Content

Block-I

UNIT-I

MARKUP LANGUAGES

1.1 Learning Objectives

1.2 Introduction to Hyperlinks

1.3 Web Standards

1.4 The History of Markup Languages

1.5 HTML

1.6 XML

1.7 XHTML

1.8 WML-Wireless Markup Language

1.9 Cascading Style Sheets (CSS)

1.10 DHTML

1.11 Client-Side and Server-Side Technologies

1.12 Basics of HTML

1.13 HTML Document Structure

1.14 Conclusion

1.15 Check Your Progress

1.16 Answer Check Your Progress

1.17 Model Question

1.18 References

1.19 Suggested Readings

UNIT-II

HTML

2.1 Learning Objectives

2.2 Introduction to Hyperlinks

2.3 Hyperlinks

2.4 Hyperlinks - Examples

2.5 Links to the same document

2.6 Frames and Framesets

2.7 Frame attributes

2.8 Nested framesets

2.9 Conclusion

2.10 Check your progress

2.11 Answer Check your progress

2.12 Model Question

2.13 References

2.14 Suggested readings

UNIT-III

HTML TABLES

3.1 Learning Objectives

3.2 Introduction

3.3 HTML Tables

3.4 Tables Attributes

3.5 Rows and Columns

3.6 Introduction to Forms

- 3.6.1 The <FORM> tag
- 3.6.2 The <INPUT> tag
- 3.6.3 Text input

3.7 Summary

3.8 Check your progress

3.9 Answer Check your progress

3.10 Model Question

3.11 References

3.12 Suggested readings

UNIT-IV

CASCADING STYLE SHEETS (CSS)

4.1 Learning Objectives

4.2 Introduction

4.3 Inline Style sheet

4.4 Internal Style Sheet

4.5 External Style sheet

4.6 Imported Style Sheet

4.7 Cascading Rules

4.8 Order rules

4.9 Order rules

4.10 Difference between ID and Classes

4.11 Anatomy of a style rule

4.12 Style Classes

4.13 Attribute Selector

4.14 Summary

4.15 Check your progress

4.16 Answer Check your progress

4.17 Model Question

4.18 References

4.19 Suggested readings

Block-II

UNIT-V

eXtensible Markup Language (XML)

5.1 Learning Objectives

5.2 HTML – eXtensible Hyper Text Markup Language

5.3 Need for XHTML

5.4 XHTML – Base syntactic rules

5.5 XHTML – other syntactic rules

5.6 DTD – Document Type Definition

5.7 XML

5.8 XML Trees

5.9 Namespaces

5.10 Defining XML Data Formats

5.11 Summary

5.15 Check your progress

5.16 Answer Check your progress

5.17 Model Question

5.18 References

5.19 Suggested readings

UNIT-VI

JAVASCRIPT AND HTML DOM

6.1 Learning Objectives

6.2 Document Object Model or DOM

6.3 JavaScript - Document Object Model or DOM

6.4 Types of DOM nodes

6.5 Traversing the DOM tree

6.6 Working with DOM

6.7 Summary

6.8 Check your progress

6.9 Answer Check your progress

6.10 Model Question

6.11 References

6.12 Suggested readings

UNIT-VII

JAVASCRIPT AND EVENT HANDLING

7.1 Learning Objectives

7.2 JavaScript - Event Handlers

7.2.1 onClick event type

7.2.2 onSubmit event type

7.2.3 onMouseOver and onMouseOut event

7.3 javascript cookies

7.4 create cookies

7.5 Store Cookies

7.6 Summary

7.7 Check your progress

7.8 Answer Check your progress

7.9 Model Question

7.10 References

7.11 Suggested readings

UNIT-VIII

JAVA SERVER PAGES(JSP)

8.1 Learning Objectives

8.2 Introduction to JSP

8.3 Comparison of Servlets with JSP

8.4 Advantages of JSP

8.5. JSP engines

8.6 JSP Architecture

8.7 Life cycle of a JSP Page

8.8 Directory structure of JSP

8.9 Anatomy of a JSP page

8.10 JSP Components and Tags

8.11 Creation of a small Web application using JSP

8.12 Summary

8.13 Check your progress

8.14 Answer Check your progress

8.15 Model Question

8.16 References

8.17 Suggested readings

Block-III

UNIT-IX

JAVA SERVER PAGES(JSP) II

9.1 Learning Objectives

9.2 JSP Elements – JSP Actions

9.3 JSP Implicit Objects

9.4 JSP - Exception Handling

9.5 JSP Session Tracking

9.6 Summary

9.7 Check your progress

9.8 Answer Check your progress

9.9 Model Question

9.10 References

9.11 Suggested readings

UNIT-X

JAVA SERVER PAGES (JSP) WITH JDBC

10.1 Learning Objectives

10.2 JDBC

10.3 Basic steps to connect database

10.4 Install Mysql

10.5 Steps to run JSP using Tomcat server

10.6 Creating a table using JSP example

10.7 Creating a table: database

10.8 SELECT operation

10.9 INSERT operation

10.10 DELETE Operation

10.11 Summary

10.12 Check your progress

10.13 Answer Check your progress

10.14 Model Question

10.15 References

10.16 Suggested readings

UNIT-XI

JDBC – INTRODUCTION

11.1 Learning Objectives

11.2 JDBC

11.3 JDBC Architecture

11.4 JDBC Drivers

11.5 Working with JDBC

11.6 JDBC Object Classes

11.7 Summary

11.8 Check your progress

11.9 Answer Check your progress

11.10 Model Question

11.11 References

11.12 Suggested readings

UNIT-XII

JAVA SERVLET IMPLEMENTATION

12.1 Learning Objectives

12.2 Tomcat Installation

12.3 Servlet Implementation

12.4 The doGet method

12.5 The doPost method

12.6 Summary

12.7 Check your progress

12.8 Answer Check your progress

12.9 Model Question

12.10 References

12.11 Suggested readings

Title	Web Technology
Authors	
Adaption and Typesetting	Dr. Ashutosh Kumar Bhatt Associate Professor School of Computer Science and IT Uttarakhand Open University
ISBN:	
Acknowledgement	
This textbook has been adapted from "e-PG Pathshala" available at https://epgp.inflibnet.ac.in/Home/ViewSubject?catid=7	
Published By: Uttarakhand Open University	

Block-I

UNIT-I MARKUP LANGUAGES

1.1 Learning Objectives

1.2 Introduction to Hyperlinks

1.3 Web Standards

1.4 The History of Markup Languages

1.5 HTML

1.6 XML

1.7 XHTML

1.8 WML-Wireless Markup Language

1.9 Cascading Style Sheets (CSS)

1.10 DHTML

1.11 Client-Side and Server-Side Technologies

1.12 Basics of HTML

1.13 HTML Document Structure

1.14 Conclusion

2.15 Check Your Progress

1.16 Answer Check Your Progress

1.17 Model Question

1.18 References

1.19 Suggested Readings

1.1 LEARNING OBJECTIVES

The objective of this module is to know about the different Markup languages. This module also provides an introduction about HTML. It discusses about the basic tags such as Hyperlinks, Formatting, Headings, Paragraphs and Tables in HTML.

1.2 INTRODUCTION TO HYPERLINKS

Web site creation is important to create visually appealing websites. Web design is the creation and visual design of documents displayed on the World Wide Web. There are more than a handful of different scripting languages and markup languages for attractive web site creation. Web programming languages like HTML, XML and XHTML provide the tools to build and design a web site. These web programming languages or otherwise called as Markup languages are basically used for website creation and to create dynamic and interactive websites.

Before we discuss about the markup languages, let us see about the basic terminologies used in the context of Web site creation.

- Web server is a system on the Internet containing one or more web site.
- Web site is termed as a collection of one or more web pages.
- A Web pages is a single disk file with a single file name.
- Home pages are the first page in the website.

To know about how the Web works, the web information stored in the form of Web pages. Now let us understand, the web pages are available in HTML format. The web pages are stored in the computers called Web servers in the Web server file system. The computer reading the pages is called web clients with specific web browser. The most commonly used web browsers are Internet Explorer, Mozilla Firefox and Google Chrome etc. The web server waits for the request from the web clients over the Internet. The well known web servers are Internet Information Server (IIS) or Apache.

1.3 WEB STANDARDS

The Web standards are not defined or setup by the browser companies or Microsoft, but by the World Wide Web Consortium (W3C). The specifications form the Web standards as HTML, CSS, XML, XHTML etc.

W3C - World Wide Web Consortium

The World Wide Web Consortium (W3C) is an international community to develop Web standards. It is led by the Web inventor Tim Berners-Lee and CEO Jeffrey Jaffe. W3C's mission is to lead the Web to its full potential.

W3C's long term goals for the Web are:

- Universal Access: To make the Web accessible to all by promoting technologies.
- Semantic Web: To develop a software environment that permits each user to make the best use of the resources available on the Web.
- Web of Trust: To guide the Web's development with careful consideration for the novel legal, commercial, and social issues raised by this technology.

CHECK YOUR PROGRESS

True/False type questions

1. Web design is the creation and visual design of documents displayed on the World Wide Web.
2. Web programming languages like HTML, XML and XHTML provide the tools to build and design a web site.
3. Web server is a system on the Internet containing one or more Hard Disk.
4. Web site is termed as a collection of one or more web pages.
5. A Web pages is a single disk file with a single file name.
6. Home pages are the first page in the Worksheet.
7. The web information stored in the form of Web pages.
8. The Web standards are defined or setup by the browser companies or Microsoft.

Answers-

1. True
2. True
3. False
4. True
5. True
6. False
7. True
8. False

1.4 THE HISTORY OF MARKUP LANGUAGE

In the early 1970s, GML called the Generalized Markup Language was developed where every tag would be defined like the following

":h1.The Content is placed here"

Since 1980s, there evolved Standard Generalized Markup Language called SGML and HTML. SGML was originally created by IBM in 1986. It is actually a meta language, meaning it is used to create other languages. Hence SGML forms the basis for the development of Markup languages like HTML, XHTML and XML.

Currently the Markup language that is widely used is eXtensible Markup Language or in short XML. It is not intended to replace HTML. Later, there evolved XHTML which does by providing better data description.

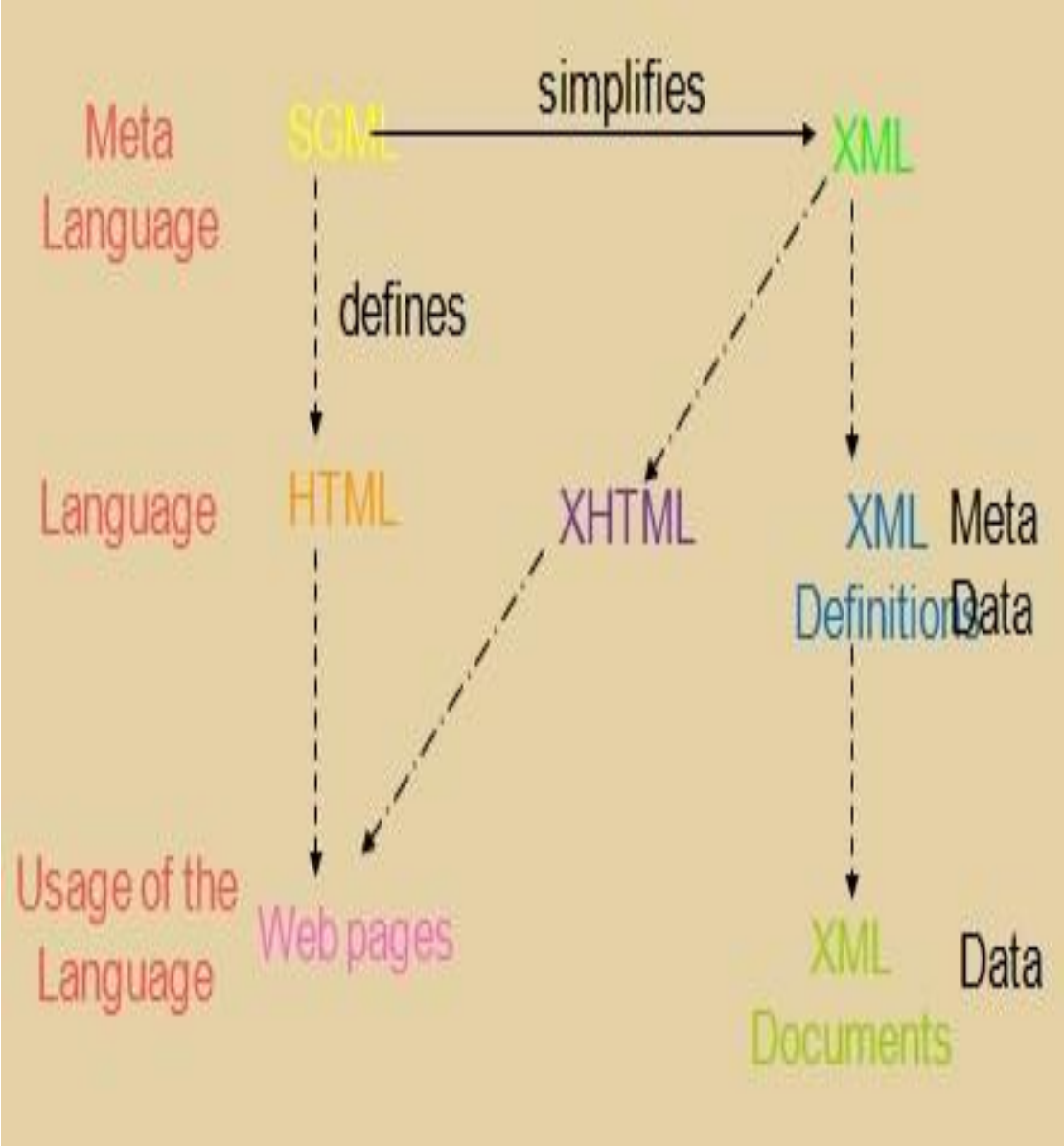


Figure 4.1 SGML, HTML and XML

The above Figure 4.1 describes about how the Markup languages evolved over time and the purpose of these languages.

SGML is a Meta language for the development of other Markup languages. Based on SGML, HTML invented by Tim Berners-Lee was developed that allows hyperlinks to display

multimedia information as web pages to the web users. XML is another Markup language whose purpose is to describe information and transfer information over the Internet. XHTML is another language which combines the features of both HTML and XML modified to conform to XML standards.

1.5 HTML

The expansion of HTML is HTML HyperText Markup Language. HTML is the set of "markup" symbols or codes inserted in a file intended for display on a World Wide Web browser. The markup tells the Web browser how to display a Web page's text, images, sound and video files for the user. It is not a programming language i.e., it cannot be used to describe computations. In HTML, the individual markup codes are referred to as elements (but many people also refer to them as tags).

History of HTML

HTML is an evolving standard as new technology/tools are added even now.

- HTML 1 (Berners-Lee, 1989): very basic, limited integration of multimedia
- HTML 2.0 (IETF, 1994): tried to standardize these & other features
- HTML 3.2 (W3C, 1996): attempted to unify into a single standard but didn't address newer technologies like Java applets & streaming video
- HTML 4.0 (W3C, 1997): current standard (but moving towards XHTML) attempted to map out future directions for HTML, not just react to vendors
- XHTML 1.0 (W3C, 2000): HTML 4.01 modified to conform to XML standards
- XHTML 1.1 (W3C, 2001): “Modularization” of XHTML 1.0
- HTML 5 (Web Hypertext Application Technology Working Group, W3C, 2006): New version of HTML4, XHTML 1.0, and DOM 2 (still a work in progress), no longer based on SGML, but “backward compatible” with parsing of older versions of HTML.
- HTML 5 is referred to as a "living language".

1.6 XML

XML is the abbreviation of eXtensible Markup Language. It provides a standard way to represent information so as to allow information to be stored and interchanged among any Internet-connected devices. It is not a markup language but it is a meta-markup language that specifies rules for creating markup languages. Browsers use XML parsers to isolate and extract the information from XML documents.

1.7 XHTML

The eXtensible HyperText Markup Language or XHTML is a reformulation of HTML 4 in XML 1.0. It consists of all HTML 4.0.1 predefined components combined with XML standards. This language has been developed as a way of making XML documents that look and act like HTML documents. Using XHTML helps to strengthen the structure and syntax of the markup.

1.8 WML-WIRELESS MARKUP LANGUAGE

Formerly called HDML (Handheld Devices Markup Languages) allows the text portions of web pages to be displayed on cell phones or PDAs via wireless media. It is part of the Wireless Application Protocol (WAP).

1.9 CASCADING STYLE SHEETS (CSS)

It provides a powerful and flexible way to control the details of web documents. HTML is more concerned about the content, CSS is used to impose a particular style on the document. It is named as cascading style sheets because they can be defined at three different levels to specify the style of a document called Inline, document level and external.

1.10 DHTML

It is used to describe a set of animated web documents that built from HTML, style sheets and scripts. There are three main parts of DHTML called,

- Positioning
- Style modifications
- Event handing

It relies on the browser for the display and manipulation of the web pages.

1.11 CLIENT-SIDE AND SERVER-SIDE TECHNOLOGIES

The Table 4.1 lists the different web technologies to develop web programs or scripts at client- side and server-side.

Client Side and Server Side Technologies

Client-Side	Server-Side
HTML, XML Cascading Style Sheets (CSS) Scripting languages: JavaScript, VBScript Java Applets ActiveX controls Plug-ins and Helpers application	CGI/Perl PHP ColdFusion Scripting Languages: Server-side JavaScript ASP, JSP, Java Servlets ISAPI/NSAPI programs

Table 4.1 Client-side and Server-side Technologies

1.12 BASICS OF HTML

Hypertext Markup Language (HTML) is a programming tool that uses hypertext to establish dynamic links to other documents. It is known as the Web's programming language and provides a general structure for creating web pages. All web pages are actually HTML files. HTML documents are simply text documents that uses simple ASCII text files to create HTML documents with HTML file extensions or suffix as .html or .htm. HTML documents can be created with Notepad in Windows and TestEdit in MAC OS. HTML editors can also be used.

The HTML documents contains the content or information of the webpage as well as special instructions called tags. Tags provide instructions on how to display text or graphics and control user inputs. Tags are generally enclosed in angle brackets: < >. Typically, there is a starting and ending tag around text. Embedded tags typically provides instruction for the structure, and appearance of the content.

Quote: HTML Editors are called “WYSIWYG” meaning "What You See Is What You Get! " Some of the examples of HTML Editors are Dreamweaver, FrontPage, GoLIve etc.

1.13 HTML DOCUMENT STRUCTURE

A HTML document contains the information in the form of text data (content of the page). The HTML document is divided into two major parts as HEAD and BODY. The head part of the document contains the information about the page, e.g. the title and the body part contains the actual content of the page.

The basic document starts with <HTML> and ends with </HTML>

The HEAD part of the document contains information about the document namely,

- Title of the page which appears at the top of the browser window).
- Meta tags which is used to describe the information about the content that is in the document (used by Search engines).
- The script code written in JavaScript and Style sheets generally appears in the document Head.

The BODY part of the document contains the actual content of the document. This is the part that will be displayed in the browser window.

A sample HTML document looks like this,

```
<HTML>
    <HEAD>
        <TITLE> My web page </TITLE>
    </HEAD>
    <BODY>
        Content of the document
    </BODY>
</HTML>
```

HTML Tags

All HTML tags are made up of a tag name and sometimes they are followed by an optional list of attributes which all appear between angle brackets < >. The content will not be displayed by the browser within the brackets unless the HTML is correctly written and the browser interprets the tags as part of the content. Attributes are properties that extend or refine the tag's functions.

Basic Syntax

Most of the HTML tags but not all have a start tag and an end tag like the following

<H1>Hello, world!</H1>

There are a few HTML tags that are standalone tags which do not use an end tag and are used for representing standalone elements on the page. Some of those tags are given below,

<img>	to display an image
 	Line break
<HR>	header

Attributes

Attributes are added within a tag to extend a tag's action. We can add multiple attributes within a single tag. Attributes appear after the tag name and each attribute should be separated by one or more spaces. Most attributes take values, which follow an equal sign "=" after the attribute's name. The attribute values are limited to 1024 characters in length. An example of attributes defined within a tag is given below,

<body bgcolor="khaki" text="#000000" link="blue" vlink="brown" alink="black"

Information which the browser will ignore are tabs and multiple spaces will appear as a single space.

For example if the text appears as below in the document,

"Hello,

How are you?"

The browser will ignore the blanks and new line and displays

Hello, How are you?

**Line break
**

This tag breaks the line and starts text at a new line. It will not add an empty line like the paragraph tag. Multiple <br> tags will display multiple line breaks in the text.

Paragraph Tag <P>

<P> is a Paragraph tag. It creates more space than a <BR> tag. It leaves one empty line after the tag. Multiple <P> tags with no intervening text is interpreted as redundant by all browsers and will display a single <P> tag.

Horizontal Rule <HR>

<HR> tag creates a Horizontal Rule. We can use attributes with <hr> such as

<hr width="70%">

Comments <!-- -->

The text enclosed within the comment tag will not be displayed. It is used to insert comments in the source code.

<!-- This is a comment -->

<!-- This is another

comment -->

We can also use the following tag as a comment,

<comment> This a comment </comment>

Headings: <h1> .. <h6>

We can create headlines of various sizes on our web page. Headlines normally appear as bold letters. An empty line will also follow the headlines. It is used for representing titles and subtitles of various sizes.

For example, H1 is the largest font heading and H6 is the smallest font heading. These Headings tag need an end tag </H1>.

Character Formatting

Special types of text that can be displayed using HTML are:

- Bold text
- Important text
- Italic text
- Emphasized text
- Marked text • Small text
- Deleted text
- Inserted text
- Subscripts
- Superscripts

The following HTML tags are used to format the appearance of the text on a web page.

<B> Bold </B>

The text enclosed within the <B> tag are displayed as Bold or it appears as dark letters.

<I> Italic </I>

The text enclosed within the <I> tag are displayed as Italic letters.

<U> Underline </U>

The text enclosed within the <U> tag are displayed as Underlined text.

<PRE> Preformatted </PRE>

The text enclosed by PRE tags is displayed in a mono-spaced font. Spaces and line breaks are supported without additional elements or special characters.

<EM> Emphasis </EM>

The text enclosed by <EM> tags are usually displayed as italics.

<STRONG> STRONG </STRONG>

Browsers display this as bold. The HTML element defines strong text, with added semantic "strong" importance.

Example

```
<p>This text is normal.</p>

<p><strong>This text is strong</strong>.</p>
```

The above text is displayed as follows.

This text is normal.

This text is strong.

<TT> TELETYPE </TT>

The text enclosed by <TT> tag is displayed in a mono-spaced font. It looks like a typewriter text, e.g. fixed-width font.

<CITE> A Beginner's Guide to HTML </CITE>

The text enclosed by <CITE> tag represents a document citation usually in **italics**. For example, it appears like this,

(A Beginner's Guide to HTML)

It is used to display for titles of books, films, etc.

<FONT SIZE="+2"> Two sizes bigger</FONT>

The size attribute can be set within the <FONT> tag as an absolute value from 1 to 7 or as a relative value using the "+" or "-" sign. Normal text size is 3 (from -2 to +4).

The color attribute can be set within the <FONT> tag,

Color = "#RRGGBB" The COLOR attribute of the FONT element.

For Example, <FONT COLOR="#RRGGBB">this text has color</FONT>

Lists

Lists are used to organize items in the browser window. There are two ways to create a list namely, Unordered list which is a Bulleted list and it is most popular. Here, list items have no particular order. The other list is an Ordered list or Numbered list.

- Ordered Lists (OL): e.g. 1,2,3
- UnOrdered Lists (UL): e.g. bullets.

Basic Syntax:

Unordered Lists

Fruit

<UL>

<LI>Banana

<LI>Grape

</UL>

Exploring Different Attributes

We have the choice of setting the TYPE Attribute to one of five numbering styles.

<OL

type=I or i (for large or small roman numerals) type=A or a (for capital or small letters) type=1 (for numbers, which is the default)>

<UL

type=disc (the default for first level lists) type=round (the default for second level lists) type=square (the default for third level lists) >

Numbered list/ Ordered List:

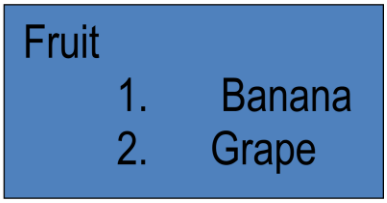
Fruit

<OL>

<LI> Banana

<LI>Grape

</OL>



We can also specify a starting number for an ordered list.

<OL TYPE="i" START="3">

<LI> List item …</LI>

</OL>

<OL TYPE ="i">

<LI> List item …</LI>

<LI> List item …</LI>

</OL>

<P> text ….</P>

<OL TYPE="i" START="3">

<LI> List item …</LI>

</OL>

List Elements

DL: Definition List.

This kind of list is different from the others. Each item in a DL consists of one or more **Definition Terms (DT elements),** followed by one or more **Definition Description (DD elements).**

<DL>

<DT> HTML **</DT>**

<DD> Hyper Text Markup Language **</DD>**

<DT> DOG **</DT>**

<DD> A human's best friend**!</DD>**

</DL>

The above HTML code appears as below in the browser.

HTML

Hyper Text Markup Language

DOG

A human's best friend!

1.14 CONCLUSION

In this module the different Markup languages have been introduced. The Web programming language Hyper Text Markup Language (HTML) has been explored and the basic tags has been explained in HTML.

1.15 CHECK YOUR PROGRESS

Fill in the blanks

1. SGML was originally created by ..... n 1986.

2. ..... Markup Language or in short XML.
3. Based on SGML, HTML invented by.....
4. XML is another Markup language whose purpose is to describe information and .....information over the Internet.
5. .....is another language which combines the features of both HTML and XML.
6. .....provides a standard way to represent information so as to allow information to be stored and interchanged among any Internet-connected devices.
7. The eXtensible HyperText Markup Language or XHTML is a reformulation of HTML 4 in.....
8. ..... is more concerned about the content, CSS is used to impose a particular style on the document.
9. There are three main parts of ..... called, Positioning Style, modifications, and Event handing.
10. A HTML document contains the information in the form of.....

1.16 ANSWER CHECK YOUR PROGRESS

1. IBM
2. eXtensible
3. Tim Berners-Lee
4. Transfer
5. XHTML
6. eXtensible Markup Language
7. XML 1.0
8. HTML
9. DHTML
10. text data

1.17 MODEL QUESTION

1. What is HTML? How is different from HTML? What are its characteristics? Explain.
2. What are the different Support Elements of HTML? Explain.
3. What are the major benefits of HTML? How will you migrate from HTML4 to HTML5? Explain.
4. What are the different tips and tricks of Sample Web Page Designing? Explain.
5. What is the importance of Div tag in HTML? How is it used? Give suitable example.
6. What do understand by the Responsive web design? What is the effect of responsive web design? How it will affect the look and feel of your web page? Explain.
7. What is the difference between Static and Dynamic Websites? How Dynamic website is better than the Static web site? Explain.
8. What are the advantage and disadvantages of static and dynamic web site? Explain with help of some example.
9. What is the use of CSS? Also list find the difference between CSS and HTML?

10. Explain the three main ways to apply CSS styles to a web page. Define different kinds of selectors used in CSS.
11. What are pseudo classes and what are they used for?
12. What are attributes and how are they used? How do margin, border and padding fit together into a web page?
13. Describe about class selector in CSS?
14. Explain external Style Sheet? How would you link to it in your web page? give example.
15. What are the limitations of CSS? What are the advantages of CSS?
16. Discuss merits of CSS3. How many ways through which you can integrate CSS in your web page?
17. What is the CSS3 animation? Explain.
18. What are the possible values of the "Position" attributes? What are CSS3 Transitions?

1.18 REFERENCES

- https://epgp.inflibnet.ac.in/Home/ViewSubject?catid=7

1.19 SUGGESTED READINGS

- Powell. Thomas A., HTML & CSS: The Complete Reference, 5E.
- lemay. Laura, rafe colburn, jennifer kyrnin, Mastering HTML, CSS & JavaScript Web, BPB Publication, 2016.
- vishvajeet. Sisodia, Basic of Web Design, HTML, CSS3, Centrum Press, 2014.

UNIT-II HTML

2.1 Learning Objectives

2.2 Introduction to Hyperlinks

2.3 Hyperlinks

2.4 Hyperlinks - Examples

2.5 Links to the same document

2.6 Frames and Framesets

2.7 Frame attributes

2.8 Nested framesets

2.9 Conclusion

2.10 Check your progress

2.11 Answer Check your progress

2.12 Model Question

2.13 References

2.14 Suggested readings

2.1 LEARNING OBJECTIVES

The last module explains about Tables and its attributes in HTML. Moreover, the module also explains about HTML Forms. In this module we will know about Hyperlinks in HTML. We will also learn about HTML Images, Frames, how to define Frames and the purpose of the Frameset document.

2.2 INTRODUCTION TO HYPERLINKS (ANCHOR TAG)

HTML provides a feature called Links that allows users to navigate from page to page by clicking on words, phrases and images. These HTML links are called hyperlinks. A hyperlink is a text or an image and when we click on, we jump to another document. A webpage can contain various links that takes us directly to other pages and even specific parts of a given page. Thus we can create hyperlinks using text or images available on a webpage.

Hyperlinks allow visitors to navigate between Web sites.

Hyperlinks are used for linking:

- within the same page. These are called as Named tags.
- To another page in a web site called Relative Link or local link.
- To another page outside a web site and these links are called Absolute link or remote link. – Email Link

Hyper Links are normally displayed as highlighted and underlined text. When you click on it, it takes you to another page on the web. The <a> tag is used for creating hyperlinks. The syntax is given as following,

highlighted text

2.3 HYPERLINKS

Now let us see about how to create the different kinds of links in a webpage. Hyperlinks are created with an "href" tag (hyperlink reference). In its simplest form the tag looks like this.

- Absolute Link: These are links to another page outside of our web site. These links specify the entire URL of the page:

<a HREF=http://www.annauniv.edu/>AU Web Site

The output of the above code is,

AU Web Site

- Relative Link: These are links to another page in our site so we do not have to specify the entire URL.

  <a HREF="index.html">Go back to main page

  The output of the above code is,

  Go back to main page

- Targeted Links : A tag includes a target attribute. If we specify target="_blank", a new browser window will be opened.

  <a HREF=http://www.annauniv.edu target="_blank"> AU

  We will see in detail about these links when we get into frames.

- Email Link: We can mail to someone at:

  Send email to e.x.

As seen earlier there are two distinct types of hyperlink called "absolute" and "relative". Relative links are divided into two further categories as "document-relative" and "site-root-relative" links. Document-relative links use directions from the source page to the destination page. It depends on whether the destination page is in the same directory as the source page or in a folder inside the source page's folder or in a folder outside the source page's folder. We can use a ../ which signifies "one directory higher". We can repeat the ../ to make the link more than one level higher which means going up two levels, to a file.

Site-root-relative links use a single forward-slash / that starts at the document root of the website and go down the directory path from there.

It is also possible to specify a target window or frame for a hyperlink where the link page can be opened.

If no target is specified, the link will be opened in the same window/frame.

The syntax for the page to be displayed in the same window/frame is given is below, here the target has to be set as "self"):

Go To Page 1

The target window options are,

_self : The link page opens in the same window and same frame.

_top : The link page opens in the same window, taking the full window if there is more than one frame.

_parent : The link page opens in the parent frame.

_blank : The link page opens in a new window.

2.4 HYPERLINKS - EXAMPLES

This is an example of HTML links and the specific code that we can use in the design of a website.

```
<a href="form.html">Fill Our Form</a> <br />
<a href="../parent.html">Parent</a> <br />
<a href="stuff/cat.html">Catalog</a> <br />
<a href=http://www.xbg.org  target="_blank">BASD</a> <br />
<a href="mailto:bugs@example.com?subject=Bug Report">Please report bugs here (by email only)</a>
<br />
<a href="apply-now.html"><img src="apply-now-button.jpg" /></a> <br />
<a href="../english/index.html">Switch to English version</a> <br />
```

In this example, the text "Fill Our Form" becomes a hyperlink to a page called "form.html". The text "Parent" signifies a link to a page one level higher. The text "Catalog" is a site-root-relative link that follows the path from directory 'stuff' to cat.html. the text "BASD" opens the page in a new window. The text "Please report bugs here (by e-mail only)" is a link to an email address with a specified subject. Next is a link which hovers over an image. The text " Switch to English version" is a link which goes one level higher and follows the path from the directory 'english' to 'index.html'.

The output of the above code is displayed in Figure 6.1.

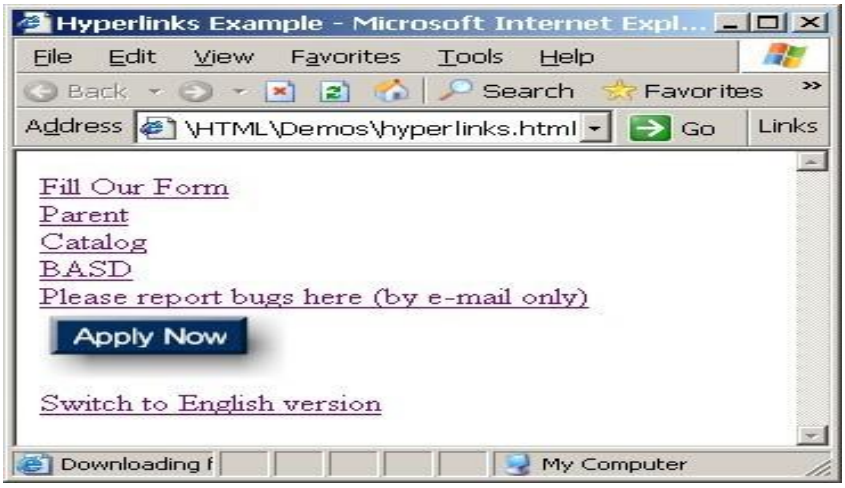


Figure 2.1 Hyperlinks

2.5 LINKS TO THE SAME DOCUMENT – Example

We can also create links to navigate within the document and jump to specific parts of a Web page. This is practical when our website has long pages. Scrolling or navigating within the page is called bookmarking the page. To make a bookmark in the web page, we must first create the bookmark with the "id" attribute and then add a link to it from within the same page. When the link is clicked, the page will scroll to the location with the bookmark.

To create a bookmark with the id attribute, specify id="section1" as shown below,

```
<h2 id="section1">Introduction</h2>
```

To add a link to the bookmark ("Introduction"), from within the same page specify the same id.

```
<p><a href="#section1">Introduction</a><br />
```

Following is an example to explain this concept. There are three sections within the web page as 'Introduction', 'Some Background' and 'History of the project'. These sections are created by bookmarking them with ids such as 'section1', 'section2' and 'section3'. Then we create links by specifying the ids of the sections in the page.

```
<h1>Table of Contents</h1>
<p><a href="#section1">Introduction</a><br />
<a href="#section2">Some background</A><br />
<a href="#section2.1“>History of the project</a><br />
...the rest of the table of contents...
```

<!-- The document text follows here -->

<h2 id="section1">Introduction</h2> ... Section 1 follows here ...

<h2 id="section2">Some background</h2>

... Section 2 follows here ...

<h3 id="section2.1">Project History</h3> ... Section 2.1 follows here ...

The following Figure 6.2 shows the output of the above code for how to link to an element within a page with a specified id.

Figure 2.2 Link to an element with a specified id within a page

CHECK YOUR PROGRESS

True/False type questions

1. HTML provides a feature called Links that allows users to navigate from page to page by clicking on words, phrases and images.
2. The Browsers are called hyperlinks.
3. A webpage can contain various links that takes us directly to other pages and even specific parts of a given page.
4. Hyperlinks allow visitors to navigate between Web sites.
5. We can create links to navigate within the document and jump to specific parts of a Web page.

Answers-

1. True
2. False

3. True
4. True
5. True

2.6 FRAMES AND FRAMESETS

HTML frames are useful when we need to divide the browser window into multiple sections and to load a separate HTML document in each section. These collections of HTML frames in the browser window are called as a frameset. The browser window is divided into frames as rows and columns in the same way as the tables are organized as rows and columns.

HTML frames are used to display documents in multiple views, which may be in independent windows or in sub windows. Multiple views on a HTML frame offer web designers a way to keep certain information visible, while other views are scrolled or replaced. Frameset provides us a promising solution when the data in some parts of a page needs to be changing while the other remains static. For example within the same window, the page header, banner section and footer section may remain static for many pages and the main document may be scrolled through or replaced by navigating in another frame.

- A frameset partitions a web browser window so that multiple web documents can be displayed simultaneously.
- We divide a document into frames using the <frameset> element
- Only the <frame> element and other <frameset> elements are the items that can be placed inside a <frameset> element
- The <frameset> element replaces the <body> element that is used in non-frame documents.
- Frames are created in horizontal rows and vertical columns.

The <frameset> Tag

The <frameset> tag is the container tag that requires a closing </frameset> tag. This tag divides the browser window into rectangular subspaces called frames. The <frameset> element contains one or more <frameset> or frame elements, along with an optional <noframes> element to provide alternate content for browsers that do not support frames or have frames disabled. This tag also determines the frame types as Column frames and Row frames. Based on the sizes specified for the rows and cols attributes the sub frames are defined with the respective dimensions in the frame set. The rows and cols attribute takes the dimension as comma-separated list of length that is specified in pixels, or as a percentage, or

as a relative length. The row dimension splits the frames from top to bottom in height and the cols attribute gives the width of each column from left to right.

Columns Example

The Figure 6.3 shows how the frames are divided into columns by specifying the cols attribute.

This frameset was created by the following code:

```
<frameset cols="35%,65%"> </frameset>
```

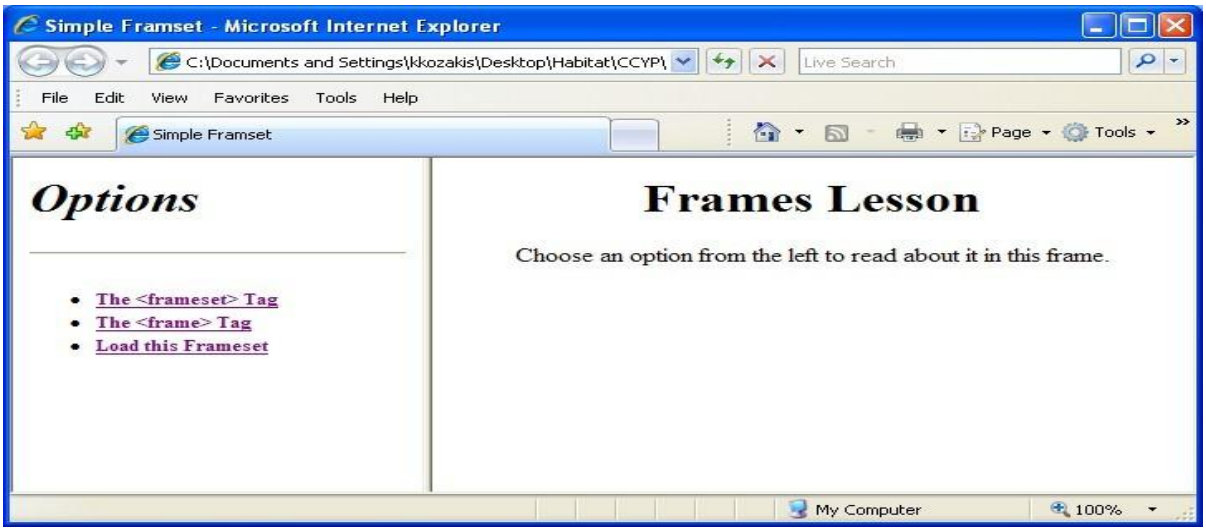


Figure 2.3 Example of Column Frame

Rows Example

The Figure 6.4 shows how the frames are divided into rows by specifying the rows attribute.

This frameset was created by the following code:

```
<frameset rows="180,*"> </frameset>
```

Figure 2.4 Example of Row Frames

The <frame> Tag

The <frame> tag defines the content in each frame and it is placed between the <frameset> </frameset> tags. The src attribute specifies the file that will appear in the frame.

In the following example, the page that will appear in the top frame is the file first.html, and the page that will appear in the lower frame is second.html.

```
<frameset rows="180,*">
    <frame src="first.html"/>
    <frame src="second.html"/>
</frameset>
```

Framesets

A complete example of using <frameset> and <frame> elements are shown in the following code. The browser window is divided into two columns using the cols attribute. The <frame> element defines the content "navigation.html" to be displayed in the left column and the "intro.html" to be displayed in the right column. The output of the code is shown in Figure 6.5.

```
<html>
 <head><title>Frames 1</title></head>
 <frameset cols="140,*">
  <frame name="navF" src="navigation.html">
  <frame name="mainF" src="intro.html">
 </frameset>
```

</html>

Figure 2.5 Example of <frameset> and <frame> element

In the code we can see the <frameset> element replaces the <body> element. The <frameset> has attributes cols or rows, can be defined in terms of pixels, percentage(%) or unspecified (*) which splits the window into two or more columns or rows.

2.7 FRAME ATTRIBUTES

Let us now consider the following example to explain about the frame attributes.

```
<frameset cols="140,*">
  <frame name="navF" src="navigation.html">
  <frame name="mainF" src="intro.html">
</frameset>
```

The name attribute uniquely identifies the frame. It is used as the target in an anchor (<a>) element. The src attribute specifies the web page to be placed in the frame initially and it may subsequently be overwritten also.

The scrolling attribute ("auto", "yes", "no") specifies whether the frame is to have scroll bars.

The frameborder attribute ("0", "1") specifies whether the frame is to have a border.

The code of the two HTML files that are displayed in the frames are given below.

navigation.html

The anchor tag <a> has a target attribute that takes the name of the frame used to display the information. Here in this example all anchors below are targeted to the "mainF" frame.

```
<html><head><title>Navigation Bar</title></head>
 <body><center>
  <a href="course.html" target="mainF">HTML Course</a><br><br>
```

```
<a href="paragraph.html" target="mainF">Paragraphs</a><br>
<a href="headings.html" target="mainF">Headings</a><br>
<a href="ulists.html" target="mainF">Unordered lists</a><br>
<a href="olists.html" target="mainF">Ordered lists</a><br>
<a href="nlists.html" target="mainF">Nested lists</a><br>
<a href="intro.html" target="mainF">Home</a><br>
</center></body>
</html>
```

intro.html

The HTML file "intro.html" is a simple document which is initially placed in the "mainF" frame. This is replaced when a user clicks on a link in the "navF" frame.

```
<html>
<head><title>Internet Computing</title></head>
<body>
<h2>Welcome to the HTML Course</h2>
<p>
<h4>Please select the subject of...</h4>
</p>
</body>
</html>
```

2.8 NESTED FRAMESETS

Nested framesets can be used to divide the window into vertical frames and horizontal frames. We can nest the <frameset> element to create a grid of frames on our Web page.

Below is an example code to create two vertical frames and two horizontal frames in the right vertical frame.

```
<html>
<head><title>Frames 2</title></head>
```

```
<frameset cols="140,*">
 <frame name="navF“ src="navigation.html">
        <frameset rows="30%,70%">
        <frame name="upperF" src="intro.html">
        <frame name="lowerF" src="course.html">
         </frameset>
</frameset>
</html>
```

The output of using nested framesets is shown in the below Figure 6.6.

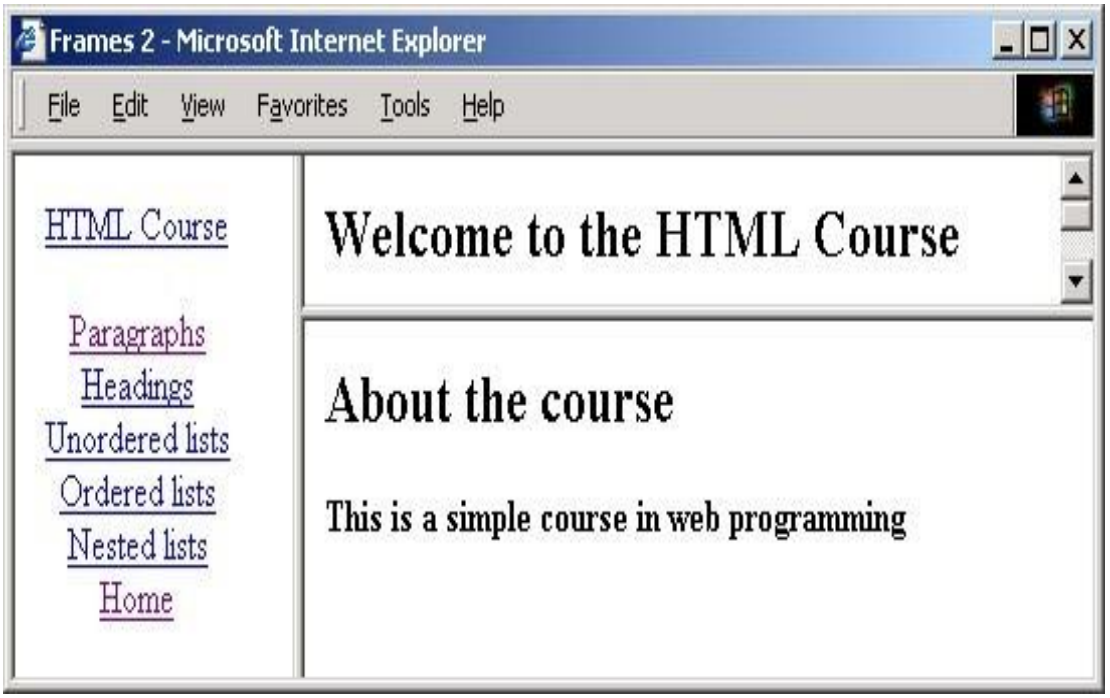


Figure 2.6 Example of Nested Frames

Noframes

Some browsers cannot process frames. Hence in order to display an alternative content in such cases, we can use the <noframes> element.

```
<html>
 <head><title>Frames 1</title></head>
```

```
<frameset cols="140,*">
 <frame name="navF" src="navigation.html">
 <frame name="mainF" src="intro.html">
</frameset>
<noframes>
 <body>
  Something here for browsers not supporting frames
 </body>
</noframes>
</html>
```

<frame> noresize attribute

If a frame has visible borders, the user can resize it by dragging the border. But in order to prevent a user from doing this, we can add the attribute noresize="noresize" to the <frame> tag.

```
<frameset cols="50%,50%">
  <frame src="frame_a.htm “noresize="noresize">
  <frame src="frame_b.htm">
</frameset>
```

Inline Frames

Inline frames are used to inserts an HTML document inside another. They are also called as "floating frames“. They are created with the <iframe> tag. The browser reads the <iframe> tag from the file, then makes a separate request to the server for the embedded file.

Inline frames are useful for web documents in which all content will remain stable, except for one section (e.g., a weekly special) i.e., the frequently changed section can be an inline frame, which can be quickly modified when necessary without editing the entire page. They are useful when documents that you prefer has to be embedded in a page instead of placing on a separate page or providing as a download (such as text or a PDF).

Simple HTML page with inline frame is given in the below code.

```
<h1>iFrame Example</h1>
    <p><strong>This text is found in frame.html </strong><p/>
    <iframe src="embedded.html" scrolling= "yes">
        Your browser does not support frames.
    </iframe>
    <p><strong>This text is also found in iframe.html. </strong></p>
```

The output of the code is shown in the Figure 2.7.

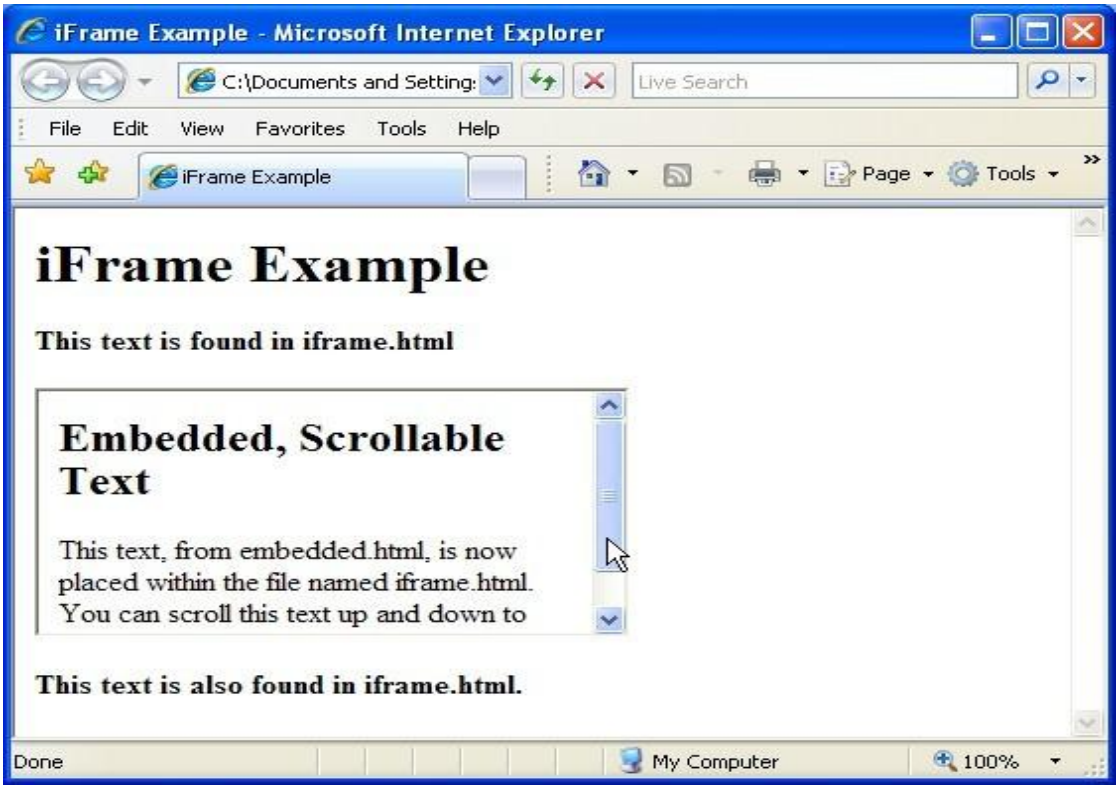


Figure 2.7 Example of <iframe>

Images

Images can be inserted in our web page by using <img> tag. Images are placed with <img> tags, with no closing tag.

The basic syntax is:

```
<img src="source_file" title="tool tip text">
```

The attribute src specifies the path to the image stored as a local file or the path to a file in a different directory under the HTML root directory, or a URL.

The title attribute is used to display the text "tool tip text" when the mouse hovers over the image, or if for some reason the image won't display. It is also very useful for the visually impaired.

Image attributes:

src	Location of image file (relative or absolute)
alt	Substitute text for display (e.g. in text mode)
height	Number of pixels of the height
width	Number of pixels of the width
border	Size of border, 0 for no border

Following is the simple syntax to use this tag.

```
<img src="./php.png" alt="PHP Logo" />
```

2.9 CONCLUSION

This module explains about HTML Hyperlinks. The module also discusses and explains in detail about HTML Frames and the purpose of inline frames. Finally it discusses about HTML images.

2.10 CHECK YOUR PROGRESS

Fill in the blanks

1. HTML ..... are useful when we need to divide the browser window into multiple sections and to load a separate HTML document in each section
2. .....are used to display documents in multiple views, which may be in independent windows or in sub windows.
3. Nested framesets can be used to divide the ..... into vertical frames and horizontal frames
4. If a frame has visible borders, the user can resize it by.....the border.
5. .....frames are used to inserts an HTML document inside another.

2.11 ANSWER CHECK YOUR PROGRESS

1. frames
2. HTML frames
3. window
4. dragging
5. Inline

2.12 MODEL QUESTION

1) What is HTML5? How is different from HTML4? What are its characteristics? Explain.

2) What are the different Support Elements of HTML5? Explain.

3) What do you understand by Semantics? Explain the difference between Semantic and non-semantic element? Give example.

4) What are the major benefits of HTML5? How will you migrate from HTML4 to HTML5? Explain.

5) Explain different Style Guide Media? Give proper syntax or example.

6) What are the different tips and tricks of Sample Web Page Designing? Explain.

7) What is the importance of Div tag in HTML? How is it used? Give suitable example.

8) What do understand by the Responsive web design? What is the effect of responsive web design? How it will affect the look and feel of your web page? Explain.

9) What is the difference between Static and Dynamic Websites? How Dynamic website is better than the Static web site? Explain.

10) What are the advantage and disadvantages of static and dynamic web site? Explain with help of some example.

2.13 REFERENCES

- https://epgp.inflibnet.ac.in/Home/ViewSubject?catid=7

2.14 SUGGESTED READINGS

- Powell. Thomas A., HTML & CSS: The Complete Reference, 5E
- Holzner. Steven, PHP: The Complete Reference
- Powell. Thomas A., JavaScript: The Complete Reference
- lemay. Laura, rafe colburn, jennifer kyrnin, Mastering HTML, CSS & JavaScript Web, BPB Publication, 2016
- vishvajeet. Sisodia, Basic of Web Design, HTML, CSS3, Centrum Press, 2014.

UNIT-III HTML TABLES

3.1 Learning Objectives

3.2 Introduction

3.3 HTML Tables

3.4 Tables Attributes

3.5 Rows and Columns

3.6 Introduction to Forms

- 3.6.1 The <FORM> tag
- 3.6.2 The <INPUT> tag
- 3.6.3 Text input

3.7 Summary

3.8 Check your progress

3.9 Answer Check your progress

3.10 Model Question

3.11 References

3.12 Suggested readings

3.1 LEARNING OBJECTIVES

The last module provides an introduction about the different Markup languages and discusses about the basic tags in HTML such as Formatting, Headings, Paragraphs and Lists in HTML. This module explains about Tables and its attributes in HTML. Moreover, this module also explains about HTML Forms.

3.2 INTRODUCTION

Tables provide a means of organizing the layout of data. A table is divided into rows and columns and these specify the cells of the table. Cells can contain text, images, links, other tables etc. The HTML tables allow web developers to arrange data like text, images, links, other tables, etc. into rows and columns of cells. Tables can also be used for organizing the layout of the web page itself. These HTML tables can be 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. In simple, we can understand that the HTML Table Element <table> represents data in two dimensions or more.

3.3 HTML TABLES

The <TABLE></TABLE> element has four sub-elements:

1. Table Row<TR></TR>.
2. Table Header <THEAD></THEAD> and Table Footer <TFOOT></TFOOT>.
3. Table Data <TD></TD>.
4. Caption <CAPTION></CAPTION>.

The Table rows can be grouped into a head, foot, and body sections using the THEAD, TFOOT and TBODY elements, respectively. Head section also called the header cell is defined with a <thead> element that contains the header information such as column names.

The <thead> tag must be used as a child of a <table> element, after any <caption>, and <colgroup> elements, and before any <tbody>, <tfoot>, and <tr> elements.

The Foot section is defined with a <tfoot> element are the footer rows that appears at the bottom of the table. The purpose of this <thead> and <tfoot> elements are that when long tables are printed, the head and foot information may be repeated on each page that contains table data.

The Table body is defined with a <tbody> element. Otherwise they are called as data cells defined with <td> element.

We can use at most one <thead> or <tfoot> but we can use multiple <tbody> elements in a table.

The caption tag is the descriptive title and used to provide a short description about the table's purpose or it can be used to name the table.

Example:

The anatomy of how a table can be created in HTML is shown in the below code.

```
<table border="1">
<tr>
        <th> Column 1 header </th>
        <th> Column 2 header </th>
</tr>
<tr>
        <td> Row1, Col1 </td>
        <td> Row1, Col2 </td>
</tr>
<tr>
        <td> Row2, Col1 </td>
        <td> Row2, Col2 </td>
</tr>
```

</table>

In the above example the table is defined with the <table> tag. Tables are divided into table rows with the <tr> tag. Table rows are divided into table data with the <td> tag. A table row can also be divided into table headings with the <th> tag. By default, all major browsers display table headings as bold and cantered:

We can also use <th></th> as row headings as well as column headings. One of the basic differences between <thead> and <th> is that <thead> is a block level element, whereas <th> is an inline-block. <th> also has a special attribute, scope so we can designate what it is a heading of, the column or the row.

3.4 TABLES ATTRIBUTES

Some of the table attributes that can be used with the <table> element is given below.

- **BGColor:** Some browsers support background colors in a table.
- **Width:** We can specify the table width as an absolute number of pixels or a percentage of the document width. We can set the width for the table cells as well.
- **Border:** We can choose a numerical value for the border width, which specifies the border in pixels.
- **CellSpacing:** Cell Spacing represents the space between cells and is specified in pixels.
- **CellPadding:** Cell Padding is the space between the cell border and the cell contents and is specified in pixels.
- **Align:** Tables can have left, right, or center alignment.
- **Background:** Background Image, will be titled in IE3.0 and above. BorderColor, BorderColorDark.

Example1: HTML Table: table1.html

```
<table border = "1" width = "40%"    summary = "This table provides information about the price of fruit">
    <caption><strong>Price of Fruit</strong></caption>
    <thead>
        <tr>
            <th>Fruit</th>
```

```
                <th>Price</th>
            </tr>
        </thead>
        <tfoot>
            <tr>
                <th>Total</th>
                <th>$3.75</th>
            </tr>
        </tfoot>

        <tbody>
            <tr>
                <td>Apple</td>
                <td>$0.25</td>
            </tr>
            <tr>
                <td>Orange</td>
                <td>$0.50</td>
            </tr>
            <tr>
                <td>Banana</td>
                <td>$1.00</td>
            </tr>
            <tr>
                <td>Pineapple</td>
                <td>$2.00</td>
            </tr>
```

```
    </tbody>
  </table>
```

The above code displays the table with a caption as "Price of Fruit". The table has a <thead> element and a <tfoot> element. The table body starts with a <tbody> element which holds the content or data of the table. The table body has four rows. Table attributes such as <border>, <width> and <summary> has been used in the example. The attribute <summary> is new in HTML5. The output of the above code is shown in the below Figure 5.1.

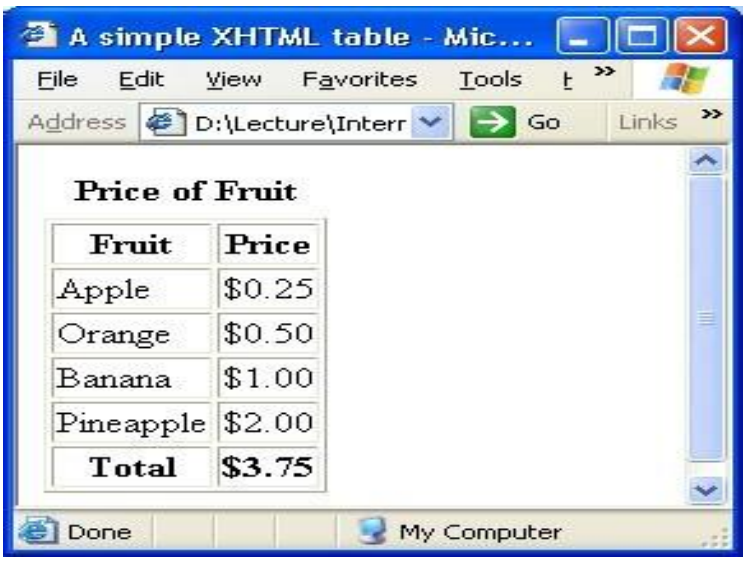


Figure 5.1

Example 2: HTML Table: table2.html

```
<html>
    <head><title>Table Cells</title></head>
    <body>
    <table cellspacing="15" cellpadding="0">
        <tr>
            <td>First</td>
            <td>Second</td>
        </tr>
    </table>
  <br/>
```

```
<table cellspacing="0" cellpadding="10">
    <tr>
        <td>First</td>
        <td>Second</td>
    </tr>
</table>
</body>
</html>
```

Example 2 explains about the table attributes cellspacing and cellpadding. The example specifies that there are two tables to differentiate and understand about cell spacing and cell padding. The first table has a cell spacing of 15 pixels that represents the space between cells. The second table has a cellpadding of 10 pixels which is the space between the cell border and the cell contents. The output of this HTML table is shown in the below Figure 5.2.

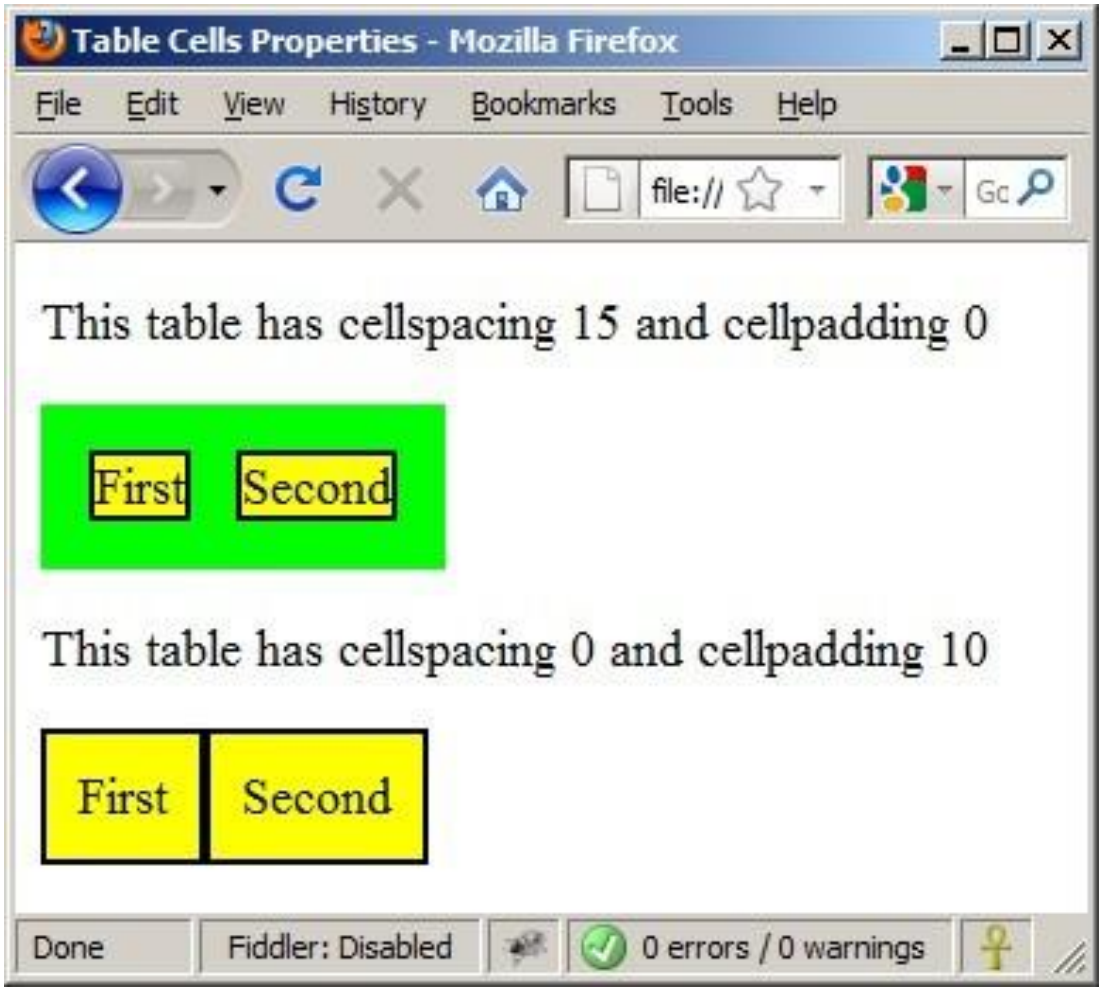


Figure 5.2

3.5 ROWS AND COLUMNS

Cells can span multiple columns and multiple rows with the colspan and rowspan attributes. In order to make a cell span more than one column we can use the colspan attribute and to make a cell span more than one row we can use the rowspan attribute.

An example to show that how we use the colspan and rowspan attributes. The table is defined with three rows. The first row specifies the heading where the heading cell 'name' spans over two columns. In the second row, the data cell 'Fishing' spans over two rows. The output of the below code is shown in FIgure 5.3.

Example 3:

```
<table border="1">
        <tr>
                <th colspan="2">Name</th>
                <th>Course</th>
                <th>Year</th>
        </tr>
        <tr>
                <td>A B</td>
                <td>Morgan</td>
                <td rowspan="2">Fishing</td>
                <td>5</td>
        </tr>
        <tr>
                <td>D J</td>
                <td>Jones</td>
                <td>Sailing</td>
                <td>8</td>
        </tr>
        <tr>
</table>
```

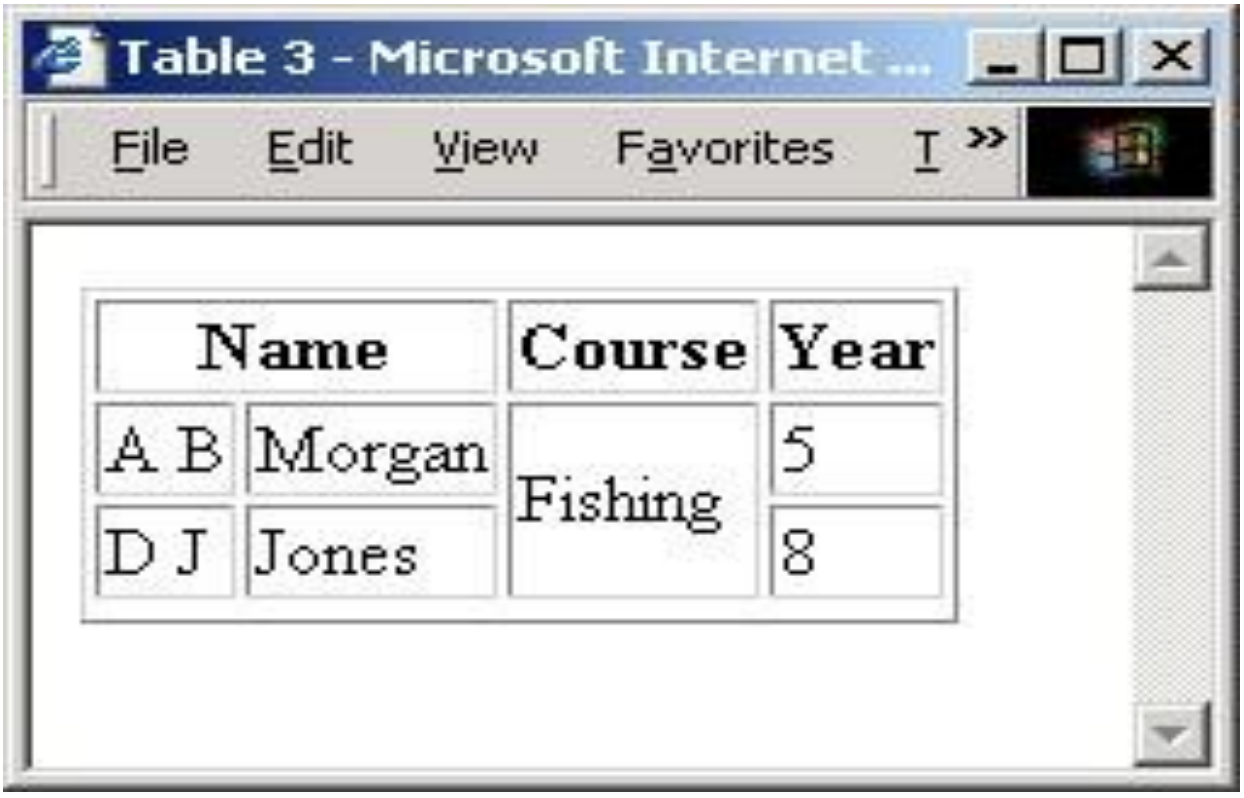


Figure 5.3

Example 4:

The table is defined with four rows. In the first row, the heading cell 'name' spans over two columns and the heading cell 'course' and 'year' spans over two rows. The second row has the data cell 'last' and 'init'.

The next two rows doesn't span over columns or rows. The output of the below code is shown in FIgure 5.4.

```
<table border="1" align="center">
        <tr>
                <th colspan="2" width="60%">Name</th>
                <th rowspan="2">Course</th>
                <th rowspan="2">Year</th>
```

```
        </tr>
        <tr>
                <th>Last</th>
                <th>Init.</th>
        </tr>
        <tr>
                <td>Morgan</td>
                <td>AB</td>
                <td>Fishing</td>
                <td align="center">5</td>
        </tr>
<!– etc -->
```

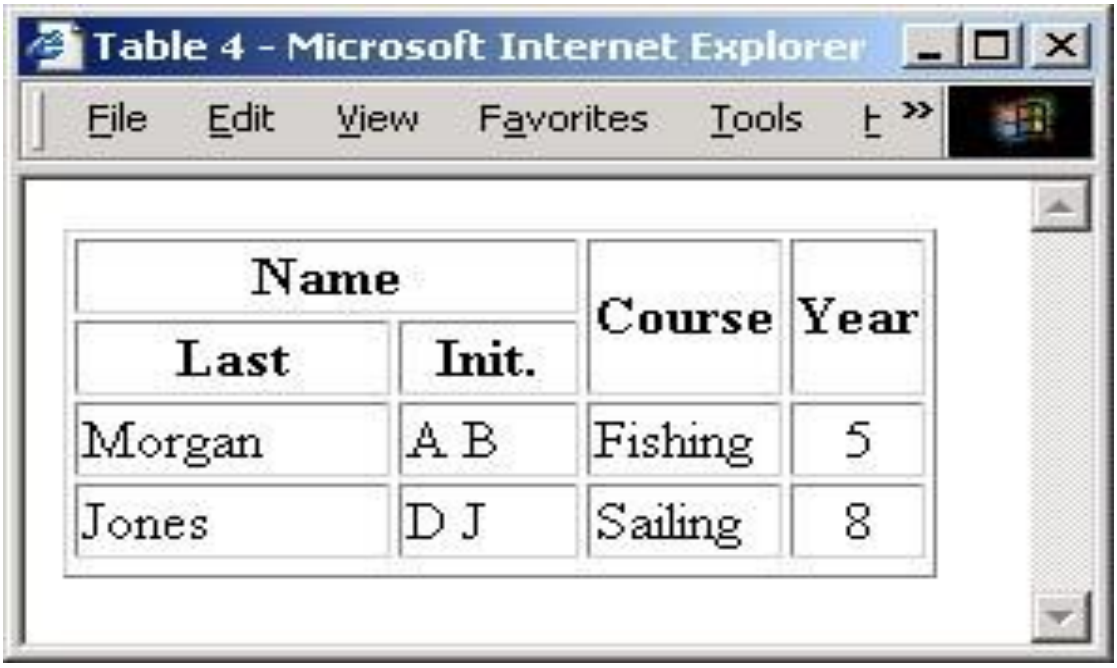


Figure 5.4

3.6 INTRODUCTION TO FORMS

An HTML form is an area of a document that contains normal content, markup, special elements called form elements or controls such as checkboxes, radio buttons, menus, text fields etc., and labels on those controls. Forms are used to create (rather primitive) GUIs on

Web pages. Users generally use a form in a web page to send information to a web server or a mail server by entering text, selecting menu items, etc., for processing.

<form> is just another kind of HTML tag.The syntax to create a form is

<form parameters> ...form elements... </form>

Form elements include: buttons, checkboxes, text fields, radio buttons, drop-down menus, etc. Other kinds of tags can be mixed in with the form elements.

A form usually contains a Submit button to send the information in the form elements to the server

The form's parameters tell JavaScript how to send the information to the server. There are two different ways it could be sent to the server.

Forms can also be used for other things, such as a GUI for simple programs

3.6.1 THE <FORM> TAG

The <form arguments> ... </form> tag encloses form elements and probably other elements as well.

The arguments to form tell what to do with the user input. Below is the list of arguments that can be used with the <form> tag.

– action="url" (required)

- Specifies where to send the data when the Submit button is clicked

– method="get" (default)

- Form data is sent as a URL with ?form_data information appended to the end of the URL. This method can be used only if data is all ASCII and not more than 100 characters

– method="post"

- Form data is sent in the body of the URL request and cannot be bookmarked by most browsers

– target="target"

- Specifies where to open the page sent as a result of the request. If 'target=_blank' means the page has to be opened in a new window. If 'target= _top' means use the same window when the page is opened.

CHECK YOUR PROGRESS

True/False type questions

1. A Form is divided into rows and columns and these specify the cells of the form.
2. HTML tables can be 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.
3. An HTML form is an area of a document that contains normal content, markup, special elements called form elements or controls such as checkboxes, radio buttons, menus, text fields etc., and labels on those controls.
4. A Table usually contains a Submit button to send the information in the form elements to the server
5. A label tag will bind the text to the control.

Answers-

1. False
2. True
3. True
4. False
5. True

3.6.2 THE <INPUT> TAG

Most, but not all, form elements use the input tag, with a type="..." argument to tell which kind of element it is. The type can be text, checkbox, radio, password, hidden, submit, reset, button, file, or image.

Other common input tag arguments include:

- name: the name of the element.
- id: a unique identifier for the element.
- value: the “value” of the element; used in different ways for different values of type.
- readonly: the value cannot be changed.
- disabled: the user can’t do anything with this element.

Other arguments are defined for the input tag but have meaning only for certain values of type

3.6.3 TEXT INPUT

A text field can be created in a form using the following syntax.

```
<input type="text" name="textfield" value="with an initial value" />
```

The output of the above code is displayed in Figure 5.5.

Figure 5.5

A multi-line text field can be created in a form using the below syntax.

```
<textarea name="textarea" cols="24" rows="2">Hello</textarea>
```

The output of the above code is displayed in Figure 5.6.

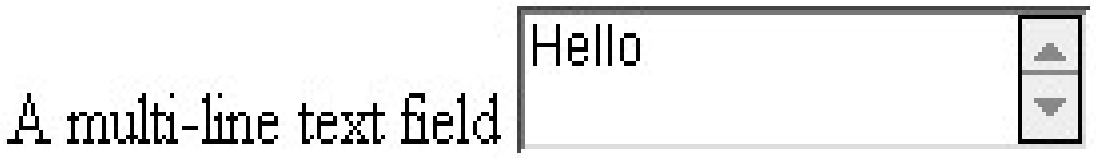


Figure 5.6

Note that two of these use the input tag, but one uses textarea.

Password field

```
<input type="password" name="textfield3" value="secret" />
```

The output of the above code is displayed in Figure 5.7.

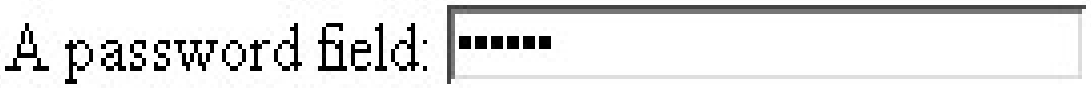


Figure 5.7

Buttons

A submit button can be created using the below syntax

```
<input type="submit" name="Submit" value="Submit" />
```

A reset button can be created using the below syntax

```
<input type="reset" name="Submit2" value="Reset" />
```

A plain button can be created using the below syntax

```
<input type="button" name="Submit3" value="Push Me" />
```

The output of the above code is displayed in Figure 5.8.

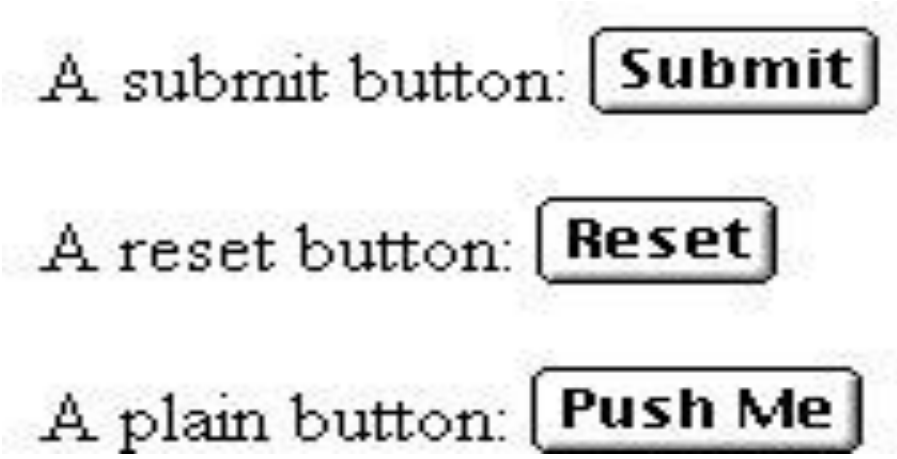


Figure 5.8

Radio buttons

Radio buttons are used when it is required to select one option from a set of alternatives. Radio buttons can be created using the below syntax,

Radio buttons can be created using the below syntax,

Radio buttons:

```
<br>
<input type="radio" name="radiobutton" value="myValue1" />male<br>
<input type="radio" name="radiobutton" value="myValue2”  checked="checked" />female
```

The output of the above code is displayed in Figure 5.9.

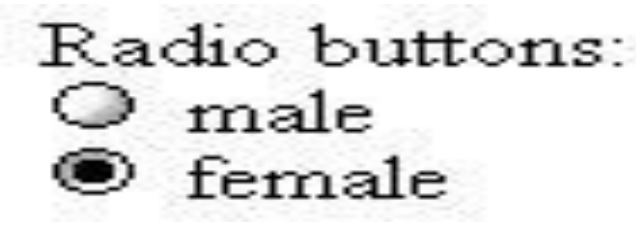


Figure 5.9

When a user clicks on a radio-button, it becomes checked, and all other radio-buttons with equal name become unchecked.

Labels

In many cases, the labels for controls are not part of the control. For example,

```
<input type="radio" name="gender" value="m" />male
```

In this case, clicking on the word “male” has no effect.

A label tag will bind the text to the control

```
<label><input type="radio" name="gender" value="m" />male</label>
```

With the above syntax, clicking on the word “male” now clicks the radio button.

Checkboxes

Checkboxes are used when more options are to be allowed at the same time. A checkbox can be created using the below syntax,

```
<input type="checkbox" name="checkbox" value="checkbox" checked="checked">
```

The output of the above code is displayed in Figure 5.10.

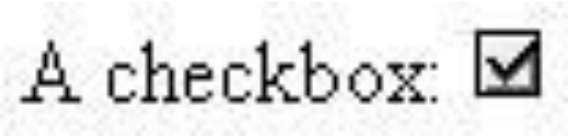


Figure 5.10

The attributes of this control are,

– type: "checkbox"

– name: used to reference this form element from JavaScript

– value: value to be returned when element is checked

Note that there is no text associated with the checkbox Unless we use a label tag, only clicking on the box itself has any effect.

Drop-down menu or list

A menu or list can be created using the following syntax,

```
<select name="select">
    <option value="red">red</option>
    <option value="green">green</option>
    <option value="BLUE">blue</option>
</select>
```

The <select> element is used to create a drop-down list and the <option> tags inside the <select> element define the available options in the list.

The output of the above code is displayed in Figure 5.11.

Figure 5.11

The additional attributes of this control are,

– size: the number of items visible in the list (default is "1")

– multiple

• If set to "true" (or just about anything else), any number of items may be selected.

• If omitted, only one item may be selected.

• If set to "false", behavior depends on the particular browser.

Hidden fields

Hidden fields are similar to text fields that does not show on the page.

A hidden field:

```
<input type="hidden" name="hiddenField" value="nyah">
```

<-- right there, don't you see it?

The output of the above code is displayed in Figure 5.12.

Figure 5.12

The purpose of this hidden field is that all input fields are sent back to the server, including hidden fields. This is a way to include information that the user doens't need to see or that we don't want others to see.The value of a hidden field can be set programmatically (by JavaScript) before the form is submitted

Now let us look at an example that includes the controls what we have seen earlier. Example

In this example, a form is created with a text field and radio buttons.

```
<html>
<head>
    <title>Get Identity</title>
</head>
<body>
    <p><b>Who are you?</b></p>
    <form method="post" action="">
        <p>Name:
            <input type="text" name="textfield">
        </p>
        <p>Gender:
```

```
                <label><input type="radio" name="gender" value="m“ />Male<label>
                <label><input type="radio" name="gender" value="f" />Female</label>
            </p>
        </form>
</body>
</html>
```

The output of the above code is displayed in Figure 5.13.

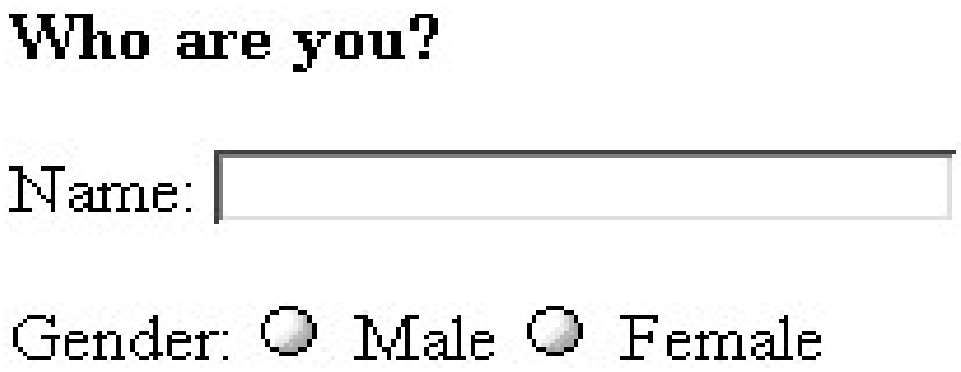


Figure 5.13

Let us now look at a complete example that includes all the controls. The form that is displayed in the web page is shown in Figure 5.14.

The list of all controls has been summarized as below,

- text
- checkbox
- radio (buttons)
- select (options)
- textarea
- password
- button
- submit
- reset

- hidden
- file
- image

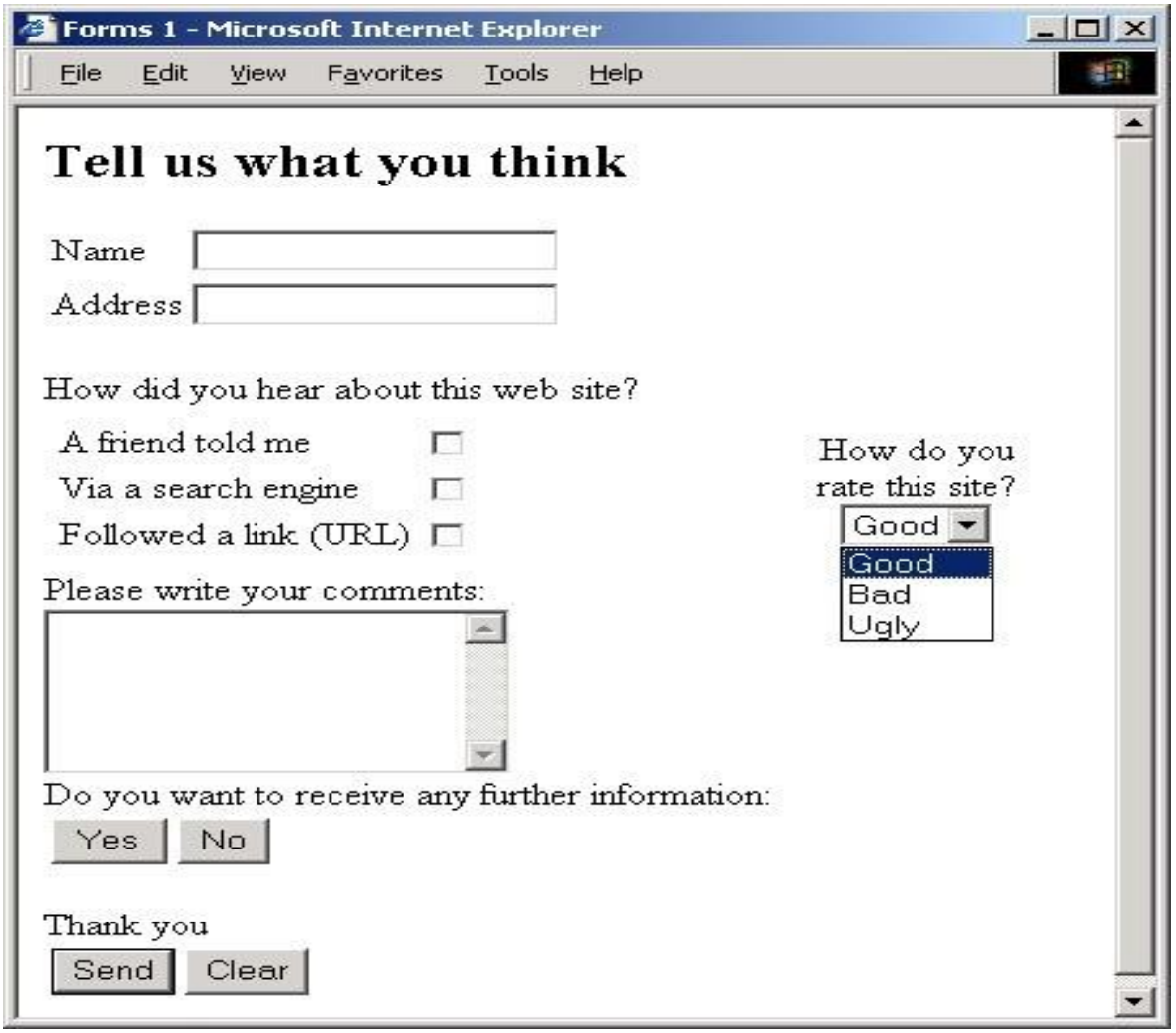


Figure 5.14

3.7 SUMMARY

In this module we looked at HTML in detail and explored about the Tables in HTML. This module also explains about the HTML Forms with syntax and examples in detail.

3.8 CHECK YOUR PROGRESS

1. We can choose a numerical value for the border width, which specifies the border in.....

2. ..... represents the space between cells and is specified in pixels.
3. .....is the space between the cell border and the cell contents and is specified in pixels.
4. In order to make a cell span more than one column we can use the .....attribute
5. To make a cell span more than one row we can use the ..... attribute.
6. Users generally use a.....in a web page to send information to a web server or a mail server by entering text, selecting menu items, etc., for processing.
7. .....elements include: buttons, checkboxes, text fields, radio buttons, drop-down menus, etc.
8. ..... .buttons are used when it is required to select one option from a set of alternatives.

3.9 ANSWER CHECK YOUR PROGRESS

1. Pixels
2. Cell Spacing
3. Cell Padding
4. Colspan
5. Rowspan
6. Form
7. Form
8. Radio

3.10 MODEL QUESTION

1. How will you create a table in HTML? How tables provide a means of organizing the layout of data?
2. The HTML tables allow web developers to arrange data like text, images, links, other tables into rows and columns of cells. Explain.
3. Cells can span multiple columns and multiple rows with the colspan and rowspan attributes.
4. What is Radio Button? When a user clicks on a radio-button, it becomes checked, and all other radio-buttons with equal name become unchecked. Explain.
5. What is Checkbox? Is checkboxes are used when more options are to be allowed at the same time? Explain.

3.11 REFERENCES

- https://epgp.inflibnet.ac.in/Home/ViewSubject?catid=7

3.12 SUGGESTED READINGS

- Powell. Thomas A., HTML & CSS: The Complete Reference, 5E.
- lemay. Laura, rafe colburn, jennifer kyrnin, Mastering HTML, CSS & JavaScript Web, BPB Publication, 2016.
- vishvajeet. Sisodia, Basic of Web Design, HTML, CSS3, Centrum Press, 2014.

UNIT-IV CASCADING STYLE SHEETS (CSS)

4.1 Learning Objectives

4.2 Introduction

4.3 Inline Style sheet

4.4 Internal Style Sheet

4.5 External Style sheet

4.6 Imported Style Sheet

4.7 Cascading Rules

4.8 Order rules

4.9 Order rules

4.10 Difference between ID and Classes

4.11 Anatomy of a style rule

4.12 Style Classes

4.13 Attribute Selector

4.14 Summary

4.15 Check your progress

4.16 Answer Check your progress

4.17 Model Question

4.18 References

4.19 Suggested readings

4.1 LEARNING OBJECTIVES

The last module provides us an understanding about how to access and manipulate an XML DOM tree using JAVA. The module also explains about the concept of parsing XML using Java platform. The objective of this module is to discuss about the CSS, its structure and how to use CSS in the webpage.

4.2 INTRODUCTION

CSS stands for "**Cascading Style Sheets**" which is a technology to control how elements are presented in the Web page.

The term **Cascading** refers to the procedure that determines which style will apply to a certain section, if we have more than one style rule.

The term **Style** refers to how you want a certain part of our page to look. We can set colors, fonts, alignment, borders, backgrounds, spacing, margins, and much more.

The term **Sheets** represents that the "sheets" are like templates, or a set of rules, for determining how the webpage will look.

So, CSS (all together) is a styling language that defines a set of rules to tell browsers how the webpage should look.

A style sheet is a document that contains style information about one or more documents written in markup languages. It enables us to control rendering of styles such as fonts, color, type face, size and other aspects of document style.

What is "Style"?

"Style" is a command that you set to tell the browser how a certain section of your webpage should look.

You can apply style on many HTML "elements like <p> <h1> <table> etc.

Advantages:

Most of the browsers cache external style sheets. So, once a style sheet is cached, there is no delay in document presentation.

The size of a document using external style sheet is comparatively smaller and hence, download time is also smaller. This speeds up overall response time.

How to write style rules?

There are two parts when we write style rules: (1) selector and (2) declaration.

Selector: It is the HTML element that you want to add style to. For e.g. <p> <h1> <table> etc.

Declaration: It is the statement of style for that element. It is made up of property and value.

They are defined as,

Selector {declaration;}

Declaration = {property: value;}

Property: defines what aspect you want to change. For e.g. color, font, margins, etc.

Value: defines the exact setting for that aspect. For e.g. red, italic, 40px, etc.

Examples:

h1 {color: red;}

h1 {color: blue; background-color: green;}

Where to put the style rules?

There are **4 ways** to attach CSS to a page:

Inline Style Sheet - CSS is not attached in the <header> but is used directly within HTML tags.

Internal Style Sheet - Best used to control styling on one page.

External Style Sheet - Best used to control styling on multiple pages.

Imported Style Sheet - To import CSS from other style sheets.

4.3 INLINE STYLE SHEET

The style information is incorporated directly into the HTML tags.

</head>

<body>

```
<p style= “text-align: center; font-weight: bold;
color: yellow;”>What was I thinking?</p>
</body>
```

4.4 INTERNAL STYLE SHEET

The style information is placed under the style tag in the head section of an HTML page.

```
<head>
    <title>My Wonderful Example</title>
    <style type=“text/css”>
    body
    {
        text-align: left;    font-family: trebuchet, verdana;
    }
    </style>
</head>
```

4.5 EXTERNAL STYLE SHEET

External style sheet is specified using the HTML <link> tag. The Style information is written in a separate file and is referenced from an HTML document.

An external style sheet is useful when the same style is applied on different documents.

```
<html>
<head><title>My Way</title>
<link rel="stylesheet" href="http://www.annauniv.edu/~myway.css" type="text/css“>
</head>
<body> </body>
</html>
```

4.6 IMPORTED STYLE SHEET

We can import the style sheet using @import statement. The import statement is used within the style tag in an HTML document as follows

<style> @import url("style.css"); </style>

Note: The import statement must be the first rule within a style tag.

Internal rules override the conflicting rules in the imported style sheets.

<style>

@import url("style.css"); p {color:green;}

</style>

The above mentioned style rule makes all paragraphs green even if style.css contains a different rule for paragraphs.

<style type="text/css" >

@import url(http://www.xxx.com/red.css);

@import url(/stylesheets/pink.css); >

</style>

A <style> tag may contain an arbitrary number of import statements, but the order in which the style sheets are imported is important in determining cascading styles.

4.7 CASCADING RULES

More specific rules gets preference over less specific rules. p b {color:green;} b{color:Red;}

In the following example the word 'hello' will be in red whereas the word 'world' will be in green color due to the above mentioned rule.

hello world

4.8 ORDER RULES

The following figure shows the rules for resolving conflicting styles

Inline Internal External Imported Browser's

Style Style Style Style Default style

----------→

Highest priority lowest priority

4.9 WORKING OF CSS

CSS works in conjunction with HTML. One way is that an HTML file (or multiple files) links to a CSS file (or multiple CSS files) and when the web browser displays the page, it references the CSS file(s) to determine how to display the content.

Another approach is that HTML elements are marked with "IDs" and "classes," which are defined in the CSS file and using this the browser knows which styles belong to which element. Each element type (<h1>, <img>, <p>, <li>, etc.) can also be styled with CSS. IDs and classes are defined by the person writing the code and there are no default IDs and classes.

CHECK YOUR PROGRESS

True/False type questions

1. The term Cascading refers to the procedure that determines which style will apply to a certain section, if we have more than one style rule.
2. The term Style refers to how you want a certain part of our page to look. We can set colors, fonts, alignment, borders, backgrounds, spacing, margins, and much more.
3. The term Sheets represents that the "sheets" are like templates, or a set of rules, for determining how the webpage will look.
4. A style sheet is a document that contains style information about one or more documents written in C languages.
5. The size of a document using external style sheet is comparatively smaller its download time is smaller, it speeds up overall response time.
6. In Inline Style Sheet CSS is attached in the <header> but is used directly within HTML tags.

Answers-

1. True
2. True
3. True
4. False

5. True
6. False

4.10 DIFFERENCE BETWEEN ID AND CLASSES

IDs and classes function the same way. They both can provide the same styling functionality to an HTML element, however IDs are unique; each element can only have one ID, and that ID can only be on the page once. Classes are not unique; an element can have multiple classes, and multiple elements can have the same class.

IDs can be used to style elements that are different from anything else on the page. Classes can be used to style multiple elements on a single page that have things in common, like font size, color, or style.

The styles for each element, ID, or class used on an HTML page are defined in a CSS document.

Elements are declared with the element (HTML) tag; styles for the element are wrapped with curly brackets like

```
h1 {          }
```

IDs are declared with a hash sign and the ID name; styles for the ID are wrapped with curly brackets like

```
#title { }
```

Classes are declared with a period and the class name; styles for the class are wrapped with curly brackets like

```
.bodytext {          }
```

4.11 ANATOMY OF A STYLE RULE

Selector

{

Property 1 : value 1;

Property 2 : value 2;

Property N : value N;

}

The selector specifies which HTML elements are affected by the style rule.

The declaration specifies what the effect will be.

Grouping Selectors

Group the same selector with different declarations together on one line.

h1 {color: black;} h1 {font-weight: bold;} h1 {background: white;}

The above declarations can be grouped as,

```
h1
{
        color: black;  font-weight: bold;  background: white;
}
```

Selectors having common declarations are grouped into a comma-separated list.

h1{color:red;} h2{color:red;} h3{color:red;}

The above declarations can be written as h1,h2,h3{color:red;}

Selectors

Selectors are patterns used to select the elements we want to style. Now we will look at the demonstration of the different selectors.

Type selectors

A type selector is a simple selector, which is the name of a document element and it matches every single element of the document.

For e.g.: The selector b selects every <b> element.

Universal selector

CSS has a special selector *, which matches with every single element in the document.

For eg: *{color:red;}

It makes all the text in the document red.

Compound Selectors

Selectors can be defined so that a style rule applies to an element only when it is a descendant of a certain other type of element. Examples:

ul ul { list-style-type : square }

This specifies that an unordered list inside another unordered list will be bulleted by squares.

h1 em em { color : red }

This specifies that emphasized text inside emphasized text in an <h1> header will appear in red. Compound selectors are more specific than simple selectors like p { color : red } div p { color : blue }

Descendant selectors

A descendant selector selects only those elements that are descendants of a particular element.

For e.g.

CascadingStyleSheet

descendantselectors

this isaparagraph

To select all the elements which are descendants of element we need to give as, p i b which selects the word 'paragraph' from the above text.

Child selectors

Child selectors select elements that are immediate children of a specified element.

For e.g.: Thisisaparagraph

The selector p>b selects only highlighted elements .

The selector p>i>b selects only the element whose parent is the element whose parent is ,in turn the element.

4.12 STYLE CLASSES

Style classes allow us to control which elements of a given type should use a style rule. This method has two parts:

In the style sheet, the selector defines the class name, which is preceded by a period. In the HTML, the tag includes the class attribute and specifies the value of the class name Example:

Define the nodec class for anchor tags as,

```
a.nodec  { text-decoration : none }
```

This suppresses the usual underlining. Use it in HTML like so,

```
<a class="nodec" href="somepage.html">Link text</a>
```

Style classes can also be “generic,” that is not tied to a specific element type.

Example:

Define the zowie class as,

```
.zowie  { text-decoration : blink }
```

Use it on an emphasized element as,

```
<em class="zowie">Important!</em>
```

Use it with no other style attributes as,

```
<span class="zowie">Buy Now!</span>
```

The and Tags

These tags are provided to allow arbitrary chunks of HTML to be treated as elements. This is usually done in order to apply style attributes to them, as in the preceding example.

A ... element defines an “inline” structure, i.e. it simply defines a stretch of text. Thus it can be used within a paragraph or table element without affecting the flow of the text.

A ... element defines a “block” structure. Usually the browser will place line breaks before and after this element, but otherwise it has no effect itself.

Pseudo-classes

These are like style classes, but an element acquires a pseudo-class by user action or by a relationship other than descendancy.

In the style sheet, a pseudo-class name is preceded by a colon.

In the HTML, the pseudo-class name is NOT used with the affected tag, because it is implied.

Pseudo classes match elements using the information other than their name, content or attribute such as states of an anchor element.

Pseudo elements , on the other hand, address sub-parts of an element such as the first letter of a paragraph.

The general form is

Selector:pseudo-class{declaration} Selector:pseudo-element{declaration}

Pseudo Classes and Elements

The :first-child, :last-child and only-child pseudo classes

The first-child pseudo class selects an element if it is the first child of its parent regardless of what this parent element is.

For eg: p:first-child selects all p elements that are the first children of any element.

The last-child pseudo class selects the last child element of any element.

For eg: p:last-child selects all p elements that are the last children of any element The only-child pseudo class selects an element if it is the only child of another element.

The anchor pseudo classes :link and :visited

The :link pseudo class applies to hyperlinks that have not been visited.

The :visited pseudo class applies to hyperlinks that have already been visited at least once.

For e.g.:

a:link {color :blue;} a:visited{color:red;}

The dynamic pseudo classes :hover, :active, :focus

The :hover pseudo class selects elements that are being designated by the user with a pointing device.

The :active pseudo class applies to an element that is currently being activated by the user. The :focus pseudo class applies to an element that has currently got the focus by keyboard events or other means.

a:focus {color:green;} a:hover{color:yellow;} a:active{color:pink;}

The :first-line pseudo element

The :first-line pseudo element allows us to add styles to the first line of an element.

For e.g.:

p:first-line{text-decoration:underline;} underlines the first line of a paragraph.

The :first-letter pseudo element

The :first-letter pseudo element is used to add style to the first letter of the first line of block elements.

For e.g.:

p:first-letter {font-size:300%; float:left;}

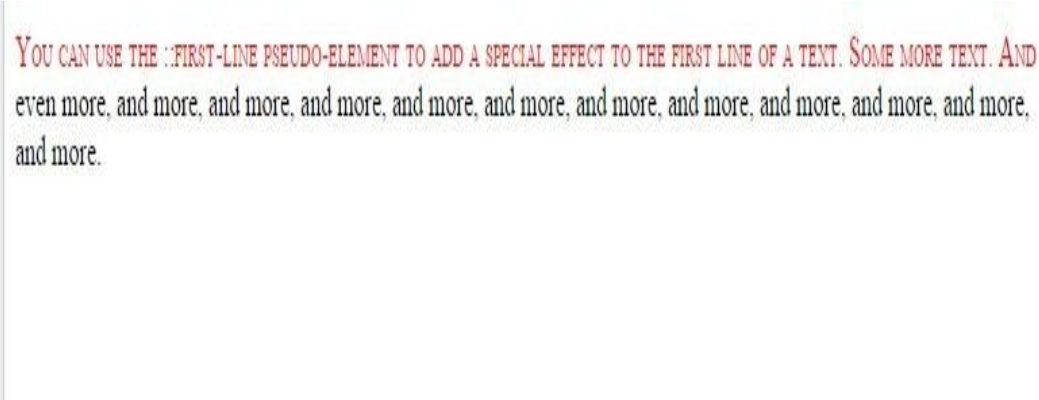
Example of Pseudo Classes – first-line

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

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.

</body>

</html>



The :before and :after pseudo elements

The :before pseudo element inserts some new content before an existing element The :after pseudo element inserts some new content after an existing element.

The content property is used to define the content to be inserted before and after the selected elements.

For e.g., :body > ol> li :before { content: ”chapter:”; }

Adds chapter before every <li> element which is a grandchildren of the <body> element.

4.13 ATTRIBUTE SELECTOR

Attribute Selector provide a way of selecting elements depending on the presence of an attribute or the presence of certain attribute values.

Simple Attribute Selector

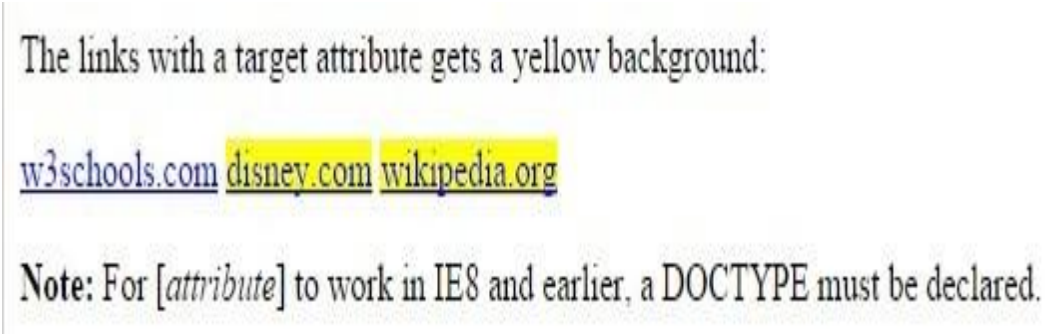
It selects elements having specified attribute.

General syntax : element[attribute_name] or [attribute_name]

For e.g.: a[href] selects all <a>elements having the attribute href.

Attribute Selector - Example

```
<!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="http://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 [<i>attribute</i>] to work in IE8 and earlier, a DOCTYPE must be declared.</p>
</body>
</html>
```



Attribute Value Selector

Selecting elements based on the attribute value.

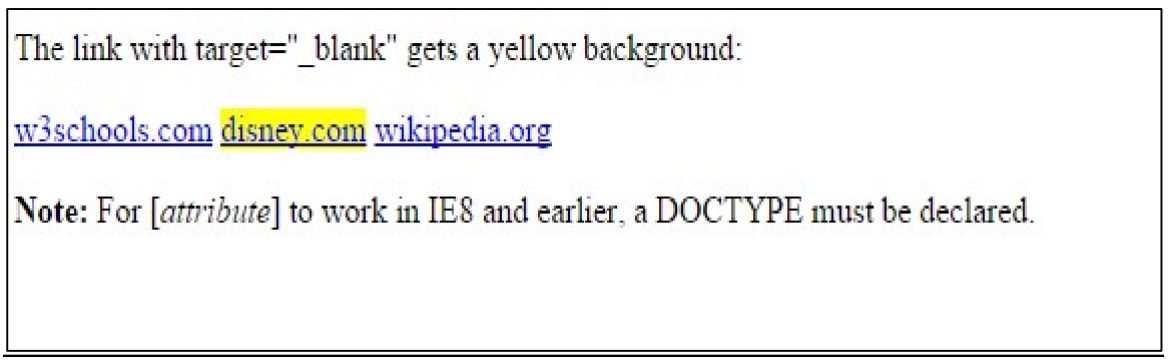
The syntax :

element[attribute_name="attribute_Value"] or [attribute_name="attribute_Value"]

Example:

p[type="note"] selects all <p>elements having the type attribute value "note".

```
<!DOCTYPE html>
<html>
<head>
    <style>
        a[target=_blank] {  background-color: yellow;}
    </style>
</head>
<body>
<p>The link with target="_blank" gets a yellow background:</p>
    <a href="http://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 [<i>attribute</i>] to work in IE8 and earlier, a DOCTYPE must be declared.</p> </body>
</html >
```



Starts-with attribute value selector

The selector selects elements having an attribute value that starts with the value specified.

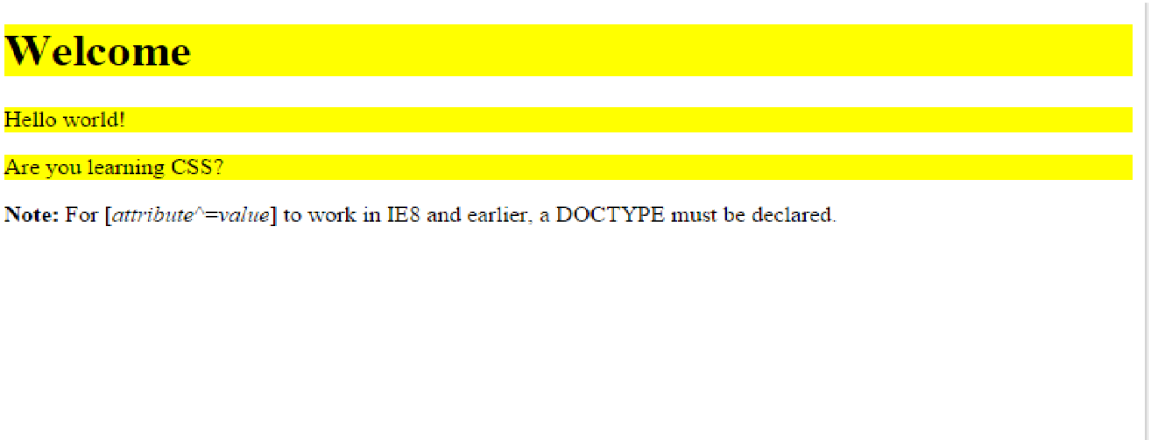
The general syntax :

Element[attribute_name^=“attribute_value”]

For example, the selector p[type^=“copy”] matches

copyright as well as copyleft..

```
<!DOCTYPE html>
<html>
<head>
	<style> [class^="top"]
	{
		background: yellow;
	}
</style>
</head>
<body>
	<h1 class="top-header">Welcome</h1>
	<p class="top-text">Hello world!</p>
	<p class="top-content">Are you learning CSS?</p>
	<p><b>Note:</b> For [<i>attribute</i>^=<i>value</i>] to work in IE8 and earlier, a DOCTYPE must be declared.</p>
</body>
</html>
```



Ends-with attribute value selector

This attribute selects elements having attribute value that ends with the value specified.

The general syntax :

Element[attribute_name$="attribute_value"]

For e.g., a[href$=".com"] - selects those anchor tags that point to .com websites and not any other domains.

Substring match attribute value selector

This selector selects those elements having an attribute value containing at least one occurrence of the value specified.

The general syntax :

Element[attribute_name*="attribute_value"] For e.g., a[href*="image"] - selects those anchor tags that have image in the href attribute.

The Class Selector

This selector is a specific case of one of many attribute value selectors with the attribute name class and "~=" substituted by "."

So p[class~="bold"] and p.bold are identical in meaning.

It matches .. as well as .. but not .. Class selectors are useful to control elements that belong to a group as well as to remove limitations of the selector.

The Class Selector - Example

```
<!DOCTYPE html>
<html>
<head>
    <style> .center
    {
        text-align: center;    color: red;
    }
    </style>
</head>
<body>
<h1 class="center">Red and center-aligned heading</h1>
<p class="center">Red and center-aligned paragraph.</p>
</body>
</html>
```

ID Selectors

The id differs from class in that id identifies a single element whereas class identifies a set of related elements. An id selector selects a single element based on its unique id attribute value

regardless of its position in the document tree. An id selector is defined by placing a # symbol before the selector name.

The selector p #para1 selects the p elements having id attribute value para1.

So it matches … but not .. .

Note: The id attributes should be unique Id selectors and other selectors can be combined.

For e.g.:

** <h1>Web Technology</h1> HTML **

And

#book1 h1{color:red;} makes the h1 elements in the div tag to be red.

ID Selectors - Example

```
<!DOCTYPE html>
<html>
<head>
    <style> #para1
    {
        text-align: center;     color: red;
    }
    </style>
</head>
<body>
    <p id="para1">Hello World!</p>
    <p>This paragraph is not affected by the style.</p>
</body>
</html>
```

4.14 SUMMARY

This module explains about CSS, adding different style sheets and its anatomy. The module also explores about Selectors, Pseudo Classes and elements. Moreover, the module also discusses about the Attribute Selectors, Class Selectors and ID selectors.

4.15 CHECK YOUR PROGRESS

1. CSS stands for .....which is a technology to control how elements are presented in the Web page.
2. ..... Best used to control styling on one page.
3. .....Best used to control styling on multiple pages.
4. .....To import CSS from other style sheets.
5. CSS works in conjunction with.....

4.16 ANSWER CHECK YOUR PROGRESS

1. Cascading Style Sheets
2. Internal Style Sheet
3. External Style Sheet
4. Imported Style Sheet
5. HTML

4.17 MODEL QUESTION

1. What is Cascading Style sheet? CSS is a styling language that defines a set of rules to tell browsers how the webpage should look. Explain how.
2. How Cascading Style sheet enable us to control rendering of styles such as fonts, color, type face, size and other aspects of document style?

3. What is external style sheet? How size of a document using external style sheet is comparatively smaller? How its download time is smaller?
4. What is the difference between Internal Style Sheet, External Style Sheet, and Imported Style Sheet?
5. How will you link a HTML file with CSS? What is the difference between ID and classes?

4.18 REFERENCES

- https://epgp.inflibnet.ac.in/Home/ViewSubject?catid=7

4.19 SUGGESTED READINGS

- Powell. Thomas A., HTML & CSS: The Complete Reference, 5E.
- lemay. Laura, rafe colburn, jennifer kyrnin, Mastering HTML, CSS & JavaScript Web, BPB Publication, 2016.
- vishvajeet. Sisodia, Basic of Web Design, HTML, CSS3, Centrum Press, 2014.

Block-II

UNIT-V eXtensible Markup Language (XML)

5.1 Learning Objectives

5.2 HTML – eXtensible Hyper Text Markup Language

5.3 Need for XHTML

5.4 XHTML – Base syntactic rules

5.5 XHTML – other syntactic rules

5.6 DTD – Document Type Definition

5.7 XML

5.8 XML Trees

5.9 Namespaces

5.10 Defining XML Data Formats

5.11 Summary

5.15 Check your progress

5.16 Answer Check your progress

5.17 Model Question

5.18 References

5.19 Suggested readings

5.1 LEARNING OBJECTIVES

The last module explains about Hyperlinks in HTML, Frames, how to define Frames and the purpose of the Frameset document. Moreover, the module also explains about HTML Images. In this module we will learn about XHTML, its syntax, tags, and document type definitions. We will learn how to use XHTML to create Web pages. In this module we will also understand the basics of creating an XML document

5.2 HTML – eXtensible Hyper Text Markup Language

XHTML is HTML defined as an XML application. It is identical to HTML but it is a stricter and cleaner HTML. It is compatible to HTML 4.0.1 and supported by all browsers. XHTML is used to define and organize the page content but not to format or style it.

XHTML uses the elements and attributes of HTML. Its uses the syntax of XML (eXtensible Markup Language).

5.3 Need for XHTML

XHTML combines the strength of HTML and XML. XHTML provides the web page with a more consistent and well-structured format so that the web pages can be easily parsed and processed by present and future web browsers. Also, XHTML pages can be rendered by all XML enabled browsers.

XHTML integrates the concepts of XML with HTML and hence it is relatively easy to introduce new elements or additional element attributes. It also conforms to the rules and standards of XML.

The following, “bad” HTML document will work fine in most browser even if it does not follow HTML rules:

```
<html>

<head>

<body>

<p>a paragraph…<br>

<a href=“#”>test
```

```
</html>
```

But browsers running on hand-held devices (e.g. mobile phones) have small computing power and cannot interpret "bad" markup language.

The difference between HTML and XML is that HTML is designed to structure (and display) data and XML is designed to describe and structure data. XHTML specifies that everything must be marked up correctly

5.4 XHTML – BASE SYNTACTIC RULES

The basic syntactic rules to be followed when creating XHTML files are given as below,

- XHTML elements must be properly nested

```
<b><i> Italic and bold text </b></i>

<b><i> Italic and bold text </i></b>
```

- XHTML elements must always be closed. For every element there should be a opening tag and a closed tag. The <p> tag, <br> tag, <img> tag doesn't have a closing tag in HTML whereas in

XHTML there is a closing tag for every element. For example

```
<p> A paragraph…

<br>

<img src="foo.jpg">


<p> A paragraph…</p>

<br />

<img src="foo.jpg" />
```

- XHTML elements must be in lowercase

- XHTML elements must have one <html> root element (which contains a <head> and a <body>)

5.5 XHTML – OTHER SYNTACTIC RULES

The other syntactic rules to be followed is that,

- The attribute names must be in lower case
- The attribute values must be enclosed in double quotes

<table width=300px>

<table width="300px">

- The "id" attribute replaces the "name" attribute
- XHTML DTD defines mandatory elements
- Attribute minimization is forbidden in XHTML. The values has to be specified with attributes.

<input checked>

<input disabled>

<input checked="checked" />

<input disabled="disabled" />

General format of an XHTML document

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html>
<head>
        <title>…</title>
</head>
<body>
        …
</body>
</html>
```

The <!Doctype>,<html>,<head>,<title>,<body> elements are mandatory in an XHTML file.

5.6 DTD – DOCUMENT TYPE DEFINITION

A DTD specifies the syntax of a document written in a Standard Generalized Markup Language (SGML) such as HTML, XHTML and XML. It specifies the hierarchical structure of the document, element names and types, element content type and attribute names and values.

XML 1.0 defines three DTDs such as Strict, Transitional and Frameset DTD.

DOCTYPE must be specified on the first line when a DTD is defined.

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">

The DOCTYPE tells the web browser which language is being used for the set of instructions that follow. In this module, we will be using XHTML Transitional, which allows us more flexibility than XHTML Strict.

W3C has recommended the use of a Document Type Definition to identify the type of markup language used in a web page.

XHTML 1.0 Transitional .This is the least strict specification for XHTML 1.0. It allows the use of both Cascading Style Sheets and traditional formatting instructions such as fonts.

XHTML 1.0 Strict . This requires exclusive use of Cascading Style Sheets.

XHTML 1.0 Frameset. This standard is required for pages using XHTML frames.

DTD example (internal to XHTML file)

<!DOCTYPE course [

<!ELEMENT course (lecture+)>

<!ELEMENT lecture (title,bibliography,notes,examples)>

<!ELEMENT title (#PCDATA)>

<!ELEMENT bibliography (#PCDATA)>

<!ELEMENT notes (#PCDATA)>

<!ELEMENT examples (#PCDATA)>

<!ATTLIST course professor CDATA #REQUIRED>

<!ATTLIST course title CDATA #REQUIRED>

<!ATTLIST course yearofstudy CDATA #REQUIRED>

<!ATTLIST course date CDATA #IMPLIED>

]>

We will study more about DTD in the next module.

XHTML validation

A valid XHTML document is an XHTML document which obeys the rules of the DTD specified by the

<!Doctype> tag. The official W3C XHTML validator is,

http://validator.w3.org/check/referer

CHECK YOUR PROGRESS

True/False type questions

1. XHTML is HTML defined as an Web browser application.
2. XHTML integrates the concepts of XML with DHTML and hence it is relatively easy to introduce new elements or additional element attributes.
3. DTD specifies the hierarchical structure of the document, element names and types, element content type and attribute names and values.
4. The DOCTYPE tells the web browser which language is being used for the set of instructions that follow.
5. XHTML 1.0 Strict requires exclusive use of Cascading Style Sheets.

Answers-

1. False
2. False
3. True
4. True
5. True

5.7 XML

XML stands for eXtensible Markup Language. It is a markup language is used to provide information about a document. XML is a meta markup language for text documents / textual data. Tags are added to the document to provide the extra information.

The basic difference between HTML and XML is, HTML tags tell a browser how to display the document whereas XML tags give a reader some idea what some of the data means.

What is XML Used For?

XML documents are used to transfer data from one place to another often over the Internet. XML is text (Unicode) based. It takes up less space and can be transmitted efficiently.

One XML document can be displayed differently in different media such as HTML, video, CD or DVD. We only have to change the XML document in order to change all the rest. Moreover, XML documents can be modularized so that the parts can be reused.

An example of an XML Document is given here,

```
<?xml version="1.0"/>
<address>
        <name>ABCD</name>
        <email>abcd@annauniv.edu</email>
        <phone>044-2235-1234</phone>
        <birthday>1985-03-22</birthday>
</address>
```

All information in an XML file has markup for the data which aids in understanding its purpose. The XML language is very expressive that means semantics comes along with the data. It is well structured, easy to read and write from programs.

Difference Between HTML and XML

- HTML tags have a fixed meaning and browsers know what it is.
- XML tags are different for different applications, and users know what they mean.
- HTML tags are used for display.
- XML tags are used to describe documents and data.

XML Rules

Following are the rules to be followed when creating an XML file,

- Tags are enclosed in angle brackets.
- Tags come in pairs with start-tags and end-tags.
- Tags must be properly nested.

– <name><email>…</name></email> is not allowed.

– <name><email>…</email><name> is.

- Tags that do not have end-tags must be terminated by a '/'.

–
 is an example.

More XML Rules

- Tags are case sensitive.
  – <address> is not the same as <Address>
- XML in any combination of cases is not allowed as part of a tag.
- Tags may not contain '<' or '&'.
- Tags follow JAVA naming conventions, except that a single colon and other characters are allowed. They must begin with a letter and may not contain white space.
- Documents must have a single root tag that begins the document.

Encoding

XML (like Java) uses Unicode to encode characters. Unicode comes in many flavors. The most common one used in the West is UTF-8. UTF-8 is a variable length code where the characters are encoded in 1 byte, 2 bytes, or 4 bytes.

Well-Formed Documents

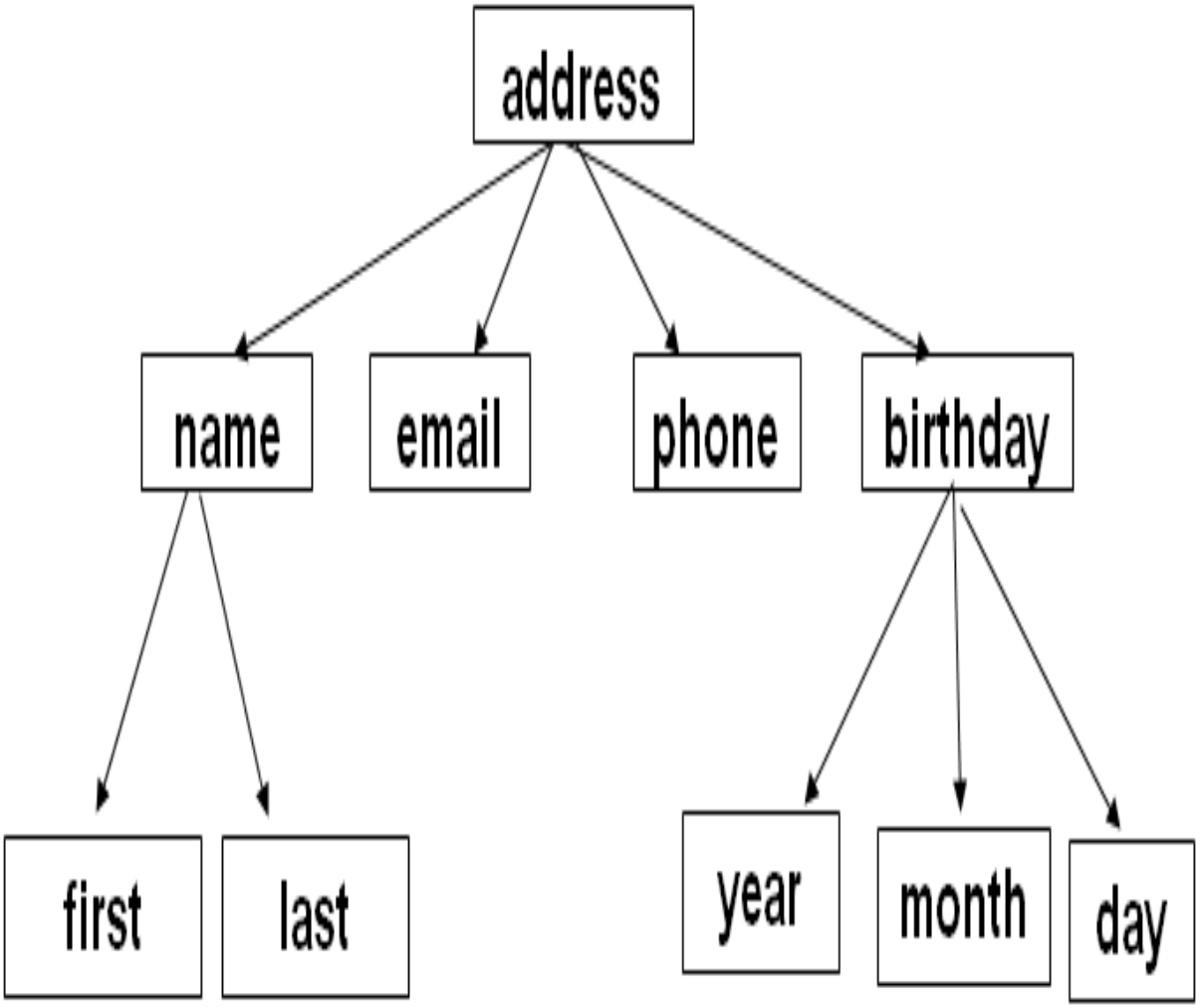
An XML document is said to be well-formed if it follows all the rules. An XML parser is used to check that all the rules have been obeyed. Recent browsers such as Internet Explorer 5 and Netscape 7 come with XML parsers. Parsers are also available for free download over the Internet. Java 1.4 also supports an open-source parser.

Example

This is an example of an XML file,

```
<?xml version = “1.0” ?>
<address>
        <name>
            <first>ABC</first>
            <last>XYZ</last>
        </name>
        <email>abc@annauniv.edu</email>
        <phone>044-2235-6789</phone>
        <birthday>
            <year>1976</year>
            <month>09</month>
        <day>26</day>
        </birthday>
</address>
```

In XML files, the data are represented with tags in such a way that they form a tree with a root. The above XML file can be viewed as a tree which is shown below.



The root element of the XML file is <address>. The <address> element has four children <name>,

<email>, <phone> and <birthday>. The <name> element has two children as <first> and <last>. The <birthday> element has three children as <year>, <month> and <day>. Hence any XML file can be represented as a tree.

5.8 XML TREES

An XML document has a single root node. The tree is a general ordered tree. A parent node may have any number of children. Child nodes are ordered, and may have siblings. Preorder traversals are usually used for getting information out of the tree.

XML Documents

An XML document consists of Elements, Attributes plus some other details such as textual information, namespaces, processing instruction and soon.

A Simple XML Document

```
<article>
  <author>ABC</author>
  <title>The Web in Ten Years</title>
  <text>
        <abstract>In order to evolve...</abstract>
        <section number="1" title="Introduction">
              The <index>Web</index> provides the universal...
        </section>
  </text>
</article>
```

Elements in XML Documents

In XML all tags are user definable/ freely definable. In the above code we have defined tags: article, title, author. All tags start with a start tag: <article> etc. and end with a end tag: </article> etc.

Elements: <article> ... </article>

Elements can have a name (article) and a content (...). Elements may be nested.

Elements may be empty:

<this_is_empty/>

Element content is typically a parsed character data (PCDATA), i.e., strings with special characters, and/or nested elements (mixed content if both).

Each XML document has exactly one root element and forms a tree. Elements with a common parent are ordered.

Elements vs. Attributes

Elements may have attributes (in the start tag) that have a name and a value, For example the element section has a attribute 'number' whose value is '1'.

<section number="1">.

The difference between elements and attributes are that only one attribute with a given name per element can be defined but an arbitrary number of subelements can be defined for an element.

Moreover, attributes have no structure, they are simply strings while elements can have subelements. An example is given below,

<person born="1912-06-23" died="1954-06-07"> Alan Turing</person> proved that…

5.9 NAMESPACES

Namespaces are generally provided to avoid element name conflicts. Name conflicts in XML can easily be avoided using a name prefix.

Namespaces in XML specification defines syntax for qualifying element or attribute names with a namespace identifier. Element/attribute names can be qualified with a namespace prefix as shown below,

(QName = prefix:local_name)

A namespace prefix is an abbreviation for a namespace identifier (URI). Namespace prefixes are mapped to namespace identifiers through namespace declarations.

(xmlns:prefix='namespace identifier')

Namespace declarations are placed within element start tags just like attributes.

Namespace Example

```
<d:student
  xmlns:d = 'http://www.develop.com/student' xmlns:i='urn:schemas/develop.
  com:identifiers'  xmlns:p = 'urn:schemas/develop.com:programming/languages'>
<i:id>3235329</i:id>
<name>XXX</name>
<p:language>C#</p:language>
<d:rating>9.5</d:rating>
```

```
</d:student>
```

When a namespace is defined for an element, all child elements with the same prefix are associated with the same namespace.

Default Namespace

Default namespace may be set for an element and its content but not for its attributes. For example,

```
<book xmlns = “http://www-dbs/dbs“>
  <description>...</description>
<book>
```

The default namespace can be overridden in the elements by specifying the namespace there using prefix or default namespace.

```
<d:student  xmlns:d='http://www.develop.com/student' xmlns='urn:foo' id='3235329'>
  <name>XYZ</name>
  <language xmlns="">C#</language>
  <rating>35</rating>
</d:student>
```

5.10 DEFINING XML DATA FORMATS

A well-formed document has a tree structure and obeys all the XML rules. A particular application may add more rules in either a DTD (document type definition) or in an XML schema. Well-formed document is defined as the document that adheres to the XML syntax rules.

Valid document is defined as the document that adheres to the rules defined in the corresponding DTD document. Only the valid documents are valuable in terms of sharing and retrieving information. Hence every XML file adheres to Document Type Definitions or XML Schema.

5.11 SUMMARY

This module provides an insight into XHTML. The module provides an introduction to XML, XML elements, attributes and namespaces. The module also provides an introduction to Document Type Definitions (DTD) which would be discussed in detail in the next module.

5.15 CHECK YOUR PROGRESS

1. ..... is used to define and organize the page content but not to format or style it.
2. XHTML uses the elements and attributes of.....
3. A ..... specifies the syntax of a document written in a Standard Generalized Markup Language (SGML) such as HTML, XHTML and XML.
4. XML 1.0 defines three ..... such as Strict, Transitional and Frameset DTD.
5. ..... must be specified on the first line when a DTD is defined.
6. .....stands for eXtensible Markup Language.

5.16 ANSWER CHECK YOUR PROGRESS

1. XHTML
2. HTML
3. DTD
4. DTDs
5. DOCTYPE
6. XML

5.17 MODEL QUESTION

1. What is XHTML? How XHTML combines the strength of HTML and XML?
2. How XHTML provides the web page with a more consistent and well-structured format?
3. What is the difference between HTML and XML? How HTML is designed to structure data and XML?
4. W3C has recommended the use of a Document Type Definition to identify the type of markup language used in a web page. Justify this statement.
5. How XHTML 1.0 Transitional allows the use of both Cascading Style Sheets and traditional formatting instructions such as fonts?
6. What is eXtensible Markup Language? Proof that XML is a meta markup language for text documents / textual data.

7. What is the basic difference between HTML and XML? HTML tags tell a browser how to display the document whereas XML tags give a reader some idea what some of the data means. Explain how.

5.18 REFERENCES

- https://epgp.inflibnet.ac.in/Home/ViewSubject?catid=7

5.19 SUGGESTED READINGS

- Powell. Thomas A., HTML & CSS: The Complete Reference, 5E.
- lemay. Laura, rafe colburn, jennifer kyrnin, Mastering HTML, CSS & JavaScript Web, BPB Publication, 2016.
- vishvajeet. Sisodia, Basic of Web Design, HTML, CSS3, Centrum Press, 2014.

UNIT-VI JAVASCRIPT AND HTML DOM

6.1 Learning Objectives

6.2 Document Object Model or DOM

6.3 JavaScript - Document Object Model or DOM

6.4 Types of DOM nodes

6.5 Traversing the DOM tree

6.6 Working with DOM

6.7 Summary

6.8 Check your progress

6.9 Answer Check your progress

6.10 Model Question

6.11 References

6.12 Suggested readings

6.1 LEARNING OBJECTIVES

The last module discusses about how to create Arrays, Functions and Objects. Moreover it also discusses about the use of built-in Objects in JavaScript.

In this module we try to understand about HTML DOM object hierarchy and to learn about how to access HTML elements using DOM. Moreover we will learn to work with DOM objects with examples.

6.2 DOCUMENT OBJECT MODEL OR DOM

The DOM is a W3C (World Wide Web Consortium) standard. It defines a standard for accessing documents using a platform and language-neutral interface and to dynamically access and update the content, structure and style of a document.

The W3C DOM standard is separated into 3 different parts:

1. Core DOM - standard model for all document types
2. XML DOM - standard model for XML documents
3. HTML DOM - standard model for HTML documents

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.

6.3 JAVASCRIPT - DOCUMENT OBJECT MODEL OR DOM

The way a document content is accessed and modified is called the Document Object Model, or DOM. DOM is an object-oriented model that describes how all elements in an HTML page are arranged. It is used to locate any object in your HTML page (an unique address). Every web page resides inside a browser window which is considered as Window object. A Document object represents the HTML document that is displayed in that window.The Document object has various properties that refer to other objects which allow access to and modification of document content.

DOM - Objects

Here is a simple hierarchy of a few important objects shown in Figure 18.1.

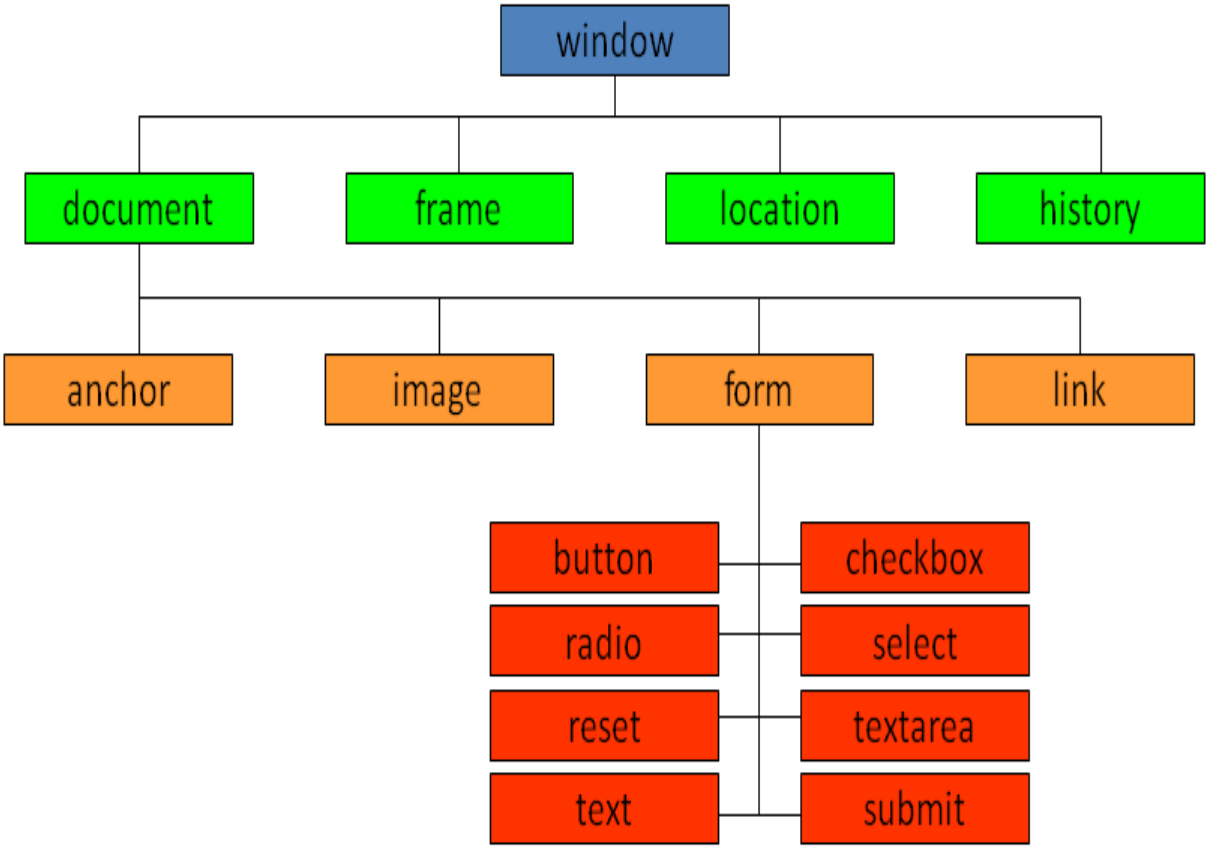


Figure 6.1 DOM - Objects

HTML DOM

The HTML DOM is a standard object model and programming interface for HTML. DOM defines the

HTML elements as objects, the properties of all HTML elements, the methods to access all HTML elements, the events for all HTML elements. The HTML DOM is a standard for how to get, change, add, or delete HTML elements.

JavaScript – HTML DOM Programming Interface

- The HTML DOM can be accessed with JavaScript and with other programming languages.
- In the DOM, all HTML elements are defined as objects.

The programming interface defines the properties and methods of each object.

- Methods are actions you can perform on HTML Elements.
- Properties are values of HTML Elements that you can set or change.

Referencing Objects

Objects can be referenced using their id or name. It should be unique in the hierarchy. It can be referenced using their numerical position in the hierarchy, by the array index. They can also be referenced using their relation to parent, child, or sibling like parentNode, previousSibling, nextSibling, firstChild, lastChild or the childNodes array.

6.4 TYPES OF DOM NODES

1. Element nodes - HTML tag
2. Text nodes - text in a block element
3. Attribute nodes - attribute/value pair

Element nodes can have children and/or attributes. Text or Attribute nodes are children in an element node. They cannot have children or attributes and not usually shown when drawing the DOM tree

CHECK YOUR PROGRESS

True/False type questions

1. The DOM is a W3C (World Wide Web Consortium) standard.
2. DOM defines a standard for accessing documents using a platform and language-neutral interface and to dynamically access and update the content, structure and style of a document.
3. The “window” Object is the lowest-level object in the JavaScript browser object hierarchy.
4. The form object is accessed as a property of the document object.
5. HTML forms can include eight types of storage elements.

Answers-

1. True

2. True
3. False
4. True
5. False

6.5 TRAVERSING THE DOM TREE

The nodes in the DOM tree can be traversed using the relationship like firstChild, lastChild, childNodes, nextSibling, previousSibling and parentNode. The description of these relationships are given in the below Table 18.1.

Table 6.1 Relationships among Nodes

name(s)	description
firstChild, lastChild	start/end of this node's list of children
childNodes	array of all this node's children
nextSibling, previousSibling	neighboring nodes with the same parent
parentNode	the element that contains this node

Accessing HTML Elements

All HTML elements (objects) are accessed through the document object. A document itself is automatically created. There are several ways to access a specific element, either using paths in the DOM tree or retrieval by tag or retrieval by ID.

Methods in document and other DOM objects for accessing descendants

There are two methods in Document object and other DOM objects for accessing the elements. They are,

getElementsByTagName getElementsByName

Accessing Elements by Paths

Example

```
function  execute()
{
        var img = document.images[0];    img.src="lighton.gif"
        var inx = document.forms[0].elements[0];  inx.value="xx"
        var iny = document.forms["form1"].elements["y"];   iny.value="yy"
}
<p><img src="lightoff.gif" alt="light off" id="img1" /></p>
<form id="form1" method="get" action="nosuch"><p>
  <input type="text" name="x"/>
  <input type="text" name="y"/>
  <input type="reset"/></p>
</form>
```

Accessing Elements by Tags

Example

```
function  execute()
{
  var spans = document.getElementsByTagName("span");  spans[0].style.color="red";
spans[1].style.color="blue";  spans[1].style.fontVariant="small-caps";
}
<p> This <span>example</span> is lovely. </p>
<p> But <span> this one </span>is even more! </p>
```

Accessing Elements by ID

Example

```
function execute()
{
  var theDiv = document.getElementById("div1");
  if (theDiv.style.visibility=="hidden" )
        { theDiv.style.visibility="visible" }
  else
        { theDiv.style.visibility="hidden"  }
}
<div id = "div1"> This text can be hidden! </div>
```

HTML DOM - Example

Example:

To change the content (i.e., the innerHTML) of the <p> element with id = "demo“.

```
<html>
<body>
<p id="demo"></p>
 <script>
        document.getElementById("demo").innerHTML = "Hello World!";
</script>
</body>
</html>
```

getElementById is a method and innerHTML is a property.

The "window" Object

It is the highest-level object in the JavaScript browser object hierarchy. It is the default object and is created automatically when a page is loaded. Since it is the default object, we may omit writing window explicitly.

Hence we use the method document.write("a test message") instead of

window.document.write("a test message");

It also includes several properties and methods for us to manipulate the webpage.

"window" Object - Property

Some of the properties of the window object is listed in Table 6.2.

Table 6.2 "window" Object - Property

Property	Description
length	An integer value representing the number of frames in the window
name	A string value containing the name of a window
parent	A string value containing the name of the parent window
Status	A string value representing status bar text

Property **Description**

length An integer value representing the number of frames in the window

name A string value containing the name of a window

parent A string value containing the name of the parent window

Status A string value representing status bar text

"window" Object - Method

Some of the methods of the window object is listed in Table 6.3.

Table 6.3 "window" Object - Method

Method	Description
alert(text)	Pop up a window with "text" as the message
close()	Closes the current window
open(url)	Open a new window populated by a URL.
setTimeout(expression, time)	Executes an expression after the elapse of the interval time.

Method Description

alert(text) Pop up a window with "text" as the message

close() Closes the current window

open(url) Open a new window populated by a URL.

setTimeout(expression, time) Executes an expression after the elapse of the interval time.

"window" Object - Attribute

Some of the attributes of the window object is listed in Table 6.4.

Table 6.4 “window” Object - Attribute

Attribute	Description
toolbar	Creates the standard toolbar
location	Creates the location entry field
directories	Creates standard directory buttons
status	Creates the status bar
menubar	Creates the menu bar at the top of a window
scrollbars	Creates scrollbars when the document exceeds the window size
resizable	Enables the user to resize the window
width	Specifies the width of the window
height	Specifies the height of the window

Attribute	**Description**
toolbar	Creates the standard toolbar
location	Creates the location entry field
directories	Creates standard directory buttons
status	Creates the status bar
menubar	Creates the menu bar at the top of a window
scrollbars	Creates scrollbars when the document exceeds the window size
resizable	Enables the user to resize the window
width	Specifies the width of the window

height Specifies the height of the window

Window object example

```
<html>
<head> <script> var w;
function openwindow()
{
        w = window.open("", 'width=100,height=100'); w.focus();
}
function myFunction()
{
        w.resizeBy(50, 50);
        w.focus();
}
</script>
</head>
<body>
        <button onclick="openwindow()">Create window</button>
        <button onclick="myFunction()">Resize window</button>
</body>
</html>
```

This example shows the use of window object. The example shows the open(), resize() and focus() method of the window object. There are two buttons, "Create Window" and "Resize window". On clicking these buttons, it invokes the open() method or resize() method. Once the window is resized, in order to gain focus on the window, focus() method is also used with the window object as shown in Figure 6.2.

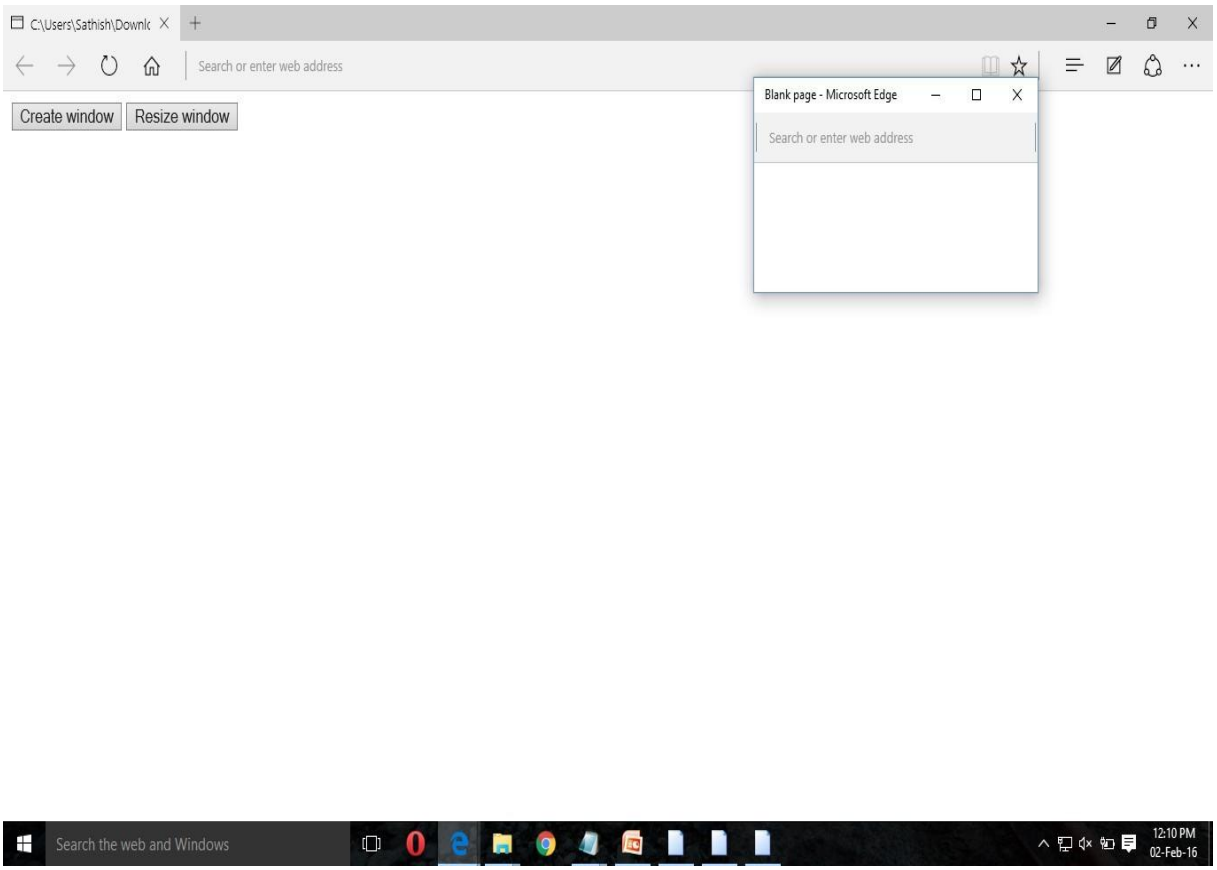


Figure 6.2 Output for Window Object Example

The "document" Object

It is one of the important objects in any window or frame. The document object represents a web document or a page in a browser window.

"document" Object - Property

Some of the properties of the document object is listed in Table 6.5.

Table 6.5 "document" Object - Property

Property	Description

bgColor	A string value representing the background color of a document
alinkColor	A string value representing the color for active links
location	A string value representing the current URL
title	A string value representing the text specified by <title> tag

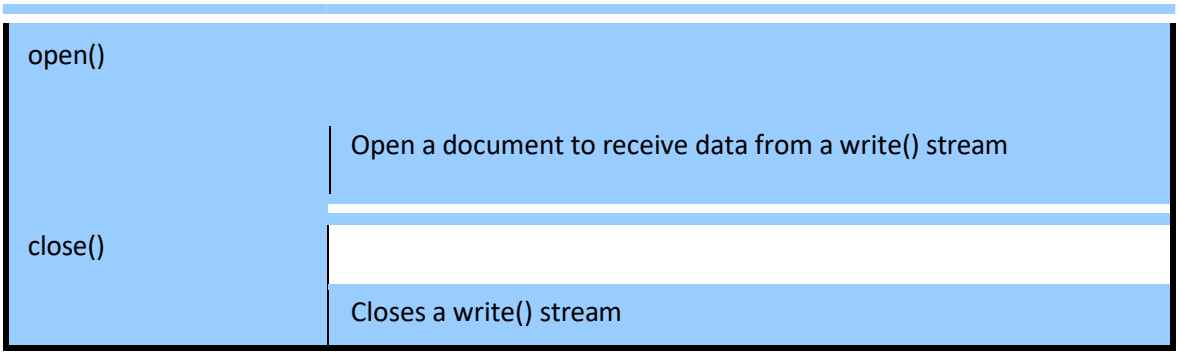
Property	Description
bgColor	A string value representing the background color of a document
alinkColor	A string value representing the color for active links
location	A string value representing the current URL
title	A string value representing the text specified by <title> tag

"document" Object - Method

Some of the methods of the document object is listed in Table 18.6.

Table 6.6 "document" Object - Method

Method	Description
clear()	Clears the document window
write(content)	Writes the text of content to a document
writeln()	Writes the text and followed by a carriage return



Method	Description
clear()	Clears the document window
write(content)	Writes the text of content to a document
writeln()	Writes the text and followed by a carriage return
open()	Open a document to receive data from a write() stream
close()	Closes a write() stream

Document Object Example

```
<html>
<body>
<script>
        document.write("Hello World!");
</script>
<p>Click the button to display the number of script elements in the document.</p>
<button onclick="myFunction()">Try it</button>
<p id="demo"></p>
<script>
function myFunction()
{
        var x = document.scripts.length;
        document.getElementById("demo").innerHTML = "Found " + x + " script elements.";
```

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

This example demonstrates how document object can be used. The write() method can be used with the document object to display output text "Hello World!". On clicking the button, the number of scripts found in the document can be displayed using the property "length" of the object as shown in Figure 6.3.

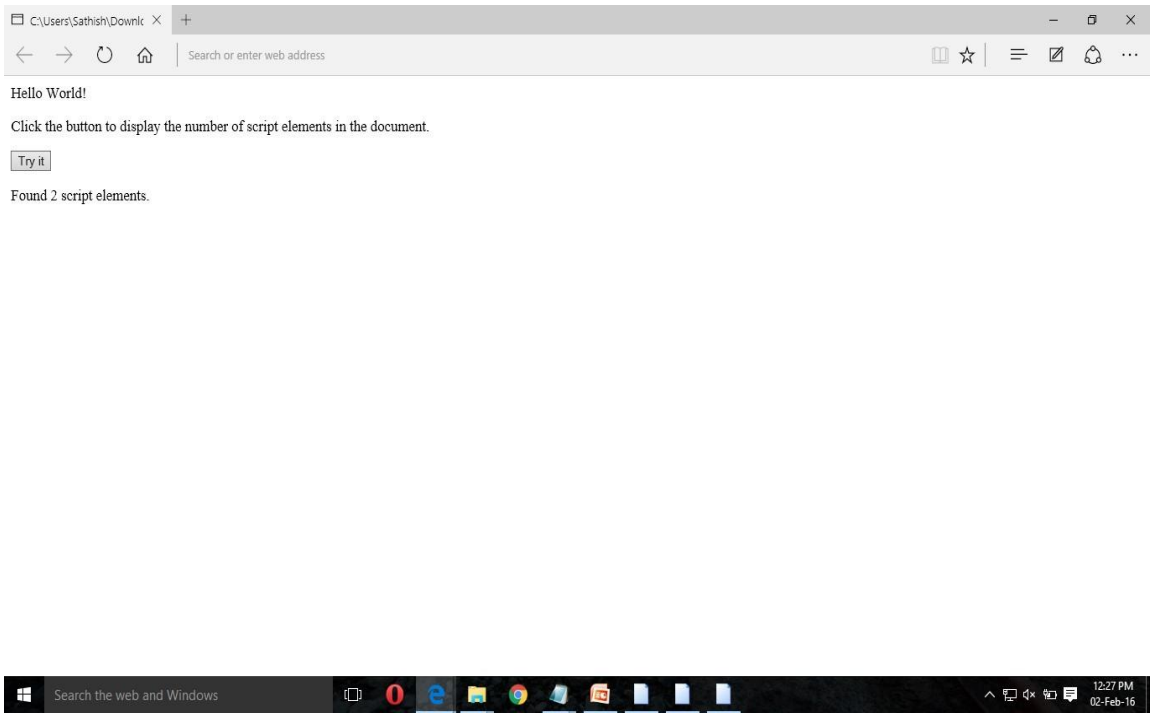


Figure 6.3 Output for Document Object Example

```
<html>
<body>
<p>Click the button to display the URL of the document.</p>
<button onclick="myFunction()">Try it</button>
<p id="demo"></p>
<script>
```

```
function myFunction()
{
   var x = document.URL;
   document.getElementById("demo").innerHTML = x;
}
</script></body>
</html>
```

Figure 6.4 displays the URL of the document. The example shows the use of the property "URL" used with the document object.

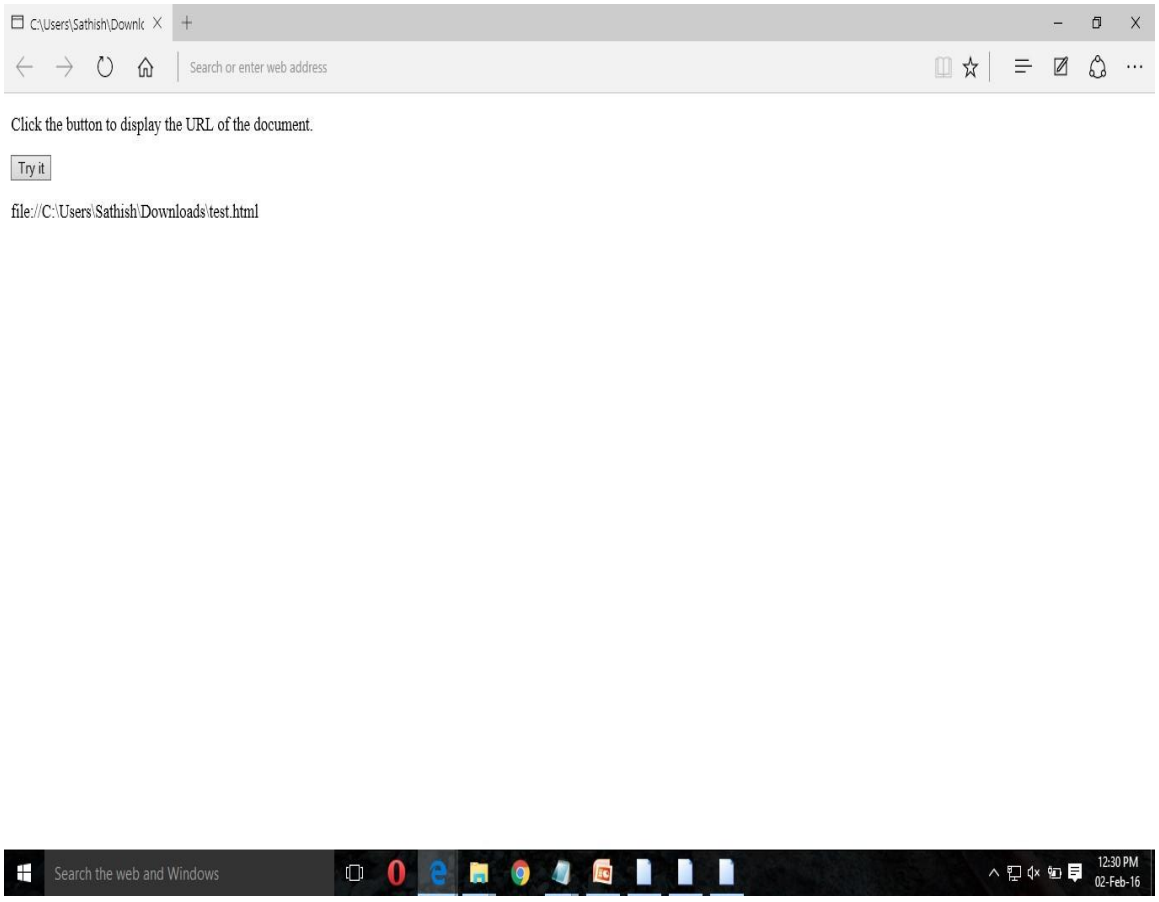


Figure 6.4 Output for Window Object Example

The "history" Object

Each time you visit a web page and click on the "Back" or "Forward" arrow buttons on your browser toolbar, you are accessing the history list. You can also add similar buttons / links that allow users to move backward and forward via the information stored in the history object.

"history" Object - Property

Some of the properties of the history object is listed in Table 6.7.

Table 6.7 "history" Object - Property

Property	Description
length	An integer value representing the number of links in the history object
current	Contains the URL of the current page
next	Contains the URL of the next entry in the history list
previous	Contains the URL of the previous entry in the history list

Property	Description
length	An integer value representing the number of links in the history object
current	Contains the URL of the current page
next	Contains the URL of the next entry in the history list
previous	Contains the URL of the previous entry in the history list

"history" Object - Method

Some of the methods of the history object is listed in Table 6.8.

Table 6.8 “history” Object - Method

Method	Description
back()	Sends the user to the previous page in the history list
forward()	Sends the user to the next page in the history list
go(x)	Sends back or forward by “x” number of pages in the history list

Method **Description**

back() Sends the user to the previous page in the history list

forward() Sends the user to the next page in the history list

go(x) Sends back or forward by “x” number of pages in the history list

History Object Example

```
<html>
<body>
<p>Display the number of URLs in the history list:</p>
<button onclick="myFunction()">Try it</button>
<p id="demo"></p>
<script>
function myFunction()
{
   document.getElementById("demo").innerHTML = history.length;
}
</script>
</body>
```

</html>

Figure 6.5 shows the property length of the history object. It displays the number of URL's in the history list.

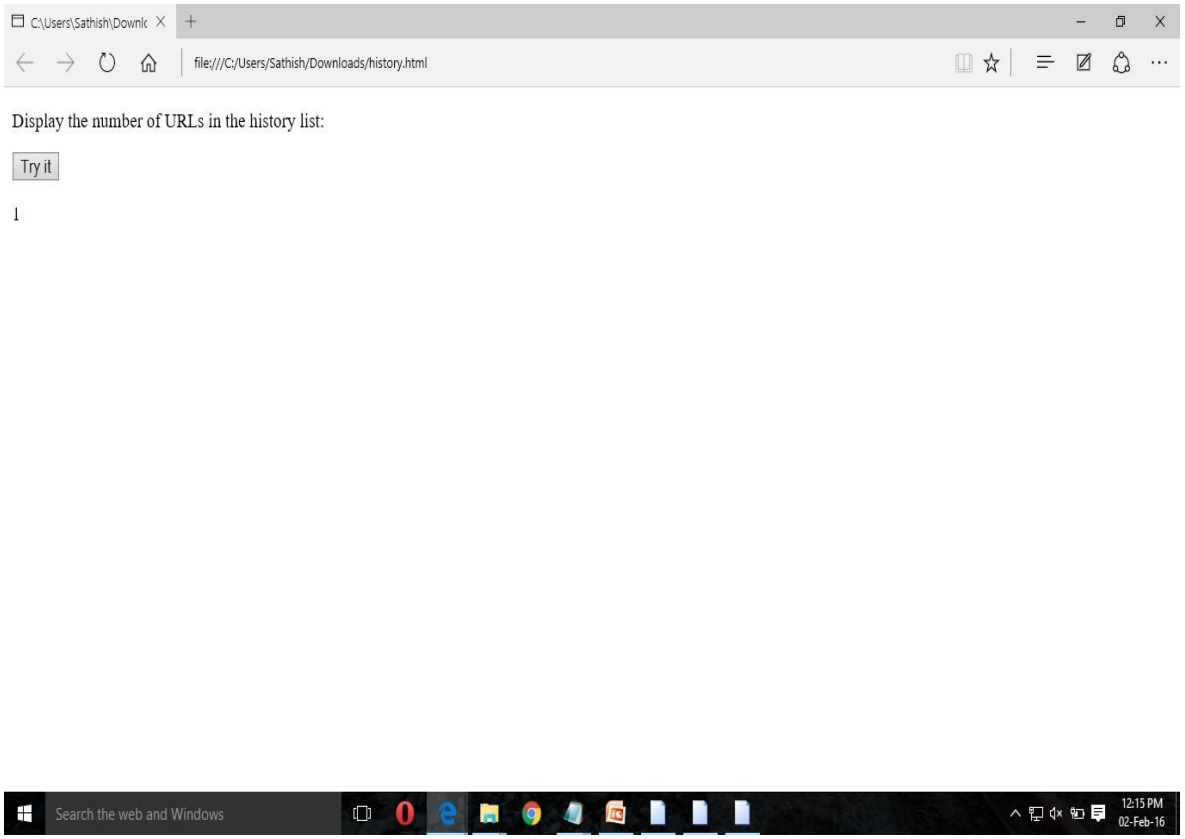


Figure 6.5 Output for History Object Example

History Object Example

<html>

<head>

<script>

function goBack()

{

window.history.go(-2)

}

</script>

```
</head>
<body>
<button onclick="goBack()">Go 2 pages back</button>
</body>
</html>
```

This example uses the method go() of the "history" object. Figure 6.6 displays the button "Go 2 pages back". On clicking the button, it calls the function goBack() which navigates two pages back and hence the output shown as the Google home page shown in Figure 6.7.

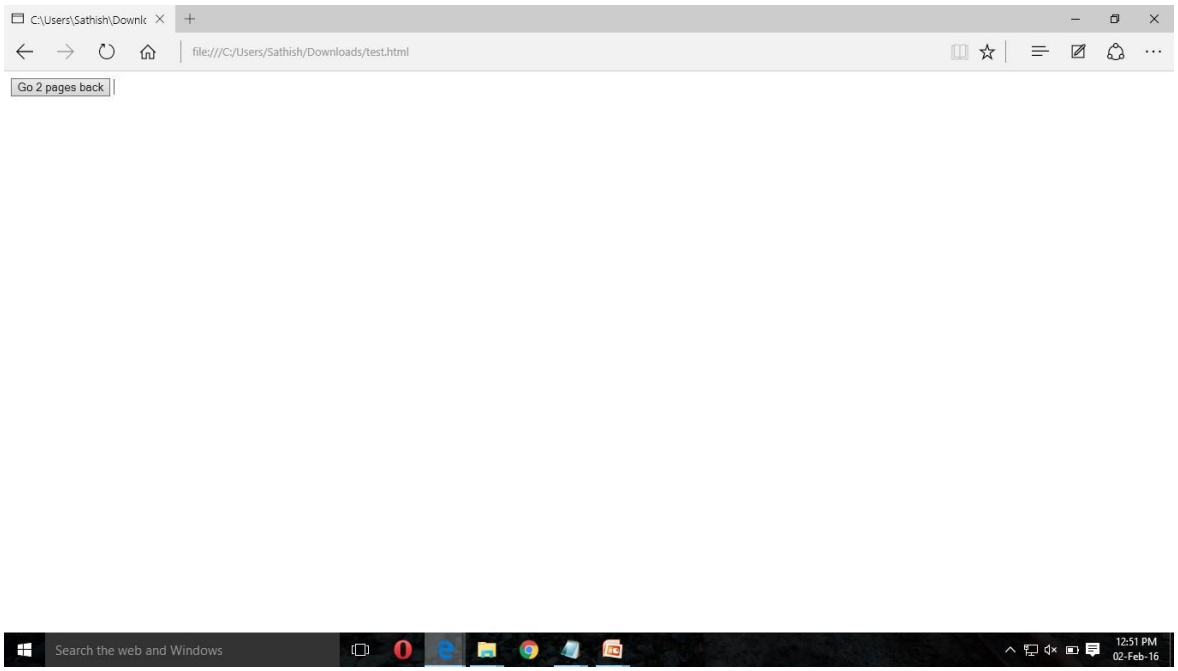


Figure 6.6 Output for History Object Example

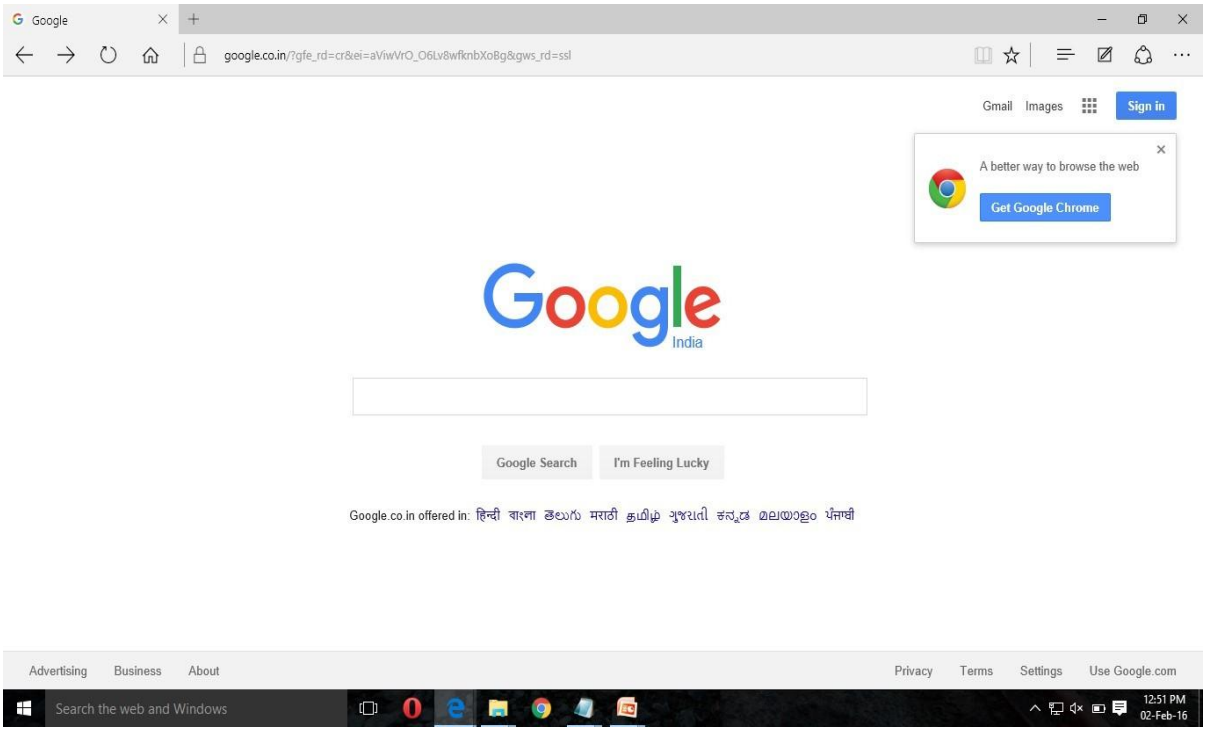


Figure 6.7 Output for History Object Example

Anchor Object - Example

```
<html>

<body>

<p> <a id="myAnchor" href="http://www.example.com:4097/test.htm#part2">Example link </a></p>

<p>Click the button to display the port number of the link above.</p>

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

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

<script>

function myFunction()

{

        var x = document.getElementById("myAnchor").port;

        document.getElementById("demo").innerHTML = x;

}
```

</script>

</ body>

</ html>

This example shows the property "port" used with the Anchor object. The property retrieves the port number of the anchor given as link and displays it as shown in Figure 6.8.

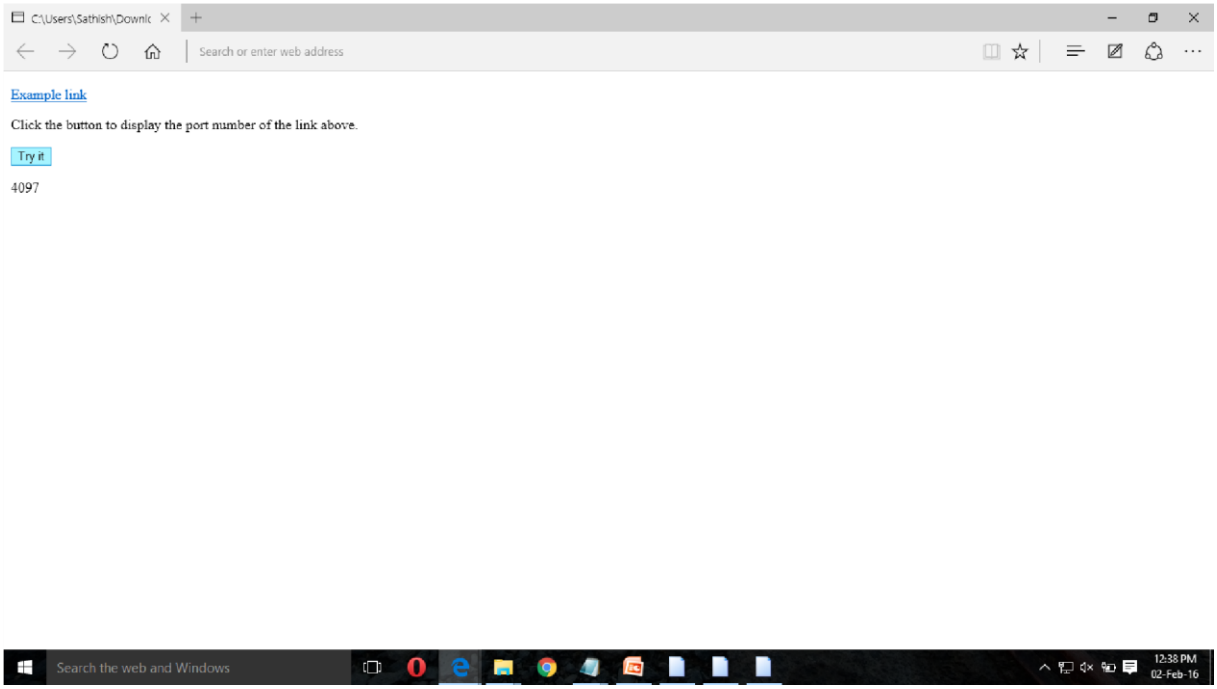


Figure 6.8 Output for Anchor Object Example

The "form" Object

The form object is accessed as a property of the document object. Each form element in a form (text input field, radio buttons), is further defined by other objects. The browser creates a unique "form" object for each form in a document. You can access the form object "form1" using the path document.form1.

Form Element-Based Objects

HTML forms can include eight types of input elements

a. Text fields, Textarea fields

b. Radio buttons

c. Check box buttons

d. Hidden fields

e. Password fields

f. Combo box select menu

g. List select menu

6.6 WORKING WITH DOM

Example:

```
<html>
<body>
        <img border="0" src="grapes.jpg" width="48" height="48"><br>
        <img border="0" src="flower.jpg" width="107" height="98"><br>
        <img border="0" src="fish.jpg" width="107" height="98"><br>
        <p>
        <script type="text/javascript">
        document.write("<hr><b>This document contains: "
                                + document.images.length + " images.</b>")
        </script>
        </p>
</body>
</html>
```

This example demonstrate the property length used with the image object. Figure 6.9 displays the number of images found in the document object.

Working with DOM - Output

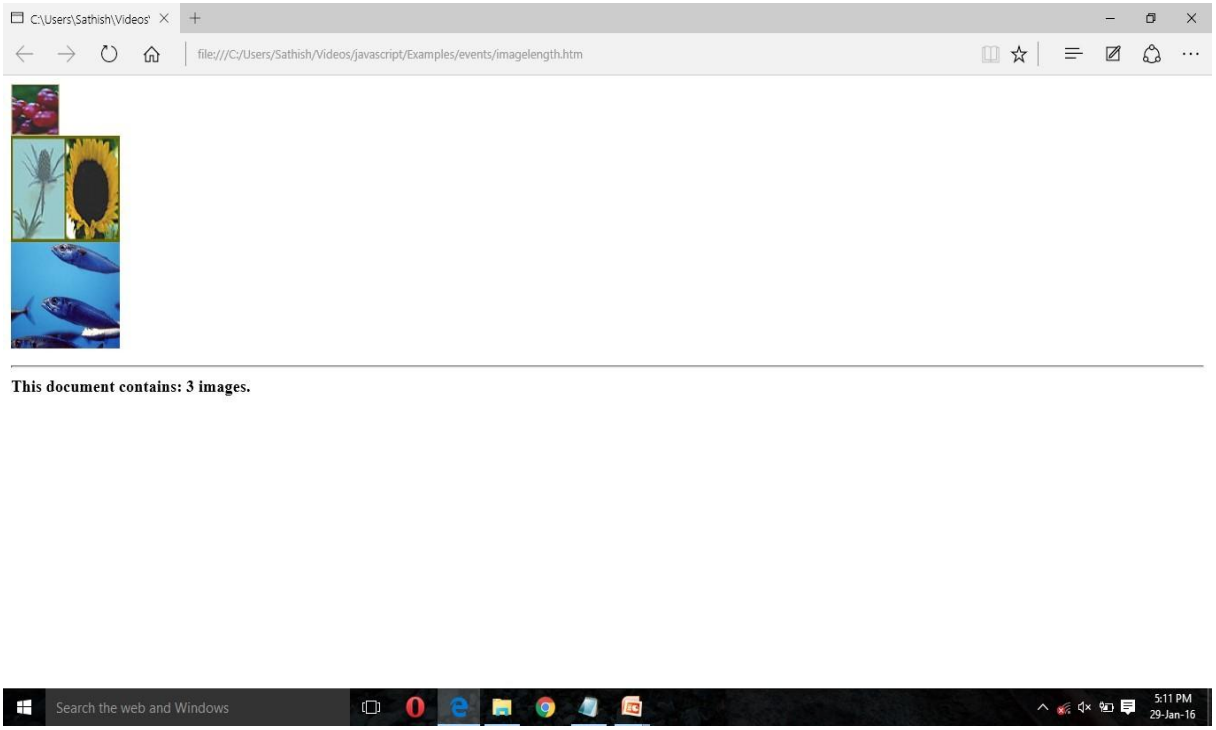


Figure 18.9 Output for Form Object Example

6.7 SUMMARY

This module is discussed about HTML DOM object hierarchy and explains about how to access HTML elements using DOM. This module also explores about working with DOM objects using JavaScript.

6.8 CHECK YOUR PROGRESS

1. The .....standard is separated into 3 different parts Core DOM, XML DOM, HTML DOM.
2. The .....allows access and modification of all document content and is standardized by the World Wide Web Consortium (W3C).
3. A Document object represents the..... document that is displayed in that window.

4. ....are values of HTML Elements that you can set or change.
5. ....should be unique in the hierarchy.

6.9 ANSWER CHECK YOUR PROGRESS

1. W3C DOM
2. W3C DOM
3. HTML
4. Properties
5. Referencing Objects

6.10 MODEL QUESTION

1. What is W3C (World Wide Web Consortium) DOM? What are the 3 different parts W3C DOM?
2. What is Java Script? How the HTML DOM can be accessed with JavaScript and with other programming languages?
3. What is DOM tree? How will you traverse the DOM tree? Also describe relationships among Nodes.
4. What are the Methods in document and DOM objects for accessing descendants?
5. There are two methods in Document object and other DOM objects for accessing the elements.

6.11 REFERENCES

- https://epgp.inflibnet.ac.in/Home/ViewSubject?catid=7

6.12 SUGGESTED READINGS

• Powell. Thomas A., HTML & CSS: The Complete Reference, 5E

• Powell. Thomas A., JavaScript: The Complete Reference

• lemay. Laura, rafe colburn, jennifer kyrnin, Mastering HTML, CSS & JavaScript Web, BPB Publication, 2016

UNIT-VII H JAVASCRIPT AND EVENT HANDLING

7.1 Learning Objectives

7.2 JavaScript - Event Handlers

7.2.1 onClick event type

7.2.2 onSubmit event type

7.2.3 onMouseOver and onMouseOut event

7.3 javascript cookies

7.4 create cookies

7.5 Store Cookies

7.6 Summary

7.7 Check your progress

7.8 Answer Check your progress

7.9 Model Question

7.10 References

7.11 Suggested readings

7.1 LEARNING OBJECTIVES

In the last module, we tried to understand about HTML DOM object hierarchy and learnt about how to access HTML elements using DOM. Moreover we have learnt to work with DOM objects with few examples.

In this module we will work with HTML forms and event handling. Moreover we will understand JavaScript Event Handling mechanisms and also we will learn about how to work with JavaScript Cookies.

7.2 JAVASCRIPT - EVENT HANDLERS

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. It allows us to execute an action of code when the event occurs. All the event handlers in JavaScript start with the keyword on, and each event handler deals with a certain type of event. (For eg, onClick)

How it works:

Step 1: User moves mouse over object

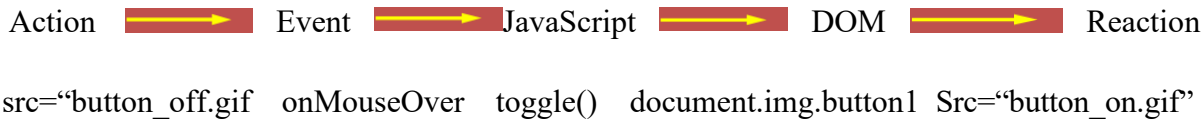
Step 2: Event senses that something happened to the object

Step 3: JavaScript (Event handler) tells the object what to do

Step 4: Using DOM, the handler locates the object on the web page

Step 5: Object's image source is changed

Here Step is an action where the user moves the mouse over an object. This action is sensed as an event, and JavaScript handles this event with an event handler. Using DOM, the object is located and the reaction to the event is performed as toggling the source image to anaother one. This is shown in the below diagram.



The following is the list of event handlers described along with the Objects on which they can be applied and when they are triggered as given in Table 7.1.

TABLE 7.1: Event Handlers

Event handler	**Applies to**	**Triggered when**

onAbort	Image	The loading of the image is cancelled.
onBlur	Button, Checkbox, FileUpload, Layer, Password, Radio, Reset, Select, Submit, Text, TextArea, Window	The object in question loses focus (e.g. by clicking outside it or pressing the TAB key).
onChange	FileUpload, Select, Text, TextArea	The data in the form element is changed by the user.
onClick	Button, Document, Checkbox, Link, Radio, Reset, Submit	The object is clicked on.
onDblClick	Document, Link	The object is double-clicked on.
onDragDrop	Window	An icon is dragged and dropped into the browser.
onError	Image, Window	A JavaScript error occurs.
onFocus	Button, Checkbox, FileUpload, Layer, Password, Radio, Reset, Select, Submit, Text, TextArea, Window	The object in question gains focus (e.g. by clicking on it or pressing the TAB key).
onKeyDown	Document, Image, Link, TextArea	The user presses a key.
onKeyPress	Document, Image, Link, TextArea	The user presses or holds down a key.
onKeyUp	Document, Image, Link, TextArea	The user releases a key.
onLoad	Image, Window	The whole page has finished loading.
onMouseDown	Button, Document, Link	The user presses a mouse button.
onMouseMove	None	The user moves the mouse.
onMouseOut	Image, Link	The user moves the mouse away from the object.
onMouseOver	Image, Link	The user moves the mouse over the object.
onMouseUp	Button, Document, Link	The user releases a mouse button.

onMove	Window	The user moves the browser window or frame.
onReset	Form	The user clicks the form's Reset button.
onResize	Window	The user resizes the browser window or frame.
onSelect	Text, Textarea	The user selects text within the field.
onSubmit	Form	The user clicks the form's Submit button.
onUnload	Window	The user leaves the page.

Now let us demonstrate some of the event type with examples and the output of it.

7.2.1 onClick EVENT TYPE

This event occurs when a user clicks the left button of his mouse.

Example 1

```
<html>
<head>
        < script type="text/javascript">   function sayHello()
        {
                alert("Hello World")
        }
</script>
</head>
<body>
        <p>Click the following button and see result</p>
        <form>
                <input type="button" onclick="sayHello()" value="Say Hello" />
        </form>
</body>
```

</html>

On clicking the button it calls the function sayHello() which alerts a message as shown in FIgure 7.1 and Figure 7.2.

Output:

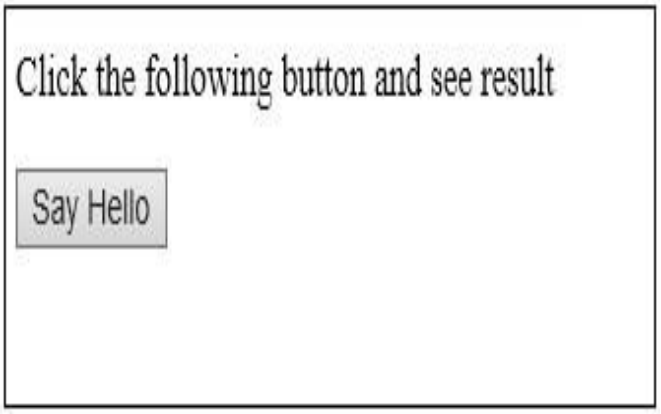


Figure 7.1 Output of onClick Event Type

Figure 7.2 Output of onClick Event Type

Example 2

<html>

<head>

<title> color world </title>

<script language = "JAVASCRIPT">

```
        function color(f, a)
        {
                document.bgColor=a;
                alert(a);
        }
</script>
</head>
<body bgcolor=gray>
        <form name=myform>
                <input type = button value=red  onClick=color(this.form,"RED")>
                <input type = button value=orange  onClick=color(this.form,"ORANGE")>
                <input type = button value=green  onClick=color(this.form,"GREEN")>
                <input type = button value=violet  onClick=color(this.form,"VIOLET")>
        </form>
</body>
</html>
```

This example is to demonstrate the onClick event type which changes the background color of the document on clicking the respective buttons as shown in Figure 7.3 and Figure 7.4.

Output

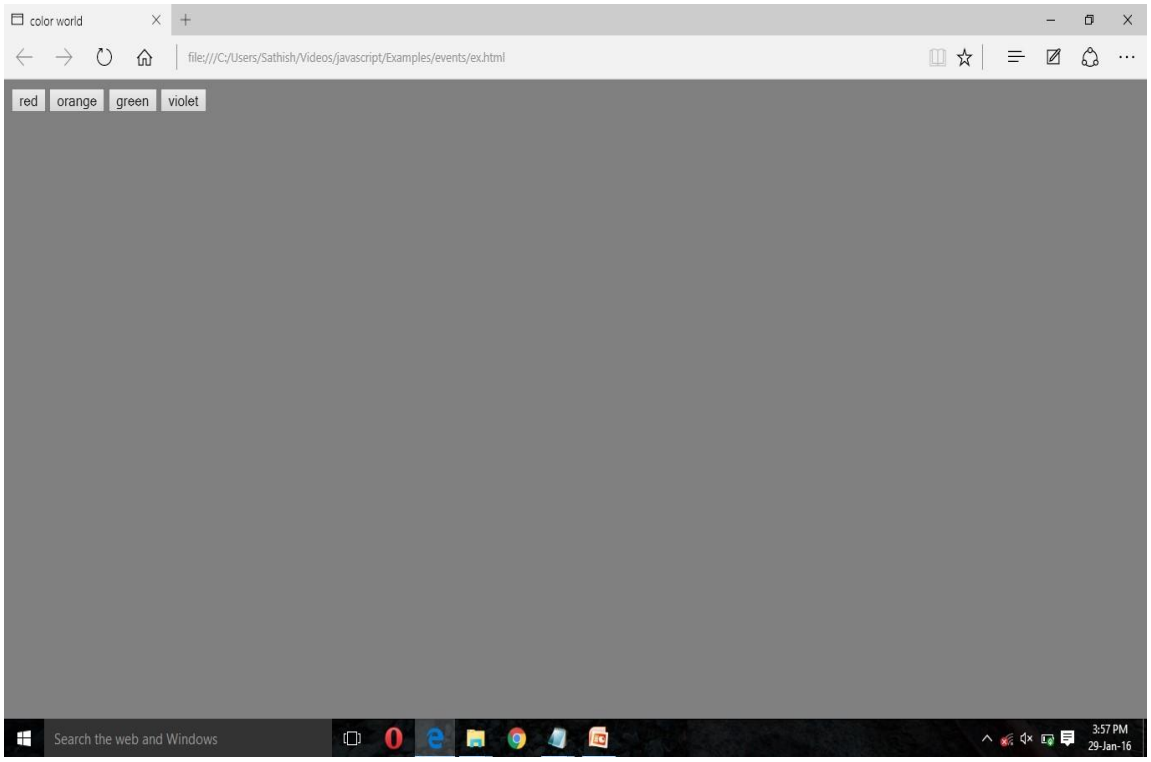


Figure 7.3 Output of onClick Event Type

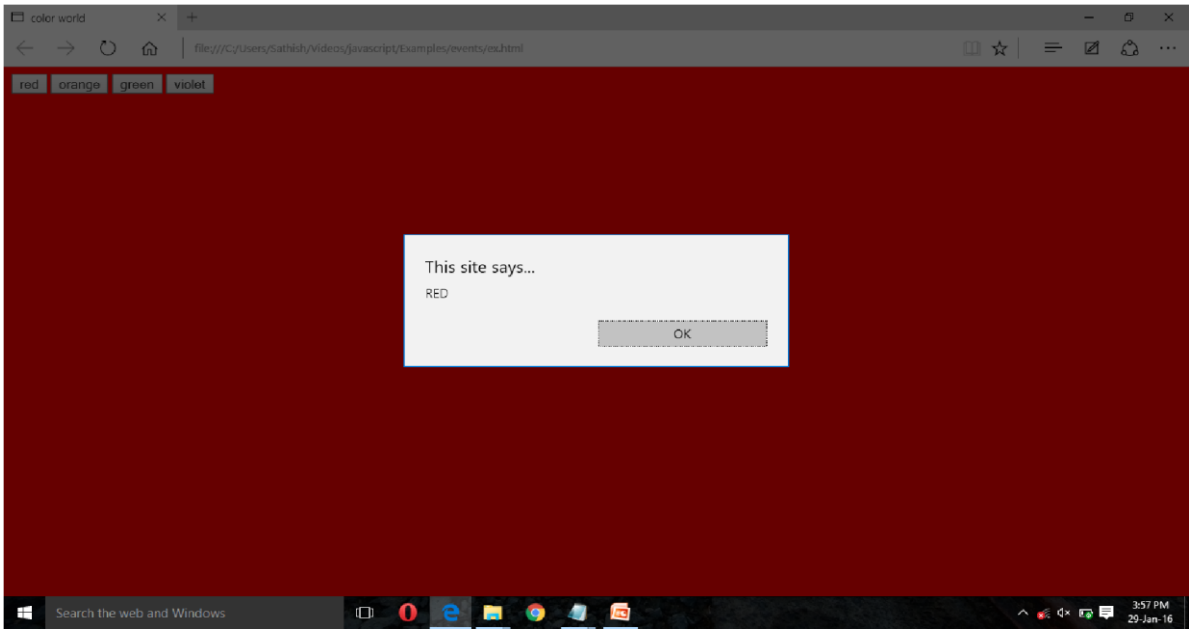


Figure 7.4 Output of onClick Event Type

7.2.2 onSubmit EVENT TYPE

The onSubmit is an event that occurs when you try to submit a form.

Syntax:

```
<html>
<head>
<script type="text/javascript">
        function validate()
        {
                all validation goes here
                .....
                return either true or false
        }
</script>
</head>
<body>
        <form method="POST" action="t.cgi" onsubmit="return validate()">
                <input type="submit" value="Submit" />
        </form>
</body> </html>
```

The example demonstates that the form is validated when a submit button is clicked as shown in Figure 7.5.

Output:

Figure 7.5

7.2.3 onMouseOver AND onMouseOut EVENT

The onMouseOver event is triggered when you bring your mouse over any element on the document and the onMouseOut event is triggered when you move your mouse out from that element.

Example 1:

```
<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>
```

This example demonstrates the onMouseOver and onMouseOut event types. When the mouse is moved over the text, the function over() is called which displays the message "Mouse Over" and when the mouse is moved out of the text , the function out() is called which displays the message "Mouse Out".

The output of this example code is shown in the Figure 7.6 and Figure 7.7.

Output:

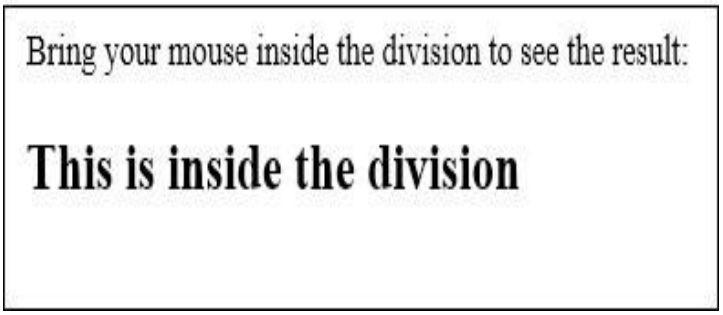


Figure 7.6

Figure 7.7

Example 2:

```
<html>
<head>
    <title> image world </title>
</head>
<body >
    <a href=ex.html onMouseOver = document.images[0].src = "FLOWER.jpg" onMouseOut = document.images[0].src = "GRAPES.jpg" >
    <img src="GRAPES.jpg" width=160 height=100 border=1>
    </a>
</body>
</html>
```

This example toggles the image on thedocument when the mouse is moved over or moved out of the image displayed. The output of this example code is shown in Figure 7.8 and Figure 7.9.

Output:

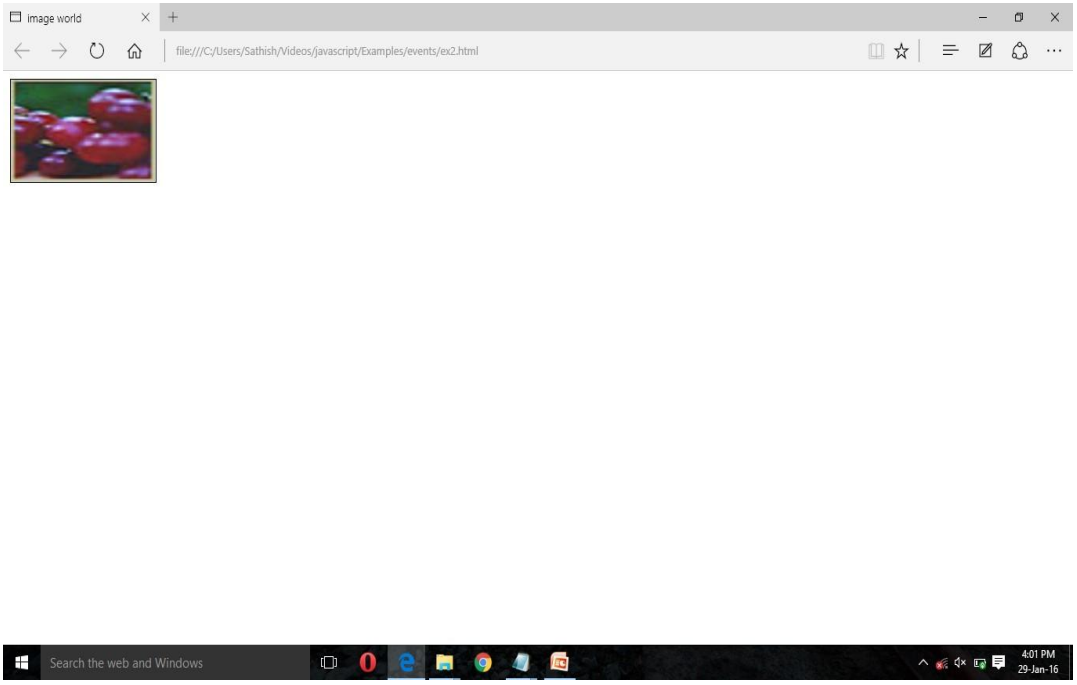


Figure 7.8

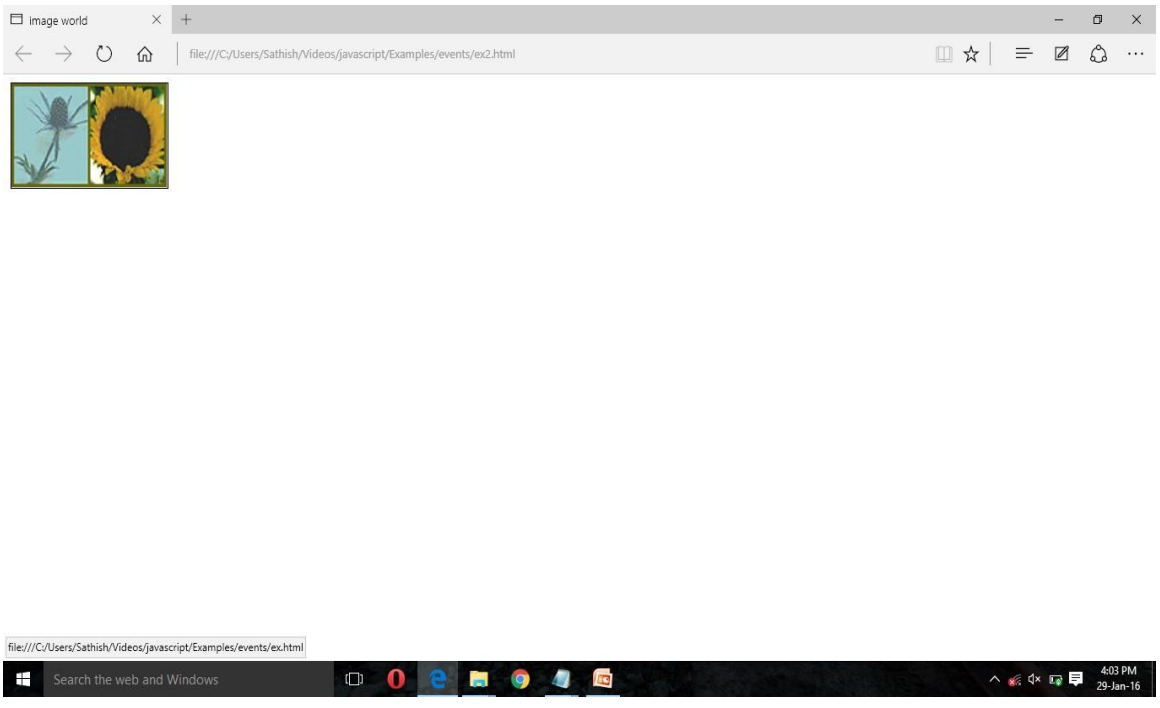


Figure 7.9

CHECK YOUR PROGRESS

True/False type questions

1. JavaScript's interaction with HTML is handled through events.
2. When the page loads, it is called an table.
3. Event allows us to execute an action of code when the event occurs.
4. All the event handlers in JavaScript start with the keyword click.
5. Each event handler deals with a certain type of event.
6. onClick event occurs when a user clicks the left button of his mouse.

Answers-

1. True
2. False
3. True
4. False
5. True
6. True

7.3 JAVA SCRIPT FORM VALIDATION

HTML form validation can be done by writing a script code using JavaScript.

Example:

An example to validate the name to be filled out without leaving it blank is shown below.

```
function validateForm()
{
    var x = document.forms["myForm"]["fname"].value;
    if (x == null || x == "")
    {
        alert("Name must be filled out");    return false;
```

```
        }
}
```

JavaScript Forms

```
<form name="myForm" action="demo_form.asp" onsubmit="return validateForm()" method="post">
Name: <input type="text" name="fname">
        <input type="submit" value="Submit">
</form>
```

Example 1:

```
<html>
<head>
<script type="text/javascript">
function newValue()
{
        var x=document.forms.myForm
        x[0].value="The mere act of aiming at something makes you big" x[0].select()
}
</script>
</head>
<body>
<body>
<form name="myForm">
        <textarea name="myTextarea" rows="10" cols="20">
        The roots of education are bitter but the fruit is sweeter
        </textarea><br />
        <input type="button" onclick="newValue()" value="New Value">
```

</form></body></html>

This example is to demonstrate about validating a form which has a textarea. The textarea is initially shown with the text. On clicking the button, the function newValue() is called which displays another text. The method select() is used to display the textarea as selected. The output is shown in Figure 7.10.

Output:

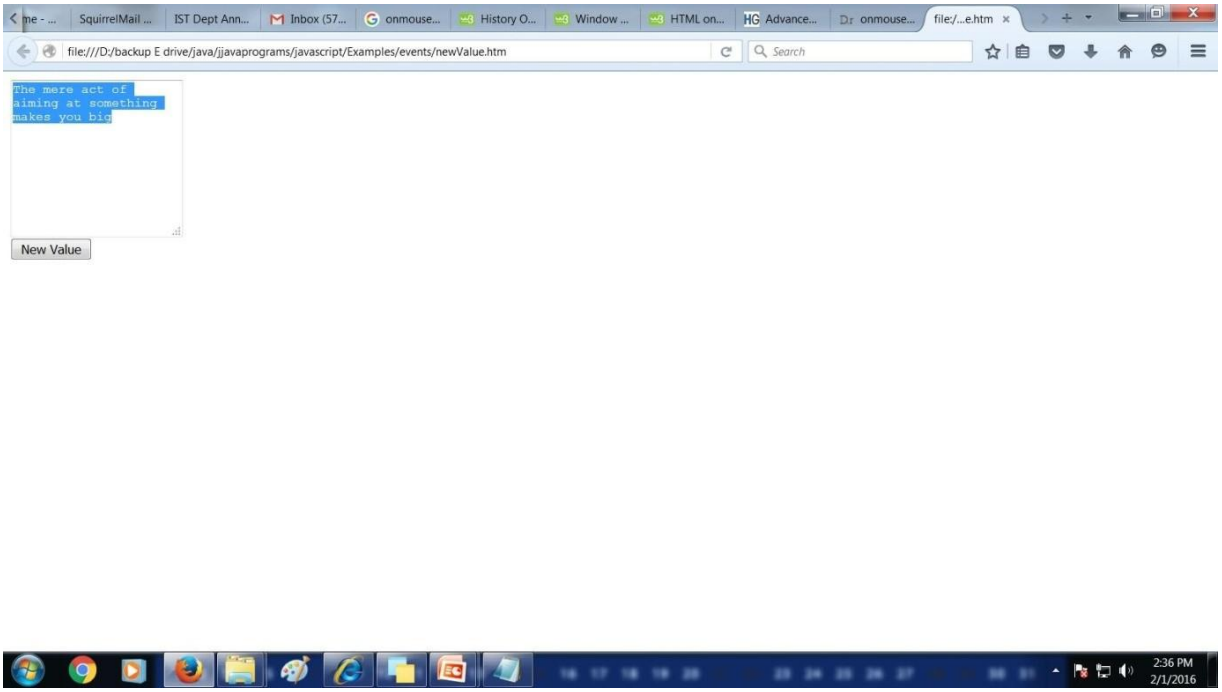


Figure 7.10

Example 2:

<html>

<head>

<title> Forms</title>

<!--This code checks the Checkbox When the button is clicked -->

<script Language='JavaScript'>

function Chk(f1)

{

```
        f1.Check.checked=true;
        window.alert("The Checkbox just got checked");
        f1.Check.checked=false;  f1.Radio[0].checked=true;  f1.Radio[1].checked=false;
        window.alert("The Radio button just got checked");
}
</script>
</head>
<body>
<form>
        Client Name :      <Input Type =Text Name="Text" Value=""><br><br>
        Client Address :  <Input Type =Text Name="Text1" Value=""><br><br>
        Client E-mail Address : <Input Type =Text Name="Text2" Value=""><br><br>
        <Input Type="radio" Name="Radio" Value=""> Male
        <Input Type="radio" Name="Radio" Value=""> Female<br><br>
        <Input Type=“checkbox" Name="Check" Value=""> Employed <br><br>
        <Input Type="Button" Name="Bt" Value="Set Element Array Value"
onClick="Chk(this.form)">
</form>
</body>
</html>
```

This example is to demonstrate about validating a form which has a checkbox or a radiobutton. If checkbox is checked or radiobutton is checked, the property "checked" is made true and it will display the message as checked. The output is shown in Figure 7.11.

Output:

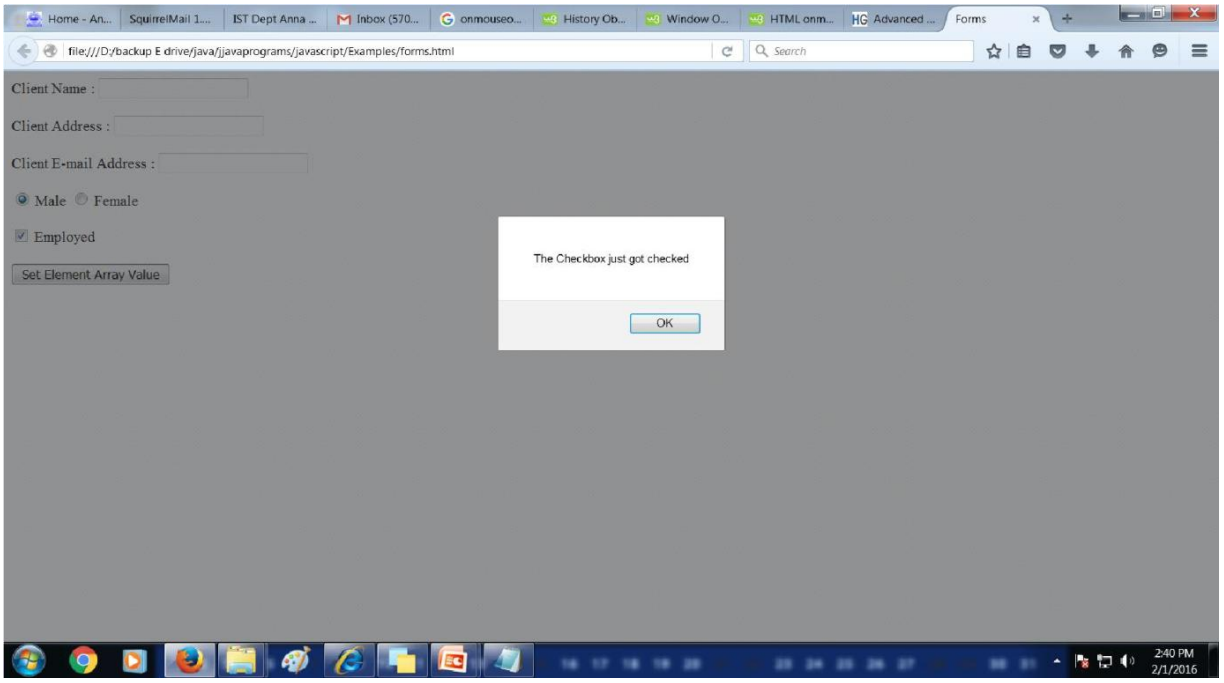


Figure 7.11

Example 3:

```
<html>
<head>
<script language="javascript">
</script>
</head>
<body>
<form>
        FirstName:<input type="text" name="Firstname" size=20 onFocus="this.blur();">
        LastName:<input type="text" name="Lastname" size=20><p>
        Address:<input type="text" name="Address" size=60><p>
        pincode:<input type="text" name="Pincode" size=6><p>
        <input type="button" name="act" value="verify">
 </form>
</body>
</html>
```

This example code demonstrates the use of function onFocus on a textfield. When a focus is gained on the textfield, it is made to blur using the function blur() which will not allow to enter any data in the textfield. The output is shown in Figure 7.12.

Output:

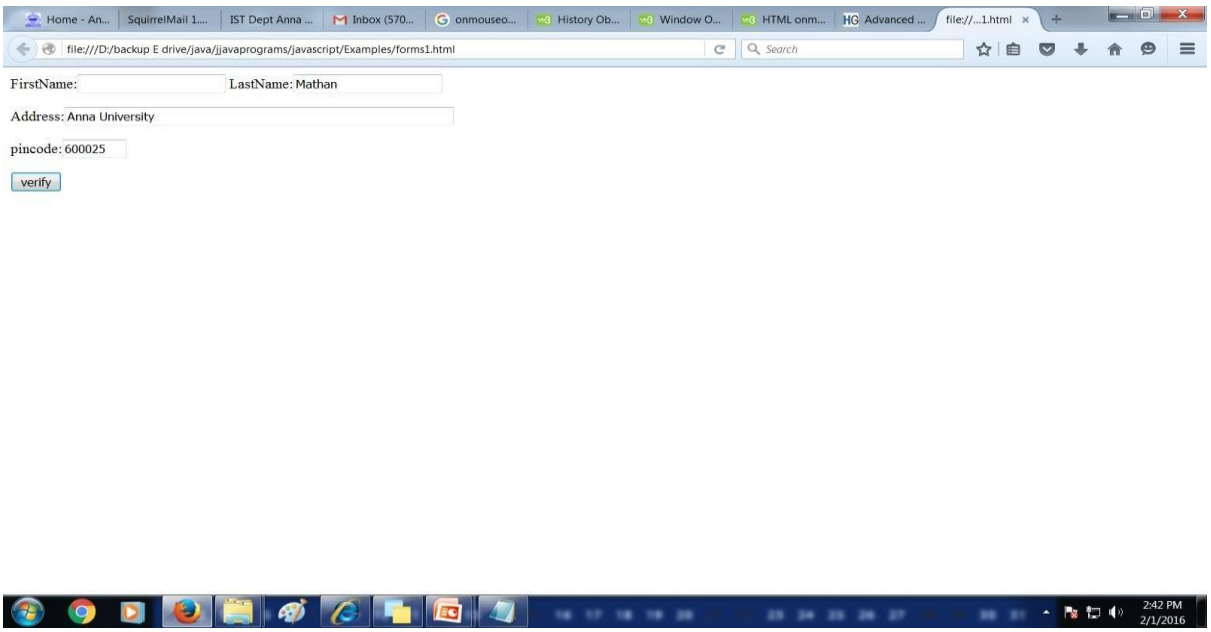


Figure 7.12

Example 4:

```
<html>
<head>
  <title>forms1</title>
<script>
 function func(f1)
  {
        alert("the form elements have been cleared");
  }
</script>
```

```
</head>
<body>
    <form onReset="func(this.form)">
        Client Name: <input type="text" name="text" value=""><br><br>
        Client address: <input type="text" name="text1" value=""><br><br>
        <input type="radio" name="radio" value="">male
        <input type="radio" name="radio" value="">female<br><br>
        <input type="checkbox" name="check" value="">employed<br><br>
        <input type="reset" name="rst" value="reset">
    </form>
</body>
</html>
```

This example code demonstrates validating a form on a reset button. The event onReset is triggered when a reset button is clicked which clears the form content and displays the message as cleared. The output is shown in Figure 7.13

Output:

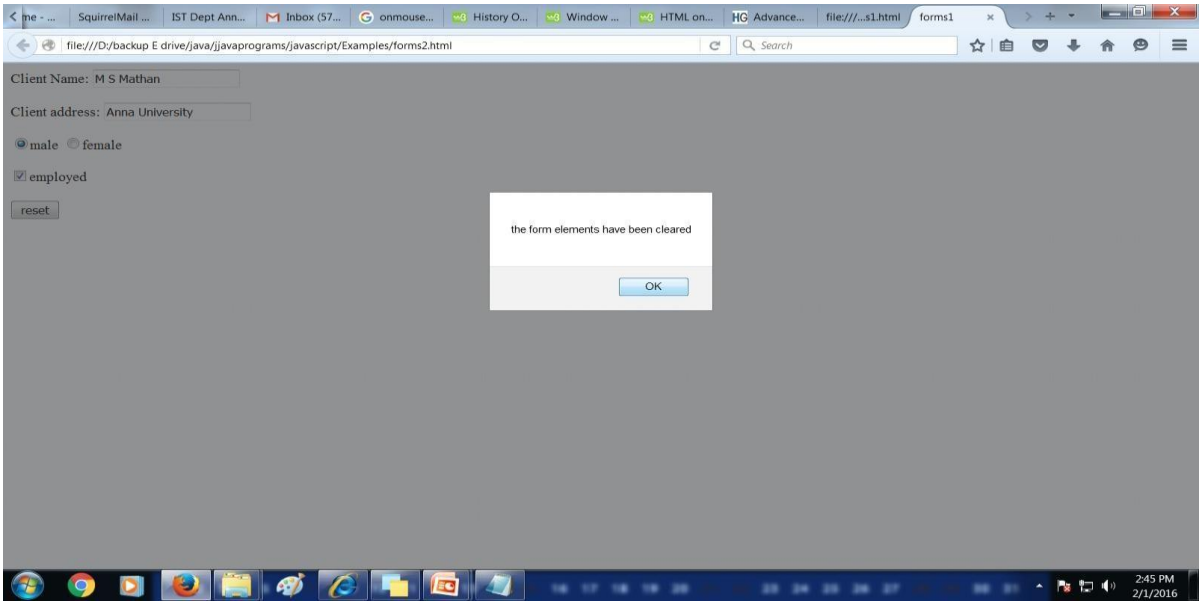


Figure 7.13

Example 5:

```
<html>
<head>
<title> using text and button objects</title>
<script language="javascript">
  function calculate(form)
    {
        form.results.value= eval(form.entry.value * 10 );
    }
 </script>
</head>
<body>
  <form>
        <input type="text" name="entry" value="">
        <input type="button" value="calculate" onClick="calculate(this.form);"
        <br>
        <input type="text" name="results" onFocus="this.blur();">
        <br>
  </form>
</body>
</html>
```

This example code demonstrates the use of onClick() and onFocus(). The onFocus is applied on results textfield which will not allow to enter any value using the blur() method. The value entered on the entry field is calculated and shown in the result field. The output of this code is shown in Figure 7.14.

Output:

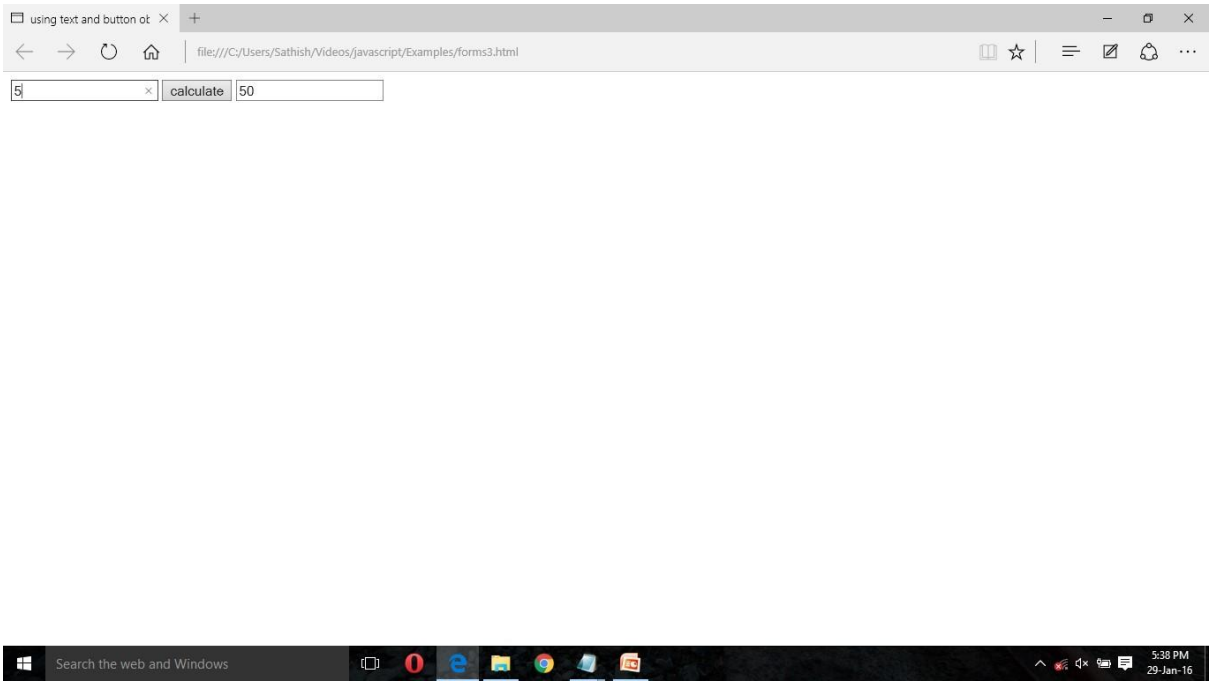


Figure 7.14

Example 6:

```
<html>
<head>
  <title>working with checkboxes</title>
  <script>
    function calculate(form, callingField)
    {
          if(callingField=="result")
          {
          if(form.square.checked)
          {
                form.entry.value= Math.sqrt(form.result.value);
          }
          else
          {
                form.entry.value=form.result.value/2;
```

```
        }
        }
        else
        {
                if(form.square.checked)
        {
        form.result.value = form.entry.value * form.entry.value;
        }
        else
        {
                form.result.value = form.entry.value*2;
        }
        }
        }
 </script>
 </head>
<body>
    <form>
    <center><br>
    <b> value: </b>
    <input type ="text" name = "entry" value = 0  onChange
="calculate(this.form,this.name);">
    <br><br>
    <b>action</b>(default-double):
    <input type ="checkbox" name = "square"  onClick =
"calculate(this.form,this.name);">square
    <br><br>
    <b> result: </b>
```

```
 <input type ="text" name = "result" value = 0  onChange =
"calculate(this.form,this.name);"> </center>

 </form>

</body>

</html>
```

This example code demonstrates the use of onClick and onChange event types. There are two textfields (entry and result) and a radiobutton in the form. If the calling field is result field, and if the checkbox is checked then it calculates the squareroot of the value entered in the result field and displays in the entry field else it calculates half of the result value and displays in the entry field. Similarly when the value entered in the entry field changes, based on the checkbox checked or not checked displays the square of the value entered in the entry field or multiplies the value by 2 and displays in the result field.

The output of this code is shown in Figure 7.15.

Output:

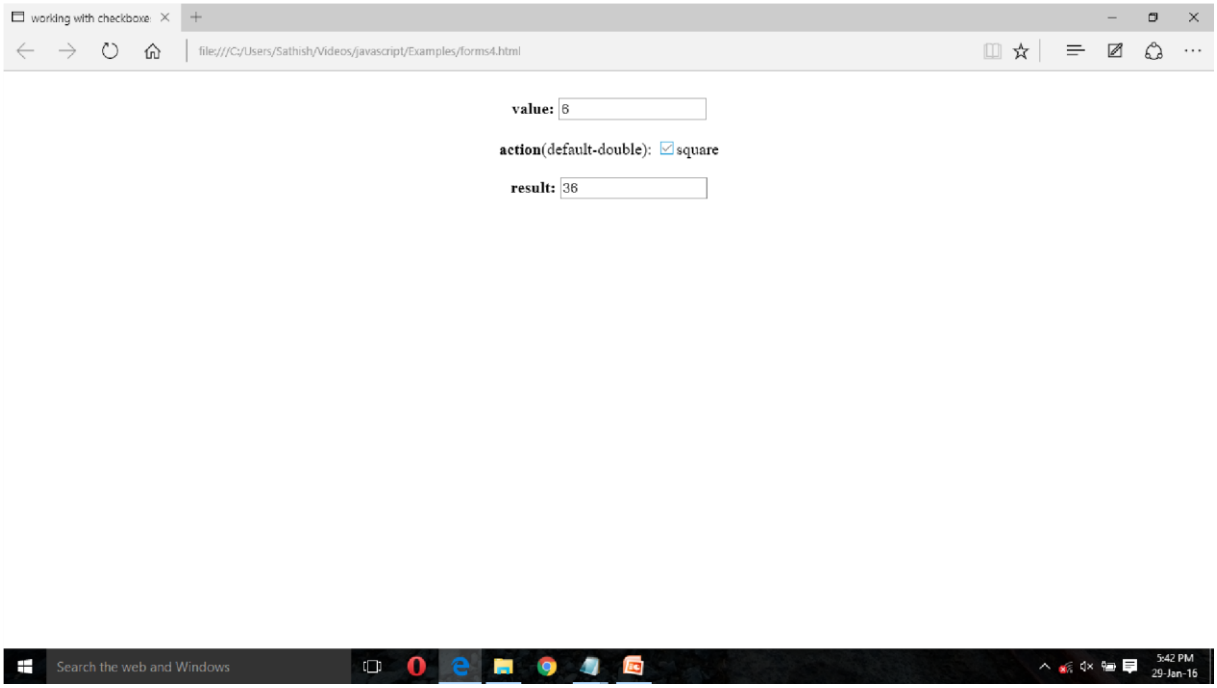


Figure 7.15

7.3 JAVASCRIPT COOKIES

A cookie is used to store information on the user's computer even when the user switches off his/her computer. It is a data that is sent from a web server to a web browser when the user

visits a site on a server. It is just a .txt file stored in a user's computer. A cookie can be associated with one or more documents on the web server. More than one cookie can be associated with a document on the web server. Every cookie has a NAME-VALUE pair associated with it. Cookies have an expiration date associated with them.

Cookies are a plain text data record of 5 variable-length fields:

1. **Expires** − The date the cookie will expire. If this is blank, the cookie will expire when the visitor quits the browser.

2. **Domain** - The domain name of your site.

3. **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.

4. **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.

5. **Name** - Value - Cookies are set and retrieved in the form of key-value pairs.

7.4 CREATE COOKIES

We can create a cookie by assigning a string value to the document.cookie object.

```
document.cookie = "key1=value1;key2=value2;expires=date";
```

7.5 STORE COOKIES

Example

```
<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>
```

The example shows how a cookie can be created and stored. It retrieves the value from the textfield and stores it as a cookie.

```
<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);
        }
    }
```

The function ReadCookie() retrieve the stored cookie from document.cookie and store in a variable.

Then we can use a for loop to read each cookie as a key-value pair.

7.5 SUMMARY

This module explains about the JavaScript Event handling mechanisms. This module also explores about working with HTML form validation using HTML DOM and Event handling and finally discusses about JavaScript Cookies.

7.7 CHECK YOUR PROGRESS

1. In .....event the whole page has finished loading.
2. ..... senses that something happened to the object.
3. In .....event the user presses or holds down a key.
4. In .....event the user moves the mouse over the object.
5. In .....event the user selects text within the field.
6. In .....event an event occurs when you try to submit a form.

7.8 ANSWER CHECK YOUR PROGRESS

1. onLoad
2. Event
3. onKeyPress

4. onMouseOver
5. onSelect
6. onSubmit

7.9 MODEL QUESTION

1. What is Event handler? Explain the use of onMouseOver and onMouseOut event? Explain with help of example.
2. What is onSubmit event type? How it works? Explain.
3. What is the difference between onMouseOver and onMouseOut event?
4. Is the onMouseOver event is triggered when you bring your mouse over any element on the document and the onMouseOut event is triggered when you move your mouse out from that element?
5. What is JavaScript Form Validation? How HTML form validation can be done by writing a script code using JavaScript?
6. A cookie is used to store information on the user's computer even when the user switches off his/her computer. Explain How?
7. How the Cookie sent data from a web server to a web browser when the user visits a site on a server.

7.10 REFERENCES

- https://epgp.inflibnet.ac.in/Home/ViewSubject?catid=7

7.11 SUGGESTED READINGS

- Powell. Thomas A., HTML & CSS: The Complete Reference, 5E.
- lemay. Laura, rafe colburn, jennifer kyrnin, Mastering HTML, CSS & JavaScript Web, BPB Publication, 2016.
- vishvajeet. Sisodia, Basic of Web Design, HTML, CSS3, Centrum Press, 2014.

UNIT-VIII JAVA SERVER PAGES(JSP)

8.1 Learning Objectives

8.2 Introduction to JSP

8.3 Comparison of Servlets with JSP

8.4 Advantages of JSP

8.5. JSP engines

8.6 JSP Architecture

8.7 Life cycle of a JSP Page

8.8 Directory structure of JSP

8.9 Anatomy of a JSP page

8.10 JSP Components and Tags

8.11 Creation of a small Web application using JSP

8.12 Summary

8.13 Check your progress

8.14 Answer Check your progress

8.15 Model Question

8.16 References

8.17 Suggested readings

8.1 LEARNING OBJECTIVES

The objectives of this chapteris to understand the advantages of JSP technology, to discuss about the architecture of JSP and how it works, to discuss about the different components and tags of JSP and finally to learn the creation of small web applications using JSP.

8.2 INTRODUCTION TO JSP

JSP stands for "Java Server Page" which is a high level abstraction of Java servlet technology. It is a server side technology which generates web pages dynamically in response to client requests. It provides a way to separate the generation of dynamic content (java) from its presentation (html) and hence in a regular html file, java code can be inserted for the dynamic parts using special tags. JSP tags can be used for a variety of purposes, such as retrieving information from a database or registering user preferences, accessing JavaBeans components, passing control between pages and sharing information between requests, pages etc.

Why use JSP?

JSP is easy to learn and it allows the developers to quickly produce web sites and applications in an open and standard way. It offers a robust platform for web development. The main reasons to use JSP includes Cross web server and Cross platform support,Java Language advantages (Object Oriented language) ,Component reusability by using JavaBeans and EJB.

8.3 COMPARISON OF SERVLETS WITH JSP

Servlets

It is a Server side Java programs which solves scalability issues because servlets run on the threads of execution and not on separate processes. It solves portability issues by running on every platform that supports Java and servlets are supported by all most popular web servers. The issues of servlets are: HTML tags are embedded in java programs within out.print () statements which reduces flexibility of programming. Moreover, application logic and formatting are closely tied and it is difficult to reuse code on a different logic.

JSP

It is also a server-side technology but it is the higher abstraction of servlets, which separates dynamic content from static contents of a web page where formatting and logic are not closely tied. Also,Java scriplets are embedded into html-like page to execute application logic and it separates the work of java programmers and page designers.

8.4 ADVANTAGES OF JSP

The advantages of JSP are:

- Easy to maintain and code.
- High Performance and Scalability.
- JSP is built on Java technology, so it is platform independent.

8.5 JSP ENGINES

To process JSP pages, JSP engine is required and it runs under the supervision of the servlet engine. It is typically connected with a web server or this can be integrated inside a web server or an application server. Some of the web servers that can be used for this are Apache Tomcat, Java Web Server, Web logic and Web Sphere.

8.6 JSP ARCHITECTURE

Now, the architecture of JSP and its working functionalities discussed here. The overall view of the JSP Architecture is shown in Figure 1.The web server needs a JSP engine ie. container to process JSP pages. The JSP container is responsible for intercepting requests for JSP pages. A JSP container works with the Web server to provide the runtime environment and other services a JSP needs. It knows how to understand the special elements that are part of JSPs.

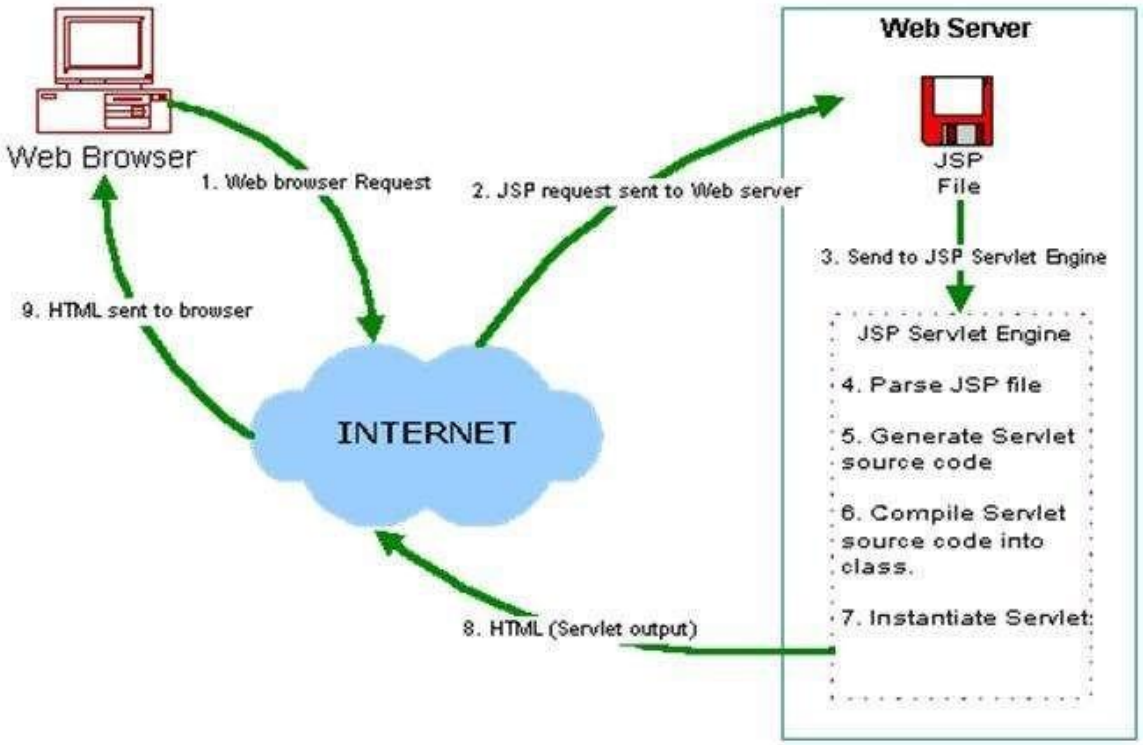


Figure 1: JSP Architecture.

How JSP works?

User gives JSP request page using web browser and the request is sent to the web server. From there, the JSP files are submitted to JSP Servlet Engine and based on the request, the output is sent back from web server to web browser and displayed as HTML pages. The working principle of JSP is given as the following pseudo code:

1. User requests a JSP page using a web browser through Internet.

2. The JSP request is transmitted to the Web Server (request for a .jsp file) using a URL.

3. Web server processes the requested .jsp file and sends the corresponding JSP file to the JSP Servlet Engine.

4. The Servlet engine checks whether servlet for the corresponding JSP file exists. If the servlet does not exists, it performs the following functions:

a. Translates JSP source code into servlet source code (Internally, all HTML code gets converted into suitable println statements).

b. Compiles the servlet source code to generate a class file (bytecode file).

c. Loads the class file and a corresponding instance has been created for that.

d. Initializes the servlet instance by calling the jspInit() method.

e. Invokes the _jspService() method, transmits both the request and response objects.

5. If the servlet already exists (instance is already running), simply forwards the request to the instance.

6. Now, the servlet output that has been generated is sent through the Internet from the web server to the web browser of the user and suitable HTML results are displayed on the user's browser.

CHECK YOUR PROGRESS

True/False type questions

1. JSP is easy to learn and it allows the developers to quickly produce web sites and applications in an open and standard way.
2. JSP offers a robust platform for web development.
3. HTML run on the threads of execution and not on separate processes.

4. JSP is built on Java technology, so it is platform independent.
5. A JSP container works with the Web server to provide the runtime environment and other services a JSP needs.

Answers-

1. True
2. True
3. False
4. True
5. True

8.7 LIFE CYCLE OF A JSP PAGE

A JSP page is converted into Servlet in order to service requests. The translation of a JSP page to a Servlet has also been involved in the Lifecycle of JSP. JSP Lifecycle is exactly same as the Servlet Lifecycle, with one additional first step, which is, translation of JSP code to Servlet code. Following are the JSP Lifecycle steps,namely jspInit(),jspService() and jspDestroy(). The pictorial representation of a JSP page is shown in the Figure 2.

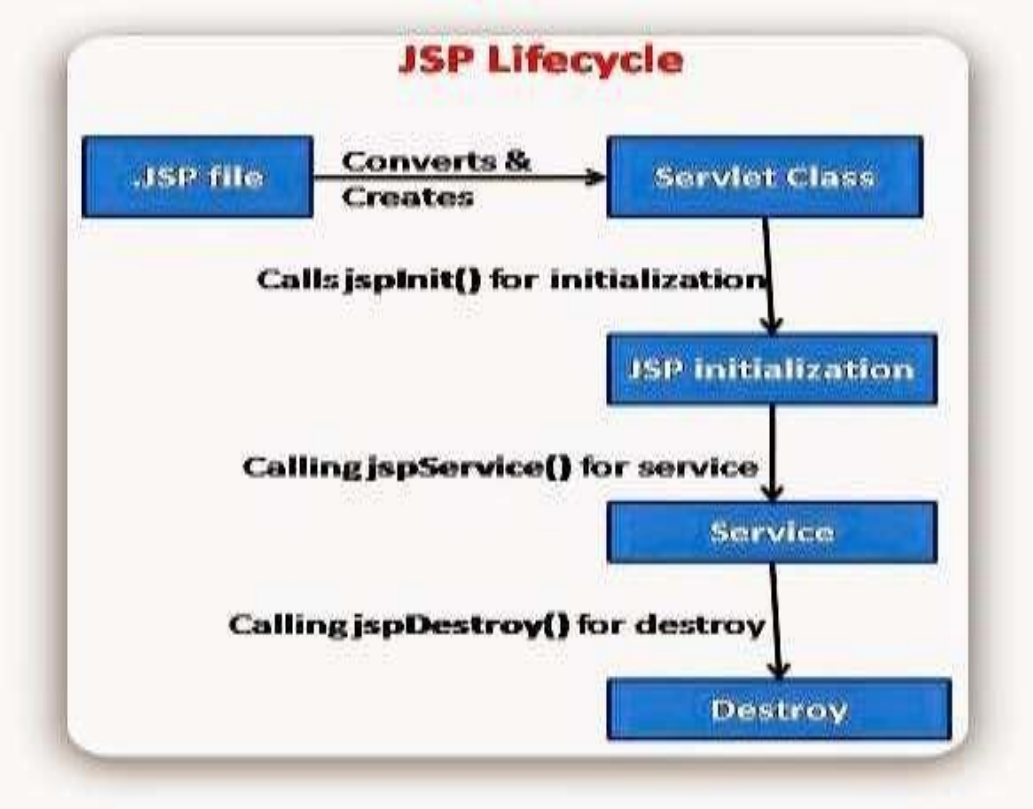


Figure 2:A JSP Lifecycle.

The detailed explanation of JSP Lifecycle is given below:

Translation: JSP gets translated into a Servlet for the first time. JSP container parses the JSP pages. It then translate the JSP pages to generate corresponding servlet source code. If JSP file name is hello.jsp, usually it is named as hello_jsp.java by the container.

Compilation: After translating into a servlet, it has been compiled and a class file gets created for that servlet. The compilation process involves three steps:

- Parsing the JSP.
- Turning the JSP into a servlet.
- Compiling the servlet.

Initialization: After successfully creating a class file, JSP engine invokesjspInit() to initialize the class file. Typically initialization is performed only once.

Service: After initialization, service (jspService()) method has been invoked and it handles the request and creates the response to be delivered to the browser. The _jspService() method of a JSP is invoked once per request and it is responsible for generating the response for that request.This method is also responsible for generating responses to the entire seven HTTP methods ie. GET, POST, DELETE etc.

Destroy: Once the service has been completed and response is sent back, jspDestroy() method has been invoked to clean up the resources.

If JSP page gets changed, JSP engine identifies automatically and translates it into a new servlet by passing through Translation, Compilation, Loading and Initialization phases.

A JSP Example

Let us see a simple JSP example as shown below:

```
<html>
<body>
    <% out.print(2*5); %>
</body>
</html>
```

To create ajsp page, some html code has to be written, and saved with .jsp as extension. Here in this example, this is saved as index.jsp. Later, this index.jsp file need to be kept in a folder the folder could be pasted in the web-apps directory of apache tomcat (Web server) in order to run the index.jsp page.

How to run a simple JSP Page?

Inorder to execute a JSP page ,the following steps need to be followed :

– Install the web server (For example: Apache Tomcat)

– Start the web server & put the .jsp file in a folder

(for example :myapplication as folder name)

– Deploy the folder on the web server(example:Tomcat with port no: 8080)

– visit the browser using the url specified below to view the web page
http://localhost:port no/Folder name/Filename.jsp

Example : http://localhost:8080/myapplication/index.jsp

8.8 DIRECTORY STRUCTURE OF JSP

Before continuing further with JSP, it is mandatory to know the directory structure of JSP.The directory structure of JSP page is same as the servlet. Here the .jsp page is either placed outside the WEB-INF folder or in any directory inside the Web-apps folder of Apache Tomcat as shown below in Figure 3.

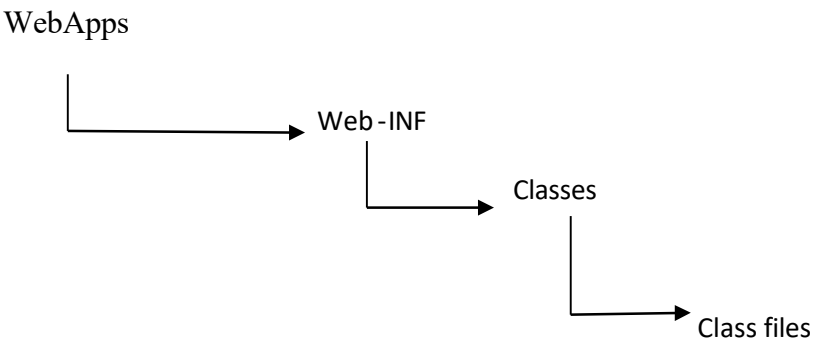


Figure 3. Directory Structure of JSP

8.9 ANATOMY OF A JSP PAGE

Next let us have the brief knowledge about the anatomy of a JSP page. It contains two parts namely HTML/XML markups and JSP constructs/ Elements. These JSP constructs/Elements are of 3 different types consisting of:

1. Scripting elements, where Java code is inserted using scripting elements.

a. It is of 3 types: Scriplets, Declarations and Expressions.

2. Directives that controls overall structure and behavior of the generated servlet

3. Actions that control the behavior of the JSP engine to use existing components.

a. Additionally a JSP page includes JSP comments also.

8.10 JSP COMPONENTS AND TAGS

Some of most commonly used JSP components and tags are listed below:

– Directives:<%@...%> Used for JSP directives such as page and include

– Declarations:<%!...%>Used for variables, methods and inner class declarations.

– Expressions:<%=...%>Used for JSP expressions

– Scriplets:<%...%>Used for JSP scriplets that can contain arbitrary Java statements.

– Action:<%@...%> Used for JSP directives such as page and include

JSP Directives

Each of the JSP elements are described in detail in this section. The JSP directives are messages that informs the web container that how a Java Server Page can be translated into a corresponding servlet.There are different three types of JSP directives:page directive, include directive and taglib directive

The general syntax for the JSP directives is

<%@ directive attribute="value" %>

A glimpse of JSP Directives along with their description is shown in the Table 1. Table 1: JSP directives and their descriptions

Directive	Description
<%@ page ... %>	Defines page-dependent attributes, such as scripting language, error page, and buffering requirements.
<%@ include ... %>	Includes a file during the translation phase.
<%@ taglib ... %>	Declares a tag library, containing custom actions, used in the page

Page directive

The pagedirective is used to provide instructions to the container regarding the current JSP page. Page Directives are allowed to code anywhere in our JSP page. By convention, page directives are coded at the top of the JSP page. Following is the basic syntax of page directive:

<%@ page attribute="value" %>

We can also write the XML equivalent syntax as follows:

<jsp:directive.page attribute="value" />

An example for page directive is shown below:

Example:

<%@ page import="package.class" %>

<%@ page import="java.util.*" %>

<%@ page contentType="text/html" %>

<% response.setContentType("text/html"); %>

JSP Include Directive

The include directive is used to include the contents of any resource. It may be a jsp file, html file or text file. The job of the include directive is to include the original content of the included resource during page translation time (the JSP page is translated only once so it is better to include a static resource).The main advantage of include directive is Code Reusability.The syntax of include directive is as follows:

<%@ include file="resourceName" %>

An example of include directive is

<html>

<body>

<%@ include file="header.html" %>

Today is: <%= java.util.Calendar.getInstance().getTime() %>

</body>

</html>

JSP Taglib directive

This provides output based on the common custom logic.It declares that our JSP page uses a set of custom tags, identified from the location of the library, and provides a means for identifying the custom tags in our JSP page.

<%@ tagliburi="{TLD_FILE}" prefix="{PREFIX}" %>

The components involved in tag libraries are listed below:

- Tag handler class-Common custom logic and HTML generation
- Descriptor configuration file - Directive's properties mapping information
- Taglib directive- This is used in JSP to utilize the tag handler functionality

Scripting Elements

In JSP, it is possible to embed a java code inside a java server page using the scriplet tag.This provision has been enabled due to the existence of scripting elements. This can contain any number of JAVA language statements, variable or method declarations, or expressions that are valid in the page scripting language.

There are three types of scripting elements:

- Expression • Scriplets
- Declaration

JSP expression tag

The code placed within the JSP expression tag has been evaluated, and converted to a string format and inserted in the place of expression as output stream of the response without using out.print() to write data.This is mainly used to print the values of variable or method. The expression element can contain any expression that is valid according to the Java Language specification but we cannot use a semicolon to end an expression.

The syntax of a JSP expression tag is listed below:

<%= Expression %>

An example of JSP expression tag is

```
<html>
<body>
<%= new java.util.Date() %>
<%=3+4 %>
</body>
</html>
```

The above expression tag is to output the current date instead of using a print statement.

JSP Scriplets

A scriplet tag is used to execute/ insert arbitrary piece of java source code in a JSP and inserted in the _jspService() method. The syntax for JSP scriplet is

<% java source code %>

An example of a JSP Scriplet that prints the user name is given below:

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>
```

An Example to print the current date using Scriplet is given below:

```
<%
```

java.util.Date d = new java.util.Date();

out.println ("Date and time is :" + d);

%>

JSP Declaration Tag

The JSP declaration tag is used to declare variables, methods and inner classes. It declares one or more variables or methods that can be used in Java code later in the JSP file. This can be used to declare the variables or methods before we use it in the JSP file.The code written inside the jsp declaration tag is placed outside the service() method of auto generated servlet.This doesn't get memory at each request. The syntax of JSP declaration tag is

<%! variable or method declaration %>

The difference between a JSP Scriplet tag and Declaration tag is listed in Table 2.

Table 2:Differences between JSP Scriplet and Declaration tag.

JspScriplet Tag	Jsp Declaration Tag
The jspscriptlet tag can declare only variables not methods.	The jsp declaration tag can declare variables as well as methods.
The declaration of scriplet tag is placed inside the _jspService() method.	The declaration of jsp declaration tag is placed outside the _jspService() method.

An example of JSP declaration tag that declares the variables is shown below:

index.jsp

<html>

<body>

<%! int data=50; %>

<%= "Value of the variable is:"+data %>

</body>

</html>

Another example of JSP declaration tag that declares the method is given below:

index.jsp

<html>

```
<body>
<%!
int cube(int n)
{
return n*n*n;
}
%>
<%= "Cube of 3 is:"+cube(3) %>
</body>
</html>
```

JSPElements - Example

All the three JSP elements are reviewed in the above examples. A complete example depicting the usage of all the JSP elements along with its output is shown in Figure 4.

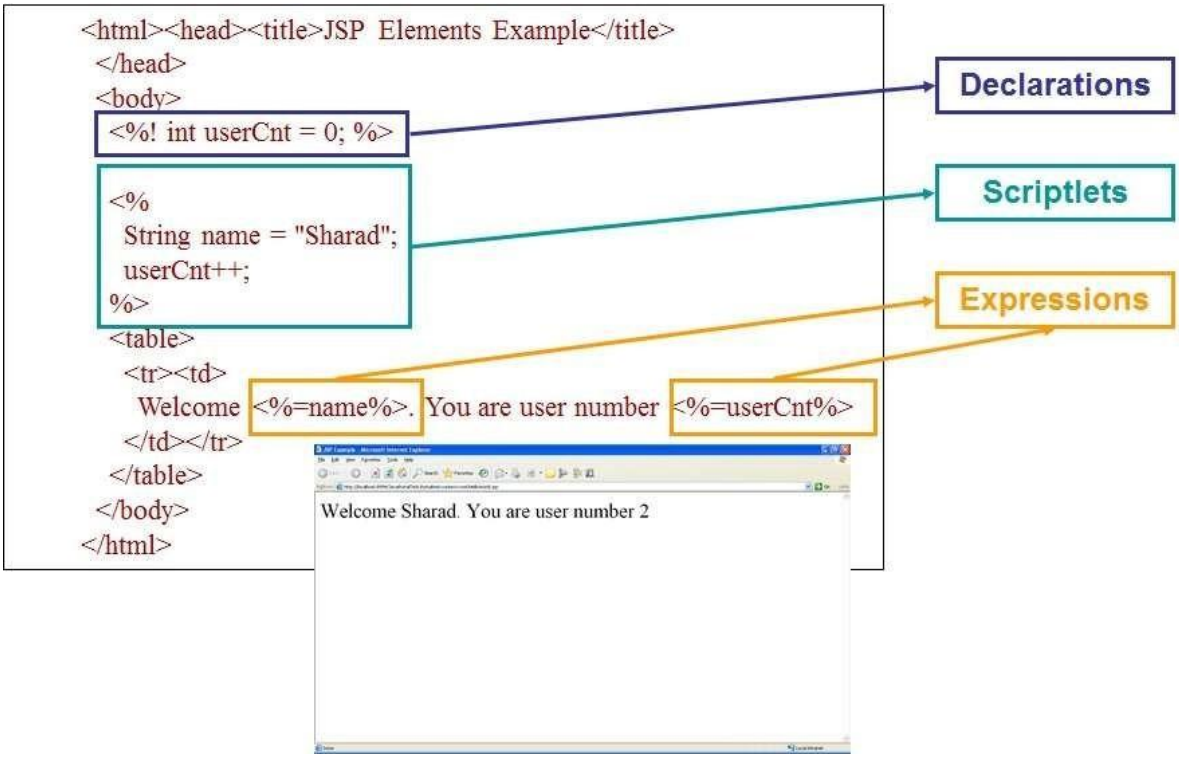


Figure 4: An example containing all the JSP elements and its output.

JSP Comments Tag

Comments are used to document the JSP pages and it can be inserted anywhere in the JSP page.Anything inside the comment tag is ignored by the JSP engine and not added to servlet's source code.The syntax for JSP comments Tag is

<%-- JSP comment --%>

An example depicting the JSP comment tag is as follows:

```
<html>
<body>
<%-- Prints cube results --%>
<% intcube(int n){   return n*n*n;
}  %>
<%= "Cube of 3 is:"+cube(3) %></body></html>
```

8.11 CREATION OF A SMALL WEB APPLICATION USING JSP

In this section, we are going to see the creation of a small web application using JSP. Here two jsp files has been created – Index.jsp and Display.jsp. Index.jsp file accepts a name from the user through an user interface. Later, in Display.jsp file, the user name has been parsed and a welcome message has been displayed using that name.

Index.jsp

```
<%@page contentType="text/html" pageEncoding="UTF-8"%>
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
   "http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
<title>JSP Page</title>
</head>
<body>
<form method="post" action="display.jsp">
```

```
Enter Name    <input type="text" name="t1" value=""><br>
<input type="submit" name="result">
</form>
</body>
</html>
```

Display.jsp

```
<%@ page language="java"%>
<html>
<body>
<%
String str=request.getParameter("t1");
out.println("Welcome to" + " "+ str);
%>
</body>
</html>
```

Output of Index.jsp and Display.jsp has been given in the figures 5 and 6.

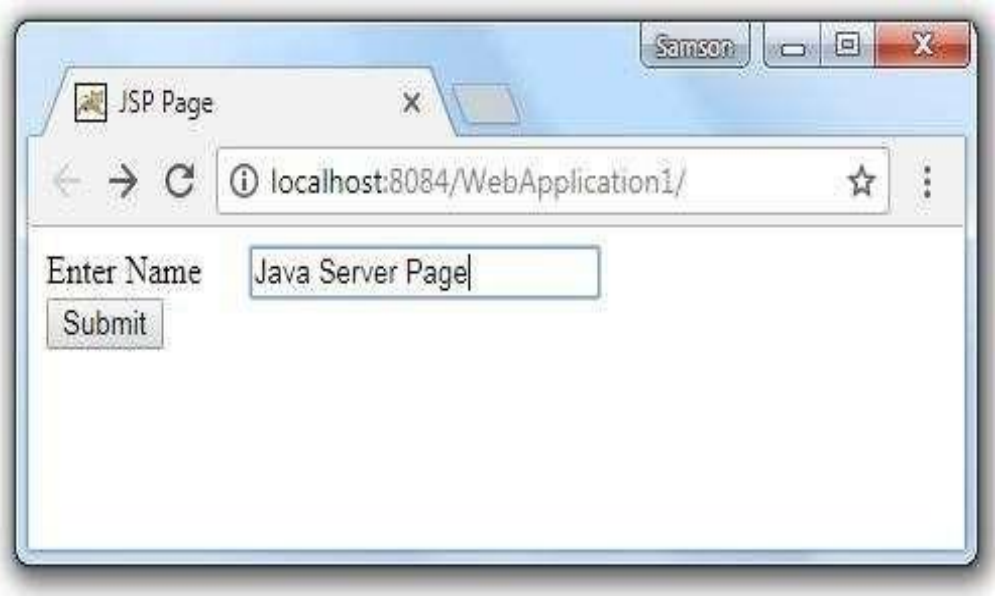


Figure 5. Output of Index.jsp

Figure 6. Output of Display.jsp

8.12 SUMMARY

In this module, we have discussed about the basics of JSP with its architecture and learnt the different ways in which a JSP works and discussed about the various elements of JSP with examples. We have also learnt the basic rules of the six JSP syntax elements—directives, declarations, scriplets, expressions, actions, and comments. Finally, we have seen a creation of a small web application using JSP.

8.13 CHECK YOUR PROGRESS

1. JSP stands for .....
2. Java servlet technology is a ..... side technology which generates web pages dynamically in response to client requests.
3. ....is a server-side technology and it is the higher abstraction of servlets, which separates dynamic content from static contents of a web page.
4. Apache Tomcat, Java Web Server, Web logic and Web Sphere are the.....
5. User requests a JSP page using a web browser through.....
6. .....processes the requested .jsp file and sends the corresponding JSP file to the JSP Servlet Engine.

8.14 ANSWER CHECK YOUR PROGRESS

1. Java Server Page
2. Server
3. JSP
4. web servers
5. Internet
6. Web server

8.15 MODEL QUESTION

1. JSP tags can be used for a variety of purposes, such as retrieving information from a database or registering user preferences, accessing JavaBeans components, passing control between pages and sharing information between requests, pages. Explain How?
2. What is Java Server Pages? How JSP used for Cross web server and Cross platform support?
3. Is Java Servlet develop Server side Java programs? How it solves scalability issues?

4. Why the JSP engine is required to process JSP pages? Also explain why it runs under the supervision of the servlet engine.
5. Explain the overall view of the JSP Architecture? Also explain the JSP and its working functionalities.
6. What is JSP container? How the JSP container is responsible for intercepting requests for JSP pages.

8.16 REFERENCES

- https://epgp.inflibnet.ac.in/Home/ViewSubject?catid=7

8.17 SUGGESTED READINGS

- Powell. Thomas A., JavaScript: The Complete Reference
- lemay. Laura, rafe colburn, jennifer kyrnin, Mastering HTML, CSS & JavaScript Web, BPB Publication, 2016
- vishvajeet. Sisodia, Basic of Web Design, HTML, CSS3, Centrum Press, 2014.

Block-III

UNIT-IX JAVA SERVER PAGES(JSP) II

9.1 Learning Objectives

9.2 JSP Elements – JSP Actions

9.3 JSP Implicit Objects

9.4 JSP - Exception Handling

9.5 JSP Session Tracking

9.6 Summary

9.7 Check your progress

9.8 Answer Check your progress

9.9 Model Question

9.10 References

9.11 Suggested readings

9.1 LEARNING OBJECTIVES

The objectives of this chapter is to understand the functionality of JSP elements/Action tagsand to learn the nature and functionality of JSP implicit objects. This is also intendedto understand the usage of Java Beans with JSP along with the concept of Session Tracking using JSP.

9.2 JSP ELEMENTS – JSP ACTIONS

Let us explore the JSP Action elements first. Actions are high-level JSP elements which are used to create, modify or use other JSP objects. Unlike directives and scripting elements, JSP actions are coded using strict XML syntax form. By using of JSP action elements we can dynamically insert a file, reuse JavaBeans components, forward the user to another page, or generate HTML for the Java plugins etc.Usage of syntax for the Action element, is given below as it conforms to the XML standard:

<jsp:action_name attribute="value" />

Action elements are basically predefined functions and the following JSP actions are available for use:

<jsp:param>,

<jsp:include>,

<jsp:forward>,

<jsp:fallback>,

<jsp:plugin>,

<jsp:useBean>,

<jsp:setProperty>,

<jsp:getProperty>

Let us explore the element tags one by one,

JSP Elements: JSP Actions (param)

The <jsp:param> action is basically used to provide information in the form of name value pairs to the current request. This action can also be used in alongwith jsp:include, jsp:forward andjsp:pluginactions. The syntax of jsp:param is

<jsp:param name=”parameter_name” value=”parameter_value” /> Example for jsp:param tag is:

<jsp: include page “process.jsp”>

<jsp:param name = “user” value = “mona”/> <jsp:param name = “sessionid” value = “1234D3F”/> </jsp:include>

JSP Elements: JSP Actions (include)

The jsp:include action tag is used to include the content of other resources such as a jsp, html or servlet file in a JSP page. The jsp:include tag can also be used to include static as well as dynamic pages. The jsp include action tag tries to include the resources at the request time and it is a good opinion for dynamic pages. The advantage of jsp: include is code reusability where we can make of use a page number of times.For example we can use jsp include action tag for including header and footer pages in all the pages. This may save a lots of time.

Syntax of jsp:include action tag without parameter is:

<jsp:include page="relativeURL | <%= expression %>" /> Syntax of jsp:include action tag with parameter is:

<jsp:include page="relativeURL | <%= expression %>">

<jsp:param name="parametername" value="parametervalue | <%=expression%>" /> </jsp:include>

An example of jsp:include action tag without parameter is given below. In this example, index.jsp file includes the content of the printdate.jsp file.

File: index.jsp

```
<h2>this is index page</h2>

<jsp:include page="printdate.jsp" />

 <h2>end section of index page</h2>
```

File: printdate.jsp

```
<% out.print ("Today is:"+java.util.Calendar.getInstance ().getTime ()); %>
```

Now the difference betweenjsp:include and include directive used in JSPshould be known. This is explained in Table 1.

Table 1. Comparison between jsp:include and include directive

	jsp:include action	**include directive**
Syntax:	<jsp:include page= .../>	<jsp:include page= .../>
When does inclusion occur?	Request time	Page translation time
What is included?	Output of the included page	Actual content of the file
How manyservlets result?	More than one (main page, and one for each included page)	One (included file is inserted into the main page, then the page is translated)
Maintenance	Changes included in pages does not require recompi- lation of the main page	The main page needs to be recompiled when the included page get changed.

JSP Elements: JSP Actions (forward)

While we use forward action tag , the current page stops processing the request and forwards the request to another page. After execution forwarding to the new page, execution never returns to the calling or current page.

Syntax of JSP Action(forward) tag is :

<jsp:forward page= “file_name” />

Or it can be also be written as:

<jsp: forward page= “file_name”>

<jsp:param name=”parameter_name” value=”parameter_value” /></jsp: forward> An example of jsp:forward action tag without parameter is discussed here. In the following example, an attempt has been made to simply forward the request to the print date .jsp file.

index.jsp

```
<html>
<body>
<h2>this is index page</h2>
 <jsp:forward page="printdate.jsp" />
```

</body>

</html>

JSP Elements: JSP Actions (useBean)

Usage of bean tag through JSP actions instantiates a Java bean object into the JSP page. (UseBeanProperty has been used in two flavors: getProperty and setProperty. jsp:useBean creates a new instance of a bean and its syntax is

<jsp:useBean id=“object_Name" class=“class name“ >

Where id is the name of the object, class is a Javabean class from which the object gets instantiated.

This is equivalent to the JSP systax:

<% class_name object_name = new class_name(); %>

Example to instantiate a Factorial bean in a JSP page is given below:

<jsp:useBean id = “fact” class = “bean.Factorial”/>

This is equivalent to the JSP scriplet

<% bean.Factorial fact = new bean.Factorial(); %>

JSP Actions (jsp:setProperty)

jsp:getProperty reads and outputs the value of a bean property (ie.,calling a getter method) and its syntax is

<jsp:getProperty name="beanName" property="propertyName" />

An example for gtPropertyis <jsp:getProperty name = “fact” property = “value”/> This is equivalent to JSP Scriplet:

<%= fact.getValue() %> 1.4.2 JSP Actions (jsp:setProperty)

jsp:setProperty modifies a bean property (by calling a setter method)

<jsp:setProperty name="beanName" property="propertyName” value="propertyValue" />

An example for Set Property is:

<jsp: setProperty name=“fact” property = “value” value =“5” />

This is equivalent to the JSP Scriplet:

<%fact.setValue(5)%>

CHECK YOUR PROGRESS

True/False type questions

1. Servlet and Java script are high-level JSP elements which are used to create, modify or use other JSP objects.
2. Unlike directives and scripting elements, JSP actions are coded using strict DBMS syntax form.
3. By using of JSP action elements we can dynamically insert a file, reuse JavaBeans components, forward the user to another page, or generate HTML for the Java plugins etc.
4. The <jsp:param> action is basically used to provide information in the form of name value pairs to the current request.
5. The <jsp:param> action can also be used alongwith jsp:include, jsp:forward andjsp:pluginactions.

Answers-

1. False
2. False
3. True
4. True
5. True

9.3 JSP IMPLICIT OBJECTS

JSP implicit objects are created by the container automatically. When a web server processes a JSP page(during the translation of a JSP page to a Servlet source to help developers). These objects can also be used directly in scriptlets that goes in service method. Nine Implicit Objects can be used in a JSP page as discussed below:

a) Request object: Request implicit object is to get the data on a JSP page which has been entered by the user in frontend

```
String Uid= request.getParameter("user-id");

String Pass= request.getParameter("password");
```

b) Response Object: This is the HttpServletResponse object associated with the response to the client this is used to send responses to the client.

response.setContentType("text/html");

response.setContentType("image/gif"); response.setContentType("image/png");

response.setContentType("application/pdf");

c) Session:This is the most frequently used implicit object, which is used for storing the user's data to make it available on other JSP pages till the user session is active.

d) Out: This is used for writing content to the client(browser). This has several methods which can be used for properly formatting output message to the browser and for dealing with the buffer. Example:

out.println("hi"); out.println("hello");

e) Application:This is used for getting application-wide initialization parameters and to maintain useful data across whole JSP application.

Example:

String s = (String)application.getAttribute("MyAttr");

This example would return the object stored in the attribute "MyAttr".

f) Exception:This object is used in exception handling for displaying error messages. This object is only available to the JSP pages, whereisErrorPage attribute is set to true.

g) Page : JSP page implicit object is an instance of java.lang.Object class, which indicates the current JSP page. The Page object provide reference to the generated servlet class.

h) PageContext Object : This implicit object is an instance of javax.servlet.jsp.PageContextabstract classimplementation. PageContext to get and set attributes with different scopes and to forward request to other resources. PageContext objectccan be used also holds references to other implicit objects.

i) Config Object : This is used to get the JSP initparamsconfigured in the deployment descriptor.

JSP Implicit objects – Class files (instantiation)

The JSP Class files which are involved in Implicit objects are defined below:

- application: javax.servlet.ServletContext
- config: javax.servlet.ServletConfig
- exception: java.lang.Throwable
- out: javax.servlet.jsp.JspWriter

- page: java.lang.Object
- pageContext: javax.servlet.jsp.PageContext
- request: javax.servlet.ServletRequest
- response: javax.servlet.ServletResponse
- session: javax.servlet.http.HttpSession

Example for the JSP Implicit objects

To demonstrate the usage of JSP Implicit objects, an example which makes use of Request, Response, Out and session objects has been given below,

Index.html

```
<html>
<head>
<title>Login Page</title>
</head>
<body>
<form action="checkdetails.jsp">
    UserId:<input type="text" name="id" /><br><br> Password: <input type="text" name="pass" /><br><br>
    <input type="submit" value="Sign In!!"/>
</form>
</body>
</html>
```

Sign in Form: Receiving id and password from User Login page checkdetails.jsp

```
<html>
<head>
<title>Check Credentials</title>
</head><body>
<% String uid=request.getParameter("id");
```

```
String password=request.getParameter("pass");
session.setAttribute("session-uid", uid);
if(uid.equals("Chaitanya") &&password.equals("BeginnersBook"))
        {  response.sendRedirect("success.jsp"); }
else
        { response.sendRedirect("failed.jsp"); } %>
</body>
</html>
```

This JSP page verifies the input id/pass against hard-coded values. success.jsp

```
<html>
<head>
<title>Success Page</title>
</head>
<body>
<% String data=(String)session.getAttribute ("session-uid");
        out.println("Welcome "+ data+"!!"); %>
</body>
</html>
```

This JSP page would execute if id/pass are matched to the hardcoded userid/password.

failed.jsp

```
<html>
<head>
<title>Sign-in Failed Page</title>
</head>
<body>
<% String data2=(String)session.getAttribute ("session-uid");
out.println("Hi "+ data2+". Id/Password are wrong. Please try Again."); %>
```

</body>

</html>

The control will be redirected to this page if the credentials entered by user are wrong.

Output of JSP page thatreceives id and password andcorrectlyverified is shown in Figure 1.

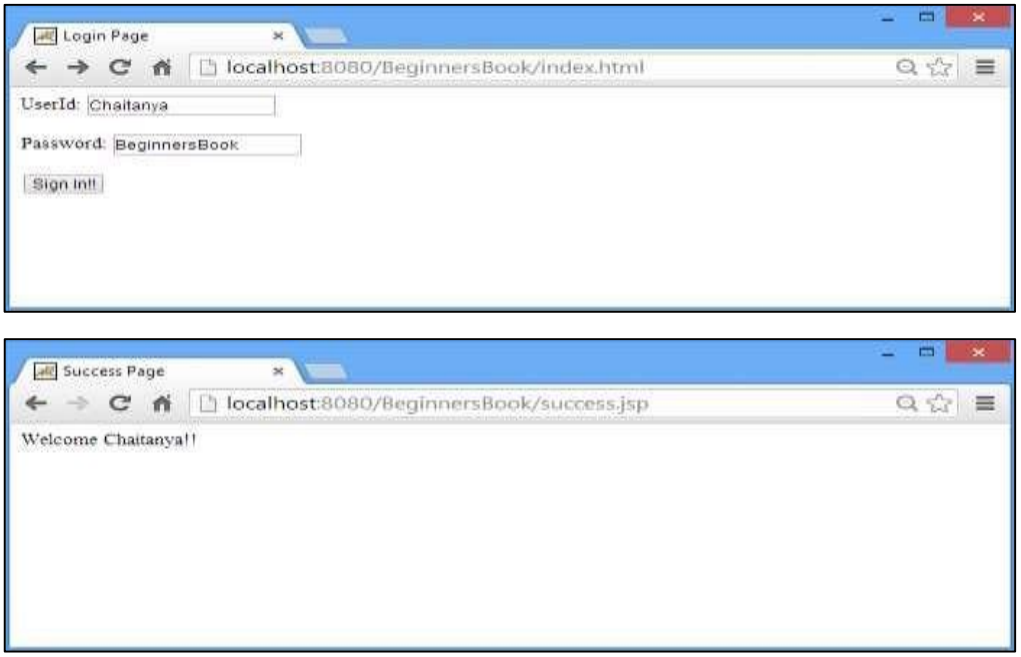


Figure 1. Output of JSP page that receive id and password andcorrect verification

If Sign-In has been failed ,when Id and Password provided is are wrong the output is displayed shown in Figure2.

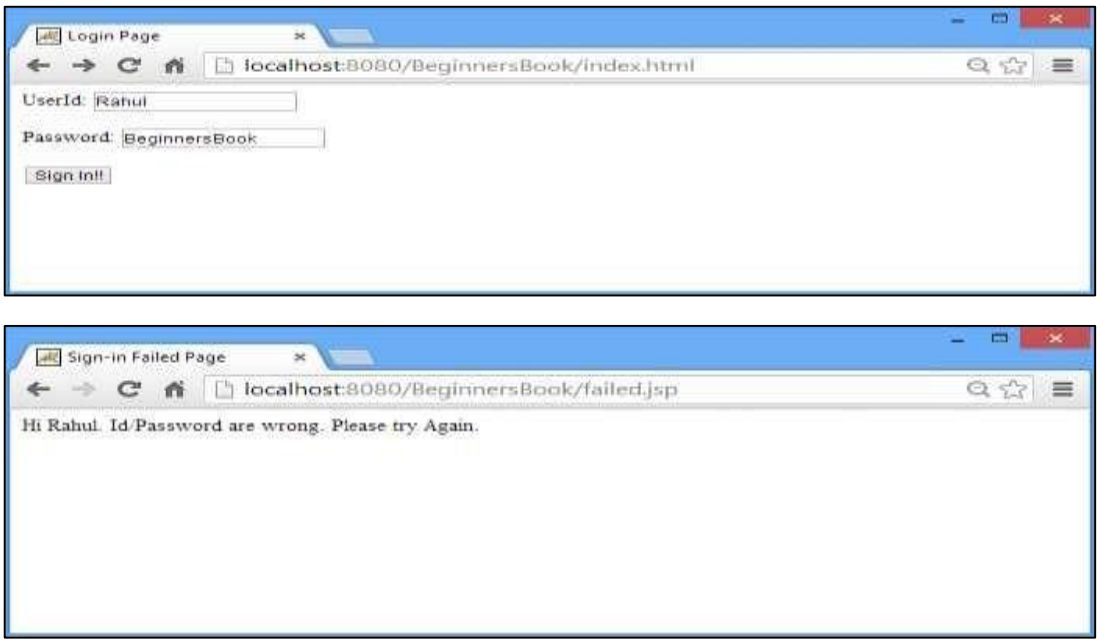


Figure 2: Output for Sign In failed page

9.4 JSP - EXCEPTION HANDLING

The exception is normally an object that is thrown during the runtime. Exception Handling is the processused to handle the runtime errors. There may be a possibility that exception may occur at any time in our web application. So handling exceptions is a safer side for the web developers.

In JSP, there are two ways to perform exception handling, one is by using errorPage and isErrorPage attributes of page directive and the other one is by using<error-page> element in web.xml file.

Example of JSPexception handling using the elements of page directive

To demonstrate the usage of JSP Exception Handling, three jsp pages have been designed in this example. Out of which, one page has been defined/created to handle the exceptions(in this example names error.jsp). Another page is defined with errorPage attribute of page directive.(as defined by the process.jsp page) and one more page is used for getting values.

Define and create a page to handle the exceptions, named error.jsp page, where the possibility of exception may arise, For example : 3 files are designed

– index.jsp for getting input values

– process.jsp for dividing the two numbers and displaying the result – error.jsp for handling the exception

index.jsp

```
<form action="process.jsp">
        No1:<input type="text" name="n1" /><br/><br/>
        No1:<input type="text" name="n2" /><br/><br/>
        <input type="submit" value="divide"/>
</form>
```

process.jsp

```
<%@ page errorPage="error.jsp" %>
<%      String num1=request.getParameter("n1");
        String num2=request.getParameter("n2");
```

```
int a=Integer.parseInt(num1);
int b=Integer.parseInt(num2);
int c=a/b;
out.print("division of numbers is: "+c);
%>
```

error.jsp

```
<%@ page isErrorPage="true" %>
<h3>Sorry an exception occured!</h3>
  Exception is: <%= exception %>
```

Output of this example is given in Figure 3 and Figure 4.

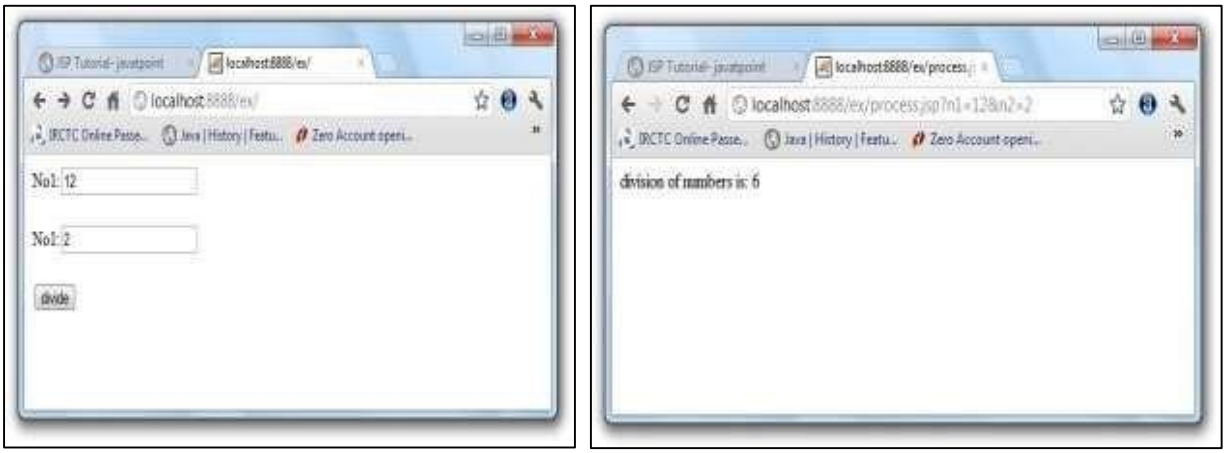


Figure 3. Output for division with correct answer (using Exception Handling)

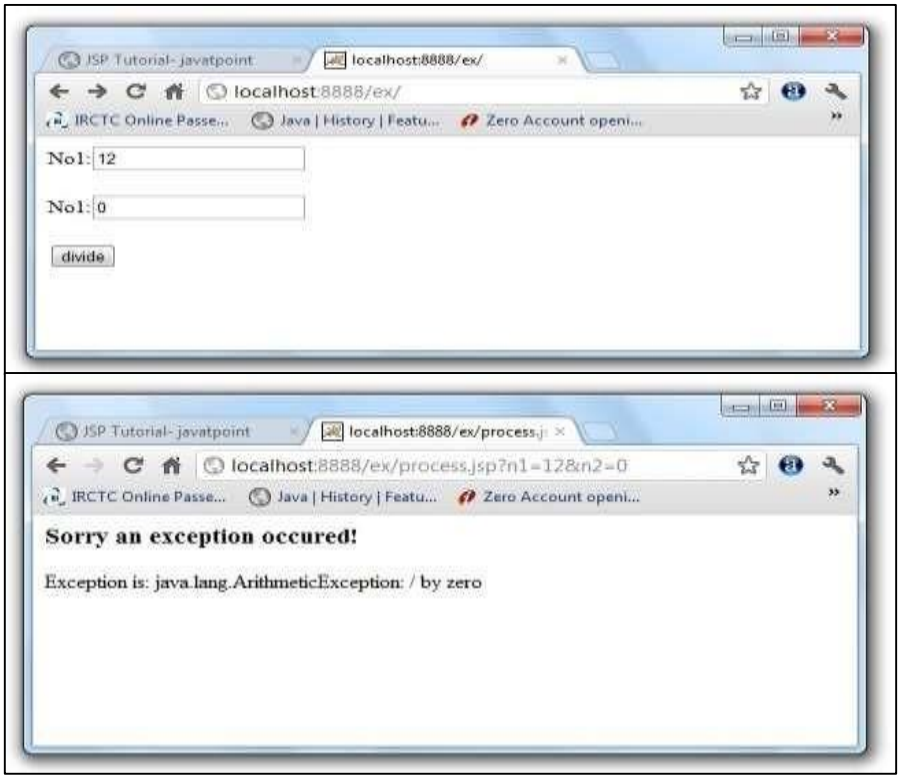


Figure 4. Output for division when an exception occurred (using Exception Handling)

9.5 JSP SESSION TRACKING

In a web application, server may be responding to several clients at a time. Hence, so session tracking is a way by which a server can identify the client uniquely.As we know HTTP protocol is stateless which means client needs to open a separate connection every time it interacts with server and server treats each request as a new request. In order to identify the client, server needs to maintain the state and to do so, there are several session tracking techniques. They are listed below:

Session Tracking Techniques

Session Tracking has following fields:

- Hidden fields
- URL rewriting
- Cookies
- Session Objects

With Cookies, Hidden Fields and URL rewriting approaches, client always sends a unique identifier with each request and server determines the user session based on that unique

identifier whereas in the session tracking approach using Session Object it uses all the other three techniques internally.

Session tracking using Cookies:

Out of the four session tracking techniques, session tracking using cookie has been discussed here, If cookie is associated with the client request, server will associate it with corresponding user session otherwise it will create a new unique cookie and send it back with response. Cookie objects are can be created using a name value pair.For example we can create a cookie with the name sessionId and with a unique value for each client. Later this can be added it in a response so that it will be sent back to the client.

Syntax of creating Cookie in a JSP is:

```
Cookie cookie = new Cookie(“sessionID”, “some unique value”);
response.addCookie(cookie);
```

The following examples illustrate the usage of cookie1.jsp is used to accept the username. Cookie2.jsp creates and sets a cookie with maximum age, whereas cookie3.jsp file is used to display the values of the cookies. cookie1.jsp:

```
<html>
<body>
<form method = "post" action="cookie2.jsp">
    <font>Username<input type = "text" name = "name"></font>
    </font><br>
    <input type = "submit" name = "submit" value = "submit" >
</form>
</body>
</html>
```

cookie2.jsp:

```
<%@ page language="java" import="java.util.*"%>
<%    String name=request.getParameter("name");
      Cookie cookie = new Cookie ("name",name);
```

```
        response.addCookie(cookie);
        cookie.setMaxAge(50 * 50); //Time is in Minutes
%>
<a href="cookie3.jsp">Continue</a>
```

cookie3.jsp:

```
 <p>Display the value of the Cookie</p>
 <%
        Cookie[] cookies = request.getCookies();
        for (inti=0; i<cookies.length; i++)
        {
                if(cookies[i].getName().equals("name"))
                out.println("Hello"+cookies[i].getValue());
        }
 %>
```

With respect to the above code, output of Cookie1.jsp, Cookie2.jsp and Cookie3.jsp has been shown in the figures 5, 6, 7 respectively.

Figure 4. Output of Cookie1.jsp

Figure 5. Output of Cookie2.jsp

Figure 6. Output of Cookie3.jsp

9.6 SUMMARY

This Chapter illustrated the usage of JSP action tags with its various types and formats.This Chapter has also demonstrated the usage of various implicit objects with suitable examples. Later the usage of cookies in session tracking has also been dealt.

9.7 CHECK YOUR PROGRESS

1. The.....action tag is used to include the content of other resources such as a jsp, html or servlet file in a JSP page.
2. The jsp:include tag can also be used to include static as well as..... pages.
3. After execution .....to the new page, execution never returns to the calling or current page.

4. While we use.....action tag , the current page stops processing the request and forwards the request to another page.
5. ..... reads and outputs the value of a bean property (ie.,calling a getter method)
6. JSP.....objects are created by the container automatically.
7. JSP ..... Objects can also be used directly in scriptlets that goes in service method.
8. .....object associated with the response to the client this is used to send responses to the client.
9. .....is the most frequently used implicit object, which is used for storing the user's data to make it available on other JSP pages till the user session is active.

9.8 ANSWER CHECK YOUR PROGRESS

1. jsp:include
2. dynamic
3. forwarding
4. forward
5. jsp:getProperty
6. implicit
7. Implicit
8. HttpServletResponse
9. Session

9.9 MODEL QUESTION

1. What is the use of jsp:include tag? The jsp include action tag tries to include the resources at the request time? Explain why.
2. What are the advantages of jsp: include? How it is beneficial for code reusability? Also explain how we can use a page number of times.
3. What are the different JSP Elements? What are their uses? Explain each.
4. What is the difference between <jsp:setProperty> and <jsp:getProperty>?
5. What do you understand by JSP Implicit Objects? How many Implicit Objects can be used in a JSP page?
6. What is the difference between Page Context Object and Config Object? How Page Context object can be used also holds references to other implicit objects?

9.10 REFERENCES

- https://epgp.inflibnet.ac.in/Home/ViewSubject?catid=7

9.11 SUGGESTED READINGS

- Holzner. Steven, PHP: The Complete Reference
- Powell. Thomas A., JavaScript: The Complete Reference
- vishvajeet. Sisodia, Basic of Web Design, HTML, CSS3, Centrum Press, 2014.

UNIT-X JAVA SERVER PAGES (JSP) WITH JDBC

10.1 Learning Objectives

10.2 JDBC

10.3 Basic steps to connect database

10.4 Install Mysql

10.5 Steps to run JSP using Tomcat server

10.6 Creating a table using JSP example

10.7 Creating a table: database

10.8 SELECT operation

10.9 INSERT operation

10.10 DELETE Operation

10.11 Summary

10.12 Check your progress

10.13 Answer Check your progress

10.14 Model Question

10.15 References

10.16 Suggested readings

10.1 LEARNING OBJECTIVES

This module focuses to know about how to install MySQL and to run JSP pages in Tomcat. The basic steps to connect to a database are also explained along with line commands to use in the code. The main goal of this section lies in the discussion about how to access and manipulate database from a JSP Page using JDBC.

10.2 JDBC

JDBC is an application programming interface between Java programs and database management systems. JDBC is a core part of the Java platform and is included in the standard JDK distribution. The purpose of JDBC is to connect database and manipulate the data in database from a Servlet page or from a JSP page.

10.3 BASIC STEPS TO CONNECT DATABASE

1. Establish a connection.
2. Create JDBC Statements.
3. Execute SQL Statements.
4. Get ResultSet.
5. Close connections.

Step Description

1. Establish a connection

A connection to MySQL is established by using the package,

```
import java.sql.*;
```

Then load the vendor specific driver,

```
Class.forName("oracle.jdbc.driver.OracleDriver");
```

This code dynamically loads a driver class, for Oracle database.

Make the connection,

```
Connection conn = DriverManager.getConnection(DB_URL,USER,PASS);
```

This code establishes connection to database by obtaining a Connection object.

2. Create JDBC statement(s)

JDBC statements are created using the following code,

```
Statement stmt = con.createStatement() ;
```

This creates a Statement object for sending SQL statements to the database.

3. Executing SQL Statements

SQL statements are executed using the query,

```
sql = "SELECT attributes FROM Table";
```

4. Get ResultSet

The Resultset object is obtained by the executeQuery() function,

```
ResultSet rs = Stmt.executeQuery(sql);
```

5. Close Connection

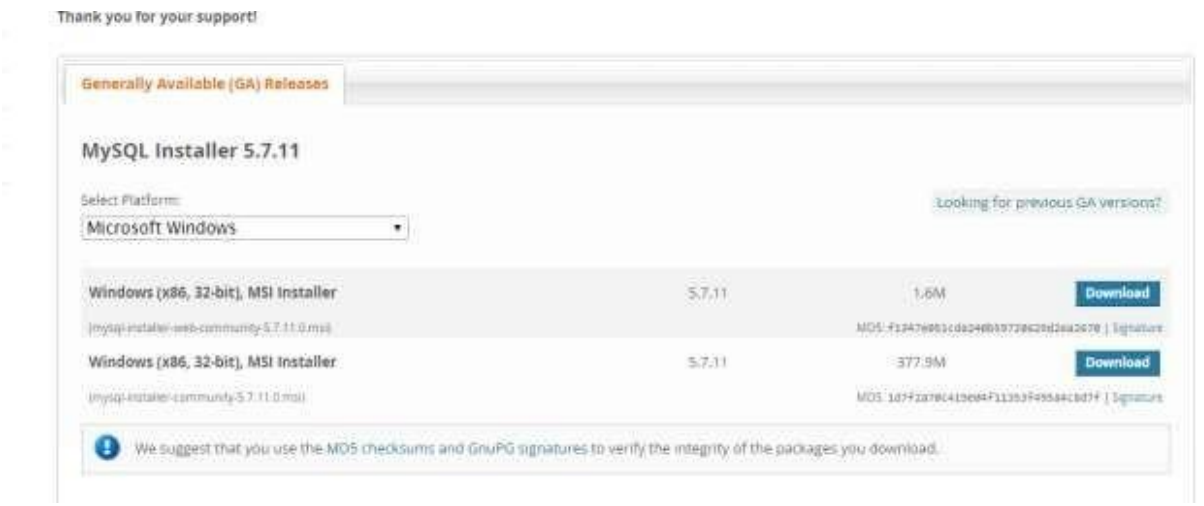
The JDBC connection and SQL query execution is closed by calling the close() function,

```
con.close();
```

10.4 INSTALL MYSQL

Install Mysql Database from MYSQL official website dev.mysql.com

Select the Platform type (i.e Windows or Mac)and download the Installer.



CHECK YOUR PROGRESS

True/False type questions

1. Multithrading is an application programming interface between Java programs and database management systems.
2. JDBC is a core part of the Java platform and is included in the standard JDK distribution.
3. The purpose of Java Script is to connect database and manipulate the data in database from a Servlet page or from a JSP page.
4. Basic steps to connect database are Establish a connection, Create JDBC Statements, Execute SQL Statements, Get ResultSet, Close connections.
5. Class.forName("oracle.jdbc.driver.OracleDriver") this code dynamically loads a driver class, for Oracle database.

Answers-

1. False
2. True
3. False
4. True
5. True

10.5 STEPS TO RUN JSP USING TOMCAT SERVER

STEP 1: To run JSP pages, Download TOMCAT SERVER from Website.

URL: https://tomcat.apache.org/download-60.cgi

STEP 2: The tomcat folder has the following files as bin, conf, lib, logs, temp, webapps and work.

STEP 3: Create directory "myjsp" under the webapps directory.

STEP 4: Save this file in shown in the given path,

C:/ Program Files/…/…/webapps/myjsp/Filename.

STEP 5: Download Mysql connector jar file from the following url.

URL:https://dev.mysql.com/downloads/connector/j/3.1.html

STEP 6:Copy the jar file to apache-tomcat\webapps\ROOT\WEB-INF\lib directory.

STEP 7: To run JSP program

- Start the tomcat service,

  Start -> run -> services.msc -> Select Apache Tomcat -> start

- Open new tab in browser or open new window,

  http://localhost:8080/myjsp/Filename.jsp

10.6 CREATING A TABLE USING JSP EXAMPLE

Code

The following code is for creating a table using a JSP page. First establish a connection to the database from the JSP page. When connection is successful, execute the SQl statement "CREATE table test (testid mediumint(8), name varchar(100))" using the executeUpdate() function.

```
<%@page contentType="text/html" pageEncoding="UTF-8"%>
<!DOCTYPE html><%@ page import="java.sql.*" %>
<%@ page import="com.mysql.jdbc.Driver" %>
<%!
        String driver = "com.mysql.jdbc.Driver";
        String url = "jdbc:mysql://localhost/PUB";
        String name = "root"; String pass="password";
%>
<html>
<head>
        <title>testJSP</title>
</head>
<body>
<p>Attempting to open JDBC connection to:... </p><%=url%>
 <%
Try
{
        String tableStr = "CREATE table test (testid mediumint(8), name varchar(100))";
        Class.forName( driver );
        Connection con = DriverManager.getConnection( url, name, pass );
        Statement stat = con.createStatement();
        %>
        <p> executing: <%=tableStr%></p>
        <%
        stat.executeUpdate( tableStr );
        %>
        <p> success .... </p>
```

```
        <%
        con.close();
}
catch (SQLException sqle)
{
        out.println("<p> Error opening JDBC, cause:</p> <b> " + sqle + "</b>");
 }
catch(ClassNotFoundException cnfe)
{
        out.println("<p> Error opening JDBC, cause:</p> <b>" + cnfe + "</b>");
}
%>
</body>
</html>
```

10.7 CREATING A TABLE: DATABASE

OUTPUT:

The output of the above code is shown below.

DATABASE:

10.8 SELECT OPERATION

Example Code:

The following code is for selecting records from a table using a JSP page and display the records in a table format. First establish a connection to the database from the JSP page. When connection is successful, execute the SQL statement "select * from Publisher" using the executeQuery() function.

```
<%@page contentType="text/html" pageEncoding="UTF-8"%>

<!DOCTYPE html>

<%@ page import="java.sql.*" %>

<% Class.forName("com.mysql.jdbc.Driver"); %>

<HTML>
  <HEAD>
    <TITLE>The Publishers Database Table </TITLE>
  </HEAD>
  <BODY>
    <H1>The Publishers Database Table </H1>
    <%
      Connection connection = DriverManager.getConnection(
      "jdbc:mysql://localhost/PUB", "root", "password");
      Statement statement = connection.createStatement() ;
```

```
        ResultSet resultset = statement.executeQuery("select * from Publisher") ;
      %>
      <TABLE BORDER="1">
        <TR>
          <TH>ID</TH>
          <TH>Name</TH>
          <TH>City</TH>
          <TH>State</TH>
          <TH>Country</TH>
        </TR>
        <% while(resultset.next())
        { %>
        <TR>
          <TD> <%= resultset.getString(1) %></td>
          <TD> <%= resultset.getString(2) %></TD>
          <TD> <%= resultset.getString(3) %></TD>
          <TD> <%= resultset.getString(4) %></TD>
          <TD> <%= resultset.getString(5) %></TD>
        </TR>
        <% } %>
      </TABLE>
    </BODY>
</HTML>
```

OUTPUT:

The output of the above code is shown below.

The Publishers Database Table

ID	Name	City	State	Country
1	ARNOLD	CHENNAI	TAMILNADU	INDIA
2	ANAND	BANGLORE	KARNATAKA	INDIA

10.9 INSERT OPERATION

Example Code:

The following code is for inserting records into a table using a JSP page. First establish a connection to the database from the JSP page. When connection is successful, execute the SQl statement "INSERT INTO Employees VALUES (105, 22, 'Jeeva', 'Kumar')".

```
<%@ page import="java.io.*,java.util.*,java.sql.*"%>

<%@ page import="javax.servlet.http.*,javax.servlet.*" %>

<%@ taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c"%>

<%@ taglib uri="http://java.sun.com/jsp/jstl/sql" prefix="sql"%>

<html>

<head>

<title>INSERT Operation</title>

</head>

<body>

<sql:setDataSource var="snapshot" driver="com.mysql.jdbc.Driver"
url="jdbc:mysql://localhost/TEST"  user="root"  password="password"/>

<sql:update dataSource="${snapshot}" var="result">

INSERT INTO Employees VALUES (105, 22, 'Jeeva', 'Kumar');
```

</sql:update>

<sql:query dataSource="${snapshot}" var="result">

SELECT * from Employees;

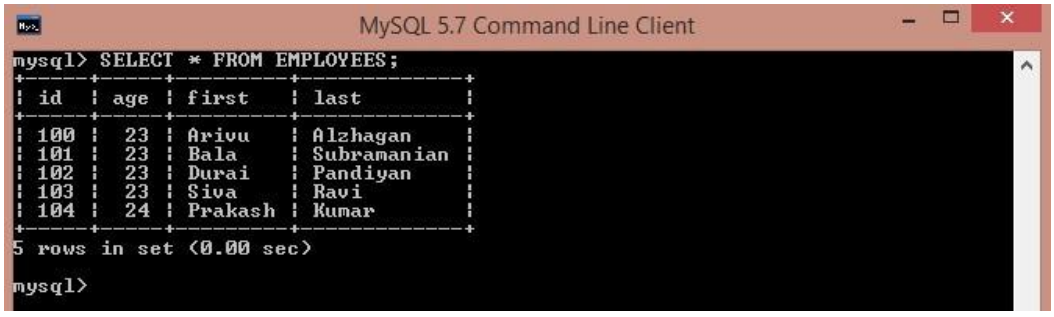
</sql:query>

Output:

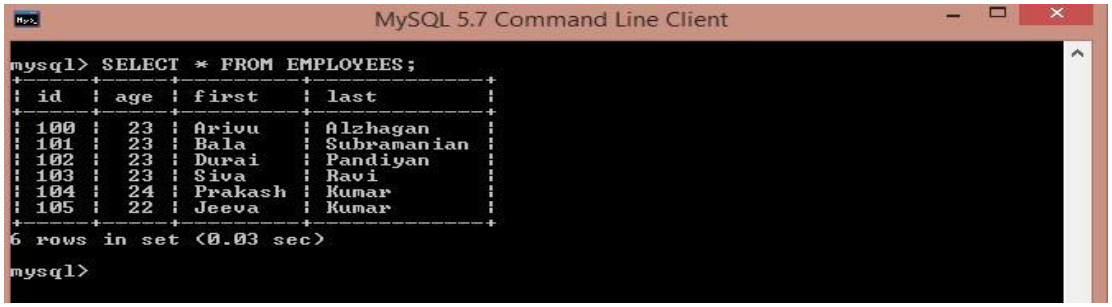
The output of the above code is shown below.

Emp ID	First Name	Last Name	Age
100	Arivu	Alzhagan	23
101	Bala	Subramanian	23
102	Durai	Pandiyan	23
103	Siva	Ravi	23
104	Prakash	Kumar	24
105	Jeeva	Kumar	22

Database (Before Executing):



Database (After Executing):



10.10 DELETE OPERATION

The following code is for deleting records from a table using a JSP page. First establish a connection to the database from the JSP page. When connection is successful, execute the SQL statement "DELETE FROM Employees WHERE Id = ?".

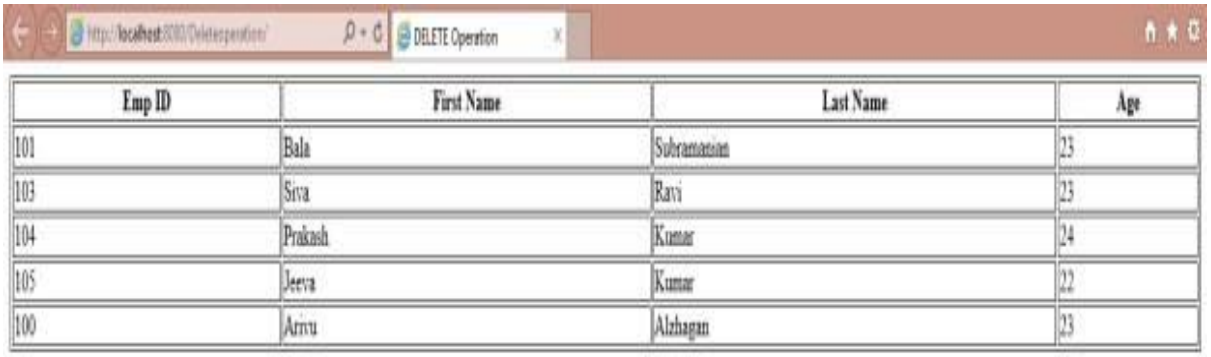
Example

```
<%@ page import="java.io.*,java.util.*,java.sql.*"%>

<%@ page import="javax.servlet.http.*,javax.servlet.*" %>

<%@ taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c"%>

<%@ taglib uri="http://java.sun.com/jsp/jstl/sql" prefix="sql"%>

 <html>

<head>

<title>DELETE Operation</title>

</head>

<body>

 <sql:setDataSource var="snapshot" driver="com.mysql.jdbc.Driver"
url="jdbc:mysql://localhost/TEST" user="root" password="password"/>

 <c:set var="empId" value="102"/>

 <sql:update dataSource="${snapshot}" var="count">

  DELETE FROM Employees WHERE Id = ?

 <sql:param value="${empId}" />

</sql:update>

<sql:query dataSource="${snapshot}" var="result">
```

```
  SELECT * from Employees;
</sql:query>
 <table border="1" width="100%">
<tr>
  <th>Emp ID</th>
  <th>First Name</th>
  <th>Last Name</th>
  <th>Age</th>
</tr>
<c:forEach var="row" items="${result.rows}">
<tr>
  <td><c:out value="${row.id}"/></td>
  <td><c:out value="${row.first}"/></td>
  <td><c:out value="${row.last}"/></td>
  <td><c:out value="${row.age}"/></td>
</tr>
</c:forEach>
</table>
</body>
</html>
```

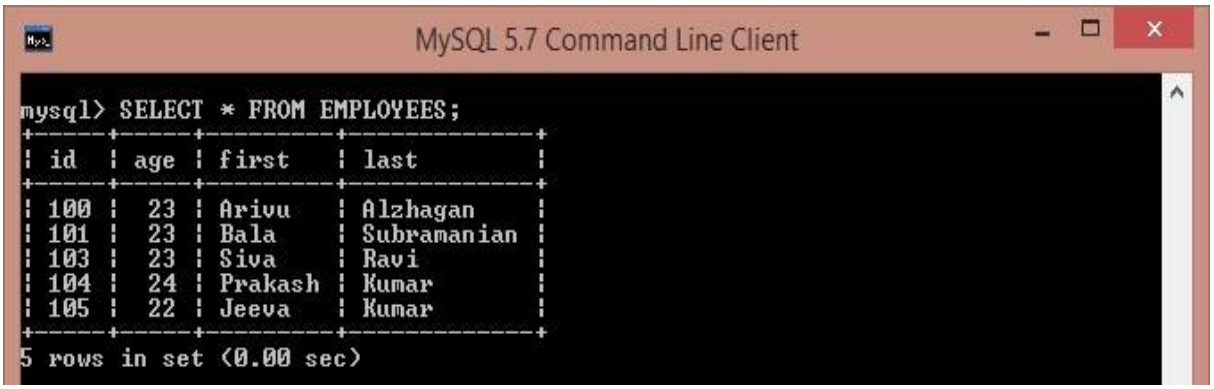
Output:

The output of the above code is shown below.



Emp ID	First Name	Last Name	Age
101	Bala	Subramanian	23
103	Siva	Ravi	23
104	Prakash	Kumar	24
105	Jeeva	Kumar	22
100	Arivu	Alzhagan	23

Database:



Login Application

Example

//Index.html

<body>

<form action="Login.jsp" method="post"> User name :<input type="text" name="userid" /> password :<input type="password" name="password" />

<input type="submit" />

</form>

</body>

Login.jsp

<%@ page import ="java.sql.*" %>

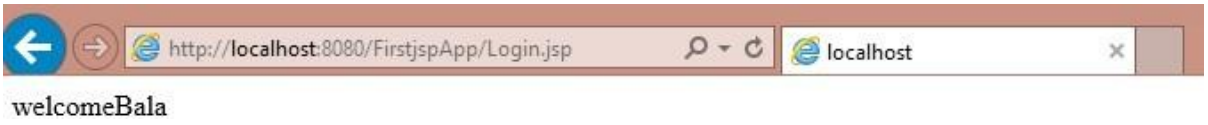
<%@ page import ="javax.servlet.http.*" %>

```
<%
String userid=request.getParameter("userid");
session.putValue("userid",userid);
String password=request.getParameter("password");
Class.forName("com.mysql.jdbc.Driver");
java.sql.Connection con =
DriverManager.getConnection("jdbc:mysql://localhost/test","root","password");
Statement st= con.createStatement();
ResultSet rs=st.executeQuery("select * from users where user_id='"+userid+"'");
if(rs.next())
{
if(rs.getString(2).equals(password))
{
out.println("welcome"+userid);
}
else
{
out.println("Invalid password try again");
}
}
else
%>
```

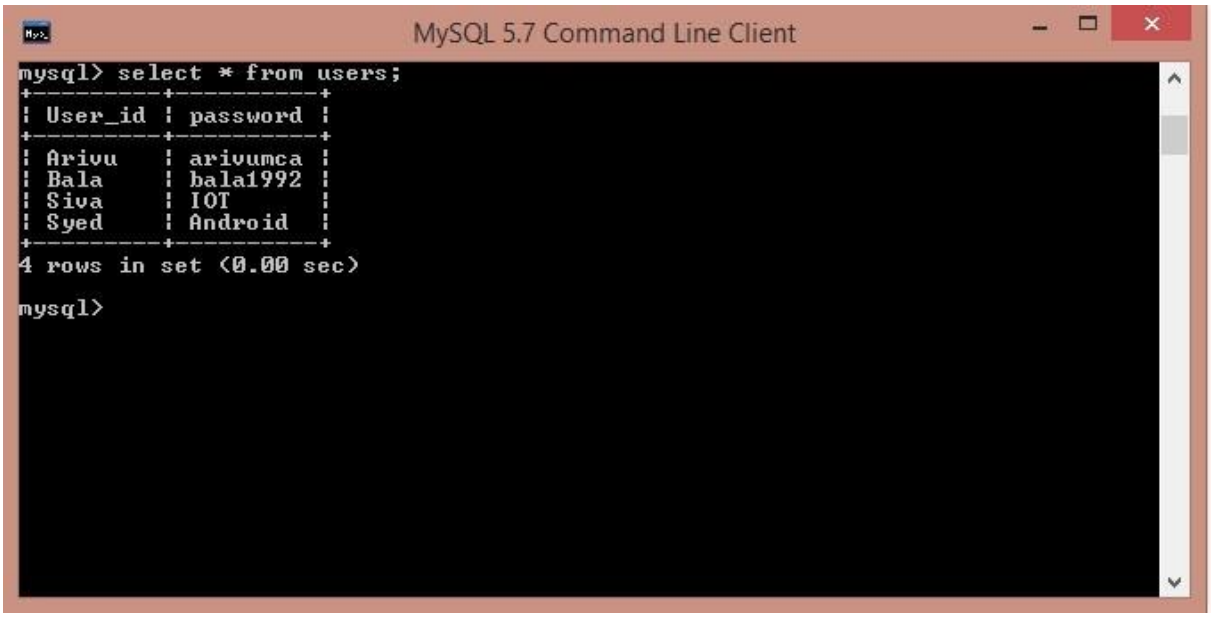
Output:

The output of the above code is shown below.

Success



Database:



10.11 SUMMARY

In this section we discussed about how to install MySQL and run JSP pages in Tomcat. The module also explains about the operational views of how to Create table, Insert and Delete data in JDBC from JSP page illustrated with needed examples. Finally, this module also demonstrates a simple login Application created using JSP with JDBC.

10.12 CHECK YOUR PROGRESS

1. Connection conn = DriverManager.getConnection(DB_URL,USER,PASS) statement can establishes ..... to database by obtaining a Connection object.
2. Statement stmt = con.createStatement() creates a Statement object for sending .....statements to the database.
3. The JDBC connection and SQL query execution is closed by calling the .....function.
4. SQL statement "INSERT INTO Employees VALUES (105, 22, ‘Jeeva', ‘Kumar')" is used to ..... new record into a table.
5. To .....a records from a table the SQL statement "DELETE FROM Employees WHERE Id = ?" is used.

10.13 ANSWER CHECK YOUR PROGRESS

1. connection
2. SQL
3. close()
4. insert
5. delete

10.14 MODEL QUESTION

1. How will you install Mysql Database from MYSQL official website dev.mysql.com? Which Platform types are used to download the Installer?
2. What is JDBC? What are the basic steps to connect database?
3. What is tomcat server? What are the Steps to run a JSP program using tomcat server? Explain.
4. How will you create a table using Java Server Pages(JSP)? Explain with help of suitable example.
5. How will you write a SELECT query in MySql? Write a code for selecting records from a table using a JSP page and display the records in a table format.

10.15 REFERENCES

- https://epgp.inflibnet.ac.in/Home/ViewSubject?catid=7

10.16 SUGGESTED READINGS

- Powell. Thomas A., JavaScript: The Complete Reference
- lemay. Laura, rafe colburn, jennifer kyrnin, Mastering HTML, CSS & JavaScript Web, BPB Publication, 2016
- Vishvajeet. Sisodia, Basic of Web Design, HTML, CSS3, Centrum Press, 2014.

UNIT-XI JDBC – INTRODUCTION

11.1 Learning Objectives

11.2 JDBC

11.3 JDBC Architecture

11.4 JDBC Drivers

11.5 Working with JDBC

11.6 JDBC Object Classes

11.7 Summary

11.8 Check your progress

11.9 Answer Check your progress

11.10 Model Question

11.11 References

11.12 Suggested readings

11.1 LEARNING OBJECTIVES

This module gives an introduction about Java Database Connectivity (JDBC) API followed by the coverage of JDBC Architecture. This module also explains about JDBC, an application programming interface (API) for the programming language Java, enabling the users to understand the design of JDBC, its classes and relations among them.

11.2 JDBC

JDBC stands for Java Database Connectivity. It is used for accessing databases from Java applications.

Java is a very standardized programming language, but there are many versions of SQL databases.

ODBC vs. JDBC

Open DataBase Connectivity API can be used to access relational databases. ODBC can be used with Java applied as the form of JDBC-ODBC Bridge. ODBC is not appropriate for direct use from the Java programming language. It uses C interface. There occur a number of drawbacks in the security, implementation, robustness, automatic portability during the calls from Java to native C code.

Programs that are developed using Java-JDBC API are platform and vendor independent. JDBC involves the logical quote of "write once, compile once, run anywhere". It provides a standard API for tool or database developers and makes it possible to write database applications using pure Java API.

JDBC driver manager and JDBC drivers provide the bridge between the database and Java world.

11.3 JDBC ARCHITECTURE

Figure 11.1 Architecture of JDBC

Figure 11.1 shows the architecture of the working of JDBC. The application program written in Java code calls the JDBC library. JDBC loads a driver and through the driver, the application code talks to a particular database. A machine can have more than one driver and more than one database. The ideal thing about JDBC is that it can change database engines without changing any application code.

11.4 JDBC DRIVERS

There are four types of JDBC Drivers listed below,

- Type I: Bridge
- Type II: Native
- Type III: Middleware
- Type IV: Pure

JDBC Driver Types

Figure 11.2 shows the types of Drivers.

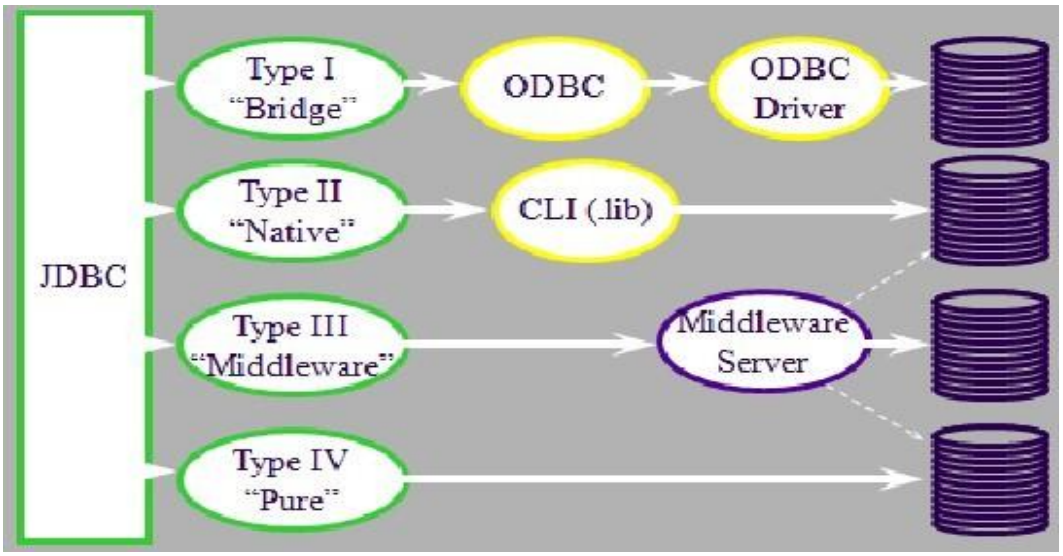


Figure 11.2 Types of JDBC Drivers

The Type I driver is an JDBC-ODBC bridge driver that uses ODBC driver to connect to the database. This JDBC-ODBC bridge driver converts JDBC method calls into the ODBC function calls.

The Type II drivers are the Native API driver that uses the client-side libraries of the database. This driver converts JDBC method calls into native calls of the database API. It is not written entirely in Java.

The Type III drivers are called the Network Protocol driver that uses middleware which is an application server that converts JDBC calls directly or indirectly into the vendor-specific database protocol. It is fully written in Java.

The Type IV drivers are the thin driver that converts JDBC calls directly into the vendor-specific database protocol. That is why it is known as thin driver. It is fully written in Java language.

Type I Drivers

These drivers uses bridging technology. They require installation/configuration on client machines.

These drivers are not recommended for Web. Example of Type I Driver is ODBC Bridge.

Type II Drivers

These drivers are native API drivers. They require installation/configuration on client machines. Type II drivers are used to leverage existing CLI libraries and are usually not thread-safe, mostly obsolete now.

Examples include Intersolv Oracle Driver, WebLogic drivers.

Type III Drivers

These drivers call middleware server, usually on database host. They are very flexible and allow access to multiple databases using one driver. There is only the need to download one driver, but it is another server application to install and maintain. An example of it is Symantec DBAnywhere.

Type IV Drivers

There is full 100% percentile Pure Java referred as the Holy Grail. These drivers use Java networking libraries to talk directly to database engines. A disadvantage here is the need to download a new driver for each database engine. Examples include Oracle, mySQL etc.

Let us now look in detail about JDBC Drivers.

JDBC Driver

The Driver class names for each of the database is given below,

In Oracle, the driver class name can be given as,

oracle.jdbc.driver.OracleDriver

In MySQL, the driver class name can be given as,

com.mysql.jdbc.Driver

In MS SQL, the driver class name can be given as,

com.microsoft.jdbc.sqlserver.SQLServerDriver

JDBC Limitations

The limitations of working with JDBC is that there are no scrolling cursors available and no bookmarks can be made while processing the records thereby it allows only sequential processing.

CHECK YOUR PROGRESS

True/False type questions

1. JDBC stands for Java Database Communication.
2. Open DataBase Connectivity API can be used to access relational databases.
3. There are Twenty types of JDBC Drivers.
4. ODBC can be used with Java and applied as the form of JDBC-ODBC Bridge.
5. ODBC is not appropriate for direct use from the Java programming language.
6. ODBC uses C interface.

Answers-

1. False
2. True
3. False
4. True
5. True
6. True

11.5 WORKING WITH JDBC

Package to be Imported

JDBC is implemented via classes in the java.sql package.

```
import java.sql.*
```

Loading a Driver Directly

```
Driver d = new foo.bar.MyDriver();

Connection c = d.connect(...);
```

We can create a Driver and load the driver. Using the Driver object we need to create a Connection object . This is not recommended but instead we can use a DriverManager but this is useful if we know that we need this particular driver.

DriverManager

A DriverManager tries all the drivers. It uses the first one that works. When a driver class is first loaded, it registers itself with the DriverManager. Therefore, to register a driver, we need to just load the driver.

Registering a Driver

Either of the below two methods can be used to register a driver.

Statically load driver

```
Class.forName("foo.bar.MyDriver");

Connection c = DriverManager.getConnection(...);
```

Use the jdbc.drivers system property

11.6 JDBC OBJECT CLASSES

The classes that are used from the imported java.sql.* package is listed below.

- DriverManager

This class is used for loading and choosing the drivers.

- Driver

  This class performs the connection to actual database.

- Connection

  The Connection class creates the connectivity through a series of SQL statements to and from the database.

- Statement

  The Statement object created using this class represents a single SQL statement.

- ResultSet

  The ResultSet object represents a set of the records returned from a statement.

JDBC URL's

The format of JDBC URL is given below that has components like subprotocol and source.

For example,

jdbc:subprotocol:source

Here each driver has its own subprotocol . Each subprotocol has its own syntax for the source.

The URL for Type I Driver which is an ODBC Bridge is given below.

jdbc:odbc:DataSource

Example: jdbc:odbc:jnf

The URL to connect to MySQL database is given below,

jdbc:mysql://host[:port]/database

Example: jdbc:mysql://foo.nowhere.com:4333/jnf

DriverManager

Connection getConnection (String url, String user, String password)

This connects to given JDBC URL with given user name and password and returns a Connection object. It can throw java.sql.SQLException when connection cannot be established.

Connection

A Connection represents a session with a specific database. Within the context of a Connection, SQL statements are executed and results are returned. A program can have multiple connections to a database. A connection provides “metadata” -- information about the database, tables, and fields, along with the methods to deal with transactions.

Obtaining a Connection

The snippet code for obtaining the connection with JDBC-ODBC bridge is given below. First we need to dynamically load and register the driver. Then establish a connection by creating a Connection object using the DriverManager class.

```
String url = "jdbc:odbc:jnf";
try
{
        Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
        Connection con = DriverManager.getConnection(url);
}
catch (ClassNotFoundException e)
{ e.printStackTrace(); }
catch (SQLException e)
 { e.printStackTrace(); }
```

Connection Methods

In the context of a Connection, SQL statements are executed and results are returned. Statement objects can be created using createStatement() method on the Connection object. Similarly PreparedStatement object can be created when we use prepareStatement() method on the Connection object and CallableStatement object can be created when we use prepareCall() method on the Connection object. The syntax of these methods are given below.

1. Statement createStatement()

   This returns a new Statement object.

2. PreparedStatement prepareStatement(String sql)

   This returns a new PreparedStatement object.

3. CallableStatement prepareCall(String sql)

   This returns a new CallableStatement object.

Statement

A Statement object is used for executing a static SQL statement and obtaining the results produced by it.

Statement Methods

There are three methods which works with the Statement object.

ResultSet executeQuery(String)

This method executes an SQL SELECT statement that returns a single ResultSet.

int executeUpdate(String)

This method is used to execute an SQL INSERT, UPDATE or DELETE statement and returns the number of rows changed.

boolean execute(String)

This method is used to execute an SQL statement that may return multiple results.

ResultSet

A ResultSet provides access to a table of data generated by executing a Statement. Only one ResultSet per Statement can be open at once. The table rows are retrieved in sequence. A ResultSet maintains a cursor pointing to its current row of data. The 'next()' method is used to move the cursor to the next row.

ResultSet Methods

Below is the list of methods to work with the ResultSet object.

- boolean next()

  This method activates the next row, the first call to next() activates the first row and returns false if there are no more rows.

- void close()

  This method disposes the ResultSet. It allows you to re-use the statement that created it automatically called by most statement methods.

- Type getType(int columnIndex)

  This method returns the given field as the given type fields indexed starting at 1 (not 0).

- Type getType(String columnName)

  This method performs the same function as before, but uses name of field and less efficient.

- int findColumn(String columnName)

  This method looks up column index given column name.

- String getString(int columnIndex)

This method retrieves the value of the designated column in the current row of this ResultSet object as a String in the Java programming language.

- boolean getBoolean(int columnIndex)
  This method retrieves the value of the designated column in the current row of this ResultSet object as a boolean in the Java programming language.

- byte getByte(int columnIndex)
  This method retrieves the value of the designated column in the current row of this ResultSet object as a byte in the Java programming language.

- short getShort(int columnIndex)
  This method retrieves the value of the designated column in the current row of this ResultSet object as a short in the Java programming language.

- int getInt(int columnIndex)
  This method retrieves the value of the designated column in the current row of this ResultSet object as an int in the Java programming language.

- long getLong(int columnIndex)
  This method retrieves the value of the designated column in the current row of this ResultSet object as a long in the Java programming language.

- float getFloat(int columnIndex)
  This method retrieves the value of the designated column in the current row of this ResultSet object as a float in the Java programming language.

- double getDouble(int columnIndex)

This method retrieves the value of the designated column in the current row of this ResultSet object as a double in the Java programming language.

- Date getDate(int columnIndex)
  This method retrieves the value of the designated column in the current row of this ResultSet object as a java.sql.Date object in the Java programming language.

- Time getTime(int columnIndex)
  This method retrieves the value of the designated column in the current row of this ResultSet object as a java.sql.Time object in the Java programming language.

- Timestamp getTimestamp(int columnIndex)
  This method retrieves the value of the designated column in the current row of this ResultSet object as a java.sql.Timestamp object in the Java programming language.

isNull

In SQL, NULL means the field is empty. It is not the same as 0 or “”. In JDBC, you must explicitly ask if a field is null by calling,

ResultSet.isNull(column)

11.7 SUMMARY

This module gave an introduction about JDBC API that defines how a client may access a database. It also discussed about the JDBC Architecture that uses a driver manager and database-specific drivers to provide a transparent connectivity to heterogeneous databases. The module also provides us with a basic understanding about the design of JDBC API, the classes, methods and relations in them.

11.8 CHECK YOUR PROGRESS

1. JDBC is used for accessing ..... from Java applications.
2. Programs that are developed using Java-JDBC API are ..... and vendor independent.
3. ..... provides a standard API for tool or database developers and makes it possible to write database applications using pure Java API.
4. JDBC driver manager and JDBC drivers provide the ..... between the database and Java world.
5. JDBC loads a driver and through the driver and a machine can have more than one driver and more than one.....
6. The ideal thing about JDBC is that it can change .....without changing any application code.
7. The JDBC-ODBC bridge driver converts JDBC method calls into the ..... function calls.

11.9 ANSWER CHECK YOUR PROGRESS

1. Databases
2. platform
3. JDBC
4. Bridge
5. Database
6. database engines
7. ODBC

11.10 MODEL QUESTION

1. What is JDBC? What are the advantages and limitations of JDBC? Explain.
2. What are JDBC Drivers? What are their roles? What are the different types of types of JDBC? Explain each.
3. How JDBC is implemented via classes in the java.sql package?
4. Explain the process of Loading a Driver Directly? How can you create a Connection object using the Driver object? Explain.
5. What are the different JDBC object classes? Explain each with help of example or proper syntax.

11.11 REFERENCES

1. https://docs.oracle.com/javase/7/docs/api/java/sql/ResultSet.html
   www.javatpoint.com/jdbc-driver
2. https://epgp.inflibnet.ac.in/Home/ViewSubject?catid=7

11.12 SUGGESTED READINGS

- Powell. Thomas A., JavaScript: The Complete Reference
- lemay. Laura, rafe colburn, jennifer kyrnin, Mastering HTML, CSS & JavaScript Web, BPB Publication, 2016
- Vishvajeet. Sisodia, Basic of Web Design, HTML, CSS3, Centrum Press, 2014.

UNIT-XII JAVA SERVLET IMPLEMENTATION

12.1 Learning Objectives

12.2 Tomcat Installation

12.3 Servlet Implementation

12.4 The doGet method

12.5 The doPost method

12.6 Summary

12.7 Check your progress

12.8 Answer Check your progress

12.9 Model Question

12.10 References

12.11 Suggested readings

12.1 LEARNING OBJECTIVES

This module demonstrates about Tomcat Installation to build web applications using Servlets and JSP pages. It demonstrates about the implementation of Servlets. This section also enables us to understand how to create a basic Servlets with passing parameters.

12.2 TOMCAT INSTALLATION

Step 1: Download and Install Tomcat

Go to https://tomcat.apache.org/download-80.cgi to download and install Tomcat. This is shown in Figure 1.

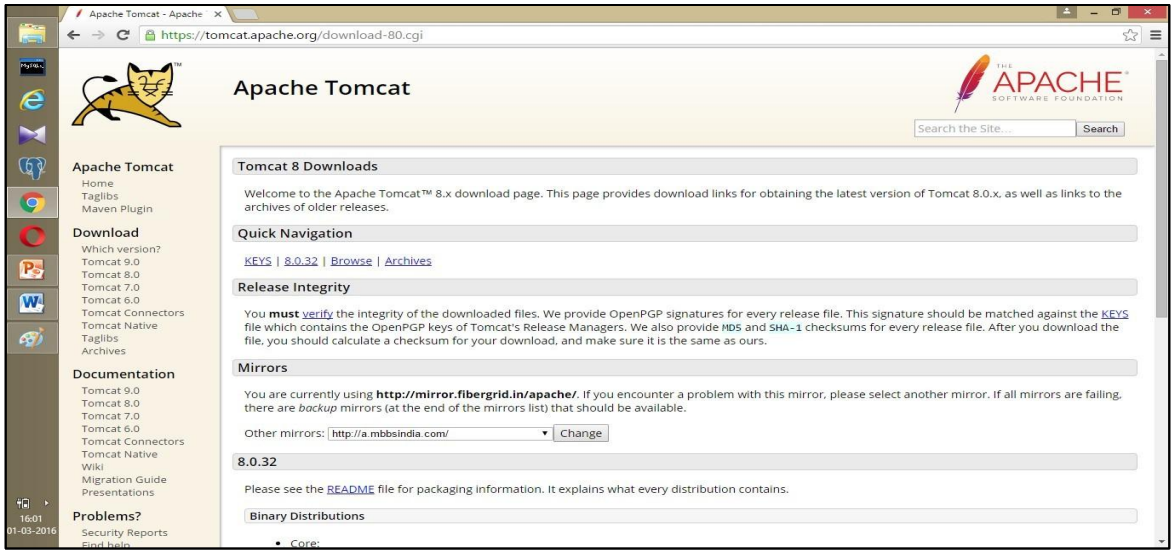


Figure 1 Download and Install Tomcat

Step 2. Then go to the Binary Distribution/Core/ and download the "zip" package.

This is shown in Figure 2.

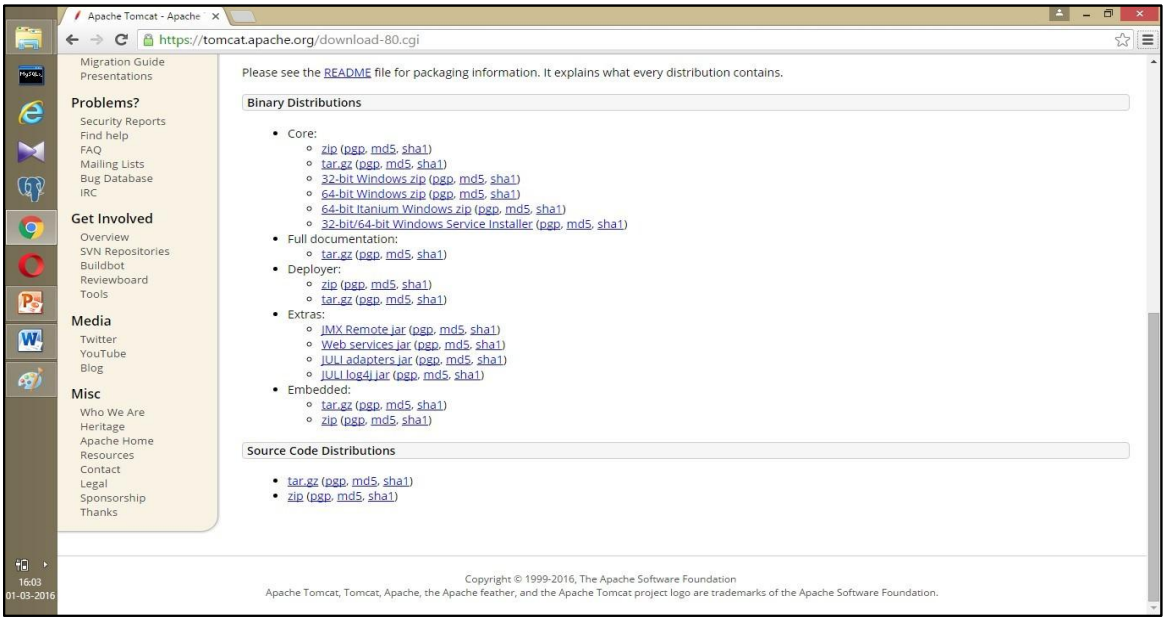


Figure 2. Download the "zip" Package

Check the installed directory to ensure that it contains the following sub-directories like bin folder, logs folder, webapps folder, work folder, temp folder, conf folder and lib folder as shown in Figure 3.

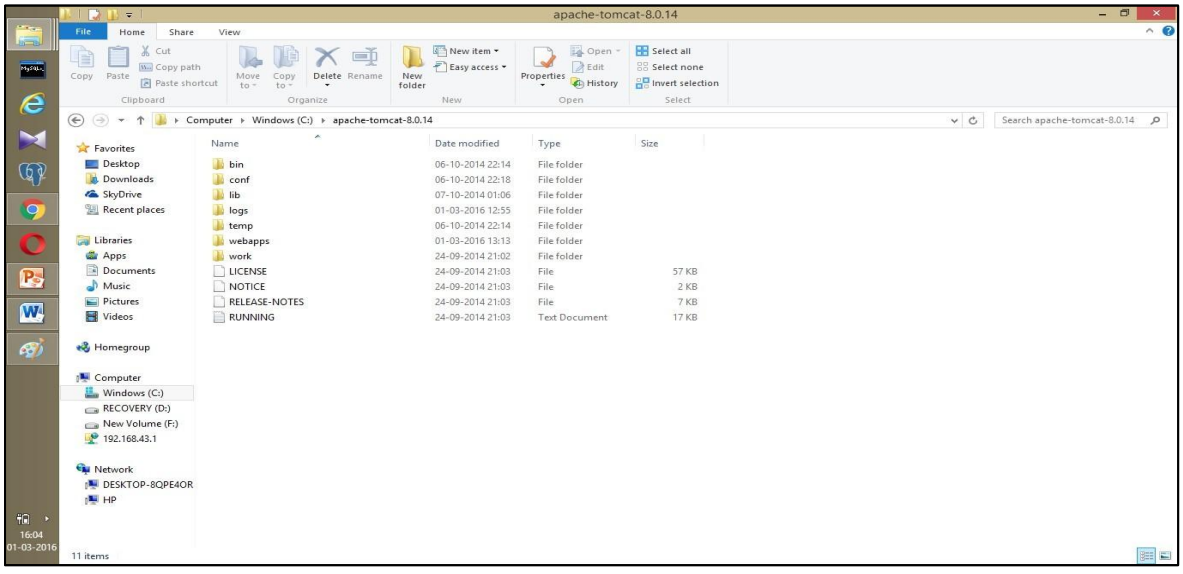


Figure 3 Checking the Installed Directory

Create Environment Variables

Now, We need to create an environment variable called "JAVA_HOME" and set it to our JDK installed directory. To create the JAVA_HOME environment variable in Windows, follow these steps:

Step 1: Right Click “My Computer” then go to Properties. This is shown in Figure 4.

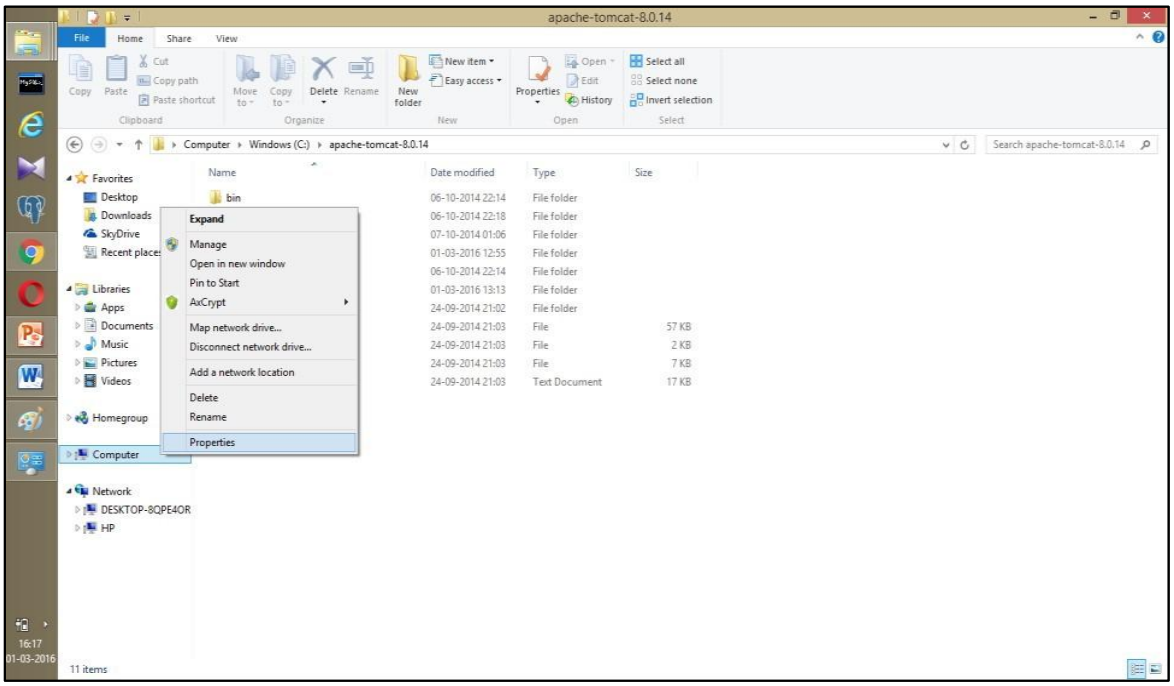


Figure 4. Right Click “My Computer” then go to Properties

Step 2: Select “Advanced System Settings” on the left side of the control panel. This is shown in Figure 5.

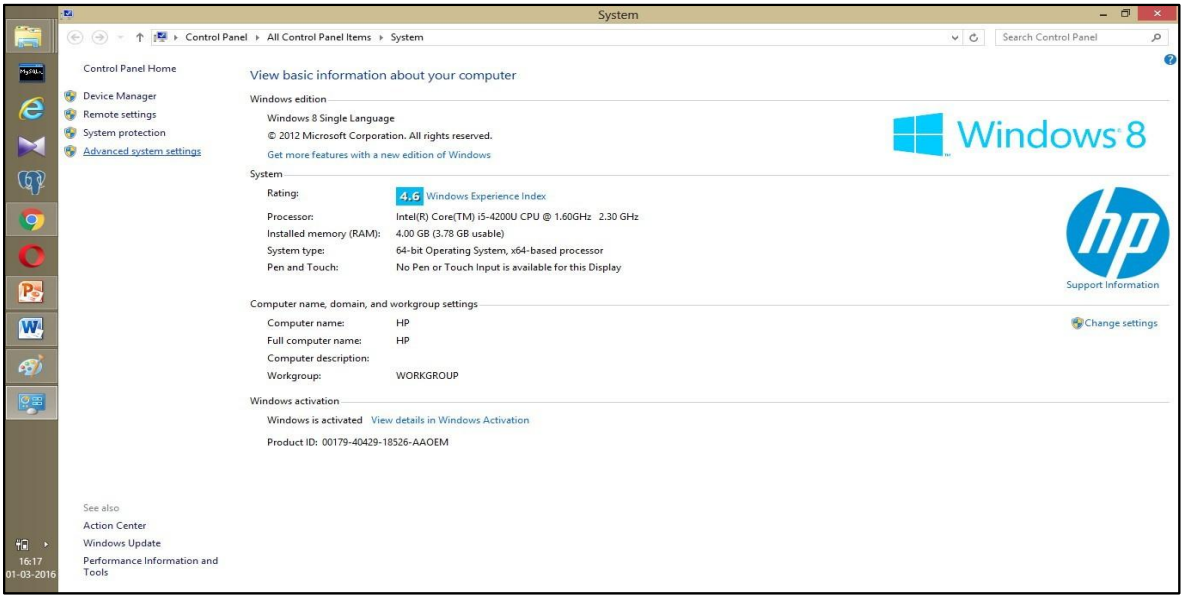


Figure 5 Select “Advanced System Settings”

Step 3: Select “Environment variables” in the System properties dialog box. This is shown in Figure 6.

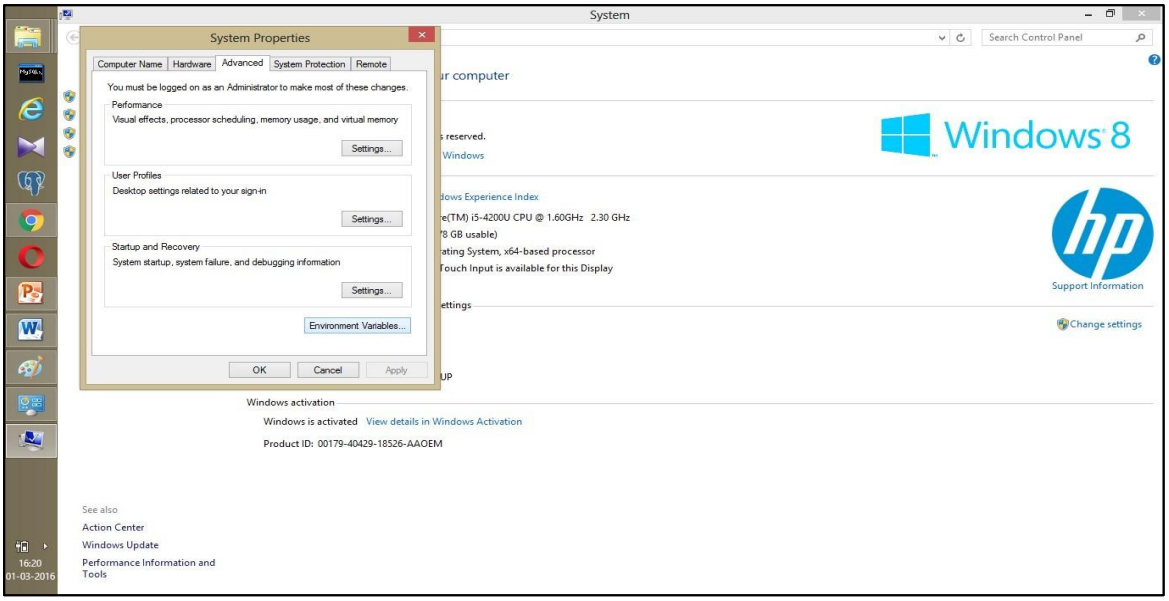


Figure 6 Select “Environment variables”

Step 4: Select "New" (or "Edit" for modification). In "Variable Name", enter "JAVA_HOME". In "Variable Value", enter your JDK installed directory (e.g., "c:\Program Files\Java\jdk1.8.0_05"). This is shown in Figure 7.

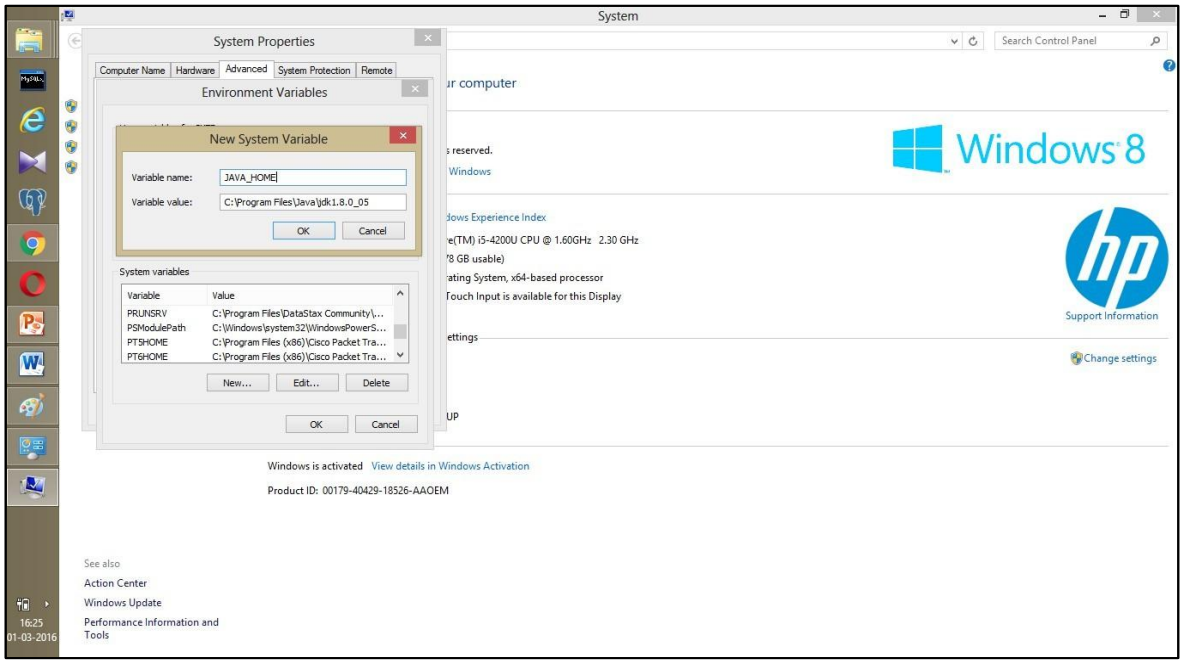


Figure 7 Enter "Variable Name" and "Variable Value"

12.3 SERVLET IMPLEMENTATION

To create a new Servlet do the following steps:

Step 1: Go to webapps folder in Tomcat Apache directory, then create New Folder and name it as “HelloWorld”. This is shown in Figure 8.

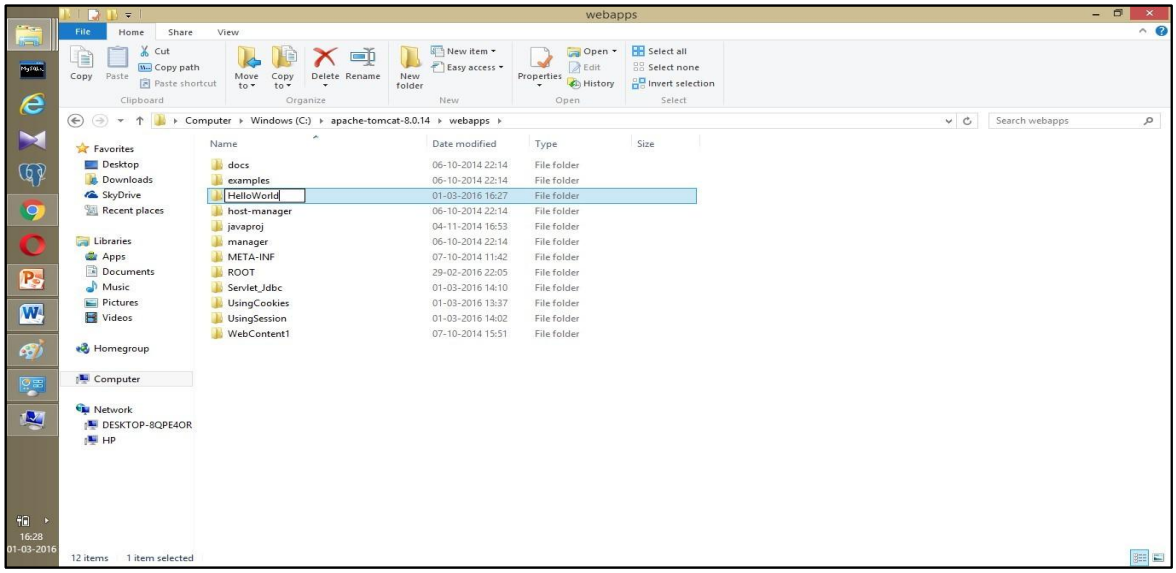


Figure 8 Create New Folder “HelloWorld”

Step 2: Create a new folder inside the HelloWorld folder and name it as “WEB-INF”.

Inside the WEB-INF folder, create a folder name “classes” as shown in Figure 9.

Step 3: Create a Servlet page inside the Helloworld folder and name it as “HelloWorld.java”.

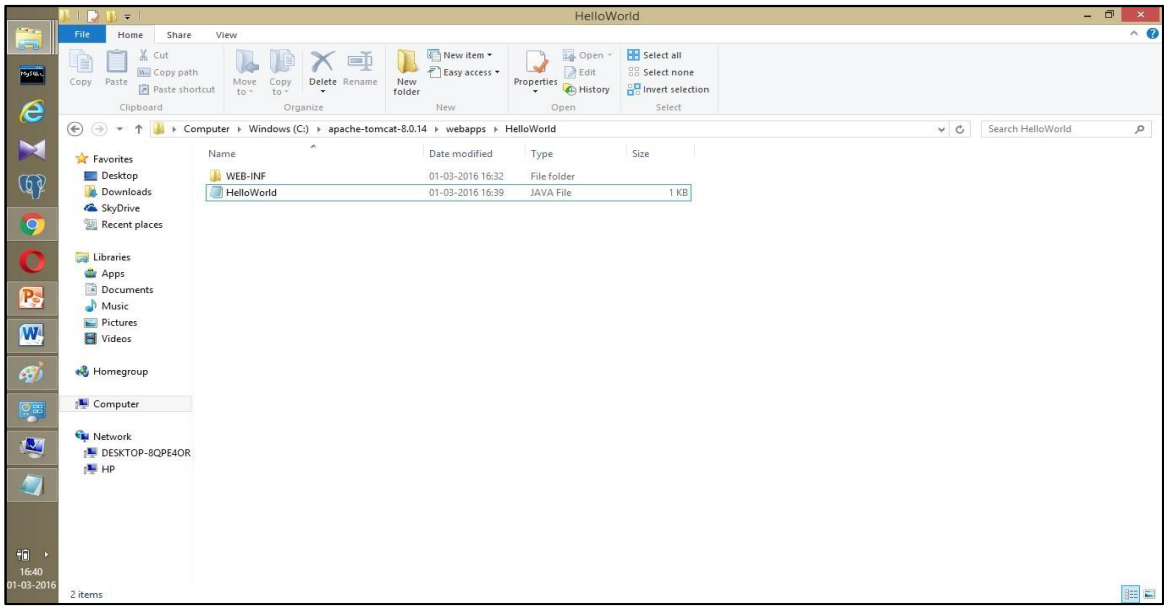


Figure 9 Create Servlet "HelloWorld.java"

HelloWorld.java

```
import javax.servlet.*;

import javax.servlet.http.*;

import java.io.*;

public class Hello extends HttpServlet

{

public void doGet(HttpServletRequest req,HttpServletResponse res) throws ServletException,IOException

{

        res.setContentType("text/html");

        PrintWriter out=res.getWriter();
        out.println("<html><head><title>Hello WorldServlet</title></head>

                <body><h1>HelloWorld</h1></body></html>");

}

}
```

Step 4: Inside the WEB-INF folder, Create an XML file named “web.xml”.

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

<web-app xmlns="http://xmlns.jcp.org/xml/ns/javaee"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://xmlns.jcp.org/xml/ns/javaee
http://xmlns.jcp.org/xml/ns/javaee/web-app_3_1.xsd"version="3.1">

<session-config>

<session-timeout> 30

</session-timeout>

</session-config>

 <servlet>

<servlet-name>hello</servlet-name>

<servlet-class>HelloWorld</servlet-class>

</servlet>

 <servlet-mapping>

<servlet-name>hello</servlet-name>

<url-pattern>/HelloWorld</url-pattern>

</servlet-mapping> </web-app>
```

Step 5: Compile HelloWorld.java file.

Follow these steps in compilation:

1. Go to cmd prompt, then go to the directory “C:\apache-tomcat-8.0.14\webapps\HelloWorld” as shown in Figure 10.

Figure 10 In 'cmd' prompt, go to directory “C:\apache-tomcat8.0.14\webapps\HelloWorld”

2. Now we need to include the servlet-api.jar file to compile the HelloWorld servlet file.

3. Go to lib folder inside the apache folder (for example : C:\apache-tomcat-8.0.14\lib\servlet-api.jar) 4. Now compile the Servlet by specifying the classpath of the servlet-api.jar as shown in Figure 11.

javac -cp "C:\apache-tomcat-8.0.14\lib\servlet-api.jar" HelloWorld.java

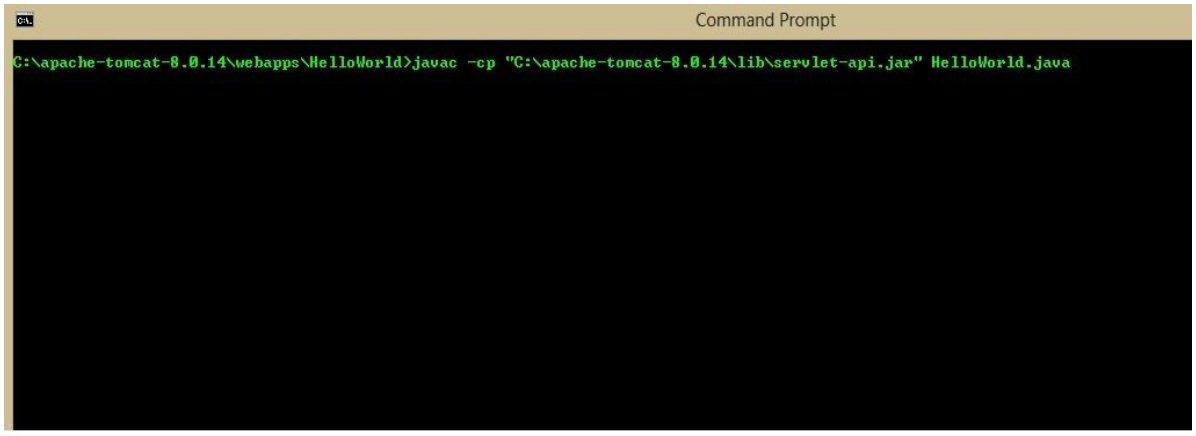


Figure 11 Specifying the Classpath

5. After compilation, the class file will be generated inside the HelloWorld folder.

6. Cut the HelloWorld.class file and paste it inside the WEB-INF\classes folder. This is shown in Figure 12.

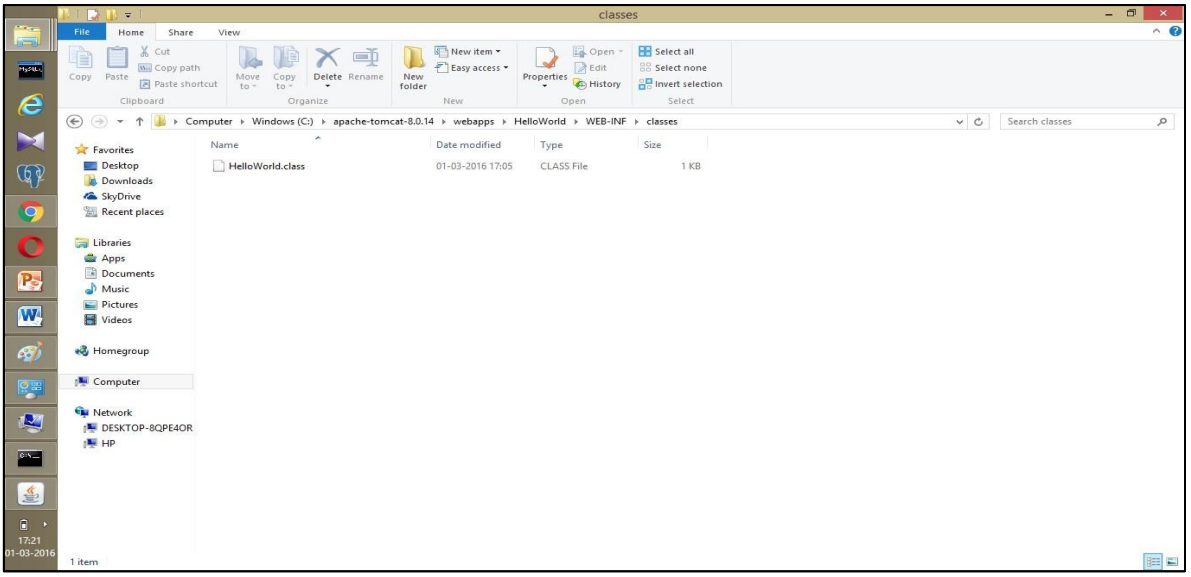


Figure 12 HelloWorld.class file inside the WEB-INF\classes folder

Step 6: Start the Tomcat Server.

Follow these steps for starting the server.

1. Go to bin folder inside the apache folder and double click the startup.bat file. It will start the Tomcat server. This is shown in Figure 13.

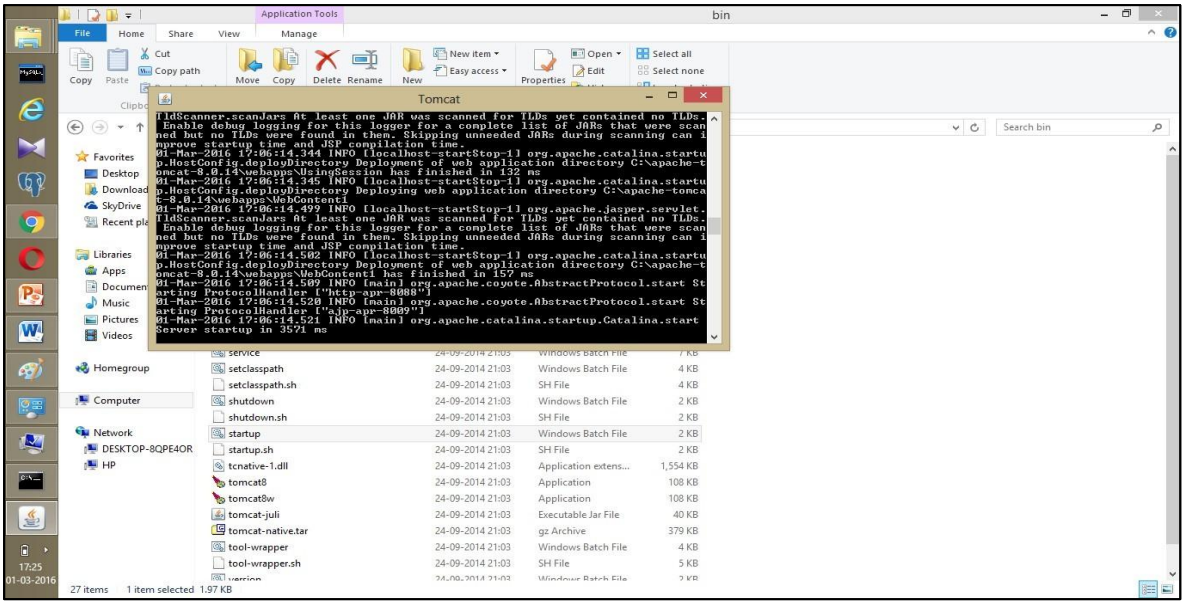


Figure 13 Start the Tomcat Server

2. Now go to web browser and type the following URL.

http://localhost:8080/HelloWorld/HelloWorld

The output will be displayed as shown in Figure 14

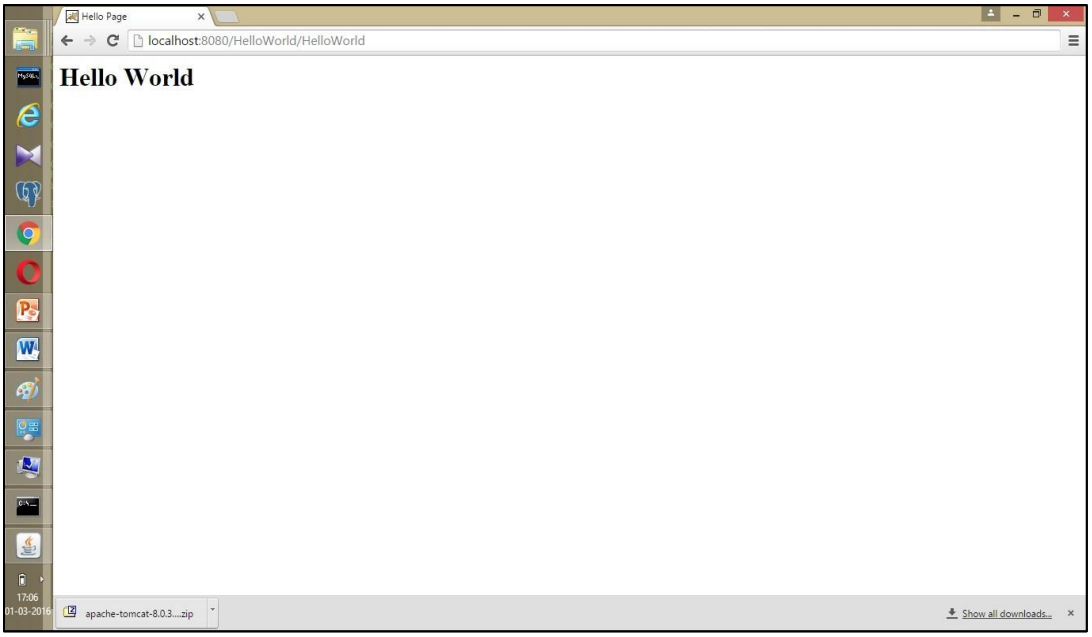


Figure 14 Output of the Servlet "Hello World"

12.4 THE doGet METHOD

public void doGet(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException

This method services a GET request. The method uses request to get the information that was sent to it and the method does not return a value; instead, it uses response to get an I/O stream, and outputs its response. Since the method does I/O, it can throw an IOException. Any other type of exception should be encapsulated as a ServletException.

Parameters:

The request object is an HttpServletRequest object that contains the request the client has made of the servlet

The response object is an HttpServletResponse object that contains the response the servlet sends to the client

Using the HttpServletResponse

The second parameter to doGet (or doPost) is HttpServletResponse response. Everything sent via the Web has a “MIME type”. The first thing we must do with response is set the MIME type of our reply.

```
response.setContentType("text/html");
```

This tells the client to interpret the page as HTML. Because we will be outputting character data, we need a PrintWriter, handily provided for us by the getWriter method of response.

```
printWriter out = response.getWriter();
```

Then use the println method of out one or more times to write the response to the client .

```
out.println(docType +"<HTML>\n" +"<HEAD> ... </BODY></HTML>");
```

First Servlet Program

```
<html>
        <body>
        <form  method = "get"
        action="http:\\127.0.0.1:8080\servlet-html\servlet\Example">
        <input type=submit value = "submit">
        </form>
        </body>
</html>
```

Sample Servlet Program

Following is the sample source code structure of a servlet example.

```
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;
public class Example extends HttpServlet
{
    public void doGet(HttpServletRequest req,   HttpServletResponse res)
                                        throws ServletException ,IOException
```

```
        {
                res.setContentType("text/html");
                PrintWriter out = res.getWriter();
                out.println("<html>Hai --- JNF</html>");
        }
}
```

Hello.java

This example illustrates about implementing a 'Hello' Servlet that extends the HttpServlet class. The code provides the implementation of the doGet() method. Notice that get request is the default request method.

Initially we set the content type to be text/html. Then we use the getWriter() method of the HttpServletResponse to write the response to the servlet which returns an OuputStream. This OutputStream is assigned to a PrintWriter object 'out'. So now using this 'out' object, we write the html response to the client.

This html code will print the Hello World text in green color in the client response browser window.

```
import java.io.*;
import java.util.*;
import javax.servlet.*;
import javax.servlet.http.*;
public class Hello extends HttpServlet
{
        protected void doGet(HttpServletRequest req, HttpServletResponse res)
                                throws ServletException, IOException
        {
                res.setContentType("text/html");
                PrintWriter out = res.getWriter();
                out.println("<html>");
```

```
        out.println("<body>");

        out.println("<h1><font color='green'>Hello World!</font></h1>");
        out.println("</body>");    out.println("</html>");    out.close();

    }

}
```

Output of Hello Servlet

Open a browser window and type the path where our servlet is stored i.e.,

localhost:8084/MyFirstServlet/servlet/Hello

The output message "HelloWorld" is displayed in green color as shown in Figure 15.

Figure 15

Parameters to doGet

The parameters are the way in which a user can send information to the Web Server. Input parameters from the client HTML form is read from the HttpServletRequest parameter. Our first program does not get any input. Output is sent through the HttpServletResponse object, which we have named as response.

I/O in Java is very flexible but also quite complex, so this object acts as an “assistant”.

Getting the parameters

From the request object we can read the parameter submitted by the user's browser either through an HTTP GET or POST method.

Basically we need to know that we can call the request.getParameter(paramname) method where the paramname is the name of paramater to get the passed parameter from inside the Servlet.

Input parameters are retrieved via messages to the HttpServletRequest object request. Most of the interesting methods of the class HttpServletRequest are inherited from the super interface ServletRequest.

public Enumeration getParameterNames();

This returns an Enumeration of the parameter names. If the key names are not known, the Servlet can enumerate (iterate) through the keys and obtain data in that manner.

```
Enumeration enum = request.getParameterNames();
while(enum.hasMoreElements())
{
    String key = (String) enum.nextElement();
    // …
```

If no parameters are retrieved, it returns an empty Enumeration.

There are two important methods used with HttpServletRequest object to retrieve the parameters sent from the client HTML page:

- **public String getParameter(String name)**

This method returns the value of the parameter name as a String. If the parameter doesn't exist, it returns null. If name has multiple values, only the first value is returned.

- **public String[] getParameterValues(name)**

This method returns an array of values of the parameter name. If the parameter doesn't exist, it returns null.

Here, we illustrate with an example program how we can pass parameters to Servlets, that can be read and processed by the Servlet.

Example 1 - Passing Parameters

First we develop an HTML page that sends information to a Servlet. To implement this requirement we have to make one html form. In this example we have created an HTML form that has a drop-down list in which user can select any color displayed in the list. A submit button is provided which on pressing it, the request will go to the server and servlet will do the corresponding processing.

Example1.html

```
<html>
<body>
<form  method = "get" action="http:\\127.0.0.1:8080\servlethtml\servlet\Example1">
        <input type=submit value="submit">
        <input type=input name="Read" value="">
                <select name="Color">
                        <option >Red
                        <option >Blue
                        <option value="dss">Green
                        <option value="dss">Yellow
                </select>
</form>
</body>
</html>
```

Example1.java

In the servlet class the value is read from the form by using the method getParameter(). The output is then displayed to the client by the object of the PrintWriter class.

```
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;
public class Example1 extends HttpServlet
```

```
{
        public void doGet(HttpServletRequest req,HttpServletResponse res)
                                                        throws ServletException ,IOException
        {
                 res.setContentType("text/html");
                PrintWriter out = res.getWriter();
                String str= req.getParameter("Read");
                String str2= req.getParameter(“Color");
                out.println("Hai  "+ str + str2);
        }
}
```

Example 2 - Passing Parameters

This example illustrates about sending the parameters through which a user can send information to the Web Server. For example, we write a HTML page in the client side where we send the message entered in a text box to the server, so that the Servlet reads this text value and send it as response to the client. And we will also have one submit button, on pressing the submit button the request will go to the server and Servlet will do the corresponding validation.

ParamPassing.html

```
<html>
<head>
</head>
<body>
        <form method=GET action="http://localhost:8080/servlethtml/ParamPassing.html">
                        <input type=submit name="Submit" value="First">
                        <input type=text name="TextBox" value="">
        </form>
```

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

ParamPassing.java

The servlet class collects the value from the text box by using the method getParameter(). The output is then displayed to the client by the object of the PrintWriter class.

```
import javax.servlet.*;
import javax.servlet.http.*;
import java.io.*;
public class ParamPassing extends HttpServlet
{
    public void doGet(HttpServletRequestreq,HttpServletResponse res)
                                                    throws IOException,ServletException
    {
        res.setContentType("text/html");
        PrintWriterpw=res.getWriter();
        String str=req.getParameter("TextBox");
        //pw.println("<html><body> "+str+ "</body></html>");
        pw.println(str);
    }
    public void doPost(HttpServletRequestreq, HttpServletResponse res)
                                                    throws IOException,ServletException
    {
        doGet(req,res);
    }
}
```

The output of this example is shown in Figure 16.

Figure 16

CHECK YOUR PROGRESS

True/False type questions

1. The full form of JDK is Java Development kit.
2. The do Get method is used to services a GET request.
3. The doPost() performs visible data submission.
4. In order to send a POST request to server, the method "POST" is used.
5. The doGet() performs invisible data submission.

Answers-

1. True
2. True
3. False
4. True
5. False

12.5 THE doPost METHOD

We can use doPost() when we want to intercept on HTTP POST requests. The doPost() performs invisible data submission whereas doGet() performs visible data submission. The doGet() method is not recommended for everything and doPost() is generally used to carry passwords.

Using "GET" the request parameters are passed to the server by appending at the end of the URL, whereas in a "POST" request form elements or parameters are passed as a part of HTTP body and does not append at the end of URL. So some sensitive information is sent to the server, a POST request is sent.

HelloPost.html

In order to send a POST request to server, we need to mention the method to be "POST" in the form as shown in the HTML code.

```
<html>
        <head>
                <title>
                        My Page
                </title>
        </head>
<body>
        <form  method="Post" action="http://127.0.0.1:8080/servlet/HelloPost">
                <input type="text" name="username" value="">
                 <input type=submit >
        </form>
</body>
</html>
```

HelloPost.java

By looking at POST request, the servlet engine will call appropriate class as mapped in web.xml and invokes the doget() method, which in turn calls the dopost() method.

```
import javax.servlet.*;
import javax.servlet.http.*;
import java.io.*;
public class HelloPost extends HttpServlet
{
    public void doPost(HttpServletRequest req,HttpServletResponse res)
                                        throws ServletException, IOException
    {
        res.setContentType("text/html");
        PrintWriter out=res.getWriter();
        out.println("<html><head><title>Hello
        Page</title></head><body>Hello</body></html>");
    }
}
```

Passing Parameters and doPost method

This example illustrates the use of doPost by overriding this method. The doPost method inturn invokes the doGet method which reads the request data, write the response headers, get the response's writer or output stream object, and finally, write the response data.

```
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;
public class Test extends HttpServlet
{
    public void doGet(HttpServletRequest req, HttpServletResponse res)
                                        throws ServletException, IOException
    {
        res.setContentType(“text/html”); PrintWriter out = res.getWriter();
```

```
        String pin = req.getParameter("to"); String orig = req.getParameter("from");
        out.println("Sending page to " + pin + " from " + orig);
        // Actually send the page.
    }
    public void doPost(HttpServletRequest in, HttpServletResponse out)
                                                    throws ServletException, IOException
    {
        doGet(in, out);
    }
}
```

12.6 SUMMARY

In this module we have discussed about the installation of Tomcat to work with Servlets. The module covers the method of acquiring the required software components to use Tomcat for web application development, thereby demonstrating how to implement Servlets. Ultimately, this section has provided simple Servlet programs to have a clear understanding about creating dynamic web applications.

12.7 CHECK YOUR PROGRESS

Fill in the blanks:

1. The _____performs invisible data submission whereas _____performs visible data submission.
2. _____method uses request to get the information that was sent to it and the method does not return a value.
3. Using_____ request parameters are passed to the server by appending at the end of the URL.
4. _____request form elements or parameters are passed as a part of HTTP body and does not append at the end of URL.
5. The full form of JDK is_____

12.8 ANSWER CHECK YOUR PROGRESS

1. Do Post() and Do Get()
2. Do Get Method
3. Get
4. Post
5. Java Development Kit

12.9 MODEL QUESTION

1. What is Servlet in java application? What is the function of doGet method of Servlet in Java?
2. What is the difference between doGet() and doPost() methods? Explain with help of example.
3. How can we create a new Servlet? Write suitable program to create a new Servlet.
4. What are passing Parameters. Explain with the help of Example?
5. How a POST request is send to a server? Also explain the difference between Java Server Pages and Servlet.

12.10 REFERENCES

- https://epgp.inflibnet.ac.in/Home/ViewSubject?catid=7

12.11 SUGGESTED READINGS

- Powell. Thomas A., JavaScript: The Complete Reference
- lemay. Laura, rafe colburn, jennifer kyrnin, Mastering HTML, CSS & JavaScript Web, BPB Publication, 2016
- Vishvajeet. Sisodia, Basic of Web Design, HTML, CSS3, Centrum Press, 2014.