



ALAGAPPA UNIVERSITY

[Accredited with 'A+' Grade by NAAC (CGPA:3.64) in the Third Cycle
and Graded as Category-I University by MHRD-UGC]
(A State University Established by the Government of Tamil Nadu)



KARAIKUDI – 630 003

DIRECTORATE OF DISTANCE EDUCATION

M.Sc. [Computer Science]

III – Semester

341 33

WEB TECHNOLOGY

Author:

Dr. A. PADMAPRIYA

Associate Professor

Department of Computer Science

Alagappa University

Karaikudi

"The copyright shall be vested with Alagappa University"

All rights reserved. No part of this publication which is material protected by this copyright notice may be reproduced or transmitted or utilized or stored in any form or by any means now known or hereinafter invented, electronic, digital or mechanical, including photocopying, scanning, recording or by any information storage or retrieval system, without prior written permission from the Alagappa University, Karaikudi, Tamil Nadu.

SYLLABI-BOOK MAPPING TABLE

Web Technology

Syllabi	Mapping in Book
BLOCK I : HTML,XHTML AND STYLE SHEETS	
UNIT 1 : Introduction:HTML, XML and WWW, Basic HTML, document Body text, hyperlinks	(Pages 1-26)
UNIT 2 : Lists, using color and images, tables, multimedia objects	(Pages 27-45)
UNIT 3 : Style sheets: using styles, examples, formatting blocks of information	(Page 46-70)
BLOCK II : CLIENT SIDE PROGRAMMING	
UNIT 4 : Introduction: Dynamic HTML, JavaScript, variables, string manipulations, mathematical functions, operators, arrays, functions	(Pages 71-118)
UNIT 5 : Regular expressions, cookies and Events	(Pages 119-135)
UNIT 6 : Dynamic HTML with JavaScript: Data validation, messages and confirmations, writing to a different frame, Rollover buttons, moving images	(Pages 136-150)
BLOCK III : HOST OBJECTS	
UNIT 7 : Browsers and DOM, DOM history and levels, Intrinsic event handling	(Pages 151-164)
UNIT 8 : Representing web Data: XML, Documents and vocabularies, versions and declarations, namespaces	(Pages 165-175)
UNIT 9 : JavaScript and XML: Ajax, DOM based XML processing, SAX,XSL,XSLT,XPATH	(Pages 176-198)
BLOCK IV : SERVER SIDE PROGRAMMING	
UNIT 10 : Java Servlets, history of web applications, The power of Servlets, HTTP servlet basics, the servlet API, page generations	(Pages 199-206)
UNIT 11 : The servlet Lifecycle: The servlet alternative, servlet reloading, Init and Destroy, single thread model, background processing, load on startup, client side caching, server side caching	(Pages 207-218)
UNIT 12 : Retrieving information: the servlet, the server, the client	(Pages 219-226)

BLOCK V : JSP TECHNOLOGY

UNIT 13 : Introduction: Need, HTTP and servlet basics, HTTP request/response model, Servlets, anatomy of a JSP page, JSP application design with MVC **(Pages 227-235)**

UNIT 14 : Setting up JSP Environment: Installing the JSDK, Installing Tomcat server, testing tomcat, creating, installing, running a JSP page **(Pages 236-245)**

MODEL QUESTION PAPER **(pages 246-247)**

CONTENTS

BLOCK 1 : COMPUTER SECURITY INTRODUCTION

UNIT 1	Introduction to HTML	1-26
1.0	Introduction	
1.1	Objectives	
1.2	Internet	
1.3	World Wide Web	
1.4	HTML	
1.5	XML	
1.6	Basic HTML Tags	
1.7	Hyperlinks	
1.8	Answers to Check Your Progress	
1.9	Let us Sum Up	
1.10	Self-Assessment Exercises	
1.11	Suggested Readings	
UNIT 2	HTML Tags	27-45
2.0	Introduction	
2.1	Objectives	
2.2	Lists	
2.3	Colors	
2.4	Images	
2.5	Tables	
2.6	Multimedia Objects	
2.7	Answers to Check Your Progress	
2.8	Let us Sum Up	
2.9	Self-Assessment Exercises	
2.10	Suggested Readings	
UNIT 3	Style Sheets	46-70
3.0	Introduction	
3.1	Objectives	
3.2	Cascading Style Sheets	
3.3	Formatting Block of Information	
3.4	CSS Selectors	
3.5	Ways to Insert Styles	
3.6	Answers to Check Your Progress	
3.7	Let us Sum Up	

- 3.8 Self-Assessment Exercises
- 3.9 Suggested Readings

BLOCK II : CLIENT SIDE PROGRAMMING

UNIT 4 JavaScript	71-118
4.0 Introduction	
4.1 Objectives	
4.2 Dynamic HTML	
4.3 Java Script	
4.4 Variables	
4.5 Operators	
4.6 Statements	
4.7 Objects	
4.8 Mathematical Functions	
4.9 String Manipulators	
4.10 Arrays	
4.11 Functions	
4.12 Answers to Check Your Progress	
4.13 Let us Sum up	
4.14 Self-Assessment Exercises	
4.15 Suggested Readings	
UNIT 5 Cookies and Events	119-135
5.0 Introduction	
5.1 Objectives	
5.2 Regular Expressions	
5.3 Cookies	
5.4 Events	
5.5 Answers to Check Your Progress	
5.6 Let us Sum up	
5.7 Self-Assessment Exercises	
5.8 Suggested Readings	
UNIT 6 Dynamic HTML using JavaScript	136-150
6.0 Introduction	
6.1 Objectives	
6.2 Data Validation	
6.3 Messages and Confirmation	

- 6.4 Writing to a different frame
- 6.5 Rollover buttons
- 6.6 Moving images
- 6.7 Answers to Check Your Progress
- 6.8 Let us Sum up
- 6.9 Self-Assessment Exercises
- 6.10 Suggested Readings

BLOCK III : HOST OBJECTS

UNIT 7 Document Object Model 151-164

- 7.0 Introduction
- 7.1 Objectives
- 7.2 Document Object Model
- 7.3 Browsers and DOM
- 7.4 DOM history and levels
- 7.5 Document elements
- 7.6 Intrinsic Event Handling
- 7.7 Answers to Check Your Progress
- 7.8 Let us Sum up
- 7.9 Self-Assessment Exercises
- 7.10 Suggested Readings

UNIT 8 Representing Web Data 165-175

- 8.0 Introduction
- 8.1 Objectives
- 8.2 XML Basics
- 8.3 XML and HTML
- 8.4 Documents and Vocabularies
- 8.5 Versions and declarations
- 8.6 Namespaces
- 8.7 Answers to Check Your Progress
- 8.8 Let us Sum up
- 8.9 Self-Assessment Exercises
- 8.10 Suggested Readings

UNIT 9 JavaScript and XML 176-198

- 9.0 Introduction
- 9.1 Objectives

- 9.2 Reading XML data
- 9.3 Ajax
- 9.4 DOM based XML processing
- 9.5 SAX
- 9.6 XSL, XSLT, XPATH
- 9.7 Answers to Check Your Progress
- 9.8 Let us Sum up
- 9.9 Self-Assessment Exercises
- 9.10 Suggested Readings

BLOCK IV : SERVER SIDE PROGRAMMING

UNIT 10	Java Servlets	199-206
10.0	Introduction	
10.1	Objectives	
10.2	History of web applications	
10.3	The power of Servlets	
10.4	HTTP Servlet basics	
10.5	The Servlet API	
10.6	Page Generations	
10.7	Answers to Check Your Progress	
10.8	Let us Sum up	
10.9	Self-Assessment Exercises	
10.10	Suggested Readings	
UNIT 11	Servlet Life Cycle	207-218
11.0	Introduction	
11.1	Objectives	
11.2	The Servlet alternative	
11.3	Servlet Reloading	
11.4	Init And Destroy	
11.5	Single Thread Model	
11.6	Background Processing	
11.7	Load On Startup	
11.8	Client Side Caching and Server Side Caching	
11.9	Answers to Check Your Progress	
11.10	Let us Sum up	
11.11	Self-Assessment Exercises	
11.12	Suggested Readings	

UNIT 12	Retrieving Information	219-226
12.0	Introduction	
12.1	Objectives	
12.2	The Servlet	
12.3	The Server	
12.4	The Client	
12.5	Answers to Check Your Progress	
12.6	Let us Sum up	
12.7	Self-Assessment Exercises	
12.8	Suggested Readings	

BLOCK V : JSP TECHNOLOGY

UNIT 13	Java Server Pages	227-235
13.0	Introduction	
13.1	Objectives	
13.2	Need for JSP	
13.3	HTTP Servlet basics	
13.4	HTTP request/response model	
13.5	Anatomy of a JSP page	
13.6	JSP application design with MVC	
13.7	Answers to Check Your Progress	
13.8	Let us Sum up	
13.9	Self-Assessment Exercises	
13.10	Suggested Readings	
UNIT 14	Setting up JSP Environment	236-245
14.0	Introduction	
14.1	Objectives	
14.2	Installing the JSDK	
14.3	Installing the Tomcat Server	
14.4	Testing Tomcat	
14.5	Creating, installing and running a JSP page	
14.6	JSP Program Example	
14.7	Answers to Check Your Progress	
14.8	Let us Sum up	
14.9	Self-Assessment Exercises	
14.10	Suggested Readings	
	Model Question paper	246-247

INTRODUCTION

Web programming is about more than creating and formatting webpages and websites, though that is often a starting point for many. Using scripting languages such as JavaScript, Perl and PH, it is possible to add a lot more functionality to a web site. This course material will cover the essentials of working with the most important web technologies. The scope of this material begins by creating reasonably simple webpages with HTML, then working through related document and content tagging systems such as dynamic HTML and eventually XML.

This book follows the self-instruction mode or the SIM format wherein each unit begins with an 'Introduction' to the topic followed by an outline of the 'Objectives'. The content is presented in a simple and structured form with 'Check Your Progress' questions for better understanding. At the end of the each unit a list of 'Key Words' is provided along with a 'Summary' and a set of 'Self-Assessment Questions and Exercises' for effective recap.

BLOCK – I

HTML, XHTML AND STYLE SHEETS

UNIT- 1 INTRODUCTION TO HTML

Notes

Structure

- 1.0 Introduction
- 1.1 Objectives
- 1.2 Internet
- 1.3 World Wide Web
- 1.4 HTML
- 1.5 XML
- 1.6 Basic HTML Tags
- 1.7 Hyperlinks
- 1.8 Answers to Check Your Progress
- 1.9 Let us Sum up
- 1.10 Self-Assessment Exercises
- 1.11 Suggested Readings

1.0 INTRODUCTION

infrastructure that supports innumerable applications essential to everyday life. We use Web sites and the information they contain to create and connect with a seemingly unlimited amount of information. Web-based systems and technologies are now used for a vast number of applications, and this unit aims to provide an overview of one such technology – HTML.

Since its development just two decades ago, the World Wide Web has grown to become the

1.1 OBJECTIVES

After going through this unit, you will be able to:

- Learn the basics of Internet and www
- Understand the evolution of markup languages
- Understand about tags
- Create simple web pages using HTML

1.2 INTERNET

Computers are increasingly interconnected creating new pathways to the information assets. The term "**Internet**" is a generalization that covers thousands of interconnected networks around the world based on very different technologies.

The networks usually differ in any of the possible network specific parameter such as transmission medium, geographical size, number of nodes, transmission speed, throughput, reliability etc. The reason behind this generalization is because the Internet is independent of the physical hardware. In short, *it represents a homogeneous interface to its users in spite of the heterogeneous hardware* that it is based on.

Internet is the collection of networks connected via the public backbone and communicating across networks using TCP/IP.

Evolution of Internet

- The Internet what we are using today is developed from the very first network called ARPANET.
- The ARPANET (Advanced Research Projects Agency Network) developed to link US Defence Department researchers with those in several Universities in the USA. It became operational in late 1969.
- The first appearance of the term ‘Internet’ - which was coined by the Network Working Group - was in 1974 as an abbreviation for ‘Internetworking’.
- The things developed from there, with electronic mail soon becoming an important form of communication within the research community that used this technology.
- At that time, making use of the Internet was not something that the average person or business could easily do or find much value in.
- The later advent of the Web that use of the Internet became common, and a general topic of conversation in many communities.

The evolution of today’s Internet is visualized in the following figure. It has moved from Internet of content to Internet of Things.

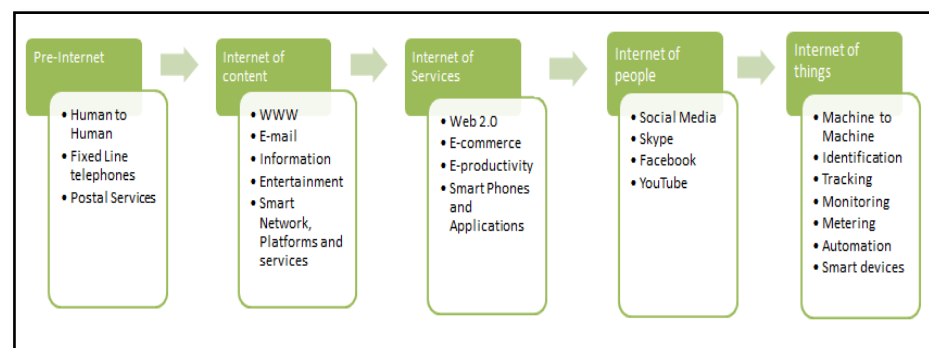


Fig 1.1. Evolution of today’s Internet

The Internet Timeline is given below. Only the notable events are listed in the table.

Table 1.1. Internet Timeline

Internet Timeline	
<i>Early research and development</i>	
<ul style="list-style-type: none"> • 1963: ARPANET concepts developed • 1969: ARPANET carries its first packets • 1972: Internet Assigned Numbers Authority (IANA) established • 1974: Transmission Control Program specification published • 1976: X.25 protocol approved • 1979: Internet Activities Board (IAB) • 1980: USENET news using UUCP • 1980: Ethernet standard introduced 	<i>Notes</i>
<i>Merging the networks and creating the Internet</i>	
<ul style="list-style-type: none"> • 1982: TCP/IP protocol suite formalized • 1982: Simple Mail Transfer Protocol (SMTP) • 1983: Domain Name System (DNS) • 1985: First .COM domain name registered • 1986: NSFNET with 56 kbit/s links • 1986: Internet Engineering Task Force (IETF) • 1988: OSI Reference Model released • 1991: World Wide Web (WWW) • 1992: Internet Society (ISOC) established • 1994: Full text web search engines 	
<i>Commercialization, privatization, broader access leads to the modern Internet</i>	
<ul style="list-style-type: none"> • 1995: New Internet architecture with commercial ISPs connected at NAPs • 1995: IPv6 proposed • 1999: IEEE 802.11b wireless networking • 2000: Dot-com bubble bursts • 2001: New top-level domain names activated • 2004: UN Working Group on Internet Governance (WGIG) • 2006: First meeting of the Internet Governance Forum • 2010: First internationalized country code top-level domains registered • 2016: ICANN contract with U.S. Dept. of Commerce ends, IANA oversight passes to the global Internet community • Continues... 	

The simple architecture of Internet is given in the figure below for better understanding.

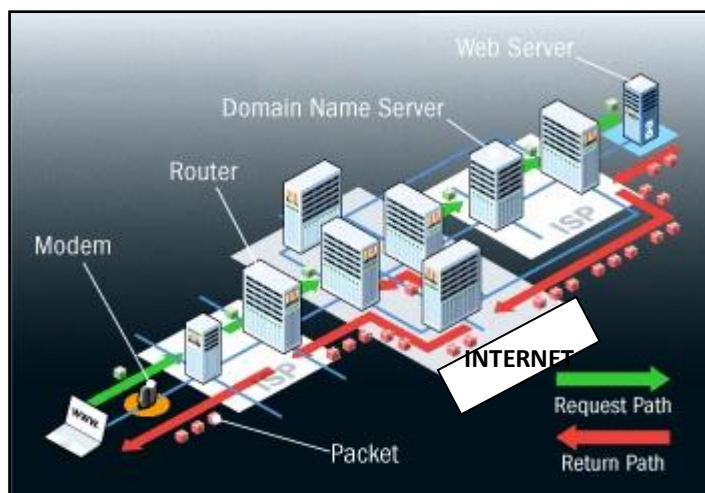


Fig 1.2. Simple Internet Architecture

Notes

The Internet has brought revolutionary change in the world of technologies, bringing the entire globe interconnected. But it follows some specific architecture and structure for communication. The most popular is the Client-Server architecture. In this client-server model,

- **Servers** – distributes and controls the majority of the resources as well as services for clients
- **Clients** – make use of the resources and services offered by the server

Such structural designs are made up of one or more client systems connected to central or main servers through a network. All such systems associated with it share computing resources. Here,

1. The client computer sends a request for data to the server through the internet
2. The server accepts the requested, process it and deliver the data packets requested back to the client.

One special feature is that the server computer has the potential to manage numerous clients at the same time. Also, a single client can connect to numerous servers at a single timestamp, where each server provides a different set of services to that specific client.

The following figure represents how the files are sent over the Internet. The file will be split into small packets; IP address is added to the packets and transmitted. At the receiving end the packets are rearranged in order before presenting them to the receiver.

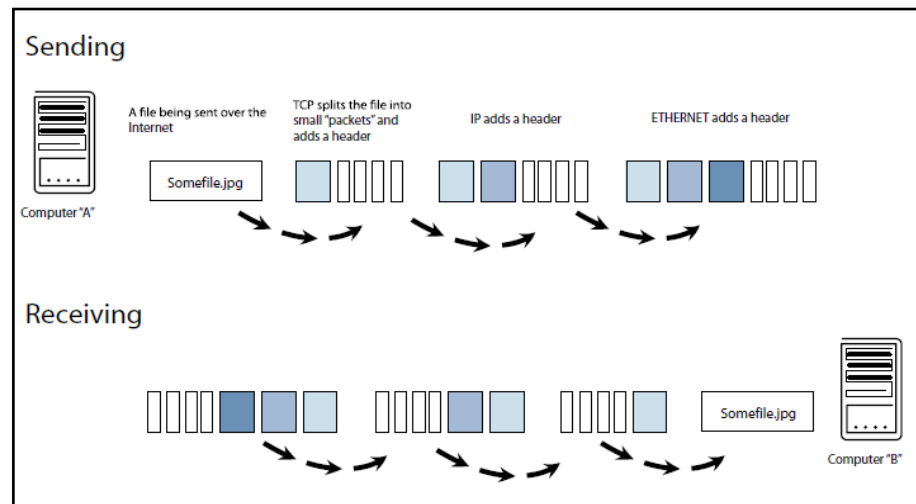


Fig 1.3. Diagram representing how files are sent over Internet

1.3 WORLD WIDE WEB

Today everyone knows of the World Wide Web and very many people around the world make daily use of its facilities. It is hard to imagine what it must have been like before the Web became such an important part of our lives.

But it was only in 1989 that Tim Berners-Lee, based at the European Laboratory for Particle Physics (CERN), in looking for a solution to the management and sharing of the large amounts of scientific information his colleagues created. He wrote a proposal for a large online hypertext database that by 1991 had become what we now call the World Wide Web. Thus the Web began as a means of improving information sharing and document handling between the research scientists at CERN and throughout the world.

It was designed to allow pages containing hypertext to be stored in a way that allowed other computers access to these pages. It was probably not until about the mid-1990s, however, that the Web began to really gain in popularity. It is no exaggeration to say that the Web has now become quite ubiquitous.



Fig 1.4. World Wide Web

The World Wide Web (WWW) is most often called the web. The World Wide Web is a collection of documents and services, distributed across the Internet and linked together by hypertext links. The web is therefore a *subset* of the Internet, not the same thing. All the computers use a communication standard called Hyper Text Transfer Protocol (HTTP).

- ✓ Web information is stored in documents called Web Pages.
Example : www.alagappauniversity.ac.in/dde.html
- ✓ Collection of Web Pages with information on a subject is called Web Site.
Example : www.alagappauniversity.ac.in
- ✓ Web pages are nothing but files stored on computers called Web servers.
Example : Google Web Server (GWS), Yahoo Server
- ✓ Computers reading the Web pages are called Web clients.
Example : Users device such as Computer, Laptops, Smart Phones etc

- ✓ Web clients view the pages with a program called a Web browser.



Fig 1.5. Web Browsers

The functioning of the Web is dependent upon many things, but perhaps the three most important are:

- First, that the web clients (browsers) are multiprotocol clients, capable of interacting with several different kinds of servers
- Second, that there is a common addressing scheme that makes it possible to unambiguously identify what you want and where to find it, anywhere on the web
 - A **URL (Uniform Resource Locator)** gives the address or location of any specific website. Each URL defines the path that will transmit the document, the Internet protocol being used, and the server on which the website is located.
 - Each Internet address is translated into a series of numbers called an IP address. A domain name is used by an organisational entity to identify its website and is based on the **Domain Name System (DNS)** hierarchy
- Finally, the fact that web browsers are extensible and therefore capable of handling a virtually unlimited variety of resource types.

Much of what is available on the web consists of web documents, which are often (somewhat misleadingly) called “home pages.” A *home page* appears to be a single entity, but is actually made up of a number of separate and distinct files, which may incorporate one or more of the following, in different configurations:

- *Text*
- *Graphics*
- *Animation*
- *Audio*
- *Video*
- *Hyperlinks (text or graphics which lead you from document to document)*
- *Interactive element, including specially embedded programs such as:*
 - *Plug-ins (downloadable sets of software that enable the user to use part of a web document.)*

Notes

The base document of a home page is a file which is mostly text, containing commands (“*markup*”) which determines how the above elements are configured. The rules governing this markup form a simple programming language called “*HTML*”.

1.4HTML

HTML, otherwise known as HyperText Markup Language, is the standard markup language used to create Web pages. HTML was originally developed by Tim Berners-Lee while at CERN, and popularized by the Mosaic browser developed at NCSA. During the course of the 1990s it has blossomed with the explosive growth of the Web. During this time, HTML has been extended in a number of ways. The Web depends on Web page authors and vendors sharing the same conventions for HTML. This has motivated joint work on specifications for HTML. HTML 2.0 (November 1995) was developed under the aegis of the Internet Engineering Task Force (IETF) to codify common practice in late 1994. HTML 3.0 (1995) proposed much richer versions of HTML. The versions of HTML are listed below

Version	Year
HTML	1991
HTML 2.0	1995
HTML 3.2	1997
HTML 4.01	1999
XHTML	2000
HTML5	2014

HTML also lets designers to create hyperlinks. **Hyperlinks** are areas of text, images, buttons, or other parts of a page where the viewer can click to navigate to additional content. Clicking a link can open a new web page, site, document, video, or animation.

- **Hypertext** allows words (or other objects) in one document to be linked to other documents. It provides a dynamic means of organising and accessing information where pages of information are connected together by hypertext links.

Structure of an HTML document

All HTML documents follow the same basic structure. They have the root tag as `<html>`, which contains `<head>` tag and `<body>` tag.

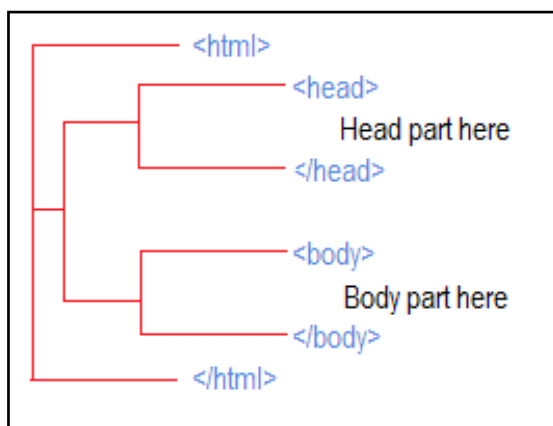


Fig 1.6. HTML Document Structure

- The head tag is used for control information by the browser
- The body tag contains the actual user information that is to be displayed on the screen.

Editors

HTML and CSS use text as their foundation. Because of this, even the most simple text editor, such as *Notepad* on the PC, or *TextEdit* on the Mac is capable of creating webpages. In addition to text editors, there are also fully featured web editors and design tools, such as *Adobe Dreamweaver* and *Microsoft Expression Web*. These are WYSIWYG tools (“What You See Is What You Get”) that provide a visual layout environment, code editing, along with website management tools.

Windows *Notepad* can be found on any Windows system in the Accessories panel. A HTML document can be created by following these steps:

1. Choose Start → Programs → Accessories → Notepad, and when the Notepad window appears, choose File → New.

By default, the file is saved in the text (.txt) format and so any HTML tags that the file contains cannot be interpreted by a web browser.

2. In Notepad, select File → Save As. Change the file extension from .txt to .html in the file name field.
3. Specify “All Files” in the Save as type field. Set the Encoding value to UTF-8 instead of ANSI; this is the necessary encoding for HTML pages.
4. Click the Save button.

HTML Elements

HyperText Markup Language is designed to specify the logical organisation of a document, with important hypertext extensions. It consists of series of *elements*. The elements tell the browser how to display the content. The HTML elements are represented by *tags*. The browsers do not display the HTML tags, but use them to depict the content of the page.

Notes

The detailed rules for HTML (the names of the tags/elements, how they can be used) are defined using another language known as the *Standard Generalized Markup Language*, or *SGML*.

HTML allows you to mark selections of text as titles or paragraphs, and then leaves the interpretation of these marked elements up to the browser. The web browser looks at the tags and displays them accordingly.

A simple example of HTML element is:

```
<p>Do you want to have lunch?</p>
```

The text to be displayed, `Do you want to have lunch?`, is wrapped by two tags indicating that it is a paragraph. The first tag is the opening tag `<p>` and the second is the closing tag `</p>`. These tags are generally not displayed in the browser, which reads the text from the web server and formats the text as a paragraph to display on the viewer's screen.

Example 1.1:

```
<html>

<head>
<title>
Basic HTML document
</title>
</head>

<body>
<h1>
Welcome to the world of Web Technologies
</h1>
<p>
A sample html program
</p>
</body>
</html>
```

Besides head and body tag, there are some other tags like title, which is a sub tag of head, which displays the information in the title bar of the browser. `<h1>` is used to display the line in its own format i.e., bold with some big font size. `<p>` is used to write the content in the form of paragraph.

Points to be noted

- ✓ Tags are delimited by angled brackets.
- ✓ They are not case sensitive i.e., <head>, <HEAD> and <Head> is equivalent.
- ✓ If a browser not understands a tag it will usually ignore it.
- ✓ Some characters have to be replaced in the text by escape sequences.
- ✓ White spaces, tabs and newlines are ignored by the browser.

Check Your Progress 1

1. What do you mean by client server computing?
2. What is atag?
3. Name the parts of the HTML document.

1.5XML

The XML stands for Extensible Markup Language (XML). It is a general-purpose specification for creating custom markuptlanguages. It is classified as an extensible language because it allows its users to define their own elements. Its primary purpose is to facilitate the sharing of structured data across different informationsystems, particularly via the Internet. It is used both to encode documents and to serialize data.

- XML is software and hardware independent.
- XML was designed to store and transport data.
- XML was designed to be both human- and machine-readable.

Example 1.2:

```
<?xml version="1.0"?>
<note>
  <to>Teddy</to>
  <from>Osito</from>
  <heading>Reminder</heading>
  <body>Internal Assessment this week!</body>
</note>
```

The XML above is quite self-descriptive

- ✓ It has sender information.
- ✓ It has receiver information
- ✓ It has a heading
- ✓ It has a message body.

HTML and XML look similar, because they are both SGML languages (SGML - Standard GeneralizedMarkup Language).

XML vs HTML

XML as well as HTML were designed with different goals

- XML was designed to carry data – with focus on what data is
- HTML was designed to display data – with focus on how data looks
- XML tags are not predefined – but user defined
- HTML tags are predefined
- XML is used to describe the structure of the document.
- HTML is used to describe the way in which the document is presented.

Notes

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

  <studdetail>

    <regno>20191001</regno>
    <name>
      <firstname>AADHI</firstname>
      <lastname>SUNDAR</lastname>
    </name>
    <country name="INDIA"/>
    <degree>M.Sc</degree>

  </studdetail>

</college>
```

The first line is the processing instruction which tells applications how to handle the XML. It also serves as version declaration and says that the file is XML

Valid or Well Formed XML

XML documents may be either valid or well formed.

A *well-formed XML document* is one which follows all of the rules of XML.

- Tags are matched
- Tags do not overlap
- Empty elements should be ended properly
- The document should contain an XML declaration.

A *valid XML document* has its own DTD. A DTD (Document Type Definition) defines what tags are legal and where they can occur in the XML. XML is said to be well structured if it follows the rules defined in DTD. There are many XML parsers that check the document and its DTD.

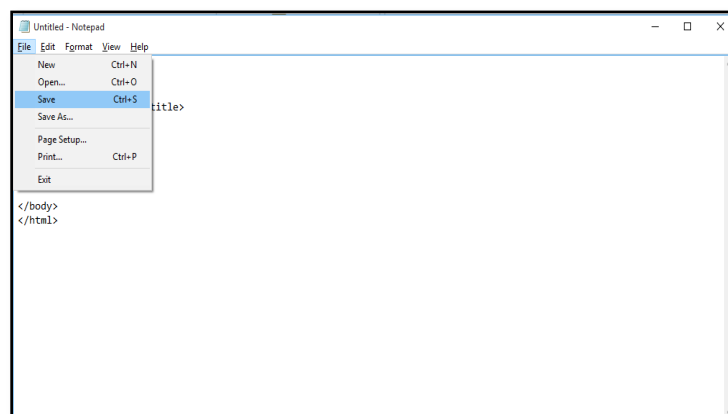
1.6 BASIC HTML TAGS

In any programming environment the first program while learning the language is 'Hello world' program. This program is used to check the programming environment. It will give an idea to the user about the program editor, how to save the web page and how to open the web page in the browser. It is one of the easiest codes used ever, but still we have to run this source code for better understanding the programming environment.

1. For creating any HTML source code, we need to open the text editor for writing the code.
2. Notepad is the most commonly used editor for creating HTML source code. Select the *Notepad* option from the *Start* Menu.
Or
Start → Programs → Accessories → Notepad, and when the Notepad window appears, choose File → New
3. In the opened notepad type the following code.

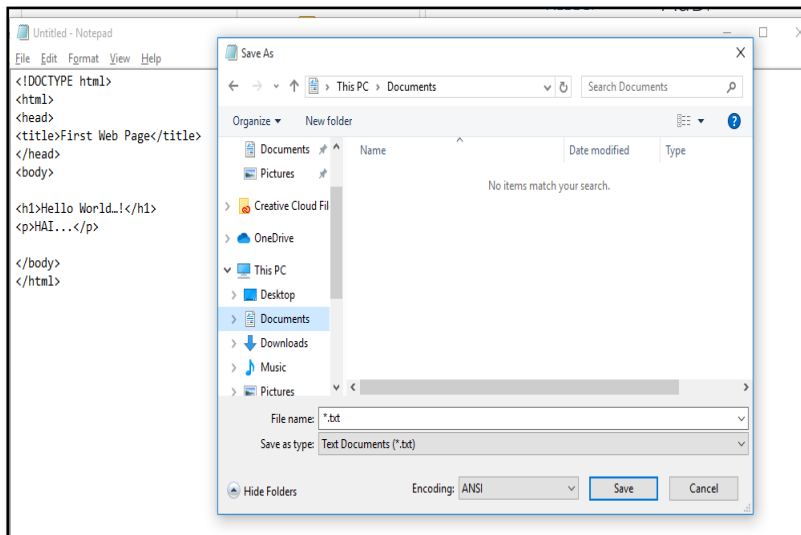
```
<!DOCTYPE html>
<html>
<head>
<title>First Web Page</title>
</head>
<body>
<h1>Hello World...!</h1>
<p>HAI...</p>
</body>
</html>
```

4. Save the file by click the Save option from the file menu

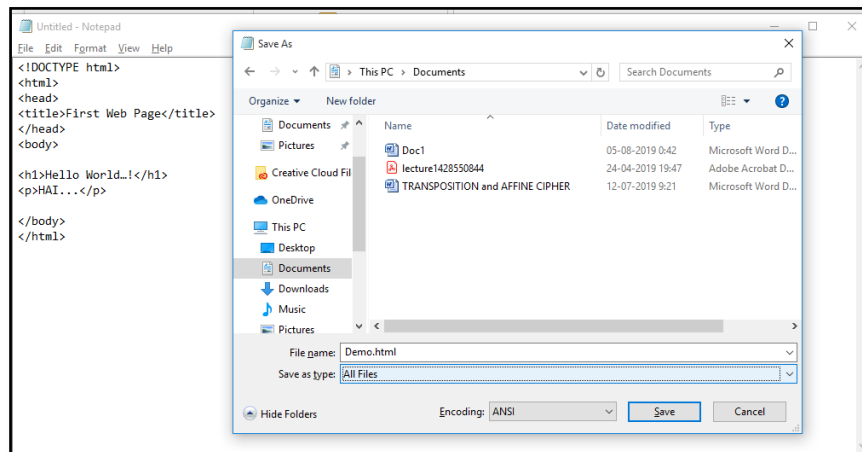


The Save dialogue box will appear as show below:

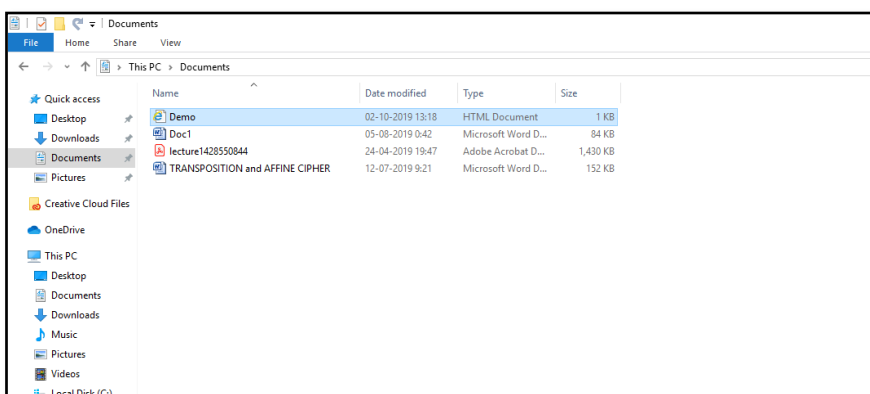
Notes



5. Choose the directory, give appropriate file name with extension *.html* and change the *Save as type* to 'All files'



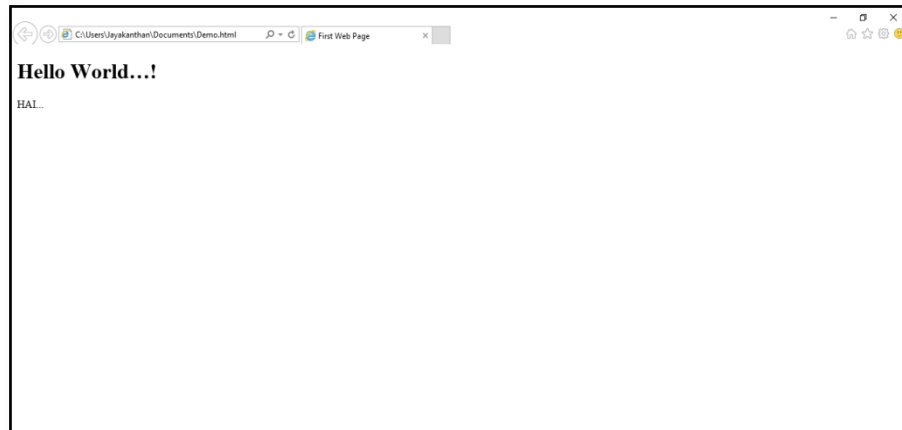
6. The created web page will be saved in the selected location



7. To view the web page created, *double click* the file name. The web page will be opened in the default browser. The commonly used browsers are Internet explorer, Chrome, Opera, NetScape.

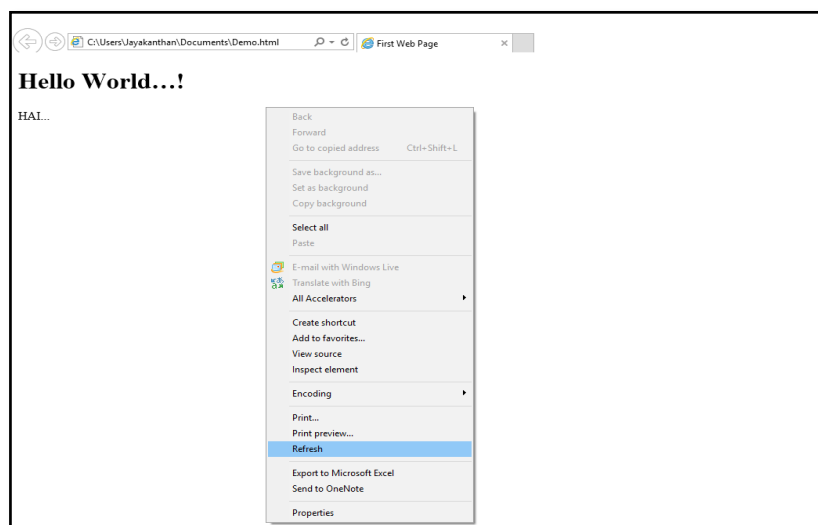
Notes

8. The output of the first web page is shown below

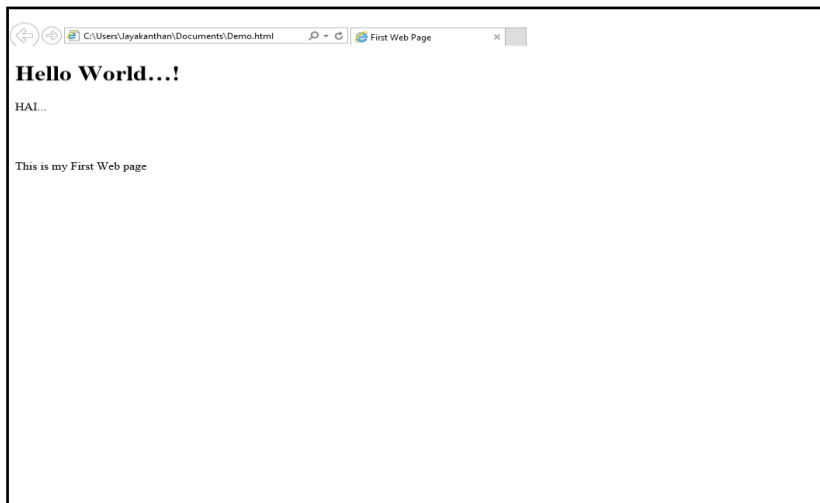


9. In case if the user needs to modify the HTML source, the source can be changed using the editor. To view the updated web page we need to press **Refresh** or **F5** key in the browser.

```
<!DOCTYPE html>
<html>
<head>
<title>First Web Page</title>
</head>
<body>
<h1>Hello World...!</h1>
<p>HAI...</p>
<br><br>
<p>This is my First Web Page </p>
</body>
</html>
```



10. The contents of the web page will be refreshed to reflect the changes in the source.



Notes

Example 1.3:

```
<!DOCTYPE html>
<html>
<head>
<title>Page Title</title>
</head>
<body>
<h1>My First Heading</h1>
<p>My first paragraph.</p>
</body>
```

- The `<!DOCTYPE html>` declaration defines this document to be HTML5.
- The `<html>` element is the root element of an HTML page.
- The `<head>` element contains meta information about the document.
- The `<title>` element specifies a title for the document.
- The `<body>` element contains the visible page content.
- The `<h1>` element defines a large heading.
- The `<p>` element defines a paragraph.

Comments

Comments in HTML documents start with `<!` and end with `>`. Each comment can contain as many lines of text as you like. If comment is having more lines, then each line must start and end with `--` and must not contain `--` within its body.

```
<html>
<! Single line Comment >

<! -- this is a comment line - -
-- which can have more lines - ->
```

Notes

HTML Elements

An HTML element usually consists of a start tag and an end tag, with the content inserted in between:

```
<tagname>Content goes here...</tagname>
```

The HTML element is everything from the start tag to the end tag:

```
<p>My first paragraph.</p>
```

HTML elements can be nested (elements can contain elements). All HTML documents consist of nested HTML elements.

```
<body>
  <h1>My First Heading</h1>
  <p> My first paragraph </p>
</body>
```

Empty HTML Elements

HTML elements with no content are called empty elements.
 is an empty element without a closing tag. The
 tag defines a line break.

Example 2:

```
<p>This is a paragraph without a line break.</p>
```

Output

This is a paragraph without a line break.
.....

```
<p>This is a <br> paragraph with a line break.</p>
```

Output

This is a
paragraph with a line break.

HTML Attributes

Attributes provide additional information about HTML elements.

- All HTML elements can have *attributes*
- Attributes provide *additional information* about an element
- Attributes are always specified in the *start tag*
- Attributes usually come in name/value pairs like: *name="value"*

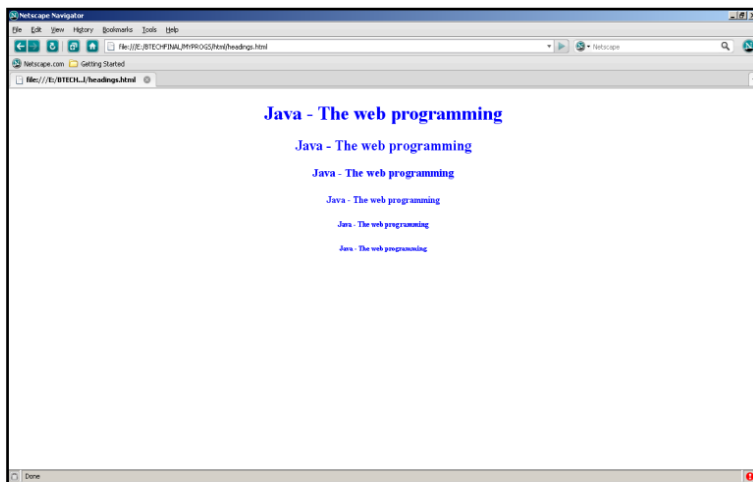
The following piece of code represents the values assigned to `text` attribute for the `body` tag and `align` attribute for `h1` tag.

```
<body text="blue">
<h1 align="center">
```

Notes

Example 1.4:

```
<html>
<body text="blue">
<center>
<h1 align="center"> Java - The web programming
<h2 align="center"> Java - The web programming
<h3 align="center"> Java - The web programming
<h4 align="center"> Java - The web programming
<h5 align="center"> Java - The web programming
<h6 align="center"> Java - The web programming
</center>
</body>
</html>
```



Body tag :

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

```
<body bgcolor="#F2F3F4"
background= "c:\image1.gif">
```

The following code will set the background color, text color and the size of the font to be displayed in the web page.

Notes

Example 1.5:

```
<html>
<head>
<title> example </title>
</head>

<body bgcolor="black" text="white">
<basefont size=7>

This is where you would include the text and
images on your web page.
</body>

</html>
```

Paragraph tag:

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

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

Heading tag:

HTML is having six levels of heading that are commonly used. The largest heading tag is <h1>. The different levels of heading tag besides <h1> are <h2>, <h3>, <h4>, <h5> and <h6>. These heading tags also contain attribute called as align.

```
<h1 align="left" | "right" | "center">
```

hr tag:

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

```
<hr width="50%">.
```

font tag:

This sets font size, color and relative values for a particular text.

```
<font size="10" color="#f1f2f3">
<font size=4 face="courier" color="red">
This text will be displayed using courier font
</font>
```

Text Formatting Tags

There are several tags to format the text. The following table describes some of the ways of formatting the text in the HTML document.

Tag	Text Format
	Bold text
<i>	Italic text
<u>	Underline text
	Important text
<strike>	Strike through text
	Emphasized text
<mark>	Marked text
<small>	Small text
	Deleted text
<ins>	Inserted text
<sub>	Subscript text
<sup>	Superscript text

Notes

Example 1.6:

```
<html>
<head>
<title>FORMATTING TAG</title>

<body bgcolor="pink">
<font color="blue" size="6" face="Times New
Roman">
<center>FORMATTING TAG</center>

<p>I am<b> bold </b><br><br>
<p>I am <i> italic </i><br><br>
<p>I am<u> underlined </u><br><br>

<p>The following word uses
<strike> strike tyle</strike><br><br>
<p>

The following word uses
<tt> monospaced</tt>
<br><br>

<p>The following word uses a<sup>
superscript</sup><br><br>
<p>The following word uses a
<sub>subscript</sub><br><br>

<p>I want to drink <del>cold</del>
<ins> coffee </ins><br><br>
<p>I want to drink <del>milk</del>
<ins> shake </ins><br><br>
```

Notes

```

<p>The following word uses a <big> big </big>
typeface.<br><br>
<p>The following word uses a <small> small
</small> typeface.<br><br>

<div id="menu" align="middle" >
<a href="/index.htm">HOME</a>
<a href="/about/contact_us.htm">CONTACT</a> |
<a href="/about/index.htm">ABOUT</a>
</div>

<div id="content" align="left" bgcolor="white">
<h5>Content Articles</h5>
<p>Actual content goes here.....
</div>

<br><br>

<p>This is the example of
<span style="color:green">span tag</span>
and the
<span style="color:red">div tag</span>
along with CSS<br><br></p>
</font>

</body>
</head>
</html>

```

Some other text formats are list below

Tag	Text Format
<abbr>	Defines an abbreviation or acronym
<address>	Defines contact information for the author/owner of a document
<bdo>	Defines the text direction
<blockquote>	Defines a section that is quoted from another source
<cite>	Defines the title of a work
<q>	Defines a short inline quotation

Character Entity References

Character entity references have the format &name; where name is a case-sensitive alphanumeric string. These are character escape sequence which are required if you want to display special characters that HTML uses as control sequences.

Entity References	Character
 	Space
&	&
<	<
>	>
"	“
'	‘

1.7 HYPERLINKS

HTML links are called as hyperlinks. Hyperlinks can be created using the anchor tag.

- The user can click on a link and jump to another document or other portion of the same document.
- When the user moves the mouse over a link, the mouse arrow will turn into a little hand.
- A link does not have to be text. It can be an image or any other HTML element.

Anchor tag is used to link two HTML pages and is represented by `<a>`. The simple syntax of the `anchor` tag is given below.

```
<a href="url">link text</a>
```

`href` is an attribute which is used for giving the path of a file which the user wants to link.

Two steps are necessary to create an anchor.

1. Create the anchor itself
2. Create a link to the anchor from another point in the document

By default, links will appear as follows in all browsers:

- An unvisited link is underlined and blue
- A visited link is underlined and purple
- An active link is underlined and red

In addition to the `href`, the following table lists the most commonly used attributes of the `anchor` tag.

Attribute	Value	Description
href	<i>URL</i>	Specifies the URL of the page the link goes to
name	<i>section_name</i>	Specifies the name of an anchor
target	<i>_blank</i> <i>_parent</i> <i>_self</i> <i>_top</i> <i>framename</i>	Specifies where to open the linked document

Notes

download	filename	Specifies that the target will be downloaded when a user clicks on the hyperlink
rel	alternate author bookmark external help license next nofollow noreferrer noopener prev search tag	Specifies the relationship between the current document and the linked document
type	media_type	Specifies the media type of the linked document

Example for name attribute:

```

<a href="#chap2">
Chapter Two
</a>
...
...
<a name="chap2">
Chapter 2
</a>
    
```

Table of Contents

[Introduction](#)

[Chapter One](#)

[Chapter Two](#)

Introduction

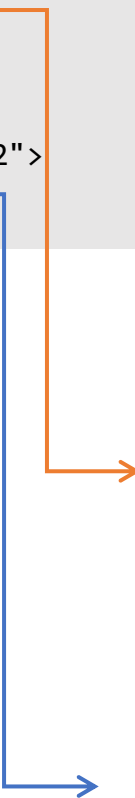
(Text for Introduction)

Chapter 1

(Text for Chapter 1)

Chapter 2

(Text for Chapter 2)



Example 1.7:**Index.html**

```

<html>
<body>

<h1 style="color:blue;">WELCOME</h1>
<center>
<h1><u>
<font color="blue">
Dictionary using phrase</font>
</u></h1>
</center>

<h2>
<a href="phrase1.html" target="ss">
once in a blue moon</a>
</h2>

<br><br>
<h2>
<a href="phrase2.html" target="ss">
Friend at court</a>
</h2>

<br><br>
<h2>
<a href="phrase3.html" target="ss">
Fit as a fiddle</a>
</h2>
<br><br>
</body>
</html>

```

phrase1.html

```

<html>
<body bgcolor="red">
<b>
<h1>
Answer: Rarely
</h1>
</b>
</body>
</html>

```

phrase2.html

```

<html>
<body bgcolor="pink">
<b>
<h1>
Answer: Influential person to help
</h1>

```

Notes

Notes

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

phrase3.html

```
<html>
<body bgcolor="green">
<b><h1>
Answer: In a perfect condition
</h1></b>
</body>
</html>
```

Check Your Progress 2

1. What is XML?
2. What are attributes?
3. State the purpose of anchor tag.

1.8 ANSWERS TO CHECK YOUR PROGRESS

1. In this client-server model,
 - i. **Servers** – distributes and controls the majority of the resources as well as services for clients
 - ii. **Clients** – make use of the resources and services offered by the server
2. The **elements** tell the browser how to display the content. The HTML elements are represented by **tags**.
3. The parts of the HTML document
 - i. Head part
 - ii. Body part
4. The XML stands for Extensible Markup Language (XML). It is a general-purpose specification for creating custom markup languages. It is classified as an extensible language because it allows its users to define their own elements. Its primary purpose is to facilitate the sharing of structured data across different information systems, particularly via the Internet
5. Attributes provide additional information about HTML elements.
 - i. All HTML elements can have **attributes**
 - ii. Attributes provide **additional information** about an element
 - iii. Attributes are always specified in the **start tag**
 - iv. Attributes usually come in name/value pairs like: **name="value"**
6. The anchor tag is used to link two HTML documents

1.9 LET US SUM UP

Internet is the collection of networks connected via the public backbone and communicating across networks using TCP/IP.

The World Wide Web (WWW) is most often called the web. The World Wide Web is a collection of documents and services, distributed across the Internet and linked together by hypertext links.

Web information is stored in documents called *Web Pages*

Collection of Web Pages with information on a subject is called *Web Site*

Web pages are nothing but files stored on computers called *Web servers*

Computers reading the Web pages are called *Web clients*

Web clients view the pages with a program called a *Web browser*

A *URL (Uniform Resource Locator)* gives the address or location of any specific website. Each URL defines the path that will transmit the document, the Internet protocol being used, and the server on which the website is located

Each Internet address is translated into a series of numbers called an IP address. A domain name is used by an organisational entity to identify its website and is based on the Domain Name System (DNS) hierarchy

Markup determines how the above elements in the web page are configured

Hyperlinks are areas of text, images, buttons, or other parts of a page where the viewer can click to navigate to additional content

The detailed rules for HTML (the names of the tags/elements, how they can be used) are defined using another language known as the *Standard Generalized Markup Language*, or *SGML*

HTML document consists of series of *elements*. The elements tell the browser how to display the content. The HTML elements are represented by *tags*.

The XML stands for Extensible Markup Language (XML). It is a general-purpose specification for creating custom markup languages.

1.10 SELF-ASSESSMENT EXERCISES

Short Questions

1. Define Internet.
2. What is an URL?
3. What is the need for DNS?
4. State the difference between HTML and XML
5. List the attributes of anchor tag.

Detail Questions

Notes

1. Describe the evolution of Internet
2. Write short notes on WWW.
3. Describe the structure of HTML document.
4. Brief about XML
5. Discuss about elements and attributes in HTML
6. With suitable examples explain the formatting tags of HTML
7. Explain about the anchor tag

1.11 SUGGESTED READINGS

1. Web Programming: Building Internet applications, Chris Bates, Wiley India
2. Web technologies – A computer science perspective, Jeffrey C Jackson, Pearson Education, 2006
3. Robert W. Sebesta “Programming the world wide web” Pearson Education
4. Bates, Developing web applications, wiley, 2006
5. www.w3schools.com

UNIT- 2HTML TAGS

Structure

- 2.0 Introduction
- 2.1 Objectives
- 2.2 Lists
- 2.3 Colors
- 2.4 Images
- 2.5 Tables
- 2.6 Multimedia Objects
- 2.7 Answers to Check Your Progress
- 2.8 Let us Sum Up
- 2.9 Self-Assessment Exercises
- 2.10 Suggested Readings

Notes

2.0 INTRODUCTION

A static web page also known as flat page or stationary page is a web page that is delivered to the user's web browser exactly as stored, in contrast to dynamic web pages which are generated by a web application. This unit will describe the elements used to create static web pages. Lists, images, tables and multimedia objects are the most commonly used elements of a static as well as dynamic web page.

2.1 OBJECTIVES

After going through this unit, you will be able to:

- Learn different types of listing options in HTML
- Understand how to include images in a web page
- Learn how to handle tables and multimedia objects
- Create static web pages using HTML

2.2 LISTS

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

- `` – An unordered list. This will list items using plain bullets.
- `` – An ordered list. This will use different schemes of numbers to list user items.
- `<dl>` – A definition list. This arranges user items in the same way as they are arranged in a dictionary.

Unordered Lists

An unordered list is a collection of related items that have no special order or sequence. This list is created by using HTML `` tag. Each item in the list is marked with a bullet.

Example 2.1:

```
<html>
<head>
<title>An Unordered List</title>
</head>

<body>
<ul>
<li>Beetroot</li>
<li>Ginger</li>
<li>Potato</li>
<li>Radish</li>
</ul>
</body>
</html>
```

Output

An Unordered List

- Beetroot
- Ginger
- Potato
- Radish

By default the bullet is of type disc. Following are the other possible options for the unordered list types

```
<ul type = "disc">
<ul type = "square">
<ul type = "circle">
```

Output for type="square"

An Unordered List:

- Beetroot
- Ginger
- Potato
- Radish

Output for type="circle"

An Unordered List:

- Beetroot
- Ginger
- Potato
- Radish

Ordered Lists

If the user wants to display the items in a numbered list instead of bulleted, then HTML ordered list will be used. This list is created by using `` tag. The numbering starts at one and is incremented by one for each successive ordered list element tagged with ``.

Notes

Example 2.2:

```
<!DOCTYPE html>
<html>
<head>
<title>HTML Ordered List</title>
</head>
<body>
<ol>
<li>Beetroot</li>
<li>Ginger</li>
<li>Potato</li>
<li>Radish</li>
</ol>
</body>
</html>
```

Output

1. Beetroot
2. Ginger
3. Potato
4. Radish

By default, it is a number. Following are the other possible options

```
<ol type = "1"><! Default-Case Numerals>
<ol type = "I"><! Upper-Case Numerals>
<ol type = "i"><! Lower-Case Numerals>
<ol type = "A"><! Upper-Case Letters>
<ol type = "a"><! Lower-Case Letters>
```

Output for type="I"

- I. Beetroot
- II. Ginger
- III. Potato
- IV. Radish

The start attribute of `` tag is used to specify the starting number of the type specified.

```
<ol type = "i" start = "4"><! Starts with iv>
```

Output for type="i" start="4"

- iv. Beetroot
- v. Ginger
- vi. Potato
- vii. Radish

Definition Lists

HTML and XHTML supports a list style which is called definition list where entries are listed like in a dictionary or encyclopaedia. The definition list is the ideal way to present a glossary, list of terms, or other name/value list.

Definition List makes use of following three tags.

```
<dl><! Defines the start of the list>
  <dt><! A term>
  <dd><! Term definition>
</dl><! Defines the end of the list>
```

Example 2.3:

```
<!DOCTYPE html>
<html>
<head>
<title>HTML Definition List</title>
</head>
<body>
<dl>
<dt><b>HTML</b></dt>
<dd>This stands for Hyper Text Markup Language
</dd>
<dt><b>HTTP</b></dt>
<dd>This stands for Hyper Text Transfer Protocol
</dd>
</dl>
</body>
</html>
```

Output**HTML**

This stands for Hyper Text Markup Language

HTTP

This stands for Hyper Text Transfer Protocol

2.3COLORS

Color can be used for background, elements and links. HTML colors are specified using predefined color names, or HEX, RGB,HSL, RGBA, HSLA values.

To change the color of links or of the page background hexadecimal (HEX) values are placed in the <body> tag.

```
<body bgcolor = "#nnnnnn" text = "#nnnnnn" link=
"#nnnnnn" vlink= "#nnnnnn" alink = "#nnnnnn">
```

Notes

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

In the six figure hexadecimal #rrggbb values, first two figures represent the red value, next two figures represent the green value and the last two figures represent the blue value. The following figure shows some example color values.


Color Name	Hex Value	Color
aqua	#00ffff	
black	#000000	
blue	#0000ff	
fuchsia	#ff00ff	
green	#008000	
gray	#808080	
lime	#00ff00	
maroon	#800000	
navy	#000080	
olive	#808000	
purple	#800080	
red	#ff0000	
silver	#c0c0c0	
teal	#008080	
white	#ffffff	
yellow	#ffff00	

Fig 2.1. Color Values

The following figure shows some example color names.



Fig 2.2. Color Names

Example 2.4:

```

<!DOCTYPE html>
<html>
<head>
<title>HTML Definition List</title>
</head>
<body>
<h1 style="background-color: DodgerBlue;">Hello
World</h1>
<br>
<p style="background-color: Tomato;">
Color can be used for background, elements and
links. HTML colors are specified using predefined
color names, or RGB, HEX, HSL, RGBA, HSLA values.
This paragraph will be displayed with tomato as
its background color</p>

</body>
</html>

```

Output

Hello World

Color can be used for background, elements and links. HTML colors are specified using predefined color names, or RGB, HEX, HSL, RGBA, HSLA values. This paragraph will be displayed with tomato as its background color

Example 2.5:

```

<h1 style="border: 2px solid Tomato;">
Hello World</h1>
<h1 style="border: 2px solid DodgerBlue;">
Hello World</h1>

```

Output

Hello World

Hello World

In HTML, a color can be specified as an **RGB** value, using this formula:

```
rgb(red, green, blue)
```

Each parameter (red, green, blue) defines the intensity of the color between 0 and 255.

For example, `rgb(255, 0, 0)` is displayed as red, because red is set to its highest value (255) and the others are set to 0.

To display black, set all color parameters to 0, `rgb(0, 0, 0)`.

To display white, set all color parameters to 255, `rgb(255, 255, 255)`.

The mixing of the RGB values is shown below:

`rgb(255, 99, 71)`

In HTML, a color can be specified using hue, saturation, and lightness (**HSL**) in the form:

`hsl(hue, saturation, lightness)`

- **Hue** is a degree on the color wheel from 0 to 360. 0 is red, 120 is green, and 240 is blue.
- **Saturation** is a percentage value, 0% means a shade of gray, and 100% is the full color.
- **Lightness** is also a percentage, 0% is black, 50% is neither light or dark, 100% is white

Example



HSLA color values are an extension of HSL color values with an alpha channel - which specifies the opacity for a color.

An HSLA color value is specified with:

`hsla(hue, saturation, lightness, alpha)`

The **alpha** parameter is a number between 0.0 (fully transparent) and 1.0 (not transparent at all)

Notes

Notes

Example`hsla(9, 100%, 64%, 0)``hsla(9, 100%, 64%, 0.2)``hsla(9, 100%, 64%, 0.8)``hsla(9, 100%, 64%, 1)`

2.4 IMAGES

The `` tag defines an image in an HTML page. Actually images are not technically inserted into an HTML page; images are linked to HTML pages. The `` tag creates a holding space for the referenced image.

```

```

The `` tag has two required attributes: `src` and `alt`. The other attributes used for the image tag are `height`, `width` and `align`.

Example 2.6:

```
<html>
<head>
<title>basictags</title>
</head>

<body bgcolor="pink">
<font color="black">
<font size="5">

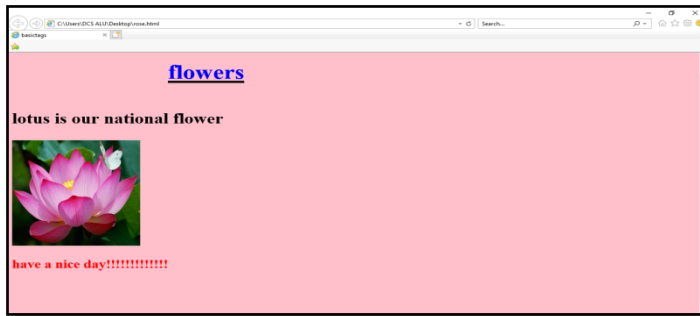
<marquee><h1><u><b>
<font color="blue">flowers</b></u></h1>
</marquee>

<h2>lotus is our national flower
</h2>



<h3>
<font color="red">have a nice
day!!!!!!!!!!!!!!!!!!!!</h3>

</body>
</html>
```

OutputNote:

To link an image to another document, simply nest the `` tag inside `<a>` tags.

Check Your Progress 1

1. What are the types of listings used in HTML?
2. Expand RGB, HEX and HSL.
3. State the purpose of 'alt' attribute in image tag.

2.5 TABLES

The HTML tables allow web authors to arrange data like text, images, links, other tables, etc. into rows and columns of cells.

The HTML tables are created using the `<table>` tag in which the `<tr>` tag is used to create table rows and `<td>` tag is used to create data cells. The elements under `<td>` are regular and left aligned by default

Example 2.7:

```
<html>
<head>
<title>HTML Tables</title>
</head>

<body>
<table border = "1">
<tr>
<td>Row 1, Column 1</td>
<td>Row 1, Column 2</td>
</tr>

<tr>
<td>Row 2, Column 1</td>
<td>Row 2, Column 2</td>
</tr>
</table>
```

Notes

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

Notes

Output

Row 1, Column 1	Row 1, Column 2
Row 2, Column 1	Row 2, Column 2

Here, the border is an attribute of `<table>` tag and it is used to put a border across all the cells. If user do not need a border, then user can use `border = "0"`.

Table Heading

Table heading can be defined using `<th>` tag. Headings, which are defined in `<th>` tag are centered and bold by default.

Example 2.8:

```
<html>
<head>
<title>HTML Table Header</title>
</head>
<body>

<table border = "1">
<tr>
<th>Name</th>
<th>Salary</th>
</tr>

<tr>
<td>Ramesh Raman</td>
<td>5000</td>
</tr>

<tr>
<td>Shabbir Hussein</td>
<td>7000</td>
</tr>
</table>
</body>
</html>
```

Output

Name	Salary
Ramesh Raman	5000
Shabbir Hussein	7000

Cellpadding and Cellspacing Attributes

Two attributes called `cellpadding` and `cellspacing` are used to adjust the white space in the table cells. The `cellspacing` attribute defines space between table cells, while `cellpadding` represents the distance between cell borders and the content within a cell.

Example 2.9:

```
<html>
<head>
<title>HTML Table Cellpadding</title>
</head>

<body>

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

<tr>
<th>Name</th>
<th>Salary</th>
</tr>

<tr>
<td>Ramesh Raman</td>
<td>5000</td>
</tr>

<tr>
<td>Shabbir Hussein</td>
<td>7000</td>
</tr>

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

Output

Name	Salary
Ramesh Raman	5000
Shabbir Hussein	7000

Colspan and Rowspan Attributes

`colspan` attribute is used if the user wants to merge two or more columns into a single column. Similarly `rowspan` is used to merge two or more rows.

Notes

Example 2.10:

```

<html>
<head>
<title>HTML Table Colspan/Rowspan</title>
</head>

<body>
<table border = "1">
<tr>
<th>Column 1</th>
<th>Column 2</th>
<th>Column 3</th>
</tr>

<tr>
<td rowspan = "2">Row 1 Cell 1</td>
<td>Row 1 Cell 2</td>
<td>Row 1 Cell 3</td>
</tr>

<tr>
<td>Row 2 Cell 2</td>
<td>Row 2 Cell 3</td>
</tr>

<tr>
<td colspan = "3">Row 3 Cell 1</td>
</tr>

</table>
</body>
</html>

```

Output

Column 1	Column 2	Column 3
Row 1 Cell 1	Row 1 Cell 2	Row 1 Cell 3
	Row 2 Cell 2	Row 2 Cell 3
Row 3 Cell 1		

Tables Backgrounds

User can set table background and border color using one of the following attributes

- **bgcolor** attribute – To set background color for whole table or just for one cell.
- **background** attribute – To set background image for whole table or just for one cell.
- User can also set border color also using **bordercolor** attribute.

Example 2.11:

```

<head>
<title>HTML Table Background</title>
</head>
<body>
<table border = "1" bordercolor = "green"
bgcolor = "yellow">

<tr>
<th>Column 1</th>
<th>Column 2</th>
<th>Column 3</th>
</tr>

<tr>
<td rowspan = "2">Row 1 Cell 1</td>
<td>Row 1 Cell 2</td>
<td>Row 1 Cell 3</td>
</tr>

<tr>
<td>Row 2 Cell 2</td>
<td>Row 2 Cell 3</td>
</tr>

<tr>
<td colspan = "3">Row 3 Cell 1</td>
</tr>

</table>
</body>
</html>

```

*Notes***Output**

Column 1	Column 2	Column 3
Row 1 Cell 1	Row 1 Cell 2	Row 1 Cell 3
	Row 2 Cell 2	Row 2 Cell 3
Row 3 Cell 1		

Here is an example of using background attribute. Here we will use an image available in /images directory.

Example 2.12:

```

<html>
<head>
<title>HTML Table Background</title>
</head>

```

Notes

```

<body>
<table border = "1" bordercolor = "green"
background = "/images/test.png">
<tr>
<th>Column 1</th>
<th>Column 2</th>
<th>Column 3</th>
</tr>

<tr>
<td rowspan = "2">Row 1 Cell 1</td>
<td>Row 1 Cell 2</td><td>Row 1 Cell 3</td>
</tr>

<tr>
<td>Row 2 Cell 2</td>
<td>Row 2 Cell 3</td>
</tr>

<tr>
<td colspan = "3">Row 3 Cell 1</td>
</tr>
</table>
</body>
</html>

```

Output

Column 1	Column 2	Column 3
Row 1 Cell 1	Row 1 Cell 2	Row 1 Cell 3
	Row 2 Cell 2	Row 2 Cell 3
Row 3 Cell 1		

Nested Tables

Not only tags but tables also nested in a web page.

Example 2.13:

```

<html>
<head>
<title>HTML Table</title>
</head>
<body>
<table border = "1" width = "100%">

<tr>
<td>
<table border = "1" width = "100%">

```

```

<tr>

<th>Name</th>
<th>Salary</th>
</tr>

<tr>
<td>Ramesh Raman</td>
<td>5000</td>
</tr>

<tr>
<td>Shabbir Hussein</td>
<td>7000</td>
</tr>

</table>
</td>
</tr>

</table>
</body>
</html>

```

*Notes*Output

Name	Salary
Ramesh Raman	5000
Shabbir Hussein	7000

2.6 MULTIMEDIA OBJECTS

The following are the major components of any multimedia system

- ✓ ***Text***
- ✓ ***Graphics***
- ✓ ***Animation***
- ✓ ***Audio***
- ✓ ***Video***

These multimedia objects can be embedded into the web page with the help of the following tags.

- <audio controls
- <video>
- <bgsound>
- etc

Notes

Example 2.14:

```

<html>
<head>
<title>HTML embed Tag</title>
</head>
<body>

<bgsound src = "/html/userrfile.mid">
<noembed>
    <img src = "userrimage.gif" >
    </noembed>
</bgsound>

</body>
</html>

```

Example 2.15:

```

<html>
<body>
<audio controls>
<source src="horse.ogg" type="audio/ogg">
<source src="horse.mp3" type="audio/mpeg">
</audio>
</body>
</html>

```

Output**Example 2.16:**

```

<html>
<head>
</head>
<body>

<center>
<h1>Html Image Example</h1>


<h1>Marquee Text</h1>

```

```

<marquee scrollamount="10" direction="left"
behavior="scroll">
Sample Marquee Text </marquee>

<h2>Playing videos in Html</h2>

<video width="320" height="240" controls>
<source src="SampleVideo.mp4" type="video/mp4">
<source src="movie.ogg" type="video/ogg">
Your browser does not support the video tag.
</video>

<h3>playing-audio in Html</h3>

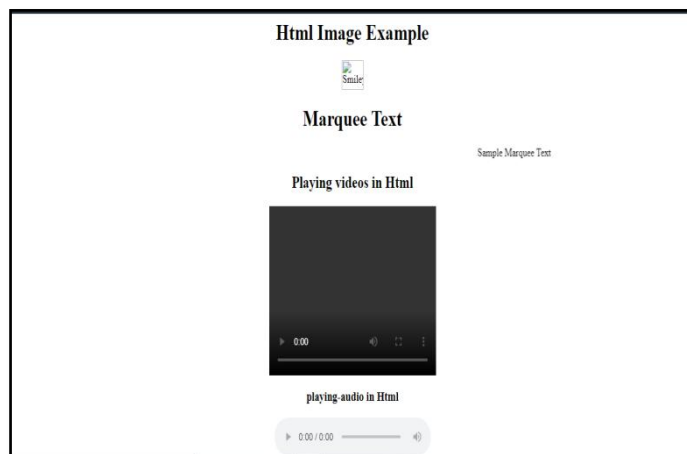
<audio controls>
<source src="rain.mp3" type="audio/mpeg">
Your browser does not support the audio element.
</audio>

</center>
</body>

<html>

```

Output



HTML 4 introduces the `<object>` element, which offers an all-purpose solution to generic object inclusion. The `<object>` element allows HTML authors to specify everything required by an object for its presentation by a user agent.

A HTML document can be embedded in another HTML document itself

```

<object data = "data/test.htm" type = "text/html"
width = "300" height = "200">
alt : <a href = "data/test.htm">test.htm</a>
</object>

```

APDFfile can be embedded in another HTML document

```
<object data = "data/test.pdf" type =
"application/pdf" width = "300" height = "200">
alt : <a href = "data/test.pdf">test.htm</a>
</object>
```

Check Your Progress 2

4. What is need for tables in HTML?
5. How to merge two columns?
6. What is the difference between `td` and `th` tags.

2.7 ANSWERS TO CHECK YOUR PROGRESS

1. The three types of lists used in HTML are,
 - i. *Unordered list*
 - ii. *Ordered list*
 - iii. *Definition list*
2. The *elements* tell the browser how to display the content. The HTML elements are represented by *tags*.
3. The alt attribute is used to display the text if the image is not properly opened.
4. The HTML tables allow web authors to arrange data like text, images, links, other tables, etc. into rows and columns of cells
5. `colspan` attribute is used if the user wants to merge two or more columns into a single column
6. `th` is used to represent header cells whereas `td` is used to represent data cells.

2.8 LET US SUM UP

The different tags used for listing are as follows.

- `` – An unordered list. This will list items using plain bullets.
- `` – An ordered list. This will use different schemes of numbers to list user items.
- `<dl>` – A definition list. This arranges user items in the same way as they are arranged in a dictionary.

HTML colors are specified using predefined color names, or HEX, RGB, HSL, RGBA, HSLA values.

The `` tag creates a holding space for the referenced image.

The HTML tables allow web authors to arrange data like text, images, links, other tables, etc. into rows and columns of cells.

Two attributes called `cellpadding` and `cellspacing` are used to adjust the white space in the table cells.

`colspan` attribute is used if the user wants to merge two or more columns into a single column. Similarly `rowspan` is used to merge two or more rows.

The `<object>` element allows HTML authors to specify everything required by an object for its presentation by a user agent.

Multimedia contents can be included in a web page using `img`, `bgsound`, `audio`, `video` and `object`

Notes

2.9 SELF ASSESSMENT EXERCISES

Short Questions

1. How to change the type of numerals in ordered list?
2. List the attributes of lists.
3. How will you represent colors using HSL?
4. What is the purpose of alt attribute in image tag?
5. State the need for tables.

Detail Questions

1. Describe the types of lists.
2. Write short notes on different coloring option used in HTML.
3. How to insert an image into a web page? Describe.
4. With suitable examples explain the TABLE tag of HTML.
5. Explain how to include multimedia contents into a web page.

2.10 SUGGESTED READINGS

1. Web Programming: Building Internet applications, Chris Bates, Wiley India
2. Web technologies – A computer science perspective, Jeffrey C Jackson, Pearson Education, 2006
3. Robert W. Sebesta “Programming the world wide web” Pearson Education
4. Bates, Developing web applications, wiley, 2006
5. www.w3schools.com

UNIT -3 STYLE SHEETS

Structure

- 3.0 Introduction
- 3.1 Objectives
- 3.2 Cascading Style Sheets
- 3.3 Formatting Block of Information
- 3.4 CSS Selectors
- 3.5 Ways to Insert Styles
- 3.6 Answers to Check Your Progress
- 3.7 Let us Sum Up
- 3.8 Self-Assessment Exercises
- 3.9 Suggested Readings

3.0 INTRODUCTION

This unit presents basic information about Cascading Style Sheets (CSS), a style sheet technology designed to work with HTML and XML documents. CSS provides a great deal of control over the presentation of a document, but to exercise this control intelligently requires an understanding of a number of features. This unit will help the reader to understand them better.

3.1 OBJECTIVES

After going through this unit, you will be able to:

- Learn different options to apply styles to web pages
- Understand how to format blocks of information
- Learn how to insert styles into a web page

3.2 CASCADING STYLE SHEETS

One of the most important aspects of HTML is the capability to separate presentation and content. A style is simply a set of formatting instructions that can be applied to a piece of text. The style is defined either embedded in each individual page itself or in an external style sheet file using a style sheet language such as CSS or XSLT.

Benefits of CSS

Separation of style and content has many benefits, but has only become practical in recent years due to improvements in popular web browser's CSS implementations.

- ✓ **Speed:** Overall, user's experience of a site utilizing style sheets will generally be quicker than sites that don't use the technology. 'Overall' as the first page will probably load more slowly because the style sheet AND the content will need to be transferred. Subsequent pages will load faster because no style information will need to be downloaded – the CSS file will already be in the browser's cache.
- ✓ **Maintainability:** Holding all the presentation styles in one file significantly reduces maintenance time and reduces the chance of human errors, thereby improving presentation consistency. For example, the font color associated with a type of text element may be specified -and therefore easily modified -throughout an entire website simply by changing one short string of characters in a single file. The alternate approach, using styles embedded in each individual page, would require a cumbersome, time consuming, and error-prone edit of every file.
- ✓ **Accessibility:** Sites that use CSS with either XHTML or HTML are easier to fine-tune so that they appear extremely similar in different browsers (Explorer, Mozilla, Opera, Safari, etc.).

Sites using CSS "degrade gracefully" in browsers unable to display graphical content, such as Lynx, or those so very old that they cannot use CSS. Browsers ignore CSS that they do not understand, such as CSS 3 statements. This enables a wide variety of user agents to be able to access the content of a site even if they cannot render the style sheet or are not designed with graphical capability in mind. For example, a browser using a refreshable for output could disregard layout information entirely, and the user would still have access to all page content.

- ✓ **Customization:** If a page's layout information is all stored externally, a user can decide to disable the layout information entirely, leaving the site's bare content still in a readable form. Site authors may also offer multiple style sheets, which can be used to completely change the appearance of the site without altering any of its content.

Most modern web browsers also allow the user to define their own style sheet, which can include rules that override the author's layout rules. This allows users, for example, to bold every hyperlink on every page they visit.

- ✓ **Consistency:** Because the semantic file contains only the meanings an author intends to convey, the styling of the various elements of the document's content is very consistent. For example, headings, emphasized text, lists and mathematical expressions all receive consistently applied style properties from the external style sheet. Authors need not concern themselves with the style properties at the time of composition. These presentational details can be deferred until the moment of presentation.

- ✓ **Portability:** The deferment of presentational details until the time of presentation means that a document can be easily re-purposed for an entirely different presentation medium with merely the application of a new style sheet already prepared for the new medium and consistent with elemental or structural vocabulary of the semantic document. A carefully authored document for a web page can easily be printed to a hard-bound volume complete with headers and footers, page numbers and a generated table of contents simply by applying a new style sheet.

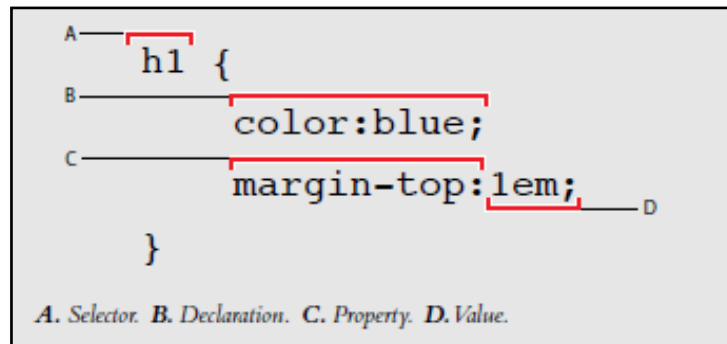
CSS Styles

A CSS style sheet consists of one or more style rules (sometimes called statements). This form of rule is called a ruleset and consists of two parts :

- A *selector*
- A *set of declarations / DeclarationBlock* – which is enclosed in { }

Example 3.1:

```
h1 {
  color:blue;
  margin-top:1em;
}
```



The *selector* string indicates the elements to which the rule should apply.

The *declaration* within the declaration block has two parts

- i. A property
- ii. A value

Declarations must be separated using colons and terminated using semicolons.

Syntax:

```
Selector{property: value; property: value .....}
```

Properties and values in styles

Some of the commonly used attributes and their values are listed below for reference.

Font Attributes	Values
Font-family	Comma is delimiter, sequence of fonts like cursive,sans etc
Font-style	Normal, italic, oblique
Font-weight	Normal,bold,bolder,lighter,or one of these numerical values(100 to 900)
Font-size	It is absolute size (xx-small,x-small,small,medium,large,x-large,xx-large),relative size(larger,smaller),a number(pixels)

Notes

Color and background Attributes	Values
Color	Sets an element text color
Background-color	Used to set back color
Background-image	Set background image

Text Attributes	Values
Text-decoration	None,underline,overline,line-through,blink
Vertical-align	top,bottom.middle,text-top,text-bottom
Text-transform	Capitalize,uppercase,lowercase
Text-align	Left,right,center,justify

Measurement units

Unitname	Abbreviation	Explanation
Em	Em	Height of the font
Pica	pc	1 pica is 12 points
Point	pt	1/72 of inch
pixel	px	One dot on screen
millimeter	mm	Printing unit
Centimeter	cm	Printing unit
inch	in	Printing unit

Margin related Attributes	Values
Margin-top	Percentage or length
Margin-bottom	Length or percentage
Margin-left	Length or percentage
Margin-right	Length or percentage

Program Example :

```
<html>
```

```
<head>
<title>My Web Page</title>

<style type="text/css">
h1
{
font-family:mssanserif;
font-size:30;
font-style:italic;
fontweight: bold;
color:red;
background-color:blue;
border:thin groove
}

.m
{
border-width:thick;
border-color:red;
border-style:dashed
}

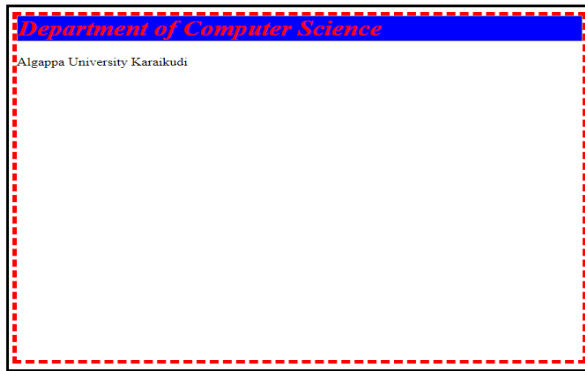
.mid
{
font-family:BankGothicLtBT;
text-decoration:link;
textttransformation:uppercase;
text-indentation:60%
}

</style>
</head>

<body class="m">

<h1> Department of Computer Science </h1>
<p class="mid">Algappa University Karaikudi</p>
</div>

</body>
</html>
```



Notes

Style Sheets

3.3 FORMATTING BLOCKS OF INFORMATION

Every HTML element has a default display value depending on what type of element it is. The two display values are

- ✓ Block
- ✓ Inline

Block-level Elements

A block-level element always starts on a new line and takes up the full width available (stretches out to the left and right as far as it can).

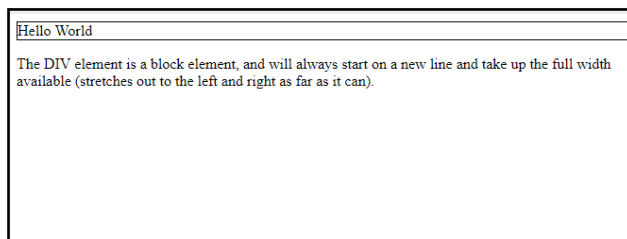
Example 3.2:

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

<div style="border: 1px solid black">Hello
World</div>

<p>The DIV element is a block element, and will
always start on a new line and take up the full
width available (stretches out to the left and
right as far as it can).</p>

</body>
</html>
```



Block level elements in HTML:

<address>		<header>
<article>		<hr>
<aside>		
<blockquote>		<main>
<canvas>		<nav>
<dd>	Style Sheets	<noscript>
<div>		
<dl>		<p>
<dt>		<pre>
<fieldset>		<section>
<figcaption>		<table>
<figure>		<tfoot>
<footer>		
<form>		<video>
<h1>-<h6>		

The <div> Element

The <div> element is a block-level element. The <div> element is often used as a container for other HTML elements. The <div> element has no required attributes, but style, class and id are common. When used together with CSS, the <div> element can be used to style blocks of content.

```
<div style="background-color:black;color:white;padding:20px;">
  <h2>London</h2>
  <p>London is the capital city of England. It is
the most populous city in the United Kingdom, with
a metropolitan area of over 13 million
inhabitants.
  </p>
</div>
```

Inline Elements

An inline element does not start on a new line and only takes up as much width as necessary.

This is an inlineelement inside a paragraph.

Example 3.3:

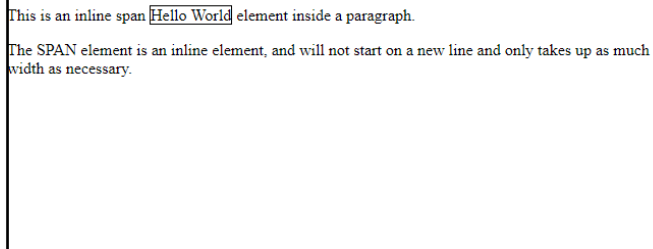
```
<!DOCTYPE html>
<html>
<body>

<p>This is an inline span <span style="border: 1px
solid black">Hello World</span> element inside a
paragraph.</p>
```

```
<p>The SPAN element is an inline element, and will not start on a new line and only takes up as much width as necessary.</p>
```

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

Style Sheets



This is an inline span **Hello World** element inside a paragraph.
The SPAN element is an inline element, and will not start on a new line and only takes up as much width as necessary.

Inline elements in HTML:

<a>	<map>
<abbr>	<object>
<acronym>	<output>
	<q>
<bdo>	<samp>
<big>	<script>
 	<select>
<button>	<small>
<cite>	
<code>	
<dfn>	<sub>
	<sup>
<i>	<textarea>
	<time>
<input>	<tt>
<kbd>	<var>
<label>	

The Element

The element is often used as a container for some text. The element has no required attributes, but style, class and id are common. When used together with CSS, the element can be used to style parts of the text.

```
<h1>My  
<span style="color:red">Important</span>  
Heading</h1>
```

3.4 CSS Selectors

Notes

CSS selectors are used to "find" (or select) the HTML elements you want to style. Generally CSS selectors are divided into the following five categories.

Style Sheets

- Simple selectors (select elements based on name, id, class)
- Combinator selectors (select elements based on a specific relationship between them)
- Pseudo-class selectors (select elements based on a certain state)
- Pseudo-elements selectors (select and style a part of an element)
- Attribute selectors (select elements based on an attribute or attribute value)

Simple Selectors

Example 3.4:

```
/* Headers have dark background */  
h1,h2,h3,h4,h5,h6 { background-color:purple }
```

A rule can also apply to multiple element types by using a selector string consisting of the comma-separated names of the element types. In the preceding style rule, each of the selectors (comma-separated components of the selector string) was simply the name of an element type. This form of selector is called a **type selector**. (Grouping Selector)

Example 3.5:

```
/* All elements bold */  
* {font-weight:bold }
```

The **Universal selector** is denoted by an asterisk (*). The universal selector represents every possible element type. So, for example, the above rule specifies a value of bold for the font-weight property of every element in the document.

Example 3.6:

```
/* Elements with certain id's have light  
background */  
#p1, #p3 { background-color:aqua }
```

Another form of selector is the **ID selector**. Recall that every element in an XHTML document has an associated `id` attribute, and that if a value is assigned to the `id` attribute for an element then no other element's `id` can

be assigned the same value. If a selector is preceded by a number sign (#), then it represents an id value rather than an element typename.

Example 3.7:

```
/* Elements in certain classes are italic, large
font, or both */

#p4, .takeNote { font-style:italic }
span.special { font-size:x-large }
```

Another HTML attribute that is frequently used with style sheets is class. This attribute is used to associate style properties with an element as follows. First, the stylesheet must contain one or more rulesets having **Class Selectors**, which are selectors that are preceded by a period (.), such as .takeNote in the rule given above.

Then any element that specifies takeNote (without the leading period) as the value of its class attribute will be given the properties specified in the declaration block of the corresponding style rule. Thus, the first paragraph of the example is displayed in an italic font. An element can be assigned to multiple style classes by using a space-separated list of class names as the value of the class attribute. For example, a span element with start tag

```
<span class="takeNote special cool">
```

will be affected by any rules for the takeNote, special, and cool classes. Thus, the second sentence of the second paragraph of the example is italicized, since it belongs to the takeNote class, among others. If a class name does not correspond to a class selector in any of the style rules for a document, then that class value is ignored.

ID and class selectors can also be prefixed by an element type name, which restricts the selector to elements of the specified type. For example, the style rule

```
span.special { font-size:x-large }
```

applies only to span elements that have a class value of special.

Selector	Example	Example description
.class	.intro	Selects all elements with class="intro"
#id	#firstname	Selects the element with id="firstname"
*	*	Selects all elements

Notes

<i>element</i>	p	Selects all <p> elements
<i>element, element, ..</i>	div, p	Selects all <div> elements and all <p> elements

Combinator Selectors:

A CSS selector can contain more than one simple selector. Between the simple selectors, we can include a combinator. There are four different combinators in CSS:

- descendant selector (space)
- child selector (>)
- adjacent sibling selector (+)
- general sibling selector (~)

Selector	Example	Description
Descendant Selector	div p { background-color: yellow; }	The descendant selector matches all elements that are descendants of a specified element
Child Selector	div > p { background-color: yellow; }	The child selector selects all elements that are the children of a specified element.
Adjacent Sibling Selector	div + p { background-color: yellow; }	The adjacent sibling selector selects all elements that are the adjacent siblings of a specified element. Sibling elements must have the same parent element, and "adjacent" means "immediately following"
General Sibling Selector	div ~ p { background-color: yellow; }	The general sibling selector selects all elements that are siblings of a specified element

Example 3.8:

```
<!DOCTYPE html>
<html>
<head>
<style>
div p {
  background-color: yellow;
```

```

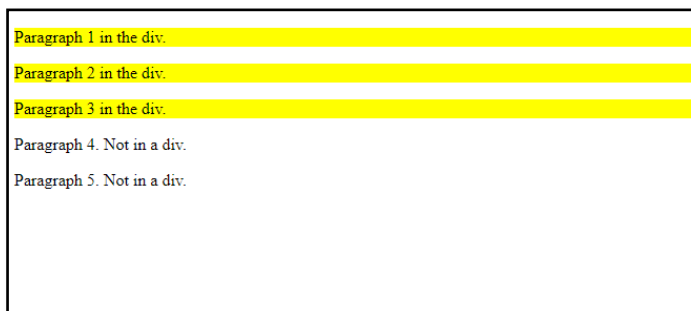
}
</style>
</head>
<body>

<div>
<p>Paragraph 1 in the div.</p>
<p>Paragraph 2 in the div.</p>
<section><p>Paragraph 3 in the div.</p></section>
</div>

<p>Paragraph 4. Not in a div.</p>
<p>Paragraph 5. Not in a div.</p>

</body>
</html>

```



Example 3.9:

```

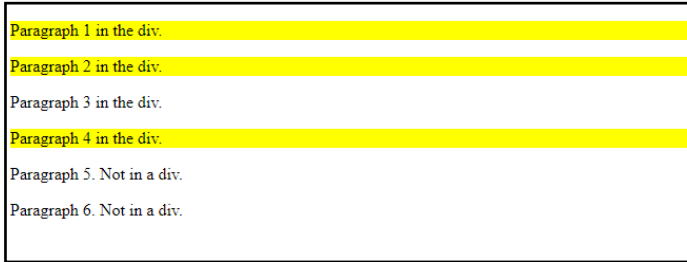
<!DOCTYPE html>
<html>
<head>
<style>
div > p {
  background-color: yellow;
}
</style>
</head>
<body>

<div>
<p>Paragraph 1 in the div.</p>
<p>Paragraph 2 in the div.</p>
<section><p>Paragraph 3 in the
div.</p></section><!-- not Child but Descendant --
>
<p>Paragraph 4 in the div.</p>
</div>

<p>Paragraph 5. Not in a div.</p>
<p>Paragraph 6. Not in a div.</p>

```

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



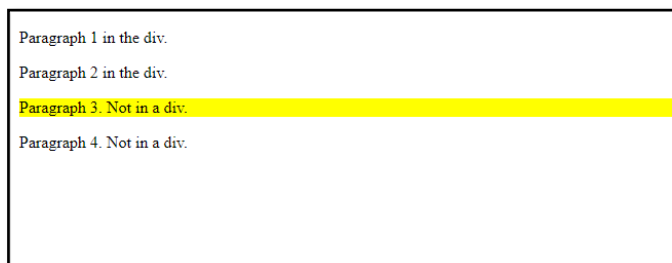
Example 3.10:

```
<!DOCTYPE html>
<html>
<head>
<style>
div + p {
  background-color: yellow;
}
</style>
</head>
<body>

<div>
<p>Paragraph 1 in the div.</p>
<p>Paragraph 2 in the div.</p>
</div>

<p>Paragraph 3. Not in a div.</p>
<p>Paragraph 4. Not in a div.</p>

</body>
</html>
```



Example 3.11:

```
<!DOCTYPE html>
<html>
<head>
```

```

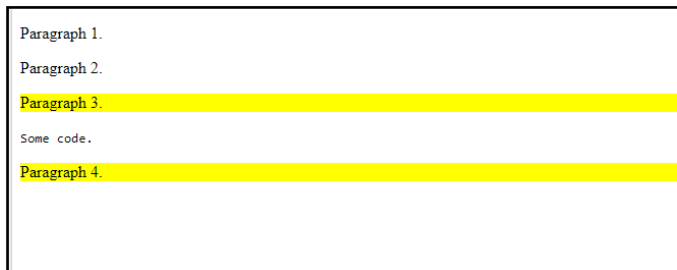
<style>
div ~ p {
  background-color: yellow;
}
</style>
</head>
<body>

<p>Paragraph 1.</p>
<div>
<p>Paragraph 2.</p>
</div>

<p>Paragraph 3.</p>
<code>Some code.</code>
<p>Paragraph 4.</p>
</body>
</html>

```

Notes



Pseudo-class Selectors:

A pseudo-class is used to define a special state of an element. For example, it can be used to:

- Style an element when a user move mouse over it
- Style visited and unvisited links differently
- Style an element when it gets focus

In addition to ID and class selectors, several predefined pseudo-classes are associated with “a” (anchor) elements that have an href attribute (source anchors).

Example 3.12:

```

/* Hyperlink ('a' element) styles */
a:link { color:black }
a:visited { color:yellow }
a:hover { color:green }
a:active { color:red }

```

The following table lists the pseudo-class selectors for “a” element.

Selector	Associated“a”Elements
<code>a:visited</code>	Any element with <code>href</code> corresponding to a URL that has been visited recently by the user
<code>a:link</code>	Any element that does not belong to the <code>a:visited</code> pseudo-class
<code>a:active</code>	An element that is in the process of being selected; for example, the mouse has been clicked on the element but not released
<code>a:hover</code>	An element over which the mouse cursor is located but that does not belong to the <code>a:active</code> pseudo-class

Pseudo-classes can be combined with CSS classes

Example 3.13:

```

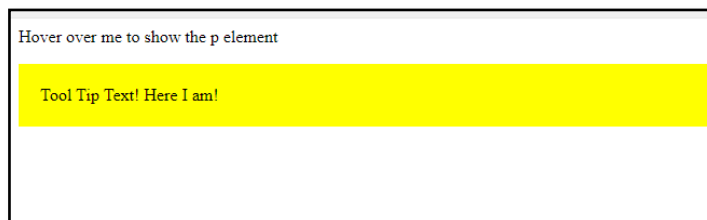
<!DOCTYPE html>
<html>
<head>
<style>
p {
  display: none;
  background-color: yellow;
  padding: 20px;
}

div:hover p {
  display: block;
}
</style>
</head>
<body>

<div>Hover over me to show the p element
<p>Tool Tip Text! Here I am!</p>
</div>

</body>
</html>

```



Pseudo-Elements Selectors:

A CSS pseudo-element is used to style specified parts of an element. For example, it can be used to:

- Style the first letter, or line, of an element
- Insert content before, or after, the content of an element

Syntax

```
selector::pseudo-element
{
  property:value;
}
```

Notes

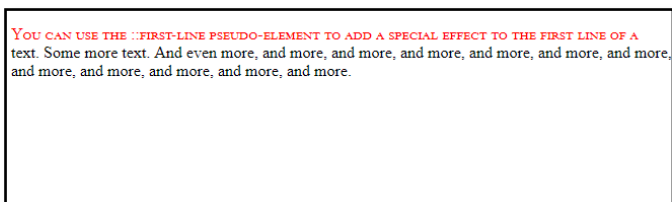
Style Sheets

Example 3.14:The ::first-line Pseudo-element

```
<!DOCTYPE html>
<html>
<head>
<style>
p::first-line {
  color: #ff0000;
  font-variant: small-caps;
}
</style>
</head>
<body>

<p>You can use the ::first-line pseudo-element to
add a special effect to the first line of a text.
Some more text. And even more, and more, and more,
and more, and more, and more, and more, and more,
and more, and more, and more, and more.</p>

</body>
</html>
```



Example 3.15:The ::before Pseudo-element

```
<!DOCTYPE html>
<html>
<head>

<style>
h1::before {
  content: url(smiley.gif);
}
```

Notes

```
</style>
</head>
<body>

<h1>This is a heading</h1>

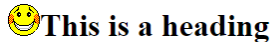
<p>The ::before pseudo-element inserts content
before the content of an element.</p>
```

Style Sheets

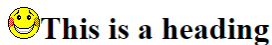
```
<h1>This is a heading</h1>

<p>
<b>Note:</b> IE8 supports the content property
only if a !DOCTYPE is specified.
</p>

</body>
</html>
```



The ::before pseudo-element inserts content before the content of an element.



Note: IE8 supports the content property only if a !DOCTYPE is specified.

Example 3.16: The ::selection Pseudo-element

```
<!DOCTYPE html>
<html>
<head>
<style>
::-moz-selection { /* Code for Firefox */
  color: red;
  background: yellow;
}

::selection {
  color: red;
  background: yellow;
}
</style>
</head>
<body>

<h1>Select some text on this page:</h1>

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

<div>This is some text in a div element.</div>
```



```
<p><strong>Note:</strong> ::selection is not
supported in Internet Explorer 8 and earlier
versions.</p>

<p><strong>Note:</strong> Firefox supports an
alternative, the ::-moz-selection property.</p>

</body>
</html>
```

Select some text on this page:

This is a paragraph.
 This is some **text in** a div element.
 Note: ::selection is not supported in Internet Explorer 8 and earlier versions.
 Note: Firefox supports an alternative, the ::-moz-selection property.

All CSS Pseudo Elements

Selector	Example	Example description
::after	p::after	Insert content after every <p> element
::before	p::before	Insert content before every <p> element
::first-letter	p::first-letter	Selects the first letter of every <p> element
::first-line	p::first-line	Selects the first line of every <p> element
::selection	p::selection	Selects the portion of an element that is selected by a user

Attribute Selectors:

It is possible to style HTML elements that have specific attributes or attribute values. The [attribute] selector is used to select elements with a specified attribute.

The following example selects all <a> elements with a target attribute:

Example 3.17:

```
<!DOCTYPE html>
<html>
<head>
<style>
a[target] {
  background-color: yellow;
}
```

```

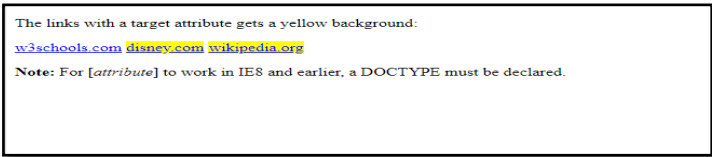
</style>
</head>
<body>

<p>The links with a target attribute gets a yellow
background:</p>

<a
href="https://www.w3schools.com">w3schools.com</a>
<a href="http://www.disney.com"
target="_blank">disney.com</a>
<a href="http://www.wikipedia.org"
target="_top">wikipedia.org</a>

<p><b>Note:</b>
For [attribute] to work in IE8 and earlier,
a DOCTYPE must be declared.
</p>

</body>
</html>
    
```



The [attribute="value"] selector is used to select elements with a specified attribute and value.

```

a[target="_blank"]
{
    background-color: yellow;
}
    
```

Selector	Example	Example description
[attribute]	[target]	Selects all elements with a target attribute
[attribute=value]	[target=_blank]	Selects all elements with target="_blank"
[attribute~value]	[title~=flower]	Selects all elements with a title attribute containing the word "flower"
[attribute =value]	[lang =en]	Selects all elements with a lang attribute value starting with "en"
[attribute^=value]	a[href^="https"]	Selects every <a> element whose href attribute value begins with "https"
[attribute\$=value]	a[href\$=".pd"]	Selects every <a>element

alue]	f"]	whose href attribute value ends with ".pdf"
[attribute*=v alue]	a[href*="w3s chools"]	Selects every <a> element whose href attribute value contains the substring "w3schools"

3.5 WAYS TO INSERT STYLES

There are three ways of inserting a style sheet:

- i. External CSS
- ii. Internal CSS
- iii. Inline CSS

External CSS

With an external style sheet, you can change the look of an entire website by changing just one file. Each HTML page must include a reference to the external style sheet file inside the <link> element, inside the head section.

Example 3.18:

demo.html

```
<!DOCTYPE html>
<html>
<head>

<link rel="stylesheet" type="text/css"
href="mystyle.css">

</head>
<body>

<h1>This is a heading</h1>
<p>This is a paragraph.</p>

</body>
</html>
```

An external style sheet can be written in any text editor, and must be saved with a .css extension. The external .css file should not contain any HTML tags.

mystyle.css

```
body
{
  background-color: lightblue;
}

h1
```

```
{
  color: navy;
  margin-left: 20px;
}
```

Internal CSS

An internal style sheet may be used if one single HTML page has a unique style. The internal style is defined inside the `<style>` element, inside the head section.

Example 3.19:

```
<!DOCTYPE html>
<html>
<head>
<style>
body {
  background-color: linen;
}

h1 {
  color: maroon;
  margin-left: 40px;
}
</style>
</head>
<body>

<h1>This is a heading</h1>
<p>This is a paragraph.</p>

</body>
</html>
```

Inline CSS

An inline style may be used to apply a unique style for a single element. To use inline styles, add the style attribute to the relevant element. The style attribute can contain any CSS property.

Example 3.20:

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

<h1 style="color:blue;text-align:center;">
This is a heading</h1>
<p style="color:red;">
```

```
This is a paragraph.</p>
</body>
</html>
```

Cascading Order

What style will be used when there is more than one style specified for an HTML element?

All the styles in a page will "cascade" into a new "virtual" style sheet by the following rules, where number one has the highest priority:

1. Inline style (inside an HTML element)
2. External and internal style sheets (in the head section)
3. Browser default

So, an inline style has the highest priority, and will override external and internal styles and browser defaults.

Check Your Progress

1. List the benefits of CSS
2. Write down the syntax for styles.
3. What are block level elements?
4. What do you mean by inline elements?
5. List the categories of CSS selectors.
6. How will you insert styles?

3.6 ANSWERS TO CHECK YOUR PROGRESS

1. Benefits of CSS are
 - i. Speed
 - ii. Maintainability
 - iii. Accessibility
 - iv. Customization
 - v. Consistency
 - vi. Portability.
2. A CSS style sheet consists of one or more style rules (sometimes called statements). This form of rule is called a ruleset and consists of two parts
 - *A selector*
 - *A set of declarations / Declaration Block – which is enclosed in { }.*

```
Selector{property: value; property: value .....}
```

3. A block-level element always starts on a new line and takes up the full width available.
4. An inline element does not start on a new line and only takes up as much width as necessary
5. Generally CSS selectors are divided into the following five categories.
 - Simple selectors (select elements based on name, id, class)
 - Combinator selectors (select elements based on a specific relationship between them)
 - Pseudo-class selectors (select elements based on a certain state)
 - Pseudo-elements selectors (select and style a part of an element)
 - Attribute selectors (select elements based on an attribute or attribute value)
6. There are three ways to insert styles namely
 - External CSS
 - Internal CSS
 - Inline CSS

3.7 LET US SUM UP

A style is simply a set of formatting instructions that can be applied to a piece of text.

The style is defined either embedded in each individual page itself or in an external style sheet file using a style sheet language such as CSS or XSLT.

A CSS style sheet consists of one or more style rules (sometimes called statements). This form of rule is called a ruleset and consists of two parts

- A *selector*
- A *set of declarations / Declaration Block* – which is enclosed in { }

The *selector* string indicates the elements to which the rule should apply.

The *declaration* within the declaration block has two parts

- i. A property
- ii. A value

Every HTML element has a default display value depending on what type of element it is. The two display values are

- ✓ Block
- ✓ Inline

The <div> element is a block-level element. The <div> element is often used as a container for other HTML elements. The <div> element has no required attributes, but style, class and id are common.

The `` element is often used as a container for some text. The `` element has no required attributes, but style, class and id are common.

A rule can also apply to multiple element types by using a selector string consisting of the comma-separated names of the element types. This form of selector is called a ***type selector***.

The ***Universal selector*** is denoted by an asterisk (*). The universal selector represents every possible element type.

A CSS selector can contain more than one simple selector. Between the simple selectors, we can include a combinator.

A pseudo-class is used to define a special state of an element. For example, it can be used to:

- Style an element when a user move mouse over it
- Style visited and unvisited links differently
- Style an element when it gets focus

A CSS pseudo-element is used to style specified parts of an element. For example, it can be used to:

- Style the first letter, or line, of an element
- Insert content before, or after, the content of an element

It is possible to style HTML elements that have specific attributes or attribute values. The `[attribute]` selector is used to select elements with a specified attribute

There are three ways of inserting a style sheet:

- i. External CSS
- ii. Internal CSS
- iii. Inline CSS.

3.8 SELF-ASSESSMENT EXERCISES

Short Questions

1. What do you mean by ID Selector?
2. State the purpose of Class Selectors.
3. What are the four types of Combinator Selectors?
4. What is the difference between pseudo class and pseudo element selectors?
5. What is drawback of inline styles?

Detail Questions

1. Write a note on Style Sheets
2. Discuss about the commonly used style property values
3. How to format blocks of information? Discuss.
4. Describe the types of CSS selectors.
5. Brief about `<div>` and ``
6. Discuss about attribute selectors
7. Explain about the ways to include styles into a page.

Notes

3.9SUGGESTED READINGS

1. Web Programming: Building Internet applications, Chris Bates, Wiley India
2. Web technologies – A computer science perspective, Jeffrey C Jackson, Pearson Education, 2006
3. Robert W.Sebesta “Programming the world wide web” Pearson Education
4. Bates, Developing web applications, wiley,2006
5. www.w3schools.com

BLOCK – II

CLIENT SIDE PROGRAMMING

Notes

UNIT- 4 JAVASCRIPT

Structure

- 4.0 Introduction
- 4.1 Objectives
- 4.2 Dynamic HTML
- 4.3 JavaScript
- 4.4 Variables
- 4.5 Operators
- 4.6 Statements
- 4.7 Objects
- 4.8 Mathematical Functions
- 4.9 String Manipulators
- 4.10 Arrays
- 4.11 Functions
- 4.12 Answers to Check Your Progress
- 4.13 Let us Sum up
- 4.14 Self-Assessment Exercises
- 4.15 Suggested Readings

4.0 INTRODUCTION

JavaScript is the programming language of the Web. JavaScript is the premier client-side interpreted scripting language. The overwhelming majority of modern websites use JavaScript, and all modern web browsers include JavaScript interpreters, making JavaScript the most ubiquitous programming language in history. JavaScript is part of the triad of technologies that all Web developers must learn: HTML to specify the content of web pages, CSS to specify the presentation of web pages, and JavaScript to specify the behavior of web pages. By combining all of these technologies, developers can create interesting and interactive websites. This unit will provide you basic insight into JavaScript.

4.1 OBJECTIVES

After going through this unit, you will be able to:

- Learn the need for dynamic web pages
- Understand how to write programs using Java Script
- Know the basics of arrays and functions

4.2 DYNAMIC HTML

The term Dynamic HTML, often abbreviated as DHTML, refers to the technique of making Web pages dynamic by client-side scripting to manipulate the document content and presentation. Web pages can be made more lively, dynamic, or interactive by DHTML technique. There is no such thing as a single Dynamic HTML standard. DHTML is an amalgam of specifications that stem from multiple standards efforts and proprietary technologies that are built into the two most popular DHTML-capable browsers.

DHTML is not a language but a term used to describe the way of making dynamic and interactive web pages. It is a combination of HTML, JavaScript, Cascading Style Sheets (CSS) and Document Object Model (DOM). Dynamic content is added to static HTML pages using scripts and styles. DHTML uses client side scripting languages like JavaScript to change the static attributes of a HTML page to generate a dynamic effect. This means all DHTML effects achieved are after loading of content on a page without interacting with server again.

Features of DHTML

- *Dynamic Content* – Content on the page is modified dynamically based on the user input. Below is the example of the content change when hovering the mouse over the text
- *Dynamic Style* – The appearance of an element on a web page is modified dynamically like color change or font change. Below is the example of dynamic font change on mouse hover
- *Dynamic Positioning* – The position of an element is dynamically changed relative to other elements on the page.
- *Dynamic Binding* – Linking an object at run time based on the conditions at that moment.

Advantages of DHTML

- ✓ DHTML supports adding styles to static content in various manners.
- ✓ It is dynamic so it can be changed even during the run time execution.
- ✓ Webmasters are often limited to use default fonts such as Arial or Times Roman. DHTML allows downloadable fonts which make the web pages looking more attractive.
- ✓ DHTML page is also saved as an .html file.

It is worth to note here that using multiple scripts on a web page will reduce the page loading speed and slower the site. Also dynamic pages may not perform well on search engines compared to static HTML pages.

Static vs Dynamic Webpages

Notes

Parameters	Static Webpage	Dynamic Webpage
Definition	<ul style="list-style-type: none"> • Loads the same content every time the page is reloaded. 	<ul style="list-style-type: none"> • Loads different content each time the page is loaded or refreshed. • Provides interactive features within the page without re-loading.
Browser	<ul style="list-style-type: none"> • Browser sends the HTTP request and receives the response from the server. • Interprets the markups in the received HTML document and display it as a webpage. 	<ul style="list-style-type: none"> • Client side scripting works in the same manner like a static page and the browser process the script code. • Server side scripting involves processing of the script code at server side before sending it to the client's browser.
Browser Settings	<ul style="list-style-type: none"> • No special settings required. 	<ul style="list-style-type: none"> • JavaScript is to be enabled in all type of browsers to see dynamic behaviour of a site.
Complexity	<ul style="list-style-type: none"> • Less complex 	<ul style="list-style-type: none"> • Client side scripting involves medium complexity. • Server side scripting involves high complexity.
Cost	<ul style="list-style-type: none"> • In most of the cases it costs only a domain name. • Free hosting is available with most of the website builder tools. 	<ul style="list-style-type: none"> • Separate hosting account is required to access server side, hence sites involving server side scripting needs high cost compared to the static or sites involving client side scripting.
Google Analytics	<ul style="list-style-type: none"> • Installing Analytics code is simple. • Copy the Analytics script code and paste it in the header section of a page. 	<ul style="list-style-type: none"> • Needs to follow the difficult process using Tracking Code wizard for installing Analytics code in a PHP or ASP page.
Interactive Features	<ul style="list-style-type: none"> • No interactive features are provided to the visitors. 	<ul style="list-style-type: none"> • Simple features like form validations are done with JavaScript. • Complex features like login module are created with PHP.
Language	<ul style="list-style-type: none"> • Static pages are generally created with Hyper Text Markup Language (HTML). • It is not necessary that all HTML pages are static. Scripts can be embedded in a static HTML document. 	<ul style="list-style-type: none"> • Dynamic pages are created with scripting languages. • Client side scripting languages includes JavaScript, ActionScript and Flash. • Server side scripting language includes PHP, ASP, JSP, ASP.NET, ColdFusion, Perl and WebDNA.

Notes

Layout and Content	<ul style="list-style-type: none"> • Both content and layout of a static webpages are fixed. 	<ul style="list-style-type: none"> • Layout and content can be changed independently in a dynamic webpage.
Loading	<ul style="list-style-type: none"> • Static page loads very fast since no script processing at client and server side is required. 	<ul style="list-style-type: none"> • Dynamic page loads slowly compared to a static page since it involves processing of client or server side scripts.
Multimedia	<ul style="list-style-type: none"> • Simple video or audio elements can be added to a static site. 	<ul style="list-style-type: none"> • Flash objects can be added to a dynamic site. • Flash objects respond to the user inputs and provide more interactive features.
Page Name	<ul style="list-style-type: none"> • Static page name mostly ends with .html or .htm. 	<ul style="list-style-type: none"> • Dynamic page name ends with .php or .asp.
Page Speed	<ul style="list-style-type: none"> • Loads very fast. 	<ul style="list-style-type: none"> • Loads slow due to script processing. • Sometimes script becomes non-responsive and forces the browser to close.
Personalized	<ul style="list-style-type: none"> • Content can't be personalized for a specific user. 	<ul style="list-style-type: none"> • Content can be personalized for a specific user based on login or any other parameter.
SEO	<ul style="list-style-type: none"> • Search engines easily index the static pages. 	<ul style="list-style-type: none"> • Since the content of a same page is changing, search engines finds it difficult to index dynamic pages. • Webmasters can redirect dynamic pages to a static one so that the search engines can index it easily.
Server	<ul style="list-style-type: none"> • Server receives the request and sends the HTML document as it is. 	<ul style="list-style-type: none"> • Server processes the script code if required before sending the document to the client.
Setup	<ul style="list-style-type: none"> • It is easy to setup a static website using any website builder tools. 	<ul style="list-style-type: none"> • It is difficult to setup a dynamic site since the setting up of content management system is more time consuming.
Some Examples	<ul style="list-style-type: none"> • Any fixed content site can be a static site. 	<ul style="list-style-type: none"> • Simple form validations using JavaScript. • Complex server side activities like login, session id tracking and payment gateway for credit card processing.
Source Code	<ul style="list-style-type: none"> • Source code of a static page will show the HTML content along with embedded client side scripts if any. • Right click on any webpage to see the source code. 	<ul style="list-style-type: none"> • Source code of a dynamic page will only show the HTML content and does not show any server side script code. • Example, check source code of any .php page and you will not find any PHP codes.

Suitability	<ul style="list-style-type: none"> • More suitable for distributing fixed information created and maintained by the site owner. 	<ul style="list-style-type: none"> • Suitable for sites providing more interactive and customized features based on user login or other inputs.
User Friendliness	<ul style="list-style-type: none"> • Less user friendly due to the fixed content. 	<ul style="list-style-type: none"> • More user friendly by providing customized content.

Notes

4.3 JAVASCRIPT

JavaScript was introduced in 1995 as a way to add programs to web pages in the Netscape Navigator browser. The language has since been adopted by all other major graphical web browsers. It has made modern web applications possible with which user can interact directly without doing a page reload for every action. JavaScript is also used in more traditional websites to provide various forms of interactivity and cleverness. The characteristics of JavaScript are

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

Advantages of JavaScript

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

One of the major strengths of JavaScript is that it does not require expensive development tools. One can start with a simple text editor such as Notepad. Since it is an interpreted language inside the context of a web browser, we don't even need to buy a compiler.

JavaScript can be implemented using JavaScript statements that are placed within the `<script>... </script>` HTML tags in a web page.

The `<script>` tag alerts the browser program to start interpreting all the text between these tags as a script. A simple syntax of JavaScript will appear as follows.

```
<script ...>
    JavaScript code
</script>
```

The script tag takes two important attributes –

- **Language** – This attribute specifies what scripting language is being used. Typically, its value will be `javascript`.
- **Type** – This attribute is recommended to indicate the scripting language in use and its value should be set to `"text/javascript"`.

So the JavaScript segment will look like

```
<script language = "javascript"
    type = "text/javascript">
    JavaScript code
</script>
```

Example 4.1

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

Points to be noted

- JavaScript programs are written using the Unicode character set
- JavaScript ignores spaces, tabs, and newlines that appear in JavaScript programs.
- Simple statements in JavaScript are generally followed by a semicolon character, just as they are in C, C++, and Java.
- JavaScript, however, allows us to omit this semicolon if each of our statements is placed on a separate line.

- JavaScript is a case-sensitive language.
- However, that HTML is not case-sensitive.
- For example, the HTML `onclick` event handler attribute is sometimes specified as `onClick` in HTML, but it must be specified as `onclick` in JavaScript code
- JavaScript is object-oriented.
- The JavaScript interpreter performs automatic garbage collection for memory management

*Notes***Example 4.2**

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

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

<p>This is web page body </p>
</body>
</html>
```

Example 4.3

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

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

<input type = "button" onclick = "sayHello()"
value = "Say Hello" />
</body>
</html>
```

Comments in JavaScript

JavaScript supports both C-style and C++-style comments, thus –

- Any text between a `//` and the end of a line is treated as a comment and is ignored by JavaScript.
- Any text between the characters `/*` and `*/` is treated as a comment. This may span multiple lines.
- JavaScript also recognizes the HTML comment opening sequence `<!--`. JavaScript treats this as a single-line comment, just as it does the `//` comment.
- The HTML comment closing sequence `-->` is not recognized by JavaScript so it should be written as `//-->`.

Example 4.4

```

<script language = "javascript" type =
"text/javascript">
<!--
    // It is single line comments similar to C++

    /*
    * This is a multi-line comment in JavaScript
    * It is similar to comments in C Programming
    */

    //-->
</script>

```

4.4 VARIABLES

A *literal* is a data value that appears directly in a program.

Example 4.5

```

12 // The number twelve
1.2 // The number one point two

"hello world" // A string of text

'Hi' // Another string

true // A Boolean value

/javascript/gi
/* A "regular expression" literal (for pattern
matching)
*/
null // Absence of an object

```


An *identifier* is simply a name. In JavaScript, identifiers are used to name variables and functions and to provide labels for certain loops in JavaScript code. A JavaScript identifier must begin with

- a letter,
- an underscore (`_`), or
- a dollar sign (`$`).

Subsequent characters can be letters, digits, underscores, or dollar signs. (*Digits are not allowed as the first character so that JavaScript can easily distinguish identifiers from numbers.*)

Example 4.6

```
i
my_variable_name
v13
_dummy
$str
```

Computer programs work by manipulating *values*, such as the number 3.14 or the text “Hello World”. The kinds of values that can be represented and manipulated in a programming language are known as *types*, and one of the most fundamental characteristics of a programming language is the set of types it supports.

JavaScript types can be divided into two categories: *primitive types* and *object types*.

- JavaScript’s primitive types include numbers, strings of text (known as *strings*), and Boolean truth values (known as *booleans*). The special JavaScript values null and undefined are primitive values, but they are not numbers, strings, or booleans.
- Any JavaScript value that is not a number, a string, a boolean, or null or undefined is an object. An object (that is, a member of the type *object*) is a collection of *properties* where each property has a name and a value (either a primitive value, such as a number or string, or an object). An ordinary JavaScript object is an unordered collection of named values. The language also defines a special kind of object, known as an *array*, which represents an ordered collection of numbered values. The JavaScript language includes special syntax for working with arrays, and arrays have some special behaviour that distinguishes them from ordinary objects.

JavaScript types can also be categorized as *mutable* and *immutable* types.

- A value of a mutable type can change. Objects and arrays are mutable: a JavaScript program can change the values of object properties and array elements.
- Numbers, booleans, null, and undefined are immutable—it doesn’t even make sense to talk about changing the value of a number, for

Notes

example. Strings can be thought of as arrays of characters, and you might expect them to be mutable. In JavaScript, however, strings are immutable: you can access the text at any index of a string but JavaScript provides no way to alter the text of an existing string

When a program needs to retain a value for future use, it assigns the value to (or “stores” the value in) a variable. A **variable** defines a symbolic name for a value and allows the value to be referred to by name. The way that variables work is another fundamental characteristic of any programming language. Before using a variable in a JavaScript program, we must declare it. Variables are declared with the `var` keyword as follows.

Example 4.7

```
<script type = "text/javascript">
<!--
    var money;
    var name;
//-->
</script>
```

Example 4.8

It is possible to declare multiple variables with the same `var` keyword as follows

```
<script type = "text/javascript">
<!--
    var money, name;
//-->
</script>
```

Storing a value in a variable is called **variable initialization**. We can do variable initialization at the time of variable creation or at a later point in time when we need that variable.

Example 4.9

For instance, we might create variable named **money** and assign the value 2000.50 to it later. For another variable, we can assign a value at the time of initialization as follows.

```
<script type = "text/javascript">
<!--
    var name = "Ali";
    var money;
    money = 2000.50;
//-->
</script>
```

JavaScript is **untyped** language. This means that a JavaScript variable can hold a value of any data type. Unlike many other languages, we don't have to tell JavaScript during variable declaration what type of value the variable will hold. The value type of a variable can change during the execution of a program and JavaScript takes care of it automatically.

JavaScript Variable Scope

JavaScript uses *lexical scoping*. Variables declared outside of a function are global variables and are visible everywhere in a JavaScript program. Variables declared inside a function have function scope and are visible only to code that appears inside that function.

JavaScript variables have only two scopes.

- **Global Variables** – A global variable has global scope which means it can be defined anywhere in the JavaScript code.
- **Local Variables** – A local variable will be visible only within a function where it is defined. Function parameters are always local to that function.

Within the body of a function, a local variable takes precedence over a global variable with the same name. If we declare a local variable or function parameter with the same name as a global variable, we effectively hide the global variable.

Example 4.10

```
<html>
<body onload = checkscope();>

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

</html>
```

JavaScript reserves a number of identifiers as the keywords of the language itself. We cannot use these words as identifiers in our programs.

Notes

A list of reserved words in JavaScript is given in the following table.

abstract	else	instanceof	switch
boolean	enum	int	synchronized
break	export	interface	this
byte	extends	long	throw
case	false	native	throws
catch	final	new	transient
char	finally	null	true
class	float	package	try
const	for	private	typeof
continue	function	protected	var
debugger	goto	public	void
default	if	return	volatile
delete	implements	short	while
do	import	static	with
double	in	super	

4.5 OPERATORS

JavaScript supports the following types of operators.

- Arithmetic Operators
- Comparison Operators
- Logical (or Relational) Operators
- Assignment Operators
- Conditional (or ternary) Operators

Example 4.11 – 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>

```

Output

```

a + b = 43
a - b = 23
a / b = 3.3
a % b = 3
a + b + c = 43Test
++a = 35
--b = 8
Set the variables to different values and then try...

```

Example 4.12 – Comparison Operators

Notes

```

<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(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) => ");
    result = (a >= b);
    document.write(result);
    document.write(linebreak);

    document.write("(a <= b) => ");
    result = (a <= b);
    document.write(result);
    document.write(linebreak);
//-->
</script>
    Set the variables to different values and
    different operators and then try...
</body>
</html>

```

Output

```

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

```

```
(a != b) => true
(a >= b) => false
a <= b) => true
Set the variables to different values and different
operators and then try...
```

Notes

Example 4.13 – Logical Operators

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

Output

```
(a && b) => false
(a || b) => true
!(a && b) => true
Set the variables to different values and different
operators and then try...
```

Example 4.14 – Bitwise Operators

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

Notes

```

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

```

Output

```

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

```

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

Example 4.15 – Assignment Operators

```

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

```

Notes

Notes

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

Example 4.16 – Conditional Operator

```
<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>
```

Output

((a > b) ? 100 : 200) => 200
 ((a < b) ? 100 : 200) => 100
 Set the variables to different values and different operators and then try...

Example 4.17 – typeof Operator

```
<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");
    document.write("Result => ");
    document.write(result);
    document.write(linebreak);
//-->
</script>
<p>Set the variables to different values and
different operators and then try...</p>
</body>
</html>

```

Notes

Output

```

Result => B is String
Result => A is Numeric
Set the variables to different values and different
operators and then try...

```

4.6 STATEMENTS

Statements are JavaScript sentences or commands. Expressions are evaluated to produce a value, but statements are executed to make something happen. Expressions with side effects, such as assignments and function invocations, can stand alone as statements, and when used this way they are known as *expression statements*. A similar category of statements are the *declaration statements* that declare new variables and define new functions.

JavaScript programs are nothing more than a sequence of *statements* to execute. By default, the JavaScript interpreter executes these statements one after another in the order they are written. Another way to “make something happen” is to alter this default order of execution, and JavaScript has a number of statements or *control structures* that do just this:

- **Conditionals** are statements like `if` and `switch` that make the JavaScript interpreter execute or skip other statements depending on the value of an expression.

- **Loops** are statements like while and for that execute other statements repetitively.
- **Jumps** are statements like break, return, and throw that cause the interpreter to jump to another part of the program.

Expression Statements

The simplest kinds of statements in JavaScript are expressions that have side effects.

```
greeting = "Hello " + name;
i *= 3;
alert(greeting);
window.close();
```

Statement block

A statement block is simply a sequence of statements enclosed within curly braces.

```
{
x = Math.PI;
cx = Math.cos(x);
console.log("cos(  $\pi$  ) = " + cx);
}
```

CONDITIONALS

A. If statement

The `if` statement is the fundamental control statement that allows JavaScript to make decisions, or, more precisely, to execute statements conditionally. This statement has two forms. The first is:

```
if (expression)
statement
```

In this form, `expression` is evaluated. If the resulting value is true, `statement` is executed. If `expression` is false, `statement` is not executed.

```
if (username == null)
// If username is null or undefined,
username = "John Doe"; // define it
```

The second form of the `if` statement introduces an `else` clause that is executed when `expression` is false. Its syntax is:

```
if (expression)
statement1
else
statement2
```

This form of the statement executes `statement1` if `expression` is true and executes `statement2` if `expression` is false. For example:

```

if (n == 1)
  console.log("You have 1 new message.");
else
  console.log("You have " + n + " new messages.");

```

Notes

`else if` is not really a JavaScript statement, but simply a frequently used programming idiom that results when repeated `if/else` statements are used:

```

if (n == 1) {
  // Execute code block #1
}
else if (n == 2) {
  // Execute code block #2
}
else if (n == 3) {
  // Execute code block #3
}
else {
  // If all else fails, execute block #4
}

```

B. Switch statement

An `if` statement causes a branch in the flow of a program's execution, and you can use the `else if` idiom to perform a multiway branch. This is not the best solution, however, when all of the branches depend on the value of the same expression. In this case, it is wasteful to repeatedly evaluate that expression in multiple `if` statements. The `switch` statement handles exactly this situation. The `switch` keyword is followed by an expression in parentheses and a block of code in curly braces:

```

switch(expression) {
  statements
}

```

For example,

```

switch(n) {
  case 1: // Start here if n == 1
    // Execute code block #1.
    break;
    // Stop here
  case 2: // Start here if n == 2
    // Execute code block #2.
    break; // Stop here

  case 3: // Start here if n == 3
    // Execute code block #3.
    break; // Stop here
}

```

Notes

```
default: // If all else fails...
// Execute code block #4.
break; // stop here
}
```

Note the `break` keyword used at the end of each case in the code above. The `break` statement, causes the interpreter to jump to the end (or “break out”) of the `switch` statement and continue with the statement that follows it. In the absence of `break` statements, a `switch` statement begins executing its block of code at the case label that matches the value of its `expression` and continues executing statements until it reaches the end of the block.

LOOPS

A. while

Just as the `if` statement is JavaScript’s basic conditional, the `while` statement is JavaScript’s basic loop. It has the following syntax:

```
while (expression)
statement
```

To execute a `while` statement, the interpreter first evaluates `expression`. If the value of the `expression` is false, then the interpreter skips over the `statement` that serves as the loop body and moves on to the next statement in the program. If, on the other hand, the `expression` is true, the interpreter executes the `statement` and repeats, jumping back to the top of the loop and evaluating `expression` again.

```
var count = 0;
while (count < 10)
{
console.log(count);
count++;
}
```

B. do/ while

The `do/while` loop is like a `while` loop, except that the loop `expression` is tested at the bottom of the loop rather than at the top. This means that the *body of the loop is always executed at least once*. The syntax is:

```
do
statement
while (expression);
```

The `do/while` loop is less commonly used than `while` in practice, it is somewhat uncommon to be certain that you want a loop to execute at least once.

```
function printArray(a)
{
  var len = a.length, i = 0;
  if (len == 0)
    console.log("Empty Array");
  else {
    do {
      console.log(a[i]);
    } while (++i < len);
  }
}
```

C. for

The `for` statement provides a looping construct that is often more convenient than the `while` statement. The `for` statement simplifies loops that follow a common pattern. Most loops have a counter variable of some kind. This variable is initialized before the loop starts and is tested before each iteration of the loop. Finally, the counter variable is incremented or otherwise updated at the end of the loop body, just before the variable is tested again. In this kind of loop, the initialization, the test, and the update are the three crucial manipulations of a loop variable. The `for` statement encodes each of these three manipulations as an expression and makes those expressions an explicit part of the loop syntax:

```
for(initialize ; test ; increment)
statement
```

`initialize`, `test`, and `increment` are three expressions (separated by semicolons) that are responsible for initializing, testing, and incrementing the loop variable. Putting them all in the first line of the loop makes it easy to understand what a `for` loop is doing and prevents mistakes such as forgetting to initialize or increment the loop variable.

```
var i,j;
for(i = 0, j = 10 ; i < 10 ; i++, j--)
sum += i * j;
```

D. for/in

The `for/in` statement uses the `for` keyword, but it is a completely different kind of loop than the regular `for` loop. A `for/in` loop looks like this:

```
for (variable in object)
statement
```

For example,

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

```
// Assign array indexes to variable i
console.log(a[i]);
// Print the value of each array element
```

JUMPS

Another category of JavaScript statements are *jump statements*. As the name implies, these cause the JavaScript interpreter to jump to a new location in the source code. The `break` statement makes the interpreter jump to the end of a loop or other statement. `continue` makes the interpreter skip the rest of the body of a loop and jump back to the top of a loop to begin a new iteration.

```
for(var i = 0; i < a.length; i++)
{
    if (a[i] == target) break;
}
```

JavaScript allows statements to be named, or `labeled`, and the `break` and `continue` can identify the target loop or other statement label.

Any statement may be `labeled` by preceding it with an identifier and a colon:

```
identifier: statement
```

For example,

```
var matrix = getData();
// Get a 2D array of numbers from somewhere
// Now sum all the numbers in the matrix.
var sum = 0, success = false;
// Start with a labeled statement that we can
//break out of if errors occur

compute_sum: if (matrix)
{
    for(var x = 0; x < matrix.length; x++)
    {
        var row = matrix[x];
        if (!row) break compute_sum;
        for(var y = 0; y < row.length; y++)
        {
            var cell = row[y];
            if (isNaN(cell)) break compute_sum;
            sum += cell;
        }
    }
    success = true;
}
```



```
// The break statements jump here.  
// If we arrive here with success == false  
// then there was something wrong with the matrix  
// we were given.  
// Otherwise sum contains the sum of all cells of  
// the matrix.
```

4.7 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 number 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;
```

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 function and a method – 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.

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");
```

User-Defined Objects

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

A. The new Operator

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 employee = new Object();
var books = new Array("C++", "Perl", "Java");
var day = new Date("August 15, 1947");
```

B. The Object() Constructor

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 properties assigned to the object are not variables and are not defined with the **var** keyword.

```
<html>
<head>
<title>User-defined objects</title>
<script type = "text/javascript">
    var book = new Object();
    // Create the 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>

```

JavaScript Native Objects

JavaScript has several built-in or native objects. These objects are accessible anywhere in your program and will work the same way in any browser running in any operating system.

Here is the list of all important JavaScript Native Objects

- JavaScript Number Object
- JavaScript Boolean Object
- JavaScript String Object
- JavaScript Array Object
- JavaScript Date Object
- JavaScript Math Object
- JavaScript RegExp Object

Check Your Progress 1

1. List the features of DHTML
2. What about the complexity of dynamic webpages?
3. What are advantages of JavaScript?
4. What do you mean by an object?
5. What is a variable?
6. What is the difference between while and do loop?

4.8 MATHEMATICAL FUNCTIONS

The **math** object provides the properties and methods for mathematical constants and functions. Unlike other global objects, **Math** is not a constructor. All the properties and methods of **Math** are static and can be called by using Math as an object without creating it.

Thus, the constant **pi** is referred as **Math.PI** and the function *sine* is called as **Math.sin(x)**, where x is the method's argument.

The syntax to call the properties and methods of Math are as follows

```

var pi_val = Math.PI;
var sine_val = Math.sin(30);

```

Here is a list of all the properties of Math and their description.

S.No.	Property	Description
1	E \	Euler's constant and the base of natural

		logarithms, approximately 2.718.
2	LN2	Natural logarithm of 2, approximately 0.693.
3	LN10	Natural logarithm of 10, approximately 2.302.
4	LOG2E	Base 2 logarithm of E, approximately 1.442.
5	LOG10E	Base 10 logarithm of E, approximately 0.434.
6	PI	Ratio of the circumference of a circle to its diameter, approximately 3.14159.
7	SQRT1_2	Square root of 1/2; equivalently, 1 over the square root of 2, approximately 0.707.
8	SQRT2	Square root of 2, approximately 1.414.

Here is a list of the methods associated with Math object and their description

S. No.	Method	Description
1	abs()	Returns the absolute value of a number.
2	acos()	Returns the arccosine (in radians) of a number.
3	asin()	Returns the arcsine (in radians) of a number.
4	atan()	Returns the arctangent (in radians) of a number.
5	atan2()	Returns the arctangent of the quotient of its arguments
6	ceil()	Returns the smallest integer greater than or equal to a number.
7	cos()	Returns the cosine of a number.
8	exp()	Returns E^N , where N is the argument, and E is Euler's constant, the base of the natural logarithm.
9	floor()	Returns the largest integer less than or equal to a number.
10	log()	Returns the natural logarithm (base E) of a number.
11	max()	Returns the largest of zero or more numbers.
12	min()	Returns the smallest of zero or more numbers.
13	pow()	Returns base to the exponent power, that is, base exponent.
14	random()	Returns a pseudo-random number between 0 and 1.
15	round()	Returns the value of a number rounded to the nearest integer.
16	sin()	Returns the sine of a number.
17	sqrt()	Returns the square root of a number.
18	tan()	Returns the tangent of a number.
19	toSource()	Returns the string "Math".

Example 4.18

```
<html>
```

```
<head>
<title>JavaScript Math Methods</title>
</head>
<body>
<script type = "text/javascript">
    var value = Math.min(10, 20, -1, 100);
    document.write("First Test Value : " +
value );

    var value = Math.pow(8, 8);
    document.write("<br />Second Test Value :
" + value );

    var value = Math.random( );
    document.write("<br />Third Test Value :
" + value );

    var value = Math.round( -20.3 );
    document.write("<br />Fourth Test Value :
" + value );

    var value = Math.floor(10.3);
    document.write("Fifth Test Value : " +
value );

    var value = Math.toSource( );
    document.write("Value : " + value );

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

Notes

4.9 STRING MANIPULATORS

Strings are used to represent text. They are written by enclosing their content in quotes.

```
`Down on the sea`
"Lie on the ocean"
'Float on the ocean'
```

String object can be created using the following syntax

```
var val = new String(string);
```

Here is a list of the properties of String object and their description.

S.No.	Property	Description
1	constructor	Returns a reference to the String function that created the object.
2	length	Returns the length of the string.
3	prototype	The prototype property allows you to add properties and methods to an object.

Here is a list of the methods available in String object along with their description.

S.No.	Method	Description
1	charAt()	Returns the character at the specified index.
2	charCodeAt()	Returns a number indicating the Unicode value of the character at the given index.
3	concat()	Combines the text of two strings and returns a new string.
4	indexOf()	Returns the index within the calling String object of the first occurrence of the specified value, or -1 if not found.
5	lastIndexOf()	Returns the index within the calling String object of the last occurrence of the specified value, or -1 if not found.
6	localeCompare()	Returns a number indicating whether a reference string comes before or after or is the same as the given string in sort order.
7	match()	Used to match a regular expression against a string.
8	replace()	Used to find a match between a regular expression and a string, and to replace the matched substring with a new substring.
9	search()	Executes the search for a match between a regular expression and a specified string.
10	slice()	Extracts a section of a string and returns a new string.
11	split()	Splits a String object into an array of strings by separating the string into substrings.
12	substr()	Returns the characters in a string beginning at the specified location through the specified number of characters.
13	substring()	Returns the characters in a string between two indexes into the string.
14	toLocaleLowerCase()	The characters within a string are converted to lower case while respecting the current locale.

15	<code>toLocaleUpperCase()</code>	The characters within a string are converted to upper case while respecting the current locale.
16	<code>toLowerCase()</code>	Returns the calling string value converted to lower case.
17	<code>toString()</code>	Returns a string representing the specified object. <i>Java Script</i>
18	<code>toUpperCase()</code>	Returns the calling string value converted to uppercase.
19	<code>valueOf()</code>	Returns the primitive value of the specified object.

Notes

Here is a list of the methods that return a copy of the string wrapped inside an appropriate HTML tag.

S.No.	Method	Description
1	<code>anchor()</code>	Creates an HTML anchor that is used as a hypertext target.
2	<code>big()</code>	Creates a string to be displayed in a big font as if it were in a <code><big></code> tag.
3	<code>blink()</code>	Creates a string to blink as if it were in a <code><blink></code> tag.
4	<code>bold()</code>	Creates a string to be displayed as bold as if it were in a <code></code> tag.
5	<code>fixed()</code>	Causes a string to be displayed in fixed-pitch font as if it were in a <code><tt></code> tag
6	<code>fontcolor()</code>	Causes a string to be displayed in the specified color as if it were in a <code></code> tag.
7	<code>fontsize()</code>	Causes a string to be displayed in the specified font size as if it were in a <code></code> tag.
8	<code>italics()</code>	Causes a string to be italic, as if it were in an <code><i></code> tag.
9	<code>link()</code>	Creates an HTML hypertext link that requests another URL.
10	<code>small()</code>	Causes a string to be displayed in a small font, as if it were in a <code><small></code> tag.
11	<code>strike()</code>	Causes a string to be displayed as struck-out text, as if it were in a <code><strike></code> tag.
12	<code>sub()</code>	Causes a string to be displayed as a subscript, as if it were in a <code><sub></code> tag
13	<code>sup()</code>	Causes a string to be displayed as a superscript, as if it were in a <code><sup></code> tag

Example 4.18

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

```
<h2>JavaScript String Methods</h2>

<p>The slice() method extract a part of a string
and returns the extracted parts in a new
string:</p>

<p id="demo"></p>

<p>The substr() method extract a part of a string
and returns the extracted parts in a new
string:</p>

<p id="demo1"></p>

<p>The search() method returns the position of the
first occurrence of a specified text in a
string:</p>

<p id="demo3"></p>

<p>Replace "Microsoft" with "W3Schools" in the
paragraph below:</p>

<button onclick="myFunction()">Try it</button>

<p id="demo2">Please visit Microsoft!</p>

<script>
var str = "Apple, Banana, Kiwi";

var res = str.slice(-12,-6);
document.getElementById("demo").innerHTML = res;

var res = str.substr(7);
document.getElementById("demo1").innerHTML = res;

var str = "Please locate where 'locate' occurs!";
var pos = str.search("locate");
document.getElementById("demo3").innerHTML = pos;

function myFunction() {
  var str =
document.getElementById("demo").innerHTML;
  var txt = str.replace("Microsoft","W3Schools");
  document.getElementById("demo2").innerHTML =
txt;
}
</script>
</body>
</html>
```


Output

JavaScript String Methods

The slice() method extract a part of a string and returns the extracted parts in a new string:
Banana

The substr() method extract a part of a string and returns the extracted parts in a new string:
Banana, Kiwi

The search() method returns the position of the first occurrence of a specified text in a string:
7

Replace "Microsoft" with "W3Schools" in the paragraph below:

Please visit Microsoft!

Notes

4.10 ARRAYS

An **array** is an ordered collection of values. Each value is called an **element**, and each element has a numeric position in the array, known as its **index**.

- ✓ JavaScript arrays are **untyped**:
An array element may be of any type, and different elements of the same array may be of different types. Array elements may even be objects or other arrays, which allows you to create complex data structures, such as arrays of objects and arrays of arrays.
- ✓ JavaScript arrays are **zero-based** and use 32-bit indexes:
The index of the first element is 0, and the highest possible index is 4294967294 (2³²−2), for a maximum array size of 4,294,967,295 elements.
- ✓ JavaScript arrays are **dynamic**:
They grow or shrink as needed and there is no need to declare a fixed size for the array when you create it or to reallocate it when the size changes.
- ✓ JavaScript arrays may be **sparse**:
The elements need not have contiguous indexes and there may be gaps. Every JavaScript array has a length property. For non-sparse arrays, this property specifies the number of elements in the array. For sparse arrays, length is larger than the index of all elements.

Creating Arrays

The easiest way to create an array is with an array literal, which is simply a comma-separated list of array elements within square brackets.

```
var empty = []; // An array with no elements
var primes = [2, 3, 5, 7, 11];
// An array with 5 numeric elements
var misc = [1.1, true, "a", ];
// 3 elements of various types + trailing comma
```

The values in an array literal need not be constants; they may be arbitrary expressions:

```
var base = 1024;
var table = [base, base+1, base+2, base+3];
```

Array literals can contain object literals or other array literals:

```
var b = [[1, {x:1, y:2}], [2, {x:3, y:4}]];
```

If you omit a value from an array literal, the omitted element is given the value `undefined`:

```
var count = [1,,3];
// An array with 3 elements,
// the middle one undefined.
var undefs = [,];
// An array with 2 elements, both undefined.
```

Array literal syntax allows an optional trailing comma, so `[,,]` has only two elements, not three.

Another way to create an array is with the `Array()` constructor. You can invoke this constructor in three distinct ways:

- Call it with no arguments:

```
var a = new Array();
```

This method creates an empty array with no elements and is equivalent to the array literal `[]`.

- Call it with a single numeric argument, which specifies a length:

```
var a = new Array(10);
```

This technique creates an array with the specified length. This form of the `Array()` constructor can be used to pre-allocate an array when you know in advance how many elements will be required. Note that no values are stored in the array, and the array index properties "0", "1", and so on are not even defined for the array.

- Explicitly specify two or more array elements or a single non-numeric element for the array:

```
var a = new Array(5, 4, 3, 2, 1, "testing, testing");
```

In this form, the constructor arguments become the elements of the new array. Using an array literal is almost always simpler than this usage of the `Array()` constructor.

Reading and Writing Array Elements

You access an element of an array using the `[]` operator. A reference to the array should appear to the left of the brackets. An arbitrary expression that has a non-negative integer value should be inside the brackets. You can use this syntax to both read and write the value of an element of an array. Thus, the following are all legal JavaScript statements:

```
var a = ["world"];
// Start with a one-element array

var value = a[0]; // Read element 0

a[1] = 3.14; // Write element 1

i = 2;
a[i] = 3; // Write element 2

a[i + 1] = "hello"; // Write element 3

a[a[i]] = a[0];
// Read elements 0 and 2, write element 3
```

Sparse Arrays

A sparse array is one in which the elements do not have contiguous indexes starting at 0. Normally, the length property of an array specifies the number of elements in the array. If the array is sparse, the value of the length property is greater than the number of elements. Sparse arrays can be created with the `Array()` constructor or simply by assigning to an array index larger than the current array length.

```
a = new Array(5);
// No elements, but a.length is 5.

a = [];
// Create an array with no elements & length = 0.

a[1000] = 0;
// Assignment adds one element but
// sets length to 1001.
```

The second special behaviour that arrays implement in order to maintain the length invariant is that if you set the length property to a non-negative integer `n` smaller than its current value, any array elements whose index is greater than or equal to `n` are deleted from the array:

```
a = [1,2,3,4,5];
// Start with a 5-element array.
```

Notes

```

a.length = 3;
// a is now [1,2,3].
a.length = 0;
// Delete all elements. a is [].

a.length = 5;
// Length is 5, but no elements, like new Array(5)
    
```

You can also set the length property of an array to a value larger than its current value. Doing this does not actually add any new elements to the array; it simply creates a sparse area at the end of the array.

Here is a list of the properties of the Array object along with their description.

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 expression matches.
4	length	Reflects the number of elements in an array.
5	prototype	The prototype property allows you to add properties and methods to an object.

In the following sections, we will have a few examples to illustrate the usage of Array properties.

Here is a 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 provided testing function.
3	filter()	Creates a new array with all of the elements of this array for which the provided filtering function returns true.
4	forEach()	Calls a function for each element in the array.
5	indexOf()	Returns the first (least) index of an element within the array equal to the specified 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 element within the array equal to the specified value, or -1 if none is found.
8	map()	Creates a new array with the results of calling a provided function on every

		element in this 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 array 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 array -- 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.
16	some()	Returns true if at least 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.

Example 4.19

```

<html>
<head>
  <title>Arrays!!!</title>
  <script type="text/javascript">
    var students = new Array("John", "Ann",
"Aaron", "Edwin", "Elizabeth");

    Array.prototype.displayItems=function()
    {
      for (i=0;i<this.length;i++){
        document.write(this[i] + "<br
/>");
      }
    }
    document.write("Students Array Example
<br />");

    students.displayItems();
  </script>
</head>
</html>

```

```
        document.write("<br />The number of items
in students array is " + students.length + "<br
/>");
        document.write("<br />The SORTED students
array<br />");

        students.sort();
        students.displayItems();

        document.write("<br />The REVERSED
students array<br />");

        students.reverse();
        students.displayItems();

        document.write("<br />THE students array
after REMOVING the LAST item<br />");
        students.pop();
        students.displayItems();

        document.write("<br />THE students array
after PUSH<br />");
        students.push("New Stuff");
        students.displayItems();
    </script>

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

Output

Students Array Example

John
Ann
Aaron
Edwin
Elizabeth

The number of items in students array is 5

The SORTED students array

Aaron
Ann
Edwin
Elizabeth
John

The REVERSED students array

John
Elizabeth

Edwin
Ann
Aaron

THE students array after REMOVING the LAST item

John
Elizabeth
Edwin
Ann

THE students array after PUSH

John
Elizabeth
Edwin
Ann
New Stuff

4.11 FUNCTIONS

A *function* is a block of JavaScript code that is defined once but may be executed, or invoked, any number of times. You may already be familiar with the concept of a function under a name such as subroutine or procedure.

JavaScript functions are parameterized: a function definition may include a list of identifiers, known as *parameters* that work as local variables for the body of the function. Function invocations provide values, or *arguments*, for the function's parameters. Functions often use their argument values to compute a return value that becomes the value of the function invocation expression. In addition to the arguments, each invocation has another value—the *invocation context*—that is the value of the `this` keyword.

If a function is assigned to the property of an object, it is known as a *method* of that object. When a function is invoked on or *through* an object, that object is the invocation context or `this` value for the function. Functions designed to initialize a newly created object are called *constructors*.

Defining Functions

Functions are defined with the `function` keyword, which can be used in a function definition expression or in a function declaration statement. In either form, function definitions begin with the keyword `function` followed by these components:

- An identifier that names the function. The name is a required part of function declaration statements: it is used as the name of a variable, and the newly defined function object is assigned to the variable. For function definition expressions, the name is optional: if present, the name refers to the function object only within the body of the function itself.

- A pair of parentheses around a comma-separated list of zero or more identifiers. These identifiers are the parameter names for the function, and they behave like local variables within the body of the function.
- A pair of curly braces with zero or more JavaScript statements inside. These statements are the body of the function: they are executed whenever the function is invoked.

```
// Print the name and value of each property of o.
// Return undefined.

function printprops(o)
{
  for(var p in o)
  console.log(p + ": " + o[p] + "\n");
}

// Compute the distance between Cartesian points
//(x1,y1) and (x2,y2).

function distance(x1, y1, x2, y2)
{
  var dx = x2 - x1;
  var dy = y2 - y1;
  return Math.sqrt(dx*dx + dy*dy);
}

// A recursive function (one that calls itself)
// that computes factorials
// Recall that x! is the product of x and all
// positive integers less than it.

function factorial(x)
{
  if (x <= 1) return 1;
  return x * factorial(x-1);
}

// This function expression defines a function
// that squares its argument.
// Note that we assign it to a variable

var square = function(x) { return x*x; }

// Function expressions can include names, which
// is useful for recursion.

var f = function fact(x)
{
  if (x <= 1)
  return 1;
```



```

else
  return x*fact(x-1);
};
// Function expressions can also be used as
// arguments to other functions:

data.sort(function(a,b) { return a-b; });

// Function expressions are sometimes defined and
// immediately invoked:

var tensquared = (function(x) {return x*x;}(10));

```

Note that the function name is optional for functions defined as expressions. A function declaration statement actually declares a variable and assigns a function object to it. A function definition expression, on the other hand, does not declare a variable. A name is allowed for functions, like the factorial function above, that need to refer to themselves. If a function definition expression includes a name, the local function scope for that function will include a binding of that name to the function object. In effect, the function name becomes a local variable within the function. Most functions defined as expressions do not need names, which makes their definition more compact.

Nested Functions

In JavaScript, functions may be nested within other functions. For example:

```

function hypotenuse(a, b)
{
  function square(x)
  { return x*x; }

  return Math.sqrt(square(a) + square(b));
}

```

The interesting thing about nested functions is their variable scoping rules: they can access the parameters and variables of the function (or functions) they are nested within. In the code above, for example, the inner function `square()` can read and write the parameters `a` and `b` defined by the outer function `hypotenuse()`.

Invoking Functions

The JavaScript code that makes up the body of a function is not executed when the function is defined but when it is invoked. JavaScript functions can be invoked in four ways:

- as functions,

- as methods,
- as constructors, and
- Indirectly through their `call()` and `apply()` methods.

The following code includes a number of regular *function invocation* expressions:

```
printprops({x:1});
var total = distance(0,0,2,1) + distance(2,1,3,5);
var probability = factorial(5)/factorial(13);
```

In an invocation, each argument expression (the ones between the parentheses) is evaluated, and the resulting values become the arguments to the function. These values are assigned to the parameters named in the function definition. In the body of the function, a reference to a parameter evaluates to the corresponding argument value.

A *method* is nothing more than a JavaScript function that is stored in a property of an object. If you have a function `f` and an object `o`, you can define a method named `m` of `o` with the following line:

```
o.m = f;
```

Having defined the method `m()` of the object `o`, invoke it like this:

```
o.m();
```

Or, if `m()` expects two arguments, you might invoke it like this:

```
o.m(x, y);
```

If a function or method invocation is preceded by the keyword `new`, then it is a *constructor invocation*. Constructor invocations differ from regular function and method invocations in their handling of arguments, invocation context, and return value.

If a constructor invocation includes an argument list in parentheses, those argument expressions are evaluated and passed to the function in the same way they would be for function and method invocations. But if a constructor has no parameters, then JavaScript constructor invocation syntax allows the argument list and parentheses to be omitted entirely. You can always omit a pair of empty parentheses in a constructor invocation and the following two lines, for example, are equivalent:

```
var o = new Object();
var o = new Object;
```

Constructor functions do not normally use the `return` keyword. They typically initialize the new object and then return implicitly when they reach the end of their body.

JavaScript functions are objects and like all JavaScript objects, they have methods. Two of these methods, `call()` and `apply()`, *invoke the*

function indirectly. Both methods allow you to explicitly specify the `this` value for the invocation, which means you can invoke any function as a method of any object, even if it is not actually a method of that object. Both methods also allow you to specify the arguments for the invocation. The `call()` method uses its own argument list as arguments to the function and the `apply()` method expects an array of values to be used as arguments.

To invoke the function `f()` as a method of the object `o` (passing no arguments), you could use either `call()` or `apply()`:

```
f.call(o);
f.apply(o);
```

Example 4.20

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

<h2>JavaScript Functions</h2>

<p>This example calls a function to convert from
Fahrenheit to Celsius:</p>
<p id="demo"></p>

<script>
function toCelsius(f) {
  return (5/9) * (f-32);
}
document.getElementById("demo").innerHTML =
toCelsius(77);
</script>

</body>
</html>
```

Output

JavaScript Functions

This example calls a function to convert from Fahrenheit to Celsius:

25

Example 4.21

```
<html>
<head>
<script type = "text/javascript">
  function sayHello(name, age)
```

```
        {
            document.write (name + " is " + age +
" years old.");
        }
</script>
</head>

<body>
<p>Click the following button to call the
function</p>

<form>
<input type = "button" onclick = "sayHello('Zara',
7)" value = "Say Hello">
</form>
<p>Use different parameters inside the function
and then try...</p>

</body>
</html>
```

Output

Click the following button to call the function

Say Hello

Use different parameters inside the function and then try...

Zara is 7 years old.

Check Your Progress 2

1. What is an array?
2. List the characteristics of JavaScript arrays.
3. What is the purpose of **length** method?
4. What do you mean by function?
5. How will you invoke functions?
6. What is the purpose of constructors?

4.12 ANSWERS TO CHECK YOUR PROGRESS

1. Features of DHTML are
 - i. Dynamic content
 - ii. Dynamic style
 - iii. Dynamic positioning
 - iv. Dynamic binding

2. Client side scripting involves medium complexity. Server side scripting involves high complexity of two parts
3. Advantages of JavaScript are
 - Less server interaction
 - Immediate feedback to visitors
 - Increased interactivity
 - Richer interfaces
4. An object (that is, a member of the type *object*) is a collection of *properties* where each property has a name and a value (either a primitive value, such as a number or string, or an object).
5. A *variable* defines a symbolic name for a value and allows the value to be referred to by name
6. The body of the loop is always executed at least once for `do` loop. The body of the `while` loop will be executed only if the condition is satisfied.
7. An *array* is an ordered collection of values. Each value is called an *element*, and each element has a numeric position in the array, known as its *index*
8. Characteristics of JavaScript arrays are
 - untyped
 - zero based
 - dynamic
 - sparse
9. The `length` method is used to reflect the number of elements in an array.
10. A *function* is a block of JavaScript code that is defined once but may be executed, or invoked, any number of times.
11. JavaScript functions can be invoked in four ways:
 - i. as functions,
 - ii. as methods,
 - iii. as constructors, and
 - iv. Indirectly through their `call()` and `apply()` methods
12. Constructor functions do not normally use the `return` keyword. They typically initialize the new object and then return implicitly when they reach the end of their body

4.13 LET US SUM UP

DHTML, refers to the technique of making Web pages dynamic by client-side scripting to manipulate the document content and presentation

The characteristics of JavaScript are

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

JavaScript programs are written using the Unicode character set

JavaScript ignores spaces, tabs, and newlines that appear in JavaScript programs.

Simple statements in JavaScript are generally followed by a semicolon character, just as they are in C, C++, and Java. JavaScript, however, allows us to omit this semicolon if each of our statements is placed on a separate line.

JavaScript is a case-sensitive language.

JavaScript is object-oriented.

The JavaScript interpreter performs automatic garbage collection for memory management.

An *identifier* is simply a name. In JavaScript, identifiers are used to name variables and functions and to provide labels for certain loops in JavaScript code.

A JavaScript identifier must begin with

- a letter,
- an underscore (`_`), or
- a dollar sign (`$`).

JavaScript types can be divided into two categories: *primitive types* and *object types*.

An object (that is, a member of the type *object*) is a collection of *properties* where each property has a name and a value (either a primitive value, such as a number or string, or an object).

JavaScript types can also be categorized as *mutable* and *immutable* types

JavaScript variables have only two scopes.

- Global Variables – A global variable has global scope which means it can be defined anywhere in the JavaScript code.
- Local Variables – A local variable will be visible only within a function where it is defined. Function parameters are always local to that function

JavaScript supports the following types of operators.

- Arithmetic Operators
- Comparison Operators
- Logical (or Relational) Operators
- Assignment Operators
- Conditional (or ternary) Operators

Conditionals are statements like if and switch that makes the JavaScript interpreter execute or skip other statements depending on the value of an expression.

Loops are statements like while and for that execute other statements repetitively.

Jumps are statements like break, return, and throw that cause the interpreter to jump to another part of the program.

JavaScript has several built-in or native objects. Here is the list of all important JavaScript Native Objects

- JavaScript Number Object
- JavaScript Boolean Object
- JavaScript String Object
- JavaScript Array Object
- JavaScript Date Object
- JavaScript Math Object
- JavaScript RegExp Object

The **math** object provides the properties and methods for mathematical constants and functions.

Strings are used to represent text. They are written by enclosing their content in quotes.

An **array** is an ordered collection of values. Each value is called an **element**, and each element has a numeric position in the array, known as its **index**

A sparse array is one in which the elements do not have contiguous indexes starting at 0.

A **function** is a block of JavaScript code that is defined once but may be executed, or invoked, any number of times.

JavaScript functions are parameterized: a function definition may include a list of identifiers, known as **parameters** that work as local variables for the body of the function

A **method** is nothing more than a JavaScript function that is stored in a property of an object.

Constructor functions do not normally use the return keyword. They typically initialize the new object and then return implicitly when they reach the end of their body

4.14 SELF-ASSESSMENT EXERCISES

Short Questions

1. What do you mean by primitive data types?
2. What are object types?
3. List any two mathematical functions
4. What are strings?
5. How to define scope of variables in JavaScript?
6. Describe any two methods for arrays.
7. How to define functions in JavaScript?

Detail Questions

1. Write a note on variables.
2. Discuss about the JavaScript Statements.
3. Explain in detail about loops.
4. Write a program to demonstrate the working of string functions
5. Explain the mathematical functions available in JavaScript.
6. Discuss in detail about arrays in JavaScript.
7. Describe about writing user defined functions in JavaScript.

4.15 SUGGESTED READINGS

1. Web Programming: Building Internet applications, Chris Bates, Wiley India
2. Web technologies – A computer science perspective, Jeffrey C Jackson, Pearson Education, 2006
3. Robert W. Sebesta “Programming the world wide web” Pearson Education
4. Bates, Developing web applications, wiley, 2006
5. www.w3schools.com
6. JavaScript: The Definitive Guide, 6th Edition, David Flanagan, O’Reilly Media, 2011.
7. Eloquent JavaScript, 3rd edition, Marijn Haverbeke, 2018

UNIT- 5 COOKIES AND EVENTS

Structure

- 5.0 Introduction
- 5.1 Objectives
- 5.2 Regular Expressions
- 5.3 Cookies
- 5.4 Events
- 5.5 Answers to Check Your Progress
- 5.6 Let us Sum up
- 5.7 Self-Assessment Exercises
- 5.8 Suggested Readings

Notes

5.0 INTRODUCTION

The communication between a web browser and server happens using a stateless protocol named HTTP. The inputs from the users need to be validated before sending the request. A regular expression is an object that describes a pattern of characters. This will be used to validate the inputs. Also Stateless protocol treats each request independent. So, the server does not keep the data after sending it to the browser. But in many situations, the data will be required again. Here come cookies into a picture. JavaScript's interaction with HTML is handled through events that occur when the user or the browser manipulates a page. This unit will describe about the regular expression, cookies and events.

5.1 OBJECTIVES

After going through this unit, you will be able to:

- Learn the need for regular expressions
- Understand how to handle cookies using JavaScript
- Know the basics of event handling in JavaScript

5.2 REGULAR EXPRESSIONS

Regular expressions are a way to describe patterns in string data. They form a small, separate language that is part of JavaScript and many other languages and systems. Regular expressions are both terribly awkward and extremely useful.

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

The JavaScript `RegExp` class represents regular expressions, and both `String` and `RegExp` 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 `RegExp()` constructor, as follows

```
var pattern = new RegExp(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.

```
let re1 = new RegExp("abc");
let re2 = /abc/;
```

Brackets

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

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

The ranges `[0-3]` is used to match any decimal digit ranging from 0 through 3, or the range `[b-v]` is used to match any lowercase character ranging from b through v.

Quantifiers

The frequency or position of bracketed character sequences and single characters can be denoted by a special character. Each special character has a specific connotation. The `+`, `*`, `?`, and `$` flags all follow 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 at least 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.

Notes

Examples

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, in turn followed by another p .
3	^.{2}\$	It matches any string containing exactly two characters.
4	(.*	It matches any string enclosed within and .
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 NUL character (\u0000)
3	\t	Tab (\u0009)
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 hexadecimal number nn; for example, \x0A is the same as \n
9	\uxxxx	The Unicode character specified by the hexadecimal number xxxx; for example, \u0009 is the same as \t
10	\cX	The control character ^X; for example, \cJ is equivalent to the newline character \n

Notes

Metacharacters

A metacharacter is simply an alphabetical character preceded by a backslash that acts to give the combination a special meaning.

S. No.	Character	Description
1	.	a single character
2	\s	a whitespace character (space, tab, newline)
3	\S	non-whitespace character
4	\d	a digit (0-9)
5	\D	a non-digit
6	\w	a word character (a-z, A-Z, 0-9, _)
7	\W	a non-word character
8	[\b]	a literal backspace (special case).
9	[aeiou]	matches a single character in the given set
10	[^aeiou]	matches a single character outside the given set
11	(foo bar baz)	matches any of the alternatives specified

```

/\d{2,4}/
// Match between two and four digits

/\w{3}\d?/
// Match exactly three word characters and an
// optional digit

/\s+java\s+/
// Match "java" with one or more spaces
// before and after

/[^(]*/
// Match zero or more characters that are not
// open parenthesis
    
```

Modifiers

Several modifiers are available that can simplify the way you work with regexps, like case sensitivity, searching in multiple lines, etc.

S.No.	Modifier	Description
1	i	Perform case-insensitive matching.
2	m	Specifies that if the string has newline or carriage return characters, the ^ and \$ operators will now match against a newline boundary, instead of a string boundary
3	g	Performs a global match that is, find all matches rather than stopping after the first match.

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	ignoreCase	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.

Notes

RegExp Methods

S.No.	Method	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 5.1

```

<!DOCTYPE html>
<html>
<body>

<p>
Search for the characters<br>
"LO"
<br>in the <strong>beginning</strong> of a word in
the phrase:<br>
"HELLO, LOOK AT YOU!"
</p>

<p>Found in position: <span id="demo"></span></p>

<script>
var str = "HELLO, LOOK AT YOU!";
var patt1 = /\bLO/;
var result = str.search(patt1);
document.getElementById("demo").innerHTML =
result;
</script>

</body>
</html>

```

Notes

Output

Search for the characters

"LO"

in the **beginning** of a word in the phrase:

"HELLO, LOOK AT YOU!"

Found in position: 7

Example 5.2

```

<!DOCTYPE html>
<html>
<body>
<p>Click the button to do a global search for any
of the specified alternatives (red|green).</p>

<button onclick="myFunction()">Try it</button>

<p id="demo"></p>

<script>
function myFunction() {
  var str = "re, green, red, green, gren, gr,
blue, yellow";
  var patt1 = /(red|gr*)/g;
  var result = str.match(patt1);
  document.getElementById("demo").innerHTML =
result;
}
</script>

</body>
</html>

```

Output

Click the button to do a global search for any of the specified alternatives (red|green).

Try it

gr,red,gr,gr,gr

5.3COOKIES

The communication between a web browser and server happens using a stateless protocol named HTTP. Stateless protocol treats each request independent. So, the server does not keep the data after sending it to the browser. But in many situations, the data will be required again. Here come cookies into a picture. With cookies, the web browser will not have to communicate with the server each time the data is required. Instead, it can be fetched directly from the computer.

A *cookie* is a piece of data that is stored on our computer to be accessed by our browser.

In many situations, using cookies is the most efficient method of remembering and tracking preferences, purchases, commissions, and other information required for better visitor experience or site statistics.

Notes

The server sends some data to the visitor's browser in the form of a cookie. The browser may accept the cookie. If it does, it is stored as a plain text record on the visitor's hard drive. Now, when the visitor arrives at another page on the site, the browser sends the same cookie to the server for retrieval. Once retrieved, the server knows/remembers what was stored earlier.

Cookies are a plain text data record of 5 variable-length fields

- **Expires** – The date the cookie will expire. If this is blank, the cookie will expire when the visitor quits the browser.
- **Domain** – The domain name of your site.
- **Path** – The path to the directory or web page that set the cookie. This may be blank if you want to retrieve the cookie from any directory or page.
- **Secure** – If this field contains the word "secure", then the cookie may only be retrieved with a secure server. If this field is blank, no such restriction exists.
- **Name=Value** – Cookies are set and retrieved in the form of key-value pairs

Cookies were originally designed for CGI programming. The data contained in a cookie is automatically transmitted between the web browser and the web server, so CGI scripts on the server can read and write cookie values that are stored on the client.

JavaScript can also manipulate cookies using the `cookie` property of the `Document` object. JavaScript can read, create, modify, and delete the cookies that apply to the current web page.

Storing Cookies

The simplest way to create a cookie is to assign a string value to the `document.cookie` object

```
document.cookie = "key1 = value1;key2 =  
value2;expires = date";
```

Here the `expires` attribute is optional. If you provide this attribute with a valid date or time, then the cookie will expire on a given date or time and thereafter, the cookies' value will not be accessible.

Notes

Example 5.3 – Setting Cookies

```

<html>
<head>
<script type = "text/javascript">
<!--
function WriteCookie() {
    if( document.myform.customer.value == "" ) {
        alert("Enter some value!");
        return;
    }
    cookievalue =
escape(document.myform.customer.value) + ";";
    document.cookie = "name=" + cookievalue;
    document.write ("Setting Cookies : " + "name="
+ cookievalue );
}
//-->
</script>
</head>

<body>
<form name = "myform" action = "">
Enter name: <input type = "text" name =
"customer"/>
<input type = "button" value = "Set Cookie"
onclick = "WriteCookie();"/>
</form>
</body>
</html>

```

Output

Enter name:

Setting Cookies : name=Cookie;

Reading Cookies

Reading a cookie is just as simple as writing one, because the value of the `document.cookie` object is the cookie. So we can use this string whenever we want to access the cookie. The `document.cookie` string will keep a list of `name=value` pairs separated by semicolons, where `name` is the name of a cookie and `value` is its string value. The `split()` function is used to break a string into key and values.

Example 5.4 – Getting Cookie values

```

<html>
<head>
<script type = "text/javascript">
<!--
function ReadCookie() {
var allcookies = document.cookie;
document.write ("All Cookies : " + allcookies );

// Get all the cookies pairs in an array
cookiearray = allcookies.split(';');

// Now take key value pair out of this array
for(var i=0; i<cookiearray.length; i++) {
    name = cookiearray[i].split('=')[0];
    value = cookiearray[i].split('=')[1];
    document.write ("Key is : " + name + " and
Value is : " + value);
}
}
//-->
</script>
</head>

<body>
<form name = "myform" action = "">
<p> click the following button and see the
result:</p>
<input type = "button" value = "Get Cookie"
onclick = "ReadCookie()"/>
</form>
</body>
</html>

```

Notes

Example 5.5 – Setting Cookie expiry Date

```

<html>
<head>
<script type = "text/javascript">
<!--
function WriteCookie() {
var now = new Date();
now.setMonth( now.getMonth() + 1 );
cookievalue =
escape(document.myform.customer.value) + ";";
document.cookie = "name=" + cookievalue;

```

Notes

```

document.cookie = "expires=" + now.toUTCString() +
";"
document.write ("Setting Cookies : " + "name=" +
cookievalue );
}
//-->
</script>
</head>

<body>
<form name = "myform" action = "">
Enter name: <input type = "text" name =
"customer"/>
<input type = "button" value = "Set Cookie"
onclick = "WriteCookie()"/>
</form>
</body>
</html>

```

Example 5.6 – Deleting Cookie

```

<html>
<head>
<script type = "text/javascript">
<!--
function WriteCookie() {
var now = new Date();
now.setMonth( now.getMonth() - 1 );
cookievalue =
escape(document.myform.customer.value) + ";";
document.cookie = "name=" + cookievalue;
document.cookie = "expires=" + now.toUTCString() +
";"
document.write("Setting Cookies : " + "name=" +
cookievalue );
}
//-->
</script>
</head>
<body>
<form name = "myform" action = "">
Enter name: <input type = "text" name =
"customer"/>
<input type = "button" value = "Set Cookie"
onclick = "WriteCookie()"/>
</form>
</body>
</html>

```

5.4EVENTS

Some programs work with direct user input, such as mouse and keyboard actions. That kind of input isn't available as a well-organized data structure - it comes in piece by piece, in real time, and the program is expected to respond to it as it happens.

A better mechanism is for the system to actively notify our code when an event occurs. Browsers do this by allowing us to register functions as *handlers* for specific events. JavaScript's interaction with HTML is handled through *events* that occur when the user or the browser manipulates a page.

When the page loads, it is called an event. When the user clicks a button, that click too is an event. Other examples include events like pressing any key, closing a window, resizing a window, etc.

Example 5.8

```
<html>
<head>
<script type = "text/javascript">
function getValue()
{
var retVal = prompt("Enter your name : ", "your
name here");
document.write("You have entered : " + retVal);
}
</script>
</head>
<body>
<p>Click the following button to see the result:
</p>
<form><input type = "button" value = "Click Me"
onclick = "getValue();" />
</form>
</body>
</html>
```

Example 5.9

```
<html>
<head>
<title>registration</title>
<script type="text/javascript">
function validate()
{
if(document.myForm.fname.value=="")
{
alert("please provide your first name!");
```

Notes

Notes

```

document.myForm.fname.focus();
return false;
}
if(document.myForm.lastname.value=="")
{
alert("please provide your lastname!");
document.myForm.lastname.focus();
return false;
}
if(document.myForm.email.value=="")
{
alert("please provide your email!");
document.myForm.email.focus();
return false;
}

var x=document.forms["myForm"]["email"].value;
var atpos=x.indexOf("@");
var dotpos=x.lastIndexOf(".");
if(atpos<1||dotpos<atpos+2||dotpos+2>=x.length)
{
alert("not a valid e-mail address");
return false;
}
if(document.myForm.phone.value=="")
{
alert("please provide your phone number");
return false;
}
if(isNaN(document.myForm.phone.value))
{
alert("Phone number should be numeric");
document.myForm.phone.focus()
return false;
}
alert("register success!");
}
</script>
</head>
<body bgcolor="pink">

// ONSUBMIT event

<form name="myForm"
onsubmit="return(validate());">

<center><h1>REGISTRATION FORM-DEMO</h1>
<table cellspacing="2" cellpadding="2" border="1">
<tr>
<td align="right">name</td>

```

```

<td><input type="text" name="fname"/></td></tr>
<tr><td align="right">lastname</td>
<td><input type="text" name="lastname"/></td></tr>
<tr><td align="right">email</td>
<td><input type="text" name="email"/></td></tr>
<tr><td align="right">phone number</td>
<td><input type="text" name="phone"
id="phone"/></td></tr>
<tr><td align="right"></td>
<td><input type="submit"
value="submit"/></td></tr>
</table>
</center>
</form>
</html>

```

*Notes***Output**
Example 5.10

```

<html>
<head>
<script type = "text/javascript">
<!--
function over() {
document.write ("Mouse Over");
}
function out() {
document.write ("Mouse Out");
}
//-->
</script>
</head>
<body>
<p>Bring your mouse inside the division to see the
result:</p>
<div onmouseover = "over()" onmouseout = "out()">
<h2> This is inside the division </h2>
</div>
</body>
</html>

```

Notes

The standard HTML 5 events are listed here for your reference. Here script indicates a JavaScript function to be executed against that event.

Attribute	Description
Offline	Triggers when the document goes offline
Onabort	Triggers on an abort event
onafterprint	Triggers after the document is printed
onbeforeonload	Triggers before the document loads
onbeforeprint	Triggers before the document is printed
onblur	Triggers when the window loses focus
oncanplay	Triggers when media can start play, but might has to stop for buffering
oncanplaythrough	Triggers when media can be played to the end, without stopping for buffering
onchange	Triggers when an element changes
onclick	Triggers on a mouse click
oncontextmenu	Triggers when a context menu is triggered
ondblclick	Triggers on a mouse double-click
ondrag	Triggers when an element is dragged
ondragend	Triggers at the end of a drag operation
ondragenter	Triggers when an element has been dragged to a valid drop target
ondragleave	Triggers when an element is being dragged over a valid drop target
ondragover	Triggers at the start of a drag operation
ondragstart	Triggers at the start of a drag operation
ondrop	Triggers when dragged element is being dropped
ondurationchange	Triggers when the length of the media is changed
onemptied	Triggers when a media resource element suddenly becomes empty.
onended	Triggers when media has reach the end
onerror	Triggers when an error occur
onfocus	Triggers when the window gets focus
onformchange	Triggers when a form changes
onforminput	Triggers when a form gets user input
onhaschange	Triggers when the document has change
oninput	Triggers when an element gets user input
oninvalid	Triggers when an element is invalid
onkeydown	Triggers when a key is pressed
onkeypress	Triggers when a key is pressed and released
onkeyup	Triggers when a key is released
onload	Triggers when the document loads
onloadeddata	Triggers when media data is loaded
onloadedmetadata	Triggers when the duration and other media data of a media element is loaded
onloadstart	Triggers when the browser starts to load the media data

onmessage	Triggers when the message is triggered
onmousedown	Triggers when a mouse button is pressed
onmousemove	Triggers when the mouse pointer moves
onmouseout	Triggers when the mouse pointer moves out of an element
onmouseover	Triggers when the mouse pointer moves over an element
onmouseup	Triggers when a mouse button is released
onmousewheel	Triggers when the mouse wheel is being rotated
onoffline	Triggers when the document goes offline
ononline	Triggers when the document comes online
onpagehide	Triggers when the window is hidden
onpageshow	Triggers when the window becomes visible
onpause	Triggers when media data is paused
onplay	Triggers when media data is going to start playing
onplaying	Triggers when media data has start playing
onpopstate	Triggers when the window's history changes
onprogress	Triggers when the browser is fetching the media data
onratechange	Triggers when the media data's playing rate has changed
onreadystatechange	Triggers when the ready-state changes
onredo	Triggers when the document performs a redo
onresize	Triggers when the window is resized
onscroll	Triggers when an element's scrollbar is being scrolled
onseeked	Triggers when a media element's seeking attribute is no longer true, and the seeking has ended
onseeking	Triggers when a media element's seeking attribute is true, and the seeking has begun
onselect	Triggers when an element is selected
onstalled	Triggers when there is an error in fetching media data
onstorage	Triggers when a document loads
onsubmit	Triggers when a form is submitted
onsuspend	Triggers when the browser has been fetching media data, but stopped before the entire media file was fetched
ontimeupdate	Triggers when media changes its playing position
onundo	Triggers when a document performs an undo
onunload	Triggers when the user leaves the document
onvolumechange	Triggers when media changes the volume, also when volume is set to "mute"
onwaiting	Triggers when media has stopped playing, but is expected to resume

Notes

Check Your Progress

1. State the purpose of regular expressions.
2. What do you mean by cookie?
3. What is the purpose of events?

5.5 ANSWERS TO CHECK YOUR PROGRESS

1. A regular expression is an object that describes a pattern of characters
2. A *cookie* is a piece of data that is stored on our computer to be accessed by our browser
3. JavaScript's interaction with HTML is handled through *events* that occur when the user or the browser manipulates a page

5.6 LET US SUM UP

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

A regular expression could be defined with the `RegExp()` constructor, as follows

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

A `pattern` is a string that specifies the pattern of the regular expression or another regular expression.

A metacharacter is simply an alphabetical character preceded by a backslash that acts to give the combination a special meaning.

A *cookie* is a piece of data that is stored on our computer to be accessed by our browser

Cookies are a plain text data record of 5 variable-length fields

- **Expires** – The date the cookie will expire. If this is blank, the cookie will expire when the visitor quits the browser.
- **Domain** – The domain name of your site.
- **Path** – The path to the directory or web page that set the cookie. This may be blank if you want to retrieve the cookie from any directory or page.

- **Secure** – If this field contains the word "secure", then the cookie may only be retrieved with a secure server. If this field is blank, no such restriction exists.
- **Name=Value** – Cookies are set and retrieved in the form of key-value pairs

The simplest way to create a cookie is to assign a string value to the `document.cookie` object

JavaScript's interaction with HTML is handled through *events* that occur when the user or the browser manipulates a page

Notes

5.7 SELF-ASSESSMENT EXERCISES

Short Questions

1. What is the purpose of using regular expressions?
2. What do you mean by modifiers?
3. Write the statement for setting a cookie
4. How to delete a cookie?
5. What is the advantage of using cookies?
6. What is an event?
7. List some of the events supported in HTML 5.

Detail Questions

1. Discuss in detail about regular expressions.
2. With suitable examples explain about cookies.
3. Describe about handling events in JavaScript.

5.8 SUGGESTED READINGS

1. Web Programming: Building Internet applications, Chris Bates, Wiley India
2. Web technologies – A computer science perspective, Jeffrey C Jackson, Pearson Education, 2006
3. Robert W. Sebesta “Programming the world wide web” Pearson Education
4. Bates, Developing web applications, wiley, 2006
5. www.w3schools.com
6. JavaScript: The Definitive Guide, 6th Edition, David Flanagan, O’Reilly Media, 2011.
7. Eloquent JavaScript, 3rd edition, Marijn Haverbeke, 2018

Notes

UNIT -6 DYNAMIC HTML WITH JAVASCRIPT

Structure

- 6.0 Introduction
- 6.1 Objectives
- 6.2 Data Validation
- 6.3 Messages and Confirmation
- 6.4 Writing to a different frame
- 6.5 Rollover buttons
- 6.6 Moving images
- 6.7 Answers to Check Your Progress
- 6.8 Let us Sum up
- 6.9 Self-Assessment Exercises
- 6.10 Suggested Readings

6.0 INTRODUCTION

Forms are used in webpages for the user to enter their required details that are further sending it to the server for processing. A form is also known as web form or HTML form. The data submitted to the server need to be validated before processing. Data validation is the process of ensuring that users need to submit only the set of characters which are required for processing. Messages and confirmations are used for this purpose. Sometimes we need to give the responses into different frames for this. This unit will deals about the above mentioned topics.

6.1 OBJECTIVES

After going through this unit, you will be able to:

- Learn the need for data validations
- Understand how to write to a different frame
- Know the processing of rollover buttons and moving images

6.2 DATA VALIDATION

Validation is simply the process of ensuring that some data might be correct data for a particular application. Form validation normally used to occur at the server, after the client had entered all the necessary data and then pressed the Submit button. If the data entered by a client was incorrect or was simply missing, the server would have to send all the data back to the client and request that the form be resubmitted with correct information. This was really a lengthy process which used to put a lot of burden on the server.

JavaScript provides a way to validate form's data on the client's computer before sending it to the web server. Form validation generally performs two functions.

- **Basic Validation** – The form must be checked to make sure all the mandatory fields are filled in. It would require just a loop through each field in the form and check for data.
- **Data Format Validation** – The data that is entered must be checked for correct form and value. User code must include appropriate logic to test correctness of data.

Notes

Example 6.1 – Basic Validation

```
<script type = "text/javascript">
<!--
    // Form validation code will come here.
    function validate() {

        if( document.myForm.Name.value == "" ) {
            alert( "Please provide your name!" );
            document.myForm.Name.focus() ;
            return false;
        }
        if( document.myForm.Email.value == "" ) {
            alert( "Please provide your Email!" );
            document.myForm.Email.focus() ;
            return false;
        }
        if( document.myForm.Zip.value == "" ||
isNaN( document.myForm.Zip.value ) ||
        document.myForm.Zip.value.length != 5
) {

            alert( "Please provide a zip in the
format #####." );
            document.myForm.Zip.focus() ;
            return false;
        }
        if( document.myForm.Country.value == "-1"
) {
            alert( "Please provide your country!"
);
            return false;
        }
        return( true );
    }
    //-->
</script>
```

Notes

Example 6.2 – Data Format Validation

```
<script type = "text/javascript">
<!--
    function validateEmail() {
        var emailID =
document.myForm.EMail.value;
        atpos = emailID.indexOf("@");
        dotpos = emailID.lastIndexOf(".");

        if (atpos < 1 || ( dotpos - atpos < 2 ))
    {
            alert("Please enter correct email ID")
            document.myForm.EMail.focus() ;
            return false;
        }
        return( true );
    }
    //-->
</script>
```

Example 6.3 – Login Form Validation

```
<html>
<body>
<Form name="idcheck">
<table>
<tr>
<td>FirstName:</td>
<td><input type="text" name="fnm"></td>
</tr>
<tr>
<td>LastName:</td>
<td><input type="text" name="lnm"></td>
</tr>
<tr>
<td>E-mail:</td>
<td><input type="text" name="eid"></td>
</tr>
</table>

<input type="button" value="submit"
onClick="emailvalid()">

<input type="Reset" value="reset">
</Form>

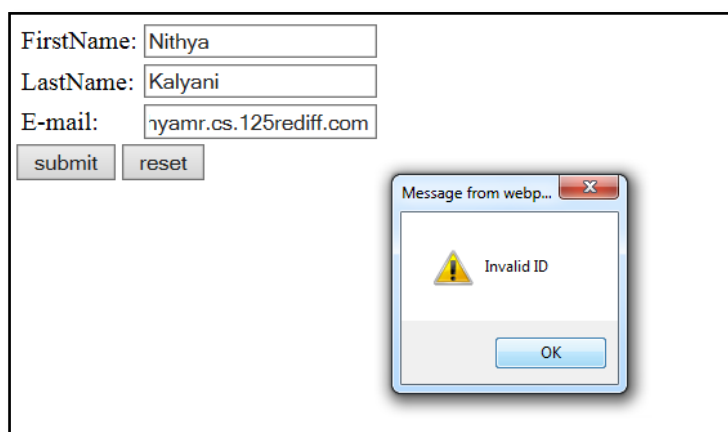
<Script type="text/javascript">

function emailvalid()
{
```

```
var first,last,id;
first=idcheck.fnm.value;
last=idcheck.lnm.value;
id=idcheck.eid.value;
var idreg=new RegExp(/^([a-zA-Z0-9._-]+@[a-zA-Z0-9.-]+\.[a-zA-Z]{2,4})$/);
var finalid=idreg.exec(id);
if(first=="")
{
    alert("Please Enter your FirstName");
    first.focus();
}
else if(last=="")
{
    alert("Please Enter your Last Name");
    last.focus();
}
else if(!finalid)
{
    alert("Invalid ID");
    id.focus();
}
else
    alert("Thank You");
}
</script>
</body>
</html>
```

Notes

Output



6.3 MESSAGES AND CONFIRMATIONS

JavaScript supports three important types of dialog boxes. These dialog boxes can be used to raise and alert, or to get confirmation on any input or to have a kind of input from the users. Here we will discuss each dialog box one by one.

Alert Dialog Box

An **alert** dialog box is mostly used to give a warning message to the users. For example, if one input field requires entering some text but the user does not provide any input, then as a part of validation, you can use an **alert** box to give a warning message.

Example 6.4

```
<html>
<head>
<script type = "text/javascript">
<!--
        function Warn() {
            alert ("This is a warning
message!");
            document.write ("This is a warning
message!");
        }
//-->
</script>
</head>

<body>
<p>Click the following button to see the result:
</p>
<form>
<input type = "button" value = "Click Me" onclick
= "Warn();" />
</form>
</body>
</html>
```

Confirmation Dialog Box

A **confirmation** dialog box is mostly used to take user's consent on any option. It displays a dialog box with two buttons: OK and Cancel.

If the user clicks on the OK button, the window method **confirm()** will return true. If the user clicks on the Cancel button, then **confirm()** returns false.

Example 6.5

```
<html>
<head>
<script type = "text/javascript">
<!--
        function getConfirmation() {
            var retVal = confirm("Do you want
to continue ?");
```

```
        if( retVal == true ) {
            document.write ("User wants to
continue!");
            return true;
        } else {
            document.write ("User does not
want to continue!");
            return false;
        }
    }
    //-->
</script>
</head>

<body>
<p>Click the following button to see the result:
</p>
<form>
<input type = "button" value = "Click Me" onclick
= "getConfirmation();" />
</form>
</body>
</html>
```

Notes

Prompt Dialog Box

The prompt dialog box is very useful when you want to pop-up a text box to get user input. Thus, it enables you to interact with the user. The user needs to fill in the field and then click OK.

This dialog box is displayed using a method called `prompt()` which takes two parameters:

- (i) a label which you want to display in the text box and
- (ii) a default string to display in the text box.

This dialog box has two buttons: `OK` and `Cancel`.

If the user clicks the `OK` button, the window method `prompt()` will return the entered value from the text box. If the user clicks the `Cancel` button, the window method `prompt()` returns null.

Example 6.6

```
<html>
<head>
<script type = "text/javascript">
<!--
        function getValue() {
            var retVal = prompt("Enter your
name : ", "your name here");
```

Notes

```
        document.write("You have entered :  
" + retVal);  
    }  
    //-->  
</script>  
</head>  
  
<body>  
<p>Click the following button to see the result:  
</p>  
<form>  
<input type = "button" value = "Click Me" onclick  
= "getValue();" />  
</form>  
</body>  
</html>
```

6.4 WRITING TO A DIFFERENT FRAME

We can also use the `document.write()` method to send dynamically created content to another frame in a frameset or to another browser window previously opened by a script in the same page. It can be done with the help of `parent.<<framename>>.document.write()` method.

All you need for this kind of content creation is a valid reference to the other frame or window.

A typical frameset document defines the physical layout of how the main browser window is to be subdivided into separate panels. Framesets can, of course, be nested many levels deep, where one frame loads a document that is, itself, a frameset document. The key to writing a valid reference to a distant frame knows the relationship between the frame that contains the script doing the writing and the target frame.

Example 6.7

parent.html

```
<HTML>  
<HEAD><TITLE>Frames Example</TITLE></HEAD>  
<!-- divide into two columns -->  
<FRAMESET COLS="50%,*">  
<FRAME SRC="frame1.html" NAME="frame1">  
<FRAME SRC="frame2.html">  
</FRAMESET>  
<BODY>  
</BODY>  
</HTML>
```


frame1.html

```
<HTML>
<HEAD><TITLE>Frames Example</TITLE>
</HEAD>
<BODY>
</BODY>
</HTML>
```

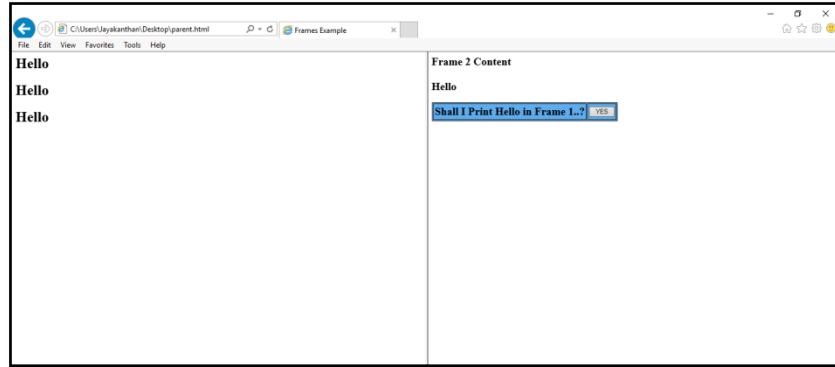
Notes

frame2.html

```
<HTML>
<HEAD>
<TITLE>Frames Example</TITLE>
</HEAD>
<BODY>
<SCRIPT LANGUAGE="JavaScript">
<!-- hide from old browsers
//
function doit()
{
parent.frame1.document.write("Hello<br>");
}
//
// end script hiding -->
</SCRIPT>
Frame 2 Content
<BR>
<BR>
Hello
<BR>
<table border=1 cellpadding=3 bgcolor="#5cacee">
<!-- Buttons -->
<tr bgcolor="#5cacee">
<form NAME="roundit">
<td><font face=arial size=-1><b>
<p align="center">Shall I Print Hello in Frame
1..? </p></b>
</font>
</td>
<td>
<p align="center">
<input TYPE="button" VALUE=" YES "
Onclick="doit()"></p>
</td>
</form>
</tr>
</table>
</BODY>
</HTML>
```

Notes

Output



6.5 ROLLOVER BUTTONS

On many web pages, JavaScript rollovers are handled by adding an `onmouseover()` and `onmouseout()` event on images.

- `onmouseover()` is triggered when the mouse moves over an element
- `onmouseout()` is triggered when the mouse moves away from the element

Example 6.8

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

function bigImg(x)
{
x.style.height="400px";
x.style.width="450px";
}

function normalImg(x)
{
x.style.height="200px";
x.style.width="250px";
}

function MouseRollover(MyImage)
{
    MyImage.src = "sf511-s03in_3.jpg";
}

function MouseOut(MyImage)
{
    MyImage.src = "Win-Seven.jpg";
}
```

```

</script>
</head>

<body>

<table border="3">

<tr>
<td>
</td>

<td></td>

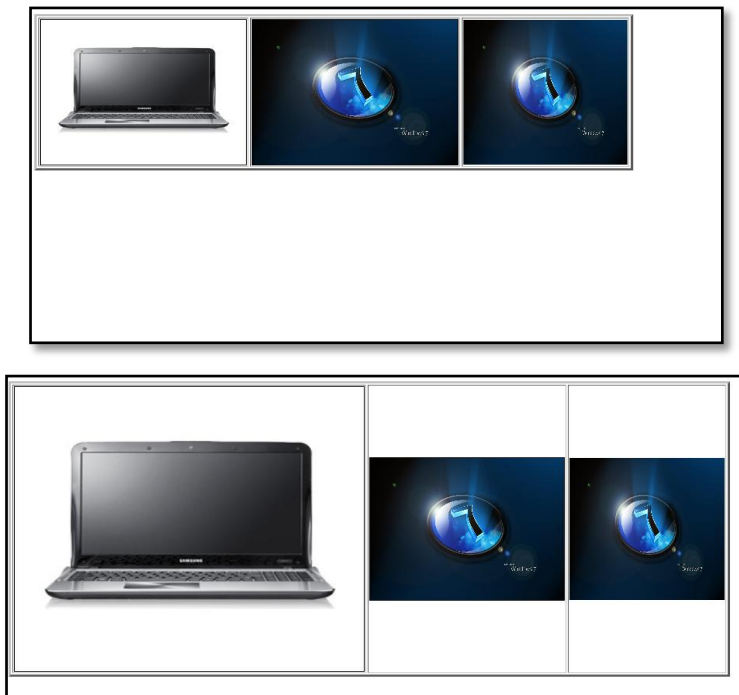
<td></td>

</td>
</tr>
</table>
</body>
</html>

```

Notes

Output



6.6 MOVING IMAGES

JavaScript can be used to move a number of DOM elements (, <div> or any other HTML element) around the page according to some sort of pattern determined by a logical equation or function.

JavaScript provides the following two functions to be frequently used in animation programs.

- `setTimeout(function, duration)`
This function calls `function` after `duration` milliseconds from now.
- `setInterval(function,duration)`
Thisfunction calls `function` after every `duration` milliseconds.
- `clearTimeout(setTimeout_variable)`
This function calls clears any timer set by the `setTimeout()` functions.

Example 6.9 – Manual Animation

```
<html><head>
<title>JavaScript Animation</title>
<script type = "text/javascript">
<!--
var imgObj = null;
function init()
{
imgObj = document.getElementById('myImage');
imgObj.style.position= 'relative';
imgObj.style.left = '0px';
}
function moveRight()
{
imgObj.style.left = parseInt(imgObj.style.left) +
10 + 'px';
}
window.onload = init;
//-->
</script>
</head>
<body>
<form>
<img id = "myImage" src = "myImage.jpg" />
<p>Click button below to move the image to
right</p>
<input type = "button" value = "Click Me" onclick
= "moveRight();" />
</form>
</body>
</html>
```

Output*Notes*

- We are using the JavaScript function `getElementById()` to get a DOM object and then assigning it to a global variable `imgObj`.
- We have defined an initialization function `init()` to initialize `imgObj`
- We have set its `position` and `left` attributes.
- We are calling initialization function at the time of window load.
- Finally, we are calling `moveRight()` function to increase the left distance by 10 pixels. You could also set it to a negative value to move it to the left side

Example 6.9 – Manual Animation

```

<html>
<head>
<title>JavaScript Animation</title>
<script type = "text/javascript">
<!--
        var imgObj = null;
        var animate ;

        function init() {
            imgObj =
document.getElementById('myImage');
            imgObj.style.position= 'relative';
            imgObj.style.left = '0px';
        }
        function moveRight() {
            imgObj.style.left =
parseInt(imgObj.style.left) + 10 + 'px';
            animate = setTimeout(moveRight,20);
// call moveRight in 20msec
        }
        function stop() {

```

Notes

```
        clearTimeout(animate);
        imgObj.style.left = '0px';
    }
    window.onload = init;
//-->
</script>
</head>

<body>
<form>
<img id = "myImage" src = "myImage.jpg" />
<p>Click the buttons below to handle animation</p>
<input type = "button" value = "Start" onclick =
"moveRight();" />
<input type = "button" value = "Stop" onclick =
"stop();" />
</form>
</body>
</html>
```

Output



- The `moveRight()` function is calling `setTimeout()` function to set the position of `imgObj`.
- We have added a new function `stop()` to clear the timer set by `setTimeout()` function and to set the object at its initial position.

Check Your Progress

1. What do you mean by validation?
2. Name the three types of dialog boxes supported by JavaScript.
3. How will you write to a different frame?

6.7 ANSWERS TO CHECK YOUR PROGRESS

1. Validation is simply the process of ensuring that some data might be correct data for a particular application
2. Three types of dialog boxes in JavaScript are Alert, Confirmation and Prompt
3. Writing into a different frame can be done with the help of `parent.<<frameName>>.document.write()` method

Notes

6.8 LET US SUM UP

Validation is simply the process of ensuring that some data might be correct data for a particular application.

Basic Validation – The form must be checked to make sure all the mandatory fields are filled in. It would require just a loop through each field in the form and check for data.

Data Format Validation – The data that is entered must be checked for correct form and value. User code must include appropriate logic to test correctness of data

Three types of dialog boxes in JavaScript are Alert, Confirmation and Prompt

An **alert** dialog box is mostly used to give a warning message to the users.

A **confirmation** dialog box is mostly used to take user's consent on any option.

The **prompt** dialog box is very useful when you want to pop-up a text box to get user input.

Writing into a different frame can be done with the help of `parent.<<frameName>>.document.write()` method

On many web pages, JavaScript rollovers are handled by adding an `onmouseover()` and `onmouseout()` event on images

JavaScript provides the following two functions to be frequently used in animation programs.

- `setTimeout(function, duration)`

This function calls `function` after `duration` milliseconds from now.

- `setInterval(function, duration)`

This function calls `function` after every `duration` milliseconds.

- `clearTimeout(setTimeout_variable)`

This function calls clears any timer set by the `setTimeout()` functions.

6.9 SELF-ASSESSMENT EXERCISES

Short Questions

1. State the importance of data validations in web forms.
2. What is the syntax of confirmation dialog box?
3. How to perform rollovers in JavaScript?
4. List the functions used to perform animations in JavaScript

Detail Questions

1. How will you perform data validations in JavaScript? Explain with examples.
2. With suitable examples explain about creating dialog boxes.
3. Explain how to write to a different frame.
4. Write a program to create rollover buttons.
5. With suitable example explain how to perform moving of images.

6.10 SUGGESTED READINGS

1. Web Programming: Building Internet applications, Chris Bates, Wiley India
2. Web technologies – A computer science perspective, Jeffrey C Jackson, Pearson Education, 2006
3. Robert W. Sebesta “Programming the world wide web” Pearson Education
4. Bates, Developing web applications, wiley, 2006
5. www.w3schools.com
6. JavaScript: The Definitive Guide, 6th Edition, David Flanagan, O’Reilly Media, 2011.
7. Eloquent JavaScript, 3rd edition, Marijn Haverbeke, 2018

BLOCK – III

HOST OBJECTS

Notes

UNIT -7 DOCUMENT OBJECT MODEL

Structure

- 7.0 Introduction
- 7.1 Objectives
- 7.2 Document Object Model
- 7.3 Browsers and DOM
- 7.4 DOM history and levels
- 7.5 Document elements
- 7.6 Intrinsic Event Handling
- 7.7 Answers to Check Your Progress
- 7.8 Let us Sum up
- 7.9 Self-Assessment Exercises
- 7.10 Suggested Readings

7.0 INTRODUCTION

It is explained that every web browser window, tab, and frame is represented by a Window object. Every Window object has a document property that refers to a Document object. The Document object represents the content of the window, and it is the subject of this unit.

7.1 OBJECTIVES

After going through this unit, you will be able to:

- Query or *select* individual elements from a document.
- *Traverse* a document as a tree of nodes, and how to find the ancestors, siblings, and descendants of any document element.
- Modify the structure of a document by creating, inserting, and deleting nodes.

7.2 DOCUMENT OBJECT MODEL

The Document Object Model, or DOM, is the fundamental API for representing and manipulating the content of HTML and XML documents. The API is not particularly complicated, but there are a number of architectural details you need to understand. First, you should understand

that the nested elements of an HTML or XML document are represented in the DOM as a tree of objects.

The tree representation of an HTML document contains nodes representing HTML tags or elements, such as `<body>` and `<p>`, and nodes representing strings of text. An HTML document may also contain nodes representing HTML comments.

Let us consider the following simple HTML document:

```
<html>
<head>
<title>Sample Document</title>
</head>
<body>
<h1>An HTML Document</h1>
<p>This is a <i>simple</i> document.
</html>
```

The DOM representation of this document is the tree pictured in the following figure.

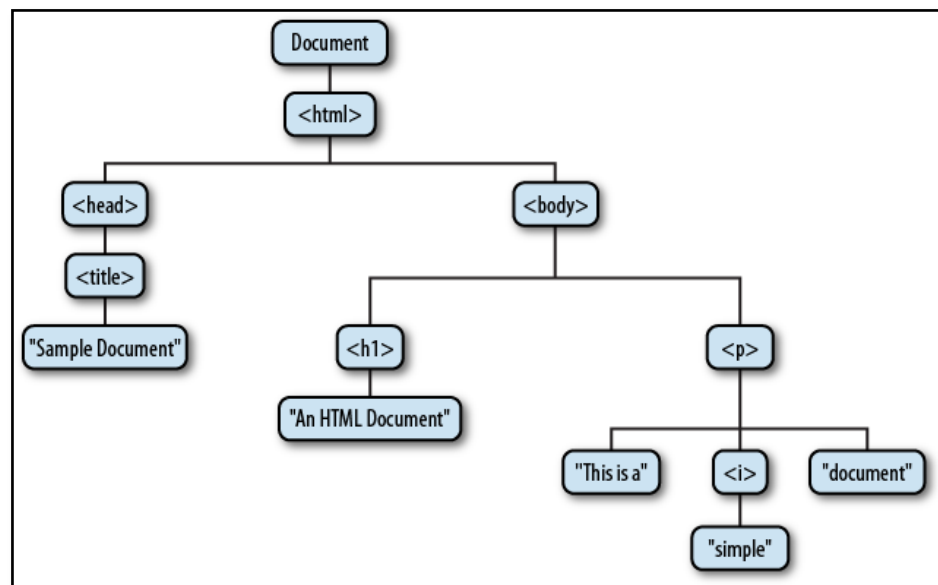


Figure 7.1. The tree representation of an HTML document

The node directly above a node is the *parent* of that node. The nodes that are one level directly below another node are the *children* of that node. Nodes at the same level, and with the same parent, are *siblings*. The set of nodes any number of levels below another node are the *descendants* of that node. And the parent, grandparent, and all other nodes above a node are the *ancestors* of that node.

7.3 BROWSERS AND DOM

There are several DOMs in existence. The following sections explain each of these DOMs in detail and describe how you can use them to access and modify document content.

- The Legacy DOM – This is the model which was introduced in early versions of JavaScript language. It is well supported by all browsers, but allows access only to certain key portions of documents, such as forms, form elements, and images.
- The W3C DOM – This document object model allows access and modification of all document content and is standardized by the World Wide Web Consortium (W3C). This model is supported by almost all the modern browsers.
- The IE4 DOM – This document object model was introduced in Version 4 of Microsoft's Internet Explorer browser. IE 5 and later versions include support for most basic W3C DOM features.

Major vendors realize the importance of DOM and have begun to make their Web browsers DOM compliant. NN 7 and IE 6 already have good DOM support. In particular NN led the way in supporting DOM Level 1 and Level 2. Most examples in this chapter will work under both NN 6, IE 6 and later versions.

To detect the extent of DOM support that a user agent (browser) provides, the following type of Javascript code can be used:

```
var imp = document.implementation;
if ( typeof imp != "undefined" &&
    imp.hasFeature("HTML", "1.0") &&
    imp.hasFeature("Events", "2.0") &&
    imp.hasFeature("CSS", "2.0")
)
{
  . . .
}
```

A browser is DOM compliant if it supports the interfaces specified by DOM. But it can also add interfaces not specified or add fields and methods to the required interfaces. For example NN and IE both add innerHTML to the HTML Element interface. It is easy to test if a field or method is available in a browser. For example,

```
if ( document.getElementById )
  . . .
```

tests if the getElementById method is available in the document object.

DOM compatibility

If you want to write a script with the flexibility to use either W3C DOM or IE 4 DOM depending on their availability, then you can use a capability-testing approach that first checks for the existence of a method or property to determine whether the browser has the capability you desire. For example,

Notes

```

if (document.getElementById)
{
    // If the W3C method exists, use it
} else if (document.all)
{
    // If the all[] array exists, use it
} else
{
    // Otherwise use the legacy DOM
}

```

7.4 DOM HIERARCHY AND LEVELS

In **DOM**, the Objects are organized in a hierarchy. This hierarchical structure applies to the organization of objects in a Web document.

- **Window object** – Top of the hierarchy. It is the outmost element of the object hierarchy.
- **Document object** – Each HTML document that gets loaded into a window becomes a **document** object. The document contains the contents of the page.
- **Form object** – Everything enclosed in the `<form>...</form>` tags sets the **form** object.
- **Form control elements** – The form object contains all the **elements** defined for that object such as text fields, buttons, radio buttons, and checkboxes.

Example 7.1

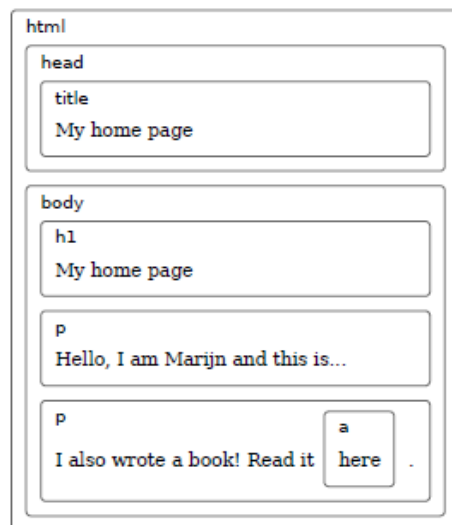
```

<!doctype html>
<html>
<head>
<title>My home page</title>
</head>
<body>

<h1>My home page</h1>
<p>Hello, I am Marijn and this is my home
page.</p>
<p>I also wrote a book! Read it
<a
href="http://eloquentjavascript.net">here</a>.</p>
</body>
</html>

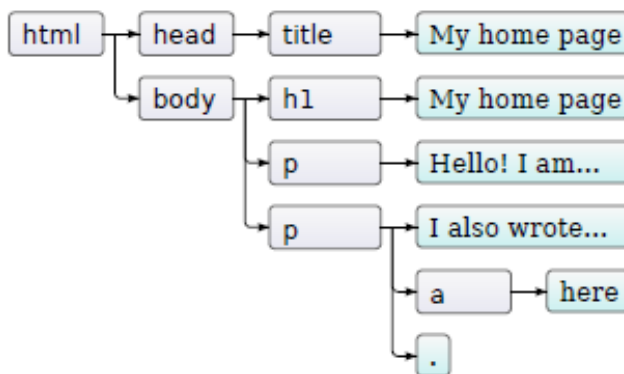
```

DOM Hierarchy of Example 7.1

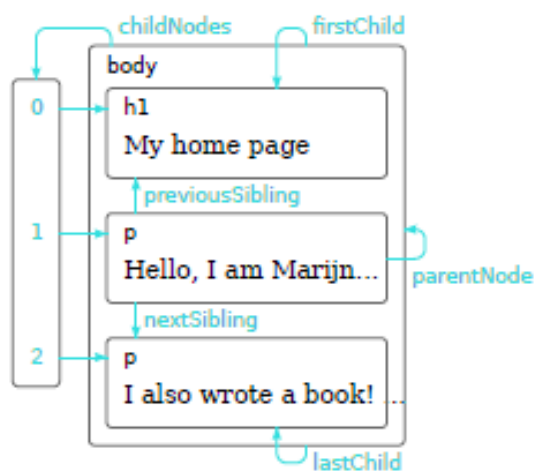


Notes

Tree Structure of Example 7.1



Relationships among the elements in Example 7.1



Notes

The Document object, its Element objects, and the Text objects that represent text in the document are all Node objects. Node defines the following important properties:

parentNode - The Node that is the parent of this one, or null for nodes like the Document object that have no parent.

childNodes - A read-only array-like object (a NodeList) that is a live representation of a Node's child nodes.

firstChild, lastChild - The first and last child nodes of a node, or null if the node has no children.

nextSibling, previousSibling - The next and previous sibling node of a node. Two nodes with the same parent are siblings. Their order reflects the order in which they appear in the document. These properties connect nodes in a doubly linked list.

nodeType - The kind of node this is. Document nodes have the value 9. Element nodes have the value 1. Text nodes have the value 3. Comment nodes are 8 and Document-Fragment nodes are 11.

nodeValue - The textual content of a Text or Comment node.

nodeName - The tag name of an Element, converted to uppercase.

Using these Node properties, the second child node of the first child of the Document can be referred to with expressions like these:

```
document.childNodes[0].childNodes[1]
document.firstChild.firstChild.nextSibling
```

Suppose the document in question is the following:

```
<html><head><title>Test</title></head><body>Hello
World!</body></html>
```

Then the second child of the first child is the `<body>` element. It has a `nodeType` of 1 and a `nodeName` of "BODY".

```
<script>
function replaceImages()
{
let images =
document.body.getElementsByTagName("img");
for (let i = images.length - 1; i >= 0; i--)
{
let image = images[i];
if (image.alt)
{
let text = document.createTextNode(image.alt);
```

```

image.parentNode.replaceChild(text, image);
}
}
}
</script>

```

Notes

7.5 DOCUMENT ELEMENTS

Most of the client-side JavaScript programs work by manipulating one or more document elements. When these programs start, they can use the global variable `document` to refer to the `Document` object. In order to manipulate elements of the document, however, they must obtain or select the `Element` objects that refer to those document elements. The DOM defines a number of ways to select elements; you can query a document for an element or elements:

- with a specified `id` attribute;
- with a specified `name` attribute;
- with the specified `tag` name;
- with the specified CSS class or classes; or
- matching the specified CSS selector

In the following section we will explore all the types of selecting the element.

Selecting Elements By ID

Any HTML element can have an `id` attribute. The value of this attribute must be unique within the document—no two elements in the same document can have the same ID. You can select an element based on this unique ID with the `getElementById()` method of the `Document` object.

```
var sec1 = document.getElementById("section1");
```

Selecting Elements by Name

The HTML `name` attribute was originally intended to assign names to form elements, and the value of this attribute is used when form data is submitted to a server. Like the `id` attribute, `name` assigns a name to an element. Unlike `id`, however, the value of a `name` attribute does not have to be unique: multiple elements may have the same name, and this is common in the case of radio buttons and checkboxes in forms. Also, unlike `id`, the `name` attribute is only valid on a handful of HTML elements, including forms, form elements, `<iframe>`, and `` elements.

To select HTML elements based on the value of their `name` attributes, you can use the `getElementsByName()` method of the `Document` object:

```
var radiobuttons =
document.getElementsByName("favorite_color");
```

Selecting Elements by Type

You can select all HTML or XML elements of a specified type (or tag name) using the `getElementsByTagName()` method of the Document object. To obtain a read-only arraylike object containing the Element objects for all `` elements in a document, for example, you might write:

```
var spans = document.getElementsByTagName("span");
```

Like `getElementsByName()`, `getElementsByTagName()` returns a `NodeList` object.

Selecting Elements by CSS Class

HTML5 defines a method, `getElementsByClassName()`, that allows us to select sets of document elements based on the identifiers in their class attribute.

```
// Find all elements that have "warning"
// in their class attribute
var warnings =
document.getElementsByClassName("warning");
```

Like `getElementsByTagName()`, `getElementsByClassName()` can be invoked on both HTML documents and HTML elements, and it returns a live `NodeList` containing all matching descendants of the document or element. `getElementsByClassName()` takes a single string argument, but the string may specify multiple space-separated identifiers.

Selecting Elements with CSS Selectors

CSS stylesheets have a very powerful syntax, known as *selectors*, for describing elements or sets of elements within a document. Elements can be described by ID, tag name, or class:

```
#nav
// An element with id="nav"
div
// Any <div> element
.warning
// Any element with "warning" in its class
// attribute
```

More generally, elements can be selected based on attribute values:


```
p[lang="fr"]
// A paragraph written in French: <p lang="fr">

*[name="x"]
// Any element with a name="x" attribute
```

Notes

These basic selectors can be combined:

```
span.fatal.error
// Any <span> with "warning" and "fatal"
// in its class

span[lang="fr"].warning
// Any warning in French
```

Selectors can also specify document structure:

```
#log span
// Any <span> descendant of the element
// with id="log"

#log>span
// Any <span> child of the element with id="log"

<body>h1:first-child
// The first <h1> child of the <body>
```

Selectors can be combined to select multiple elements or multiple sets of elements:

```
div, #log
// All <div> elements plus the element
// with id="log"
```

As you can see, CSS selectors allow elements to be selected in all of the ways described above: by ID, by name, by tag name, and by class name.

7.6 INTRINSIC EVENT HANDLING

Intrinsic event handlers are ways to attach specific scripts to your documents that are executed only when something happens to an element. Not all event handlers apply to all elements.

Example 7.2

```
<html>
<body>
<p>
When you enter the input field, a function is
triggered which sets the background color to
```

Notes

```
yellow. When you leave the input field, a function
is triggered which removes the background color.
</p>

<form id="myForm">
<input type="text" id="myInput">
</form>

<script>

var x = document.getElementById("myForm");
x.addEventListener("focus", myFocusFunction,
true);
x.addEventListener("blur", myBlurFunction, true);

function myFocusFunction()
{
document.getElementById("myInput").style.backgroun
dColor = "yellow";
}

function myBlurFunction()
{
document.getElementById("myInput").style.backgroun
dColor = "";
}

</script>

</body>
</html>
```

Output

When you enter the input field (child of FORM), a function is triggered which sets the background color to yellow. When you leave the input field, a function is triggered which removes the background color.



When you enter the input field (child of FORM), a function is triggered which sets the background color to yellow. When you leave the input field, a function is triggered which removes the background color.

Notes

Example 7.3

```
<html><body>
<h1 onclick="this.innerHTML='Oops!'">
Click on this text!</h1>
</body></html>
```

Output

Click on this text!

Oops!

Example 7.4

```
<html>
<body>
<p>
When you enter the input field, a function is
triggered which sets the background color to
yellow. When you leave the input field, a function
is triggered which removes the background color.
</p>

<form id="myForm">
<input type="text" id="myInput">
</form>

<script>

var x = document.getElementById("myForm");
x.addEventListener("focus", myFocusFunction,
true);
```

Notes

```
x.addEventListener("blur", myBlurFunction, true);

function myFocusFunction()
{
document.getElementById("myInput").style.backgroun
dColor = "yellow";
}

function myBlurFunction()
{
document.getElementById("myInput").style.backgroun
dColor = "";
}

</script>

</body>
</html>
```

Check Your Progress

1. What is DOM?
2. What are the levels in DOM hierarchy?
3. List the ways to select the DOM elements.

7.7 ANSWERS TO CHECK YOUR PROGRESS

1. The Document Object Model, or DOM, is the fundamental API for representing and manipulating the content of HTML and XML documents.
2. In **DOM**, the Objects are organized in a hierarchy
 - Window object.
 - Document object.
 - Form object.
 - Form control elements.
3. The DOM defines a number of ways to select elements
 - with a specified id attribute;
 - with a specified name attribute;
 - with the specified tag name;
 - with the specified CSS class or classes; or
 - matching the specified CSS selector

7.8 LET US SUM UP

The Document Object Model, or DOM, is the fundamental API for representing and manipulating the content of HTML and XML documents.

DOMs can be Legacy DOM, W3C DOM or IE4 DOM

In **DOM**, the Objects are organized in a hierarchy

- **Window object** – Top of the hierarchy. It is the outmost element of the object hierarchy.
- **Document object** – Each HTML document that gets loaded into a window becomes a **document** object. The document contains the contents of the page.
- **Form object** – Everything enclosed in the `<form>...</form>` tags sets the form object.
- **Form control elements** – The form object contains all the **elements** defined for that object such as text fields, buttons, radio buttons, and checkboxes.

The DOM defines a number of ways to select elements:

- with a specified id attribute;
- with a specified name attribute;
- with the specified tag name;
- with the specified CSS class or classes; or
- matching the specified CSS selector

Intrinsic event handlers are ways to attach specific scripts to your documents that are executed only when something happens to an element.

7.9 SELF-ASSESSMENT EXERCISES

Short Questions

1. What is the purpose of DOM?
2. Draw the tree structure of a HTML document.
3. List the properties of Node object.
4. How will you select elements in DOM?

Detail Questions

1. Discuss in details about DOM hierarchy.
2. With suitable examples explain properties of Node object.
3. Describe the ways to select the elements of DOM.
4. Write a program to implement the concept of intrinsic even handling.

Notes

Notes

7.10 SUGGESTED READINGS

1. Web Programming: Building Internet applications, Chris Bates, Wiley India
2. Web technologies – A computer science perspective, Jeffrey C Jackson, Pearson Education, 2006
3. Robert W. Sebesta “Programming the world wide web” Pearson Education
4. Bates, Developing web applications, wiley, 2006
5. www.w3schools.com
6. JavaScript: The Definitive Guide, 6th Edition, David Flanagan, O’Reilly Media, 2011.
7. Eloquent JavaScript, 3rd edition, Marijn Haverbeke, 2018

UNIT- 8 REPRESENTING WEB DATA

Structure

- 8.0 Introduction
- 8.1 Objectives
- 8.2 XML Basics
- 8.3 XML and HTML
- 8.4 Documents and Vocabularies
- 8.5 Versions and declarations
- 8.6 Namespaces
- 8.7 Answers to Check Your Progress
- 8.8 Let us Sum up
- 8.9 Self-Assessment Exercises
- 8.10 Suggested Readings

Notes

8.0 INTRODUCTION

XML stands for eXtensible Markup Language and is a text-based markup language derived from Standard Generalized Markup Language (SGML). This unit will provide you the basics of XML. The web data are usually represented using XML.

8.1 OBJECTIVES

After going through this unit, you will be able to:

- Understand the basics of XML
- Represent web data using XML
- Learn about Namespaces.

8.2 XML BASICS

XML stands for eXtensible Markup Language, developed by W3C in 1996. XML 1.0 was officially adopted as a W3C recommendation in 1998. XML was designed to carry data, not to display data. XML is designed to be self-descriptive. XML is a subset of SGML that can define your own tags. A Meta Language and tags describe the content. XML Supports CSS, XSL, DOM.

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

There are three important *characteristics* of XML that make it useful in a variety of systems and solutions –

- **XML is extensible** – XML allows you to create your own self-descriptive tags, or language, that suits your application.
- **XML carries the data, does not present it** – XML allows you to store the data irrespective of how it will be presented.
- **XML is a public standard** – XML was developed by an organization called the World Wide Web Consortium (W3C) and is available as an open standard.

A short list of XML *usage* is given below

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

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

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

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

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

A programming language consists of grammar rules and its own vocabulary which is used to create computer programs. These programs instruct the computer to perform specific tasks. XML does not qualify to be a programming language as it does not perform any computation or algorithms. It is usually stored in a simple text file and is processed by special software that is capable of interpreting XML.

Notes

Example 8.1

```
<?xml version = "1.0"?>
<contact-info>
<name>Pragati</name>
<company>Hummingbird</company>
<phone>(011) 123-4567</phone>
</contact-info>
```

There are two kinds of information in the above example

- Markup, like `<contact-info>`
- The text, or the character data, such as `Pragati`, `Hummingbird` and `(011) 123-4567`.

8.3 XML AND HTML

HTML is about displaying information, whereas XML is about carrying information. In other words, XML was created to structure, store, and transport information. HTML was designed to display the data.

- Using XML, we can create own tags where as in HTML it is not possible instead it offers several built in tags.
- XML is platform independent neutral and language independent.
- XML tags and attribute names are case sensitive where as in HTML it is not.
- XML attribute values must be single or double quoted where as in HTML it is not compulsory
- XML elements must be properly nested
- All XML elements must have a closing tag
- XML is used to create new internet languages. Here are some examples:
 - WSDL for describing available web services
 - WAP and WML as markup languages for handheld devices
 - RSS languages for news feeds
 - RDF and OWL for describing resources and ontology
 - SMIL for describing multimedia for the web

8.4 DOCUMENTS AND VOCABULARIES

An XML *document* is a basic unit of XML information composed of elements and other markup in an orderly package. An XML *document* can contain wide variety of data. For example, database of numbers, numbers representing molecular structure or a mathematical equation.

An XML document consists of two parts namely

- ✓ Document Prolog
- ✓ Document Elements

Document Prolog Section

Document Prolog comes at the top of the document, before the root element. This section contains

- XML declaration
- Document type declaration

Document Elements Section

Document Elements are the building blocks of XML. These divide the document into a hierarchy of sections, each serving a specific purpose. You can separate a document into multiple sections so that they can be rendered differently, or used by a search engine. The elements can be containers, with a combination of text and other elements.

For the Example 8.1

```

<?xml version = "1.0"?>
<contact-info>
<name>Pragati</name>
<company>Hummingbird</company>
<phone>(011) 123-4567</phone>
</contact-info>

```

} **Document Prolog**

} **Document Elements**

8.5 VERSIONS AND DECLERATIONS

The XML document can optionally have an XML declaration. It is written as follows.

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

Where version is the XML version and encoding specifies the character encoding used in the document.

Syntax Rules for XML Declaration

- The XML declaration is case sensitive and must begin with "<?xml>" where "xml" is written in lower-case.
- If document contains XML declaration, then it strictly needs to be the first statement of the XML document.
- The XML declaration strictly needs be the first statement in the XML document.
- An HTTP protocol can override the value of *encoding* that is given in the XML declaration.

Notes

Tags and Elements

An XML file is structured by several XML elements also called XML nodes or XML tags. The names of XML elements are enclosed in triangular brackets <> as shown below.

```
<element>
```

Syntax Rules for Tags and Elements

Each XML element needs to be closed either with start or with end elements as shown below.

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

or in simple-cases,

```
<element/>
```

Nesting of Elements

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

Example 8.2 – Incorrect Nesting of Elements

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

Example 8.3 – Correct Nesting of Elements

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

Notes

Root Element

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

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

The Following example shows a correctly formed XML document

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

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

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

XML Attributes

An attribute specifies a single property for the element, using a name/value pair. An XML element can have one or more attributes.

```
<a href = "http://www.demo.com/">Demo..!</a>
```

Here href is the attribute name and <http://www.demo.com/> is attribute value.

Syntax Rules for XML Attributes

- Attribute names in XML (unlike HTML) are case sensitive. That is, *HREF* and *href* are considered two different XML attributes.
- Same attribute cannot have two values. The following example shows incorrect syntax because the attribute *b* is specified twice

```
<a b = "x" c = "y" b = "z">....</a>
```

- Attribute names are defined without quotation marks, whereas attribute values must always appear in quotation marks.
- Following example demonstrates incorrect xml syntax

```
<a b = x>....</a>
```

In the above example, the attribute value is not defined in quotation marks.

XML References

References usually allow you to add or include additional text or markup in an XML document. References always begin with the symbol "&" which is a reserved character and end with the symbol ";". XML has two types of references –

- **Entity References** – An entity reference contains a name between the start and the end delimiters. For example **&**; where *amp* is *name*. The *name* refers to a predefined string of text and/or markup.
- **Character References** – These contain references, such as **A**;, contains a hash mark (“#”) followed by a number. The number always refers to the Unicode code of a character. In this case, 65 refers to alphabet "A".

Notes

XML Text

The names of XML elements and XML attributes are case-sensitive, which means the name of start and end elements need to be written in the same case. To avoid character encoding problems, all XML files should be saved as Unicode UTF-8 or UTF-16 files.

Whitespace characters like blanks, tabs and line-breaks between XML elements and between the XML attributes will be ignored.

Some characters are reserved by the XML syntax itself. Hence, they cannot be used directly. To use them, some replacement-entities are used, which are listed below.

Not Allowed Character	Replacement Entity	Character Description
<	<	less than
>	>	greater than
&	&	ampersand
'	'	apostrophe
"	"	quotation mark

8.6 NAMESPACE

A *Namespace* is a set of unique names. Namespace is a mechanism by which element and attribute name can be assigned to a group. The Namespace is identified by URI (Uniform Resource Identifiers).

A Namespace is declared using reserved attributes. Such an attribute name must either be `xmlns` or begin with `xmlns:` shown as below.

```
<element xmlns:name = "URL">
```

Notes

Syntax

- The Namespace starts with the keyword `xmlns`.
- The word `name` is the Namespace prefix.
- The URL is the Namespace identifier.

Namespace affects only a limited area in the document. An element containing the declaration and all of its descendants are in the scope of the Namespace.

Example 8.4

```
<?xml version = "1.0" encoding = "UTF-8"?>
<cont:contact xmlns:cont =
"www.hbacademy.com/profile">
<cont:name>Pragati</cont:name>
<cont:company>Hummingbird</cont:company>
<cont:phone>(011) 123-4567</cont:phone>
</cont:contact>
```

Here, the Namespace prefix is `cont`, and the Namespace identifier (URI) as `www.hbacademy.com/profile`. This means, the element names and attribute names with the `cont` prefix (including the `contact` element), all belong to the `www.hbacademy.com/profile` namespace.

Example 8.5

```
<?xml version = "1.0"?>
<?xml-stylesheet type = "text/xsl" href =
"students.xsl"?>
<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>

<student rollno = "593">
<firstname>Jasvir</firstname>
<lastname>Singh</lastname>
```

```
<nickname>Jazz</nickname>
<marks>90</marks>
</student>

</class>
```

Notes

Check Your Progress

1. What is XML?
2. What is an XML document?
3. What is a namespace?

8.7 ANSWERS TO CHECK YOUR PROGRESS

1. XML stands for eXtensible Markup Language, developed by W3C in 1996.
2. An XML *document* is a basic unit of XML information composed of elements and other markup in an orderly package. An XML *document* can contains wide variety of data
3. A Namespace is a set of unique names. Namespace is a mechanism by which element and attribute name can be assigned to a group

8.8 LET US SUM UP

XML stands for eXtensible Markup Language, developed by W3C in 1996.

XML tags identify the data and are used to store and organize the data, rather than specifying how to display it like HTML tags, which are used to display the data.

There are three important *characteristics* of XML that make it useful in a variety of systems and solutions

- XML is extensible.
- XML carries the data, does not present it
- XML is a public standard

A short list of XML *usage* is given below

- XML can work behind the scene to simplify the creation of HTML documents for large web sites.
- XML can be used to exchange the information between organizations and systems.

Notes

- XML can be used for offloading and reloading of databases.
- XML can be used to store and arrange the data, which can customize your data handling needs.
- XML can easily be merged with style sheets to create almost any desired output.
- Virtually, any type of data can be expressed as an XML document

An XML *document* is a basic unit of XML information composed of elements and other markup in an orderly package. An XML *document* can contains wide variety of data.

An XML document consists of two parts namely

- ✓ Document Prolog
- ✓ Document Elements

An XML file is structured by several XMLelements also called XMLnodes or XMLtags.

An XML document can have only one root element.

An attribute specifies a single property for the element, using a name/value pair. An XMLelement can have one or more attributes.

References usually allow you to add or include additional text or markup in an XML document. References always begin with the symbol "&" which is a reserved character and end with the symbol ";".

XML has two types of references

- ✓ Entity References
- ✓ Character References

A Namespace is a set of unique names. Namespace is a mechanism by which element and attribute name can be assigned to a group. The Namespace is identified by URI(Uniform Resource Identifiers).

8.9 SELF-ASSESSMENT EXERCISES

Short Questions

1. What is XML?
2. What are the characteristics of XML.
3. List the uses of XML.
4. What are the two types of XML references?

Detail Questions

1. What is the difference between XML and HTML? Explain.
2. Discuss about XML tags and attributes.
3. How to define version for an XML document? Give examples.
4. Write a XML program to represent book details as web data.
5. Write a note on namespaces.

8.10 SUGGESTED READINGS

1. Web Programming: Building Internet applications, Chris Bates, Wiley India
2. Web technologies – A computer science perspective, Jeffrey C Jackson, Pearson Education, 2006
3. Robert W. Sebesta “Programming the world wide web” Pearson Education
4. Bates, Developing web applications, wiley, 2006
5. www.w3schools.com

Notes

UNIT- 9 JAVASCRIPT AND XML

Structure

Notes

- 9.0 Introduction
- 9.1 Objectives
- 9.2 Reading XML data
- 9.3 Ajax
- 9.4 DOM based XML processing
- 9.5 SAX
- 9.6 XSL, XSLT, XPATH
- 9.7 Answers to Check Your Progress
- 9.8 Let us Sum up
- 9.9 Self-Assessment Exercises
- 9.10 Suggested Readings

9.0 INTRODUCTION

XML is more suitable for some kind of data interchange but, and even if the web is based in SGML/XML standards. Despite the fact that XML is parsed by browser for a long time, the amount of tools to process XML with JavaScript is quite limited. This might be a result of the arrival of JSON and the small number of users actually needing real mixed content. This unit will discuss about the approaches for parsing XML documents

9.1 OBJECTIVES

After going through this unit, you will be able to:

- Understand how to read XML data
- Perform DOM based XML processing
- Learn about Ajax, SAX
- Know the basics of XSL, XSLT, XPATH.

9.2 READING XML DATA

Several key methods and properties in JavaScript can help in getting information from an XML file. In the section, a very simple XML file is used to demonstrate pulling data from XML into an HTML page using JavaScript to parse (interpret) the XML file.

All major browsers have a built-in XML parser to access and manipulate XML.

Example 9.1

The XML file contains a typical arrangement of data using a level of categories that you might find in a bookstore or library arrangement.

writers.xml

```
<?xml version="1.0" ?>
<writers>
<EnglishLanguage>
<fiction>
<pen>
<name>Jane Austin</name>
<name>Rex Stout</name>
<name>Dashiell Hammett</name>
</pen>
</fiction>
</EnglishLanguage>
</writers>
```

readXML.css

```
body {
  font-family:verdana;
  color:#ff4d00;
  font-size:14pt;
  font-weight:bold;
  background-color:#678395;
}
div {background-color:#c1d4cc;}
#blueBack {background-color:#c1d4cc}
```

readNode.html

```
<html>
<head>
<link rel="stylesheet" href="readXML.css"
type="text/css">
<title>
Read the whole list
</title>
<xml ID="writersXML"
SRC="writers.xml"></xml>
<script language="JavaScript">
function findWriters() {
  var myXML, myNodes;
  var display="";
  myXML= document.all("writersXML").XMLDocument;
  //Put the <name> element into an object.
  myNodes=myXML.getElementsByTagName("name");
  //Extract the different values using a loop.
```

Notes

Notes

```

        for(var
counter=0;counter<myNodes.length;counter++) {
            display +=
myNodes.item(counter).firstChild.nodeValue +
"\n";
        }
        document.show.me.value=display;
    }
</script>
</head>
<body>
<span ID="blueBack">
Read All Data
</span>
<div>
<form name="show">
<textarea name="me" cols=30
rows=5></textarea><p>
<input type="button" value="Show all"
onClick="findWriters()">
</form>
</div>
</body>
</html>

```

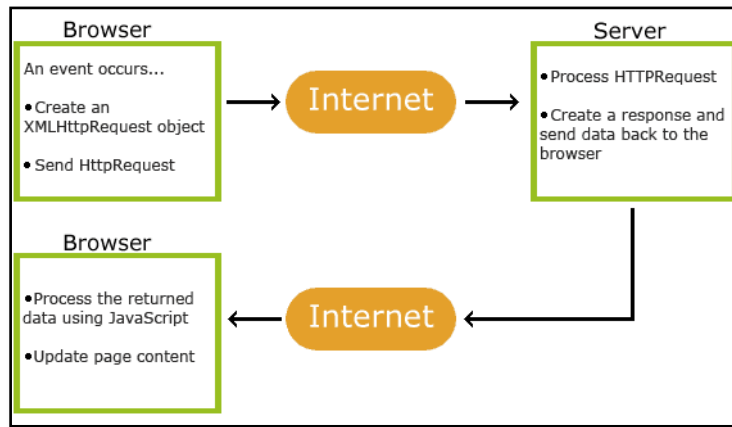
9.3AJAX

AJAX stands for **A**synchronous **J**avaScript **A**nd **X**ML. AJAX is not a programming language. AJAX just uses a combination of:

- A browser built-in XMLHttpRequest object (to request data from a web server)
- JavaScript and HTML DOM (to display or use the data)

AJAX allows web pages to be updated asynchronously by exchanging data with a web server behind the scenes. This means that it is possible to update parts of a web page, without reloading the whole page.

1. An event occurs in a web page (the page is loaded, a button is clicked)
 2. An XMLHttpRequest object is created by JavaScript
 3. The XMLHttpRequest object sends a request to a web server
 4. The server processes the request
 5. The server sends a response back to the web page
 6. The response is read by JavaScript
 7. Proper action (like page update) is performed by JavaScript
-



Notes

Example 9.2

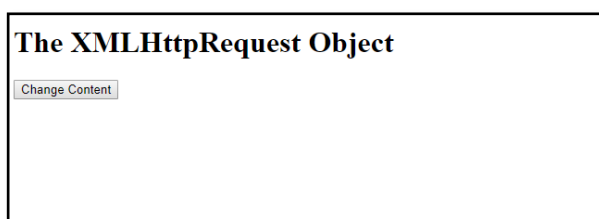
```

<!DOCTYPE html>
<html>
<body>

<div id="demo">
<h1>The XMLHttpRequest Object</h1>
<button type="button" onclick="loadDoc()">Change
Content</button>
</div>

<script>
function loadDoc() {
  var xhttp = new XMLHttpRequest();
  xhttp.onreadystatechange = function() {
    if (this.readyState == 4 && this.status ==
200) {
      document.getElementById("demo").innerHTML =
this.responseText;
    }
  };
  xhttp.open("GET", "ajax_info.txt", true);
  xhttp.send();
}
</script>
</body>
</html>

```

Output – Initial Screen

After clicking the “Change Content” Button**AJAX**

AJAX is not a programming language.

AJAX is a technique for accessing web servers from a web page.

AJAX stands for Asynchronous JavaScript And XML.

AJAX is usually called as a developer's dream, because we can:

- Update a web page without reloading the page
- Request data from a server - after the page has loaded
- Receive data from a server - after the page has loaded
- Send data to a server - in the background

AJAX can be used for interactive communication with an XML file.

Example 9.3***cd_catalog.xml***

```
<?xml version="1.0" ?>
<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>
<CD>
<TITLE>Greatest Hits</TITLE>
<ARTIST>Dolly Parton</ARTIST>
<COUNTRY>USA</COUNTRY>
<COMPANY>RCA</COMPANY>
<PRICE>9.90</PRICE>
<YEAR>1982</YEAR>
</CD>
</CATALOG>
```

catalog.html

```

<!DOCTYPE html>
<html>
<style>
table,th,td {
  border : 1px solid black;
  border-collapse: collapse;
}
th,td {
  padding: 5px;
}
</style>
<body>

<h1>The XMLHttpRequest Object</h1>

<button type="button" onclick="loadDoc()">Get my
CD collection</button>
<br><br>
<table id="demo"></table>

<script>
function loadDoc() {
  var xhttp = new XMLHttpRequest();
  xhttp.onreadystatechange = function() {
    if (this.readyState == 4 && this.status ==
200) {
      myFunction(this);
    }
  };
  xhttp.open("GET", "cd_catalog.xml", true);
  xhttp.send();
}
function myFunction(xml) {
  var i;
  var xmlDoc = xml.responseXML;
  var
table="<tr><th>Artist</th><th>Title</th></tr>";
  var x = xmlDoc.getElementsByTagName("CD");
  for (i = 0; i <x.length; i++) {
    table += "<tr><td>" +

x[i].getElementsByTagName("ARTIST")[0].childNodes[
0].nodeValue +
    "</td><td>" +

x[i].getElementsByTagName("TITLE")[0].childNodes[0
].nodeValue +
    "</td></tr>";
  }
}

```

Notes

Notes

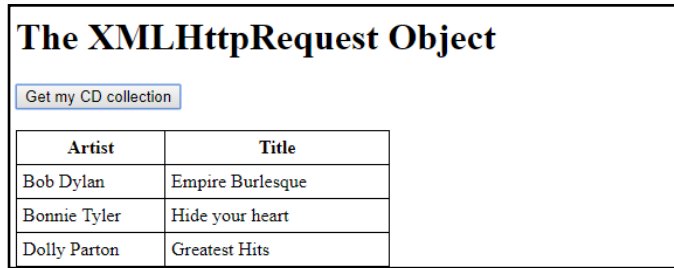
```

    document.getElementById("demo").innerHTML =
table;
}
</script>

</body>
</html>

```

Output



9.4 DOM BASED XML PROCESSING

The XML DOM (Document Object Model) defines the properties and methods for accessing and editing XML. However, before an XML document can be accessed, it must be loaded into an XML DOM object. All modern browsers have a built-in XML parser that can convert text into an XML DOM object.

An XML DOM parser is created using the following statement

```

parser = new DOMParser();

```

The parser creates a new XML DOM object using the text string:

```

xmlDoc = parser.parseFromString(text, "text/xml");

```

Example 9.4

```

<!DOCTYPE html>
<html>
<body>
<p id="demo"></p>

<script>
var parser, xmlDoc;
var text = "<bookstore><book>" +
"<title>Everyday Italian</title>" +
"<author>Giada De Laurentiis</author>" +

```



```

"<year>2005</year>" +
"</book></bookstore>";

parser = new DOMParser();
xmlDoc = parser.parseFromString(text,"text/xml");

document.getElementById("demo").innerHTML =
xmlDoc.getElementsByTagName("title")[0].childNodes
[0].nodeValue;
</script>

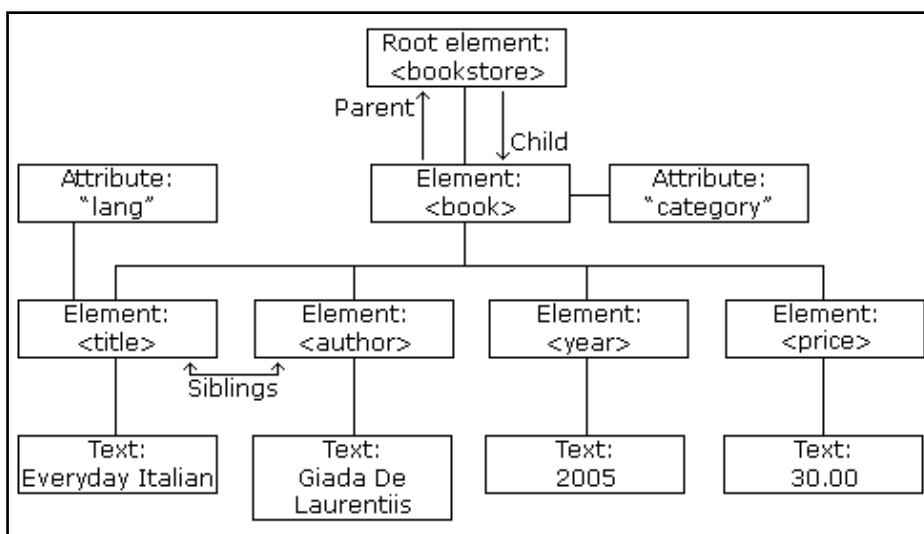
</body>
</html>

```

Notes

Output

Everyday Italian

DOM Tree for the Bookstore xml data:**9.5SAX**

SAX (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.

Notes

- 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.

Uses a SAX parser

- We can process the XML document in a linear fashion from top to down.
- The document is not deeply nested.
- You are processing a very large XML document whose DOM tree would consume too much memory. Typical DOM implementations use ten bytes of memory to represent one byte of XML.
- The problem to be solved involves only a part of the XML document.
- Data is available as soon as it is seen by the parser, so SAX works well for an XML document that arrives over a stream.

Disadvantages of SAX

- We have no random access to an XML document since it is processed in a forward-only manner.
- If you need to keep track of data that the parser has seen or change the order of items, you must write the code and store the data on your own.

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.

Notes

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 9.5

input.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>

<student rollno = "593">
<firstname>jasvir</firstname>
<lastname>singn</lastname>
```

Notes

```
<nickname>jazz</nickname>
<marks>90</marks>
</student>
</class>
```

UserHandler.java

```
package com.aludde.xml;

import org.xml.sax.Attributes;
import org.xml.sax.SAXException;
import org.xml.sax.helpers.DefaultHandler;

public class UserHandler extends DefaultHandler {

    boolean bFirstName = false;
    boolean bLastName = false;
    boolean bNickName = false;
    boolean bMarks = false;

    @Override
    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;
        }
    }

    @Override
    public void endElement(String uri,
        String localName, String qName) throws
SAXException {
        if (qName.equalsIgnoreCase("student")) {
```

```

        System.out.println("End Element : " +
qName);
    }
}

@Override
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;
    }
}
}
}

```

*Notes****SAXParserDemo.java***

```

package com.aludde.xml;

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.xml");
            SAXParserFactory factory =
SAXParserFactory.newInstance();

```

Notes

```

        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;

    @Override
    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;
        }
    }

    @Override
    public void endElement(String uri,
        String localName, String qName) throws
SAXException {

        if (qName.equalsIgnoreCase("student")) {

```

```

        System.out.println("End Element : " +
qName);
    }
}

@Override
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;
    }
}
}
}

```

*Notes***Output**

```

Roll No : 393
First Name: dinkar
Last Name: kad
Nick Name: dinkar
Marks: 85
End Element :student
Roll No : 493
First Name: Vaneet
Last Name: Gupta
Nick Name: vinni
Marks: 95
End Element :student
Roll No : 593
First Name: jasvir
Last Name: singn
Nick Name: jazz
Marks: 90
End Element :student

```

Notes

9.6 XSL, XSLT, XPATH

Before learning XSLT, we should first understand XSL which stands for **EX**tensible **S**tylesheet **L**anguage. It is similar to XML as CSS is to HTML.

Need for XSL

In case of HTML document, tags are predefined such as **table**, **div**, and **span**; and the browser knows how to add style to them and display those using CSS styles. But in case of XML documents, tags are not predefined. In order to understand and style an XML document, World Wide Web Consortium (W3C) developed XSL which can act as XML based Stylesheet Language. An XSL document specifies how a browser should render an XML document.

The Extensible Stylesheet Language (XSL) has three major subcomponents:

- **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.

XSLT

XSLT, Extensible Stylesheet Language Transformations, provides the ability to transform XML data from one format to another automatically.

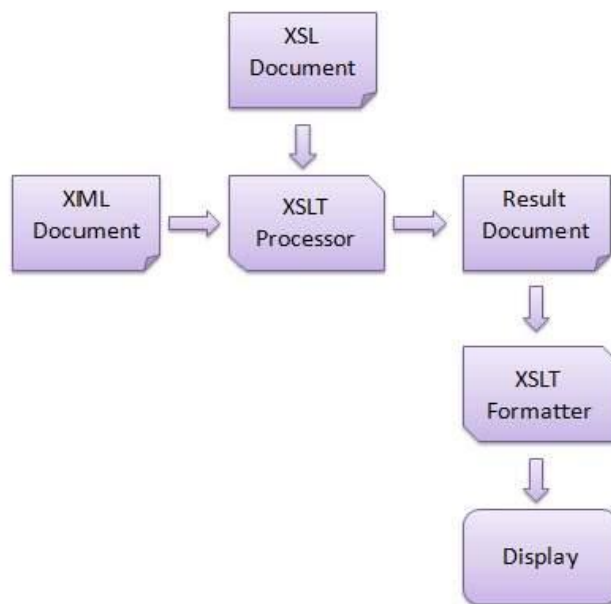


Figure 9.1. Working of XSLT

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 applies the transformation rules on the target XML document and then it generates a formatted document in the form of XML, HTML, or text format. This formatted document is then utilized by XSLT formatter to generate the actual output which is to be displayed to the end-user.

Notes

Advantages

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

Example 9.6

```
<?xml version="1.0" encoding="UTF-8"?>
<breakfast_menu>

<food>
<name>Belgian Waffles</name>
<price>$5.95</price>
<description>Two of our famous Belgian Waffles
with plenty of real maple syrup</description>
<calories>650</calories>
</food>

<food>
<name>Strawberry Belgian Waffles</name>
<price>$7.95</price>
<description>Light Belgian waffles covered with
strawberries and whipped cream</description>
<calories>900</calories>
</food>

<food>
<name>Berry-Berry Belgian Waffles</name>
<price>$8.95</price>
<description>Light Belgian waffles covered with an
assortment of fresh berries and whipped
cream</description>
<calories>900</calories>
</food>

<food>
<name>French Toast</name>
<price>$4.50</price>
<description>Thick slices made from our homemade
sourdough bread</description>
```

Notes

```

<calories>600</calories>
</food>

<food>
<name>Homestyle Breakfast</name>
<price>$6.95</price>
<description>Two eggs, bacon or sausage, toast,
and our ever-popular hash browns</description>
<calories>950</calories>
</food>

</breakfast_menu>

```

Output before applying XSLT

```

Belgian Waffles $5.95 Two of our famous Belgian Waffles with plenty of real maple syrup 650
Strawberry Belgian Waffles $7.95 Light Belgian waffles covered with strawberries and
whipped cream 900 Berry-Berry Belgian Waffles $8.95 Light Belgian waffles covered with an
assortment of fresh berries and whipped cream 900 French Toast $4.50 Thick slices made from
our homemade sourdough bread 600 Homestyle Breakfast $6.95 Two eggs, bacon or sausage,
toast, and our ever-popular hash browns 950

```

Using XSLT to transform XML into HTML

```

<?xml version="1.0" encoding="UTF-8"?>
<html xsl:version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
<body style="font-family:Arial;font-size:12pt;background-color:#EEEEEE">

<xsl:for-each select="breakfast_menu/food">

<div style="background-color:teal;color:white;padding:4px">

<span style="font-weight:bold"><xsl:value-of select="name"/> - </span>

<xsl:value-of select="price"/>

</div>
<div style="margin-left:20px;margin-bottom:1em;font-size:10pt">

<p>
    <xsl:value-of select="description"/>
    <span style="font-style:italic"> (<xsl:value-of select="calories"/> calories per

```

```

serving)</span>
</p>
</div>
</xsl:for-each>
</body>
</html>

```

Notes

Output after applying XSLT

Belgian Waffles - \$5.95
Two of our famous Belgian Waffles with plenty of real maple syrup (650 calories per serving)
Strawberry Belgian Waffles - \$7.95
Light Belgian waffles covered with strawberries and whipped cream (900 calories per serving)
Berry-Berry Belgian Waffles - \$8.95
Light Belgian waffles covered with an assortment of fresh berries and whipped cream (900 calories per serving)
French Toast - \$4.50
Thick slices made from our homemade sourdough bread (600 calories per serving)
Homestyle Breakfast - \$6.95
Two eggs, bacon or sausage, toast, and our ever-popular hash browns (950 calories per serving)

XPATH

XPath is an official recommendation of the World Wide Web Consortium (W3C). It defines a language to find information in an XML file. It is used to traverse elements and attributes of an XML document. XPath provides various types of expressions which can be used to enquire relevant information from the XML document.

- **Structure Definitions** – XPath defines the parts of an XML document like element, attribute, text, namespace, processing-instruction, comment, and document nodes
- **Path Expressions** – XPath provides powerful path expressions select nodes or list of nodes in XML documents.
- **Standard Functions** – XPath provides a rich library of standard functions for manipulation of string values, numeric values, date and time comparison, node and QName manipulation, sequence manipulation, Boolean values etc.
- **Major part of XSLT** – XPath is one of the major elements in XSLT standard and is must have knowledge in order to work with XSLT documents.
- **W3C recommendation** – XPath is an official recommendation of World Wide Web Consortium (W3C).

Points to remember

- XPath is core component of XSLT standard.
- XSLT cannot work without XPath.
- XPath is basis of XQuery and XPointer.

Notes

An XPath expression generally defines a pattern in order to select a set of nodes. These patterns are used by XSLT to perform transformations or by XPointer for addressing purpose.

XPath specification specifies seven types of nodes which can be the output of execution of the XPath expression.

- Root
- Element
- Text
- Attribute
- Comment
- Processing Instruction
- Namespace

XPath uses a path expression to select node or a list of nodes from an XML document.

Following is the list of useful paths and expression to select any node/ list of nodes from an XML document.

S.No.	Expression & Description
1	node-name Select all nodes with the given name "nodename"
2	/ Selection starts from the root node
3	// Selection starts from the current node that match the selection
4	. Selects the current node
5	.. Selects the parent of the current node
6	@ Selects attributes
7	student Example – Selects all nodes with the name "student"
8	class/student Example – Selects all student elements that are children of class
9	//student Selects all student elements no matter where they are in the document

Example 9.7

In this example, we've created a sample XML document, students.xml and its stylesheet document **students.xsl** which uses the XPath expressions under **select** attribute of various XSL tags to get the values of roll no, firstname, lastname, nickname and marks of each student node.

students.xml

```

<?xml version = "1.0"?>
<?xml-stylesheet type = "text/xsl" href =
"students.xsl"?>
<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>
<student rollno = "593">
<firstname>Jasvir</firstname>
<lastname>Singh</lastname>
<nickname>Jazz</nickname>
<marks>90</marks>
</student>
</class>

```

*Notes**students.xsl*

```

<?xml version = "1.0" encoding = "UTF-8"?>
<xsl:stylesheet version = "1.0"
  xmlns:xsl = "http://www.w3.org/1999/XSL/Transform">
<xsl:template match = "/">
<html>
<body>
<h2>Students</h2>
<table border = "1">
<tr bgcolor = "#9acd32">
<th>Roll No</th>
<th>First Name</th>
<th>Last Name</th>
<th>Nick Name</th>
<th>Marks</th>
</tr>
<xsl:for-each select = "class/student">
<tr>
<td><xsl:value-of select = "@rollno"/></td>
<td><xsl:value-of select = "firstname"/></td>
<td><xsl:value-of select = "lastname"/></td>
<td><xsl:value-of select = "nickname"/></td>
<td><xsl:value-of select = "marks"/></td>
</tr>
</xsl:for-each>
</table>

```

```

</body>
</html>
</xsl:template>

</xsl:stylesheet>

```

Output**Students**

Roll No	First Name	Last Name	Nick Name	Marks
393	Dinkar	Kad	Dinkar	85
493	Vaneet	Gupta	Vinni	95
593	Jasvir	Singh	Jazz	90

Check Your Progress

1. Expand AJAX.
2. How to create an XML DOM parser?
3. What is SAX?
4. What are the subcomponents of XSL?

9.7 ANSWERS TO CHECK YOUR PROGRESS

1. AJAX stands for Asynchronous JavaScript And XML.
2. An XML DOM parser is created using the following statement

```
parser = new DOMParser();
```
3. SAX (Simple API for XML) is an event-based parser for XML documents. Unlike a DOM parser, a SAX parser creates no parse tree
4. The Extensible Stylesheet Language (XSL) has three major subcomponents:
 - **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

9.8 LET US SUM UP

Several key methods and properties in JavaScript can help in getting information from an XML file.

AJAX stands for **A**synchronous **J**avaScript **A**nd **X**ML.

AJAX allows web pages to be updated asynchronously by exchanging data with a web server behind the scenes. This means that it is possible to update parts of a web page, without reloading the whole page.

The XML DOM (Document Object Model) defines the properties and methods for accessing and editing XML.

An XML DOM parser is created using the following statement

```
parser = new DOMParser();
```

SAX (Simple API for XML) is an event-based parser for XML documents. Unlike a DOM parser, a SAX parser creates no parse tree.

XSL stands for **E**Xtensible **S**tylesheet **L**anguage.

The Extensible Stylesheet Language (XSL) has three major subcomponents:

- **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.

9.9 SELF-ASSESSMENT EXERCISES

Short Questions

1. List the uses of SAX
2. What is the need for XSL?
3. State the advantages of XSLT.

Detail Questions

1. Explain about the XSL with suitable example.
2. Describe the working of XSLT.
3. Discuss in detail about XPATH

Notes

9.10 SUGGESTED READINGS

1. Web Programming: Building Internet applications, Chris Bates, Wiley India
2. Web technologies – A computer science perspective, Jeffrey C Jackson, Pearson Education, 2006
3. Robert W. Sebesta “Programming the world wide web” Pearson Education
4. Bates, Developing web applications, wiley, 2006
5. www.w3schools.com

BLOCK – IV

SERVER SIDE PROGRAMMING

Notes

UNIT- 10 JAVA SERVLETS

Structure

- 10.0 Introduction
- 10.1 Objectives
- 10.2 History of web applications
- 10.3 The power of Servlets
- 10.4 HTTP Servlet basics
- 10.5 The Servlet API
- 10.6 Page Generations
- 10.7 Answers to Check Your Progress
- 10.8 Let us Sum up
- 10.9 Self-Assessment Exercises
- 10.10 Suggested Readings

10.0 INTRODUCTION

The rise of server-side Java applications is one of the latest and most exciting trends in Java programming. Java servlets are a key component of server-side Java development. A servlet is a small, pluggable extension to a server that enhances the server's functionality. Servlets allow developers to extend and customize any Java-enabled server - a web server, a mail server, an application server, or any custom server - with a previously unknown degree of portability, flexibility, and ease.

10.1 OBJECTIVES

After going through this unit, you will be able to:

- Understand the history of web applications
- Learn the basics of Java Servlets
- Develop a Servlet API

10.2 HISTORY OF WEB APPLICATIONS

While servlets can be used to extend the functionality of any Java-enabled server, they are most often used to extend web servers, providing a powerful, efficient replacement for CGI scripts. When you use a servlet to create dynamic content for a web page or otherwise extend the functionality of a web server, you are in effect creating a web application.

While a web page merely displays static content and lets the user navigate through that content, a web application provides a more interactive experience. A web application can be as simple as a keyword search on a document archive or as complex as an electronic storefront. Web applications are being deployed on the Internet and on corporate intranets and extranets, where they have the potential to increase productivity and change the way that companies, large and small, do business.

To understand the power of servlets, we need to step back and look at some of the other approaches that can be used to create web applications.

Common Gateway Interface

The Common Gateway Interface, normally referred to as CGI, was one of the first practical techniques for creating dynamic content. With CGI, a web server passes certain requests to an external program. The output of this program is then sent to the client in place of a static file. The advent of CGI made it possible to implement all sorts of new functionality in web pages, and CGI quickly became a de facto standard, implemented on dozens of web servers.

It's interesting to note that the ability of CGI programs to create dynamic web pages is a side effect of its intended purpose: to define a standard method for an information server to talk with external applications. This origin explains why CGI has perhaps the worst life cycle imaginable. When a server receives a request that accesses a CGI program, it must create a new process to run the CGI program and then pass to it, via environment variables and standard input, every bit of information that might be necessary to generate a response. Creating a process for every such request requires time and significant server resources, which limits the number of requests a server can handle concurrently. Figure 10.1 shows the CGI life cycle. Even though a CGI program can be written in almost any language, the Perl programming language has become the predominant choice.

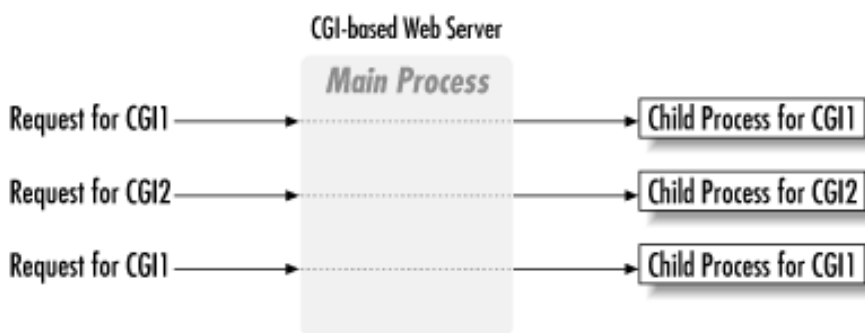


Figure 10.1 The CGI life cycle

Another often-overlooked problem with CGI is that a CGI program cannot interact with the web server or take advantage of the server's abilities once it begins execution, because it is running in a separate process. For example, a CGI script cannot write to the server's log file.

Other Solutions

CGI/Perl has the advantage of being a more-or-less platform-independent way to produce dynamic web content. Other well-known technologies for creating web applications, such as ASP and server-side JavaScript, are proprietary solutions that work only with certain web servers.

Java Servlets

A Servlet is a generic server extension - a Java class that can be loaded dynamically to expand the functionality of a server. Servlets are commonly used with web servers, where they can take the place of CGI scripts. A servlet is similar to a proprietary server extension, except that it runs inside a Java Virtual Machine (JVM) on the server (Figure 10.2), so it is safe and portable. Servlets operate solely within the domain of the server: unlike applets, they do not require support for Java in the web browser.

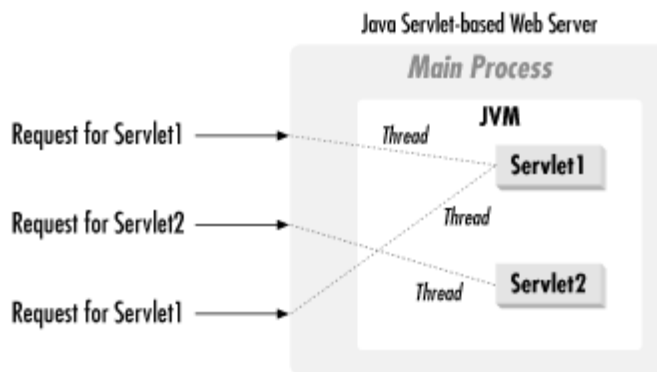


Figure 10.2 The Servlet life cycle

Unlike CGI and FastCGI, which must use multiple processes to handle separate programs and/or separate requests, servlets can all be handled by separate threads within the same process or by threads within multiple processes spread across a number of backend servers.

This means that servlets are also *efficient and scalable*. Because servlets run with bidirectional communication to the web server, they *can interact very closely with the server* to do things that are not possible with CGI scripts. Another advantage of servlets is that they are *portable*: both across operating systems as we are used to with Java and also across web servers.

10.3 THE POWER OF SERVLETS

The Servlets offer a number of advantages over other approaches. They are listed below.

Portability

Because servlets are written in Java and conform to a well-defined and widely accepted API, they are highly portable across operating systems and across server implementations. With servlets, you can truly "write once, serve everywhere."

Power

Servlets can harness the full power of the core Java APIs: networking and URL access, multithreading, image manipulation, datacompression, database connectivity (JDBC), object serialization, internationalization, remote method invocation (RMI), and legacyintegration (CORBA). Servlets can also take advantage of the J2EE platform that includes support for Enterprise JavaBeans (EJBs), distributed transactions (JTS), standardized messaging (JMS), directory lookup (JNDI), and advanced database access(JDBC 2.0).

Efficiency and Endurance

Servlet invocation is highly efficient. Once a servlet is loaded, it remains in the server's memory as a single object instance. Thereafter, the server invokes the servlet to handle a request using a simple, lightweight method invocation. Servlets are naturally enduring objects. Because a servlet stays in the server's memory as a single object instance, it automatically maintains its state and can hold on to external resources, such as database connections.

Safety

Servlets support safe programming practices on a number of levels. Because they are written in Java, servlets inherit the strong typesafety of the Java language. In addition, the Servlet API is implemented to be type-safe.

Elegance

The elegance of servlet code is striking. Servlet code is clean, object oriented, modular, and amazingly simple.

Integration

Servlets are tightly integrated with the server. This integration allows a servlet to cooperate with the server in ways that a CGIprogram cannot.

Extensibility and Flexibility

The Servlet API is designed to be easily extensible. As it stands today, the API includes classes with specialized support for HTTPservlets. But at a later date, it could be extended and optimized for another type of servlets, either by Sun or by a third party. Servlets are also quite flexible in how they create content. They can generate simple content using `out.println()` statements, orthey can generate complicated sets of pages using a template engine. Servlets can even be builtupon to create brand new technologies like JavaServer Pages.

10.4HTTP SERVLET BASICS

HTTP is a simple, stateless protocol. A *client*, such as a web browser, *makes a request*, the web *server* *responds*, and the transaction is done. When the client sends a request, the first thing it specifies is an HTTP command, called a *method* that tells the server the type of action it wants performed.

When a client connects to a server and makes an HTTP request, the request can be of several different types, called methods. The most frequently used methods are GET and POST. Put simply, the GET method is designed for getting information (a document, a chart, or the results from a database query), while the POST method is designed for posting information (a credit card number, some new chart data, or information that is to be stored in a database). Simply

- GET is for reading
- POST is for tacking up new material

The GET method, although it's designed for reading information, can include as part of the request some of its own information that better describes what to get—such as an x, y scale for a dynamically created chart. This information is passed as a sequence of characters appended to the request URL in what's called a *query string*.

The POST method uses a different technique to send information to the server because in some cases it may need to send megabytes of information. A POST request passes all its data, of unlimited length, directly over the socket connection as part of its HTTP request body. The exchange is invisible to the client. The URL doesn't change at all.

In addition to GET and POST, there are several other lesser-used HTTP methods. There's the HEAD method, which is sent by a client when it wants to see only the headers of the response, to determine the document's size, modification time, or general availability. There's also PUT, to place documents directly on the server, and DELETE, to do just the opposite. These last two aren't widely supported due to complicated policy issues.

10.5 THE SERVLET API

Servlets use classes and interfaces from two packages: `javax.servlet` and `javax.servlet.http`. The `javax.servlet` package contains classes and interfaces to support generic, protocol independent servlets. These classes are extended by the classes in the `javax.servlet.http` package to add HTTP-specific functionality.

Every servlet must implement the `javax.servlet.Servlet` interface. Most servlets implement this interface by extending one of two special classes:

```
javax.servlet.GenericServlet  
or  
javax.servlet.http.HttpServlet.
```

A protocol-independent servlet should subclass `GenericServlet`, while an HTTP servlet should subclass `HttpServlet`, which is itself a subclass of `GenericServlet` with added HTTP-specific functionality.

Notes

Unlike a regular Java program, and just like an applet, a servlet does not have a `main()` method. Instead, certain methods of a servlet are invoked by the server in the process of handling requests. Each time the server dispatches a request to a servlet, it invokes the servlet's `service()` method.

A generic servlet should override its `service()` method to handle requests as appropriate for the servlet. The `service()` method accepts two parameters: a request object and a response object. The request object tells the servlet about the request, while the response object is used to return a response. Figure 10.3 shows how a generic servlet handles requests.

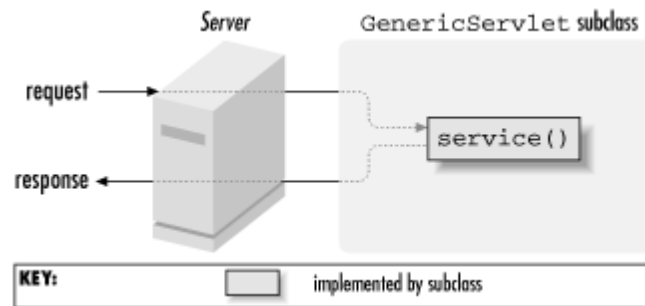


Figure 10.3A generic servlet handling a request

10.6 PAGE GENERATIONS

The most basic type of HTTP servlet generates a full HTML page. Such a servlet has access to the same information usually sent to a CGI script, plus a bit more. A servlet that generates an HTML page can be used for all the tasks for which CGI is used currently, such as for processing HTML forms, producing reports from a database, taking orders, checking identities, and so forth.

Example 10.1 : A Servlet That Prints "Hello World"

```
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;
public class HelloWorld extends HttpServlet
{
    public void doGet(HttpServletRequest req,
        HttpServletResponse res)
        throws ServletException, IOException {
        res.setContentType("text/html");
        PrintWriter out = res.getWriter();
        out.println("<HTML>");
        out.println("<HEAD><TITLE>Hello
World</TITLE></HEAD>");
        out.println("<BODY>");
        out.println("<BIG>Hello World</BIG>");
        out.println("</BODY></HTML>");
    }
}
```

```
}
}
```

Check Your Progress

1. List the advantages of Servlets
2. What are the common methods used to process requests?
3. What are the classes used to implement Servlet interface?

Notes

10.7 ANSWERS TO CHECK YOUR PROGRESS

1. Advantages of using Servlets
 - Portability
 - Power
 - Efficiency and Endurance
 - Safety
 - Elegance
 - Integration
 - Extensibility and Flexibility.
2. The most frequently used methods are GET and POST.
 - GET is for reading
 - POST is for tacking up new material
3. Most servlets implement its interface by extending one of two special classes:
`javax.servlet.GenericServlet(or)javax.servlet.http.HttpServlet`

10.8 LET US SUM UP

A servlet is a small, pluggable extension to a server that enhances the server's functionality.

The Common Gateway Interface, normally referred to as CGI, was one of the first practical techniques for creating dynamic content

Servlets are commonly used with web servers, where they can take the place of CGI scripts

Advantages of using Servlets

- Portability
- Power
- Efficiency and Endurance

Notes

- Safety
- Elegance
- Integration
- Extensibility and Flexibility

A *client*, such as a web browser, *makes a request*, the web *server* responds, and the transaction is done.

When the client sends a request, the first thing it specifies is an HTTP command, called a *method* that tells the server the type of action it wants performed.

The most frequently used methods are GET and POST.

- GET is for reading
- POST is for tacking up new material

A sequence of characters appended to the request URL is called a *query string*

Servlets use classes and interfaces from two packages: `javax.servlet` and `javax.servlet.http`.

The most basic type of HTTP servlet generates a full HTML page.

10.9 SELF-ASSESSMENT EXERCISES

Short Questions

1. What is CGI?
2. What do you mean by a method?
3. What is a query string?
4. List the packages for Servlets.

Detail Questions

1. Write a note on history of web applications.
2. Discuss about the power of Servlets.
3. Write a simple Servlet to display “Hello World”. Explain the code.

10.10 SUGGESTED READINGS

1. Web Programming: Building Internet applications, Chris Bates, Wiley India
2. Web technologies – A computer science perspective, Jeffrey C Jackson, Pearson Education, 2006
3. Robert W. Sebesta “Programming the world wide web” Pearson Education
4. Bates, Developing web applications, wiley, 2006
5. Java servlet Programming, Josen Hunter, o’Reilly, 2010

UNIT- 11 SERVLET LIFE CYCLE

Structure

- 11.0 Introduction
- 11.1 Objectives
- 11.2 The Servlet alternative
- 11.3 Servlet Reloading
- 11.4 Init And Destroy
- 11.5 Single Thread Model
- 11.6 Background Processing
- 11.7 Load On Startup
- 11.8 Client Side Caching and Server Side Caching
- 11.9 Answers to Check Your Progress
- 11.10 Let us Sum up
- 11.11 Self-Assessment Exercises
- 11.12 Suggested Readings

Notes

11.0 INTRODUCTION

The servlet life cycle is one of the most exciting features of servlets. This life cycle is a powerful hybrid of the life cycles used in CGI programming and lower-level NSAPI and ISAPI programming. ISAPI (Internet Server API) is a Microsoft API, and provides speed improvements over CGI programs. NSAPI (Netscape Server API) is a Netscape API, provides speed improvements over CGI programs.

11.1 OBJECTIVES

After going through this unit, you will be able to:

- Understand the Servlet life cycle
- Learn about Single thread model
- Know the basics of Client side and Server side caching

11.2 THE SERVLET ALTERNATIVE

The servlet life cycle allows servlet engines to address both the performance and resource problems of CGI and the security concerns of low-level server API programming. A servlet engine may execute all its servlets in a single Java virtual machine (JVM). Because they are in the same JVM, servlets can efficiently share data with each other, yet they are prevented by the Java language from accessing one another's private data. Servlets may also be allowed to persist between requests as object instances, taking up far less memory than full-fledged processes.

Notes

Servers have significant flexibility in how they choose to support servlets. The only hard and fast rule is that a servlet engine must conform to the following lifecycle contract:

1. *Create and initialize the servlet.*
2. *Handle zero or more service calls from clients.*
3. *Destroy the servlet and then garbage collects it.*

It's perfectly legal for a servlet to be loaded, created and instantiated in its own JVM, only to be destroyed and garbage collected without handling any client requests or after handling just one request.

To demonstrate the servlet life cycle, we'll begin with a simple example. Example 11.1 shows a servlet that counts and displays the number of times it has been accessed.

Example 11.1 – A simple counter

```
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;
public class SimpleCounter extends HttpServlet
{
    int count = 0;

    public void doGet(HttpServletRequest req,
        HttpServletResponse res)
        throws ServletException, IOException
    {
        res.setContentType("text/plain");
        PrintWriter out = res.getWriter();
        count++;
        out.println("Since loading, this servlet has been
            accessed " +
            count + " times.");
    }
}
```

The code is simple – it just prints and increments the instance variable named `count` – but it shows the power of persistence. When the server loads this servlet, the server creates a single instance to handle every request made of the servlet. That's why this code can be so simple. The same instance variables exist between invocations and for all invocations.

From the servlet-developer's perspective, each client is another thread that calls the servlet via the `service()`, `doGet()`, or `doPost()` methods, as shown in Figure 11.1.

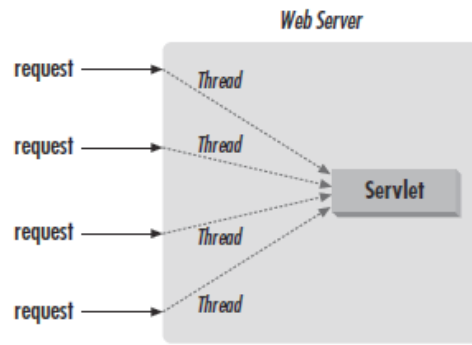


Figure 11.1 Many threads, one servlet instance

If, instead, it needed to track the count for all instances, it can in some cases use a class, or static, variable. These variables are shared across all instances of a class. Example 11.2 demonstrates with a servlet that counts three things: the times it has been accessed, the number of instances created by the server (one per name), and the total times all of them have been accessed.

Example 11.2 – A more holistic counter

```
import java.io.*;
import java.util.*;
import javax.servlet.*;
import javax.servlet.http.*;
public class HolisticCounter extends HttpServlet {
    static int classCount = 0;
    // shared by all instances
    int count = 0; // separate for each servlet
    static Hashtable instances = new Hashtable();
    public void doGet(HttpServletRequest req,
        HttpServletResponse res)
        throws ServletException, IOException {
        res.setContentType("text/plain");
        PrintWriter out = res.getWriter();
        count++;
        out.println("Since loading, this servlet instance
            has been accessed " +
            count + " times.");
        // Keep track of the instance count by putting a
        // reference to this instance in a Hashtable.
        // Duplicate entries are ignored. The size() method
        // returns the number of unique instances stored.
        instances.put(this, this);
        out.println("There are currently " +
            instances.size() + " instances.");
        classCount++;
        out.println("Across all instances, this servlet
            class has been " + "accessed " + classCount + "
            times.");
    }
}
```

Notes

11.3 SERVLET RELOADING

Servlet reloading may appear to be a simple feature, but it's actually quite a trick and requires quite a hack. `ClassLoader` objects are designed to load a class just once. To get around this limitation and load servlets again and again, servers use custom class loaders that load servlets from the default servlets directory. This explains why the servlet classes are found in `server_root/servlets`, even though that directory doesn't appear in the server's classpath.

When a server dispatches a request to a servlet, it first checks if the servlet's classfile has changed on disk. If it has changed, the server abandons the class loader used to load the old version and creates a new instance of the custom class loader to load the new version. Old servlet versions can stay in memory indefinitely, but the old versions are not used to handle any more requests.

Servlet reloading is *not* performed for classes found in the server's classpath (such as `server_root/classes`) because those classes are loaded by the core, primordial class loader. These classes are loaded once and retained in memory even when their class files change.

11.4 INIT AND DESTROY

Just like applets, servlets can define `init()` and `destroy()` methods. A servlet's `init(ServletConfig)` method is called by the server immediately after the server constructs the servlet's instance. Depending on the server and its configuration, this can be at any of these times:

- ✓ When the server starts
- ✓ When the servlet is first requested, just before the `service()` method is invoked
- ✓ At the request of the server administrator

In any case, `init()` is guaranteed to be called before the servlet handles its first request. The `init()` method is typically used to perform servlet initialization – creating or loading objects that are used by the servlet in the handling of its requests. In order to provide a new servlet any information about itself and its environment, a server had to call a servlet's `init()` method and pass along an object that implements the `ServletConfig` interface.

The server calls a servlet's `destroy()` method when the servlet is about to be unloaded. In the `destroy()` method, a servlet should free any resources it has acquired that will not be garbage collected. The `destroy()` method also gives a servlet a chance to write out its unsaved cached information or any persistent information that should be read during the next call to `init()`.

Example 11.3 –Servlet Life cycle example

```

// Import required java libraries
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;

// Extend HttpServlet class
public class HelloWorld extends HttpServlet {

    private String message;

    public void init() throws ServletException {
        // Do required initialization
        message = "Hello World";
    }

    public void doGet(HttpServletRequest request,
        HttpServletResponse response)
        throws ServletException, IOException {

        // Set response content type
        response.setContentType("text/html");

        // Actual logic goes here.
        PrintWriter out = response.getWriter();
        out.println("<h1>" + message + "</h1>");
    }

    public void destroy() {
        // do nothing.
    }
}

```

Notes

11.5 SINGLE THREAD MODEL

Although it is standard to have one servlet instance per registered servlet name, it is possible for a servlet to elect instead to have a pool of instances created for each of its names, all sharing the duty of handling requests. Such servlets indicate this desire by implementing the `javax.servlet.SingleThreadModel` interface. This is an empty, tag interface that defines no methods or variables and serves only to flag the servlet as wanting the alternate life cycle.

A server that loads a `SingleThreadModel` servlet must guarantee, according to the Servlet API documentation, “*that no two threads will execute concurrently the service method of that servlet*”. To accomplish this, each thread uses a free servlet instance from the pool, as shown in Figure 11.2. Thus, any servlet implementing `SingleThreadModel` can be considered thread safe and isn’t required to synchronize access to its instance variables.

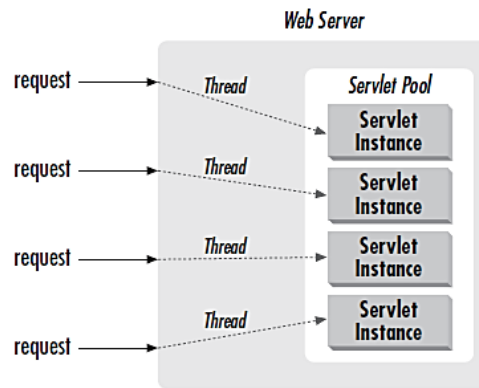


Figure 11.2 The Single Thread Model

The life cycle can be useful, however, in avoiding synchronization while still performing efficient request handling.

11.6 BACKGROUND PROCESSING

Servlets can do more than simply persist between accesses. They can also execute between accesses. Any thread started by a servlet can continue executing even after the response has been sent. This ability proves most useful for long-running tasks whose incremental results should be made available to multiple clients. A background thread started in `init()` performs continuous work while request handling threads display the current status with `doGet()`. It's a similar technique to that used in animation applets, where a single thread changes the picture and another paints the display.

Example 11.4 shows a servlet that searches for prime numbers above one quadrillion.

Example 11.4 – On the Hunt for Primes

```
import java.io.*;
import java.util.*;
import javax.servlet.*;
import javax.servlet.http.*;
public class PrimeSearcher extends HttpServlet
implements Runnable {
    long lastprime = 0; // last prime found
    Date lastprimeModified = new Date();
    // when it was found
    Thread searcher; // background search thread
    public void init() throws ServletException {
        searcher = new Thread(this);
        searcher.setPriority(Thread.MIN_PRIORITY);
        // be a good citizen
        searcher.start();
    }
    public void run() {
```

```

// QTTTBBBMMMTTTOOO
long candidate = 1000000000000001L;
// one quadrillion and one
// Begin loop searching for primes
while (true) { // search forever
if (isPrime(candidate)) {
lastprime = candidate; // new prime
lastprimeModified = new Date();
// new "prime time"
}
candidate += 2; // evens aren't prime. Between
// candidates take a 0.2 second break. Another way
// to be a good citizen with system resources.
try {
searcher.sleep(200);
}
catch (InterruptedException ignored) { }
}
}
private static boolean isPrime(long candidate) {
// Try dividing the number by all odd numbers
// between 3 and its sqrt
long sqrt = (long) Math.sqrt(candidate);
for (long i = 3; i <= sqrt; i += 2) {
if (candidate % i == 0) return false;
// found a factor
< BACK CONTINUE >
This document is created with trial version of
CHM2PDF Pilot 2.15.72.
}
// Wasn't evenly divisible, so it's prime
return true;
}
public void doGet(HttpServletRequest req,
HttpServletRequest res)
throws ServletException, IOException {
res.setContentType("text/plain");
PrintWriter out = res.getWriter();
if (lastprime == 0) {
out.println("Still searching for first prime...");
}
else {
out.println("The last prime discovered was " +
lastprime);
out.println(" at " + lastprimeModified);
}
}
public void destroy() {
searcher.stop();
}
}

```

Notes

11.7 LOAD ON STARTUP

To have the `PrimeSearcher` start searching for primes as quickly as possible, we can configure the servlet's web application to load the servlet at server start. This is accomplished by adding the `<load-on-startup>` tag to the `<servlet>` entry of the deployment descriptor.

The tag can also contain a positive integer indicating the order in which the servlet should be loaded relative to other servlets in the context.

For example, the `web.xml` shown in Example 11.5 guarantees `first` is loaded before `second`, while `anytime` could be loaded anytime during the server startup.

Example 11.5 – A Little Servlet Parade

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!DOCTYPE web-app
PUBLIC "-//Sun Microsystems, Inc.//DTD Web
Application 2.2//EN"
"http://java.sun.com/j2ee/dtds/web-app_2_2.dtd">
<web-app>
<servlet>
<servlet-name>
first
</servlet-name>
<servlet-class>
First
</servlet-class>
<load-on-startup>10</load-on-startup>
</servlet>
<servlet>
<servlet-name>
second
</servlet-name>
<servlet-class>
Second
</servlet-class>
<load-on-startup>20</load-on-startup>
</servlet>
<servlet>
<servlet-name>
anytime
</servlet-name>
<servlet-class>
Anytime
</servlet-class>
<load-on-startup/>
</servlet>
</web-app>
```

11.8 CLIENT SIDE CACHING AND SERVER SIDE CACHING

Client side caching

Let us consider, a web browser that repeatedly accesses `PrimeSearcher` should need to call `doGet()` only after the searcher thread has found a new prime. Until that time, any call `todoGet()` just generates the same page the user has already seen, a page probably stored in the browser's cache. What's really needed is a way for a servlet to report when its output has changed. That's where the `getLastModified()` method comes in.

Notes

Most web servers, when they return a document, include as part of their response a Last-Modified header. An example Last-Modified header value might be:

```
Tue, 06-May-98 15:41:02 GMT
```

This header tells the client the time the page was last changed. That information alone is only marginally interesting, but it proves useful when a browser reloads a page.

Most web browsers, when they reload a page, include in their request an If-Modified-Since header. Its structure is identical to the Last-Modified header:

```
Tue, 06-May-98 15:41:02 GMT
```

This header tells the server the Last-Modified time of the page when it was last downloaded by the browser. The server can read this header and determine if the file has changed since the given time. If the file has changed, the server must send the newer content. If the file hasn't changed, the server can reply with a simple, short response that tells the browser the page has not changed, and it is sufficient to redisplay the cached version of the document.

Here's a `getLastModified()` method for our `PrimeSearcher` example that returns when the last prime was found:

```
public long getLastModified(HttpServletRequest req)
{
    return lastprimeModified.getTime() / 1000 * 1000;
}
```

Server side caching

The `getLastModified()` method can be used, with a little trickery, to help manage a server-side cache of the servlet's output. Servlets implementing this trick can have their output caught and cached on the server side, then automatically resent to clients as appropriate according to the servlet's `getLastModified()` method. This can greatly speed servlet

Notes

page generation, especially for servlets whose output takes a significant time to produce but changes only rarely, such as servlets that display database results.

To implement this server-side caching behavior, a servlet must:

- Extend `com.oreilly.servlet.CacheHttpServlet` instead of `HttpServlet`
- Implement `getLastModified(HttpServletRequest)` method as usual

Check Your Progress

1. List the activities involved in the servlet life cycle
2. When the `init()` method will be invoked?
3. What is the name of the function used for caching?

11.9 ANSWERS TO CHECK YOUR PROGRESS

1. The servlet engine must conform to the following life cycle contract:
 - Create and initialize the servlet.
 - Handle zero or more service calls from clients.
 - Destroy the servlet and then garbage collects it.
2. Depending on the server and its configuration, this can be at any of these times:
 - When the server starts
 - When the servlet is first requested, just before the `service()` method is invoked
 - At the request of the server administrator
3. `getLastModified()` method is a way for a servlet to report when its output has changed or retrieve the cached data.

11.10 LET US SUM UP

ISAPI (Internet Server API) is a Microsoft API, and provides speed improvements over CGI programs. NSAPI (Netscape Server API) is a Netscape API, provides speed improvements over CGI programs.

A servlet engine may execute all its servlets in a single Java virtual machine (JVM).

The servlet engine must conform to the following lifecycle contract:

- *Create and initialize the servlet.*
- *Handle zero or more service calls from clients.*
- *Destroy the servlet and then garbage collects it.*

When the server loads the servlet, the server creates a single instance to handle every request made of the servlet

When a server dispatches a request to a servlet, it first checks if the servlet's classfile has changed on disk. If it has changed, the server abandons the class loader used to load the old version and creates a new instance of the custom class loader to load the new version.

A servlet's `init(ServletConfig)` method is called by the server immediately after the server constructs the servlet's instance.

Depending on the server and its configuration, this can be at any of these times:

- ✓ When the server starts
- ✓ When the servlet is first requested, just before the `service()` method is invoked
- ✓ At the request of the server administrator

The server calls a servlet's `destroy()` method when the servlet is about to be unloaded.

A server that loads a `SingleThreadModel` servlet must guarantee, according to the Servlet API documentation, *“that no two threads will execute concurrently the service method of that servlet”*.

A background thread started in `init()` performs continuous work while request handling threads display the current status with `doGet()`.

We can configure the servlet's web application to load the servlet at server start. This is accomplished by adding the `<load-on-startup>` tag to the `<servlet>` entry of the deployment descriptor.

`getLastModified()` method is a way for a servlet to report when its output has changed.

11.11 SELF-ASSESSMENT EXERCISES

Short Questions

1. What are the functions involved in the Servlet life cycle?
2. What do you mean background processing?
3. What is the purpose of caching?

Detail Questions

1. Describe the Servlet Life Cycle.
2. State the need for Servlet reloading. Explain.
3. Discuss about the single thread model.
4. With suitable example explain about load on startup.
5. Explain about client and server side caching.

Notes

11.12 SUGGESTED READINGS

1. Web Programming: Building Internet applications, Chris Bates, Wiley India
2. Web technologies – A computer science perspective, Jeffrey C Jackson, Pearson Education, 2006
3. Robert W. Sebesta “Programming the world wide web” Pearson Education
4. Bates, Developing web applications, wiley, 2006
5. Java servlet Programming, Joson Hunter, o’Reilly, 2010, 2nd Edition

UNIT- 12 RETRIEVING INFORMATION

Structure

- 12.0 Introduction
- 12.1 Objectives
- 12.2 The Servlet
- 12.3 The Server
- 12.4 The Client
- 12.5 Answers to Check Your Progress
- 12.6 Let us Sum up
- 12.7 Self-Assessment Exercises
- 12.8 Suggested Readings

Notes

12.0 INTRODUCTION

To build a successful web application, you often need to know a lot about the environment in which it is running. You may need to find out about the server that is executing your servlets or the specifics of the client that is sending requests. And no matter what kind of environment the application is running in, you most certainly need information about the requests that the application is handling. A number of methods provide servlets access to this information. For the most part, each method returns one specific result. Compared this to the way environment variables are used to pass a CGI program its information, the servlet approach has several advantages:

- Stronger type checking
- Delayed calculation
- More interaction with the server

The following table lists each CGI environment variable and the corresponding HTTP servlet method.

CGI Environment Variable	HTTP Servlet Method
SERVER_NAME	<code>req.getServerName()</code>
SERVER_SOFTWARE	<code>getServletContext().getServerInfo()</code>
SERVER_PROTOCOL	<code>req.getProtocol()</code>
SERVER_PORT	<code>req.getServerPort()</code>
REQUEST_METHOD	<code>req.getMethod()</code>
PATH_INFO	<code>req.getPathInfo()</code>
PATH_TRANSLATED	<code>req.getPathTranslated()</code>
SCRIPT_NAME	<code>req.getServletPath()</code>
DOCUMENT_ROOT	<code>getServletContext().getRealPath("/")</code>
QUERY_STRING	<code>req.getQueryString()</code>
REMOTE_HOST	<code>req.getRemoteHost()</code>
REMOTE_ADDR	<code>req.getRemoteAddr()</code>
AUTH_TYPE	<code>req.getAuthType()</code>
REMOTE_USER	<code>req.getRemoteUser()</code>
CONTENT_TYPE	<code>req.getContentType()</code>
CONTENT_LENGTH	<code>req.getContentLength()</code>
HTTP_ACCEPT	<code>req.getHeader("Accept")</code>

<code>HTTP_USER_AGENT</code>	<code>req.getHeader("User-Agent")</code>
<code>HTTP_REFERER</code>	<code>req.getHeader("Referer")</code>

Notes

12.1 OBJECTIVES

After going through this unit, you will be able to:

- Learn about the methods used for processing
 - Servlets
 - Servers
 - Clients

12.2 THE SERVLET

Each registered servlet name can have specific initialization (init) parameters associated with it. Init parameters are available to the servlet at any time; they are set in the *web.xml* deployment descriptor and generally used in `init()` to set initial or default values for a servlet or to customize the servlet's behavior in some way.

Getting a Servlet Init Parameter

A servlet uses the `getInitParameter()` method for access to its init parameters:

```
public String
ServletConfig.getInitParameter(String name)
```

This method returns the value of the named init parameter or null if it does not exist. The return value is always a single String. It is up to the servlet to interpret the value.

```
public void init() throws ServletException
{
String greeting = getInitParameter("greeting");
}
```

Getting Servlet Init Parameter Names

A servlet can examine all its init parameters using `getInitParameterNames()`:

```
public Enumeration
ServletConfig.getInitParameterNames()
```

This method returns the names of all the servlet's init parameters as an Enumeration of String objects or an empty Enumeration if no parameters exist. It's most often used for debugging.

Getting a Servlet's Name

Also in the `ServletConfig` interface there's a method that returns the servlet's registered name:

```
public String ServletConfig.getServletName()
```

Using the servlet name in the key, each servlet instance can easily keep a separate attribute value within the shared context.

Example 12.1

```
import java.util.*;
import javax.servlet.*;
public class InitSnoop extends GenericServlet {
// No init() method needed

public void service(ServletRequest req,
ServletResponse res)
throws ServletException, IOException {
res.setContentType("text/plain");
PrintWriter out = res.getWriter();
out.println("Init Parameters:");
Enumeration enum = getInitParameterNames();
while (enum.hasMoreElements()) {
String name = (String) enum.nextElement();
out.println(name + ": " + getInitParameter(name));
}
}

public void doGet(HttpServletRequest
req,HttpServletResponse res)
throws ServletException, IOException {
String name = getServletName();
ServletContext context = getServletContext();
Object value = context.getAttribute(name +
".state");
}
}
}
```

12.3 THE SERVER

A servlet can find out much about the server in which it is executing. It can learn the hostname, listening port, and server software, among other things. A servlet can display this information to a client, use it to customize its behavior based on a particular server package, or even use it to explicitly restrict the machines on which the servlet will run.

There are *five methods* that a servlet can use to learn about its server: two that are called using the `ServletRequest` object passed to the servlet and

three that are called from the `ServletContext` object in which the servlet is executing

A servlet can get the name of the server and the port number for a particular request with `getServerName()` and `getServerPort()`, respectively:

```
public String ServletRequest.getServerName()
public int ServletRequest.getServerPort()
```

These methods are attributes of `ServletRequest` because the values can change for different requests if the server has more than one name (a technique called *virtual hosting*).

The `getServerInfo()` and `getAttribute()` methods of `ServletContext` provide information about the server software and its attributes:

```
public String ServletContext.getServerInfo()
public Object ServletContext.getAttribute(String
name)
```

`getServerInfo()` returns the name and version of the server software, separated by a slash. The string returned might be something like Tomcat Web Server. Some servers add extra information at the end describing the server operating environment.

`getAttribute()` returns the value of the named server attribute as an `Object` or null if the attribute does not exist. Servers have the option to place hardcoded attributes in the context for use by servlets.

Servlets can also add their own attributes to the context using the `setAttribute()` method. Attribute names should follow the same convention as package names. The package names `java.*` and `javax.*` are reserved for use by the Java Software division of Sun Microsystems, and `com.sun.*` is reserved for use by Sun Microsystems.

A listing of all current attributes stored by the server and other servlets can be obtained using `getAttributeNames()` :

```
public Enumeration
ServletContext.getAttributeNames()
```

Because these methods are attributes of the `ServletContext` in which the servlet is executing, you have to call them through that object:

```
String serverInfo =
getServletContext().getServerInfo();
```

The `javax.servlet.context.tempdir` attribute maps to a temporary directory where short-lived working files can be stored.


```
File dir = (File) getServletContext()
    .getAttribute("javax.servlet.context.tempdir");
File f = File.createTempFile("xxx", ".tmp", dir);
```

First, this servlet locates its temporary directory. Then, it uses the `createTempFile()` method to create a temporary file in that directory with an `xxx` prefix and `.tmp` suffix.

The `ServletContext` class has two methods namely `getInitParameter()` and `getInitParameterNames()` for retrieving contextwide initialization information:

```
public String
ServletContext.getInitParameter(String name)
public Enumeration
ServletContext.getInitParameterNames()
```

12.4 THE CLIENT

For each request, a servlet has the ability to find out about the client machine and, for pages requiring authentication, about the actual user. This information can be used for logging access data, associating information with individual users, or restricting access to certain clients.

A servlet can use `getRemoteAddr()` and `getRemoteHost()` to retrieve the IP address and hostname of the client machine, respectively:

```
public String ServletRequest.getRemoteAddr()
public String ServletRequest.getRemoteHost()
```

Both values are returned as `String` objects. The information comes from the socket that connects the server to the client, so the remote address and hostname may be that of a proxy server.

The IP address or remote hostname can be converted to a `java.net.InetAddress` object using `InetAddress.getByName()`:

```
InetAddress remoteInetAddress =
    InetAddress.getByName(req.getRemoteAddr());
```

The servlet can get the name of the user that was accepted by the server, using the `getRemoteUser()` method:

```
public String HttpServletRequest.getRemoteUser()
```

An HTTP servlet gets its request parameters as part of its query string (for GET requests) or as encoded POST data (for POST requests), or sometimes both. Fortunately, every servlet retrieves its parameters the same way, using `getParameter()` and `getParameterValues()`:

Notes

```
public String ServletRequest.getParameter(String
name)
public String[]
ServletRequest.getParameterValues(String name)
```

`getParameter()` returns the value of the named parameter as a `String` or `null` if the parameter was not specified. The value is guaranteed to be in its normal, decoded form. If there's any chance a parameter could have more than one value, you should use the `getParameterValues()` method instead. This method returns all the values of the named parameter as an array of `String` objects or `null` if the parameter was not specified. A single value is returned in an array of length 1. If you call `getParameter()` on a parameter with multiple values, the value returned is the same as the first value returned by `getParameterValues()`.

A servlet can use several methods to find out exactly what file or servlet the client requested. After all, only the most conceited servlet would always assume itself to be the direct target of a request. A servlet may be nothing more than the handler for some other content. No method directly returns the original Uniform Resource Locator (URL) used by the client to make a request. The `javax.servlet.http.HttpUtils` class, however, provides a `getRequestURL()` method that does about the same thing:

```
public static StringBuffer
HttpUtils.getRequestURL(HttpServletRequest req)
```

This method reconstructs the request URL based on information available in the `HttpServletRequest` object. It returns a `StringBuffer` that includes the scheme (such as HTTP), server name, server port, and extra path information.

Check Your Progress

1. List the advantages of the using Servlets when compared to CGI.
2. What is Virtual Hosting?
3. List some of the methods associated with Servlets.

12.5 ANSWERS TO CHECK YOUR PROGRESS

1. The servlet has advantages like stronger type checking, delayed calculation, more interaction with the server
2. *Virtual hosting* is a technique in which the methods are attributes of `ServletRequest` because the values can change for different requests if the server has more than one name.
3. Some of the methods associated with Servlets are listed below
 - `getServletName()`
 - `getServerName()`

- `getServerPort()`
- `getServerInfo()`

12.6 LET US SUM UP

Notes

The servlet approach has several advantages:

- Stronger type checking
- Delayed calculation
- More interaction with the server

`Init` parameters are available to the servlet at any time; they are set in the `web.xml` deployment descriptor and generally used in `init()` to set initial or default values for a servlet

Virtual hosting is a technique in which the methods are attributes of `ServletRequest` because the values can change for different requests if the server has more than one name.

Function	Purpose
<code>getInitParameter()</code>	This method returns the value of the named init parameter or null if it does not exist. The return value is always a single String
<code>getInitParameterNames()</code>	This method returns the names of all the servlet's init parameters as an Enumeration of String objects or an empty Enumeration if no parameters exist. It's most often used for debugging
<code>getServletName()</code>	Using the servlet name in the key, each servlet instance can easily keep a separate attribute value within the shared context
<code>getServerName()</code>	Get the name of the server name
<code>getServerPort()</code>	Get the name of the server port number
<code>getServerInfo()</code>	Returns the name and version of the server software, separated by a slash.
<code>getAttribute()</code>	Returns the value of the named server attribute as an Object or null if the attribute does not exist.
<code>setAttribute()</code>	Servlets can also add their own attributes to the context using this method
<code>getAttributeNames()</code>	A listing of all current attributes stored by the server and other servlets can be obtained
<code>createTempFile()</code>	To create a temporary file in the directory
<code>getInitParameter()</code>	Retrieving context wide initialization parameter information
<code>getInitParameterNames()</code>	Retrieving context wide initialization parameter name information
<code>getRemoteAddr()</code>	To retrieve the IP address of the client machine
<code>getRemoteHost()</code>	To retrieve the hostname of the client machine
<code>InetAddress.getByName()</code>	The IP address or remote hostname can be converted to a

	<code>java.net.InetAddress</code>
<code>getRemoteUser()</code>	The servlet can get the name of the user that was accepted by the server
<code>getParameter()</code>	Returns the value of the named parameter as a <code>String</code> or <code>null</code> if the parameter was not specified
<code>getParameterValues()</code>	This method returns all the values of the named parameter as an array of <code>String</code> objects or <code>null</code> if the parameter was not specified
<code>getRequestURL()</code>	It returns a <code>StringBuffer</code> that includes the scheme (such as <code>HTTP</code>), server name, server port, and extra path information.

12.7 SELF-ASSESSMENT EXERCISES

Short Questions

1. List the methods associated with `Server`.
2. Name some of the methods of `Client`.

Detail Questions

1. Describe the methods used to retrieve information about the `Servlet`.
2. Discuss about the methods used to retrieve information of the `Server`.
3. Explain the methods of `Client`.

12.8 SUGGESTED READINGS

1. Web Programming: Building Internet applications, Chris Bates, Wiley India
2. Web technologies – A computer science perspective, Jeffrey C Jackson, Pearson Education, 2006
3. Robert W. Sebesta “Programming the world wide web” Pearson Education
4. Bates, Developing web applications, wiley, 2006
5. Java servlet Programming, Jason Hunter, O’Reilly, 2010, 2nd Edition

BLOCK – V

JSP TECHNOLOGY

Notes

UNIT-13 JAVA SERVER PAGES

Structure

- 13.0 Introduction
- 13.1 Objectives
- 13.2 Need for JSP
- 13.3 HTTP and Servlet basics
- 13.4 HTTP request/response model
- 13.5 Anatomy of a JSP page
- 13.6 JSP application design with MVC
- 13.7 Answers to Check Your Progress
- 13.8 Let us Sum up
- 13.9 Self-Assessment Exercises
- 13.10 Suggested Readings

13.0 INTRODUCTION

In late 1999, Sun Microsystems added a new element to the collection of EnterpriseJava tools: JavaServer Pages (JSP). JavaServer Pages are built on top of Javaserivlets and designed to increase the efficiency in which programmers, and even nonprogrammers, can create web content.

JavaServer Pages is a technology for developing web pages that include dynamic content. Unlike a plain HTML page, which contains static content that always remains the same, a JSP page can change its content based on any number of variable items, including the identity of the user, the user's browser type, information provided by the user, and selections made by the user.

13.1 OBJECTIVES

After going through this unit, you will be able to:

- Understand the need for JSP Applications
- Learn about the HTTP request/response model
- Know the anatomy of a JSP page
- Design JSP application using MVC

13.2 NEED FOR JSP

Notes

A JSP page contains standard markup language elements, such as HTML tags, just like a regular web page. However, a JSP page also contains special JSP elements that allow the server to insert dynamic content in the page. JSP elements can be used for a variety of purposes, such as retrieving information from a database or registering user preferences. When a user asks for a JSP page, the server executes the JSP elements, merges the results with the static parts of the page, and sends the dynamically composed page back to the browser, as illustrated in Figure 13.1.

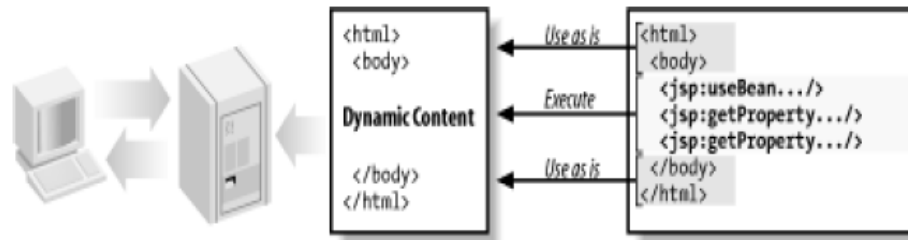


Figure 13.1. Generating dynamic content with JSP elements

JSP defines a number of standard elements that are useful for any web application, such as accessing JavaBeans components, passing control between pages and sharing information between requests, pages, and users. Developers can also extend the JSP syntax by implementing application-specific elements that perform tasks such as accessing databases and Enterprise JavaBeans, sending email, and generating HTML to present application-specific data.

Numerous CGI alternatives and enhancements, such as FastCGI, mod_perl from Apache, NSAPI from Netscape, ISAPI from Microsoft, and Java servlets from Sun Microsystems, have been created over the years.

While these solutions offer better performance and scalability, all these technologies suffer from a common problem: *they generate web pages by embedding HTML directly in programming language code*. This pushes the creation of dynamic web pages exclusively into the domain of programmers. JavaServer Pages, however, changes all that.

Embedding Dynamic Elements in HTML Pages

Instead of embedding HTML in programming code, JSP lets you embed special active elements into HTML pages. These elements look similar to HTML elements, but behind the scenes they are actually componentized Java programs that the server executes when a user requests the page.

```
<%@ taglib prefix="c"
uri="http://java.sun.com/jsp/jstl/core" %>
<html>
<body bgcolor="white">
<jsp:useBean id="clock" class="java.util.Date" />
<c:choose>
```

```

<c:when test="{clock.hours < 12}">
<h1>Good morning!</h1>
</c:when>
<c:when test="{clock.hours < 18}">
<h1>Good day!</h1>
</c:when>
<c:otherwise>
<h1>Good evening!</h1>
</c:otherwise>
</c:choose>
Welcome to our site, open 24 hours a day.
</body>
</html>

```

Notes

Compilation

Another benefit that is important to mention is that a JSP page is always compiled before it's processed by the server

Using the Right Person for Each Task

JSP allows you to separate the markup language code, such as HTML, from the programming language code used to process user input, access databases, and perform other application tasks.

Integration with Enterprise Java APIs

JavaServer Pages are built on top of the Java Servlets API, JSP has access to all the powerful Enterprise Java APIs

JSP combines the most *important features* found in the alternatives:

- JSP supports both scripting and element-based dynamic content, and allows developers to create custom tag libraries to satisfy application-specific needs.
- JSP pages are compiled for efficient server processing.
- JSP pages can be used in combination with servlets that handle the business logic, the model favored by Java servlet template engines.

In addition, JSP has a couple of *unique advantages* that make it stand out from the crowd:

- *JSP is a specification, not a product.* This means vendors can compete with different implementations, leading to better performance and quality.
- *JSP is an integral part of J2EE, a complete platform for enterprise class applications.* This means that JSP can play a part in the simplest applications to the most complex and demanding.

13.3 HTTP AND SERVLET BASICS

- Web applications can be defined as an application running on a server a user accesses through a thin, general-purpose client
- Servlets are modules that extend request/response-oriented servers, such as Java-enabled web servers. For example, a servlet might be responsible for taking data in an HTML order-entry form and applying the business logic used to update a company's order database.
- HTTP stands for Hyper Text Transfer Protocol which is a basis of data communication in the internet.
- When the web server receives the request, it looks at the URI and decides, based on configuration information, how to handle it. It may handle it internally by simply reading an HTML file from the filesystem, or it can forward the request to some component that is responsible for the resource corresponding to the URI.
- This can be a program that uses database information, for instance, to dynamically generate an appropriate response. To the browser it makes no difference how the request is handled; all it cares about is getting a response. The response message looks similar to the request message.

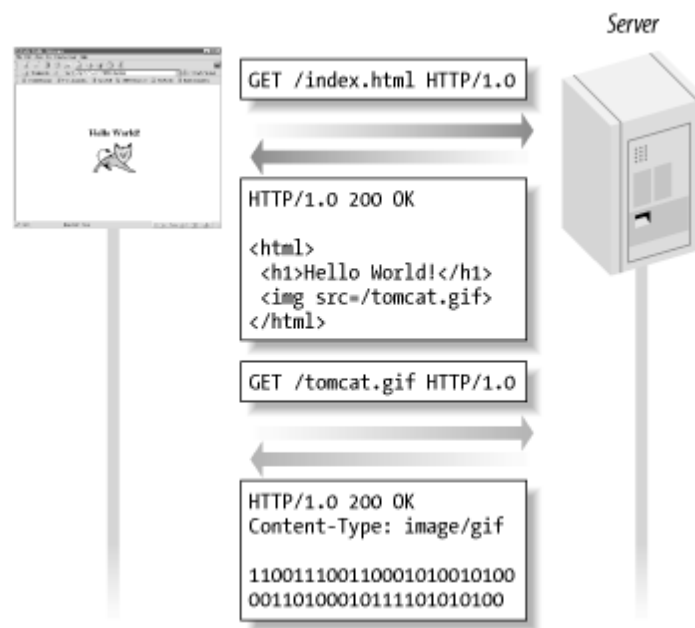


Figure 13.2. Interaction between a web client and a server

13.4 HTTP REQUEST/RESPONSE MODEL

HTTP and all extended protocols based on HTTP are based on a very simple communications model. Here's how it works: a client, typically a web browser, sends a request for a resource to a server, and the server sends back a response corresponding to the resource (or a response with an error message if it can't process the request for some reason). A resource can be a

number of things, such as a simple HTML file returned verbatim to the browser or a program that generates the response dynamically. This request/response model is illustrated in Figure 13.3.

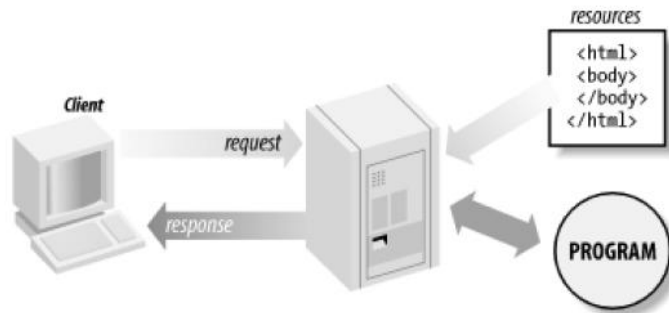


Figure 13.3. HTTP request/response with two resources

Notes

This simple model implies three important facts you need to be aware of:

- HTTP is a stateless protocol. This means that the server doesn't keep any information about the client after it sends its response, and therefore it can't recognize that multiple requests from the same client may be related.
- Web applications can't easily provide the kind of immediate feedback typically found in standalone GUI applications such as word processors or traditional client/server applications. Every interaction between the client and the server requires a request/response exchange. Performing a request/response exchange when a user selects an item in a list box or fills out a form element is usually too taxing on the bandwidth available to most Internet users.
- There's nothing in the protocol that tells the server how a request is made; consequently, the server can't distinguish between various methods of triggering the request on the client. For example, HTTP doesn't allow a web server to differentiate between an explicit request caused by clicking a link or submitting a form and an implicit request caused by resizing the browser window or using the browser's Back button. In addition, HTTP doesn't contain any means for the server to invoke client-specific functions, such as going back in the browser history list or sending the response to a certain frame. Also, the server can't detect when the user closes the browser.

13.5 ANATOMY OF A JSP PAGE

A JSP page is simply a regular web page with JSP elements for generating the parts that differ for each request, as shown in Figure 13.4. Everything in the page that isn't a JSP element is called template text. Template text can be any text:

- ✓ HTML
- ✓ WML
- ✓ XML or even plain text.

Notes

Since HTML is by far the most common web page language in use today, most of the descriptions and examples in this book use HTML, but keep in mind that JSP has no dependency on HTML; it can be used with any markup language. Template text is always passed straight through to the browser. When a JSP page request is processed, the template text and dynamic content generated by the JSP elements are merged, and the result is sent as the response to the browser.

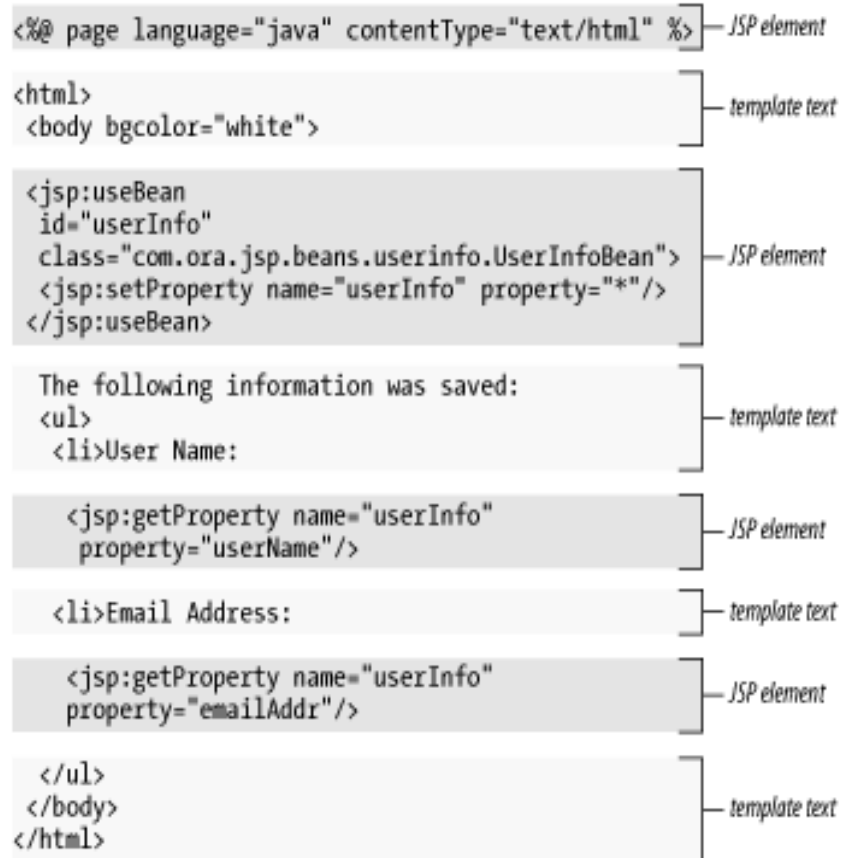


Figure 13.4. Template text and JSP elements

Example of JSP scriptlet tag that prints the user name

In this example, we have created two files `index.html` and `welcome.jsp`. The `index.html` file gets the username from the user and the `welcome.jsp` file prints the username with the welcome message.

File: `index.html`

```
<html>
<body>
<form action="welcome.jsp">
<input type="text" name="uname">
<input type="submit" value="go"><br/>
</form>
</body>
</html>
```

File: welcome.jsp

```
<html>
<body>
<%
String name=request.getParameter("uname");
out.print("welcome "+name);
%>
</form>
</body>
</html>
```

Notes

13.6 JSP APPLICATION DESIGN WITH MVC

JSP technology can play a part in everything from the simplest web application, such as an online phone list or an employee vacation planner, to complex enterprise applications, such as a human resource application or a sophisticated online shopping site. A design model called Model-View-Controller (MVC) is suitable for both simple and complex applications. MVC was first described by Xerox in a number of papers published in the late 1980s. The key point of using MVC is to separate logic into three distinct units:

- ✓ The Model
- ✓ The View and
- ✓ The Controller.

The terms are described below.

- **Model** represents the state of the application i.e. data. It can also have business logic.
- **View** represents the presentation i.e. UI (User Interface).
- **Controller** acts as an interface between View and Model. Controller intercepts all the incoming requests.

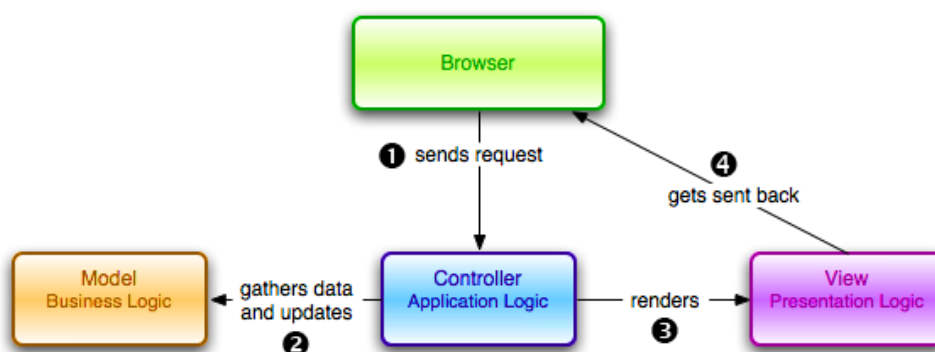


Figure 13.5. The MVC Architecture

In a server application, we commonly classify the parts of the application as business logic, presentation, and request processing. Business logic is the term used for the manipulation of an application's data, such as customer, product, and order information. Presentation refers to how the application data is displayed to the user, for example, position, font, and

Notes

size. And finally, request processing is what ties the business logic and presentation parts together. In MVC terms, the Model corresponds to business logic and data, the View to the presentation, and the Controller to the request processing.

Check Your Progress

1. What is JSP?
2. What are the advantages of JSP?
3. What do you mean by MVC?

13.7 ANSWERS TO CHECK YOUR PROGRESS

1. JavaServer Pages is a technology for developing web pages that include dynamic content
2. Advantages of JSP are
 - ✓ Embedding Dynamic Elements in HTML Pages
 - ✓ Compilation
 - ✓ Using the Right Person for Each Task
 - ✓ Integration with Enterprise Java APIs
3. A design model called Model-View-Controller (MVC) is suitable for both simple and complex applications.

13.8 LET US SUM UP

JavaServer Pages is a technology for developing web pages that include dynamic content

JSP page can change its content based on a number of variable items, including the identity of the user, the user's browser type, information provided by the user, and selections made by the user.

When a user asks for a JSP page, the server executes the JSP elements, merges the results with the static parts of the page, and sends the dynamically composed page back to the browser

Important features of JSP are

- JSP supports both scripting and element-based dynamic content, and allows developers to create custom tag libraries to satisfy application-specific needs.
- JSP pages are compiled for efficient server processing.
- JSP pages can be used in combination with servlets that handle the business logic, the model favored by Java servlet template engines

Advantages of JSP

- Embedding Dynamic Elements in HTML Pages
- Compilation
- Using the Right Person for Each Task
- Integration with Enterprise Java APIs

Notes

The HTTP request / response model conveys the following facts

- ✓ HTTP is a stateless protocol.
- ✓ Every interaction between the client and the server requires a request/response exchange.
- ✓ The server can't distinguish between various methods of triggering the request on the client

A design model called Model-View-Controller (MVC) is suitable for both simple and complex applications.

The key point of using MVC is to separate logic into three distinct units:

- ✓ The Model
- ✓ The View and
- ✓ The Controller

Model represents the state of the application i.e. data. It can also have business logic.

View represents the presentation i.e. UI(User Interface).

Controller acts as an interface between View and Model. Controller intercepts all the incoming requests.

13.9 SELF-ASSESSMENT EXERCISES

Short Questions

1. What the need for JSP?
2. State the important features of JSP.
3. What is the difference between Servlet and JSP?
4. What do you mean by MVC?

Detail Questions

1. Write a note on HTTP request / response model.
2. Discuss about the anatomy of a JSP page.
3. Explain about the MVC design.

13.10 SUGGESTED READINGS

1. Robert W. Sebesta "Programming the world wide web" Pearson Education
2. Bates, Developing web applications, wiley, 2006
3. Java server pages, Hans Bergsten, o'reilly, 2010

Notes

UNIT- 14 SETTING UP JSP ENVIRONMENT

Structure

- 14.0 Introduction
- 14.1 Objectives
- 14.2 Installing the JSDK
- 14.3 Installing the Tomcat Server
- 14.4 Testing Tomcat
- 14.5 Creating, installing and running a JSP page
- 14.6 JSP Program Example
- 14.7 Answers to Check Your Progress
- 14.8 Let us Sum up
- 14.9 Self-Assessment Exercises
- 14.10 Suggested Readings

14.0 INTRODUCTION

A development environment is where you would develop your JSP programs, test them and finally run them. This unit will describe the steps involved in setting the environment for JSP programming. The software that are to be installed are described in the subsequent sections.

14.1 OBJECTIVES

After going through this unit, you will be able to:

- Install the JSDK
- Install and test Tomcat Server
- Create, install and run a JSP page

14.2 INSTALLING THE JSDK

This step involves downloading an implementation of the Java Software Development Kit (JSDK) and setting up the PATH environment variable appropriately.

You can download SDK from Oracle's Java site – Java SE Downloads.

Once you download your Java implementation, follow the given instructions to install and configure the setup.

Finally set the PATH and JAVA_HOME environment variables to refer to the directory _____ that _____ contains `java` and `javac`,

typically `java_install_dir/bin` and `java_install_dir` respectively.

If you are running Windows and install the SDK in `C:\jdk1.5.0_20`, you need to add the following line in your `C:\autoexec.bat` file.

```
set PATH = C:\jdk1.5.0_20\bin;%PATH%
set JAVA_HOME = C:\jdk1.5.0_20
```

Alternatively, on Windows NT/2000/XP, you can also right-click on My Computer, select Properties, then Advanced, followed by Environment Variables. Then, you would update the PATH value and press the OK button.

On Unix (Solaris, Linux, etc.), if the SDK is installed in `/usr/local/jdk1.5.0_20` and you use the C shell, you will put the following into your `.cshrc` file.

```
setenv PATH /usr/local/jdk1.5.0_20/bin:$PATH
setenv JAVA_HOME /usr/local/jdk1.5.0_20
```

Alternatively, if you use an Integrated Development Environment (IDE) like NetBeans, Borland JBuilder, Eclipse, IntelliJ IDEA, or Sun ONE Studio, compile and run a simple program to confirm that the IDE knows where you installed Java.

14.3 INSTALLING THE TOMCAT SERVER

A number of Web Servers that support JavaServer Pages and Servlets development are available in the market. Some web servers can be downloaded for free and Tomcat is one of them.

Apache Tomcat is an open source software implementation of the JavaServer Pages and Servlet technologies and can act as a standalone server for testing JSP and Servlets, and can be integrated with the Apache Web Server. Here are the steps to set up Tomcat on your machine.

- ✓ Download the latest version of Tomcat from <https://tomcat.apache.org/>.
- ✓ Once you downloaded the installation, unpack the binary distribution into a convenient location. For example, in `C:\apache-tomcat-5.5.29` on windows, or `/usr/local/apache-tomcat-5.5.29` on Linux/Unix and create `CATALINA_HOME` environment variable pointing to these locations.

Tomcat can be started by executing the following commands on the Windows machine

```
%CATALINA_HOME%\bin\startup.bat
```

or

237

Notes

```
C:\apache-tomcat-5.5.29\bin\startup.bat
```

Tomcat can be started by executing the following commands on the Unix (Solaris, Linux, etc.) machine

```
$CATALINA_HOME/bin/startup.sh
```

or

```
/usr/local/apache-tomcat-5.5.29/bin/startup.sh
```

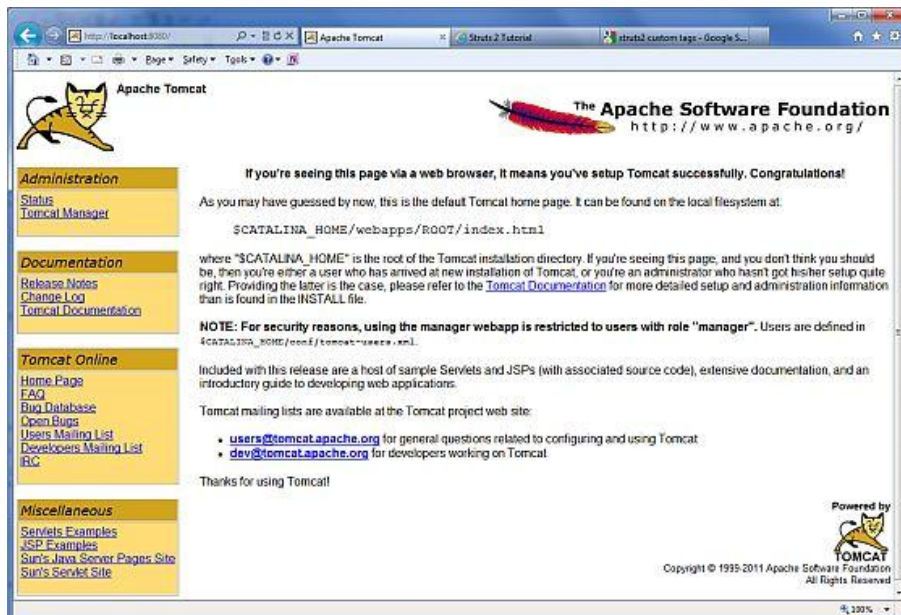
14.4 TESTING TOMCAT

The Tomcat installation directory contains a number of subdirectories. Some of them are

- ✓ bin
- ✓ conf
- ✓ webapps
- ✓ logs
- ✓ work

To test the server, run the startup script as described in the platform-specific sections, and (assuming you're running Tomcat on the same machine as the browser and that you're using the default 8080 port for Tomcat) open a browser and enter this URL in the Location/Address field: <http://localhost:8080/>.

Upon execution, you will receive the following output



Further information about configuring and running Tomcat can be found in the documentation included here, as well as on the Tomcat web site <https://tomcat.apache.org/>.

Tomcat can be stopped by executing the following commands on the Windows machine

```
%CATALINA_HOME%\bin\shutdown
```

or

```
C:\apache-tomcat-5.5.29\bin\shutdown
```

Tomcat can be stopped by executing the following commands on Unix (Solaris, Linux, etc.) machine

```
$CATALINA_HOME/bin/shutdown.sh
```

or

```
/usr/local/apache-tomcat-5.5.29/bin/shutdown.sh
```

Setting up CLASSPATH

Since servlets are not part of the Java Platform, Standard Edition, you must identify the servlet classes to the compiler. If you are running Windows, you need to put the following lines in your C:\autoexec.bat file.

```
set CATALINA = C:\apache-tomcat-5.5.29
set CLASSPATH = %CATALINA%\common\lib\jsp-
api.jar;%CLASSPATH%
```

Alternatively, on Windows NT/2000/XP, you can also right-click on My Computer, select Properties, then Advanced, then Environment Variables. Then, you would update the CLASSPATH value and press the OK button.

On Unix (Solaris, Linux, etc.), if you are using the C shell, you would put the following lines into your .cshrc file.

```
setenv CATALINA = /usr/local/apache-tomcat-5.5.29
setenv CLASSPATH $CATALINA/common/lib/jsp-
api.jar:$CLASSPATH
```

Note :

Assuming that your development directory is C:\JSPDev (Windows) or /usr/JSPDev (Unix), then you would need to add these directories as well in CLASSPATH.

14.5 CREATING, INSTALLING AND RUNNING A JSP PAGE

A JSP page is just a regular HTML page with a few special elements. A JSP page should have the file extension .jsp, which tells the server that the page needs to be processed by the JSP container. Without this clue, the server is unable to distinguish a JSP page from any other type of file and sends it unprocessed to the browser. When working with JSP pages, you just need a regular text editor such as Notepad on Windows or Emacs on Unix. There are a number of tools that may make it easier for you, such as syntax-aware editors that color-code JSP and HTML elements. Some Interactive

Notes

Development Environments (IDE) even include a small web container that allows you to easily execute and debug the pages during development.

Creating a JSP page

The first example JSP page, named `easy.jsp`, is shown in Example 14.1.

Example 14.1. JSP page showing a dynamically calculated sum(easy.jsp)

```
<%@ page contentType="text/html" %>
<%@ taglib prefix="c"
uri="http://java.sun.com/jsp/jstl/core" %>
<html>
<head>
<title>JSP is Easy</title>
</head>
<body bgcolor="white">
<h1>JSP is as easy as ...</h1>
<!-- Calculate the sum of 1 + 2 + 3 dynamically --
%>
1 + 2 + 3 = <c:out value="{1 + 2 + 3}" />
</body>
</html>
```

The `easy.jsp` page displays static HTML plus the sum of 1, 2, and 3, calculated at runtime and dynamically added to the response. We'll look at all the different pieces soon, but first you may want to run the example to see how it works.

Installing a JSP page

A complete web application may consist of several different resources: JSP pages, servlets, applets, static HTML pages, custom tag libraries, and other Java class files. Until very recently, an application with all these components had to be installed and configured in different ways for different servers, making it hard for web application developers to provide easy-to-use installation instructions and tools. The `web.xml` file is given below.

```
/index.html
/cover.gif
/unit14/easy.jsp
/WEB-INF/web.xml
/WEB-INF/classes/JSPSourceServlet.class
/WEB-INF/lib/orataglib_3_0.jar
```

Running a JSP page

First start the Tomcat server and load the book examples main page by typing the URL `http://localhost:8080/ora/index.html` in the browser address field.

14.6 JSP EXAMPLE

STEP1:

- Create a table in SQL as given below

```
Create table profile (id number(10), name
varchar(10), email varchar(10),
password varchar(10), location varchar(10));
```

*Notes***STEP2:**

- Go to webpage folder and create new html files
- Save them as index.jsp and insertregistration.jsp and paste the following code.

index.jsp

```
<%@ page language="java" contentType="text/html;
charset=ISO-8859-1"
pageEncoding="ISO-8859-1"%>

<html>
<head>
<meta http-equiv="Content-Type"
content="text/html; charset=ISO-8859-1">
<title>Simple Registration Form</title>
</head>

<body>
<h1>Student Register Form</h1>

<form action="example.jsp">

<table>
<tr>
<td>UserName</td>
<td><input type="text" name="username" /></td>
</tr>

<tr>
<td>Password</td>
<td><input type="password" name="password" /></td>
</tr>

<tr>
<td>Contact No</td>
<td><input type="text" name="contact" /></td>
</tr>

</table>
```

```
<input type="submit" value="Submit" /></form>
</body>
</html>
```

Notes

insertregistration.jsp

```
%@page
import="java.sql.*,java.lang.*,dbconnection.*,java
.text.SimpleDateFormat,java.util.*,java.io.*,javax
.servlet.*, javax.servlet.http.*"
errorPage="Error.jsp"%

<%@page import=" java.security.MessageDigest"%>
<%@page import=" java.security.*"%>
<%@page import="javax.crypto.*"%>

<%
Connection con;
Statement st = null;
ResultSet rs1=null;
int id=0;

try
{
Class.forName("oracle.jdbc.driver.OracleDriver");

con=DriverManager.getConnection("jdbc:oracle:thin:
@HbAdmin:1521:XE","system","system");

st=con.createStatement();

String sql1="select max(id) from profile";
rs1=st.executeQuery(sql1);

while(rs1.next())
{

if(rs1.getInt(1)==0)
id=1;
else
id=rs1.getInt(1)+1;

session.setAttribute("id",id);
String name=null, location=null,gender=null,
email=null, password=null,phone=null,image=null,
comment=null;
int report=0;
int upload=0;

try
{
```

```
name=request.getParameter("name");
email=request.getParameter("email");
password=request.getParameter("password");
location=request.getParameter("location");

Connection con1=db.getconnection();
PreparedStatement ps=con1.prepareStatement("INSERT
INTO profile VALUES(?,?,?,?);");

ps.setInt(1,id);
ps.setString(2,name);
ps.setString(3,email);
ps.setString(4,password);
ps.setString(5,location);
int x=ps.executeUpdate();

if(x!=0)
{
response.sendRedirect("index.jsp?message=successfu
lly registered");
}

else
{
response.sendRedirect("index.html?message=fail");
}

}
catch (Exception e)

{
out.println(e.getMessage());
}
}

}
catch (Exception eq)
{
out.println(eq.getMessage());
}
%>
```

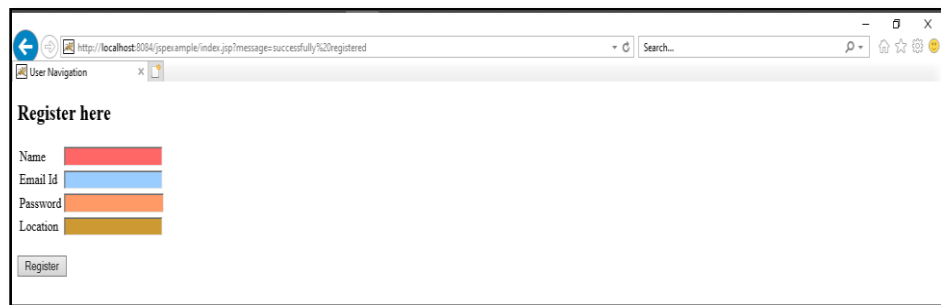
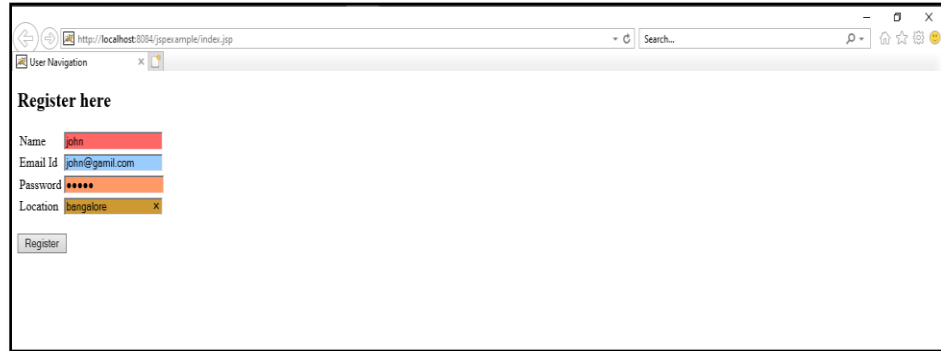
STEP 4:

- Check database connection and save files in the web page folder.
- Check the *web.xml* file

STEP 5:

- Right click on the project.
- Build and Run the project

Output



Check Your Progress

1. What are the software need to be installed for running JSP?
2. List some of the folders created automatically after the installation of Tomcat for JSP.
3. What is the need for Tomcat?

14.7 ANSWERS TO CHECK YOUR PROGRESS

1. The software need for running JSP are the JSDK and a web server such as Tomcat.
2. The Tomcat installation directory contains a number of subdirectories. Some of them are
 - ✓ bin
 - ✓ conf
 - ✓ webapps
 - ✓ logs
 - ✓ work
3. Apache Tomcat is an open source software implementation of the JavaServer Pages and Servlet technologies and can act as a standalone server for testing JSP and Servlets, and can be integrated with the Apache Web Server.

14.8 LET US SUM UP

A number of Web Servers that support JavaServer Pages and Servlets development are available in the market.

Apache Tomcat is an open source software implementation of the JavaServer Pages and Servlet technologies and can act as a standalone server for testing JSP and Servlets, and can be integrated with the Apache Web Server.

The Tomcat installation directory contains a number of subdirectories. Some of them are

- ✓ bin
- ✓ conf
- ✓ webapps
- ✓ logs
- ✓ work

Notes

14.9 SELF-ASSESSMENT EXERCISES

Short Questions

1. What is the ID for the local port?
2. Mention the address from where the JSDK and Tomcat can be downloaded.

Detail Questions

1. Explaining in detail about installing JSDK.
2. Describe the steps involved in installing Tomcat.
3. Write a program example in JSP

14.10 SUGGESTED READINGS

1. Web Programming: Building Internet applications, Chris Bates, Wiley India
2. Web technologies – A computer science perspective, Jeffrey C Jackson, Pearson Education, 2006
3. Robert W. Sebesta “Programming the world wide web” Pearson Education
4. Bates, Developing web applications, wiley, 2006
5. Java servlet Programming, JASON HUNTER, O’Reilly, 2010, 2nd Edition
6. Java server pages, Hans Bergsten, O’Reilly, 2010

MODEL QUESTION PAPER

DISTANCE EDUCATION

M. Sc DEGREE EXAMINATION

341 33– WEB TECHNOLOGY

Third Semester

(CBCS – 2018-19 Academic Year Onwards)

Time : 3 hours

Max Marks :75

PART - A (10 x 2=20 Marks)

Answer all questions.

1. Define the term hyperlink
2. What do you mean by Style sheets?
3. What is a cookier?
4. State the purpose of data validation.
5. What do you mean by DOM?
6. Expand SAX and XSL.
7. State the advantages of Servlets.
8. What is the purpose of `init()` and `destroy()`.
9. What do you mean by client side caching?
10. What is the purpose of Tomcat?

PART - B (5 x 5 Marks = 25 Marks)

Answer all questions choosing either (a) or (b)

- 11.a). Write a note on basic HTML tags.
OR
11. b). Describe the usage of table tag in HTML.
- 12.a). What do you mean by build in functions in Java Script? Explain.
OR
12. b). Write a note on cookies.
- 13.a). Describe about Document Object Model (DOM).
OR
13. b). Write a note on XSLT and XPATH.

14.a). Explain about Single Thread Model.

OR

14. b). Describe some functions used for retrieving information.

15.a). Explain the anatomy of a JSP page.

OR

15. b). Write a note on MVC.

Notes

Part – C (3 x 10 = 30 Marks)

Answer any three questions.

16. Explain in detail about Style sheets.

17. How will you perform validations in Java Script? Explain with example.

18. Discuss in detail about representing and processing XML using Java Script.

19. Describe the Servlet life cycle with example.

20. Explain about creating, installing and running a JSP page.
