CHAPTER

3 XML AND WEB SERVERS

Chapter Outlines

3.1 XML

XML is a software- and hardware-independent tool for storing and transporting data.

3.1.1 ORGANIZATION OF DATA IN THE FORM OF XML

XML : XML stands for eXtensible Markup Language.

XML is a markup language much like HTML

XML was designed to store and transport data

XML was designed to be self-descriptive

XML is a W3C Recommendation

Example for an XML File :

```
<note>
<to>Tove</to>
<from>Jani</from>
<heading>Reminder</heading>
<body>Don't forget me this weekend!</body>
</note>
```

The above is an XML file which is quite self-descriptive. It has sender and receiver information. It also has a heading and a message body. But still, this XML document does not DO anything. XML is just information wrapped in tags. Someone must write a piece of software to send, receive, store, (or) display it.

Difference Between XML and HTML are : XML and HTML were designed with different goals :

XML	HTML
XML was designed to carry data - and to focus on what data is	HTML was designed to display data and to focus on how data looks.
XMl is case sensitive	HTML is case insensitive

XML tags are not predefined	HTML tags are predefined
XML can separate data from HTML. With XML, your data is stored outside your HTML.	When HTML is used to display data, the data is stored inside your HTML.
XML is content aware	HTML is content unaware.
Does a syntax check	Does not check syntax.

Example :

```
<? xml version="1.0" encoding="UTF-8"?>
<note>
  <to>Tove</to>
  <from>Jani</from>
  <heading>Reminder</heading>
  <body>Don't forget me this weekend!</body>
</note>
```

Explanation : In the above example the line <? xml version="1.0" encoding="UTF-8"?> is called as XML prolog. The XML prolog is optional. If it exists, it must come first in the document.

Root : XML documents must contain one root element that is the parent of all other elements. in the above example the tag <note> is root element.

In XML, it is illegal to omit the closing tag. All elements must have a closing tag.

For example :

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

The above two are illegal. They must have closing tags.

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

XML tags are case sensitive. The tag <Letter> is different from the tag <letter>.

In XML, all elements must be properly nested within each other :

```
<b><i>This text is bold and italic</i></b>
```

Properly nested" simply means that since the <i> element is opened inside the <b> element, it must be closed inside the <b> element.

In XML, the attribute values must always be quoted.

INCORRECT :

```
<note date=12/11/2007>
  <to>Tove</to>
  <from>Jani</from>
</note>
```

CORRECT :

```
<note date="12/11/2007">
  <to>Tove</to>
  <from>Jani</from>
</note>
```

Entity references are special characters which have special meaning in XML. The following are some of the special characters in XML.

<	< less than
>	< Greater than
&	& ampersand
'	'apostrophe
"	"quotation mark

The syntax for writing comments in XML is similar to that of HTML.

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

Two dashes in the middle of a comment are not allowed.

Not allowed :

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

3.1.2 SIGNIFICANCE OF NAMESPACE

XML Namespaces provide a method to avoid element name conflicts.

The namespace can be defined by an xmlns attribute in the start tag of an element.

syntax. xmlns:prefix="URI".

Example :

```
<root>
<h:table xmlns:h="http://www.w3.org/TR/html4/">
  <h:tr>
    <h:td>Apples</h:td>
    <h:td>Bananas</h:td>
  </h:tr>
</h:table>
<f:table xmlns:f="http://www.w3schools.com/furniture">
  <f:name>African Coffee Table</f:name>
  <f:width>80</f:width>
  <f:length>120</f:length>
</f:table>
</root>
```

(or) Namespaces can also be declared in the XML root element :

```
<root
xmlns:h="http://www.w3.org/TR/html4/"
```

```
xmlns:f="http://www.w3schools.com/furniture">
<h:table>
 <h:tr>
  <h:td>Apples</h:td>
  <h:td>Bananas</h:td>
 </h:tr>
</h:table>
<f:table>
 <f:name>African Coffee Table</f:name>
 <f:width>80</f:width>
 <f:length>120</f:length>
</f:table>
</root>
```

Displaying XML : Most browsers will display an XML document with color-coded elements. for example as given below.

```
<?xml version="1.0" encoding="UTF-8"?>
- <note>
    <to>Tove</to>
    <from>Jani</from>
    <heading>Reminder</heading>
    <body>Don't forget me this weekend!</body>
</note>
```

To display output of an XML file. CSS file should be used. For example as given below.

```
<?xml version="1.0" encoding="UTF-8"?>
<?xml-stylesheet type="text/css" href="cd_catalog.css"?>
<CATALOG>
<CD>
 <TITLE>Empire Burlesque</TITLE>
 <ARTIST>Bob Dylan</ARTIST>
 <COUNTRY>USA</COUNTRY>
 <COMPANY>Columbia</COMPANY>
```



```
<PRICE>10.90</PRICE>
<YEAR>1985</YEAR>
</CD>
<CD>
<TITLE>Hide your heart</TITLE>
<ARTIST>Bonnie Tyler</ARTIST>
<COUNTRY>UK</COUNTRY>
<COMPANY>CBS Records</COMPANY>
<PRICE>9.90</PRICE>
<YEAR>1988</YEAR>
</CD>
The  css file is cd_catalog.css
CATALOG {
   background-color: #ffffff;
   width: 100%;
}
CD {
   display: block;
   margin-bottom: 30pt;
   margin-left: 0;
}
TITLE {
   display: block;
   color: #ff0000;
   font-size: 20pt;
}
ARTIST {
   display: block;
   color: #0000ff;
   font-size: 20pt;
}
COUNTRY, PRICE, YEAR, COMPANY {
```

```
    display: block;
    color: #000000;
    margin-left: 20pt;
}
```

3.1.3 COMPARE AND CONTRAST DTD AND SCHEME

The purpose of a DTD is to define the structure of an XML document. It defines the structure with a list of legal elements. the example below illustrates DTD.

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE note
[
<!ELEMENT note (to,from,heading,body)>
<!ELEMENT to (#PCDATA)>
<!ELEMENT from (#PCDATA)>
<!ELEMENT heading (#PCDATA)>
<!ELEMENT body (#PCDATA)>
]>
<note>
<to>Tove</to>
<from>Jani</from>
<heading>Reminder</heading>
<body>Don't forget me this weekend!</body>
</note>
```

The DTD above is interpreted like this:

" !DOCTYPE note defines that the root element of the document is note"

" !ELEMENT note defines that the note element must contain the elements: "to, from, heading, body"

" !ELEMENT to defines the to element to be of type "#PCDATA"

" !ELEMENT from defines the from element to be of type "#PCDATA"

" !ELEMENT heading defines the heading element to be of type "#PCDATA"

" !ELEMENT body defines the body element to be of type "#PCDATA"

Note : #PCDATA means parse-able text data.

3.1.4 PARSING PROCESS OF XML BY DOM AND SAX

In computer technology, a parser is a program, usually part of a compiler, that receives input in the form of sequential source program instructions, interactive online commands, markup tags, or some other defined interface and breaks them up into parts (for example, the nouns (objects), verbs (methods), and their attributes or options) that can then be managed by other programming (for example, other components in a compiler).

A parser may also check to see that all input has been provided that is necessary.

DOM Parser : DOM parser parses the entire XML document and loads it into memory, then models it in a ***"TREE"*** structure for easy traversal (or) manipulation.

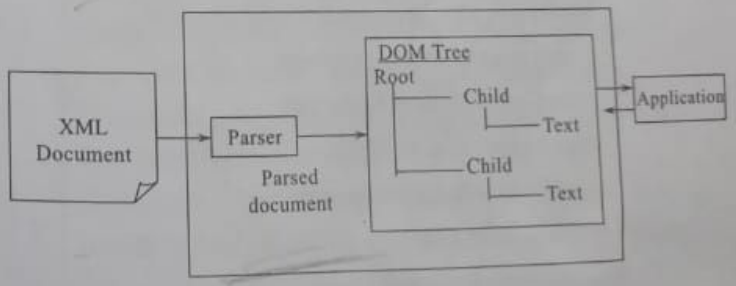

FIG 3.1 :

1. **DOM XML Parser Example :** This example shows you how to get the node by ***"name"***, and display the value.

```
/Users/mkyong/staff.xml
Markup
<?xml version="1.0"?>
<company>
    <staff id="1001">
            <firstname>yong</firstname>
            <lastname>mook kim</lastname>
            <nickname>mkyong</nickname>
            <salary>100000</salary>
</staff>
    <staff id="2001">
            <firstname>low</firstname>
            <lastname>yin fong</lastname>
            <nickname>fong fong</nickname>
            <salary>200000</salary>
    </staff>
</company>
ReadXMLFile.java
Java
package com.mkyong.seo;
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;
import java.io.File;
public class ReadXMLFile {
 public static void main(String argv[]) {
  try {
File fXmlFile = new File("/Users/mkyong/staff.xml");
```



```
DocumentBuilderFactory dbFactory = Document Builder
                                Factory.newInstance();
DocumentBuilder dBuilder = dbFactory.newDocument
                                        Builder();
Document doc = dBuilder.parse(fXmlFile);
//optional, but recommended
//read this - http://stackoverflow.com/questions/13786607/
normalization-in-dom-parsing-with-java-how-does-it-work
doc.getDocumentElement().normalize();
System.out.println("Root element :" + doc.get Document
Element().getNodeName());
NodeList nList = doc.getElementsByTagName("staff");
System.out.println("----------------------------");
for (int temp = 0; temp < nList.getLength(); temp++) {
Node nNode = nList.item(temp);
System.out.println("\nCurrent Element :" + nNode.get
NodeName());
if (nNode.getNodeType() == Node.ELEMENT_NODE) {
Element eElement = (Element) nNode;
System.out.println("Staff id : " + eElement.getAttribute("id"));
System.out.println("First Name : " + eElement.getElementsBy
TagName("firstname").item(0).getTextContent());
System.out.println("Last Name : " + eElement.getElementsBy
TagName("lastname").item(0).getTextContent());
System.out.println("Nick Name : " + eElement.getElementsBy
TagName("nickname").item(0).getTextContent());
System.out.println("Salary : " + eElement.getElementsByTag
Name("salary").item(0).getTextContent());
            }
   }
  } catch (Exception e) {
   e.printStackTrace();
```

```
    }
  }
}
Result
Root element :company
----------------------------
Current Element : staff
Staff id : 1001
First Name : yong
Last Name : mook kim
Nick Name : mkyong
Salary : 100000
Current Element :staff
Staff id : 2001
First Name : low
Last Name : yin fong
Nick Name : fong fong
Salary : 200000
```

Advantages :

- XML DOM is language and platform independent.
- XML DOM is travesible-Information in XML DOM is organized in a hierarchy which allows developer to navigate around the hierarchy looking for specific information.
- XML DOM is modifiable - It is dynamic in nature providing developer a scope to add, edit, move or remove nodes at any point on the tree.

Disadvantages :

- It consumes more memory (if the XML structure is large) as program written once remains in memory all the time until and unless removed explicitly.

- Due to the larger usage of memory its operational speed, compared to SAX is slower.
- <?xml version="1.0"?>
- <Company>
- <Employee category="technical">
- <FirstName>Tanmay</FirstName>
- <LastName>Patil</LastName>
- <ContactNo>1234567890</ContactNo>
- </Employee>

The following XML document node.xml.

```
<Employee category= "non-technical">
  <FirstName>Taniya</FirstName>
  <LastName>Mishra</LastName>
  <ContactNo>1234667898</ContactNo>
</Employee>
</Company>
```

The document object model of the above XML document would be as follows,

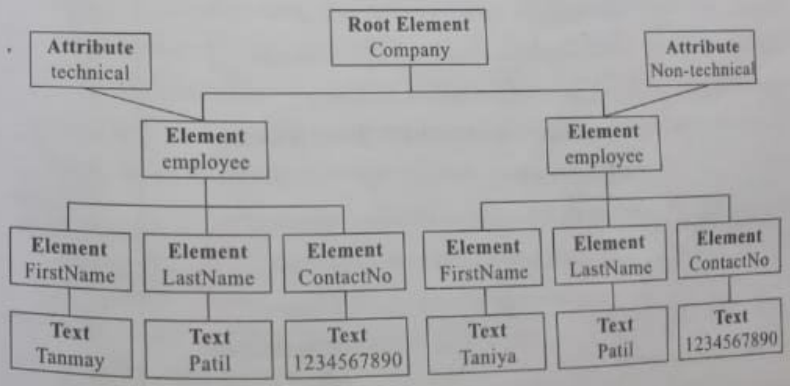

FIG 3.2 :

From the above diagram we can interface,

- Node object can have only one parent node object. This occupies the position above all the nodes. Here it is Company.
- The parent node can have multiple nodes called as child nodes. Thesechild nodes can have additional node called as attribute node. In the above example we have two attribute nodes Technical and Non-Technical. The attribute node is not actually a child of the element node, but is still associated with it.
- These child nodes in turn can have multiple child nodes. The text within the nodes is called as text node.
- The node objects at the same level are called as siblings.
- The DOM Identifies,
- The objects to represent the interface and manipulate the document.
- The relationship among the objects and interfaces.

Following example demonstrate simple XML document whose node tree is structure is shown in the diagram below,

```
<?xml version="1.0"?>
<Company>
  <Employee category="Technical">
    <FirstName>Tanmay</FirstName>
    <LastName>Patil</LastName>
    <ContactNo>1234567890</ContactNo>
  </Employee>
  <Employee category="Non-Technical">
    <FirstName>Taniya</FirstName>
    <LastName>Mishra</LastName>
    <ContactNo>1234667898</ContactNo>
  </Employee>
</Company>
```


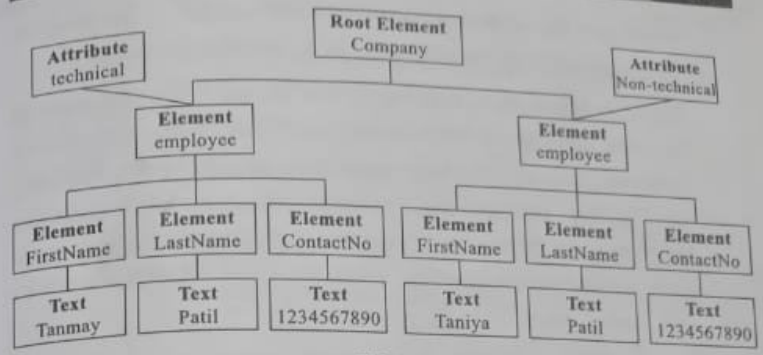

FIG 3.3 :

- The topmost node of a tree is called the root. The root node is <Company> which in turn contains the two nodes of <Employee>. These nodes are referred to as child nodes.
- The child node <Employee> of root node <Company>, in turn consists of its own child node (<FirstName>, <LastName>, <ContactNo>).
- The two child nodes, <Employee> have attribute values Technical and Non-Technical, are referred as attribute nodes.
- The text within the every node is called as text node.

DOM as an API contains interfaces that represent different types of information that can be found in an XML document, such as elements and text. These interfaces include the methods and properties necessary to work with these objects. Properties define the characteristic of the node whereas methods give the way to manipulate the nodes.

We will study about the XML Loading and Parsing. XML DOM is language and platform independent.

In order to describe the interfaces provided by the API, the W3C uses an abstract language called the Interface Definition Language (IDL). The advantage of using IDL is that the

developer learns how to use the DOM with his or her favorite language and can switch easily to a different language.

The disadvantage is that, since it is abstract, the IDL cannot be used directly by Web developers. Due to the differences between programming languages, they need to have a mapping (or) binding - between the abstract interfaces and their concrete languages. DOM has been mapped to programming languages such as Javascript, JScript, Java, C, C++, PLSQL, Python, and Perl.

Parser : A parser is a software application that is designed to analyze a document, in our case XML document and do something specific with the information.

Following example demonstrates how to load XML (node.xml) data using Ajax and Javascript when XML content is received as XML file. Here Ajax function, gets the content of an xml file and stores it in XML DOM. Once the DOM object is create it is then parsed.

```
<!DOCTYPE html>
<html>
  <body>
    <div>
      <b>FirstName:</b> <span id="FirstName"></
span><br>
      <b>LastName:</b> <span id="LastName"></
span><br>
      <b>ContactNo:</b> <span id="ContactNo"></
span><br>
      <b>Email:</b> <span id="Email"></span>
    </div>
    <script>
    //if browser supports XMLHttpRequest
```



```
    if (window.XMLHttpRequest)
    {// Create an instance of XMLHttpRequest object. code
    for IE7+, Firefox, Chrome, Opera, Safari
      xmlhttp = new XMLHttpRequest();
    }
    else
    {// code for IE6, IE5
     xmlhttp = new ActiveXObject("Microsoft.XMLHTTP");
    }
    // sets and sends the request for calling "node.xml"
     xmlhttp.open("GET","/dom/node.xml",false);
     xmlhttp.send();
    // sets and returns the content as XML DOM
     xmlDoc=xmlhttp.responseXML;
    //parsing the DOM object
     document.getElementById("FirstName").innerHTML=
    xmlDoc.getElementsByTagName("FirstName")[0].childNodes
[0].nodeValue;
     document.getElementById("LastName").innerHTML=
    xmlDoc.getElementsByTagName("LastName")[0].childNodes
[0].nodeValue;
     document.getElementById("ContactNo").innerHTML=
    xmlDoc.getElementsByTagName("ContactNo")[0].childNodes
[0].nodeValue;
     document.getElementById("Email").innerHTML=
    xmlDoc.getElementsByTagName("Email")[0].childNodes
[0].nodeValue;
    </script>
  </body>
</html>
```

Most of the details of the code are in the script code :

- Internet Explorer uses the ActiveXObject ("Microsoft. XMLHTTP") to create an instance of XMLHttpRequest object, other browsers use theXMLHttpRequest() method.
- The responseXML transforms the XML content directly in XML DOM.
- Once the XML content is transformed into JavaScript XML DOM, you can access any XML element by using JS DOM methods and properties. We have used DOM properties such as childNodes, nodeValue and DOM methods such as getElementsById(ID), getElementsBy TagName(tags_name).

XML DOM-Traversing : Traversing is a process in which looping is done in a systematic manner by going across each an every element step by step in a node tree.

The following example (traverse_example.htm) demonstrates the DOM traversing. Here we traverse through each child node of <Employee> element.

```
!DOCTYPE html>
<html>
<style>
table,th,td
{
border:1px solid black;
border-collapse:collapse
}
</style>
<body>
<div id="ajax_xml">
</div>
<script>
```



```
//if browser supports XMLHttpRequest
if (window.XMLHttpRequest)
{// Create an instance of XMLHttpRequest object. code
for IE7+, Firefox, Chrome, Opera, Safari
var xmlhttp = new XMLHttpRequest();
}
else
{// code for IE6, IE5
var xmlhttp = new ActiveXObject("Microsoft.XMLHTTP");
}
// sets and sends the request for calling "node.xml"
xmlhttp.open("GET","/dom/node.xml",false);
xmlhttp.send();
// sets and returns the content as XML DOM
var xml_dom=xmlhttp.responseXML;
// this variable stores the code of the html table
var html_tab = '<table id="id_tabel" align="center"><tr>
<th>EmployeeCategory</th><th>FirstName</th><th>
LastName</th><th>ContactNo</th><th>Email</th></tr>';
var arr_employees = xml_dom.getElementsByTagName
("Employee");
// traverses the "arr_employees" array
for(var i=0; i<arr_employees.length; i++) {
var employee_cat = arr_employees[i].getAttribute('category');
// gets the value of 'category' element of current "Element"
tag
// gets the value of first child-node of 'FirstName' element
of current "Employee" tag
var employee_firstName = arr_employees[i].getElements
ByTagName('FirstName')[0].childNodes[0].nodeValue;
// gets the value of first child-node of 'LastName' element
of current "Employee" tag
var employee_lastName =
```

```
arr_employees[i].getElementsBy
TagName('LastName')[0].childNodes[0].nodeValue;
    // gets the value of first child-node of 'ContactNo' element
of current "Employee" tag
    var employee_contactno =
arr_employees[i].getElementsBy
TagName('ContactNo')[0].childNodes[0].nodeValue;
    // gets the value of first child-node of 'Email' element of
current "Employee" tag
    var employee_email = arr_employees[i].getElementsBy
TagName('Email')[0].childNodes[0].nodeValue;
    // adds the values in the html table
    html_tab += '<tr><td>'+ employee_cat+ '</td><td>'+
employee_firstName+ '</td><td>'+ employee_lastName+ '</
td><td>'+ employee_contactno+ '</td><td>'+ employee
_email+ '</td></tr>';
    }
    html_tab += '</table>';
    // adds the html table in a html tag, with id="ajax_xml"
    document.getElementById('ajax_xml').innerHTML =
html_tab;
    </script>
  </body>
</html>
```

- This code loads node.xml.
- Next, the XML content is transformed into JavaScript XML DOM object.
- Next, array of elements (with tag Element) using the method getElementsByTagName() is obtained.
- Next, we traverse through this array and display the child node values in a table.

Execution : Save this file as traverse_example.html on the server path (this file and node.xml should be on the same path in your server).

XML DOM - Navigation

The XML DOM consist of various properties of the nodes which help us navigate through the nodes, such a,

- parentNode
- childNodes
- firstChild
- lastChild
- nextSibling
- previousSibling.

SAX : SAX (Simple API for XML) is an application program interface (API) that allows a programmer to interpret a Web file that uses the Extensible Markup Language (XML) - that is, a Web file that describes a collection of data. SAX is an alternative to using the Document Object Model (DOM) to interpret the XML file. As its name suggests, it's a simpler interface than DOM and is appropriate where many or very large files are to be processed, but it contains fewer capabilities for manipulating the data content.

What is a SAX parser ? SAX (Simple API for XML) is an event-driven online algorithm for parsing XML documents, with an API interface developed by the XML-DEV mailing list. SAX provides a mechanism for reading data from an XML document that is an alternative to that provided by the Document Object Model (DOM).

From the program, if the xml file has to be read the developer has to write his own code to parse(read) the xml file. It is complicated. To avoid this Java programming language has given parsers like DOM and SAX to read the xml file. XML parser consumes lot of time and complicated. DOM/SAX parser is used to parse the xml file. Using this code can be developed fast and accurate.

DOM read XML and stores in tree format. SAX reads xml file in event based format.

3.1.5 APPLICATIONS OF XML

1. Configuration of files.
2. Media for data interchange.
3. B2B transctions on the web.

3.2 WEB SERVERS

3.2.1 CLIENT-SIDE AND SERVER-SIDE SCRIPTING

Client-Side Environment : The client-side environment used to run scripts is usually a browser. The processing takes place on the end users computer. The source code is transferred from the web server to the users computer over the internet and run directly in the browser.

The scripting language needs to be enabled on the client computer. Sometimes if a user is conscious of security risks they may switch the scripting facility off. When this is the case a message usually pops up to alert the user when script is attempting to run.

Server-Side Environment : The server-side environment that runs a scripting language is a web server. A user's request is fulfilled by running a script directly on the web server to generate dynamic HTML pages. This HTML is then sent to the client browser. It is usually used to provide interactive web sites that interface to databases or other data stores on the server.

This is different from client-side scripting where scripts are run by the viewing web browser, usually in JavaScript. The primary advantage to server-side scripting is the ability to highly customize the response based on the user's requirements, access rights, or queries into data stores.

3.2.2 ARCHITECTURE OF WEB SERVER

Web server is a computer where the web content is stored. Basically web server is used to host the web sites but there exists other web servers also such as gaming, storage, FTP, email etc.

Web site is collection of web pages whileweb server is a software that respond to the request for web resources.

Web Server Architecture follows the following two approaches,

1. Concurrent approach.
2. Single-Process-Event-Driven Approach.

1. **Concurrent Approach :** Concurrent approach allows the web server to handle multiple client requests at the same time. It can be achieved by following methods,

   (i) Multi-process. (ii) Multi-threaded.

   (iii) Hybrid method.

   (i) Multi-Processing : In this a single process (parent process) initiates several single-threaded child processes and distribute incoming requests to these child processes. Each of the child processes are responsible for handling single request.

   It is the responsibility of parent process to monitor the load and decide if processes should be killed or forked.

   (ii) Multi-Threaded : Unlike Multi-process, it creates multiple single-threaded process.

   (iii) Hybrid : It is combination of above two approaches. In this approach multiple process are created and each process initiates multiple threads. Each of the threads handles one connection. Using multiple threads in single process results in less load on system resources.

Leading web servers available today,

1. Apache HTTP Server.
2. Internet Information Services (IIS).
3. Light tpd.
4. Sun Java System Web Server.
5. Jig saw Server.

3.2.3 VARIOUS HTTP REQUEST TYPES AND THEIR DIFFERENCE

S.No	Method and Description	
1.	GET	The GET method is used to retrieve information from the given server using a given URI. Requests using GET should only retrieve data and should have no other effect on the data.
2.	HEAD	Same as GET, but transfers the status line and header section only.
3.	POST	A POST request is used to send data to the server, for example, customer information, file upload, etc., using HTML forms.
4.	PUT	Replaces all current representations of the target resource with the uploaded content.
5.	DELETE	Removes all current representations of the target resource given by a URI.
6.	CONNECT	Establishes a tunnel to the server identified by a given URI.
7.	OPTIONS	Describes the communication options for the target resource.
8.	TRACE	Performs a message loop-back test along the path to the target resource.

HTTP Request Types :

- Also known as request methods.
- Most popular are get and post.
  - Retrieve and send client form data to Web server
  - Get request.

- Sends form content as part of URL.
- Retrieves appropriate resource from Web server.
- Limits query to 1024 characters.
  - post request
- Updates contents of Web server (Posting new messages to forum).
- Has no limit for length of query.
- Not part of URL and cannot be seen by use.
- Posts data to server-side form handler.
- Browsers cache (save on disk) Web pages.
  - Allows for quick reloading.
  - Cache responses to get request.
  - Do not cache responses to post request.

3.2.4 INSTALLATION PROCESS OF IIS PWS AND APACHE WEB SERVERS

1. To open the Windows Features dialog box, click Start and then click Control Panel.
2. In the Control Panel, click Programs.
3. Click Turn Windows features on (or) off.
4. You may receive the Windows Security warning. Click Allow to continue. The Windows Features dialog box is displayed.
5. Expand Internet Information Services. Additional categories of IIS features are displayed. Select Internet Information Services to choose the default features for installation.
6. Expand the additional categories displayed and select any additional features you want to install, such as Web Management Tools.

7. If you are installing IIS for evaluation purposes, you may want to select additional features to install. Select the check boxes for all IIS features you want to install and then click OK to start installation.
8. The progress indicator appears.
9. When the installation completes, the Windows Features dialog box closes and the Control Panel is displayed.
10. IIS is now installed with a default configuration on Windows Vista or Windows 7. To confirm that the installation succeeded, type the following URL into your browser, http://localhost.

FIG 3.4 : Default Web Site

11. Next, you can use Internet Information Services Manager to manage and configure IIS. To open IIS Manager, click Start, type inetmgr in the Search Programs and Files box, and then press ENTER.

Installation of PWS (Perosnal Web Server) :

1. Insert your Windows 98 CD-ROM in your CD-ROM drive.
2. Click Start and then click Run.
3. In the Open box, type the following path to the Setup.exe file, where x is the letter of your CD.

   x:\add-ons\pws\setup.exeROM drive:
4. Click OK.
5. Follow the instructions in Personal Web Server Setup.

Installation of Apache :

Step-1 : Configure IIS, Skype and other software (optional).

If you have a Professional or Server version of Windows, you may already have IIS installed. If you would prefer Apache, either remove IIS as a Windows component or disable its services.

Apache listens for requests on TCP/IP port 80. The default installation of Skype also listens on this port and will cause conflicts. To switch it off, start Skype and choose Tools > Options > Advanced > Connection. Ensure you untick "*Use port 80 and 443 as alternatives for incoming connections*".

Step-2 : Download the files.

We are going to use the unofficial Windows binary from Apache Lounge. This version has performance and stability improvements over the official Apache distribution, although I am yet to notice a significant difference. However, it is provided as a manually installable ZIP file from www.apachelounge.com/download/

You should also download and install the Windows C++ runtime from Microsoft.com. You may have this installed already, but there is no harm installing it again.

As always, remember to virus scan all downloads.

Step-3 : Extract the files.

We will install Apache in C:Apache2, so extract the ZIP file to the root of the C: drive.

Apache can be installed anywhere on your system, but you will need to change the configuration file paths accordingly.

Step-4 : Configure Apache.

Apache is configured with the text file confhttpd.conf contained in the Apache folder. Open it with your favourite text editor.

Note that all file path settings use a '/' forward-slash rather than the Windows backslash. If you installed Apache anywhere other than C:Apache2, now is a good time to search and replace all references to "c:/Apache2".

There are several lines you should change for your production environment,

Line 46, listen to all requests on port 80 :

Listen * : 80

Line 116, enable mod-rewrite by removing the # (optional, but useful) :

LoadModule rewrite_module modules/mod_rewrite.so

Line 172, specify the server domain name:

ServerName localhost : 80

Line 224, allow.htaccess overrides :

AllowOverride All

Step-5 : Change the web page root (optional)

By default, Apache return files found in its htdocs folder. I would recommend using a folder on an another drive or partition to make backups and re-installation easier. For the purposes of this example, we will create a folder called D:WebPages and change httpd.conf accordingly.

Line 179, set the root.

DocumentRoot "D:/WebPages"

and line 204:

<Directory "D:/WebPages">

Step 6 : test your installation

Your Apache configuration can now be tested. Open a command box (Start > Run > cmd) and enter:

cd Apache2bin

httpd -t

Correct any httpd.conf configuration errors and retest until none appear.

Step-7 : Install Apache as a Windows service.

The easiest way to start Apache is to add it as a Windows service. From a command prompt, enter:

cd Apache2bin

httpd-k install

Open the Control Panel, Administrative Tools, then Services and double-click Apache 2.2. Set the Startup type to "Automatic" to ensure Apache starts every time you boot your PC.

Alternatively, set the Startup type to "Manual" and launch Apache whenever you choose using the command "net start Apache 2.2".

Step-8 : Test the web server

Create a file named index.html in Apache's web page root (either htdocs or D:WebPages) and add a little HTML code:

<html>

<head><title>testing Apache</title></head>

<body> Apache is working! </body>

</html>

Ensure Apache has started successfully, open a web browser and enter the address http://localhost/. If all goes well, your test page should appear.

3.2.5 COMPARE/CONTRAST IIS, PWS AND APACHE

	IIS	PWS	Apache
COMPANY	Microsoft corporation	Microsoft corporation	Apache foundation corporation
Version	5.0	4.0	1.3.20
Platforms	Windows 2000	Windows 95/98.	Unix, windows NT/2000,supports windows 95/08.
Brief	Most popular web server for windows 2000.	A basic webserver for publishing personal webpages.	Currently most popular webserver.
Price	Included with windows 2000.	Free ware.	Free ware.

CHAPTER 4

JAVA SCRIPTS

Chapter Outlines

4.0 INTRODUCTION

JavaScript is a client-side scripting language. A scripting language is a light weight programming language.

JavaScript is one of the most used languages when it comes to building web applications as it allows developers to wrap HTML and CSS code in it to make web apps interactive.

JavaScript is a light weight, cross-platform, single-threaded, and interpreted compiled programming language which is also known as the scripting language for web pages.

JavaScript can enhance the dynamics and interactive features of your page by allowing you to perform calculations, check forms, write interactive games, add special effects, customize graphics selections, create security passwords and more.

4.1 NEED FOR CLIENT SIDE SCRIPTING

Client-side scripting is when the server sends the code along with the HTML web page to the client. The script is referred to by the code.

This code will be processed on the client machine and the HTML page will NOT perform a Post Back to the web-server.

Traditionally, client-side scripting is used for page navigation, data validation and formatting.

The language used in this scripting is JavaScript. JavaScript is compatible and is able to run on any internet browser.

The two main benefits of client-side scripting are :

1. The user's actions will result in an immediate response because they don't require a trip to the server.
2. Fewer resources are used and needed on the web-server.

4.2 VARIOUS CLIENT SIDE SCRIPTING LANGUAGES

1. **JavaScript :** It is the most commonly used client-side scripting or programming language. It is written in the ECMAScript programming language.

   JavaScript is a dynamically typed (also known as weakly typed) object-oriented scripting language. It uses an integrated interpreter to run directly in the browser.

   Weakly typed indicates that the variables can be implicitly transformed from one data type to another.

2. **VBScript :** This scripting language was developed by Microsoft, based on Visual Basic. It was mostly used to improve the functionality of web pages in Internet Explorer. The Internet Explorer web browser interprets VBScript. No one really uses it now.

3. **JQuery :** jQuery is a JavaScript library that is fast, tiny, and lightweight. It's used to turn a lot of JavaScript code into user-friendly functionality.

   The jQuery language is used by the majority of the world's largest firms, including Google, Microsoft, IBM, Netflix, and others.

4. **Tcl/Tk :** (pronounced "*tickle*"; the full name is Tool Command Language/Tool Kit) is a scripting language used in applications such as automated testing.

   Tcl/Tk is less well known, but possibly more widely used than Python, Perl and PHP. It can be used to create cross-platform applications, is easily deployed and very extensible. Experienced programmers can pick it up and become productive very quickly.

4.3 VARIOUS OPERATORS

JavaScript supports the following types of operators :

- Arithmetic operators.
- Comparison operators.
- Logical (or) Relational operators.
- Assignment operators.
- Conditional (or) ternary operators.

Arithmetic Operators : Assume variable a holds 10 and variable B holds 20, then,

S.No.	Operator and Description
1.	**+ (Addition)** Adds two operands **Ex :** A + B will give 30
2.	**– (Subtraction)** Subtracts the second operand from the first **Ex :** A - B will give -10
3.	*** (Multiplication)** Multiply both operands **Ex :** A * B will give 200
4.	**/ (Division)** Divide the numerator by the denominator **Ex :** B / A will give 2
5.	**% (Modulus)** Outputs the remainder of an integer division **Ex :** B % A will give 0
6.	**++ (Increment)** Increases an integer value by one **Ex :** A++ will give 11
7.	**-- (Decrement)** Decreases an integer value by one **Ex :** A-- will give 9

Program on arithmetic operators,

```
<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>
Set the variables to different values and then try...
</body>
</html>
```

Comparison Operators : Assume variable A holds 10 and variable B holds 20, then.

S.No.	Operator and 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.

Logical Operators : Assume variable A holds 10 and variable B holds 20, then

S.No.	Operator and 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.

Bitwise Operators ; Assume variable A holds 2 and variable B holds 3, then –

S.No.	Operator and 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. **Ex :** (A ^ B) is 1.

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 number of places specified in the second operand. New bits are filled with zeros. Shifting a value left by one position is equivalent to multiplying 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 number 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.

Assignment Operators : JavaScript supports the following assignment operators.

S.No.	Operator and 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 and assigns the result to the left operand. **Ex :** C += A is equivalent to C = C + A
3.	**– = (Subtract and Assignment)** It subtracts the right operand from the left operand and assigns the result to the left operand. **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 operand 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 assigns the result to the left operand. **Ex :** C %= A is equivalent to C = C % A

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 and Description
1.	**(Conditional)** If Condition is true? Then value X : Otherwise value Y

Program :

```
<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>
     <p>Set the variables to different values and different operators and then try...</p>
    </body>
</html>
```

4.4 IF, IF/ELSE AND SWITCH CONDITIONAL STATEMENTS

If Statement : Use the if statement to specify a block of JavaScript code to be executed if a condition is true.

Syntax :

```
if (condition) {
   block of code to be executed if the condition is true
}
```

```
<html>
<body>
<p>Display "Good day!" if the hour is less than 18:00:</p>
<p id="demo">Good Evening!</p>
<script>
if (new Date().getHours() < 18) {
document.getElementById("demo").innerHTML = "Good day!";
}
</script>
</body>
</html>
```

The else Statement : Use the else statement to specify a block of code to be executed if the condition is false.

```
if (condition) {
   block of code to be executed if the condition is true
} else {
   block of code to be executed if the condition is false
}
```


else if Statement :

Use the else if statement to specify a new condition if the first condition is false.

```
if (condition1) {
   block of code to be executed if condition1 is true
} else if (condition2) {
   block of code to be executed if the condition1 is false and condition2 is true
} else {
   block of code to be executed if the condition1 is false and condition2 is false
}
program:
<html>
<body>
<p>Click the button to get a time-based greeting:</p>
<button onclick="myFunction()">Try it</button>
<p id="demo"></p>
<script>
function myFunction() {
   var greeting;
   var time = new Date().getHours();
   if (time < 10) {
      greeting = "Good morning";
   } else if (time < 20) {
      greeting = "Good day";
   } else {
      greeting = "Good evening";
   }
document.getElementById("demo").innerHTML = greeting;
}
</script>
</body>
</html>
```

4.5 WHILE, DO/WHILE AND FOR ITERATIVE STATEMENTS

JavaScript supports different kinds of loops :

- **for** - loops through a block of code a number of times.
- **for/in** - loops through the properties of an object.
- **while** - loops through a block of code while a specified condition is true.
- **do/while** - also loops through a block of code while a specified condition is true.

1. **While Loop :** The while loop loops through a block of code as long as a specified condition is true.

Syntax :

```
while (condition) {
    code block to be executed
}
```

```
<html>
<body>
<p>Click the button to loop through a block of code as long as i is less than 10.</p>
<button onclick="myFunction()">Try it</button>
<p id="demo"></p>
<script>
function myFunction() {
  var text = "";
  var i = 0;
  while (i < 10) {
    text += "<br>The number is " + i;
    i++;
  }
  document.getElementById("demo").innerHTML = text;
}
</script>
</body>
</html>
```


2. **Do/While loop :** This loop will execute the code block once, before checking if the condition is true, then it will repeat the loop as long as the condition is true.

```
do {
    code block to be executed
}while (condition);
```

```
<html>
<body>
<p>Click the button to loop through a block of code as long as i is less than 10.</p>
<button onclick="myFunction()">Try it</button>
<p id="demo"></p>
<script>
function myFunction() {
  var text = ""
  var i = 0;
  do {
    text += "<br>The number is " + i;
    i++;
  }
  while (i < 10)
  document.getElementById("demo").innerHTML = text;
}
</script>
</body>
</html>
```

3. **For Loop :**

Syntax :

```
for (statement 1; statement 2; statement 3) {
  code block to be executed
}
```

Statement 1 is executed before the loop (the code block) starts.

Statement 2 defines the condition for running the loop (the code block).

Statement 3 is executed each time after the loop (the code block) has been executed.

```
<html>
<body>
<p>Click the button to loop through a block of code five times.</p>
<button onclick="myFunction()">Try it</button>
<p id="demo"></p>
<script>
function myFunction() {
  var text = "";
  var i;
  for (i = 0; i < 5; i++) {
    text += "The number is " + i + "<br>";
  }
  document.getElementById("demo").innerHTML = text;
}
</script>
</body>
</html>
```


4.6 PROCESS OF DEBUGGING JAVA SCRIPT CODE

Searching for errors in programming code is called **code *debugging*.**

Debugging is not easy. But fortunately, all modern browsers have a built-in debugger.

Following are the steps to activate debugging.

1. **Chrome :**
   - Open the browser.
   - From the menu, select tools.
   - From tools, choose developer tools. (Ctrl+Shift+I)
   - Finally, select console.
2. **Firefox Firebug :**
   - Open the browser.
   - Go to the web page : http://www.getfirebug.com
   - Follow the instructions how to : Install Firebug
3. **Internet Explorer :**
   - Open the browser.
   - From the menu, select tools.
   - From tools, choose developer tools.
   - Finally, select Console.

Opera :

- Open the browser.
- Go to the webpage : http://dev.opera.com
- Follow the instructions how to :

Add a developer console button to your toolbar.

4.7 IMPLEMENT FUNCTIONS

4.7.1 CALL A FUNCTION

JavaScript functions are defined with the function keyword.

Syntax :

```
function functionName(parameters) {
  code to be executed
}
```

Declared functions are not executed immediately. They are "saved for later use", and will be executed later, when they are invoked (called upon).

```
<html>
<body>
<p>This example calls a function which performs a calculation, and returns the result:</p>
<p id="demo"></p>
<script>
function myFunction(a, b) {
  return a * b;
}
document.getElementById("demo").innerHTML = myFunction(4, 3);
</script>
</body>
</html>
```

4.7.2 PARAMETER PASSING

```
functionName(parameter1, parameter2, parameter3) {
  code to be executed
}
```

Function parameters are the names listed in the function definition.

Function arguments are the real values passed to (and received by) the function.

4.7.3 GLOBAL FUNCTIONS PROVIDED BY JAVA SCRIPT

Function	Description
decodeURI()	Decodes a URI.
decodeURIComponent()	Decodes a URI component.
encodeURI()	Encodes a URI.
encodeURIComponent()	Encodes a URI component.
eval()	Evaluates a string and executes it as if it was script code.
isFinite()	Determines whether a value is a finite, legal number.
isNaN()	Determines whether a value is an illegal number.
Number()	Converts an object's value to a number.
parseFloat()	Parses a string and returns a floating point number.
parseInt()	Parses a string and returns an integer.
String()	Converts an object's value to a string.

4.7.4 SCOPE AND LIFETIME OF VARIABLES WITHIN FUNCTIONS

Scope is the set of variables you have access to,

(i) Local JavaScript variable

(ii) Global JavaScript variable

1. **Local JavaScript Variables :** Variables declared within a JavaScript function, become LOCAL to the function.

   Local variables have local scope : They can only be accessed within the function.

   Local variables are created when a function starts, and deleted when the function is completed.

2. **Global JavaScript Variables :** A variable declared outside a function, becomes GLOBAL.

A global variable has global scope : All scripts and functions on a web page can access it.

4.7.5 SMALL PROGRAMS USING RECURSION

Recursion is an important programming technique, in which a function calls itself.

```
function factorial(num)
{
  // If the number is less than 0, reject it.
  if (num < 0) {
    return -1;
  }
  // If the number is 0, its factorial is 1.
  else if (num == 0) {
    return 1;
  }
  // Otherwise, call this recursive procedure again.
  else {
    return (num * factorial(num - 1));
  }
}
var result = factorial(8);
document.write(result);
```

4.8 IMPLEMENT ARRAYS

An array is a special variable, which can hold more than one value at a time.

Using an array literal is the easiest way to create a JavaScript Array.

4.8.1 SINGLE AND MULTI DIMENSIONAL ARRAYS

Using an array literal is the easiest way to create a JavaScript Array.

Syntax :

```
var array-name = [item1, item2, ...];
```

Example : var cars = ["Saab", "Volvo", "BMW"];

4.8.2 DECLARE AN ARRAY

```
//Creates an array with no numbered entries
var arr = new Array(5);
test(arr);
// length: 5
var arr = [];
arr.length = 5;
test(arr);
// length: 5
var arr = ['a', 'b', 'c', 'd', 'e'];
test(arr);
// length:5, 0:a, 1:b, 2:c, 3:d, 4:e
var arr = [undefined, undefined, undefined, undefined, undefined];
test(arr);
// length:5, 0:undefined, 1:undefined, 2:undefined, 3:undefined, 4:undefined
```

4.8.3 MANIPULATE AN ARRAY

Changing the Length :

```
var arr = ['a', 'b', 'c', 'd', 'e', 'f'];
test(arr);
// length:6, 0:a, 1:b, 2:c, 3:d, 4:e, 5:f
arr.length = 5;
test(arr);
// length:5, 0:a, 1:b, 2:c, 3:d, 4:e
var arr = ['a','b','c','d','e','f',..];
```

```
test(arr);
// length:8, 0:a, 1:b, 2:c, 3:d, 4:e, 5:f
arr.length = 7;
test(arr);
// length:7, 0:a, 1:b, 2:c, 3:d, 4:e, 5:f
```

Removing Entries : JavaScript provides three methods pop, shift and splice that can remove entries from the array and which therefore reduce the length of the array. In each case the value (or values) removed are returned by the call.

```
// pop() removes the last element from an array
var arr = ['a','b','c','d','e','f'];
var el = arr.pop();
test(arr); // length:5, 0:a, 1:b, 2:c, 3:d, 4:e
console.log(el); // f
// shift() removes the first element from an array
var arr = ['a','b','c','d','e','f'];
var el = arr.shift();
test(arr); // length:5, 0:b, 1:c, 2:d, 3:e, 4:f
console.log(el); // a
// splice() can remove existing elements
var arr1 = ['a','b','c','d','e','f'];
var arr2 = arr1.splice(0,2); // remove 2 elements starting at index 0
test(arr1); // length:4, 0:c, 1:d, 2:e, 3:f
test(arr2); // length:2, 0:a, 1:b
var arr1 = ['a','b','c','d','e','f',,,'i'];
var arr2 = arr1.splice(6,2); // remove 2 elements starting at index 6
test(arr1); // length:7, 0:a, 1:b, 2:c, 3:d, 4:e, 5:f, 6:i
test(arr2); // length:2
```


Adding Entries : JavaScript provides e three methods push, unshift and slice for inserting new entries.

```
// push() adds one or more elements to the end of an array
var arr = ['a','b','c','d','e','f',,,,];
arr.push('j');
test(arr);
// length:10, 0:a, 1:b, 2:c, 3:d, 5:f, 9:j
// unshift() adds one or more elements to the beginning of an array
var arr = ['a','b','c','d','e','f',,,,];
arr.unshift('x');
test(arr);
// length:10, 0:x, 1:a, 2:b, 3:c, 4:d, 5:e, 6:f
arr1 = ['a','b','c','d','e','f',,,'i'];
arr2 = arr1.splice(6,0,'g','h'); // removes 0 elements from index 6, and inserts 'g', 'h'
test(arr1); // length:11, 0:a, 1:b, 2:c, 3:d, 5:f, 6:g, 7:h, 10:i
test(arr2); // length:0
<html>
<body>
<p>The push method appends a new element to an array.</p>
<button onclick="myFunction()">Try it</button>
<p id="demo"></p>
<script>
var fruits = ["Banana", "Orange", "Apple", "Mango"];
document.getElementById("demo").innerHTML = fruits;
function myFunction() {
    fruits.push("Lemon")
    document.getElementById("demo").innerHTML = fruits;
```

```
}
</script>
</body>
</html>
program below illustrates one dimensional array in javascript
<script>
var people = ["joseph","Maria", "Brian", "Susan"];
document.write(people[0]);
</script>
programs below illustrate multi-dimensional arrays in javascript
<script>
var people = [
["joseph", 27, "united states"],
["Maria", 21, "united states" ],
["Brian", 35, "united states" ],
["Susan", 42, "united states"]
];
document.write(people[0][0]);//display name joseph
</script>
<script>
var people = [
["joseph", 27, "United states"],
["Maria", 21, "Uraguay" ],
["Brian", 35, "United kingdom" ],
["Susan", 42, "Australia"]
];
document.write(people[0][0]);//display name joseph
</script>
<script>
var people = [
["joseph", 27, "United states",["blue","black"]],
```



```
["Maria", 21, "Uraguay", ",["brown","green"] ],
["Brian", 35, "United kingdom", ",["green","red"] ],
["Susan", 42, "Australia", ",["blue","blonde"]]
];
document.write(people[0][3][0]);//display name joseph eye colour blue
</script>
<script>
var people = [
["joseph", 27, "United states",["blue","black"]],
["Maria", 21, "Uraguay", ",["brown","green"] ],
["Brian", 35, "United kingdom", ",["green","red"] ],
["Susan", 42, "Australia", ",["blue","blonde"]]
];
People[2][3][1]="brown";
document.write(people[2][3][1]);//display brian hair color as brown
</script>
<html>
<body>
<script>
var people = [
["joseph", 27, "United states",["blue","black"]],
["Maria", 21, "Uraguay", ",["brown","green"] ],
["Brian", 35, "United kingdom", ",["green","red"] ],
["Susan", 42, "Australia", ",["blue","blonde"]]
];
//People[2][3][1]="brown";
//document.write(people[2][3][1]);//display brian hair color as brown
for(var i=0;i<people.length;i++)
{
document.write("<h2>Person "+i+1+"</h2>");
```

```
    for(var details in people[i])
    {
        document.write(people[i][details]+"<br>");
    }
}
</script>
</body>
</html>
```

4.9 VARIOUS OBJECTS PROVIDED BY JAVA SCRIPT

Math Object : The Math object allows you to perform mathematical tasks on numbers.

The Math object includes several mathematical methods.

```
<html>
<body>
<p>Math.random() returns a random number between 0 and 1.</p>
<button onclick="myFunction()">Try it</button>
<p id="demo"></p>
<script>
function myFunction() {
    document.getElementById("demo").innerHTML = Math.random();
}
</script>
</body>
</html>
```

String Object : Allows manipulation and formatting of text strings and determination and location of substrings within strings.

The Most Common Methods :

Methods	Explanation
Length	Returns the number of characters in a string.
indexOf()	Returns the index of the first time the specified character occurs, or -1 if it never occurs, so with that index you can determine if the string contains the specified character.
lastIndexOf()	Same as indexOf, only it starts from the right and moves left.
match()	Behaves similar to indexOf and lastIndexOf, but the match method returns the specified characters, or "null", instead of a numeric value.
substr()	Returns the characters you specified : (14,7) returns 7 characters, from the 14th character.
substring()	Returns the characters you specified : (7,14) returns all characters between the 7th and the 14th.
toLowerCase()	Converts a string to lower case.
toUpperCase()	Converts a string to upper case.

Examples : The length method.

The length method returns the number of characters in a string.

```
<html>
<body>
<script type="text/javascript">
var str="Web Enabling Tools is Cool!"
document.write("<p>" + str + "</p>")
document.write("str.length")
</script>
</body>
</html>
indexOf method
```

Test if a string contains a specified character. Returns an integer if it does and -1 if it does not. Use this method in a form validation.

```
<html>
<body>
<script type="text/javascript">
var str="Web Enabling Tools is Cool!"
var pos=str.IndexOf("Enabling")
if (pos>=0)
{
document.write("School found at position: ")
document.write(pos + "<br>")
} else {
document.write("Enabling not found!")
}
```

<p>This example tests if a string contains a specified word. If the word is found it returns the position of the first character of the word in the original string. Note: The first position in the string is 0!

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

Date object

The Date object is used to work with dates and times.

The Most Common Methods :

Methods	Explanation
Date()	Returns a Date object.
getDate()	Returns the date of a Date object (from 1-31).
getDay()	Returns the day of a Date object (from 0-6. 0=Sunday, 1=Monday, etc.)
getMonth()	Returns the month of a Date object (from 0-11. 0=January, 1=February, etc.)
getFullYear()	Returns the year of the Date object (four digits).

getHours()	Returns the hour of the Date object (from 0-23).
getMinutes()	Returns the minute of the Date object (from 0-59).
getSeconds()	Returns the second of the Date object (from 0-59).

Examples : Date,

Returns today's date including date, month, and year. Note that the getMonth method returns 0 in January, 1 in February etc. So add 1 to the getMonth method to display the correct date.

```
<html>
<body>
<script type="text/javascript">
var d = new Date()
document.write(d.getDate())
document.write(".")
document.write(d.getMonth() + 1)
document.write(".")
document.write(d.getFullYear())
</script>
</body>
</html>
```

Display Weekday : A simple script that allows you to write the name of the current day instead of the number. Note that the array object is used to store the names, and that Sunday=0, Monday=1 etc.

```
<html>
<body>
<script type="text/javascript">
var d = new Date()
var weekday=new Array("Sunday","Monday","Tuesday",
"Wednesday","Thursday","Friday","Saturday")
document.write("Today is " + weekday[d.getDay()])
```

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

Boolean Object : The Boolean object represents two values, either "true" or "false".

How to create javascript boolean object :

1. By using boolean literals.
2. By using Boolean() constructor.

Syntax :

```
Var b1 = true;
 var b2 = new Boolean(value);
<html>
<body>
 <script>
var boolean1=new Boolean(true);
 var boolean2=new Boolean(false);
var boolean3=true;
document.write("Boolean1: "+boolean1+ "</br>");
document.write("Boolean2: "+boolean2+ "</br>");
document.write("Boolean3: "+boolean3);
 </script>
 </body>
 </html>
```

Number Object : The Number object represents numerical date, either integers (or) floating-point numbers.

To create javascript number object :

1. By using number literals.
2. By using Number() constructor.

Syntax :

```
var num1 = value;
var num1 = new Number(value);
<html>
<body>
 <script>
 var num1=new Number(102);//Enteger value
 var num2=Number(102.7);//Floating point value
 var num3=Number(13e4);//Exponent value
document.write("Integer Value: "+num1 + "</br>");
document.write("Floating point Value: "+num2 + "</br>");
document.write("Exponent Value: "+num3);
</script>
 </body>
</html>
```