object	"object"
Function	"function"
Undefined	"undefined"
Null	"object"

Example:

```
<html>
  <body>
    <script type="text/javascript">
      var a = 10;
      var b= "String";
      var linebreak = "<br/>";
      result = (typeof b == "string"? "B is string": "B is Numeric");
      document.write("Result => ");
      document.write (result);
      document.write (linebreak);
      result = (typeof a == "string" ? "A is String": "A is Numeric");
```

2020/8/13 14:44

```
        document.write("Result =>");
        document.write(result);
        document.write(linebreak);
    </script>
  </body>
</html>
```

output:-

```
Result => B is string
Result => A is Numeric
```

Set the variables to different values and different operators and the try....

* Expressions:-

All Expressions is any valid set of literals, variables, operator and expressions that evaluates to a single value. The value ma a number, a string or a logical value. conceptually, there are tw types of expressions: those that assign a value to a variable, an those that simply have a value. For example, the expression $\boxed{x=}$ is an expression that assigns x the value 7. This expression evaluates to 7. such expressions use assignment operators. on other hand, the expression 3+4. simply evaluates to 7, it not perform an assignment. The operators used in such expressio are referred to simply ~~as~~ as operators.

Javascript has the following kinds of expressions:

* Arithmetic : evaluates to a number
* String : evaluates to a character string, for example "cse" or "
* Logical : evaluates to true or false.

The special keyword "null" denotes a null value. In contrast, Variables that have not been assigned a value are undefine and cannot be used without a run-time error.

2020/8/13 14:45

Conditional Expressions:-

A conditional expression can have one of two values based on condition.

Syntax:

```
(condition) ? val 1 : val 2
```

If condition is true, the expression has the value of val1, otherwise it has the value of val2. you can use a conditional expression anywhere you would use a standard expression.

For example,

```
Status = (age>=18) ? "adult" : "mirror";
```

This statement assigns the value "adult" to the variable status if age is 18 or greater. otherwise, it assigns the value "minor" to status.

* Screen Input and output Statements:-

Inorder for a computer program to perform any useful work it must be able to communicate with the outside world. The process of communicating with the outside world is known as input/output. Javascript includes two output statements for displaying information on the computer's screen, and two input statements - one for reading in text input, the other for reading in numeric input.

The two output statements included in Javascript are:

document.write and document.writeln.

These statements are of the general form:

- document.write (expression);
- document.writeln (expression);

Where an "expression" can be a constant, a variable, a function call, or a collection of these things connected by appropriate operators. Both the document.write and document.writeln Statements display the value of an expression on the computer screen.

2020/8/13 14:45

The only difference between the two is that document.writeln starts a new line after printing the value on the screen. For this reason, document.writeln is pronounced as if it were written "document write line".

Program:

```
<html>
  <body>
   <pre>
     <script type="text/javascript">;
                document.writeln("Hello World");
     </script>
     </pre>
   </body>
<html>
```

Output:

· Hello World!

* The following program illustrates the difference between document.write and document.writeln.

Program: definition

```
// Main program
document.write ("There are");
document.write (119+1);
document.write ("Computer Science Engineering students in final year");
```

When run, it produces the following line of output.

```
There are 120 computer Science Engineering students in final year.
```

the statements of a program are executed one after another in the order, or sequence, they appear. Hence, the first statement to be executed is:

document.Write ("There are");

This statement prints the following characters on computer's display screen.

There are

2020/8/13 14:45

While it is not apparent in the line shown above, if you look closely at the statement, you will see that a space, or "blank" supposed to follow the word "are". Because this statement is a "w and not a "writeln" a new line of output was not begun after these characters were displayed. Thus, the next program output immediately follow the characters just printed, on the same line.

The position that the next character will be printed in is often indicated by the underscore symbol, _. Using this symbol, it is easy to see that a space does indeed follow the word "are" in the output.

There are

The next statement to be executed is:

document.write(119+1);

This statement contains the mathematical expression 119+1 which evaluates to the number 120. Since "write" and "writeln" statemen display the value of an expression, and not the expression its 120 will be printed next. The program's output now looks like the following, with the print position immediately following the 120.

There are 120.

It is important that you pay close attention to whether or not an expression is surrounded by quote marks when attempting to determi its value. If an expression is surrounded by quote marks then the characters between those quotes represent the value of the expr on. If the expression is not quoted, then its value must be determi by performing the operations specified in the expression. Thus,

- document.write(119+1); prints 120 but
- document.write("119+1"); prints 119+1

Continuing with the example, the third and last ~~sa~~ statement of the program

document.writeln("computer science engineering students in final yea

will now be executed to produce the following

2020/8/13 14:45

The numeric input statement.

age = parseInt(prompt("Enter your age.",0));

will prompt the user to enter his/her age by displaying a numeri entry pad on the screen with the prompt "Enter your age". Next to the input field. The number entered by the user will be stored in the variable age. If the user fails to enter a number, zero will be assigned to age as a default value.

Program:-1

```
//Variables
Var firstNum; /* NUMERIC*/
Var SecondNum; /* NUMERIC*/
// Main program
firstNum = parseFloat (prompt ("Enter a number.", 0));
Second Num = parseFloat (prompt ("Enter another number:",0));
document-write ("the sum of");
document-write (firstNum);
document-write ("and");
document-write (secondNum);
document-write ("is");
document-write (firstNum + Second Num);
document-writeln (".");
```

Program-2

```
<html>
<body>
  <pre>
   <script type = "text/javascript">
      document.write ("A");
      document.write ("B");
      document.writeln ("C");
      document.writeln ("1");
      document.writeln (2);
      document. write ("3");
  </script>
 </pre>
 </body>
</html>
```

2020/8/13 14:45

umeri
".Nex
stored
zero

* Control statements:

Conditional statements in Java script:-

conditional statements are also used to perform different actions based on different conditions. In Javascript, we have the following conditional statements.

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
}
```

```
<!-- Javascript on If statement:-->
<html>
  <head>
     <title> Example for if </title>
     <script language = "Java script">
     var n;
     n= parseInt(window.prompt("Enter n", "0"));
     if (n%2==0)
     document.write("n is even number "+n);
    </script>
  </head>
</html>
```

If-else statement: It is used to execute some code, if a condition is true and another code if the condition is false.

Syntax:
```
if (condition)
{
    code to be executed if condition is true
}
else
{
    code to be executed if condition is false
}
```

2020/8/13 14:45

```
<!-- Javascript on if-else -->
<html>
  <head>
    <title> IF - ELSE </title>
  </head>
   <body>
      <script type = "text/javascript">
       var n;
       n = parseInt (window.prompt ("Enter n value", "0");
       if (n % 2 = = 0)
           document.write ("n is even" +n);
        else
           document.write ("n is odd" +n);
      </script>
    </body>
</html>
```

If - else - If statement:- It is used to select one of several blocks of code to be executed.

Syntax:

```
       if (condition 1)
       {
          code to be executed if condition1 is true
       }
       else if (condition 2)
       {  code to be executed if condition2 is true
       }
       else
        { code to be executed if condition1 and
           condition2 are false
        }
```

```
<!-- program on if-else -if -->
<html>
   <head>
    <title><IF-ELSE - IF</title> </head>
    <body>
       <script type = "text/javascript">
        var marks;
```

2020/8/13 14:45

Java Script conditional loops:-

Javascript provides three kinds of conditional loops (i) While (ii) do-while (iii) for

While loop:-

The While loop checks for the condition being tested and if it is satisfied only then executes the code. It is an entry controlled loop.

Syntax:
```
While (expression)
{
    Statements;
}
```

```
<!-- program on while loop -->
<html>
  <head>
    <title> WHILE LOOP </title> </head>
    <body>
        <script type = "text/ Javascript">
          var i=0;
          While (i<=5)
          {
              document.writeln ("The number is" +i);
              document.writeln ("<br>");
              i++;
          }
        </script>
    </body>
</html>
```

do-While loop :- It first executes the code and then checks for the condition being tested. It means, it executes the code atleas once regardless of whether the condition being tested is success or not. It is an exit controlled loop.

Syntax:
```
do
{
  Statements;
} while (test_conditions);
```

2020/8/13 14:46

```
<!-- program on do-while -->
<html>
    <head>
        <title> DO-WHILE </title></head>
        <body>
            <script type = "text/javascript">
                var n, i = 1;
                n= parseInt(window . prompt ("Enter n:", "0"));
                do
                {
                    document.write(+i);
                    i++;
                }while (i<=n);
            </script>
        </body>
</html>
```

for loop :- The for loop execute in interaction, usually, incrementing or decrementing the loop index.

syntax:

```
for (initialization ; condition ; Increment / Decrement)
{
        statements ;
}
```

→ The for statement can be represented by an equivalent while statement, with initialization, loop continuation test and decrement/increment.

placed as follows:

```
Syntax:  Initialization ;
         While (loop continuation Test)
         {
             Statements ;
             Increment / Decrement ;
         }
```

```
<!-- program on for loop -->
<html>
   <head>
      <title> For Loop </title> </head>
      <body>
         <script type= "text/ javascript">
           var n;
           n= parseInt (window. prompt ("Enter n:");
           for (var i=1 ; i<=n ; i++)
           {
               document. write (+i);
           }
         </script>
      </body>
</html>
```

Break statement And Continue Statement :-

The break and continue statements alter the flow of control in the loop. The break statement when executed in a while, for, do-while, switch statements structures causes immediate exit from that structure. Execution continues with the first statement after the loop.

The continue statement when executed in a while, do-while, for loop skips the remaining statements in the body of that loop and proceeds with the next iteration of the loop.

Arrays in Javascript:

An array is a group of memory locations that all have the same name and are normally of the data types.

In Javascript, the array is slightly different because it is a spec type of object and has functionality which is not normally availa in other languages.

The data inside the array is ordered, because elements are added and accessed in a particular order, but is not sorted. There is no relatio ship with the way the data is hold and any external meaning it has. Basic operations that are performed on Arrays are creation, Accessing, Addition, searching, Removing. Refer Array objects.

2020/8/13 14:46

* / object creation

Java Script Native objects :-

Java Script has several built-in or native objects. These obje
are accessible anywhere in your program and will work the
Way in any browser running in any operating system.

Here is a list of all important JavaScript Native objects.

* JS Math object
* JS String object
* JS Date object
* JS Boolean object
* JS Document object
* JS Window object
* JS Number object
* JS Array object
* JS RegExp object

JavaScript (JS) uses objects to perform many tasks and the
it is referred to as object based programming languages.

The objects in JavaScript are classified into different types as
follows:

(i) Math object :-

The math object allows the programmers to perform mixing
common mathematical calculations. The object method are call
by writing the name of method. The argument list are writ
in the method paranthesis.

Eg. document.writeln(math.sqrt(64));

Some of the math object methods are listed as follows:

2020/8/13 14:46

Methods	Desoription.
abs(x)	Absolute value of x
ceil(x)	Round x to the smallest integer not less than x
cos(x)	Trignometric cosine of x (x is in radians).
exp(x)	Exponential method to the power of x.
floor(x)	Round x to largest integer not greater than x
log(x)	Natural logarithm of x (base e).
max(x,y)	larger value of x and y
min(x,y)	Smaller value of x and y
pow(x,y)	x raised to the power of y
round(x)	Round x to the closest integer
sqrt(x)	square root of x.

(ii) String object:-
A string is a series of characters. A string is a object of string A string is written in double quotations as example "write your string here".
The some of string objects are listed as below.

Methods	Description.
Char At(index)	Returns a string containing the characters at the specified index. If there is no character at that index. char At returns ~~as~~ an empty string.
concat(string)	concatenates the arguments to end of string that invokes the method.
tolowercase()	used to convert characters / alphabets into lower case letters.
touppercase()	used to convert characters / alphabets into upper case letters.

2020/8/13 14:46

(iv) Boolean object :- When a javascript program requires a boolean value , Javascript automatically creates a boolean object to store the value . Javascript programmers can create boolean objects explicitly with the statement

var b= new Boolean (Boolean value);

Methods	Description
to String ()	returns the String value true if the value of boolean object is true otherwise false
value of()	returns the value true if the value of boolean object is true otherwise false.

(v) Number Object :- javascript automatically creates a number object to store the value when required . Javascript programmers can create number objects explicitly with statements

var n= new number (numeric_value);

Method	Description
IS NAN()	It is used to find out the given argument is number or not Eg: isNAN (a) ⇒ returns false. isNAN (2) ⇒ returns true.
parseInt()	used to convert string to integer
parse Float()	used to convert string to floating number
is Finite()	It returns true if the value is finite otherwise it returns false.
eval()	Used to calculate final value of expression eval (2*4*8) ⇒ returns 64.

(vi): Document object :- Javascript provides the document object for manipulating the document that currently visible in browser window.

2020/8/13 14:47

Methods	Description.
document. write()	Write string to HTML document as HTML code
document. writeln()	Write string to HTML, document as HTML code and adds a new character at the end.
document. cookie()	This property is a string containing the values of all cookies stored on users computer for current document.
document. last-Modified()	This property gives the date and time that this document was modified.

(Vii) Window object:- It represents a browser Window. A Window object is created automatically with every instance of a <body or <frame set> Tag.

Objects	Properties	Methods	Event Handlers
Window	default status frames openers parent Scroll Self Status top Window	alert blur close confirm focus open prompt clear time out set time out	on load on unload on blur on focus.
Frame	default Status frames. opener parent Scroll Self status top. Window	alert blur Close Confirm focus open prompt clear timeout set timeout	None.

2020/8/13 14:47

Editing JavaScript:-

You will need any special hardware or software to write Javasc
you can just use Notepad or any other text Editor. Javascript
run on any webserver any where! The system contains a browser
such as Internet Explorer, NetScape Navigator etc..

Basic Structure of JavaScript:

```
<html>
  <head>
    <title> A Simple Javascript structure </title>
    < Script type = "text/JavaScript">
      //Javascript code goes here ....
      // Single line comment in Javascript.
    </script> </head>
~~</html>~~
   <body>
    <script type = "text/Javascript">
     //Java script code goes here...
     /* Multi - line comments */
    </script>
   </body>
</html>
```

Confirm box:

A confirm box is often used if you want the user to verify or accept something. When a confirm box pops up, the user Will have to click either "OK" or "cancel" to proceed.

Syntax : confirm ("sometext")

Status Bar:

The status bar is the grey bar along the bottom of the web
browser where information like how much the page has loaded and the URL which a link is pointing to appear.

* Objects :-

Javascript is an object oriented programming (OOP) language. A programming language can be called object-oriented if it provides four basic capabilities to developers.

- Encapsulation: The capability to store related information, whether data or methods, together in an object.
- Aggregation: The capability to store one object inside another object.
- Inheritance :- The capability of a class to rely upon another class (or no. of classes) for some of its properties and methods.
- Polymorphism:- The capability to write one function or method that works in a variety of different ways.

Objects are composed of attributes. If an attribute contains a function, it is considered to be a method of the object, Otherwise the attribute is considered a property.

Object properties:-

Object properties can be any of the three primitive data types, or any of the abstract data types, such as another object. Object properties are usually variables that are used internally in the object's methods, but can also be globally visible variables that are used throughout the page.

The Syntax for adding a property to an object is:-

```
objectName . objectProperty = propertyValue ;
```

For example, the following code gets the document title using the "title" property of the "document" object.

```
var str = document. title ;
```

Object Methods

Methods are the functions that let the object do something or let something be done to it. There is a small difference between a method and a function - at a function is a standalone unit of statements and a method is attached to an object and can be referenced by the "this" keyword.

Methods are useful for everything from displaying the contents of the object to the screen to performing complex mathematical operations on a group of local properties and parameters.

For example, Following is a simple example to show how to use the write() method of document object to write any content on the document.

```
document.write("This is test");
```

<u>User defined objects</u>:-

All user defined objects and built-in objects are descendants of an object called "object".

<u>The new operator</u>:-

The new operator is used to create an instance of an object. To create an object, the new operator is followed by the constructor method.

In the following example, the constructor methods are object(), Array(), and Date(). These constructors are built-in Javascript functions.

```
Var books = new Array("C++", "perl", "Java");
Var day = new Date("August 15, 1947");
```

<u>The object() constructor</u>:-

A constructor is a function that creates and initializes an object. Javascript provides a special constructor function called object() to build the object. The return value of the object() constructor is assigned to a variable.

The variable contains a reference to the new object. The properti assigned to the object are not variables and are not defined With the "var" keyword.

Example 1 :-

The following example demonstrates how to create an object.

```
<html>
  <head>
    <title> user-defined objects </title>
    <script type="text/Javascript">
      var book = new object();   //create an object.
      book.subject = "perl";   //Assign properties to the object.
      book.author = "Mohtashim";
    </script>
  </head>
  <body>
    <script type="text/javascript">
      document.Write ("Book name is :" + book.subject + "<br>");
      document.Write ("Book author is :" + book.author + "<br>");
    </script>
  </body>
</html>
```

Output:

```
Booke : name is : perl
Book author is : Mohtashim
```

Example 2:

This example demonstrates how to create an object with a user-defined function. Here "this" keyword is used to refer to the object that has been passed to a function.

```
<html>
  <head>
    <title> User-defined functions </title>
    <script type="text/javascript">.
      function book (title, author) {
        this.title = title;
        this.author = author;
  }
  </script>
 </head>
   <body>
      <script type = "text/javascript">
        var myBook = new book ("perl", "Mohtashim");
        document.write ("Book title is : " + myBook.title + "<br>");
        document.write ("Book author is : " + myBook.author + "<br>"
      </script>
   </body>
</html>
```

Output:

```
Book title is : perl
Book author is : Mohtashim
```

Defining Methods for an object

The previous examples demonstrate how the constructor creates the object and assigns properties. But we need to complet the definition of an object by assigning methods to it.

Example:

The following example, it shows how to add a function along with an object.

```
<html>
  <head>
    <title> user-defined objects </title>
      <script type = "text/javascript">
      // Define a function Which will work as a method.
      function addPrice (amount) {
          this.price = amount;
      }
```

```
        this.title = title;
        this.author = author;
    }
    </script>
    </head>
      <body>
        <script type ="text/javascript">
          var myBook = new book ("perl", "Mohtashim");
          document.write ("Book title is :" + myBook.title + "<br>");
          document.write ("Book author is :" + myBook.author + "<br>"
        </script>
      </body>
  </html>
```

Output:

```
Book title is : perl
Book author is : Mohtashim
```

Defining Methods for an object

The previous examples demonstrate how the constructor creates the object and assigns properties. But we need to complet the definition of an object by assigning methods to it.

Example:

The following example, it shows how to add a function along with an object.

```
<html>
  <head>
    <title> user-defined objects </title>
      <script type ="text/javascript">
      // Define a function which will work as a method.
     function addPrice (amount) {
          this.price = amount;
     }
```

Example:

```
<html>
 <head>
  <title> user - defined objects </title>
   <script type = "text / javascript">
    // Define a function which will work as a method
    function addPrice (amount) {
      with (this) {
        price = amount;
      }
    }
   function book (title, author) {
      this.title = title;
      this.author = author;
      this. price = 0;
      this. addPrice = addPrice ; // Assign that method as property.
   }
  </script>
 </head>
  <body>
   <script type = "text /javascript">
    var myBook = new book ("perl", "Mohtashim");
    myBook . addPrice (100);
     document.write ("Book title is :" + myBook.title + "<br>");
    document. write ("Book author is:" + myBook.author + "<br>");
    document. write ("Book Price is:" + myBook.price + "<br>");
   </script>
 </body>
</html>
```

Output:

```
Book title is : Perl
Book author is : Mohtashim
BOOK price is : 100
```

2020/8/13 14

* Primitive Types:-

Two kinds of Data:

In Javascript there are two different kinds of data: primitives, and objects. A primitive is simply a data type that is not an object, and has no methods.

In JS, there are 6 primitive data types:

a) Boolean b) Number c) String d) Null e) Undefined f) Symbol.

a) Boolean:

A boolean represents only one of two values: true or false. Thin of a boolean as an on/off or a yes/no switch.

```
var boo1 = true;
var boo2 = false;
```

b) Number.

There is only one type of Number in Javascript. Number can be with or without a decimal point.. A number can also be +infini -infinit +∞, -∞. and NAN (not a number).

```
var num1 = 32;
var num2 = +Infinity;
```

c) String.

Strings are used for storing text. Strings must be inside of eithe double or single quotes. In JS, Strings are immutable (they cannot be changed)

```
var str1 = 'hello, it is me';
var Str2 = "hello, it is me";
```

d) NULL:

Null has one value: null. It is explicitly nothing.

```
var nothing = null;
```

e) Undefined.

A variable that has no value is undefined.

```
var testvar;
console.log(testvar); // undefined.
```

2020/8/13 14:43

* Operators:

What is an operator?

Let us take a simple expression 4+5 is equal to 9. Here 4 and 5 are called "operands" and '+' is called the "operator". JavaScript supports the following types of operators.

* * Arithmetic operators
* * Comparison operators
* * Logical (or Relational) operators
* * Assignment operators
* * Conditional (or Ternary) operators
* * Bitwise operators.

* Arithmetic operators:

JavaScript supports the following arithmetic operators -
Assume variable A holds 10 and variable B holds 20, then -

S.NO	Operator	Description
1.	+	Adds two operands. Ex: A+B Will give 30
2.	−	Subtracts the second operand from first Ex: A-B Will give -10.
3.	*	Multiply both operands. Ex: A*B gives 200
4.	/	Divide the numerator by the denominator Ex: B/A Will give 2.
5.	%	Outputs the remainder of an integer division. Ex: B%A Will give 0.
6.	++	Increases an integer value by one Ex: A++ Will give 11
7.	--	Decreases an integer value by one. Ex: A-- Will give 9.

Note:-
Addition operator (+) works for Numeric as well as Strings e.g., "a"+10 . Will give "a10".

2020/8/13 14:43

Example:-

```
<html>
  <body>
    <script type="text/javascript">
      var a = 33;
      var b = 10;
      var c = "Test";
      var linebreak = "<br/>";
      document.write ("a+b=");
      result = a+b;
      document.write (result);
      document.write (linebreak);
      document.write ("a-b=");
      result = a-b;
      document.write (result);
      document.write (linebreak);
      document.write ("a/b =");
      result = a/b;
      document.write (result);
      document.write (linebreak);
      document.write ("a%b=");
      result = a%b;
      document.write (result);
      document.write (linebreak);
      document.write,("a+b+c=");
      result = a+b+c;
      document.write (result);
      document.write (linebreak);
      a= ++a;
      document.write ("++a=");
      result = ++a;
      document.write (result);
      document.write (linebreak);
      b= --b;
      document.write ("--b=");
      result = --b;
      document.write (result);
      document.write (linebreak);
    </script> </body> </html>.
```

2020/8/13 14

output:

```
a+b = 43
a-b = 23
a/b = 3.3
a%b = 3
a+b+c = 43Test
++a = 35
--b = 8
```

* **Comparison Operators**

Assume variable A holds 10 and variable B holds 20, then.

S.No	Operator	Description
1.	== (Equal)	checks if the value of two operands are equal or not, if yes, then the condition becomes true. Ex. (A==B) is not true
2.	!= (Not Equal)	checks if the value of two operands are equal or not, if the values are not equal, then the condition becomes true. Ex: (A!=B) is true.
3.	> (Greater than)	checks if the value of the left operand is greater than the value of the right operand, if yes, then the condition becomes true. Ex: (A>B) is not true.
4.	< (less than)	checks if the value of the left operand is less than the value of the right operand, if yes, then the condition becomes true. Ex: (A<B) is true
5.	>= (Greater than or equal to)	checks if the value of the left operand is greater than or equal to the value of the right operand, if yes, then the condition becomes true. Ex: (A>=B) is not true.
6.	<= (less than or equal to)	checks if the value of the left operand is less than or equal to the value of the right operand, if yes, then the condition becomes true. Ex: (A<=B) is true.

Example:-

```
<html>
 <body>
 <Script type="text/javascript">
   var a=10;
   var b=20;
   Var linebreak="<br/>";
   document.write("(a==b)=>");
   result=(a==b);
   document. write(result);
   document. write(~~break~~ linebreak);
   document. write("(a<b)=>");
   result=(a<b);
   document.write(result);
   document. write(linebreak);
   document. write("(a>b)=>");
   result=(a>b);
   document write(result);
   document. write(linebreak);
   result = (a!=b);
   document.write(result);
   document. write(linebreak);
   document.write("(a>=b)=>");
   result=(a>=b);
   document. write(resull);
   document. write(linebreak);
   document. write("(a<=b)=>");
   result=(a<=b);
   document. write(result);
   document. write(linebreak);
 </script>
 </body
</html>
```

Output

```
(a==b)=>false
(a<b)=> true
(a>b)=> false
(a!=b)=> true
(a<=b)=> true
(a>=b)=>false
```

* Logical Operators :-

Assume variable A holds 10 and variable B holds 20, then.

S.NO	Operator	Description
1.	&& (Logical AND)	If both the operands are non-zero, then the condition becomes true. Ex. (A&&B) is true.
2.	|| (logical OR)	If any of the two operands are non-zero, then the condition becomes true. Ex. (A||B) is true
3.	! (logical NOT)	Reverses the logical state of its operand. If a condition is true, then the logical NOT operator will make it false. Ex. !(A&&B) is false.

Example:

```
<html>
  <body>
   <script type="text/javascript">
     var a=true;
     var b=false;
     var linebreak ="<br/>";
     document.write(" (a&&b)=> ");
     result = (a&&b);
    document.write(result);
    document.write(linebreak);
    document.write("(a||b)=> ");
    result =(a||b);
    document.write(result);
    document.write(linebreak);
    document.write("!(a&&b)=>");
    result = (!(a&&b));
    document.write(result);
    document.write(linebreak);
  </script>
 </body>
</html>
```

Output:

(a&&b) => false

(a||b) => true

!(a&&b) => true

*Bitwise operators

Assume Variable A holds 2 and Variable B holds 3 then,

S.NO	Operator	Description.
1.	& (Bitwise AND)	It performs a boolean AND operation on each bit of its integer arguments. Ex. (A&B) is 2.
2.	| (Bitwise OR)	It performs a boolean OR operation on each bit of its integer arguments. Ex. (A|B) is 3.
3.	^ (Bitwise XOR)	It performs a boolean exclusive OR operation on each bit of its integer arguments. Exclusive OR means that either operand one is true or operand two is true, but not both.
4.	~ (Bitwise NOT)	It is a unary operator and operates by reversing all the bits in the operand. Ex. (~B) is -4.
5.	<< (Left shift)	It moves all the bits in its first operand to the left by the no. of places specified in the second operand. New bits are filled with zeros. Shifting a value left by one position is equivalent to multiplying it by 2, shifting two positions is equivalent to multiplying by 4, and so on. Ex. (A<<1) is 4.
6.	>> (Right shift)	Binary Right shift operator. The left operand's value is moved right by the no. of bits specified by the right operand. Ex: (A>>1) is 1.
7.	>>> (Right shift with zero).	This operator is just like the >> operator, except that the bits shifted in on the left are always zero. Ex. (A>>>1) is 1.

2020/8/13 14:44

Example:

```
<html>
  <body>
   <script type="text/javascript">
      var a=2; // Bit presentation 10
      var b=3; // Bit presentation 11.
      var linebreak ="<br/>";
        document.write("(a&b)=>");
      result = (a&b);
      document.write(result);
      document.write(linebreak);
      document.write("(a|b)=>");
      result = (a|b);
      document.write(result);
      document.write(linebreak);
      document.write("(a^b)=>");
      result = (a^b);
      document.write(result);
      document.write(linebreak);
      document.write("(~b)=>");
      result = (~b);
      document.write(result);
      document.write(linebreak);
      document.write("(a<<b)=>");
      result = (a<<b);
      document.write(result);
      document.write(linebreak);
      document.write("(a>>b)=>");
      result = (a>>b);
      document.write(result);
      document.write(linebreak);
   </script>
  </body>
</html>
```

Output:

2020/8/13 14:

Output:-
```
(a&b) => 2
(a|b) => 3
(a^b) => 1
(~b) => -4
(a<<b) => 16
(a>>b) => 0
```

* Assignment operators :-

S.NO	Operator	Description
1.	=(simple assignment)	Assigns values from the right side operand to the left side operand. Ex: C=A+B Will assign the value of A+B into C.
2.	+=(Add and assignment)	It adds the right operand to the left operand a assigns the result to the left operand. Ex: C+ is equivalent to C=C+A.
3.	-=(subtract and assignment)	It subtracts the right operand from the left op nd and assigns the result to the left opera Ex: C-=A is equivalent to C=C-A.
4.	*=(multiply and assignment)	It multiplies the right operand With the left operand and assigns the result to the left Operand. Ex: C*=A is equivalent to C=C*A.
5	/=(Divide and assignment)	It divides the left operand With the right ope and assigns the result to the left operand. Ex: C/=A is equivalent to C=C/A.
6.	%=(Modules and assignment)	It takes modulus using two operands and assi the result to the left operand. Ex. C%=A is equivalent to C=C%A.

Note:
Same logic applies to Bitwise operands so they Will become lik <<=, >>=, >>=, &=, |= and ^=.

2020/8/13 14:44

There are 120 computer science engineering students in final ye

Javascript provides two input instructions - one for reading in
data, and one for reading in numeric data.

The general format of the text input statement is:

variable = prompt (message, default value);

In this statement, variable will be replaced with an actual prog
variable that has been declared to be of type "text". This var
will hold the data entered by the user. The message field w
be replaced by a text string that contains a brief message
designed to prompt the user to enter the appropriate value. Th
defaultvalue field is used to specify a default text string that
be assigned to the variable in case the user fails to enter an
data when prompted to do so.

Thus, the text input statement;

firstName = prompt("please enter your first name.", "unknown");

Will prompt the user to enter his or her first name by displayin
an input dialog box on the screen width the prompt "please
enter your first name." next to the input field. whatever char
the user types in will be stored in the variable firstName. If t
user simply dismisses the input dialog without entering any char
firstName will be assigned the text string "unknown" inorder to
indicate that the user's first name is unknown.

The Javascript numeric input statement is similar. It has the fo

Variable = parseInt (prompt (message, defaultValue));

The most obvious difference between Javascript's text input staten
and its numeric input statement is that the numeric input statem
include the phrase "parseFloat()". This phrase insures that walton
Javascript will simply a supply a numeric entry pad to the enc
user so that only numeric data can be entered. In addition, th
statement requires that variable be replaced with a program var
declared to be of type "numeric". Also, While the message will conti
to be a text string, the defaultValue must be a numeric constan
rather than a text constant.

2020/8/13 14:4

```
marks = parseInt (window. prompt ("enter marks :"));
if (marks >70)
{
   document.write ("Distinction");
else if (marks >50 && marks <=70)
   document.write ("First Class");
else if (marks >=40 && marks <=50)
   document.write ("second class");
else
   document.write ("fail");
</script>
</body>
</html>
```

Switch Statement :- It is used to select one of many blocks of code to be executed.

Syntax:

```
switch(op)
{
   case value1 : statement1 ; break;
   case value2 : Statement2 ; break;
   :
   case value n : statement n ; break;
   default : statement ;
```

```
<!--program on switch-case-->
<html>
 <head>
  <title> SWITCH </title> </head>
  <script type="text/javascript">
   var choice, n;
   choice = window.prompt ("Enter choice [1-4]", 'choice');
   n=parseInt (choice);
   switch(n)
   {
      case 1 : document.writeln ("Grade A"); break;
      case 2 : document.writeln( "Grade B"); break;
      case 3 : document.writeln("Grade C"); break;
      case 4 : document.writeln( "Grade D"); break;
      default : document.writeln ("Individual Invalid choice");
   }
</script> </body> </html>
```

2020/8/13 14:4

Methods	Description.
to string ( )	returns the same string as the source string
value of ( ) .	returns the same string as the source s
pop( )	This function removes the last element fro the array and in doing so reduces the no elements in array by one.
push( )	Adds a list of items onto the end of the arr
reverse( )	The function swaps all elements in the arra so that which was the first is last and v versa.
shift( )	Removes the first element of an array an in so doing shortens its length by one.
join (string)	used to have all of elements in array joi together as a string.

```
<!-- program to calculate the length of a string -->
<html>
  <head>
    <title> length of a string </title> </head>
    <body bgcolor = "#999fff">
     <script type = "text/javascript">
       var p = Window. prompt ("Enter string:")>
        var x= p.length;
       document. writeln ("<b><h3>output: </h3><b>")>
       document. writeln (" length of string : " +x);
       document. close();
    </script>
  </body>
</html>
```

2020/8/13 14:47

(iii) Date object :-

Methods	Description
Date()	returns a data object
getDate()	returns the date of date object (1-31)
getDay()	returns the day of the date object (from 0-6' where 0 = Sunday 1= Monday etc--)
getMonth()	returns the month of the date object (from 0-11 where 0= january 1 = Febraury etc..)
getFullYear()	returns the year of date object (four digits)
getYear()	returns the year of date object (0-99)
getHours()	returns the hours of date object (0-23)
getMinutes()	returns the minutes of the date object (0-59)
getSeconds()	returns the seconds of the date object (0-59)

```
<! -- program on date object-->
<html>
  <head>
     <title> Date object </title>
  </head>
  <body onload ="Date1()">
    <script type = "text/javascript">
     function Date1()
     {
         var today = new Date();
         var yesterday = new Date();
         var diff = today. getDate()-1;
         yesterday . setDate (diff);
         document. write (" Today's date:" + today);
         document. write ("<br>");
         document. write ("yesterday's date:" +yesterday);
     }
    </script>
  </body>
</html>
```

2020/8/13 14:47

Objects	Properties	Methods	Event Handlers
History	length Forward go	back	none
Navigator	appCode Name app Name app version MIME Types plugins User Agent	Java Enabled	None
Location	hash host host name href pathname por Protocol Search	reload replace	None

(viii) Array Object:-

It stores a fixed-size sequential collection of elements of the same type. An array is used to store a collection of data, but it is often more useful to think of an array as a collection of variables of the same type.

Syntax:

Syntax to create an Array object -

```
var Fruits = new Array ("apple", "orange", "mango");
```

The Array parameter is a list of strings or integers. When you specify a single numeric parameter with the Array constructor, you specify the initial length of the array. The maximum length allowed for an array is 4,294,967,295.

You can create an array by simply assigning values as follows:

2020/8/13 14:47

var Fruits = ["apple", "orange", "mango"];
You will use ordinal numbers to access and to set values. inside an array as follows:
fruits[0] is the first element
fruits[1] is the second element
fruits[2] is the third element.

Array Properties:

S.No	Property	Description
1.	constructor	Returns a reference to the array function that created the object.
2.	Index	The property represents the zero-based index of the match in the string.
3.	Input	This property is only present in arrays created by regular expressions matches.
4.	length	Reflects the no.of elements in an array
5.	prototype	The prototype property allows you to add properties and methods to an object.

* Array Methods:

list of the methods of the Array object along with their description

S.NO	Method	Description
1.	concat()	Returns a new array comprised of this array joined with other array(s) and/or value(s)
2.	every()	Returns true if every element in this array satisfies the providing testing function
3.	filter()	Creates a new array with all of the element of this array for which the provided filtering function returns true

S.No	Method	Description
4	forEach( )	Calls a function for each element in the array.
5.	indexOf( )	Returns ~~true~~ if the first (least) index of an element within the array equal to the speci value, or -1 if none is found.
6.	join ( )	Joins all elements of an array into a String
7.	lastIndexOf( )	Returns the last (greatest) index of an elem within the array equal to the Specified val or -1 if none is found.
8.	map( )	Creates a new array with the results of callin a provided function on every element in thi array.
9.	pop( )	Removes the last element from an array and returns that element.
10.	push( )	Adds one or more elements to the end of an a and returns the new length of the array.
11.	reduce( )	Apply a function simultaneously against two values of the array (from left-to-right) as to reduce it to a single value
12.	reduceRight( )	Apply a function simultaneously against two values of the array (from right-to-left) as to reduce it to a single value.
13.	reverse( )	Reverses the order of the elements of an arra -- the first becomes the last, and the last becomes the first.
14.	shift( )	Removes the first element from an array and returns that element.
15.	slice( )	Extracts a section of an array and returns a new array.

S.NO	Method	Description.
16.	Some()	Returns true if atleast one element in this array satisfies the provided testing function.
17	toSource()	Represents the source code of an object.
18.	sort()	sorts the elements of an array
19.	Splice()	Adds and/or removes elements from an array.
20.	toString()	Returns a string representing the array and its elements
21.	unshift()	Adds one or more elements to the front of an array and returns the new length of the array.

* JavaScript RegdExp object:

A regular expression is an object that describes a pattern of characters.

The Javascript RegdExp class represents regular expressions, and both String and RegdExp define methods that use regular expressions to perform powerful pattern-matching and search-and-replace functions on text.

Syntax:-

A regular expression could be defined with the RegdExp() constructor, as follows:-

```
var pattern = new RegdExp(pattern, attributes); or simply
var pattern = /pattern/attributes;
```

Here is the description of the parameters:

* pattern: A String that specifies the pattern of the regular expression or another regular expression.

* Attributes: An optional string containing any of the "g", "i", and "m" attributes that specify global, case-insensitive and multi-line matches, respectively.

• **Brackets:**

Brackets ([]) have a special meaning when used in the context of regular expressions. They are used to find a range of character.

S.No	Expression	Description
1.	[...]	Any one character between the brackets
2.	[^...]	Any one character not between the brackets.
3.	[0-9]	It matches any decimal digit from 0 to 9
4.	[a-z]	It matches any character from lowercase 'a' through lowercase 'z'.
5.	[A-Z]	It matches any character from uppercase 'A' through uppercase 'Z'.
6.	[a-Z]	It matches any character from lowercase 'a' through uppercase 'Z'.

The ranges shown above are general; you could also use the range [0-3] to match any decimal digit ranging from 0 through 3, or the range [b-v] to match any lowercase character ranging from b through v.

* **Quantifiers:**

The frequency or position of bracketed character sequences and single character can be denoted by a special character. Each special character has a specific connotation. The +, *, ?, an $ flags all follows a character sequence.

S.No	Expression	Description
1.	p+	It matches any string containing one or more p's
2.	p*	It matches any string containing zero or more p's
3.	p?	It matches any string containing at most one p.
4	p{N}	It matches any string containing a sequence of N p's.
5.	p{2,3}	It matches any string containing a sequence of two or three p's
6.	p{2,}	It matches any string containing a sequence of atleast two p's
7.	p$	It matches any string with p at the end of it.
8.	^p	It matches any string with p at the beginning of it

Following examples explain more about matching characters.

S.No	Expression	Description.
1.	[^a-zA-Z]	It matches any string not containing any of the characters ranging from 'a' through 'z' and 'A' through 'Z'
2	p.p	It matches any string containing p, followed by any character, inturn followed by another p
3.	^.{2}$	It matches any string containing exactly two characters.
4.	<b>(.*)</b>	It matches any string enclosed within <b> and </b>
5.	p(hp)*	It matches any string containing a "p" followed by zero or more instances of the sequence "hp"

*Literal characters

S.No	character	Description
1.	AlphaNumeric	Itself
2.	\0	The Null character (\u0000)
3.	\t	Tab (\u009)
4.	\n	Newline (\u000A)
5.	\v	Vertical tab (\u000B)
6.	\f	form feed (\u000C)
7.	\r	carriage return (\u000D)
8.	\xnn	The latin character specified by the hexadeci mal number nn; for example, \x0A is the same as \n.
9.	\uxxxx	The unicode character specified by the hexad mal number xxxx; for example, \u0009 is same as \t
10.	~~\cX~~ \cX	The control character ^X; for example, \cJ i equivalent to the newline character \n.

*Meta characters

A metacharacter is simply an alphabetical character precede backslash that acts to give the combination a special mea For instance, you can search for a large sum of money u the '\d' metacharacter : /([\d]+)000/, Here \d will sear for any string of numerical character.

The following table lists a set of metacharacters which can used in PERL style Regular Expressions.

* RegExp properties:-

S.No	Property	Description
1.	constructor	specifies the function that creates an object's prototype
2.	global	Specifies if the "g" modifier is set
3.	ignore_case	specifies if, the "i" modifier is set
4.	lastIndex	The index at which to start the next match
5.	multiline	Specifies if the "m" modifier is set.
6.	Source	The text of the pattern

RegExp Methods:-

S.No	Methods	Description
1.	exec()	Executes a search for a match in its string parameter
2.	test()	Tests for a match in its string parameter
3.	toSource()	Returns an object literal representing the specified object; you can use this value to create a new object.
4.	toString()	Returns a string representing the specified object

Example:

```
<! DOCTYPE html>
<html>
<body>
<p> click the button to do a global search for the numbers 1 to 4
                                                    in a string </p>
 <button onclick ="myFunction()"> Try it </button>
 <p id= "demo" ></p>
```

* JavaScript Arrays

In Javascript, array is a single variable that is used to store different elements. It is often used when we want to store list of elements and access them by a single variable. Unlike most languages where array is a reference to the multiple variable, In JavaScript array is a single variable that stores multiple elements.

Declaration of an Array

There are basically two ways to declare an array.

Example:-

```
Var House = []; //method 1
Var House = new array(); //method 2
```

But generally method 1 is preferred over the method 2. Let us understand the reason for this.

Initialization of an Array:

Example (for Method 1)

```
//Initializing while declaring
var house = ["CSE", "ECE", "CIVIL", "EEE"];
//Initializing after declaring
house[0] = "CSE"
house[0] = "ECE"
house[0] = "CIVIL"
house[0] = "EEE"
```

Example (for Method 2)

```
// Initializing while declaring
// creates an array while having elements 10,20,30,40,50
var house = new Array (10, 20, 30, 40, 50);
// creates an array of 5 undefined elements
Var house1 = new Array(5);
// creates an array with element WT
Var home = new Array ("!WT");
```

As shown in above example the house contain 5 elements i.e., (10,20,30,40,50) While house1 contains 5 undefined elements instead of having a single element 5. Hence, while working with numbers this method is generally not preferred but it works fine with strings and Boolean as shown in the example above home contains a ~~String~~ single element WT

An array in JavaScript can hold different elements.
We can store Numbers, Strings and Boolean in a single array.

Example.

```
// storing number, boolean, strings in an Array.
var house = ['A', 25000, 'B', 50000, 'C', true];
```

Accessing Array Elements.

Array in JavaScript are indexed from 0 so we can access array elements as follows:

```
var house = ['A' , 25000, 'B', 50000, 'c', true];
alert (house [0] + "cost = " + house[1]);
var cost_A = house [1];
Var is_for_rent = house [5];
alert ("cost Of A = "+ cost_A);
alert ("Is house for rent =") + is_for_rent);
```

Length property of an array:

Length property of an array returns the length of an Array. Length of an array is always one more than the highest index of an Array.

Example:

```
var house = ["AB", 25000, "BC", 50000, "CD", true];
// len contains the length of an array
var len = house.length;
for (var i=0; i<len; i++)
   alert (house [i]);
```

DHTML

DHTML Stands for Dynamic HTML, it is totally different from HTML. The browsers which support the dynamic HTML are some of the versions of Netscape Navigator and Interenel Explorer of version higher than 4.0. The DHTML is based on the properties of the HTML, javascript, CSS and DOM (Document object model which is used to access individual elements of a document) which helps in making dynamic content. It is the combination of HTML, CSS, JS, and DOM. The DHTML make use of Dynamic object model to make changes in settings and also in properties and methods. It also makes uses of Scripting and it is also part of earlier computing trends.

DHTML allows different scripting languages in a web page to change their variables, which enhance the effects, looks and many others functions after the whole page have been fully loaded or under a view process, or otherwise static HTML pages on the same. But in true ways, there is nothing that as dynamic in DHML, there is only the enclosing of different technologies like CSS, HTML, JS, DOM, and different sets of static languages which make it as dynamic.

DHTML is used to create interactive and animated webpages that are generated in real-time, also known as dynamic webpages so that when such a page is accessed, the code within the page is analyzed on the web server and the resulting HTML is sent to the client's web browser.

DHTML is Not a language or a web standard

DHTML stands for Dynamic HTML.

DHTML is a TERM used to describe the technologies used to make web pages dynamic and interactive.

To most people DHTML means the combination of HTML, Javascript, DOM, and CSS.

2020/8/13 14:49

According to the World Wide Web Consortium (W3C):
" Dynamic HTML is a term used by some vendors to describe the combinati of HTML style sheets and scripts that allows documents to be animated."

HTML :-

HTML stands for Hypertext Markup language and it is a client-side markup language. It is used. to build the block of web pages.

Javascript :-

It is a client-side Scripting language. Javascript is supported by most the browser, also have cookies collection to determine the user needs.

CSS:-

The abbreviation of CSS is Cascading Style Sheet. It helps in the styling of the web pages and helps in designing of the pages. The CSS rules for DHTML will be modified at different levels using JS with event handlers which adds a significant amount of dynamism with very little code.

DOM:

It is known as a Document Object Model which act as the weakest links in it. The only defect in it is that most of the browser does not support DOM. It is a way to manipulate the static contents.

Note:-

Many times DHTML is confused with being a language like HTML but it is not. It must be kept in mind that it is an interface or browsers enhancement featu which makes it possible to access the object model through Javascript language and hence make the webpage more interactive .

Key features:-

Following are the some major key features of DHTML

* Tags and their properties can be changed using DHTML.
* It is used for real-time positioning
* Dynamic fonts can be generated using DHTML
* It is also used for data binding

2020/8/13 14:49

* It makes a webpage dynamic and be used to create animations, games, applications, along with providing new ways of navigating through websites
* The functionality of a webpage is enhanced due to the usage of low-bandwi -dth effect by DHTML.
* DHTML also facilitates the use of methods, events, properties, and codes.

Why use DHTML?

DHTML makes a webpage dynamic but Javascript also does, the question arises that what different does DHTML do? so the answer is that DHTML has the ability to change a webpages look, content and style ~~____~~ once the document has loaded on our demand without changing or deleting everything already existing on the browser's webpage. DHTML can change the content of a webpage on demand without the browser having to erase everything else, i.e being able to alter changes on a webpage even after the document has completely loaded.

Advantages:-

* Size of the files are compact in compared to other interactional media like Flash or shockwave, and it downloads faster.
* It is supported by big browser manufactures like Microsoft and Netscape
* Highly flexible and easy to make changes.
* Viewer requires no extra plug-ins for browsing through the webpage that uses DHTML, they do not need any extra requirements or special software to view it.
* User time is saved by sending less number of requests to the server. As it is possible to modify and replace elements even after a page is loaded, it is not required to create separate pages for changing styles which in turn saves time in building pages and also reduces the number of requests that are sent to the server.
* It has more advanced functionality than a static HTML. it is capable

2020/8/13 14:50

holding more content on the webpage at the sametime.

Disadvantages

* It is not supported by all the browsers. It is supported only by recent browsers such as Netscape 6, IE 5.5, and Opera 5 like browsers.
* Learning of DHTML requires a lot of pre-requistes languages such as HTML, CSS, JS etc should be known to the designer before starting with DHTML which is a long and time-consuming in itself.
* Implementation of different browsers are different. So if it worked in one browser, it might not necessarily work the same way in another browser.
* Even after being great with functionality, DHTML requires a few tools and utilities that are some expensive. For example, the DHTML text editor, Dreamweaver. Along with it the improvement cost of transferring from HTML to DHTML makes cost rise much higher.

Difference between HTML and DHTML:-

* HTML is a markup language while DHTML is a collection of technologies
* HTML is used to create static webpages while DHTML is capable of creating dynamic webpages
* DHTML is used to create animations and dynamic menus but HTM not used.
* DHTML sites are slow upon client-side technologies whereas DHTML site are comparatively faster.
* web pages created using HTML are rather simple and have no styling as it uses only one language whereas DHTML uses HTML, CSS and Javascript which results in a much better and way more presentabl webpage.
* HTML cannot be used as server side code but DHTML used as server side code.
* DHTML needs database connectivity but not in case of HTML.
* Files in HTML are stored using .htm or .html extension while DHTML uses .dhtm extension.
* HTMl requires no processing from the browser but DHTML does

DHTML : positioning elements

when it comes to positioning content on a page there is a handful properties to use that can help you manipulate the location of an element. This examples containing different positioning element types using the CSS position property. To use positioning on an element, you must first declare its position property, which specifies the type of positioning method used for an element. Using the position property values, the elements are positioned using the top, bottom, left and right properties. They also work differently depending on their position value.

There are five types of positioning values;

- Static
- relative
- fixed
- absolute
- Sticky

Static:-

HTML elements are positioned static by default and the element is positioned according to the normal flow of the document; static positioned elements are not affected by the top, bottom, left and right properties. An element with position: static; is not positioned in any special way. The CSS used for setting the position to static is:

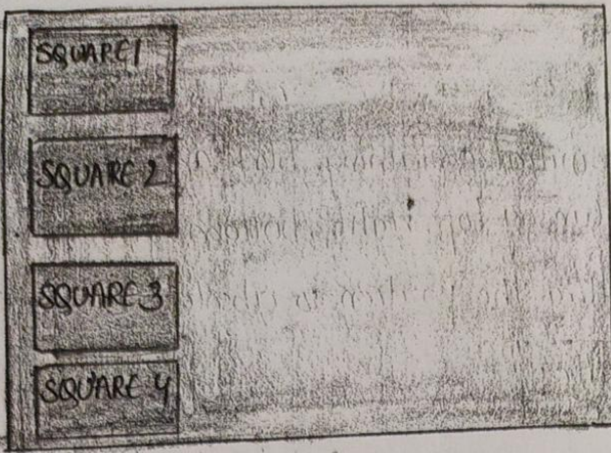
```
position: static;
```

2020/8/13 14:50

Next is an example of using the static position value:

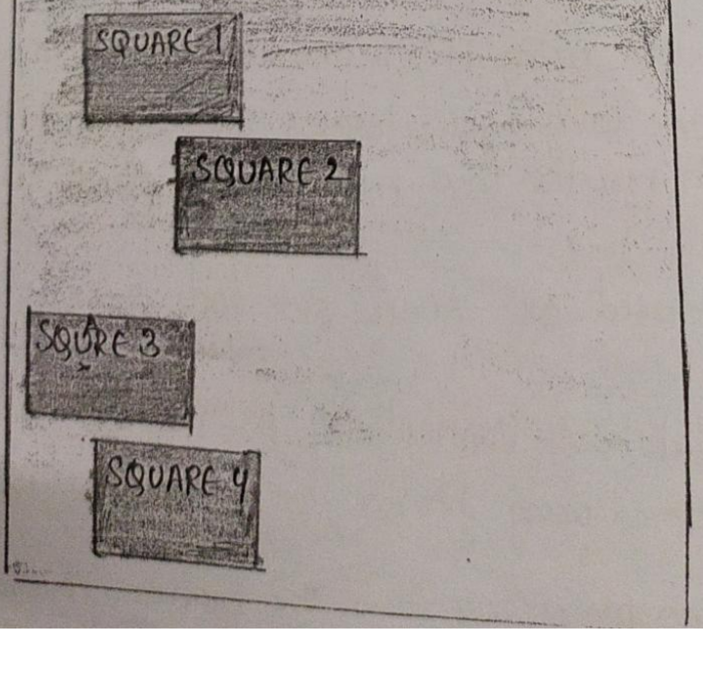
```
<!DOCTYPE html>
 <html>
 <head>
  <style>
   body{
     color: WHITE;
     font: Helvetica;
     width: 420px;
   }
  .Square-set,
  .Square{
    border-radius:15px;
  }
  .Squar-set{
    background:darkgrey;
 }
 .Square {
   position :static;
   background: Green;
   height: 70px;
   line-height: 40px;
   text-align: center;
   width: 90 px;
}
</style>
</head>
<body>
<div class = "square-set">
  <figure class =" square Square-1" > SQUARE 1 </figure>
  <figure class =" Square Square-2' >SQUARE 2 </figure>
  <figure class =" square square-3"> SQUARE 3 </figure>
  <figure class = "square Square-4" > SQU
```

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



Relative:-
The element is positioned according to the normal flow of the document positioned relative to its normal position, and then offset relative to itself based on the values of top, right, bottom, and left. The offset does not affect the position of any other elements; thus, the space given for the element in the page layout is the same as if position were static. Setting the top, right, bottom, and left properties of a relatively-positioned element will cause it to be adjusted away from its normal position. Ot content will not be adjusted to fit into any gap left by the element.

The CSS used for setting the position to relative is:



2020/8/13 1

Absolute:-

The element is removed from the normal document flow, and, in the page layout, no space is created for the element. The element is positioned relative to the closest positioned ancestor, if there is any; otherwise, it is placed relative to the initial containing block and its final position is determined by the values of top, right, bottom, and left.

The CSS used for setting the position to absolute is:

position: absolute;

An element with position: absolute; is positioned relative to the closest positioned ancestor. If an absolute positioned element has no positioned ancestors, it uses the document body, and moves together with page scrolling. A "positioned" element is one whose position is anything but static.

Next example emphasizes the absolute position of elements:

```
<!DOCTYPE html>
<html>
<head>
<style>
.square-set{
 color: WHITE;
 background: darkgrey;
 height: 200px;
 position: relative;
 border-radius: 15px;
 font: Helvetica;
 width: 420px;
}
.Square{
 background: green;
```

2020/8/13 14:50

```
height: 80px;
position: absolute;
width: 80px;
border-radius: 15px;
line-height: 60px;
}
.square-1{
   top: 10%;
   left: 6%;
}
.square-2{
position: relative;
```

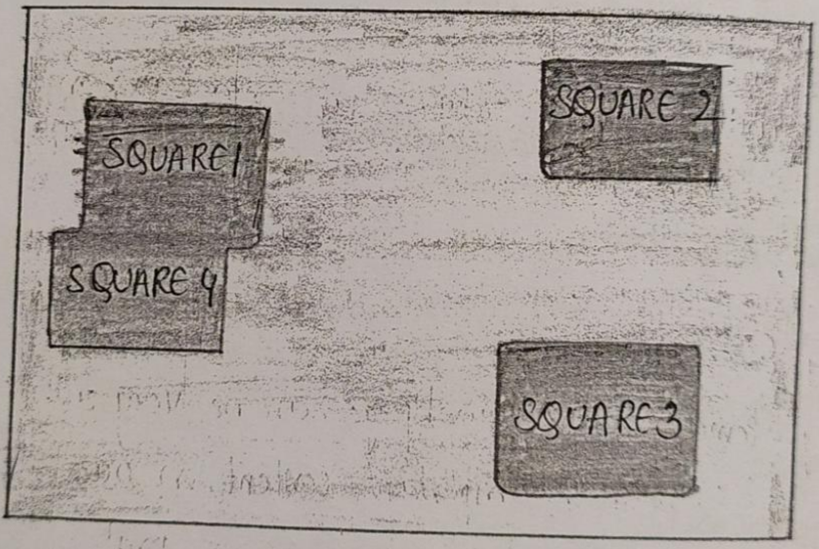
The below example uses the relative position value:

```
<!DOCTYPE html>
<html>
<head>
<style>
body{
color: white;
font: Helvetica;
width: 420.px;
}
.square-set,
.square{
  border-radius: 15px;
}
.square-set{
  background: darkgrey;
}.
.square{
  background: green;
  height: 70px;
```

2020/8/13 14:5

```
      line-height:40px;
      position:relative;
      text-align:center;
      width:80px;
  }
  .Square-1{
      top:15px;
  }
  .Square-2{
      left:50px;
  }
  .Square-3{
      bottom:-23px;
      right:30px;
  }
</style>
</head>
<body>
<div class="square-set">
    <figure class=" Square square-1"> SQUARE1 </figure>
    <figure class =" square square-2"> SQUARE2 </figure>
    <figure class = " square Square-3"> SQUARE 3 </figure>
    <figure class = " Square Squaurre-4"> SQUARE 4</firgure>
</div>
</body>
</html>

   top:5;
   right:-20px;
}
.Square-3{
   bottom:-15 px;
   right: 40px;
}
```

```
.square-4{
    bottom: 0;
}
</style>
</head>
<body>
 <div class="square-set">
   <figure class="square square-1"> SQUARE 1</figure>
   <figure class="square square-2"> SQUARE 2 </figure>
   <figure class="square square-3"> SQUARE 3</figure>
   <figure class="square square-4"> SQUARE 4 </figure>
 </div>
</body>
</html>
```

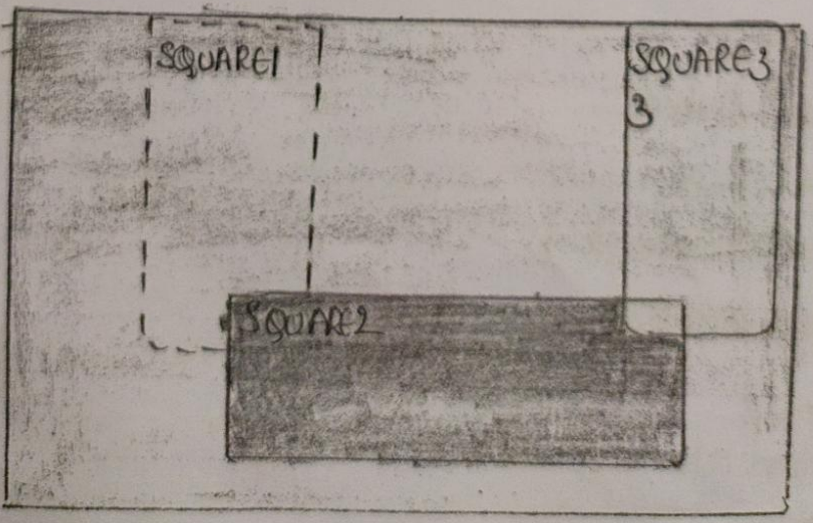


fixed :-

The element it is removed from the normal document flow, and, in the page layout, there is no space created for the element. The element is positioned relative to its initial containing block established by the viewport and its final position is determined by the values top, right, bottom and left. This value always creates a new stacking context.
The CSS used for setting the position to fixed looks like this:
    position: fixed;

Next example shows how the Z-index property works on different squares:

```
<!DOCTYPE html>
 <html>
  <head>
   <style>
    .square-set{
       color:white;
       background:purple;
       height:170px;
       position:relative;
       border-radius:15px;
      font: Helvetica;
      width:400px;
  }
  .square{
     background:orange;
     border:4px solid goldenrod;
     position:absolute;
     border-radius:15px;
     height:80px;
    width:80px;
 }
 .square-1{
   position: relative;
   Z-index:1;
   border:dashed;
   height:8em;
   margin-bottom:1em;
   margin-top:2em;
}
.square-2{
  position:absolute;
  Z-index:2;
  background: black;
  width: 65%;
  left:60px;
  top: 3cm;
}
.square -3{
   position: absolute;
   z-index: 3;
   background: lightgreen;
   width: 20%;
   left: 65%;
   top:-25px;
   height: 7em;
   opacity: 0.9;
}
</style>
</head>
< body>
   <div class=" square-set">
      <figure class=' square square-1">SQUARE 1</figure>
      <figure class = " Square Square-2'>SQUARE 2</figure>
      <figure class =" square square-3"> SQUARE 3</figure>
   </div>
</body>
</html>
```

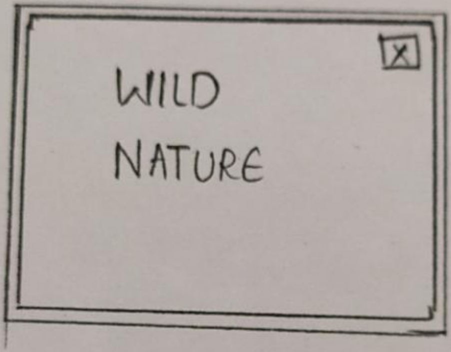


Positioning Text Over an Image :-

The below example overlays some text over an image using the Css positioning values described above:

```
<!DOCTYPE html>
<html>
<head>
<style>
.module{
  background:
   linear-gradient{
     rgba(0,4,5,0.6),
     rgba(2,0,3,0.6)
 ),
url(http://www.holtz.org/Library/Images/Slideshows/Gallery/Landscapes/43098-
DSC-64XX_landscape-keltern-1_wall.jpg);
   background-size:cover;
   width:600px;
   height:400px;
   margin:10px 00 10 px;
   position:relative;
   float:left;
}
.mid h3{
  font-family:Helvetica;
  font-weight:900;
  color:white;
  text-transform:uppercase;
  margin:0;
  position:absolute;
  top:30%;
  left:50%;
```

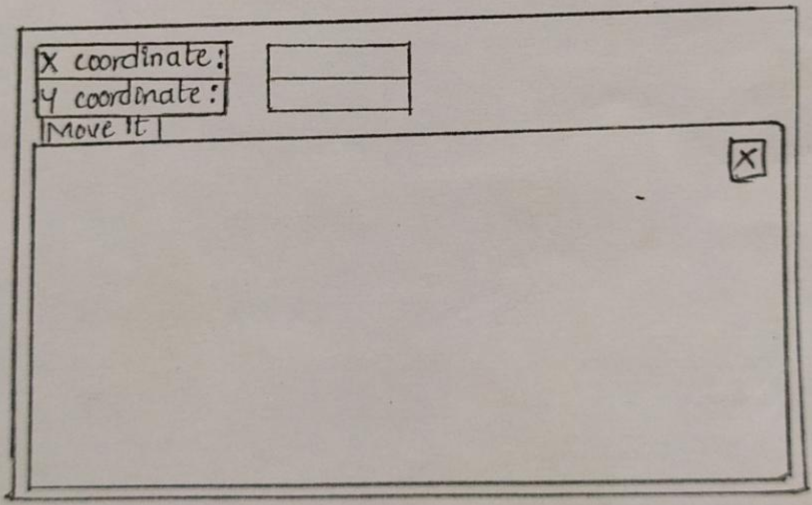
```
    font-size: 3rem;
    transform: translate(-50%, -50%);
}
</style>
</head>
<body>
 <div class=" module mid">
   <h3> wild nature</h3>
 </div>
</body>
</html>
```

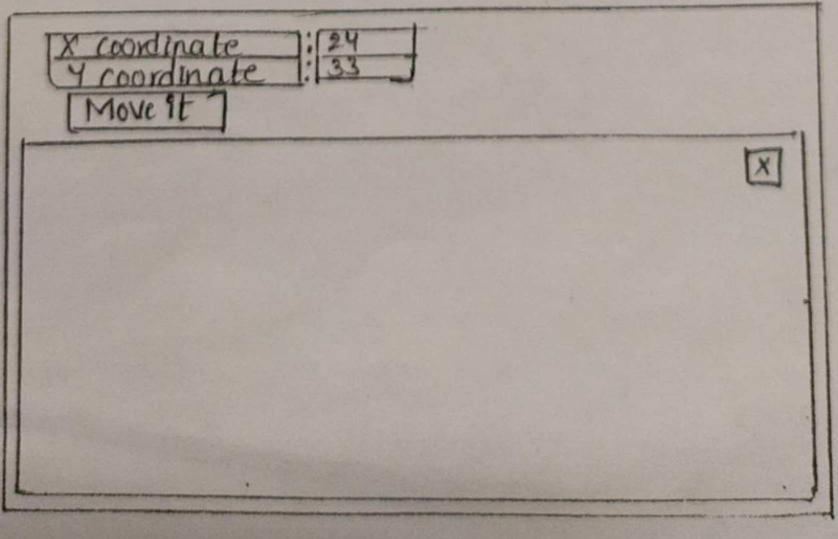


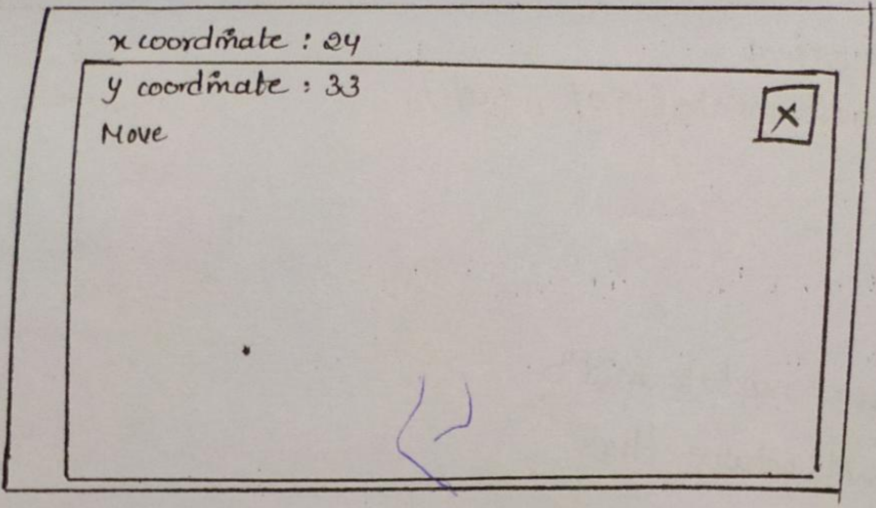
DHTML - MOVING ELEMENTS

```
<?xml version ="1.0" encoding ="utf-8"?>
<!DOCTYPE html PUBLIC "-//w3c//DTD XHTML 1.1//EN"
"http://www.w3.org/TR/xhtml11/DTD/xhtml11.dtd">
<html xmlns=" http://www.w3.org/1999/xhtml">
<head>
  <title>Moving elements </title>
  <Script type="text/javascript" src=" /move.js"> </script>
</head>
<body>
<form action =" ">
<p>
x coordinate : <input type =" text" id=" leftCoord" size ="3" /><br/>
```

```
y coordinate: <input type ="text" id=" topCoord" size="3"/ ><br/>
 <input type =" button" value ="Move it"
 onclick="moveIt('nebula',
  document.getElementById('top Coord').value,
 document.getElementById('left Coord').value)"/>
</p>
</form>
<div id=" nebula" style=" position :absolute; top : 115px;left:0;">
 <img src=" ./NGC1300.jpeg" alt=" (Picture of a galaxy)"/>
</div>
</body>
</html>
```







x coordinate : 24

y coordinate : 33

Move

AngularJS is a very powerful JavaScript Framework. It is used in Single Page Application (SPA) projects. It extends HTML DOM with additional attributes and makes it more responsive to user actions. AngularJS is open source, completely free, and used by thousands of developers around the world. It is licensed under the Apache license version 2.0.

Why to Learn AngularJS?

AngularJS is an open-source web application framework. It was originally developed in 2009 by Misko Hevery and Adam Abrons. It is now maintained by Google. Its latest version is 1.2.21.

- AngularJS is a efficient framework that can create Rich Internet Applications (RIA).
- AngularJS provides developers an options to write client side applications using JavaScript in a clean Model View Controller (MVC) way.
- Applications written in AngularJS are cross-browser compliant. AngularJS automatically handles JavaScript code suitable for each browser.
- AngularJS is open source, completely free, and used by thousands of developers around the world. It is licensed under the Apache license version 2.0.

Overall, AngularJS is a framework to build large scale, high-performance, and easyto-maintain web applications.

AngularJS is an open-source web application framework. It was originally developed in 2009 by Misko Hevery and Adam Abrons. It is now maintained by Google. Its latest version is 1.2.21.

General Features

The general features of AngularJS are as follows −

- AngularJS is a efficient framework that can create Rich Internet Applications (RIA).
- AngularJS provides developers an options to write client side applications using JavaScript in a clean Model View Controller (MVC) way.

- Applications written in AngularJS are cross-browser compliant. AngularJS automatically handles JavaScript code suitable for each browser.
- AngularJS is open source, completely free, and used by thousands of developers around the world. It is licensed under the Apache license version 2.0.

Overall, AngularJS is a framework to build large scale, high-performance, and easyto-maintain web applications.

Core Features

The core features of AngularJS are as follows −

- **Data-binding** − It is the automatic synchronization of data between model and view components.
- **Scope** − These are objects that refer to the model. They act as a glue between controller and view.
- **Controller** − These are JavaScript functions bound to a particular scope.
- **Services** − AngularJS comes with several built-in services such as $http to make a XMLHttpRequests. These are singleton objects which are instantiated only once in app.
- **Filters** − These select a subset of items from an array and returns a new array.
- **Directives** − Directives are markers on DOM elements such as elements, attributes, css, and more. These can be used to create custom HTML tags that serve as new, custom widgets. AngularJS has built-in directives such as ngBind, ngModel, etc.
- **Templates** − These are the rendered view with information from the controller and model. These can be a single file (such as index.html) or multiple views in one page using *partials*.
- **Routing** − It is concept of switching views.
- **Model View Whatever** − MVW is a design pattern for dividing an application into different parts called Model, View, and Controller, each with distinct responsibilities. AngularJS does not implement MVC in the traditional sense, but rather something closer to MVVM (Model-View-ViewModel). The Angular JS team refers it humorously as Model View Whatever.

- **Deep Linking** − Deep linking allows to encode the state of application in the URL so that it can be bookmarked. The application can then be restored from the URL to the same state.
- **Dependency Injection** − AngularJS has a built-in dependency injection subsystem that helps the developer to create, understand, and test the applications easily.

Concepts

The following diagram depicts some important parts of AngularJS which we will discuss in detail in the subsequent chapters.

Advantages of AngularJS

The advantages of AngularJS are −

- It provides the capability to create Single Page Application in a very clean and maintainable way.
- It provides data binding capability to HTML. Thus, it gives user a rich and responsive experience.
- AngularJS code is unit testable.
- AngularJS uses dependency injection and make use of separation of concerns.
- AngularJS provides reusable components.
- With AngularJS, the developers can achieve more functionality with short code.
- In AngularJS, views are pure html pages, and controllers written in JavaScript do the business processing.

On the top of everything, AngularJS applications can run on all major browsers and smart phones, including Android and iOS based phones/tablets.

Disadvantages of AngularJS

Though AngularJS comes with a lot of merits, here are some points of concern −

- **Not Secure** − Being JavaScript only framework, application written in AngularJS are not safe. Server side authentication and authorization is must to keep an application secure.
- **Not degradable** − If the user of your application disables JavaScript, then nothing would be visible, except the basic page.

AngularJS Directives

The AngularJS framework can be divided into three major parts −

- **ng-app** − This directive defines and links an AngularJS application to HTML.
- **ng-model** − This directive binds the values of AngularJS application data to HTML input controls.
- **ng-bind** − This directive binds the AngularJS application data to HTML tags.

AngularJS – Expressions

Expressions are used to bind application data to HTML. Expressions are written inside double curly braces such as in {{ expression}}. Expressions behave similar to ngbind directives. AngularJS expressions are pure JavaScript expressions and output the data where they are used.

Using numbers

```
<p>Expense on Books : {{cost * quantity}} Rs</p>
```

Using Strings

```
<p>Hello {{student.firstname + " " + student.lastname}}!</p>
```

Using Object

```
<p>Roll No: {{student.rollno}}</p>
```

Using Array

```
<p>Marks(Math): {{marks[3]}}</p>
```

Example

The following example shows the use of all the above-mentioned expressions −

testAngularJS.htm

```
<html>
   <head>
      <title>AngularJS Expressions</title>
   </head>

   <body>
      <h1>Sample Application</h1>

      <div ng-app = "" ng-init = "quantity = 1;cost = 30;
         student = {firstname:'Mahesh',lastname:'Parashar',rollno:101};
         marks = [80,90,75,73,60]">
         <p>Hello {{student.firstname + " " + student.lastname}}!</p>
         <p>Expense on Books : {{cost * quantity}} Rs</p>
         <p>Roll No: {{student.rollno}}</p>
         <p>Marks(Math): {{marks[3]}}</p>
      </div>

      <script src =
"https://ajax.googleapis.com/ajax/libs/angularjs/1.3.14/angular.min.js">
      </script>

   </body>
</html>
```

Output

Open the file testAngularJS.htm in a web browser and see the result.

AngularJS Forms

Forms in AngularJS provides data-binding and validation of input controls.

Input Controls

Input controls are the HTML input elements:

- input elements
- select elements
- button elements
- textarea elements

Data-Binding

Input controls provides data-binding by using the ng-model directive.

```
<input type="text" ng-model="firstname">
```

The application does now have a property named firstname.

The ng-model directive binds the input controller to the rest of your application.

The property firstname, can be referred to in a controller:

Example

```
<script>
var app = angular.module('myApp', []);
app.controller('formCtrl', function($scope) {
  $scope.firstname = "John";
});
</script>
```

Example

```
<form>
  First Name: <input type="text" ng-model="firstname">
</form>
```

```
<h1>You entered: {{firstname}}</h1>
```

Checkbox

A checkbox has the value true or false. Apply the ng-model directive to a checkbox, and use its value in your application.

Example

Show the header if the checkbox is checked:

```
 <form>
  Check to show a header:
  <input type="checkbox" ng-model="myVar">
</form>

<h1 ng-show="myVar">My Header</h1
```

Radiobuttons

Bind radio buttons to your application with the ng-model directive.

Radio buttons with the same ng-model can have different values, but only the selected one will be used.

Example

Display some text, based on the value of the selected radio button:

```
<form>
  Pick a topic:
  <input type="radio" ng-model="myVar" value="dogs">Dogs
  <input type="radio" ng-model="myVar" value="tuts">Tutorials
  <input type="radio" ng-model="myVar" value="cars">Cars
</form>
```

Selectbox

Bind select boxes to your application with the ng-model directive.

The property defined in the ng-model attribute will have the value of the selected option in the selectbox.

Example

Display some text, based on the value of the selected option:

```
<form>
  Select a topic:
  <select ng-model="myVar">
    <option value="">
    <option value="dogs">Dogs
    <option value="tuts">Tutorials
    <option value="cars">Cars
  </select>
</form>
```

An AngularJS Form Example

First Name:

Last Name:

RESET

form = {"firstName":"John","lastName":"Doe"}

master = {"firstName":"John","lastName":"Doe"}

Application Code

```
<div ng-app="myApp" ng-controller="formCtrl">
  <form novalidate>
    First Name:<br>
    <input type="text" ng-model="user.firstName"><br>
    Last Name:<br>
    <input type="text" ng-model="user.lastName">
    <br><br>
    <button ng-click="reset()">RESET</button>
  </form>
  <p>form = {{user}}</p>
  <p>master = {{master}}</p>
</div>

<script>
var app = angular.module('myApp', []);
app.controller('formCtrl', function($scope) {
  $scope.master = {firstName: "John", lastName: "Doe"};
  $scope.reset = function() {
    $scope.user = angular.copy($scope.master);
  };
  $scope.reset();
});
</script>
```

Example Explained

The **ng-app** directive defines the AngularJS application.

The **ng-controller** directive defines the application controller.

The **ng-model** directive binds two input elements to the **user** object in the model.

The **formCtrl** controller sets initial values to the **master** object, and defines the **reset()** method.

The **reset()** method sets the **user** object equal to the **master** object.

The **ng-click** directive invokes the **reset()** method, only if the button is clicked.

The novalidate attribute is not needed for this application, but normally you will use it in AngularJS forms, to override standard HTML5 validation

AngularJS Form Validation

AngularJS can validate input data.

Form Validation

AngularJS offers client-side form validation.

AngularJS monitors the state of the form and input fields (input, textarea, select), and lets you notify the user about the current state.

AngularJS also holds information about whether they have been touched, or modified, or not.

You can use standard HTML5 attributes to validate input, or you can make your own validation functions.

Client-side validation cannot alone secure user input. Server side validation is also necessary.

Required

Use the HTML5 attribute required to specify that the input field must be filled out:

Example

The input field is required:

```
<form name="myForm">
  <input name="myInput" ng-model="myInput" required>
</form>

<p>The input's valid state is:</p>
<h1>{{myForm.myInput.$valid}}</h1>
```

E-mail

Use the HTML5 type email to specify that the value must be an e-mail:

Example

The input field has to be an e-mail:

```
<form name="myForm">
  <input name="myInput" ng-model="myInput" type="email">
</form>

<p>The input's valid state is:</p>
<h1>{{myForm.myInput.$valid}}</h1>
```

Form State and Input State

AngularJS is constantly updating the state of both the form and the input fields.

Input fields have the following states:

- $untouched The field has not been touched yet
- $touched The field has been touched

- $pristine The field has not been modified yet
- $dirty The field has been modified
- $invalid The field content is not valid
- $valid The field content is valid

They are all properties of the input field, and are either true or false.

Forms have the following states:

- $pristine No fields have been modified yet
- $dirty One or more have been modified
- $invalid The form content is not valid
- $valid The form content is valid
- $submitted The form is submitted

They are all properties of the form, and are either true or false.

You can use these states to show meaningful messages to the user. Example, if a field is required, and the user leaves it blank, you should give the user a warning:

Example

Show an error message if the field has been touched AND is empty:

```
<input name="myName" ng-model="myName" required>
<span ng-show="myForm.myName.$touched &&
myForm.myName.$invalid">The name is required.</span>
```

CSS Classes

AngularJS adds CSS classes to forms and input fields depending on their states.

The following classes are added to, or removed from, input fields:

- ng-untouched The field has not been touched yet
- ng-touched The field has been touched
- ng-pristine The field has not been modified yet

- ng-dirty The field has been modified
- ng-valid The field content is valid
- ng-invalid The field content is not valid
- ng-valid-*key* One *key* for each validation. Example: ng-valid-required, useful when there are more than one thing that must be validated
- ng-invalid-*key* Example: ng-invalid-required

The following classes are added to, or removed from, forms:

- ng-pristine No fields has not been modified yet
- ng-dirty One or more fields has been modified
- ng-valid The form content is valid
- ng-invalid The form content is not valid
- ng-valid-*key* One *key* for each validation. Example: ng-valid-required, useful when there are more than one thing that must be validated
- ng-invalid-*key* Example: ng-invalid-required

The classes are removed if the value they represent is false.

Add styles for these classes to give your application a better and more intuitive user interface.

Example

Apply styles, using standard CSS:

```
<style>

input.ng-invalid {
  background-color: pink;
}
input.ng-valid {
  background-color: lightgreen;
}


</style>
```

Forms can also be styled:

Example

Apply styles for unmodified (pristine) forms, and for modified forms:

```
<style>

form.ng-pristine {
  background-color: lightblue;
}
form.ng-dirty {
  background-color: pink;
}


</style>
```

Custom Validation

To create your own validation function is a bit more tricky; You have to add a new directive to your application, and deal with the validation inside a function with certain specified arguments.

Example

Create your own directive, containing a custom validation function, and refer to it by using my-directive.

The field will only be valid if the value contains the character "e":

```
<form name="myForm">
<input name="myInput" ng-model="myInput" required my-directive>
</form>
```

```
<script>

var app = angular.module('myApp', []);
app.directive('myDirective', function() {
  return {
    require: 'ngModel',
    link: function(scope, element, attr, mCtrl) {
      function myValidation(value) {
        if (value.indexOf("e") > -1) {
          mCtrl.$setValidity('charE', true);
        } else {
          mCtrl.$setValidity('charE', false);
        }
        return value;
      }
      mCtrl.$parsers.push(myValidation);
    }
  };
});


</script>
```

Example Explained:

In HTML, the new directive will be referred to by using the attribute my-directive.

In the JavaScript we start by adding a new directive named myDirective.

Remember, when naming a directive, you must use a camel case name, myDirective, but when invoking it, you must use - separated name, my-directive.

Then, return an object where you specify that we require ngModel, which is the ngModelController.

Make a linking function which takes some arguments, where the fourth argument, mCtrl, is the ngModelController,

Then specify a function, in this case named myValidation, which takes one argument, this argument is the value of the input element.

Test if the value contains the letter "e", and set the validity of the model controller to either true or false.

At last, mCtrl.$parsers.push(myValidation); will add the myValidation function to an array of other functions, which will be executed every time the input value change

---

Validation Example

```
<!DOCTYPE html>
<html>
<script src="https://ajax.googleapis.com/ajax/libs/angularjs/1.6.9/angular.min.js">
</script>
<body>

<h2>Validation Example</h2>

<form  ng-app="myApp"  ng-controller="validateCtrl"
name="myForm" novalidate>

<p>Username:<br>
  <input type="text" name="user" ng-model="user" required>
  <span style="color:red" ng-show="myForm.user.$dirty &&
myForm.user.$invalid">
  <span ng-show="myForm.user.$error.required">Username is required.</span>
  </span>
</p>

<p>Email:<br>
  <input type="email" name="email" ng-model="email" required>
  <span style="color:red" ng-show="myForm.email.$dirty &&
myForm.email.$invalid">
  <span ng-show="myForm.email.$error.required">Email is required.</span>
  <span ng-show="myForm.email.$error.email">Invalid email address.</span>
  </span>
</p>

<p>
  <input type="submit"
  ng-disabled="myForm.user.$dirty && myForm.user.$invalid ||
  myForm.email.$dirty && myForm.email.$invalid">
</p>

</form>

<script>
var app = angular.module('myApp', []);
app.controller('validateCtrl', function($scope) {
  $scope.user = 'John Doe';
  $scope.email = 'john.doe@gmail.com';
});
</script>

</body>
</html>
```

Example Explained

The AngularJS directive **ng-model** binds the input elements to the model.

The model object has two properties: **user** and **email**.

Because of **ng-show**, the spans with color:red are displayed only when user or email is **$dirty** and **$invalid**.

What is Angular JS Expressions?

Expressions are variables which were defined in the double braces {{ }}. They are very commonly used within Angular JS, and you would see them in our previous tutorials.

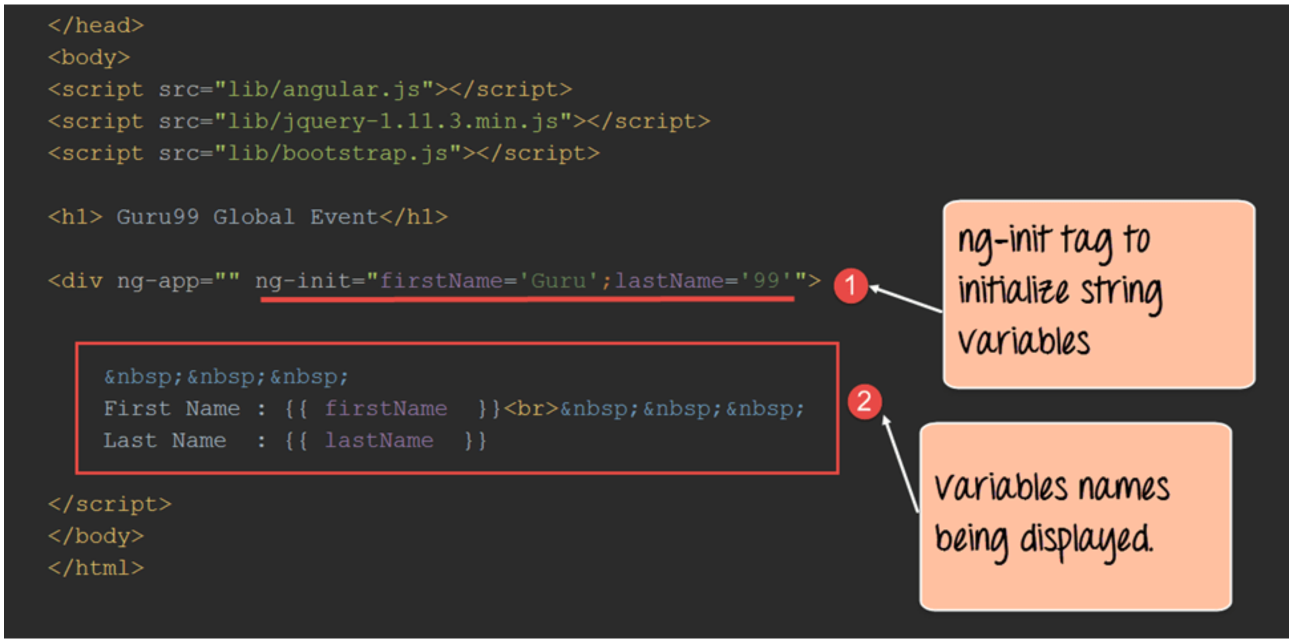
In this tutorial, you will learn-

- Explain Angular.js Expressions with example
- AngularJS Numbers
- AngularJS Strings
- AngularJS Objects
- AngularJS Arrays

AngularJS Strings

Expressions can be used to work with strings as well. Let's look at an example of Angular JS expressions with strings.

In this example, we are going to define 2 strings of "firstName" and "lastName" and display them using expressions accordingly.



```
</head>
<body>
<script src="lib/angular.js"></script>
<script src="lib/jquery-1.11.3.min.js"></script>
<script src="lib/bootstrap.js"></script>

<h1> Guru99 Global Event</h1>

<div ng-app="" ng-init="firstName='Guru';lastName='99'">

    &nbsp;&nbsp;&nbsp;
    First Name : {{ firstName  }}<br>&nbsp;&nbsp;&nbsp;
    Last Name  : {{ lastName  }}

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

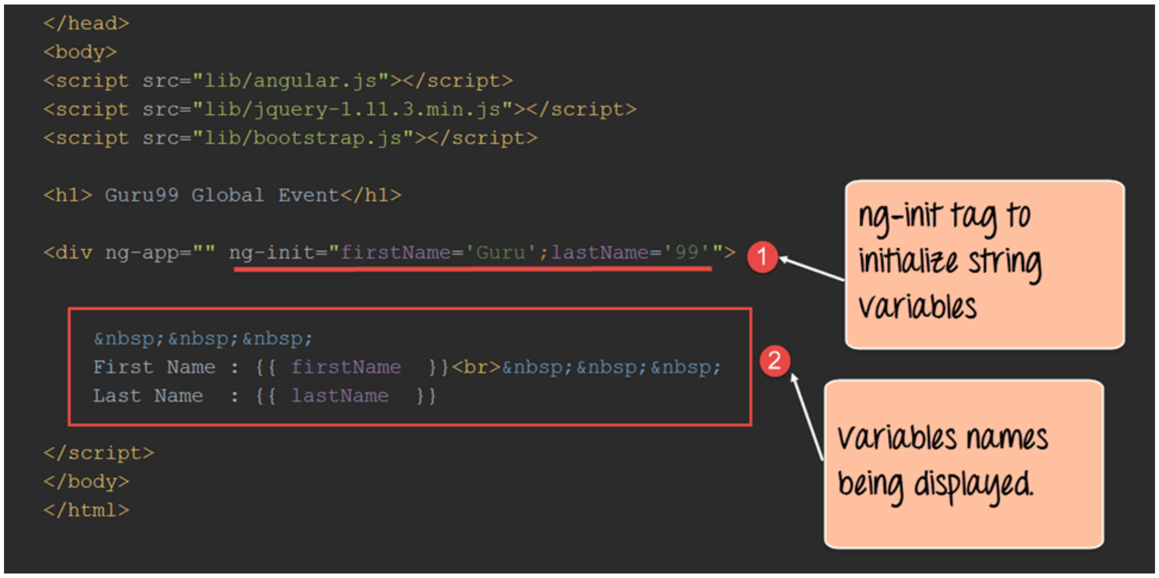
1
ng-init tag to initialize string variables
2
Variables names being displayed.

```
<!DOCTYPE html>
<html>
<head>
    <meta chrset="UTF 8">
    <title>Event Registration</title>

</head>
<body>

    <script src="https://code.angularjs.org/1.6.9/angular-route.js"></script>
    <script src="https://code.angularjs.org/1.6.9/angular.min.js"></script>

    <h1> Guru99 Global Event</h1>

    <div ng-app="" ng-init="firstName='Guru';lastName='99'">

        &nbsp;&nbsp;&nbsp;
        First Name : {{firstName}}<br>&nbsp;&nbsp;&nbsp;
        last Name : {{lastName}}

    </div>

</body>
</html>
```

AngularJS Strings

Expressions can be used to work with strings as well. Let’s look at an example of Angular JS expressions with strings.

In this example, we are going to define 2 strings of “firstName” and “lastName” and display them using expressions accordingly.



```
</head>
<body>
<script src="lib/angular.js"></script>
<script src="lib/jquery-1.11.3.min.js"></script>
<script src="lib/bootstrap.js"></script>

<h1> Guru99 Global Event</h1>

<div ng-app="" ng-init="firstName='Guru';lastName='99'">

    &nbsp;&nbsp;&nbsp;
    First Name : {{ firstName  }}<br>&nbsp;&nbsp;&nbsp;
    Last Name  : {{ lastName  }}

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

1
ng-init tag to initialize string variables
2
Variables names being displayed.

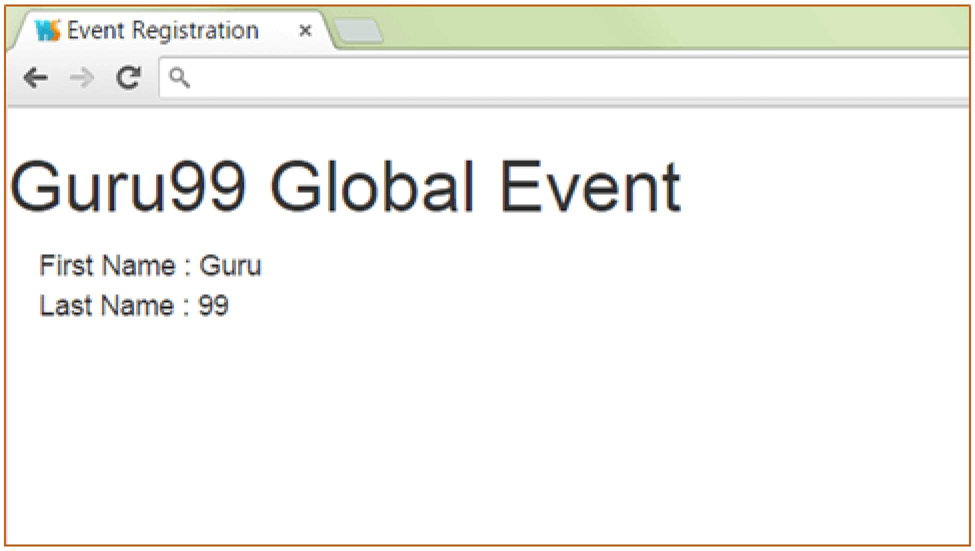
```
<!DOCTYPE html>
<html>
<head>
   <meta chrset="UTF 8">
   <title>Event Registration</title>

</head>
<body>

   <script src="https://code.angularjs.org/1.6.9/angular-route.js"></script>
   <script src="https://code.angularjs.org/1.6.9/angular.min.js"></script>

   <h1> Guru99 Global Event</h1>

   <div ng-app="" ng-init="firstName='Guru';lastName='99'">

      &nbsp;&nbsp;&nbsp;
      First Name : {{firstName}}<br>&nbsp;&nbsp;&nbsp;
      last Name : {{lastName}}

   </div>

</body>
</html>
```

Code Explanation:

1. The ng-init directive is used define the variables firstName with the value “Guru” and the variable lastName with the value of “99”.
2. We are then using expressions of {{firstName}} and {{lastName}} to access the value of these variables and display them in the view accordingly.

If the code is executed successfully, the following Output will be shown when you run your code in the browser.

Output:



Guru99 Global Event

First Name : Guru
Last Name : 99

From the output, it can be clearly seen that the values of firstName and lastName are displayed on the screen.

Angular.JS Objects

Expressions can be used to work with JavaScript objects as well.

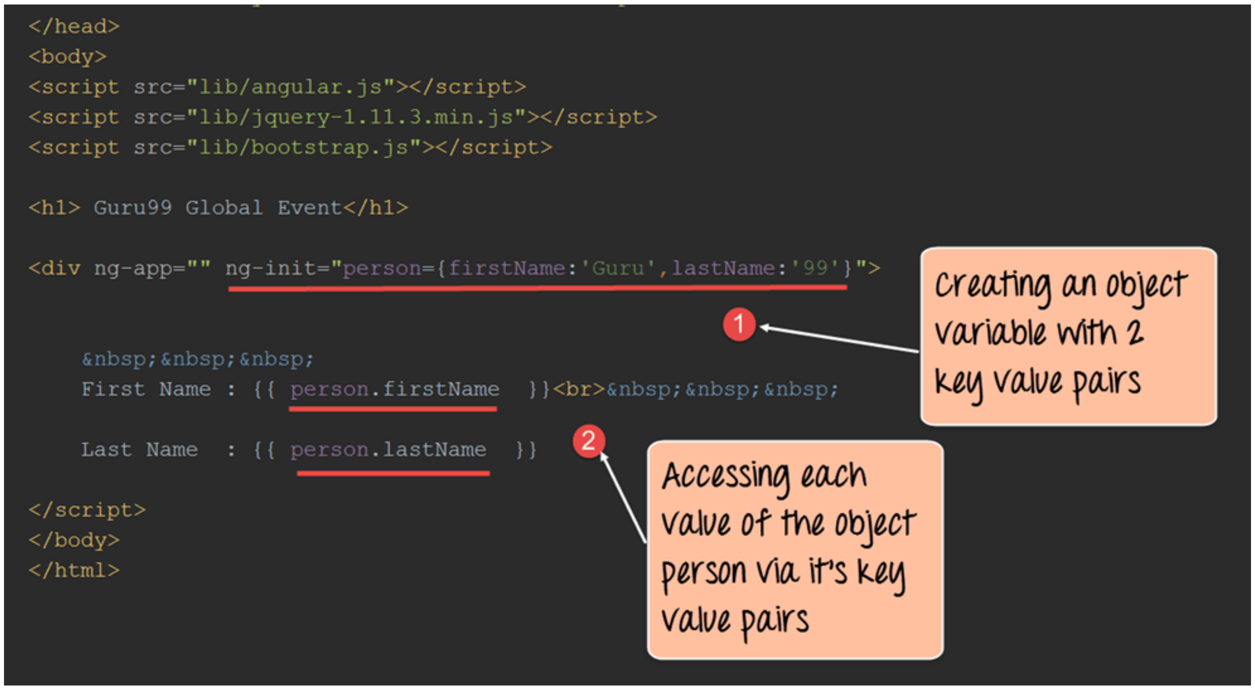
Let’s look at an example of Angular.JS expressions with javascript objects. A javascript object consists of a name-value pair.

Below is an example of the syntax of a javascript object.

Syntax:

```
var car = {type:"Ford", model:"Explorer", color:"White"};
```

In this example, we are going to define one object as a person object which will have 2 key value pairs of “firstName” and “lastName”.



```
</head>
<body>
<script src="lib/angular.js"></script>
<script src="lib/jquery-1.11.3.min.js"></script>
<script src="lib/bootstrap.js"></script>

<h1> Guru99 Global Event</h1>

<div ng-app="" ng-init="person={firstName:'Guru',lastName:'99'}">

    &nbsp;&nbsp;&nbsp;
    First Name : {{ person.firstName  }}<br>&nbsp;&nbsp;&nbsp;

    Last Name  : {{ person.lastName  }}

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

1
Creating an object variable with 2 key value pairs
2
Accessing each value of the object person via it's key value pairs

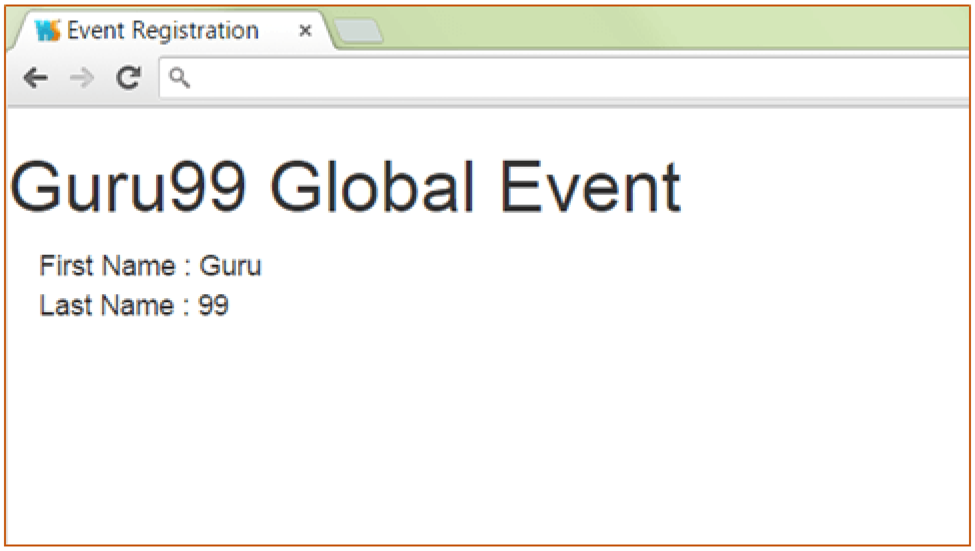
```
<!DOCTYPE html>
<html>
<head>
    <meta chrset="UTF 8">
    <title>Event Registration</title>

</head>
<body>

<script src="https://code.angularjs.org/1.6.9/angular-route.js"></script>
<script src="https://code.angularjs.org/1.6.9/angular.min.js"></script>

<h1> Guru99 Global Event</h1>

<div ng-app="" ng-init="person={firstName:'Guru',lastName:'99'}">

    &nbsp;&nbsp;&nbsp;
    First Name : {{person.firstName}}<br>&nbsp;&nbsp;&nbsp;
    Last Name : {{person.lastName}}
```

```
</div>

</body>
</html>
```
Code Explanation:

1. The ng-init directive is used to define the object person which in turn has key value pairs of firstName with the value “Guru” and the variable lastName with the value of “99”.
2. We are then using expressions of {{person.firstName}} and {{person.secondName}} to access the value of these variables and display them in the view accordingly. Since the actual member variables are part of the object person, they have to access it with the dot (.) notation to access their actual value.

If the code is executed successfully, the following Output will be shown when you run your code in the browser.

Output:



Guru99 Global Event

First Name : Guru
Last Name : 99

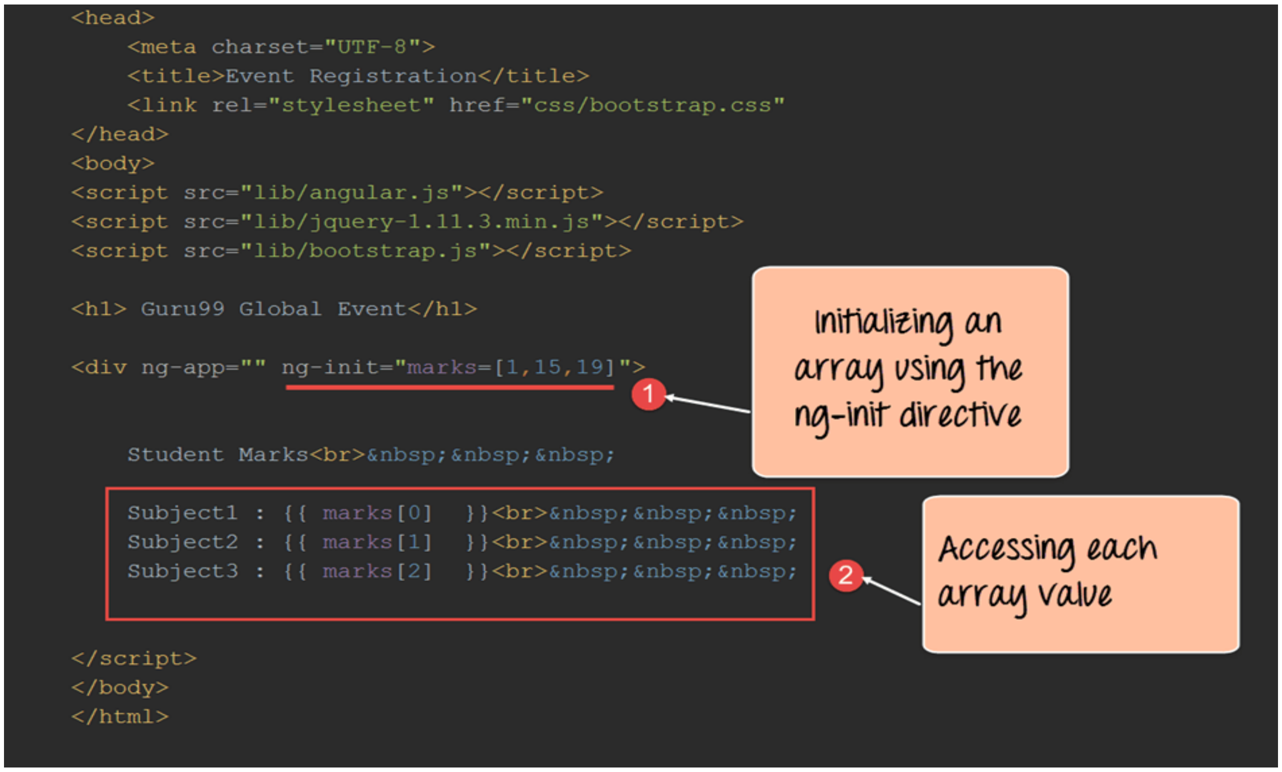
From the output,

- It can be clearly seen that the values of “firstName” and “secondName” are displayed on the screen.

AngularJS Arrays

Expressions can be used to work with arrays as well. Let’s look at an example of Angular JS expressions with arrays.

In this example, we are going to define an array which is going to hold the marks of a student in 3 subjects. In the view, we will display the value of these marks accordingly.



```
<head>
    <meta charset="UTF-8">
    <title>Event Registration</title>
    <link rel="stylesheet" href="css/bootstrap.css"
</head>
<body>
<script src="lib/angular.js"></script>
<script src="lib/jquery-1.11.3.min.js"></script>
<script src="lib/bootstrap.js"></script>

<h1> Guru99 Global Event</h1>

<div ng-app="" ng-init="marks=[1,15,19]">


    Student Marks<br>&nbsp;&nbsp;&nbsp;

    Subject1 : {{ marks[0]  }}<br>&nbsp;&nbsp;&nbsp;
    Subject2 : {{ marks[1]  }}<br>&nbsp;&nbsp;&nbsp;
    Subject3 : {{ marks[2]  }}<br>&nbsp;&nbsp;&nbsp;

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

1
Initializing an array using the ng-init directive
2
Accessing each array value

<!DOCTYPE html>
<html>
<head>
    <meta chrset="UTF 8">
    <title>Event Registration</title>
</head>
<body>

<script src="https://code.angularjs.org/1.6.9/angular-route.js"></script>
<script src="https://code.angularjs.org/1.6.9/angular.min.js"></script>

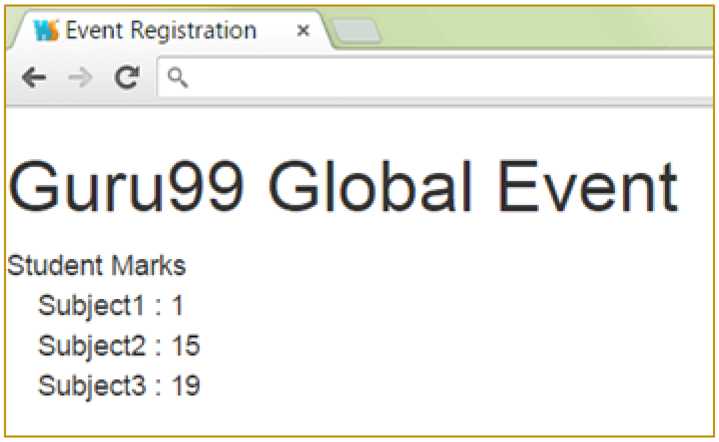
```
<h1> Guru99 Global Event</h1>

<div ng-app="" ng-init="marks=[1,15,19]">

    Student Marks<br>&nbsp;&nbsp;&nbsp;
    Subject1 : {{marks[0] }}<br>&nbsp;&nbsp;&nbsp;
    Subject2 : {{marks[1] }}<br>&nbsp;&nbsp;&nbsp;
    Subject3 : {{marks[2] }}<br>&nbsp;&nbsp;&nbsp;
</div>

</body>
</html>
```

Code Explanation:

1. The ng-init directive is used define the array with the name “marks” with 3 values of 1, 15 and 19.
2. We are then using expressions of marks [index] to access each element of the array.

If the code is executed successfully, the following Output will be shown when you run your code in the browser.

Output:



Guru99 Global Event

Student Marks
Subject1 : 1
Subject2 : 15
Subject3 : 19

From the output, it can be clearly seen that the marks being displayed, that are defined in the array.

NODEJS

Node.js tutorial provides basic and advanced concepts of Node.js. Our Node.js tutorial is designed for beginners and professionals both.

Node.js is a cross-platform environment and library for running JavaScript applications which is used to create networking and server-side applications.

Our Node.js tutorial includes all topics of Node.js such as Node.js installation on windows and linux, REPL, package manager, callbacks, event loop, os, path, query string, cryptography, debugger, URL, DNS, Net, UDP, process, child processes, buffers, streams, file systems, global objects, web modules etc. There are also given Node.js interview questions to help you better understand the Node.js technology.

What is Node.js

Node.js is a cross-platform runtime environment and library for running JavaScript applications outside the browser. It is used for creating server-side and networking web applications. It is open source and free to use.

Many of the basic modules of Node.js are written in JavaScript. Node.js is mostly used to run real-time server applications.

The definition given by its official documentation is as follows:

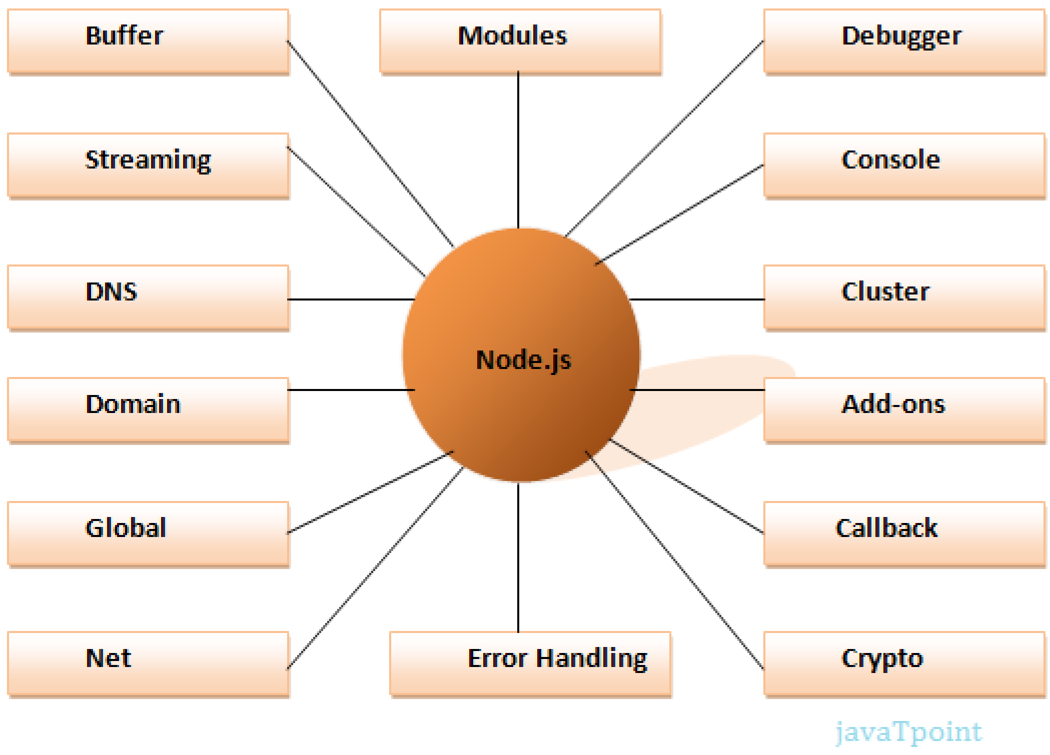
?Node.js is a platform built on Chrome's JavaScript runtime for easily building fast and scalable network applications. Node.js uses an event-driven, non-blocking I/O model that makes it lightweight and efficient, perfect for data-intensive real-time applications that run across distributed devices.?

Node.js also provides a rich library of various JavaScript modules to simplify the development of web applications.

1. Node.js = Runtime Environment + JavaScript Library

Different parts of Node.js

The following diagram specifies some important parts of Node.js:



Features of Node.js

Following is a list of some important features of Node.js that makes it the first choice of software architects.

1. **Extremely fast:** Node.js is built on Google Chrome's V8 JavaScript Engine, so its library is very fast in code execution.
2. **I/O is Asynchronous and Event Driven:** All APIs of Node.js library are asynchronous i.e. non-blocking. So a Node.js based server never waits for an API to return data. The server moves to the next API after calling it and a notification mechanism of Events of Node.js helps the server to get a response from the previous API call. It is also a reason that it is very fast.

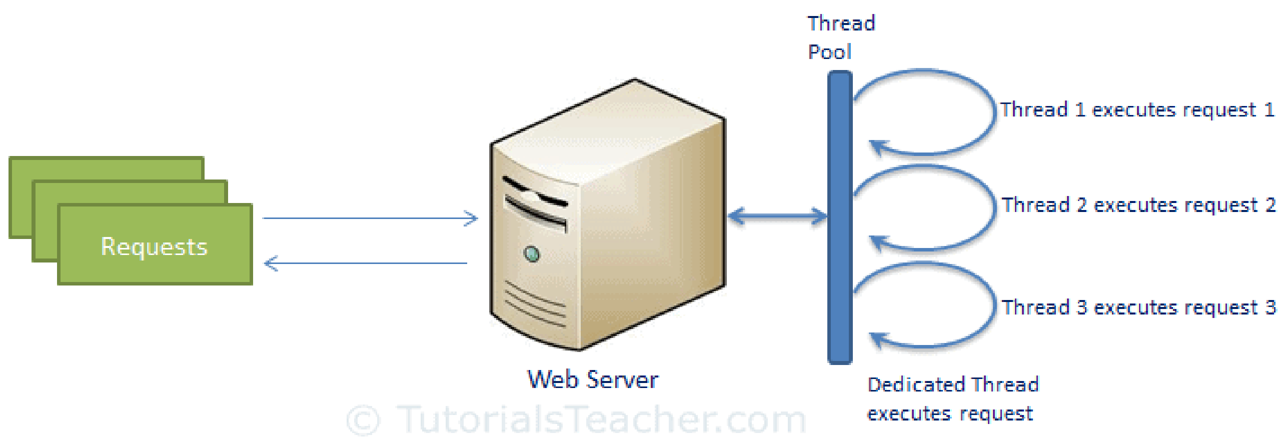
3. **Single threaded:** Node.js follows a single threaded model with event looping.
4. **Highly Scalable:** Node.js is highly scalable because event mechanism helps the server to respond in a non-blocking way.
5. **No buffering:** Node.js cuts down the overall processing time while uploading audio and video files. Node.js applications never buffer any data. These applications simply output the data in chunks.
6. **Open source:** Node.js has an open source community which has produced many excellent modules to add additional capabilities to Node.js applications.
7. **License:** Node.js is released under the MIT license.

_Node.js Process Model

In this section, we will learn about the Node.js process model and understand why we should use Node.js.

Traditional Web Server Model

In the traditional web server model, each request is handled by a dedicated thread from the thread pool. If no thread is available in the thread pool at any point of time then the request waits till the next available thread. Dedicated thread executes a particular request and does not return to thread pool until it completes the execution and returns a response.


Thread Pool
Requests
Web Server
Thread 1 executes request 1
Thread 2 executes request 2
Thread 3 executes request 3
Dedicated Thread executes request
© TutorialsTeacher.com

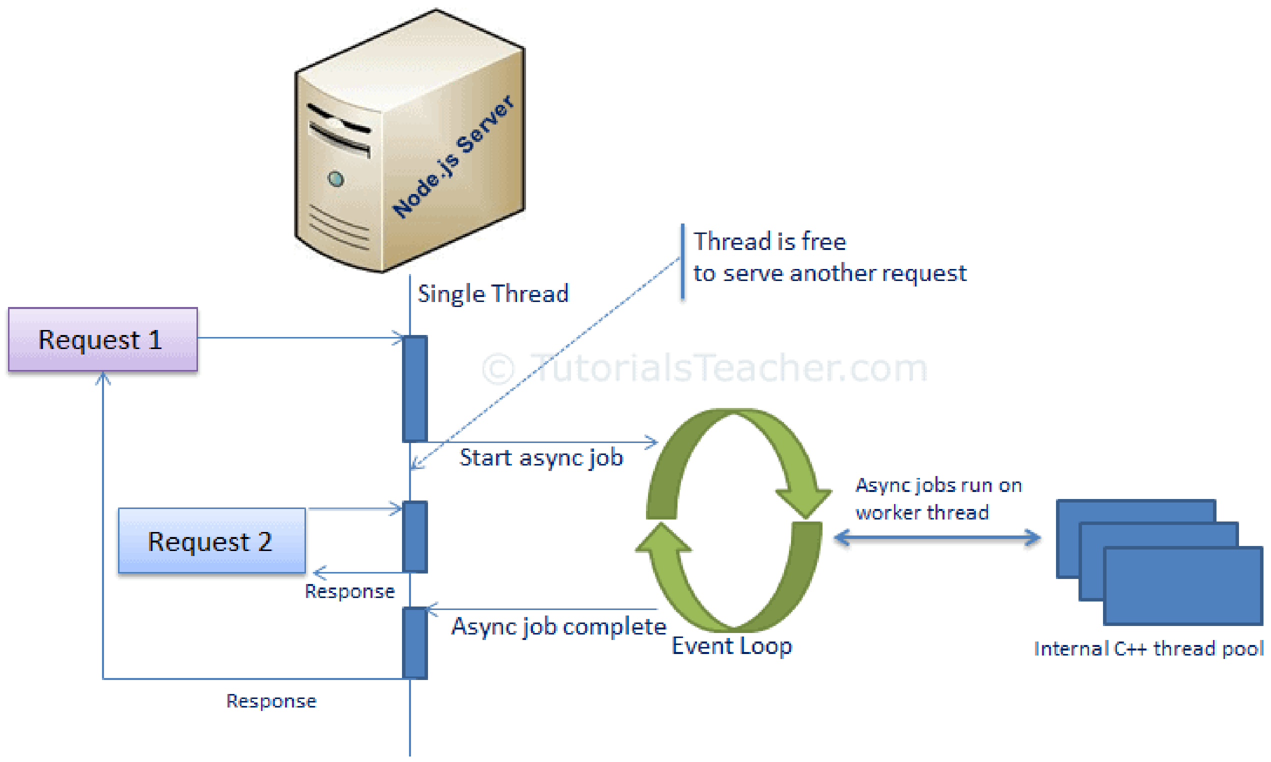
Traditional Web Server Model

Node.js Process Model

Node.js processes user requests differently when compared to a traditional web server model. Node.js runs in a single process and the application code runs in a single thread and thereby needs less resources than other platforms. All the user requests to your web application will be handled by a single thread and all the I/O work or long running job is performed asynchronously for a particular request. So, this single thread doesn't have to wait for the request to complete and is free to handle the next request. When asynchronous I/O work completes then it processes the request further and sends the response.

An event loop is constantly watching for the events to be raised for an asynchronous job and executing callback function when the job completes. Internally, Node.js uses libev for the event loop which in turn uses internal C++ thread pool to provide asynchronous I/O.

The following figure illustrates asynchronous web server model using Node.js.



Node.js Process Model

Node.js process model increases the performance and scalability with a few caveats. Node.js is not fit for an application which performs CPU-intensive operations like image processing or other heavy computation work because it takes time to process a request and thereby blocks the single thread.

Node.js Module

Module in Node.js is a simple or complex functionality organized in single or multiple JavaScript files which can be reused throughout the Node.js application.

Each module in Node.js has its own context, so it cannot interfere with other modules or pollute global scope. Also, each module can be placed in a separate .js file under a separate folder.

Node.js implements CommonJS modules standard. CommonJS is a group of volunteers who define JavaScript standards for web server, desktop, and console application.

Node.js Module Types

Node.js includes three types of modules:

1. Core Modules
2. Local Modules
3. Third Party Modules

Node.js Core Modules

Node.js is a light weight framework. The core modules include bare minimum functionalities of Node.js. These core modules are compiled into its binary distribution and load automatically when Node.js process starts. However, you need to import the core module first in order to use it in your application.

The following table lists some of the important core modules in Node.js.

Core Module	Description
http	http module includes classes, methods and events to create Node.js http server.
url	url module includes methods for URL resolution and parsing.
querystring	querystring module includes methods to deal with query string.
path	path module includes methods to deal with file paths.
fs	fs module includes classes, methods, and events to work with file I/O.
util	util module includes utility functions useful for programmers.

Loading Core Modules

In order to use Node.js core or NPM modules, you first need to import it using require() function as shown below.

```
var module = require('module_name');
```

As per above syntax, specify the module name in the require() function. The require() function will return an object, function, property or any other JavaScript type, depending on what the specified module returns.

The following example demonstrates how to use Node.js http module to create a web server.

Example: Load and Use Core http Module

Copy

```
var http = require('http');

var server = http.createServer(function(req, res){

  //write code here

});

server.listen(5000);
```

In the above example, require() function returns an object because http module returns its functionality as an object, you can then use its properties and methods using dot notation e.g. http.createServer().

Node.js Local Module

Local modules are modules created locally in your Node.js application. These modules include different functionalities of your application in separate files and folders. You can also package it and distribute it via NPM, so that Node.js community can use it. For example, if you need to connect to MongoDB and fetch data then you can create a module for it, which can be reused in your application.

Writing Simple Module

Let's write simple logging module which logs the information, warning or error to the console.

In Node.js, module should be placed in a separate JavaScript file. So, create a Log.js file and write the following code in it.

Log.js

Copy

```
var log = {
            info: function (info) {
               console.log('Info: ' + info);
            },
            warning:function (warning) {
               console.log('Warning: ' + warning);
            },
            error:function (error) {
               console.log('Error: ' + error);
            }
   };

module.exports = log
```

In the above example of logging module, we have created an object with three functions - info(), warning() and error(). At the end, we have assigned this object to **module.exports**. The module.exports in the above example exposes a log object as a module.

The *module.exports* is a special object which is included in every JS file in the Node.js application by default. Use **module.exports** or **exports** to expose a function, object or variable as a module in Node.js.

Now, let's see how to use the above logging module in our application.

Loading Local Module

To use local modules in your application, you need to load it using require() function in the same way as core module. However, you need to specify the path of JavaScript file of the module.

The following example demonstrates how to use the above logging module contained in Log.js.

app.js

Copy

```
var myLogModule = require('./Log.js');

myLogModule.info('Node.js started');
```

In the above example, app.js is using log module. First, it loads the logging module using require() function and specified path where logging module is stored. Logging module is contained in Log.js file in the root folder. So, we have specified the path './Log.js' in the require() function. The '.' denotes a root folder.

The require() function returns a log object because logging module exposes an object in Log.js using module.exports. So now you can use logging module as an object and call any of its function using dot notation e.g myLogModule.info() or myLogModule.warning() or myLogModule.error()

Run the above example using command prompt (in Windows) as shown below.

```
C:\> node app.js
```

```
Info: Node.js started
```

Thus, you can create a local module using module.exports and use it in your application.

Export Module in Node.js

Here, you will learn how to expose different types as a module using module.exports.

The module.exports is a special object which is included in every JavaScript file in the Node.js application by default. The module is a variable that represents the current module, and exports is an object that will be exposed as a module. So, whatever you assign to module.exports will be exposed as a module.

Let's see how to expose different types as a module using module.exports.

Export Literals

As mentioned above, exports is an object. So it exposes whatever you assigned to it as a module. For example, if you assign a string literal then it will expose that string literal as a module.

The following example exposes simple string message as a module in Message.js.

Message.js

 Copy

```
module.exports = 'Hello world';
```

Now, import this message module and use it as shown below.

app.js

 Copy

```
var msg = require('./Messages.js');

console.log(msg);
```

Run the above example and see the result, as shown below.

```
C:\> node app.js
```

```
Hello World
```

Note:

You must specify ./ as a path of root folder to import a local module. However, you do not need to specify the path to import Node.js core modules or NPM modules in the require() function.

Export Object

The exports is an object. So, you can attach properties or methods to it. The following example exposes an object with a string property in Message.js file.

Message.js

Copy

```
exports.SimpleMessage = 'Hello world';

//or

module.exports.SimpleMessage = 'Hello world';
```

In the above example, we have attached a property SimpleMessage to the exports object. Now, import and use this module, as shown below.

app.js

Copy

```
var msg = require('./Messages.js');

console.log(msg.SimpleMessage);
```

In the above example, the require() function will return an object { SimpleMessage : 'Hello World'} and assign it to the msg variable. So, now you can use msg.SimpleMessage.

Run the above example by writing node app.js in the command prompt and see the output as shown below.

```
C:\> node app.js
```

```
Hello World
```

In the same way as above, you can expose an object with function. The following example exposes an object with the log function as a module.

Log.js

Copy

```
module.exports.log = function (msg) {
    console.log(msg);
};
```

The above module will expose an object- { log : function(msg){ console.log(msg); } } . Use the above module as shown below.

app.js

Copy

```
var msg = require('./Log.js');

msg.log('Hello World');
```

Run and see the output in command prompt as shown below.

```
C:\> node app.js
```

```
Hello World
```

You can also attach an object to module.exports, as shown below.

data.js

Copy

```
module.exports = {
    firstName: 'James',
    lastName: 'Bond'
}
```

app.js

Copy

```
var person = require('./data.js');
console.log(person.firstName + ' ' + person.lastName);
```

Run the above example and see the result, as shown below.

```
C:\>nodeapp.js
```

```
James Bond
```

Node.js File System

Node.js includes **fs** module to access physical file system. The fs module is responsible for all the asynchronous or synchronous file I/O operations.

Let's see some of the common I/O operation examples using fs module.

Reading File

Use fs.readFile() method to read the physical file asynchronously.

```
fs.readFile(fileName [,options], callback)
```

Parameter Description:

- filename: Full path and name of the file as a string.
- options: The options parameter can be an object or string which can include encoding and flag. The default encoding is utf8 and default flag is "r".
- callback: A function with two parameters err and fd. This will get called when readFile operation completes.

The following example demonstrates reading existing TestFile.txt asynchronously.

Example: Reading File

Copy

```
var fs = require('fs');

fs.readFile('TestFile.txt', function (err, data) {
                    if (err) throw err;
```

```
    console.log(data);
});
```

The above example reads TestFile.txt (on Windows) asynchronously and executes callback function when read operation completes. This read operation either throws an error or completes successfully. The err parameter contains error information if any. The data parameter contains the content of the specified file.

The following is a sample TextFile.txt file.

TextFile.txt

Copy

```
This is test file to test fs module of Node.js
```

Now, run the above example and see the result as shown below.

```
C:\> node server.js
```

```
This is test file to test fs module of Node.js
```

Use fs.readFileSync() method to read file synchronously as shown below.

Example: Reading File Synchronously

Copy

```
var fs = require('fs');

var data = fs.readFileSync('dummyfile.txt', 'utf8');
console.log(data);
```

Writing File

Use fs.writeFile() method to write data to a file. If file already exists then it overwrites the existing content otherwise it creates a new file and writes data into it.

fs.writeFile(filename, data[, options], callback)

Parameter Description:

- filename: Full path and name of the file as a string.
- Data: The content to be written in a file.
- options: The options parameter can be an object or string which can include encoding, mode and flag. The default encoding is utf8 and default flag is "r".
- callback: A function with two parameters err and fd. This will get called when write operation completes.

The following example creates a new file called test.txt and writes "Hello World" into it asynchronously.

Example: Creating & Writing File

Copy

```
var fs = require('fs');

fs.writeFile('test.txt', 'Hello World!', function (err) {
                    if (err)
      console.log(err);
                    else
      console.log('Write operation complete.');
});
```

In the same way, use fs.appendFile() method to append the content to an existing file.

Example: Append File Content

Copy

```
var fs = require('fs');
```

```
fs.appendFile('test.txt', 'Hello World!', function (err) {
                    if (err)
      console.log(err);
                    else
      console.log('Append operation complete.');
});
```

Open File

Alternatively, you can open a file for reading or writing using fs.open() method.

```
fs.open(path, flags[, mode], callback)
```

Parameter Description:

- path: Full path with name of the file as a string.
- Flag: The flag to perform operation
- Mode: The mode for read, write or readwrite. Defaults to 0666 readwrite.
- callback: A function with two parameters err and fd. This will get called when file open operation completes.

Flags

The following table lists all the flags which can be used in read/write operation.

Flag	Description
r	Open file for reading. An exception occurs if the file does not exist.
r+	Open file for reading and writing. An exception occurs if the file does not exist.
rs	Open file for reading in synchronous mode.
rs+	Open file for reading and writing, telling the OS to open it synchronously. See notes for 'rs' this with caution.
w	Open file for writing. The file is created (if it does not exist) or truncated (if it exists).

Flag	Description
wx	Like 'w' but fails if path exists.
w+	Open file for reading and writing. The file is created (if it does not exist) or truncated (if it e
wx+	Like 'w+' but fails if path exists.
a	Open file for appending. The file is created if it does not exist.
ax	Like 'a' but fails if path exists.
a+	Open file for reading and appending. The file is created if it does not exist.
ax+	Like 'a+' but fails if path exists.

The following example opens an existing file and reads its content.

Example:File open and read

Copy

```
var fs = require('fs');

fs.open('TestFile.txt', 'r', function (err, fd) {

                    if (err) {
                    return console.error(err);
  }

                    var buffr = new Buffer(1024);

  fs.read(fd, buffr, 0, buffr.length, 0, function (err, bytes) {

                    if (err) throw err;

                    // Print only read bytes to avoid junk.
                    if (bytes > 0) {
        console.log(buffr.slice(0, bytes).toString());
      }
```

```
                    // Close the opened file.
        fs.close(fd, function (err) {
                    if (err) throw err;
        });
    });
});
```

Delete File

Use fs.unlink() method to delete an existing file.

```
fs.unlink(path, callback);
```

The following example deletes an existing file.

Example:File Open and Read

 Copy

```
var fs = require('fs');

fs.unlink('test.txt', function () {

    console.log('write operation complete.');

});
```

Important method of fs module

Method	Description
fs.readFile(fileName [,options], callback)	Reads existing file.
fs.writeFile(filename, data[, options], callback)	Writes to the file. If file exists then overwrite the content otherwise creates

Method	Description
	new file.
fs.open(path, flags[, mode], callback)	Opens file for reading or writing.
fs.rename(oldPath, newPath, callback)	Renames an existing file.
fs.chown(path, uid, gid, callback)	Asynchronous chown.
fs.stat(path, callback)	Returns fs.stat object which includes important file statistics.
fs.link(srcpath, dstpath, callback)	Links file asynchronously.
fs.symlink(destination, path[, type], callback)	Symlink asynchronously.
fs.rmdir(path, callback)	Renames an existing directory.
fs.mkdir(path[, mode], callback)	Creates a new directory.
fs.readdir(path, callback)	Reads the content of the specified directory.
fs.utimes(path, atime, mtime, callback)	Changes the timestamp of the file.
fs.exists(path, callback)	Determines whether the specified file exi
fs.access(path[, mode], callback)	Tests a user's permissions for the specifie
fs.appendFile(file, data[, options], callback)	Appends new content to the existing file.

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 WideWeb 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></br> or <br />

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>	<config>
<version>1.0</version>	<version>1.0</version>
</config>	</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>	<s:config>
<c:version>1.0</c:version>	<s:version>1.0</s:version>
</c:config>	</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.

<u>Document Type Declaration (DTD)</u>

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.

<u>1. Document Type Declaration(DTD):</u>

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.

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>
```

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());           }
  }
```

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

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>

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.

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>

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

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>
```

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());        }
}
```

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>
```

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"/>

- **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

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>
```

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>
```

<u>Annotations declaration:</u>

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>
```

<u>XML Scheme data types:</u>

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:

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

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

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

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 andvalue
- **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 forrmal 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>

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>
```

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.

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>
```

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>
```

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 &lt; and &amp; respectively. For this reason, it provides <text> element to add literal text to result with following syntax:

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

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>
```

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.

  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.

<u>Integrating PHP and AJAX</u>:-

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>
<?xmll 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:defiinitions 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/defiinitions”
        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).

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	1
			10 % 8	2
			10 % 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"

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>
```

UNIT-5

Web Servers (XAMPP, LAMP, Tomcat)

XAMPP TUTORIAL

XAMPP

XAMPP is one of the widely used cross-platform web servers, which helps developers to create and test their programs on a local webserver. It was developed by the **Apache Friends**, and its native source code can be revised or modified by the audience. It consists of **Apache HTTP Server, MariaDB, and interpreter** for the different programming languages like PHP and Perl. It is available in 11 languages and supported by different platforms such as the IA-32 package of Windows & x64 package of macOS and Linux.

What is XAMPP?

XAMPP is an abbreviation where ***X stands for Cross-Platform, A stands for Apache, M stands for MYSQL, and the Ps stand for PHP and Perl***, respectively. It is an open-source package of web solutions that includes Apache distribution for many servers and command-line executables along with modules such as Apache server, MariaDB, PHP, and Perl.

XAMPP helps a local host or server to test its website and clients via computers and laptops before releasing it to the main server. It is a platform that furnishes a suitable environment to test and verify the working of projects based on Apache, Perl, MySQL database, and PHP through the system of the host itself. Among these technologies, Perl is a programming language used for web development, PHP is a backend scripting language, and MariaDB is the most vividly used database developed by MySQL. The detailed description of these components is given below.

Components of XAMPP

As defined earlier, XAMPP is used to symbolize the classification of solutions for different technologies. It provides a base for testing of projects based on different technologies through a personal server. XAMPP is an abbreviated form of each alphabet representing each of its major components. This collection of software contains a web server named **Apache**, a database management system named **MariaDB** and scripting/ programming languages such as **PHP** and **Perl**. X denotes Cross-platform, which means that it can work on different platforms such as Windows, Linux, and macOS.

Many other components are also part of this collection of software and are explained below.

1. **Cross-Platform:** Different local systems have different configurations of operating systems installed in it. The component of cross-platform has been included to increase the utility and audience for this package of Apache distributions. It supports various platforms such as packages of Windows, Linus, and MAC OS.
2. **Apache:** It is an HTTP a cross-platform web server. It is used worldwide for delivering web content. The server application has made free for installation and used for the community of developers under the aegis of Apache Software Foundation. The

remote server of Apache delivers the requested files, images, and other documents to the user.

3. **MariaDB:** Originally, MySQL DBMS was a part of XAMPP, but now it has been replaced by MariaDB. It is one of the most widely used relational DBMS, developed by MySQL. It offers online services of data storage, manipulation, retrieval, arrangement, and deletion.
4. **PHP:** It is the backend scripting language primarily used for web development. PHP allows users to create dynamic websites and applications. It can be installed on every platform and supports a variety of database management systems. It was implemented using C language. PHP stands for **Hypertext Processor**. It is said to be derived from Personal Home Page tools, which explains its simplicity and functionality.
5. **Perl:** It is a combination of two high-level dynamic languages, namely Perl 5 and Perl 6. Perl can be applied for finding solutions for problems based on system administration, web development, and networking. Perl allows its users to program dynamic web applications. It is very flexible and robust.
6. **phpMyAdmin:** It is a tool used for dealing with MariaDB. Its version 4.0.4 is currently being used in XAMPP. Administration of DBMS is its main role.
7. **OpenSSL:** It is the open-source implementation of the Secure Socket Layer Protocol and Transport Layer Protocol. Presently version 0.9.8 is a part of XAMPP.

8. **XAMPP Control Panel:** It is a panel that helps to operate and regulate upon other components of the XAMPP. Version 3.2.1 is the most recent update. A detailed description of the control panel will be done in the next section of the tutorial.
9. **Webalizer:** It is a Web Analytics software solution used for User logs and provide details about the usage.
10. **Mercury:** It is a mail transport system, and its latest version is 4.62. It is a mail server, which helps to manage the mails across the web.
11. **Tomcat:** Version 7.0.42 is currently being used in XAMPP. It is a servlet based on JAVA to provide JAVA functionalities.
12. **Filezilla:** It is a File Transfer Protocol Server, which supports and eases the transfer operations performed on files. Its recently updated version is 0.9.41.

XAMPP Format Support

XAMPP is supported in three file formats:

- **.EXE**- It is an extension used to denote executable files making it accessible to install because an executable file can run on a computer as any normal program.
- **.7z - 7zip file**- This extension is used to denote compressed files that support multiple data compression and encryption algorithms. It is more favored by a formalist, although it requires working with more complex files.

- **.ZIP**- This extension supports lossless compression of files. A Zipped file may contain multiple compressed files. The **Deflate algorithm** is mainly used for compression of files supported by this format. The .ZIP files are quite tricky to install as compared to .EXE

Thus .EXE is the most straightforward format to install, while the other two formats are quite complicated and complex to install.

What is LAMP?

LAMP is an open-source Web development platform that uses ***Linux*** as the operating system, ***Apache*** as the Web server, ***MySQL*** as the relational database management system and ***PHP/Perl/Python*** as the object-oriented scripting language.

Sometimes LAMP is referred to as a LAMP stack because the platform has four layers. Stacks can be built on different operating systems.

LAMP is a example of a web service stack, named as an ***acronym***. The LAMP components are largely interchangeable and not limited to the original selection. LAMP is suitable for building dynamic web sites and web applications.

Since its creation, the LAMP model has been adapted to another component, though typically consisting of free and open-source software.

- intrusion prevention (IPS) system and Snort an intrusion detection (IDS)
- RRD tool for diagrams
- Nagios, Cacti, or Collectd for monitoring

LAMP Stack Components

Linux based web servers consist of four software components. These components are arranged in layers supporting one another and make up the software stack. Websites and Web Applications run on top of this underlying stack. The common software components are as follows:

1. **Linux:** Linux started in 1991. It sets the foundation for the stack model. All other layers are run on top of this layer. It is an open-source and free operating system. It is endured partly because it's flexible, and other operating systems are harder to configure.
2. **Apache:** The second layer consists of web server software, typically Apache Web Server. This layer resides on top of the Linux layer.
   Apache HTTP Server is a free web server software package made available under an open-source license. It used to be known as Apache Web Server when it was created in 1995. It offers a secure and extendable Web server that's in sync with current HTTP standards. Web servers are responsible for translating from web browsers to their correct website.
3. **MySQL:** MySQL is a relational database management system used to store application data. It is an open-source and keeps all

the data in a format that can easily be queried with the SQL language.
SQL works great with well-structured business domains, and a great workhorse that can handle even the most extensive and most complicated websites with ease.
MySQL stores details that can be queried by scripting to construct a website. MySQL usually sits on top of the Linux layer alongside Apache. In high-end configurations, MySQL can be offloaded to a separate host server.

4. **PHP:** The scripting layer consists of PHP and other similar web programming languages.
   The PHP open-source scripting language works with Apache to create dynamic web pages. We cannot use HTML to perform dynamic processes such as pulling data out of a database. To provide this type of functionality, we drop PHP code into the parts of a page that you want to be dynamic. Websites and Web Applications run within this layer.
   PHP is designed for efficiency. It makes programming easier and allowing to write new code, hit refresh, and immediately see the resulting changes without the need for compiling.

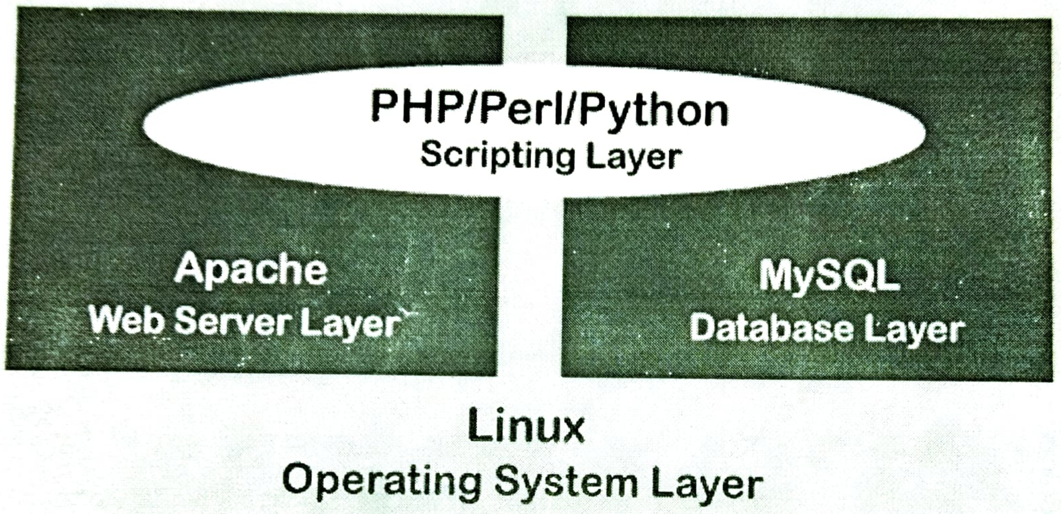
LAMP Architecture

LAMP has classic layered architecture, with Linux at the lowest level. The next layer is Apache and MySQL, followed by PHP.

Although PHP is at the top or presentation layer, the PHP component sits inside Apache.

Developers that use these tools with a Windows operating system instead of Linux are said to be using ***WAMP***, with a Macintosh system ***MAMP***, and with a Solaris system ***SAMP***.

Linux, Apache, MySQL and PHP, all of them add something unique to the development of high-performance web applications. Originally popularized from the phrase Linux, Apache, MySQL, and PHP, the acronym LAMP now refers to a generic software stack model.



The modularity of a LAMP stack may vary. Still, this particular software combination has become popular because it is sufficient to host a wide variety of website frameworks, such as ***Joomla***, ***Drupal***, and ***WordPress***.

The components of the LAMP stack are present in the software repositories of the most Linux distributions. The LAMP bundle can be combined with many other free and open-source software packages, such as the following:

- netsniff-ng for security testing and hardening

LAMP is open source and non-proprietary so we can avoid lock-in. We have the flexibility to select the right components for specific projects or business requirements.

LAMP offers flexibility in other ways as well. Apache is modular in design, and we will find there are existing, customizable modules available for many different extensions. These modules range from support for other languages to authentication capabilities.

Another advantage of LAMP is its secure architecture and well-established encryption practices that have been proven in the enterprise.

Efficiency

LAMP can help to reduce development time because it is an open-source stack that has been available for more than a decade.

We can build on what other people have done in the past and make it own. Work within an Apache module that gets 80% of the way there customize the last 20%, and save considerable time as a result.

Advantages of LAMP

LAMP has the following advantages, such as:

1. The LAMP stack consists of four components, all of which are examples of ***Free*** and ***Open-Source Software (FOSS)***. As they are free and available for download, it attracts the attention of many users who wish to avoid paying large sums of money when developing their website.

2. Because it is FOSS, the source code of the software is shared and available for people to make changes and improvements, enhancing its overall performance.
3. The LAMP stack has proven to be a secure and stable platform thanks to its vast community that contributes when any problems arise.
4. We can easily customize the stack and interchange the components with other open-source software to suit the needs.

LAMP Stack Alternatives

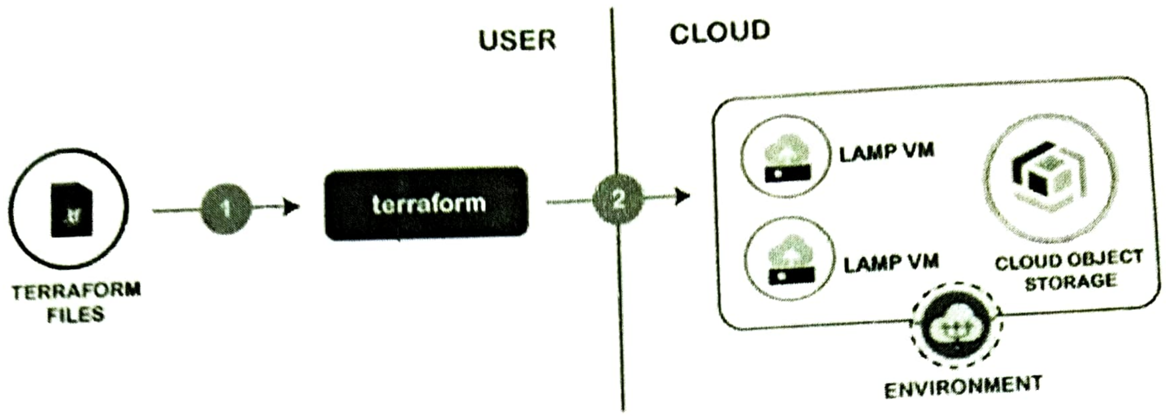
There are several variants of the four stack model as well. These variants use alternative software, replacing one or more of the standard components.

Open-source alternatives are:

- **LEMP**(Linux, NGINX, MySQL/MariaDB, PHP/Perl/Python)
- **LAPP**(Linux, Apache, PostgreSQL, PHP)
- **LEAP**(Linux, Eucalyptus, AppScale, Python)
- **LLMP**(Linux, Lighttpd, MySQL/MariaDB, PHP/Perl/Python)

While non-open source alternatives include:

- **WAMP**(Windows, Apache, MySQL/MariaDB, PHP/Perl/Python)
- **WIMP**(Windows, Internet Information Services, MySQL/MariaDB, PHP/Perl/Python)
- **MAMP**(Mac OS x, Apache, MySQL/MariaDB, PHP/Perl/



The LAMP stack order of execution shows how the elements interoperate. The process starts when the Apache webserver receives requests for web pages from a user's browser. If the request is for a PHP file, Apache passes the request to PHP, which loads the file and executes the code contained in the file. PHP also communicates with MySQL to fetch any data referenced in the code.

PHP then uses the code in the file and the data from the database to create the HTML that browsers require to display web pages. The LAMP stack is efficient at handling not only static web pages but also dynamic pages where the content may change each time it is loaded depending on the date, time, user identity and other factors.

After running the file code, PHP then passes the resulting data back to the Apache webserver to send to the browser. It can also store this new data in MySQL. And of course, all of these operations are enabled by the Linux operating system running at the base of the stack.

Flexibility

Although LAMP uses Linux as the OS, we can use the other components with an alternative OS to meet specific needs. For example, there is a WAMP stack, which uses Microsoft Windows.

UNIT-5

Servlet technology is used to create a web application (resides at server side and generates a dynamic web page).

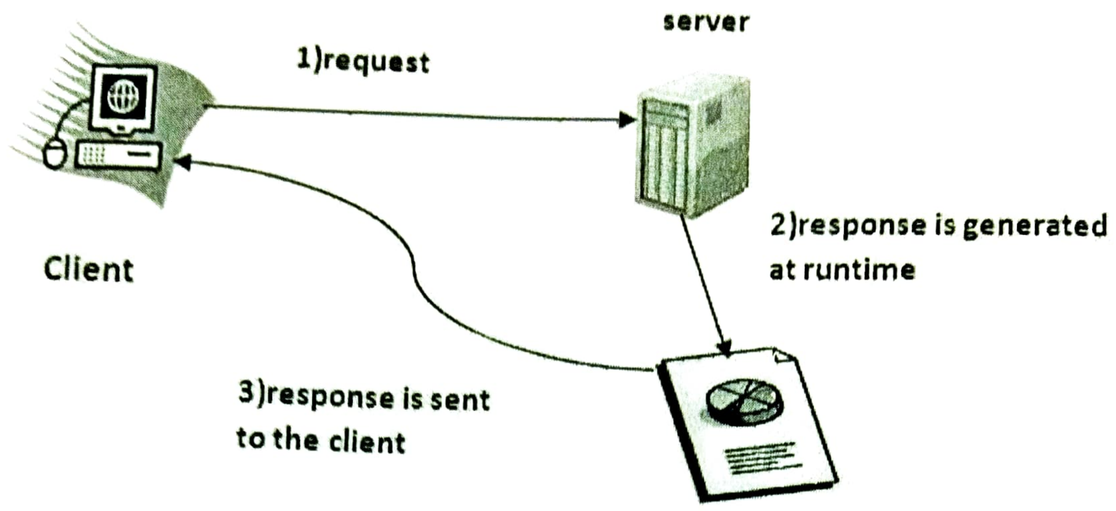
Servlet technology is robust and scalable because of java language. Before Servlet, CGI (Common Gateway Interface) scripting language was common as a server-side programming language. However, there were many disadvantages to this technology. We have discussed these disadvantages below.

There are many interfaces and classes in the Servlet API such as Servlet, GenericServlet, HttpServlet, ServletRequest, ServletResponse, etc.

What is a Servlet?

Servlet can be described in many ways, depending on the context.

- Servlet is a technology which is used to create a web application.
- Servlet is an API that provides many interfaces and classes including documentation.
- Servlet is an interface that must be implemented for creating any Servlet.
- Servlet is a class that extends the capabilities of the servers and responds to the incoming requests. It can respond to any requests.
- Servlet is a web component that is deployed on the server to create a dynamic web page.


server
1)request
Client
2)response is generated at runtime
3)response is sent to the client

- What is the web application and what is the difference between Get and Post request?
- What information is received by the web server if we request for a Servlet?
- How to run servlet in Eclipse, MyEclipse and Netbeans IDE?
- What are the ways for servlet collaboration and what is the difference between RequestDispatcher and sendRedirect() method?
- What is the difference between ServletConfig and ServletContext interface?
- How many ways can we maintain the state of a user? Which approach is mostly used in web development?
- How to count the total number of visitors and whole response time for a request using Filter?
- How to run servlet with annotation?

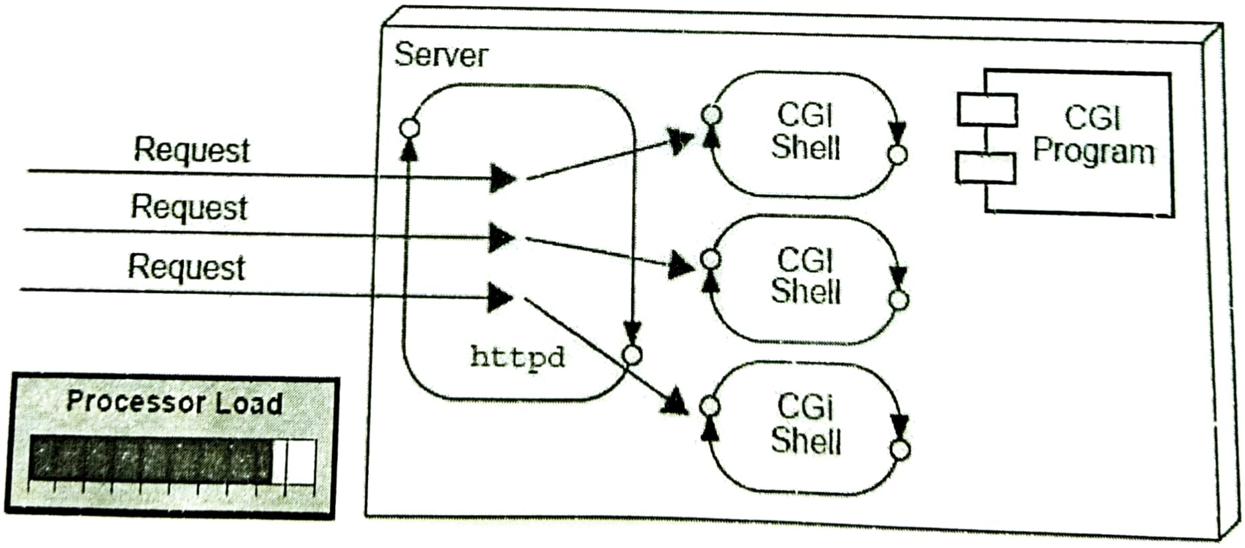
- How to create registration form using Servlet and Oracle database?
- How can we upload and download the file from the server?

What is a web application?

A web application is an application accessible from the web. A web application is composed of web components like Servlet, JSP, Filter, etc. and other elements such as HTML, CSS, and JavaScript. The web components typically execute in Web Server and respond to the HTTP request.

CGI (Common Gateway Interface)

CGI technology enables the web server to call an external program and pass HTTP request information to the external program to process the request. For each request, it starts a new process.

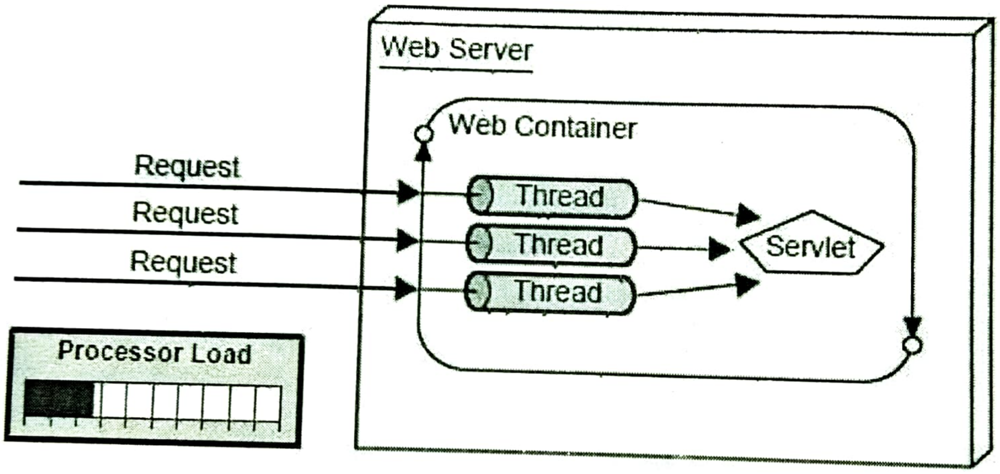


Disadvantages of CGI

There are many problems in CGI technology:

1. If the number of clients increases, it takes more time for sending the response.
2. For each request, it starts a process, and the web server is limited to start processes.
3. It uses platform dependent language e.g. C, C++, perl.

Advantages of Servlet



There are many advantages of Servlet over CGI. The web container creates threads for handling the multiple requests to the Servlet. Threads have many benefits over the Processes such as they share a common memory area, lightweight, cost of communication between the threads are low. The advantages of Servlet are as follows:

1. **Better performance:** because it creates a thread for each request, not process.
2. **Portability:** because it uses Java language.

3. **Robust:** JVM manages Servlets, so we don't need to worry abo
   the memory leak, garbage collection, etc.
4. **Secure:** because it uses java language.

Servlet Interface

Servlet interface provides commonbehaviorto all the servlets.Servlet interface defines methods that all servlets must implement.

Servlet interface needs to be implemented for creating any servlet (either directly or indirectly). It provides 3 life cycle methods that are used to initialize the servlet, to service the requests, and to destroy the servlet and 2 non-life cycle methods.

Methods of Servlet interface

There are 5 methods in Servlet interface. The init, service and destroy are the life cycle methods of servlet. These are invoked by the web container.

Method	Description
public void init(ServletConfig config)	initializes the servlet. It is the life cycle method of servlet and invoked by the web container only once.
public void service(ServletRequest request,ServletResponse response)	provides response for the incoming request. It is invoked at each request by the web container.
public void destroy()	is invoked only once and indicates that servlet is being destroyed.
public ServletConfig getServletConfig()	returns the object of ServletConfig.
public String getServletInfo()	returns information about servlet such as writer, copyright, version etc.

Steps to create a servlet example

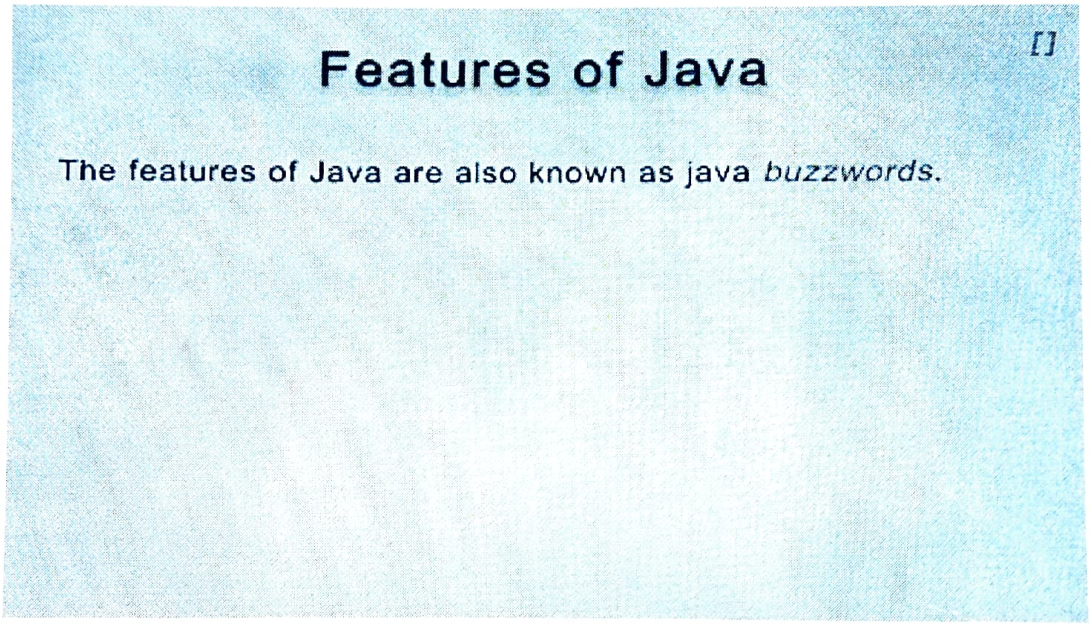
There are given 6 steps to create a **servlet example**. These steps are required for all the servers.

The servlet example can be created by three ways:

1. By implementing Servlet interface,
2. By inheriting GenericServlet class, (or)
3. By inheriting HttpServlet class

The mostly used approach is by extending HttpServlet because it provides http request specific method such as doGet(), doPost(), doHead() etc.

Here, we are going to use **apache tomcat server** in this example. The steps are as follows:



Features of Java

The features of Java are also known as java *buzzwords.*

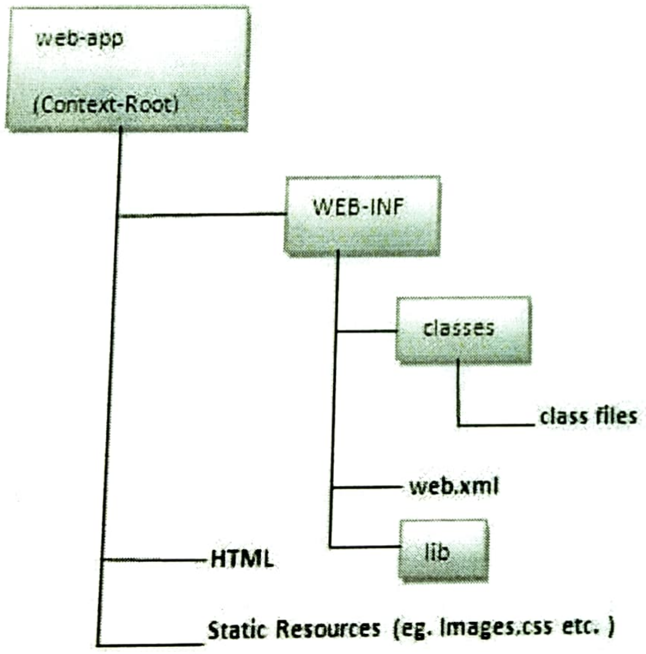
1. Create a directory structure
2. Create a Servlet
3. Compile the Servlet
4. Create a deployment descriptor
5. Start the server and deploy the project
6. Access the servlet

download this example of servlet
download example of servlet by extending GenericServlet
download example of servlet by implementing Servlet interface

1)Create a directory structures

The **directory structure** defines that where to put the different types of files so that web container may get the information and respond to the client.

The Sun Microsystem defines a unique standard to be followed by all the server vendors. Let's see the directory structure that must be followed to create the servlet.



As you can see that the servlet class file must be in the classes folder. The web.xml file must be under the WEB-INF folder.

2)Create a Servlet

There are three ways to create the servlet.

1. By implementing the Servlet interface
2. By inheriting the GenericServlet class
3. By inheriting the HttpServlet class

The HttpServlet class is widely used to create the servlet because it provides methods to handle http requests such as doGet(), doPost, doHead() etc.

In this example we are going to create a servlet that extends the HttpServlet class. In this example, we are inheriting the HttpServlet class and providing the implementation of the doGet() method. Notice that get request is the default request.

DemoServlet.java

```
import javax.servlet.http.*;
import javax.servlet.*;
import java.io.*;
public class DemoServlet extends HttpServlet{
public void doGet(HttpServletRequest req,HttpServletResponse res)
throws ServletException,IOException
{
res.setContentType("text/html");//setting the content type
PrintWriter pw=res.getWriter();//get the stream to write the data

//writing html in the stream
pw.println("<html><body>");
pw.println("Welcome to servlet");
pw.println("</body></html>");

pw.close();//closing the stream
}}
```

3)Compile the servlet

For compiling the Servlet, jar file is required to be loaded. Different Servers provide different jar files:

Jar file	Server
	Apache Tomcat

⇧ SCROLL TO TOP

2) weblogic.jar	Weblogic
3) javaee.jar	Glassfish
4) javaee.jar	JBoss

Two ways to load the jar file

1. set classpath
2. paste the jar file in JRE/lib/ext folder

Put the java file in any folder. After compiling the java file, paste the class file of servlet in **WEB-INF/classes** directory.

4)Create the deployment descriptor (web.xml file)

The **deployment descriptor** is an xml file, from which Web Container gets the information about the servet to be invoked.

The web container uses the Parser to get the information from the web.xml file. There are many xml parsers such as SAX, DOM and Pull.

There are many elements in the web.xml file. Here is given some necessary elements to run the simple servlet program.

web.xml file

```
<web-app>
```

SCROLL TO TOP

```
onoojaiswal</servlet-name>
```

```
<servlet-class>DemoServlet</servlet-class>
</servlet>

<servlet-mapping>
<servlet-name>sonoojaiswal</servlet-name>
<url-pattern>/welcome</url-pattern>
</servlet-mapping>

</web-app>
```

Description of the elements of web.xml file

There are too many elements in the web.xml file. Here is the illustration of some elements that is used in the above web.xml file. The elements are as follows:

<web-app> represents the whole application.

<servlet> is sub element of <web-app> and represents the servlet.

<servlet-name> is sub element of <servlet> represents the name of the servlet.

<servlet-class> is sub element of <servlet> represents the class of the servlet.

<servlet-mapping> is sub element of <web-app>. It is used to map the servlet.

<url-pattern> is sub element of <servlet-mapping>. This pattern is used at client side to invoke the servlet.

5)Start the Server and deploy the project

To start Apache Tomcat server, double click on the startup.bat file under apache-tomcat/bin directory.

One Time Configuration for Apache Tomcat Server

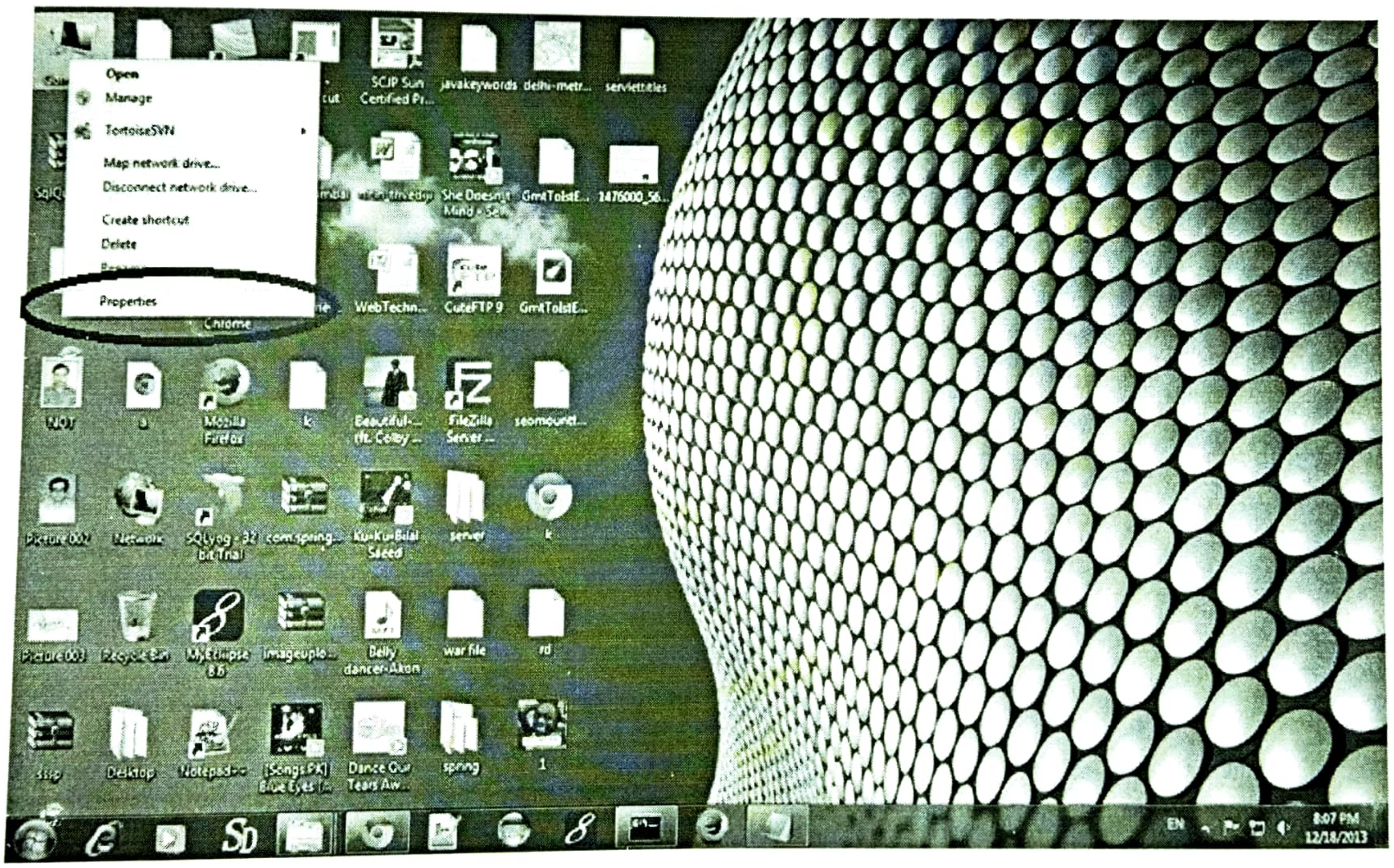
You need to perform 2 tasks:

1. set JAVA_HOME or JRE_HOME in environment variable (It is required to start server).
2. Change the port number of tomcat (optional). It is required if another server is running on same port (8080).

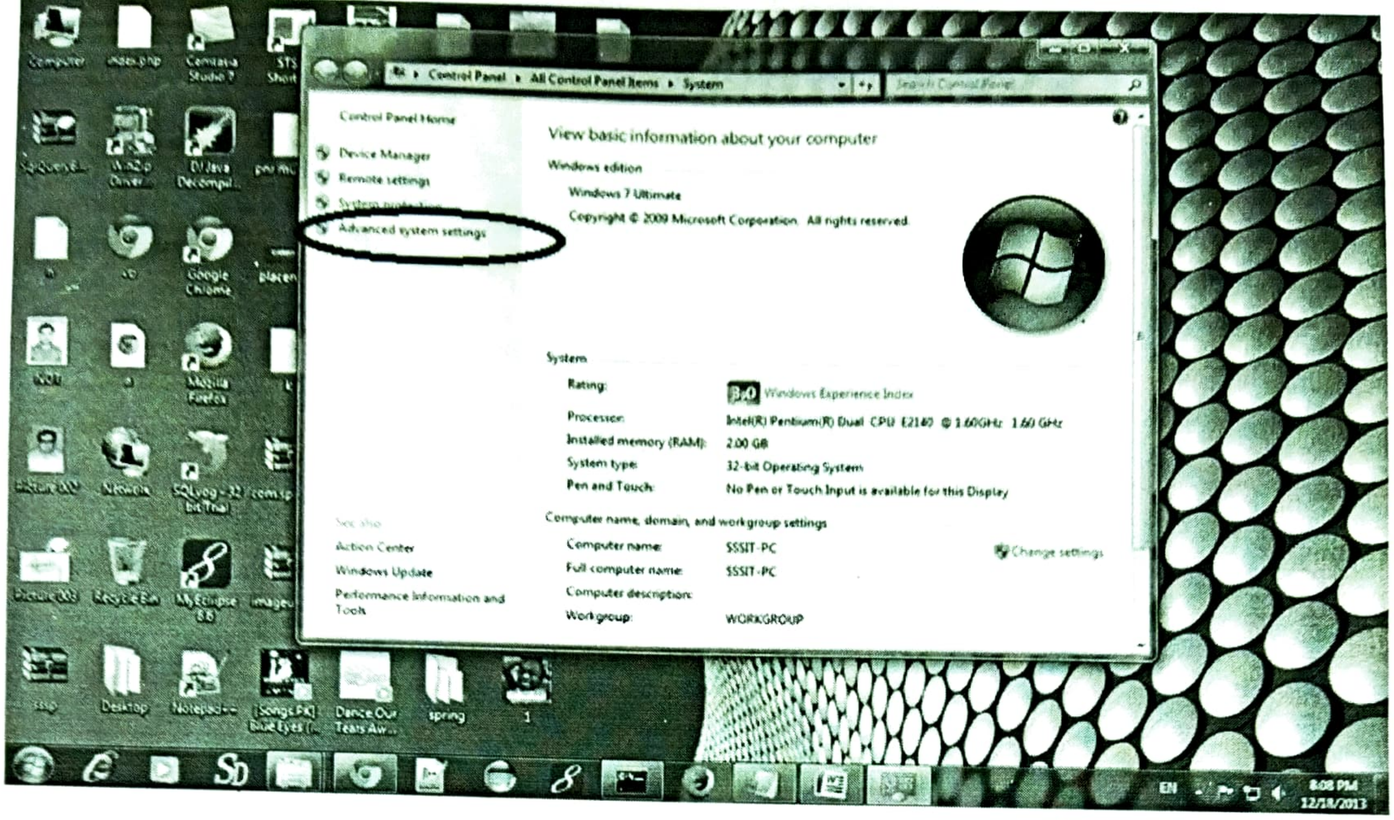
To start Apache Tomcat server JAVA_HOME and JRE_HOME must be set in Environment variables.

Go to My Computer properties -> Click on advanced tab then environment variables -> Click on the new tab of user variable -> Write JAVA_HOME in variable name and paste the path of jdk folder in variable value -> ok -> ok -> ok.

Go to My Computer properties:



Click on advanced system settings tab then environment variables:



View basic information about your computer

Windows edition

Windows 7 Ultimate

Copyright © 2009 Microsoft Corporation. All rights reserved.

System

Rating: 3.0 Windows Experience Index

Processor: Intel(R) Pentium(R) Dual CPU E2140 @ 1.60GHz 1.60 GHz

Installed memory (RAM): 2.00 GB

System type: 32-bit Operating System

Pen and Touch: No Pen or Touch Input is available for this Display

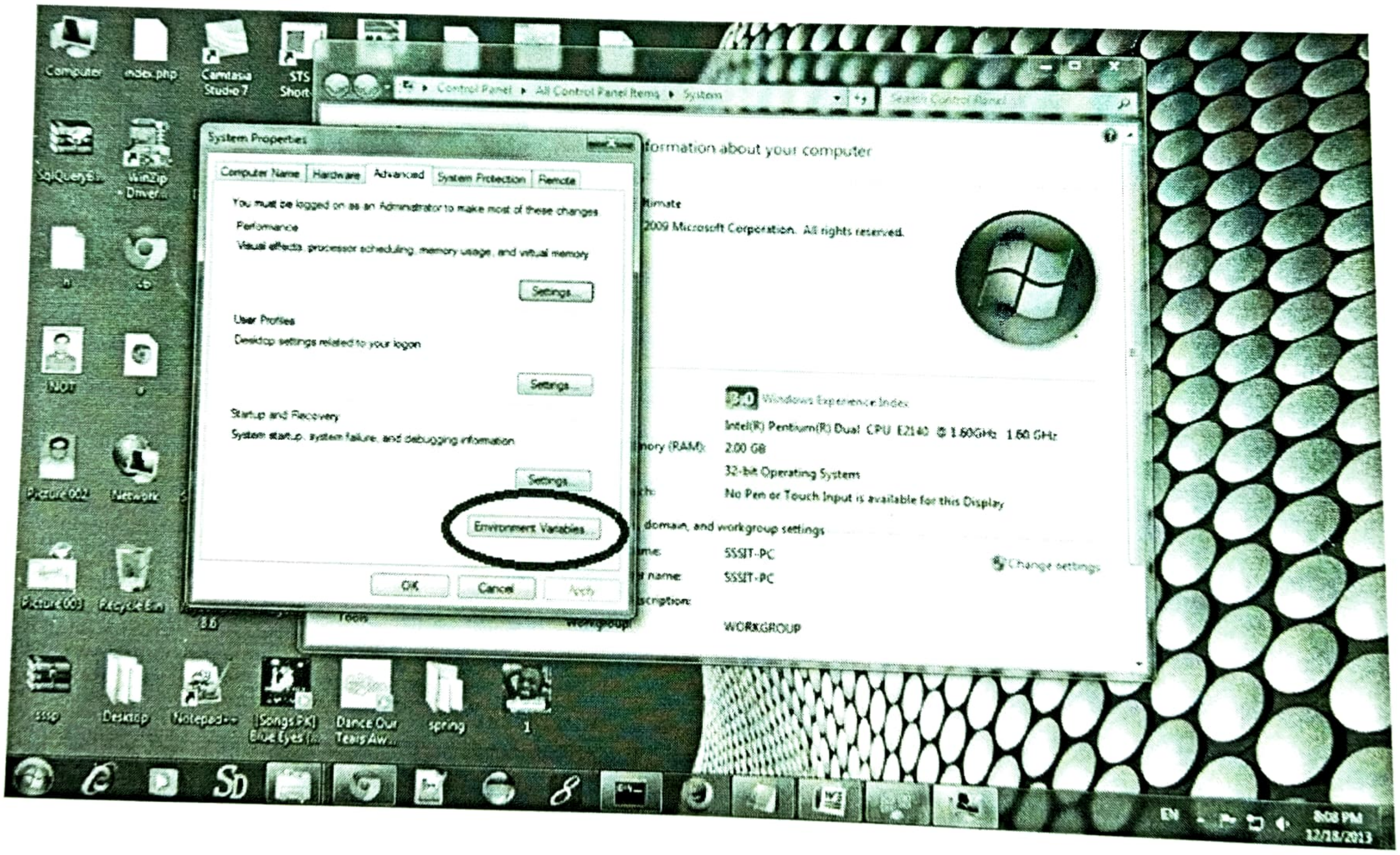
Computer name, domain, and workgroup settings

Computer name: SSSIT-PC

Full computer name: SSSIT-PC

Computer description:

Workgroup: WORKGROUP



You must be logged on as an Administrator to make most of these changes.

Performance

Visual effects, processor scheduling, memory usage, and virtual memory

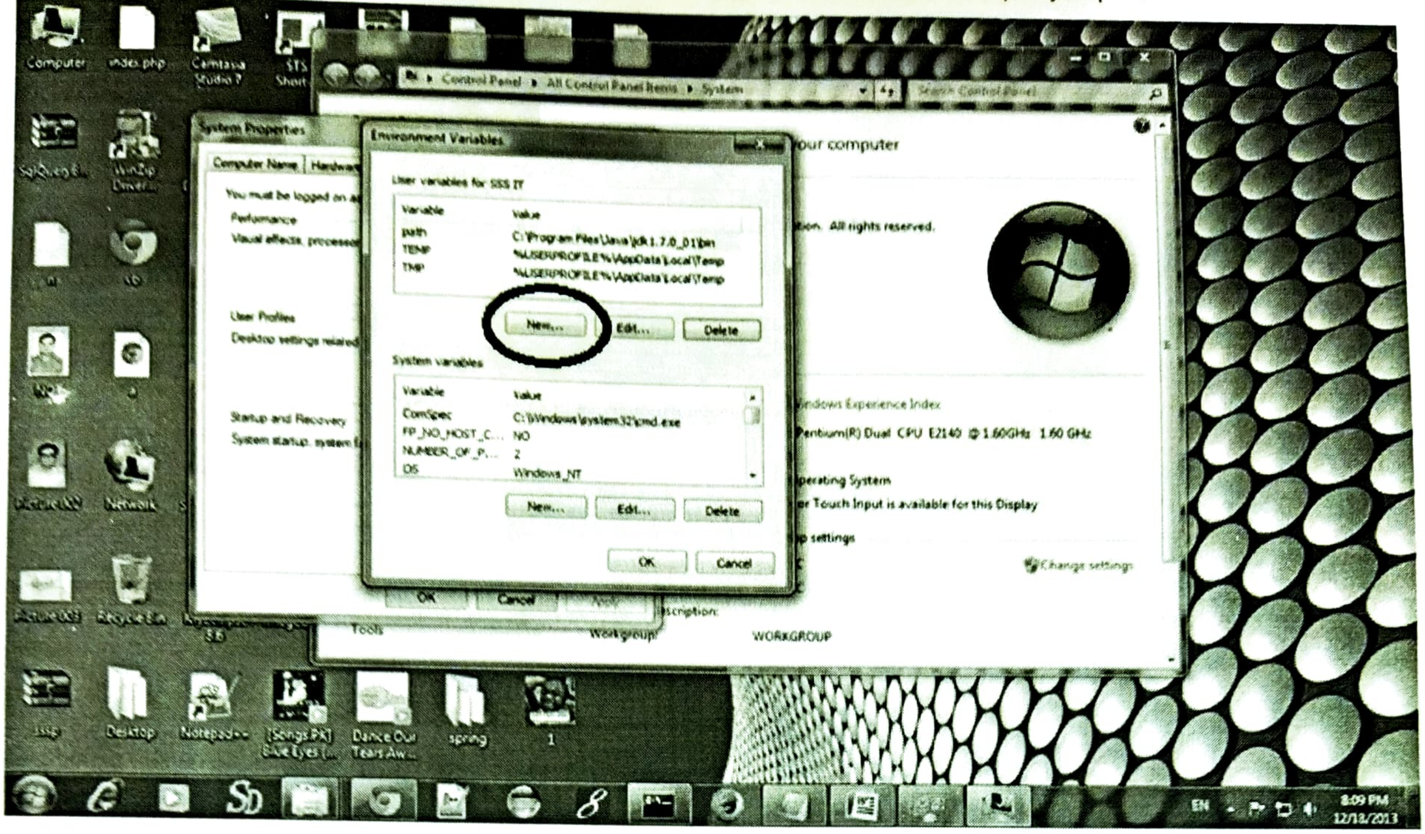
User Profiles

Desktop settings related to your logon

Startup and Recovery

System startup, system failure, and debugging information

Click on the new tab of user variable or system variable:



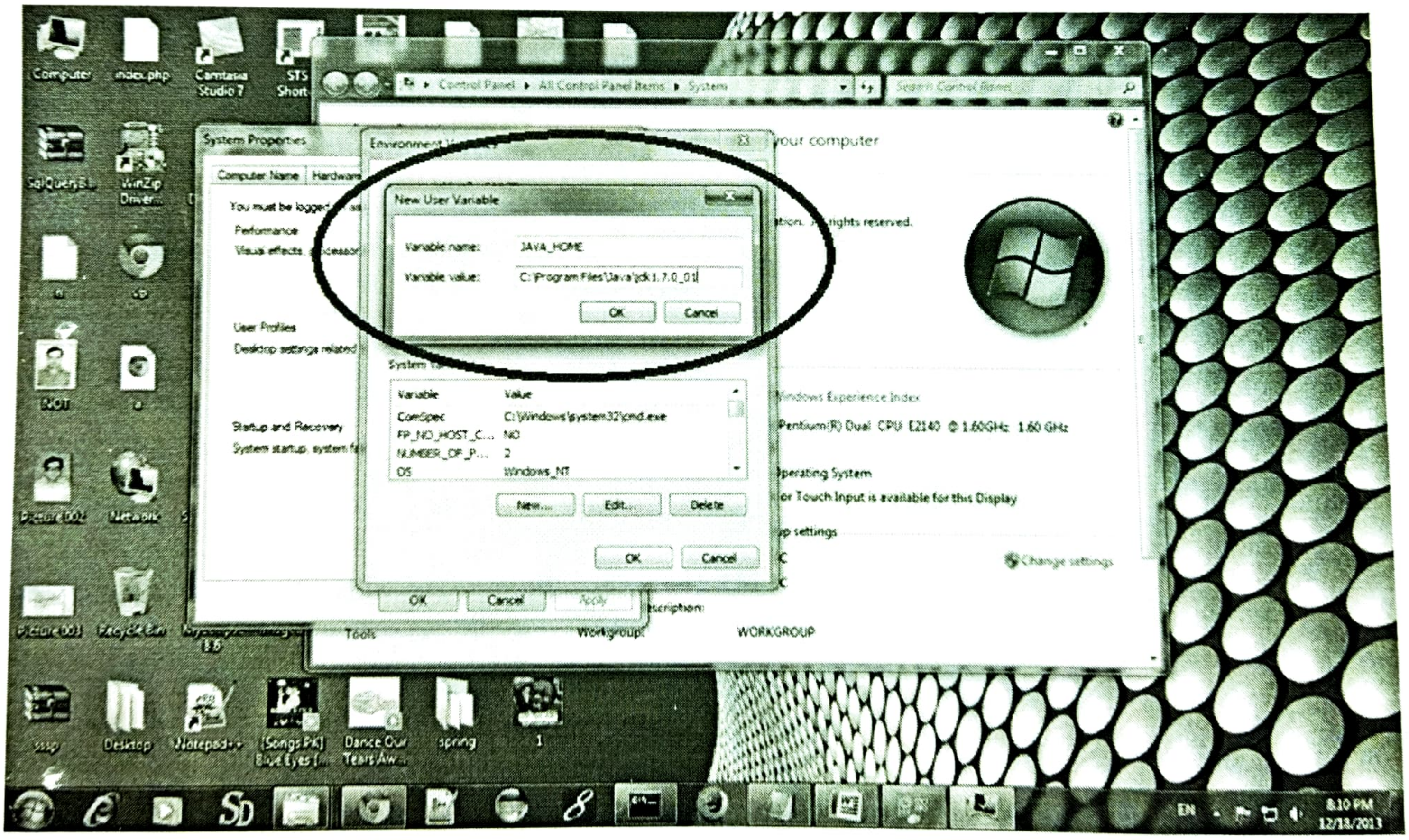
User variables for SSS IT

Variable	Value
path	C:\Program Files\Java\jdk1.7.0_01\bin
TEMP	%USERPROFILE%\AppData\Local\Temp
TMP	%USERPROFILE%\AppData\Local\Temp

System variables

Variable	Value
ComSpec	C:\Windows\system32\cmd.exe
FP_NO_HOST_C...	NO
NUMBER_OF_P...	2
OS	Windows_NT

Write JAVA_HOME in variable name and paste the path of jdk folder in variable value:



Variable name: JAVA_HOME

Variable value: C:\Program Files\Java\jdk1.7.0_01

Variable	Value
ComSpec	C:\Windows\system32\cmd.exe
FP_NO_HOST_C...	NO
NUMBER_OF_P...	2
OS	Windows_NT

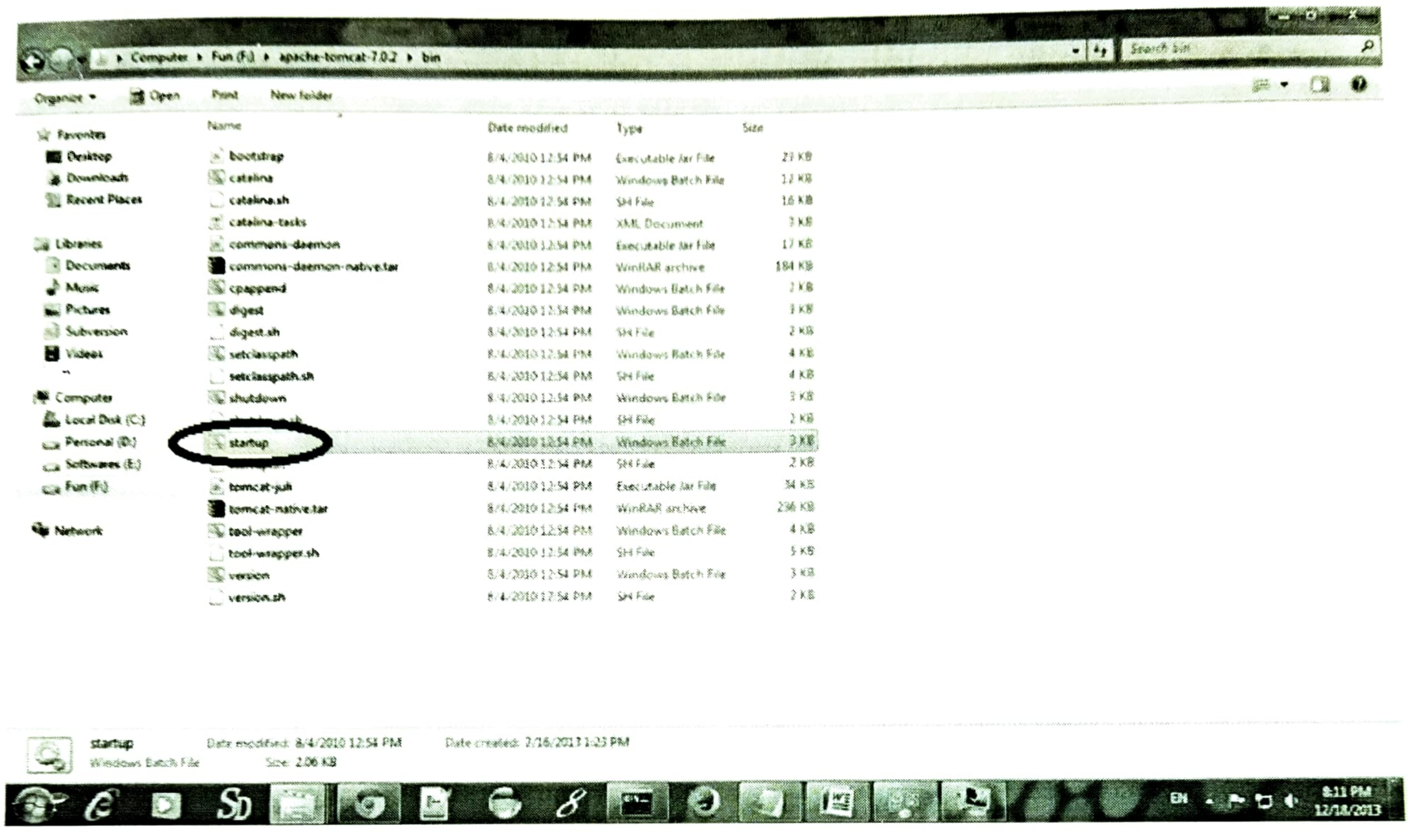
There must not be semicolon (;) at the end of the path.

After setting the JAVA_HOME double click on the startup.bat file in apache tomcat/bin.

Note: There are two types of tomcat available:

1. Apache tomcat that needs to extract only (no need to install)
2. Apache tomcat that needs to install

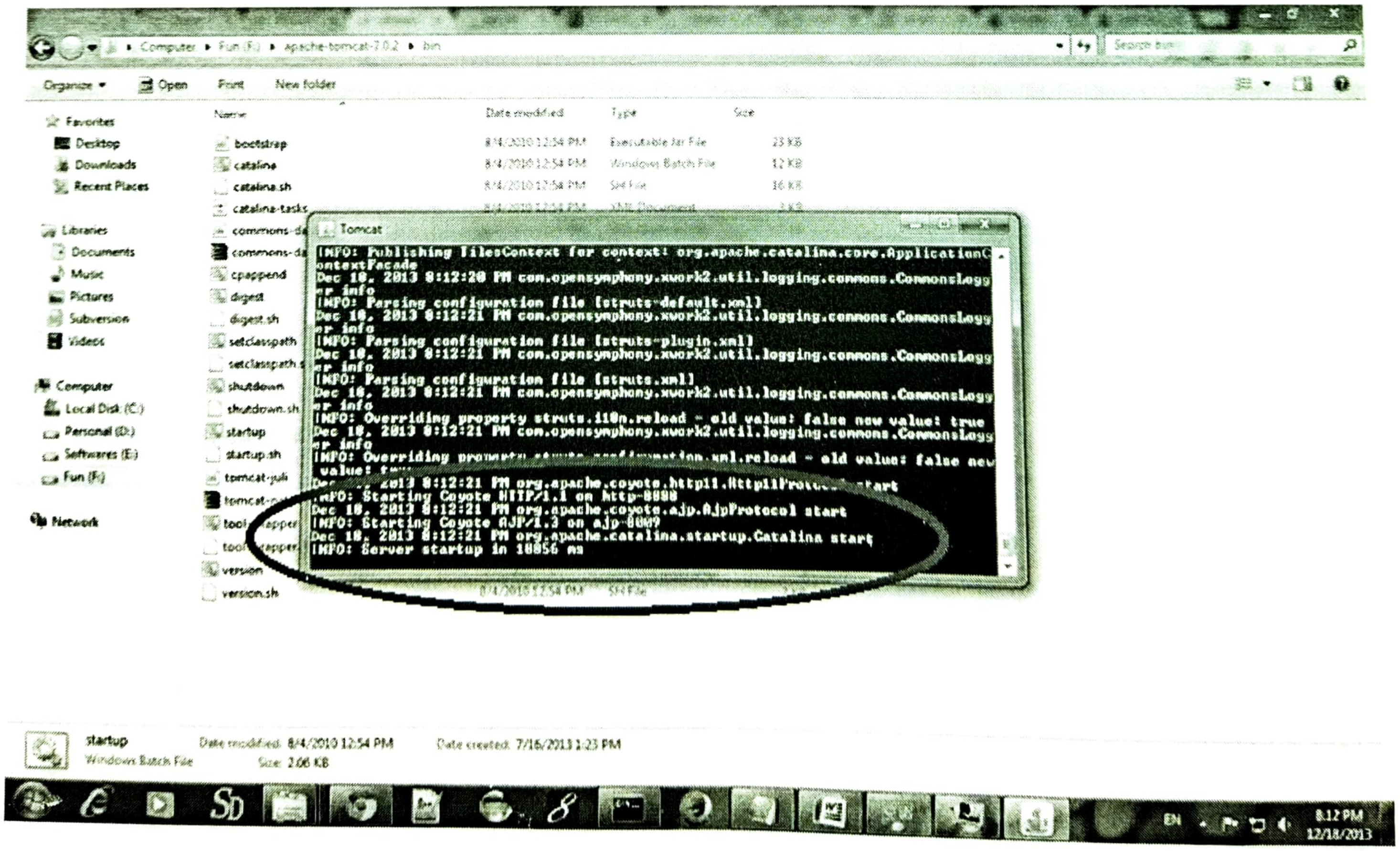
It is the example of apache tomcat that needs to extract only.



Computer ▸ Fun (F:) ▸ apache-tomcat-7.0.2 ▸ bin

Name	Date modified	Type	Size
bootstrap	8/4/2010 12:54 PM	Executable Jar File	23 KB
catalina	8/4/2010 12:54 PM	Windows Batch File	12 KB
catalina.sh	8/4/2010 12:54 PM	SH File	16 KB
catalina-tasks	8/4/2010 12:54 PM	XML Document	3 KB
commons-daemon	8/4/2010 12:54 PM	Executable Jar File	17 KB
commons-daemon-native.tar	8/4/2010 12:54 PM	WinRAR archive	184 KB
cpappend	8/4/2010 12:54 PM	Windows Batch File	2 KB
digest	8/4/2010 12:54 PM	Windows Batch File	3 KB
digest.sh	8/4/2010 12:54 PM	SH File	2 KB
setclasspath	8/4/2010 12:54 PM	Windows Batch File	4 KB
setclasspath.sh	8/4/2010 12:54 PM	SH File	4 KB
shutdown	8/4/2010 12:54 PM	Windows Batch File	3 KB
shutdown.sh	8/4/2010 12:54 PM	SH File	2 KB
startup	8/4/2010 12:54 PM	Windows Batch File	3 KB
startup.sh	8/4/2010 12:54 PM	SH File	2 KB
tomcat-juli	8/4/2010 12:54 PM	Executable Jar File	34 KB
tomcat-native.tar	8/4/2010 12:54 PM	WinRAR archive	236 KB
tool-wrapper	8/4/2010 12:54 PM	Windows Batch File	4 KB
tool-wrapper.sh	8/4/2010 12:54 PM	SH File	5 KB
version	8/4/2010 12:54 PM	Windows Batch File	3 KB
version.sh	8/4/2010 12:54 PM	SH File	2 KB

startup Date modified: 8/4/2010 12:54 PM Date created: 7/16/2013 1:23 PM
Windows Batch File Size: 2.06 KB



Tomcat

```
INFO: Publishing TilesContext for context: org.apache.catalina.core.ApplicationC
ontextFacade
Dec 18, 2013 8:12:20 PM com.opensymphony.xwork2.util.logging.commons.CommonsLogg
er info
INFO: Parsing configuration file [struts-default.xml]
Dec 18, 2013 8:12:21 PM com.opensymphony.xwork2.util.logging.commons.CommonsLogg
er info
INFO: Parsing configuration file [struts-plugin.xml]
Dec 18, 2013 8:12:21 PM com.opensymphony.xwork2.util.logging.commons.CommonsLogg
er info
INFO: Parsing configuration file [struts.xml]
Dec 18, 2013 8:12:21 PM com.opensymphony.xwork2.util.logging.commons.CommonsLogg
er info
INFO: Overriding property struts.i18n.reload - old value: false new value: true
Dec 18, 2013 8:12:21 PM com.opensymphony.xwork2.util.logging.commons.CommonsLogg
er info
INFO: Overriding property struts.configuration.xml.reload - old value: false new
 value: true
Dec 18, 2013 8:12:21 PM org.apache.coyote.http11.Http11Protocol start
INFO: Starting Coyote HTTP/1.1 on http-8080
Dec 18, 2013 8:12:21 PM org.apache.coyote.ajp.AjpProtocol start
INFO: Starting Coyote AJP/1.3 on ajp-8009
Dec 18, 2013 8:12:21 PM org.apache.catalina.startup.Catalina start
INFO: Server startup in 10056 ms
```

startup Date modified: 8/4/2010 12:54 PM Date created: 7/16/2013 1:23 PM
Windows Batch File Size: 2.06 KB

Now server is started successfully.

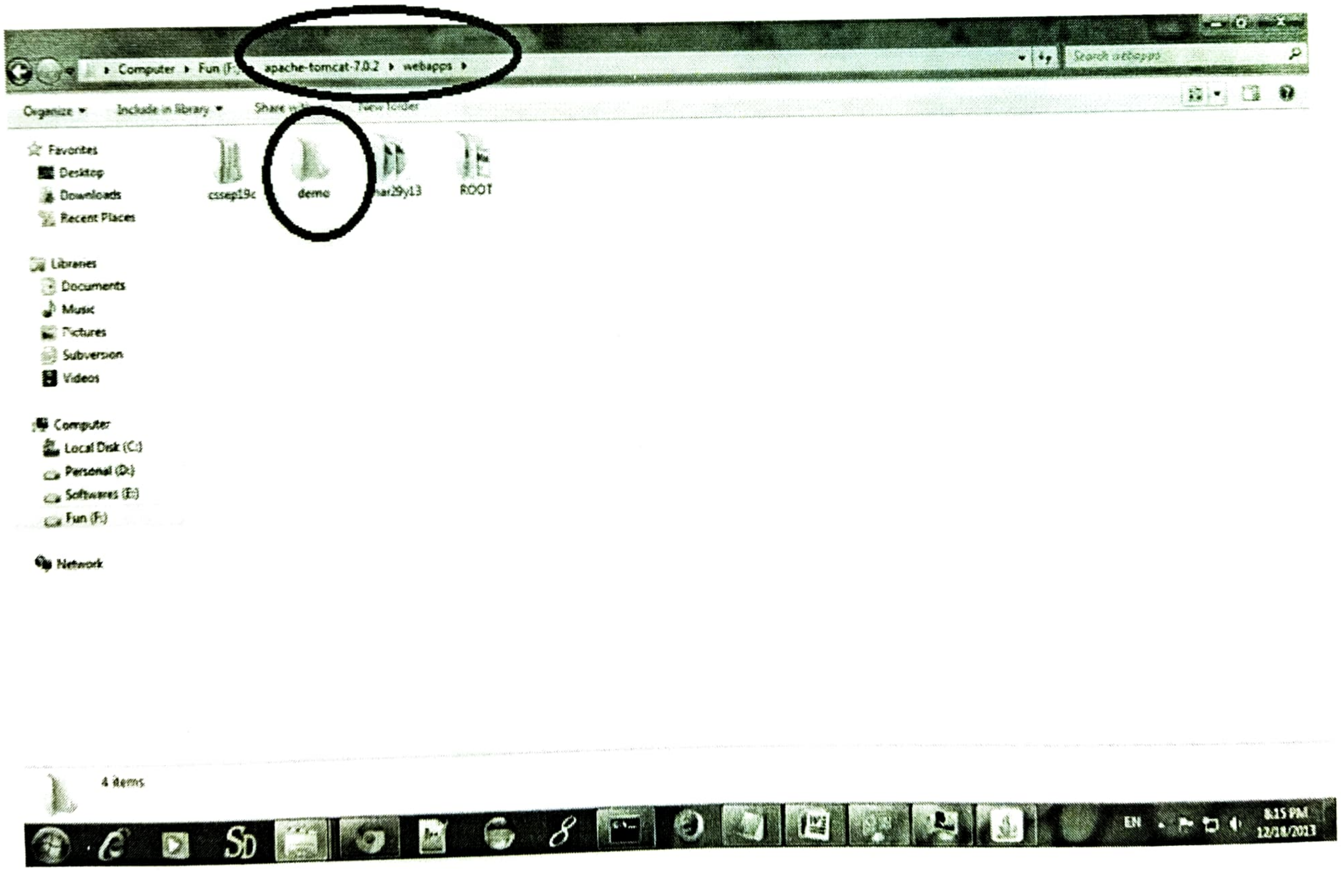
2) How to change port number of apache tomcat

Changing the port number is required if there is another server running on the same system with same port number.Suppose you have installed oracle, you need to change the port number of apache tomcat because both have the default port number 8080.

Open **server.xml file** in notepad. It is located inside the **apache-tomcat/conf** directory . Change the Connector port = 8080 and replace 8080 by any four digit number instead of 8080. Let us replace it by 9999 and save this file.

5) How to deploy the servlet project

Copy the project and paste it in the webapps folder under apache tomcat.



But there are several ways to deploy the project. They are as follows:

⇧ SCROLL TO TOP

context(project) folder into the webapps directory

war folder into the webapps directory

- By selecting the folder path from the server
- By selecting the war file from the server

Here, we are using the first approach.

You can also create war file, and paste it inside the webapps directory. To do so, you need to use jar too! to create the war file. Go inside the project directory (before the WEB-INF), then write:

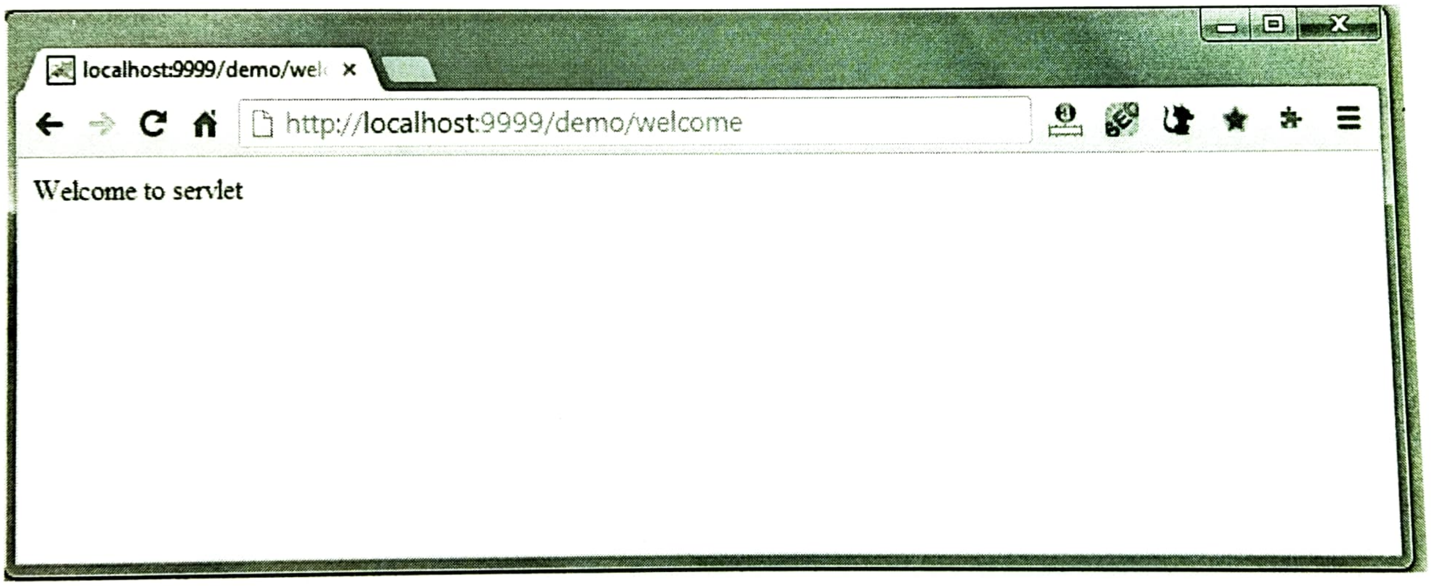
```
projectfolder> jar cvf myproject.war *
```

Creating war file has an advantage that moving the project from one location to another takes less time.

6) How to access the servlet

Open broser and write http://hostname:portno/contextroot/urlpatternofservlet. For example:

```
http://localhost:9999/demo/welcome
```



```
Welcome to servlet
```

download this example of servlet (using notepad)
download example of servlet by extending GenericServlet
download example of servlet by implementing Servlet interface

JSP Tutorial

JSP technology is used to create web application just like Servlet technology. It can be thought of as an extension to Servlet because it provides more functionality than servlet such as expression language, JSTL, etc.

A JSP page consists of HTML tags and JSP tags. The JSP pages are easier to maintain than Servlet because we can separate designing and development. It provides some additional features such as Expression Language, Custom Tags, etc.

Advantages of JSP over Servlet

There are many advantages of JSP over the Servlet. They are as follows:

1) Extension to Servlet

JSP technology is the extension to Servlet technology. We can use all the features of the Servlet in JSP. In addition to, we can use implicit objects, predefined tags, expression language and Custom tags in JSP, that makes JSP development easy.

2) Easy to maintain

JSP can be easily managed because we can easily separate our business logic with presentation logic. In Servlet technology, we mix our business logic with the presentation logic.

3) Fast Development: No need to recompile and redeploy

If JSP page is modified, we don't need to recompile and redeploy the project. The Servlet code needs to be updated and recompiled if we have to change the look and feel of the application.

4) Less code than Servlet

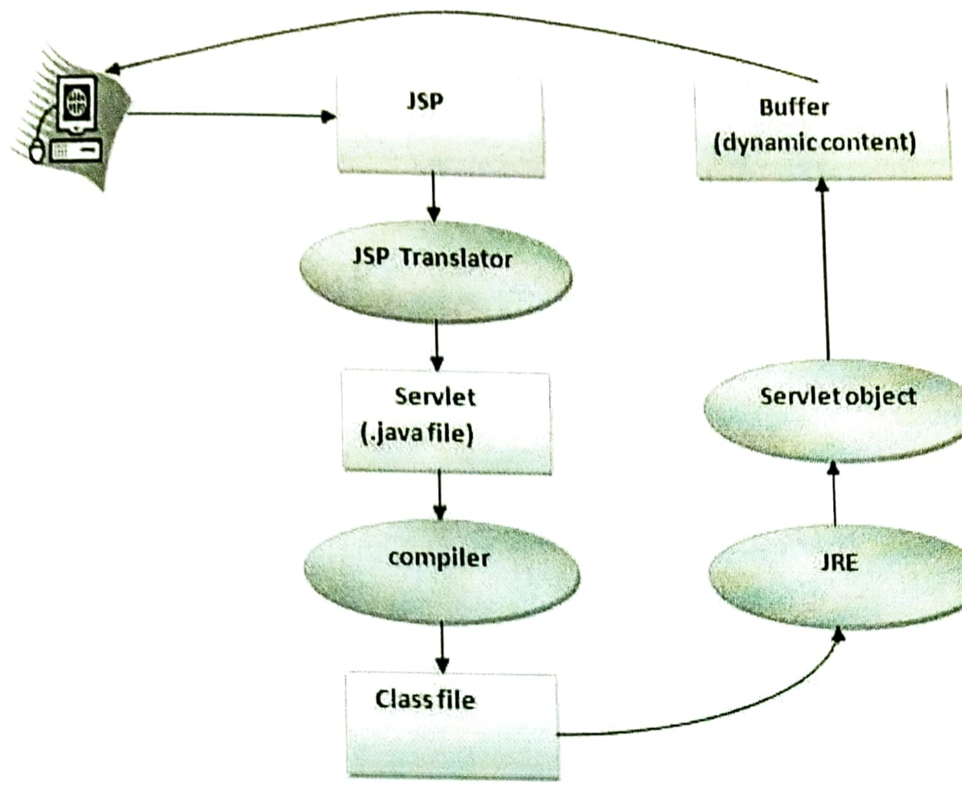
In JSP, we can use many tags such as action tags, JSTL, custom tags, etc. that reduces the code. Moreover, we can use EL, implicit objects, etc.

The Lifecycle of a JSP Page

The JSP pages follow these phases:

- Translation of JSP Page
- Compilation of JSP Page
- Classloading (the classloader loads class file)
- Instantiation (Object of the Generated Servlet is created).
- Initialization (the container invokes jspInit() method).
- Request processing (the container invokes _jspService() method).
- Destroy (the container invokes jspDestroy() method).

> Note: jspInit(), _jspService() and jspDestroy() are the life cycle methods of JSP.



As depicted in the above diagram, JSP page is translated into Servlet by the help of JSP translator. The JSP translator is a part of the web server which is responsible for translating the JSP page into Servlet. After that, Servlet page is compiled by the compiler and gets converted into the class file. Moreover, all the processes that happen in Servlet are performed on JSP later like initialization, committing response to the browser and destroy.

Creating a simple JSP Page

To create the first JSP page, write some HTML code as given below, and save it by .jsp extension. We have saved this file as index.jsp. Put it in a folder and paste the folder in the web-apps directory in apache tomcat to run the JSP page.

index.jsp

Let's see the simple example of JSP where we are using the scriptlet tag to put Java code in the JSP page. We will learn scriptlet tag later.

```
<html>
<body>
<% out.print(2*5); %>
</body>
</html>
```

How to run a simple JSP Page?

Follow the following steps to execute this JSP page:

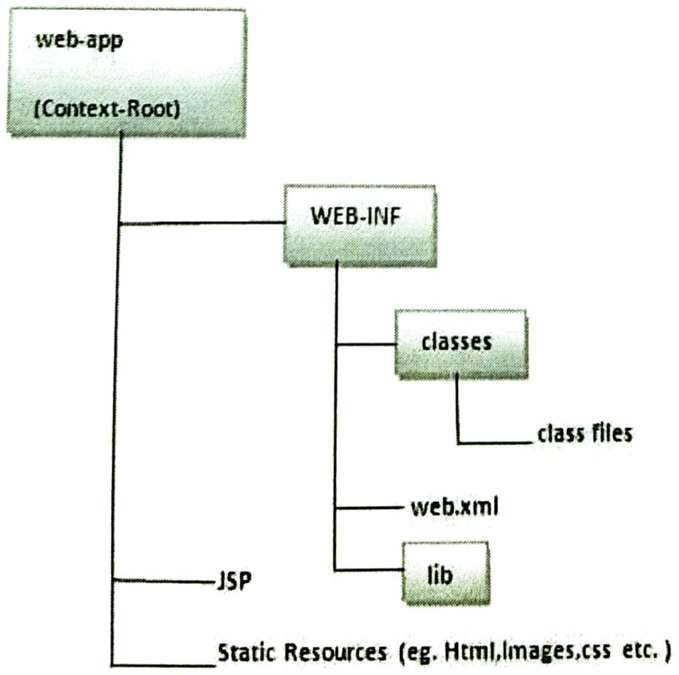
- Start the server
- Put the JSP file in a folder and deploy on the server
- Visit the browser by the URL http://localhost:portno/contextRoot/jspfile, for example, http://localhost:8888/myapplication/index.jsp

Do I need to follow the directory structure to run a simple JSP?

No, there is no need of directory structure if you don't have class files or TLD files. For example, put JSP files in a folder directly and deploy that folder. It will be running fine. However, if you are using Bean class, Servlet or TLD file, the directory structure is required.

The Directory structure of JSP

The directory structure of JSP page is same as Servlet. We contain the JSP page outside the WEB-INF folder or in any directory.



The JSP API

The JSP API consists of two packages:

1. javax.servlet.jsp
2. javax.servlet.jsp.tagext

javax.servlet.jsp package

The javax.servlet.jsp package has two interfaces and classes.The two interfaces are as follows:

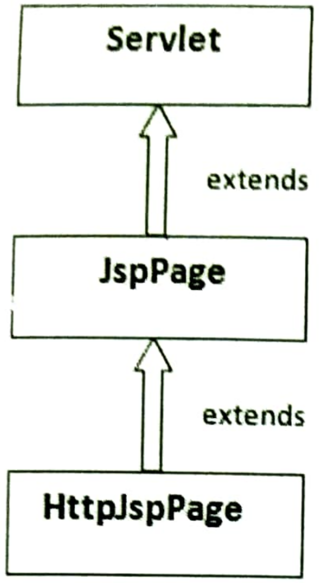
1. JspPage
2. HttpJspPage

The classes are as follows:

- JspWriter
- PageContext
- JspFactory
- JspEngineInfo
- JspException
- JspError

The JspPage interface

According to the JSP specification, all the generated servlet classes must implement the JspPage interface. It extends the Servlet interface. It provides two life cycle methods.



Methods of JspPage interface

1. **public void jspInit():** It is invoked only once during the life cycle of the JSP when JSP page is requested firstly. It is used to perform initialization. It is same as the init() method of Servlet interface.
2. **public void jspDestroy():** It is invoked only once during the life cycle of the JSP before the JSP page is destroyed. It can be used to perform some clean up operation.

The HttpJspPage interface

The HttpJspPage interface provides the one life cycle method of JSP. It extends the JspPage interface.

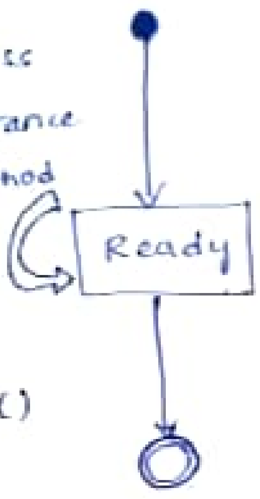
Method of HttpJspPage interface:

1. **public void _jspService():** It is invoked each time when request for the JSP page comes to the container. It is used to process the request. The underscore _ signifies that you cannot override this method.

We will learn all other classes and interfaces later.

Life Cycle of a Servlet:-

1. Load servlet class
2. Create servlet instance
3. Call the init(-) method
4. Call the service (-, -) method
5. Call the destroy() method.



* As displayed in the above diagram, there are three states of a servlet: new, ready and end.

* The servlet is in new state if servlet instance is created.

* After invoking the init() method, servlet comes in the ready state.

* In the ready state, servlet performs all the tasks. when the web container invokes the destroy() method, it shifts to the end state.

1. Servlet class is loaded

The classloader is responsible to load the Servlet class. The servlet class is loaded when the first request for the servlet is received by the web container.

If servlet is initialized, it calls the service method. Notice that servlet is initialized only once.

Syntax:-

```
public void service(ServletRequest request, Servlet
                    Response response)
    throws ServletException, IOException.
```

5, Destroy method is invoked.

The web container calls the destroy method before removing the servlet instance from the service. It gives the servlet an opportunity to clean up any resource for example memory, thread etc.

Syntax:-

```
public void destroy()
```

2, <u>Servlet instance is created</u>

The web container creates the instance of a servlet after loading the servlet class. The servlet instance is created only once in the servlet life cycle.

3, <u>init Method is invoked</u>

The web container calls the init method only once after creating the servlet instance. The init method is used to initialize the servlet. It is the life cycle method of the javax.servlet.Servlet interface.

<u>Syntax:-</u>

```
public void init(ServletConfig config) throws
        ServletException
```

4, <u>Service method is invoked</u>

The web Container calls the service method each time when request for the servlet is received. If servlet is not initialized, it follows the first three steps as described above then calls the service method.

Web Technologies
UNIT-V

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

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
   endstatement 2

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
```

```
    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"

elseputs "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}"
```

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 thefile.

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 forwriting.
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>"
```

Code No: **R1641053** **R16** **Set No. 1**

IV B.Tech I Semester Regular/Supplementary Examinations, Jan/Feb - 2022

WEB TECHNOLOGIES

(Computer Science and Engineering)

Time: 3 hours **Max. Marks: 70**

Question paper consists of Part-A and Part-B
Answer ALL sub questions from Part-A
Answer any FOUR questions from Part-B
*****

PART–A *(14 Marks)*

1. a) Define Cascading of a style sheet? [2]
   b) How does one access cookie in a java script? [2]
   c) How can you declare attributes in XML? Give an example. [2]
   d) How to create a text file in PHP? [2]
   e) What are the user defined functions in PERL? [3]
   f) How to create an array in RUBY? [3]

PART–B *(4x14 = 56 Marks)*

2. a) How can you create HTML documents with frames? Explain. [7]
   b) Create a HTML document that displays a table of basketball scores at national games in which the team names have their respective team colors. The score of the leading/winning team should appear larger and in a different font than the losing team. Use CSS. [7]

3. a) How to use Cookies and session for session tracking? Explain with an example [7]
   b) Write in brief about JSP tag extensions and libraries. [7]

4. a) What is a 'XML Parser'? Explain in detail how XML data is parsed with an example. [7]
   b) Define client side programming. Explain briefly about AJAX. [7]

5. a) Define operator. Explain different operators used in PHP. [7]
   b) Write a PHP program for a simple calculator. [7]

6. a) Discuss in brief about the types of data structures supported in Perl. [7]
   b) Write a PERL program to implement UNIX 'password' program. [7]

7. a) Describe in brief about multi dimensional arrays in Ruby. [7]
   b) Write a ruby script to display grades of a student using hashes. [7]

Code No: **R1641053** **R16** **Set No. 1**

IV B.Tech I Semester Supplementary Examinations, July/Aug - 2021

WEB TECHNOLOGIES

(Computer Science and Engineering)

Time: 3 hours **Max. Marks: 70**

Question paper consists of Part-A and Part-B
Answer ALL sub questions from Part-A
Answer any FOUR questions from Part-B
*****

PART–A *(14 Marks)*

1. a) Define the syntax of creating a list in HTML. [2]
   b) What are JSP Implicit Objects? [2]
   c) What is XML? List characteristic features of XML. [2]
   d) What is PHP? What are the common uses of PHP? [2]
   e) How are the cookies handled in PERL. [3]
   f) Write in brief about extend and include in Ruby. [3]

PART–B *(4x14 = 56 Marks)*

2. a) Discuss in brief about CSS box model. [7]
   b) Explain in brief about Conflict resolution in CSS. [7]

3. a) Explain about object, methods and events in Java Scripts. [7]
   b) With an example program, explain form validation concept in JavaScript. [7]

4. a) Collect the student’s details such as, register number, name, subject and marks using forms and generate a DTD for this XML document. Display the collected information in the descending order of marks. Write XML source code for the above. [7]
   b) Explain about various types of XML parser. [7]

5. a) How to execute a simple query in PHP? Illustrate. [7]
   b) Explain about various file operations on text files in PHP. [7]

6. a) How can you handle the files in Perl? [7]
   b) Explain in brief about how to call and identify subroutine in perl with example. [7]

7. a) Write a Ruby program that uses iterator to find out the length of a string. [7]
   b) Discuss in brief about the Rail concept in Ruby. [7]